Documentation

Bachelor Thesis
Documentation

What's up in RJ

Semester: Spring 2024

Version: 1.0
Date: 2024-06-14

Authors:	Michael Enzler Fabio Stocker
Advisor:	Prof. Frank Koch
Project Partner:	Michael Güntensperger, AdaptIT GmbH
External Co-Examinor:	Hansjörg Huser
Internal Co-Examinor:	Cyrill Brunschwiler

OST
Eastern Switzerland
University of Applied Sciences

Department of Computer Science
OST Eastern Switzerland University of Applied Sciences
Campus Rapperswil-Jona

Abstract

The project aims to further develop the already existing platform 'What's up in RJ'. The product is a web application that provides a space where organizers can share their events and activities with the community of Rapperswil-Jona. Initially, the platform provided a basic set of features, such as creating, managing, searching and filtering events and subscribing to a newsletter.

A new feature lets organizers create their own profile page, giving them the ability to present themselves to visitors. Another new feature added is the upload of images, allowing organizers to upload custom images to their profile and events. Furthermore, an admin panel is available that allows an administrator to manage organizers and user accounts. The administrator has the ability to decide which organizers are allowed to publish their profile and events by verifying their accounts.

The approach is to take the existing codebase, using the same technologies, and extend it with new features. Therefore, the web application remains a three tier application with a frontend, backend and database. The only change is the object storage, which is added to have a space to store images uploaded by organizers.

After an initial inception phase, the new features were designed and planned during the elaboration phase. The implementation of each feature then took place in the construction phase. In a final step, the platform was tested with various test cases by real end-users to evaluate the functionality and usability of the new features.

This project iteration makes the platform more attractive for all users and provides an overall better user experience. There are still many possibilities for further development and improvement to increase the platform's value for the community. With this project, the web application has been brought to the next level, and one step closer to make it a successful platform.

Management Summary

Initial Situation

Rapperswil-Jona is full of activities and events that are taking place all year round. The organizers of these events are promoting them through various channels, such as social media, posters, newspapers, and websites. With the platform 'What's Up in RJ', a central place is created, where events can be published and viewed by the public. A previously developed prototype in the context of a 'Studienarbeit' is to be further developed and improved. The existing web application provides the basic functionality to create and manage user and organizer accounts. The organizers were allowed to create, edit and delete events. All viewers were then able to view, search and filter the events. Additionally, registered users can subscribe to a newsletter. This existing prototype gets extended with new features such as image upload, admin panel, organizer profile page, and responsive design. The goal is to add needed features for the platform before going live.

Approach and Technologies

The existing three tier web application is to be refactored in a first step to create a clean code base. Then, PostgreSQL based database is extended with new tables and single fields in existing tables. The existing Node.js backend, utilizing the Express.js framework that exposes a RESTful API, is to be extended with new endpoints. The backend implements the business logic and remains the only layer to communicate with the database. Furthermore, it communicates with an external object storage provided by DigitalOcean Spaces to store the images. It also connects with the SendGrid API to send the newsletter or verification emails to the users. With the new backend endpoints, the Angular frontend can be extended to use the new functionality and display the data, including the images. As styling library, Tailwind CSS will be reused, which is required to keep the design consistent and to make the application responsive. For the deployment, DigitalOcean with its App Platform feature remains the chosen cloud provider.

Results

The result is a web application with several new features.
Organizers can create their own profile page, where they can present themselves with a description and contact information. In addition, the organizers are able to upload images to their profile page and events.
The newly added administrator can log in to access the admin panel that contains user and category management panels. Using the user management, the administrator has the new ability to verify and unverify organizers and view all organizers, users and administrators. When an organizer creates their account they are unverified by default, which means that they cannot create public events, nor is their

profile publicly visible. After an administrator verifies the organizer via the user management, the organizer is able to create public events and showcase their profile. An administrator can also create new administrators and delete existing ones. The category management enables the administrator to create, edit and delete categories that are assigned to events.
Furthermore, after creating a user or organizer account, an email with an email verification link will be sent to their inbox. The user receiving the email has to click the link to verify their email and use the full functionality of the platform i.e. subscribe to the newsletter.
Besides the received email, the user will be shown an onboarding process after creating an account. These pages explain the platform and its functionality to the user. For example, the user gets asked to subscribe to the newsletter and select their event preferences. On the other hand, the organizer is invited to create their first event through a step-by-step guide, to edit their profile page, or to browse the different pages for inspiration.
Finally, all users, logged in or not, can use the platform on different screen sizes and mobile devices due to the newly added responsive design.
The following figure 1 shows the final system architecture of the web application. The different layers, frontend, backend and database, running on the app platform are visible. Moreover, it displays their communication with each other and external services such as the object storage on DigitalOcean or the SendGrid API.

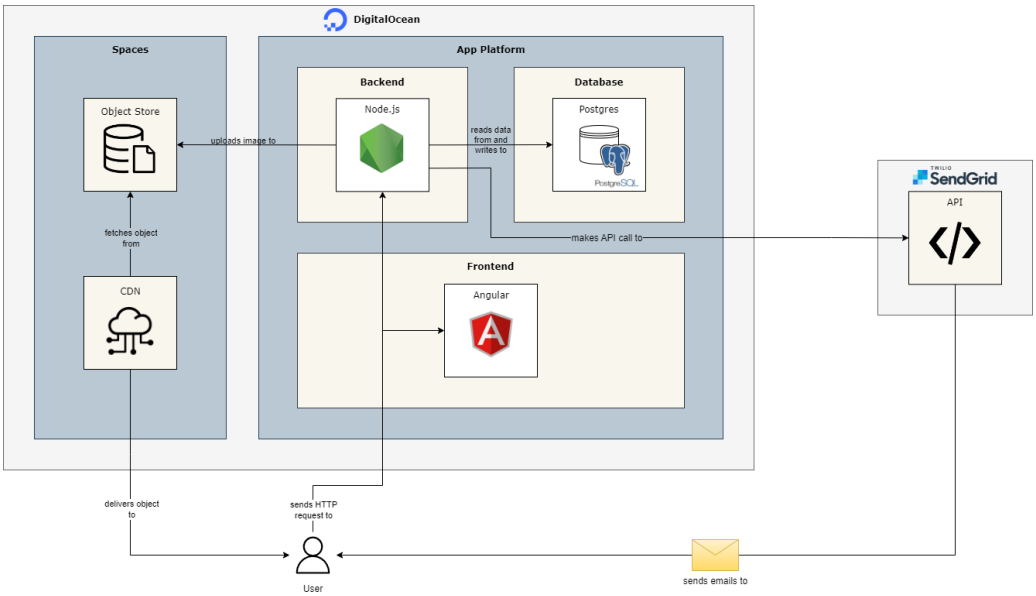


Figure 1: System architecture

Contents

Part I

Project Overview

Chapter 1

Overview

This document contains all information about the bachelor thesis 'What's up in RJ'. The four parts 'Project Overview', 'Product Documentation', 'Project Documentation' and 'Appendix' are used to divide the information in a clear way. With the 'Project Overview' part the reader gets a good understanding of the project and the context in which it is placed. The 'Product Documentation' contains all information about the product itself, such as the requirements, architecture, implementation, and results. On the other hand, the 'Project Documentation' part contains information about the project, like the planning and the time tracking. The final part, the 'Appendix', has all the additional information in it, such as protocols of meetings and other reports and documents.

1.1 Initial Situation

The idea of the product 'What's up in RJ' is originally from the project partner 'AdaptIT GmbH'. A first project in the context of the 'Studienarbeit' was started with the same collaborators as this project. During the project, the goal was to create a prototype of a platform where organizers of the region 'Rapperswil-Jona' can publish their events to a broader audience. The project was successful and the idea to further develop the platform came up. This bachelor thesis is based off of the prototype, which was created during the 'Studienarbeit', and aims to develop the platform further with new features and improvements.

1.2 Task

The task is to extend the already existing web application with new features. For organizers, it should be possible to have their own user profile where they can present themselves. Additionally, they should be able to upload pictures to their profile and events. A new user type for administrators of the website should be implemented, which can manage the different user and organizer accounts. Furthermore, the platform should realize a responsive design to be usable on different screen sizes and mobile devices.

1.3 Conditions

This project was done in the context of a bachelor thesis at the Eastern Switzerland University of Applied Sciences (OST). The team is composed of two students from the computer science department. The time frame is from 19.02.2024 to 14.06.2024, which is a total of 17 weeks. As this is a bachelor thesis, every student is to work 360 hours on the project, which translates to approximately 21 hours

per week. A student will receive one ECTS point for every 30 hours of work, which is a total of 12 ECTS points for this project.

1.4 System Context

The already existing infrastructure of the platform is used for the development in this project. The web application is a three tier application, consisting of the layers frontend, backend, and database. The frontend is visible to the user, while the backend is responsible for the business logic and communicates with the database. The database simply stores the data it receives from the backend. Chapter 2 describes the system, which the project is based on, in more detail.

1.5 Link to the Web Application

The web application can be access via the following link:
`https://whatsup.rapperswil-jona.city/`.

Part II

Product Documentation

Chapter 2

Initial Project State

2.1 Introduction

This chapter is based on the 'Studienarbeit' 'What's up in RJ' [1]. The term 'Studienarbeit' is used throughout the document to refer the three-month study project prior to the bachelor thesis. Every time the term 'Studienarbeit' is mentioned, it refers to the same document listed in the bibliography. For a more detailed overview of the initial state of the project, please refer to the 'Studienarbeit'. The 'Studienarbeit' can be seen as the previous iteration of the project, as it was developed by the same authors and in cooperation with the same industry partner. As part of the 'Studienarbeit', a basic implementation of the system was developed. This chapter will provide an overview of the initial state of the project, including the system architecture, the technologies used, and the implementation of the system.

2.2 System Architecture

2.2.1 Overview

The system is designed as a web application, which is accessible through a web browser. It is divided into three main components: the frontend, the backend, and the database. The frontend is responsible for the user interface, the backend for the business logic, and the database for storing the data. The application is hosted on DigitalOcean and available at `https://whatsup.rapperswil-jona.city/`. The initial system architecture is visualized in figure 2.1.

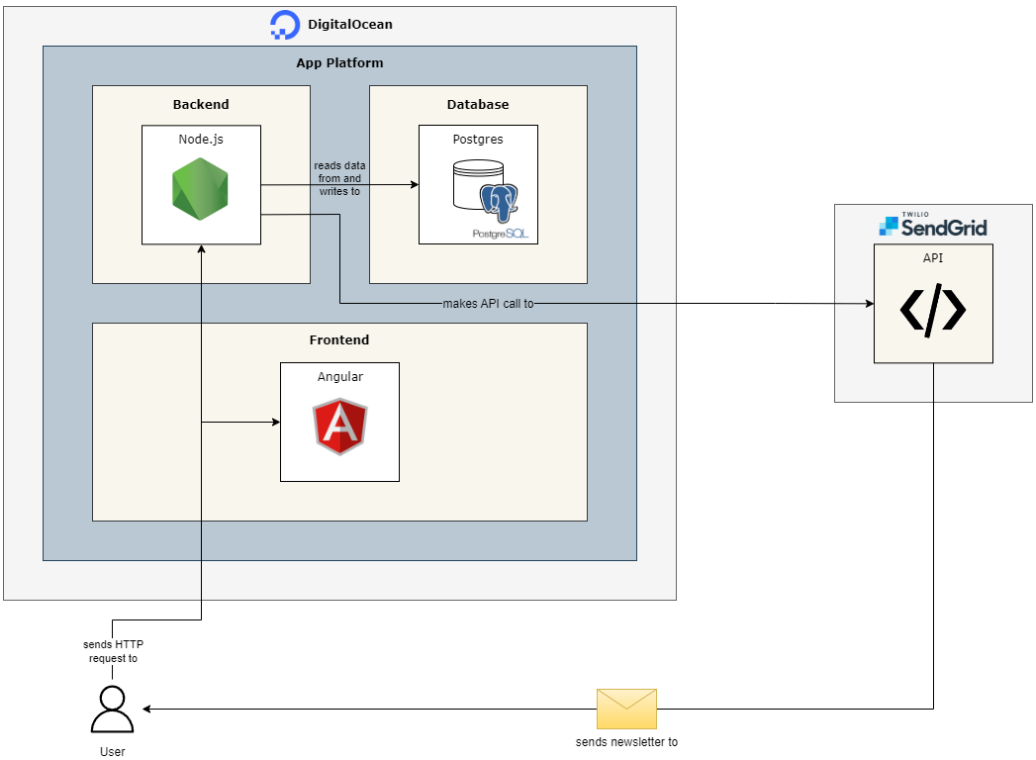


Figure 2.1: System architecture of the initial state of the project

Angular is used to build the frontend, and runs directly on the user's browser, providing a responsive and interactive user interface. The frontend communicates with the backend using HTTP requests, which are handled by the Express.js application, running on top of Node.js. The backend interacts with the database through Sequelize, an Object-Relational Mapping (ORM) library for Node.js. The database is implemented using PostgreSQL, which stores the data in a relational format. Furthermore, SendGrid's API is used to send newsletters to registered, subscribed users. These API requests are handled by the backend on a periodic basis.

2.2.2 Domain Model

The domain model of the initial system is visualized in figure 2.2.

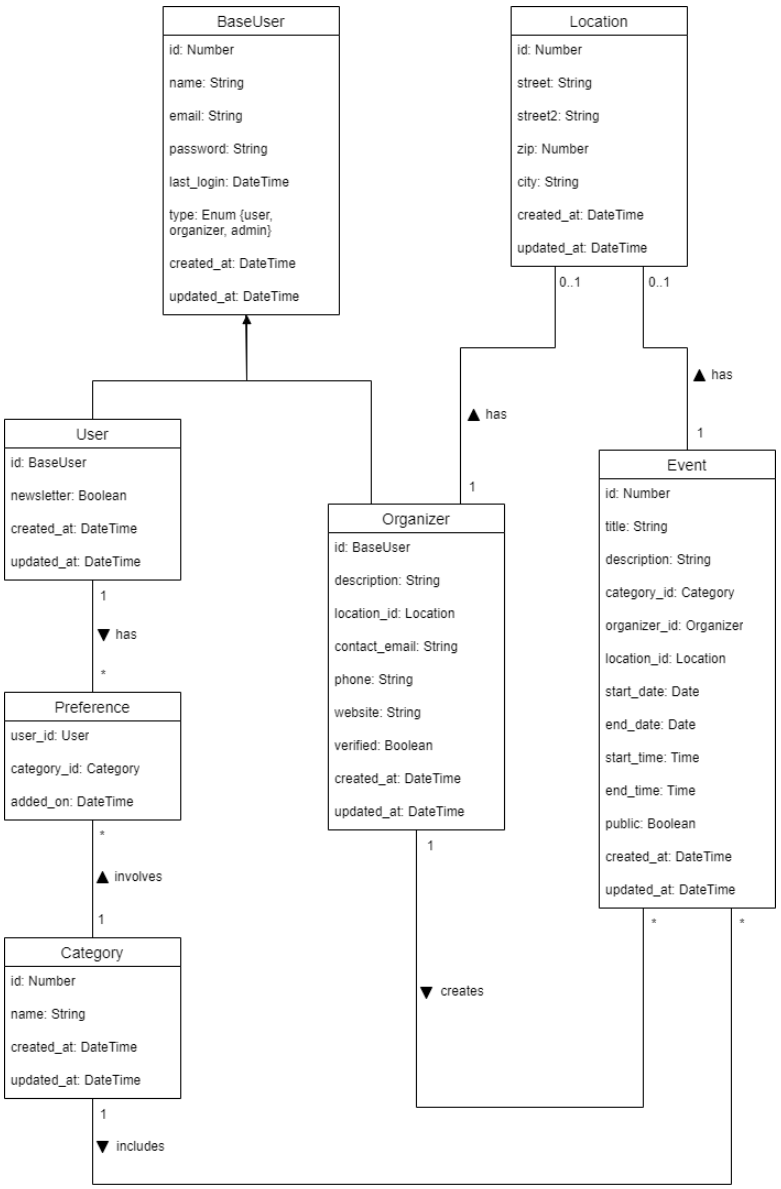


Figure 2.2: Domain model of the initial state of the project

2.3 Technologies

The following technologies are already utilized in the initial implementation of the system:

Technology	Objective	Statement
TypeScript	General	TypeScript is used for the frontend and backend of the application, providing type safety and modern features.
Angular	Frontend	Angular is used for the frontend of the application.

Continued on next page

Technology	Objective	Statement
Tailwind CSS	Frontend	Tailwind is used for styling the frontend of the application.
SCSS	Frontend	SCSS is used in combination with Tailwind to style the frontend of the application.
Node.js	Backend	Node.js is used for the backend of the application.
Express	Backend	Express runs on top of Node.js and is used for the API of the application, which communicates with the frontend.
Sequelize	Backend	Sequelize is used for the ORM of the application, which communicates with the database.
PostgreSQL	Database	PostgreSQL is used for the database of the application.
SendGrid	Email	SendGrid is used for distributing the newsletter to registered, subscribed users.
DigitalOcean	Hosting	DigitalOcean is used for hosting the core of the application.
DigitalOcean App Platform	Deployment	DigitalOcean App Platform is used for automatically deploying the application from the GitLab repository.
Postman	Testing	Postman was used for testing the API of the application.
Git	Version Control	Git is used for version control of the source code.
GitLab	Version Control Platform	GitLab is used for hosting the source code.

Table 2.1: Existing technologies used in the project

2.4 Implementation

The system is implemented as a modular application, with each component being in a separate directory. The frontend is located in the 'frontend' directory, while the backend can be found in the 'backend' directory. The database is hosted on DigitalOcean as a development database. The initial implementation of the application includes a variety of functional and non-functional requirements, which are described in the following sections.

2.4.1 Functional Requirements

The following functional requirements have been implemented in the initial state of the project:

Event Management

The system allows organizers to manage events, including creating, updating, and deleting events. The event management is available on the account page of the organizer. Split into two tabs, ongoing/future and past events, the organizer can see all their events. Events can be created with a click on the '+ New' button in the events section. To create an event, some fields are required to save the event. The only required field for non-public events is the event title. If the visibility is set to public, a start

date as well as the event category are needed. An event supports additional information, namely a description, start and end dates (incl. times), and a location (incl. street, zip code, and city). Once an organizer is verified, they can publish their events, which will then be visible to all users. The organizer can also update and delete their events. The update form is pre-filled with the current event data, which can be edited and saved. The delete button is located at the bottom of the event details page and requires a confirmation before the event is deleted. It is also available on the account page beside each event in the list. Organizers are automatically verified upon registration, as the verification process is not yet implemented.

Authentication

All users and organizers are able to securely register and log in. The registration process requires a name, email, and password, including password confirmation, for both user types. Once registered, users can log in using their email and password.

Account

Both registered users and organizers can self-manage their account. They can update their name and email, change their password, and delete their account. Deleting an account requires a dialogue confirmation before the account is deleted. It will also delete all associated data, including events, preferences and subscriptions. Users can subscribe to or unsubscribe from the newsletter, and edit their category preferences. The category preferences are used to filter the events on the homepage and in the newsletter. Organizers can manage their events by creating, updating, and deleting them, as described in the section 2.4.1. A link to a future profile editor for organizers is also available, but not yet implemented. The account page is accessible through the navigation bar for logged-in users and organizers.

Overviews

A separated, paginated overview of events and organizers exists on the frontend, which can be filtered and sorted. Events can be filtered by category and organizer, and sorted by date, title, and create date. Organizers can be filtered by category only. The overviews are available to all users, including logged-out users.

Event Details

The single page of an event is the detailed view of a published event from a verified organizer. It shows all the available information about the event, including a title, description, start and end dates (incl. times), organizer, location, category, and any additional event information. The single page is accessible to everyone, including logged-out users, as long as the event is public, and the organizer is verified.

Search

The search function allows users to search for events and organizers. A search pop-up is displayed when the user types a search query for a quick search. All search results are displayed on the search page. The page is split into two tabs, one for events and one for organizers. Both lists are paginated and can also be sorted and filtered by the same criteria as the overviews. The search function is available to all users, including logged-out users.

Homepage

The homepage is the entry point of the application. It displays a simple overview of different events and a newsletter subscribe button. The events are split into different lists, including ongoing/upcoming, and newly added events. An additional list is displayed for logged-in users who have set their preferences. The list only contains events that match the user's preferences. The newsletter subscribe button is displayed to logged-out users and registered users who have not subscribed yet. The text of the button changes depending on the user's subscription status. A header and footer are also available on the homepage, which are consistent throughout the frontend. The header contains the homepage link, the search function, and a navigation bar that dynamically changes depending on the user's authentication status. The footer contains the pages of the application and a legal section with the imprint, terms of service, and policies.

Newsletter

Users can subscribe to a newsletter, which sends them regular updates about new events in their preferred categories. This feature is restricted to registered users, logged-out users, organizers and admins cannot subscribe to the newsletter. The preferred categories can be set in the account settings, as described in the section 2.4.1.

2.4.2 Non-Functional Requirements

The following non-functional requirements have been implemented:

- **Collaborative:** The development team implements the features according to the agreed-upon priority in collaboration with the customer.
- **Performance:** The backend handles 1000 requests per minute.
- **Response Time:** Each page loads in under 200ms, except for any pages implementing authentication mechanisms (login, register, account).
- **Browser Compatibility:** The web application runs on Firefox, Chrome, and Safari.
- **Accessibility:** Access is available via the customer-provided domain over the internet.
- **User Satisfaction:** Three out of four test users rated the UI (categories: layout, responsiveness, colour, content) of the application with a minimum score of 8 out of 10, with 10 being the best.
- **Scalability:** The database handles up to 10,000 events and 1,000 users.
- **Error Handling:** Errors do not cause system failures but display an error message and reset the system to its previous state. Every error is logged in the system.
- **Security:** All communication between the frontend and backend is encrypted with an SSL certificate. Input field data is validated before processing, and SQL injection tests on input fields do not reveal vulnerabilities.
- **Data Privacy:** The web application is implemented in compliance with data protection regulations.
- **Password Security:** User passwords are not stored in plain text in the database.

- **User Data Isolation:** When a user logs into the web application, only their data or data they have access to is displayed.
- **Modularity:** Business logic in the backend is modular for extensibility.
- **API Testing:** The backend API was tested using API testing tools. Postman was used to test the API.
- **Deployment:** Implemented functionality (database, backend, frontend, etc.) is deployed automatically after a successful merge into the main branch.

2.5 Summary

The initial state of the project provides a basic implementation of the system, which builds the foundation for the bachelor thesis. Usability has been one of the primary focuses of the initial implementation, including a user-friendly and easy-to-use interface. A variety of functional and non-functional requirements have been implemented, which will be further extended in the current iteration of the project.

Chapter 3

Requirements

3.1 Functional Requirements

The requirements are divided into mandatory and optional requirements. Mandatory requirements need to be implemented, while optional ones do not. The optional requirements are marked with an asterisk (*). All functional requirements are listed as use cases in table 3.1, and figures 3.1, 3.2 and 3.3. They show the connection between actors and use cases in use case diagrams.

Actors

Following actors are defined for the system:

Organizer

Organizers are organizations or individuals that create and manage their own events. After a new organizer account is registered, the account will be verified by an admin. Once verified, they can publish their events. Their main goal is to get users to attend their events by creating appealing events on the platform and having a well-organized profile page.

User

Users are basic visitors of the platform without an account. They are able to do basic actions like searching, filtering and viewing public events. A user is able to create an account on their own and become a registered user.

Registered User

A registered user is similar to a normal user with the key difference of having an account. The account comes with various benefits such as subscribing to a newsletter, choosing event categories they are interested in, and managing their account. The newsletter only includes the chosen event preferences of the registered user, making it a pleasantly tailored experience. Additionally, they get personalized event recommendations on the homepage.

Administrator

The administrator is a central actor who is responsible for the management of the platform. They have the power to manage user and organizer accounts. A primary task is to verify new organizer accounts to ensure that the platform is not abused by malicious actors.

Use Cases

#	Actor	Goal	Description
UC1	Administrator	Manage organizers and users	The administrator is able to view all user and organizer accounts in a centralized admin panel. Additionally, they can verify new organizer accounts.
UC2	User Registered User Organizer Administrator	View responsive frontend	The platform is responsive and can therefore be used on different screen sizes.
UC3	Organizer	Upload pictures	Organizers can upload pictures for their profile and events.
UC4	Organizer	Create organizer profile page	Organizers can create a customized profile page, allowing them to present themselves as desired.
UC5*	Administrator	Approve events	The administrator can approve or reject events before they are published.
UC6*	Organizer	View dashboard and statistics	Organizers can view statistics about their events and profile with a dedicated dashboard.
UC7*	Registered user	Receive personalized newsletter	Registered users receive a personalized, AI generated newsletter with events according to their interests.
UC8*	Organizer	Get event description suggestions	The system suggests a description for the event based on the entered data using a GPT-API.
UC9*	Registered User	View onboarding pages	After signing up, the user can add their preferences and subscribe to the newsletter via onboarding pages.
UC10*	User Registered User Organizer Administrator	View map of events	The platform provides a map view of all ongoing and upcoming events for all actors.
UC11*	Registered User	Buy tickets	Registered users can click on a button to buy tickets on an external platform.
UC12*	Organizer Registered User	Email verification	The system sends an email to verify the email address of the user and organizer.
UC13*	Admin	Category management	The administrator can manage the categories of events.
UC14*	Organizer	Multiple categories for events	Organizers can assign multiple categories to their events.
UC15*	—	Crawler for events	The system crawls external websites for events and adds them to the platform.

Continued on next page

#	Actor	Goal	Description
UC16*	User Registered User Organizer Administrator	Internationalization	The platform is available in multiple languages.
UC17*	Registered User	Onboarding category selection	The design of the category selection in the onboarding process for the registered user should be improved. The implemented dropdown should be replaced with an overview of the categories.

Table 3.1: All use cases

The use cases UC12* to UC17* were defined during the project and are therefore not part of the initial task description. They are listed in a separate diagram in figure 3.3.

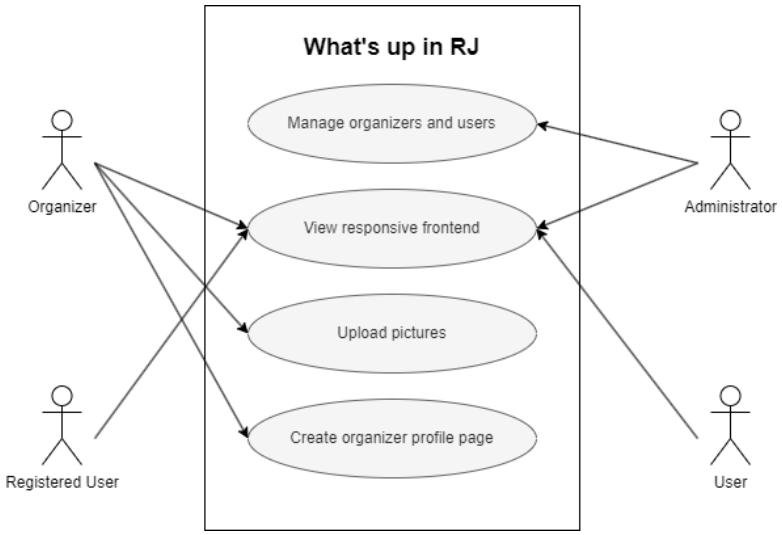


Figure 3.1: Use case diagram showing required use cases

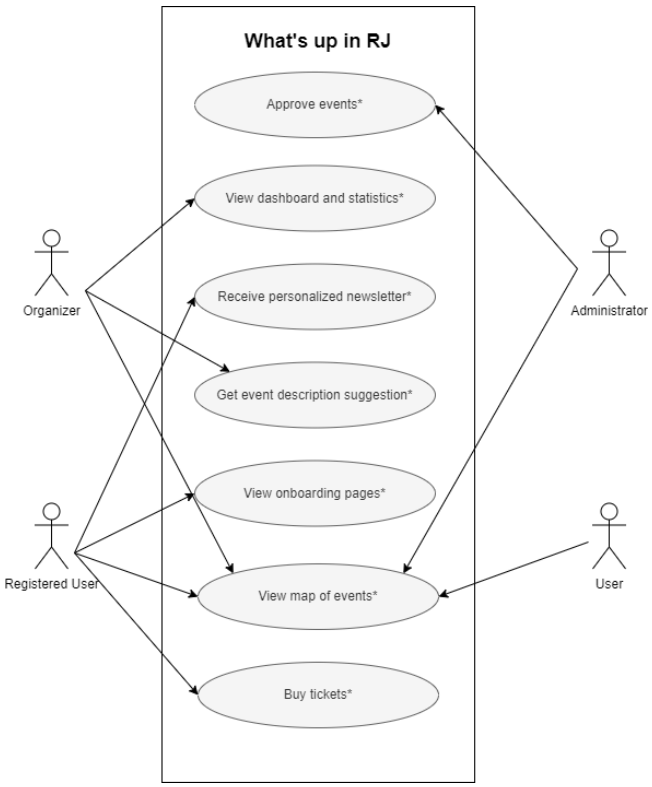


Figure 3.2: Use case diagram showing optional use cases

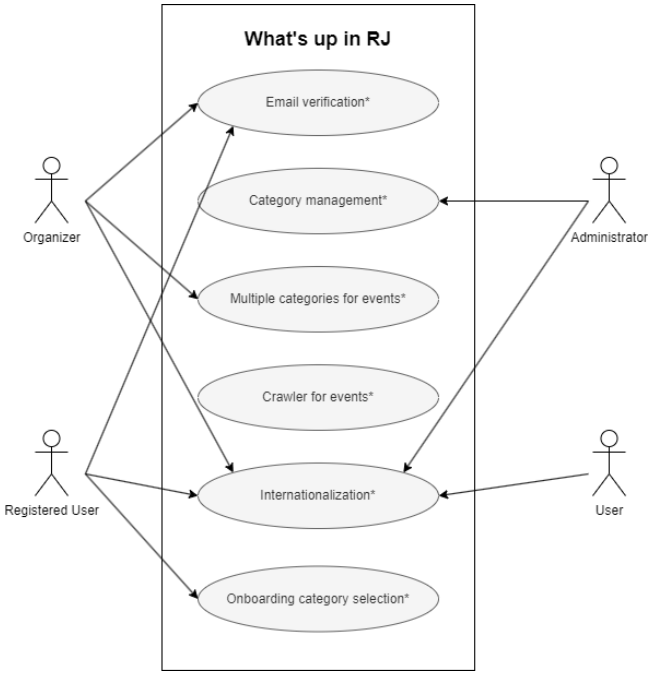


Figure 3.3: Use case diagram showing optional use cases defined during the project

3.2 Non-Functional Requirements

The non-functional requirements, short NFR, are evaluated according to ISO/IEC 25010:2011.

Collaborative

NFR1 — Collaborative: The development team implements the features according to the agreed-upon priority in collaboration with the customer.

Acceptance Criteria: All required features are implemented.

Performance

NFR2 — Performance: The backend should handle 1000 requests per minute.

Acceptance Criteria: 1000 requests per minute are handled without errors.

Response Time

NFR3 — Response Time: Each page should load in under 200ms.

Acceptance Criteria: The pages load in under 200ms.

Responsiveness

NFR4 — Responsiveness: The web application should be responsive on mobile, tablet, and desktop.

Acceptance Criteria: Responsive design is implemented and tested on all required devices.

Browser Compatibility

NFR5 — Browser Compatibility: The web application should run on Firefox, Chrome, and Safari.

Acceptance Criteria: The web application is tested on Firefox, Chrome, and Safari.

Availability

NFR6 — Availability: Access should be available via the customer-provided domain over the internet.

Acceptance Criteria: The web application is accessible via a domain, provided by the customer, over the internet.

User Satisfaction

NFR7 — User Satisfaction: Three out of four test users should rate the UI (categories: layout, responsiveness, color, content) of the application with a minimum score of 8 out of 10, with 10 being the best.

Acceptance Criteria: User satisfaction is measured with a survey and the results are evaluated.

Scalability

NFR8 — Scalability: The database should handle up to 10,000 events and 1,000 users.

Acceptance Criteria: The database handles 10,000 events and 1,000 users without errors.

Error Handling

NFR9 — Error Handling: Errors should not cause system failures but display an error message and reset the system to its previous state. Every error should be logged in the system.

Acceptance Criteria: Errors are logged, and a message is displayed to the user.

Security

NFR10 — Security: All communication between the frontend and backend should be encrypted with an SSL certificate. Input field data must be validated before processing, and SQL injection tests on input fields should not reveal vulnerabilities.

Acceptance Criteria: SSL encryption is implemented and tested. Input fields are validated and tested for SQL injection vulnerabilities.

Privacy

NFR11 — Data Privacy: The web application should be implemented in compliance with data protection regulations.

Acceptance Criteria: Privacy regulations are implemented and tested.

Password Security

NFR12 — Password Security: User passwords should not be stored in plain text in the database.

Acceptance Criteria: Passwords are hashed and salted before storing them in the database.

User Data Isolation

NFR13 — User Data Isolation: When a user logs into the web application, only their data or data they have access to should be displayed.

Acceptance Criteria: User data is isolated and only accessible to the user that owns the data.

Modularity

NFR14 — Modularity: Business logic in the backend should be modular for extensibility.

Acceptance Criteria: Business logic is modular and extensible.

API Testing

NFR15 — API Testing: The backend API should be tested using API testing tools.

Acceptance Criteria: The backend API is tested using API testing tools.

Deployment

NFR16 — Deployment: Implemented functionality (database, backend, frontend, etc.) should be deployed.

Acceptance Criteria: The implemented functionality is deployed on DigitalOcean.

3.3 Minimum Viable Product

The minimum viable product, short MVP, consists of:

- Users and organizers can be managed by the administrator.
- The platform is responsive and can be used on different screen sizes.
- Organizers can upload pictures for their profile and events.
- Customized profile pages can be created by organizers.

Chapter 4

Domain Analysis

4.1 Domain Model

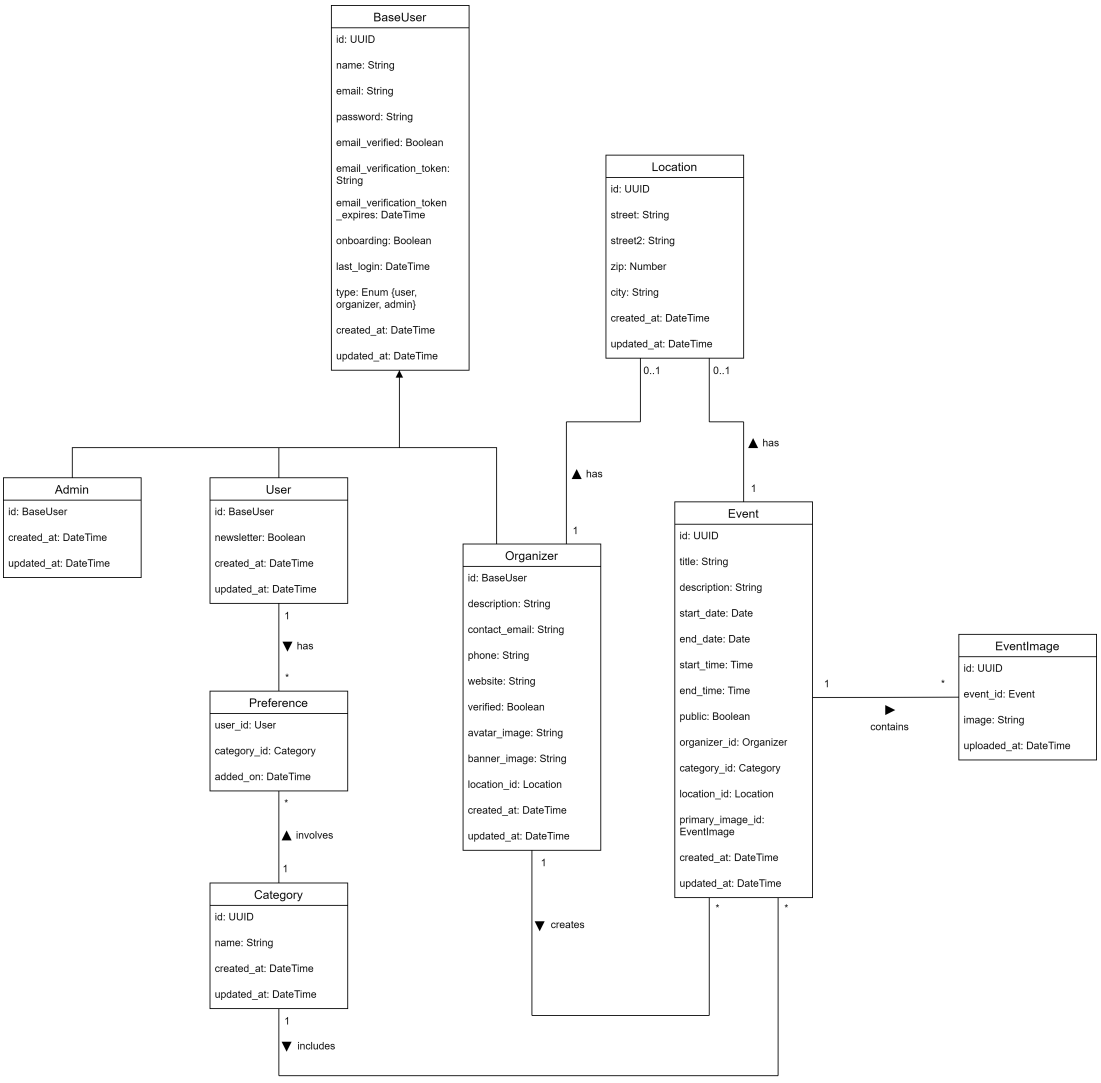


Figure 4.1: Domain model for the MVP

This domain model extends the model from the initial state of the system, which is shown in figure 2.2. Therefore, the already existing entities and attributes are not described in detail. The model contains all elements of the MVP and the functional requirements. Only the implemented optional requirements are included in the model. All other optional requirements are not included.

The base user entity is extended to include the main actors of the system, namely the user and the organizer. A user has a preference that is based on a category of an event. An organizer may also have a description, a contact email, a website or a location. These fields were already present but not actively used in the initial state of the system. Additionally, an organizer can now also have an avatar image and a banner image. All the aforementioned fields are optional, and can be added by an organizer as they see fit.

The event entity is extended with a relation to the new 'EventImage' table, which is required to store an event's images. This 'EventImage' table consists of an 'id', 'event_id', 'image', and 'uploaded_at' attribute. The 'id' attribute is a unique identifier, the primary key, for each image, while the 'event_id' attribute links to the event the image is related to. With the 'image' field the key of the image is stored, which has been generated and is used on the object store. It is needed to retrieve the image after it has been uploaded. Finally, the last field 'uploaded_at' can be used for further purposes at a later date, such as analytics, etc.

The administrator is a distinct type of base user defined within the base user's 'type' field.

The 'Event' table is extended with the 'primary_image_id' field, which is a foreign key to the 'EventImage' table. This field is used to store the primary image of an event that will be displayed on the event cards and the banner of the event single page.

To implement the email verification feature, the 'BaseUser' entity is extended with the 'email_verified' field, which is a value of type 'Boolean'. In addition, the 'email_verification_token' and 'email_verification_token_expires' fields are added to store the token and the expiration date of the token.

The field 'onboarding' is added to the 'BaseUser' entity to implement the onboarding feature. It holds a value of type 'Boolean', which indicates whether the user has completed the onboarding process or not.

Chapter 5

Architecture

The architecture of the project is divided into several sections, each of which are described in detail in the following sections. This chapter provides an overview of the existing architecture components to be reused and further developed, along with detailed explanations of the architectural changes planned for this project iteration. For additional details on the initial architecture, please refer to the section 2.2 of this document and the 'Studienarbeit'. The C4 model is used to describe the architecture of the project.

5.1 Overall Architecture

The overall architecture of the project remains the same as in the 'Studienarbeit'. The project is still divided into a frontend and a backend, and the deployment architecture also remains the same. A new feature that is planned for this project iteration is the addition of a new storage solution for user-uploaded files, such as organization logos and banners, and event images. Images for other purposes may follow in the future. The new storage solution uses DigitalOcean Spaces, an object storage service provided by DigitalOcean. Additionally, the object store comes with a Content Delivery Network (CDN).

The system context diagram in figure 5.1 shows the system as a black box and its interactions with external systems. It conveys the high-level architecture of the system by only including the main components: the users, the internal systems, and the external systems. The users are the main actors of the system, and they interact with the system through the 'What's up in RJ' system. All other systems are considered external systems, as they are not part of the core application, but they are used to provide additional functionality to the users. The email system is also considered an external system, and it is utilized to send emails to specific users. All these parts already exist, and the architecture's new part is the object storage system. This system is considered an external system, and it is used to store and serve files to the users.

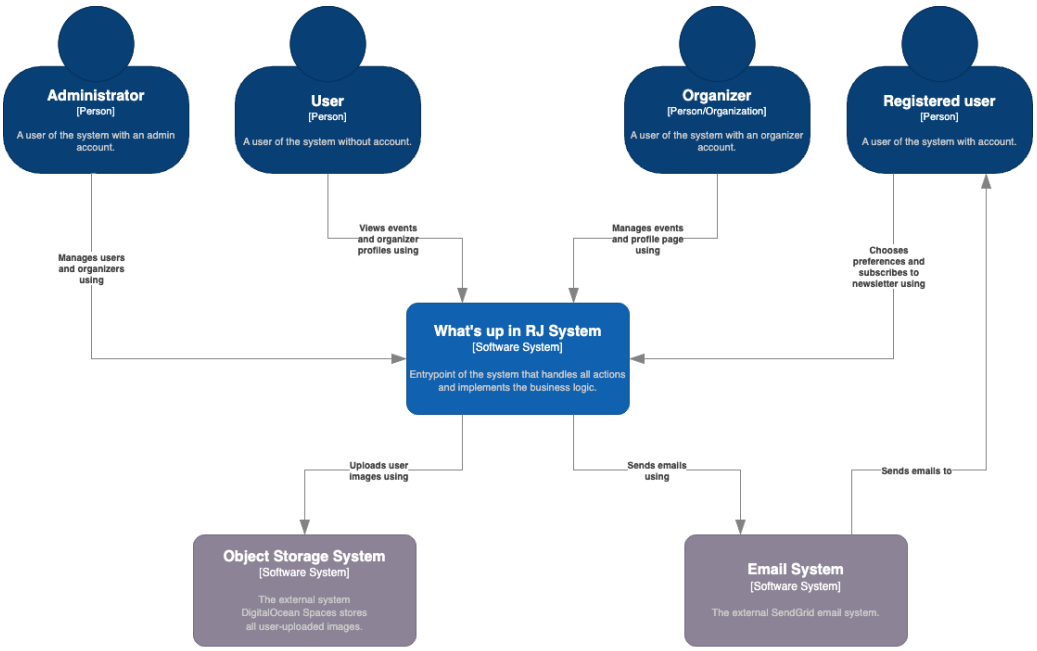


Figure 5.1: System context diagram

The container diagram in figure 5.2 displays the internal structure of the system, and how the system is divided into components. The web application is the initial entry point for the users. It is DigitalOcean's web server that serves the frontend application to the users. The frontend application is a single-page application written in Angular. Users interact with the frontend application, which in turn interacts with the API application. The backend API is a RESTful API written in Express, which is a web application framework for Node.js. The database is based on Postgres, and it is used to store most of the data. Only the backend component can interact with the database. The email system provided by SendGrid is used to send emails to registered users and organizers. It delivers periodic newsletter subscriptions, a single email to the organizers when their account has been verified, and email verification emails to all users, except admins, when they sign up, change their email address, or request a new email verification. Each email is instructed by the API application to be sent to the users through the SendGrid API. These containers existed in the initial state and are now further developed or reused in this project iteration. The object storage system is a completely new container that stores user-uploaded files and serves them to the users through the Spaces CDN. The CDN is not included in the diagram, as it is a part of the object storage system.

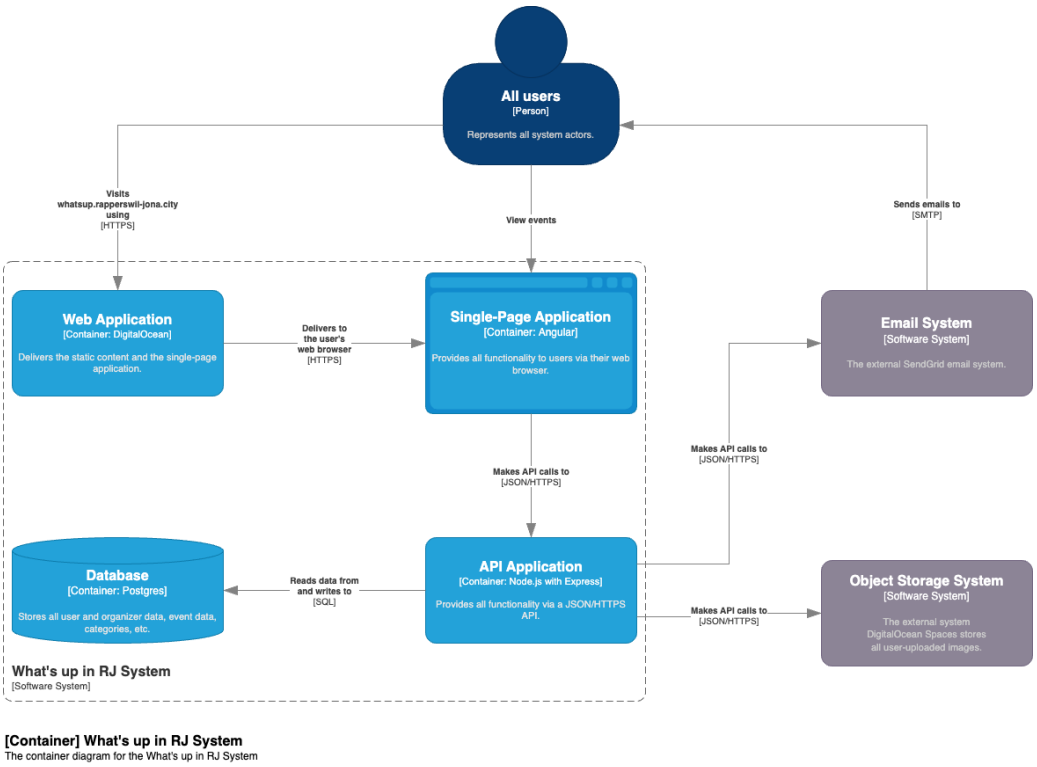


Figure 5.2: Container diagram

5.2 Backend Architecture

The backend architecture generally remains the same as in the 'Studienarbeit'. It contains the whole business logic of the system, and it is responsible for handling incoming requests from the frontend application. The backend API is extended to support the new features and improvements that are planned for this project iteration. The database schema is also adjusted accordingly, as seen in the chapter 4.

The component diagram in figure 5.3 shows the internal structure of the backend component, as well as the interactions between the subcomponents and other components and systems. The backend component is divided into several subcomponents, each of which is responsible for a specific part of the system. Once a request is received by the API application, the routers forward the request to the appropriate subcomponent based on the request's path and method. The controllers are responsible for handling the request, and they interact with the helpers and stores to perform the necessary operations. Additionally, the controllers directly interact with the object storage system and its CDN to store and serve user-uploaded files. The helpers contain useful functions that are used by the controllers to perform common operations, such as sending emails, generating tokens, and handling the database connection on startup. Standalone functions are also included in the helpers, and they are used to performing operations that occur periodically, or just once at startup. The stores are responsible for interacting with the database, and they contain the database queries and operations. To interact with the database, Sequelize is used, which is a promise-based Node.js ORM for multiple databases, including Postgres.

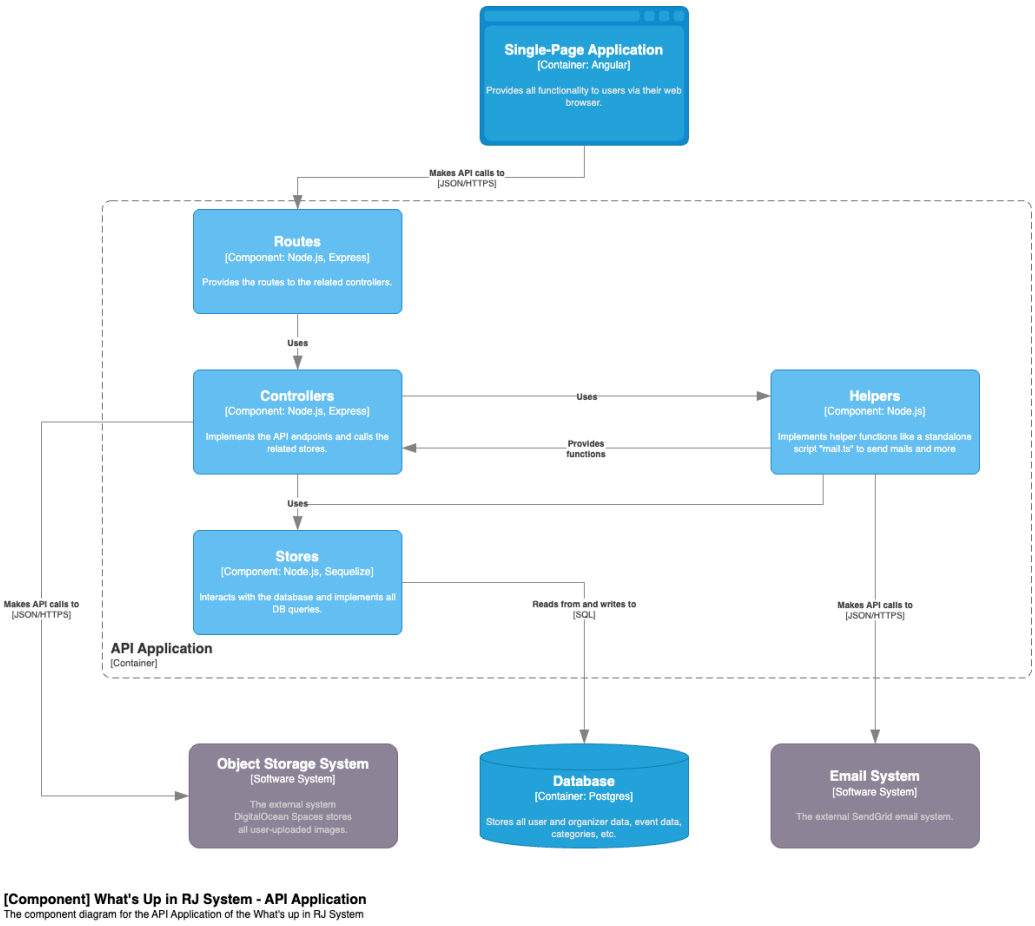


Figure 5.3: Backend component diagram

The folder structure of the backend is based on its subcomponents, and it is organized in a way that makes it easy to find and maintain the code. This structure already exists and will be reused in this project iteration. In figure 5.4, the folder structure of the backend is shown, containing only the main directories. The 'public' directory contains static files that are served directly to the users without additional processing, such as the 'robots.txt' file. TypeScript's custom definitions are included in the 'typings' directory, and they are used to defining custom types and interfaces that are used throughout the backend. The 'src' directory contains the main source code of the backend, and it is divided into the subcomponents: 'controllers', 'helpers', 'routes' and 'stores'. Additional directories are also included, which support the main subcomponents, such as the 'errors' directory, which contains custom error classes, the 'middlewares' directory, which includes custom middleware functions, and the 'models' directory, which defines the database models and associations.

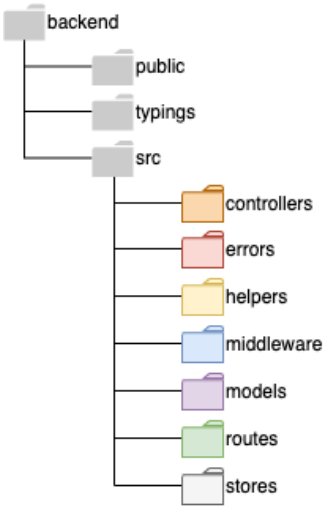


Figure 5.4: Backend directory structure

5.3 Frontend Architecture

A minor refactoring of the frontend architecture is done to keep the codebase clean and maintainable. In the 'Studienarbeit', the frontend application was a single module, and all the components, services, and other files were included in the same module. This makes the codebase harder to maintain and extend, as the codebase grows larger. To solve this issue, the frontend application is divided into several modules, each of which is responsible for a specific part of the system. This also enables additional features, such as lazy loading, which can be used to load the modules only when they are needed, and not when the application is initially loaded, thus reducing the initial load time of the application. The 'root' module is the main module of the application, and it is responsible for bootstrapping the application and loading the other modules. Meanwhile, the 'core' module is responsible for providing the core functionality of the application. It contains the services, guards and various helpers that are used throughout the application. The 'shared' module contains the shared components, directives, models, and pipes that are used throughout the application. All other modules are feature modules, and they are responsible for providing the functionality of the specific feature they are responsible for. For example, the 'events' module provides all functionality for the events feature, such as the event overview, event details, and event creation. The folder 'assets' includes two new directories 'theme-base' and 'theme'. Both folders contain CSS styles for all 'PrimeNG' components, which are used throughout the application. For example, the custom dropdown menus as well as the carousel of the event images. The 'theme-base' folder contains the base styles of the components, while the 'theme' folder is meant to overwrite the base stylings for your own custom theme.
The folder structure diagram in figure 5.5 shows the internal structure of the frontend component, and how the code is organized.

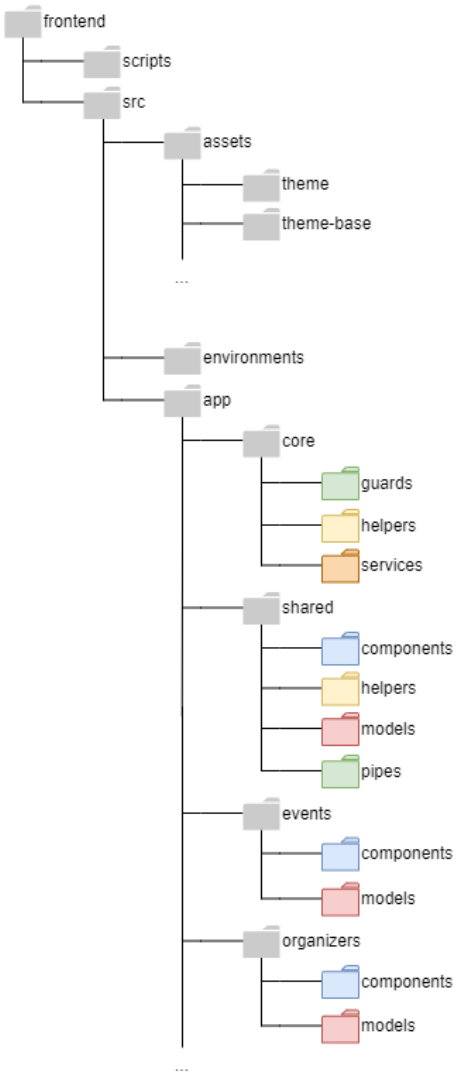


Figure 5.5: Frontend directory structure

5.4 Deployment Architecture

As shown in figure 5.6, the deployment structure displays the architecture of the project from the perspective of DigitalOcean. The project is deployed on DigitalOcean, using App Platform and Spaces, both provided by DigitalOcean. A static web page, also known as the frontend, is served by DigitalOcean's web server, no processing is done on the server. An additional benefit of using DigitalOcean's web server is that it uses a global content delivery network to serve the static files to the users, which makes the files load faster. The backend is a web service running Node.js, in need of processing power and memory, and DigitalOcean's App Platform serves it through the path '/api'. It does not support serving via a content delivery network, as it is a web service, and not a static web page. The web service is located in Frankfurt, Germany, the nearest location to the main user base. Internally, the backend uses port 3000, and the port is exposed to the public through port 443 by the web server. To secure the communication between the frontend and the backend, the backend is served over HTTPS, and the frontend communicates with the backend through the HTTPS protocol. A single Postgres development database runs inside App Platform on port 5432, and is never exposed to the public.

The object storage system is DigitalOcean Spaces, conveniently provided by DigitalOcean itself. This deployment of the frontend, backend, and database on the App Platform has already existed. The Spaces object store is a new part of the deployment. It saves user-uploaded files, and the new Spaces CDN serves the files to the users through its global network of edge locations. Thanks to the content delivery network, the files are served to the users faster and more efficiently. If a file is not found in the CDN, it automatically fetches the file from the object store and caches it in the CDN for future requests. The application is available under a custom domain, and it is secured with an SSL certificate provided by DigitalOcean.

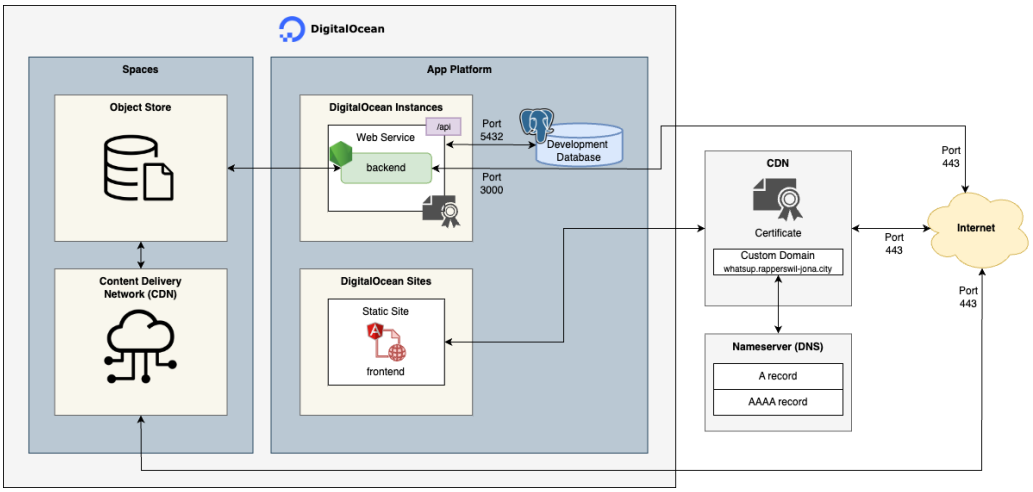


Figure 5.6: Deployment architecture diagram

5.5 Request Flow

The flows of the existing systems are the same as in the 'Studienarbeit', and they are not changed in this project iteration. With the addition of the object storage system, the following diagrams show the request flows of the new system.

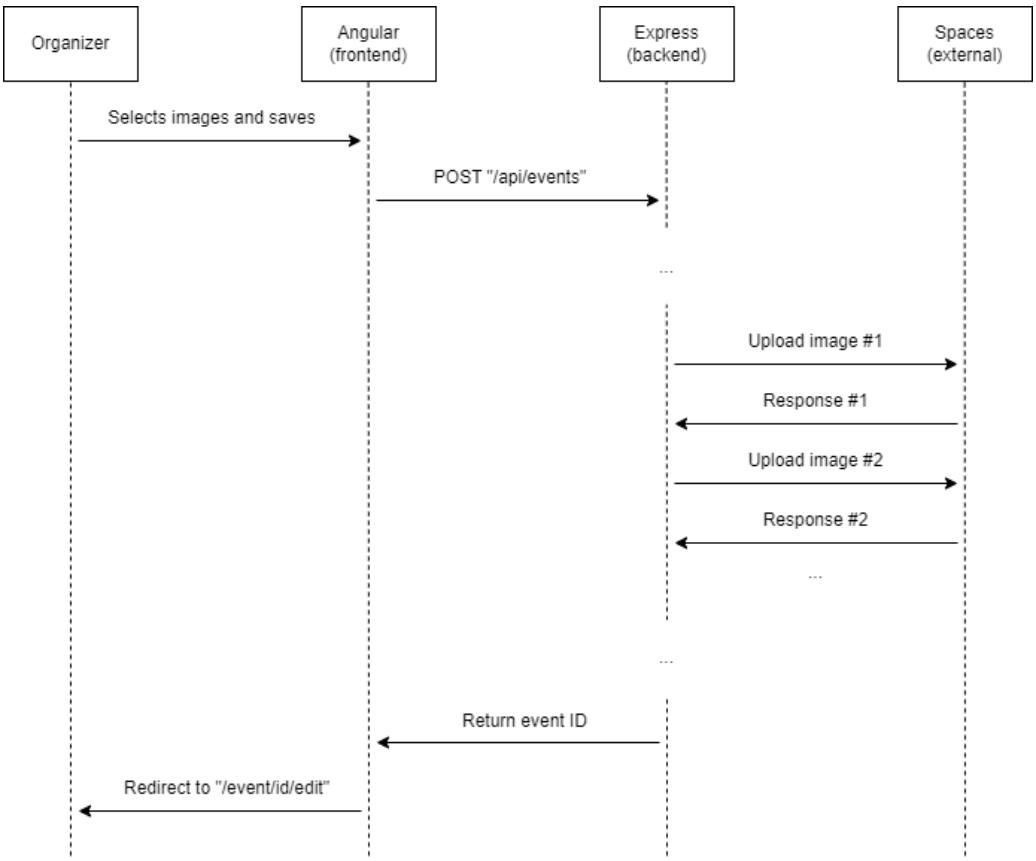


Figure 5.7: Request flow diagram: Image upload

Figure 5.7 shows the request flow of the image upload feature. When an organizer creates a new event, selects an image, and proceeds to save the event, the frontend application sends a request to the backend API to save the content of the event, including the image. The backend API temporarily stores the image in the system memory before uploading it to the object storage system. The object storage system stores the image in the object store and returns a success response to the backend API. The backend API then saves the image's unique name in the database, continues to save the rest of the event, and returns the event ID to the frontend application.

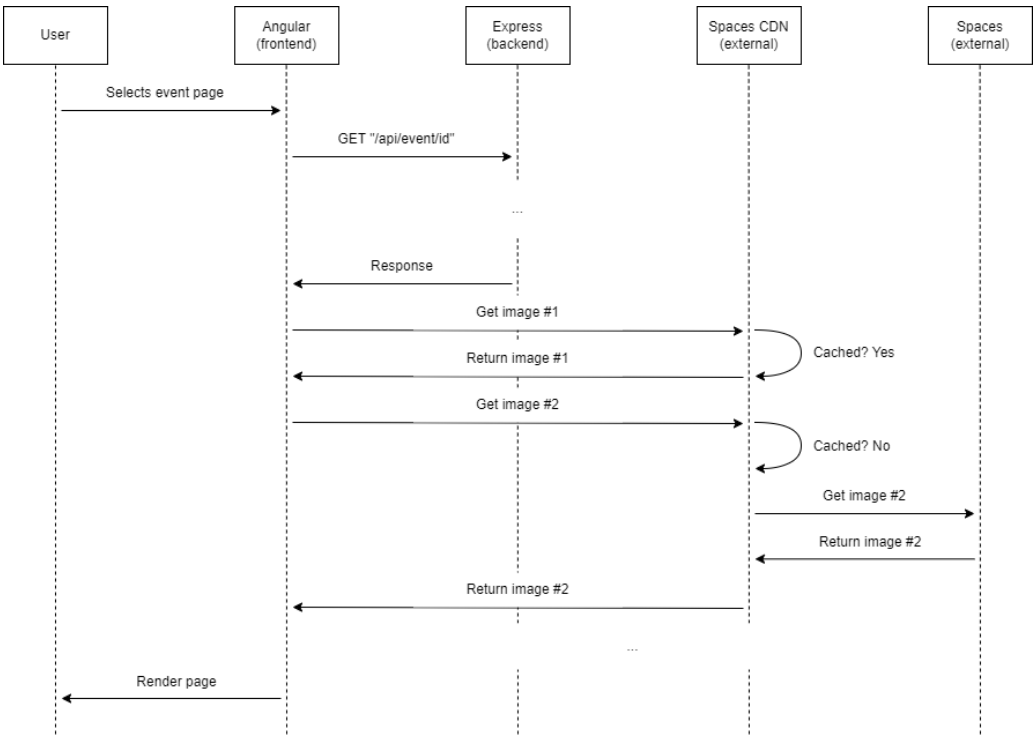


Figure 5.8: Request flow diagram: Image serve

In figure 5.8, the request flow of loading an image from the Spaces CDN can be seen. When a user requests an event page, the frontend application sends a request to the backend API to load the event details. The backend API retrieves the event details from the database and returns them to the frontend application. The frontend application then sends a request directly to the Spaces CDN to load the event image. If the image is found in the CDN cache, it is instantly served to the user. If the image is not found in the CDN cache, the CDN fetches the image from the object store and caches it for future requests. The image is then served to the user.

5.6 User Interface Mock-ups

New user interface mock-ups are created in this project iteration to reflect the new features and improvements. Figma was used to create the mock-ups, a web-based design tool that is used to create user interface designs, prototypes, and more. The types of mock-ups are split into two categories: desktop and mobile. The desktop mock-ups are designed for larger screens, such as desktop computers and laptops, and only contain the new features. On the other hand, mobile mock-ups are designed for smaller screens, such as smartphones and tablets, and contain all new and existing features. These mock-ups do not reflect the final design of the new features, but they provide a general idea of how the new features will look and work. Optional features are not included as they have not been definitely planned for this project during the time of creating the mock-ups.

5.6.1 Desktop Mock-ups

The following sections and figures show the desktop version of the new features.

Organizer Account Pages

As shown in figure 5.9, the account page of the organizer gets a new label in order to show the verification status of the organizer. It is shown at the very top of the page, and it is visible to the organizer only. The label is green if the organizer is verified, and red if the organizer is not verified yet.

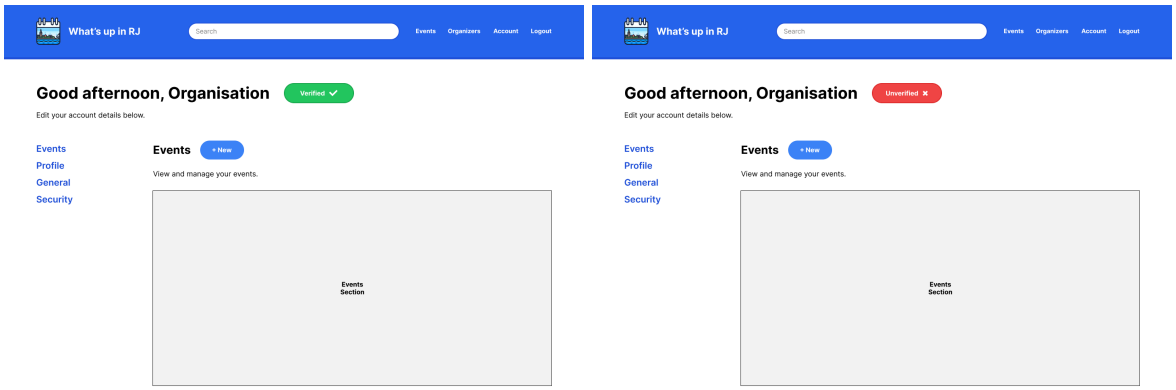


Figure 5.9: Desktop mock-ups: Organizer account page

Additionally, shown in figure 5.10, the organizer now has access to the profile editor, which is a new page that allows the organizer to edit their profile appearance and details. The profile editor contains input fields for the organizer description, organizer avatar, and organizer banner. The organizer avatar and banner are uploaded images, and the organizer description is a text input field.

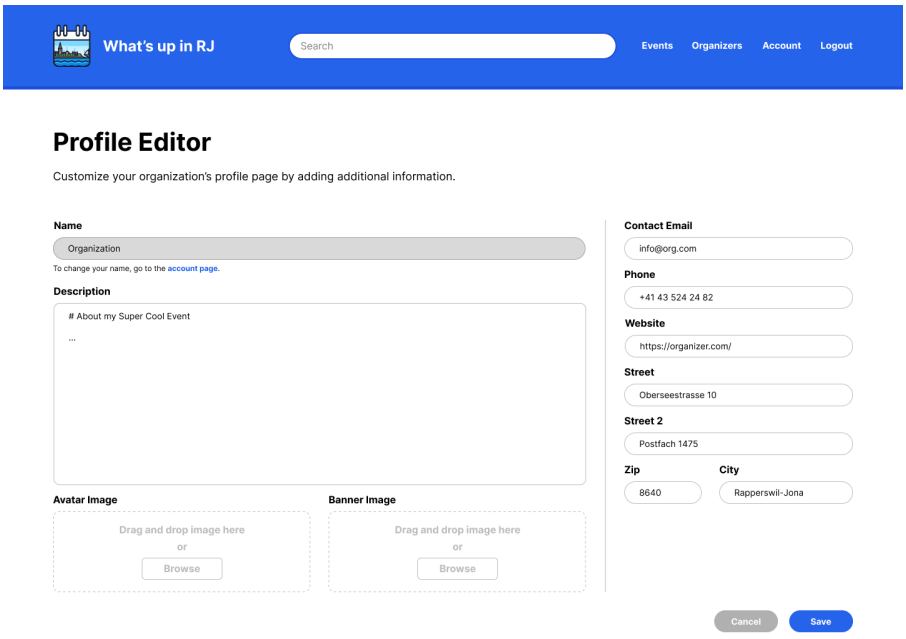


Figure 5.10: Desktop mock-up: Organizer profile editor

Organizer Single Page

In figure 5.11, the organizer single page is shown, which is another new page that displays the profile of the organizer to the users. The organizer avatar and banner are displayed at the top of the page, and the organizer description is displayed below the name and images. On the right-hand side of the page, the organizer's contact email, phone number, website, and location are displayed, if they are provided by the organizer.

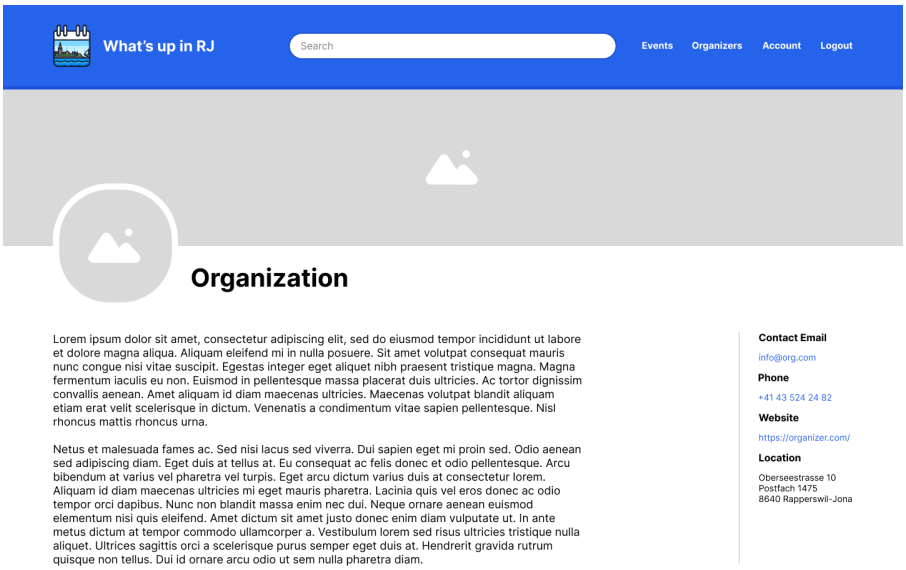


Figure 5.11: Desktop mock-up: Organizer single page

Event Pages

The event creation page gets a new input field in order to upload images for an event, see figure 5.12. The input field is located below the event description. When images are uploaded, they are displayed to the organizer in a preview, and the organizer can remove the images if they want to. Additionally, the organizer can choose a primary image for the event, which is the image that is displayed whenever the event is shown to a user, e.g. on the event overview page.

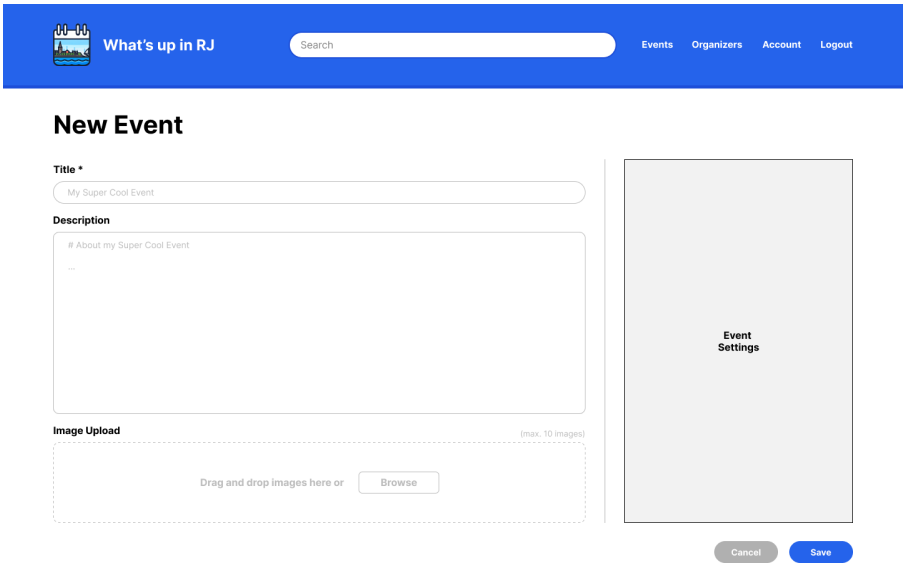


Figure 5.12: Desktop mock-up: Event creation page

The event edit page gets the same input field as the event creation page, in order to upload images for an event. Figure 5.13 visualizes what the image preview, primary image selection and image deletion look like.

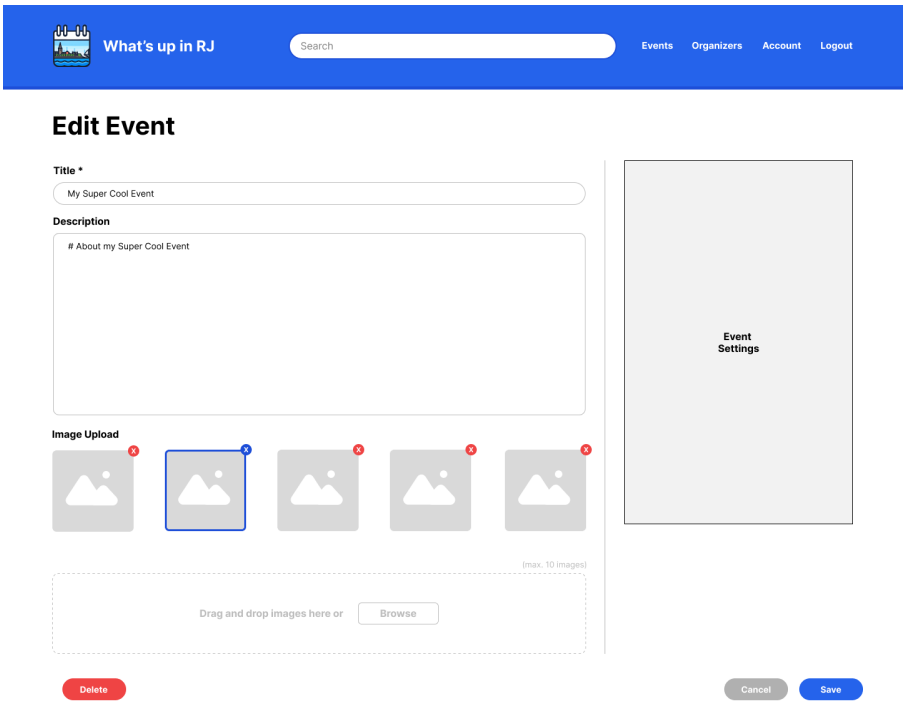


Figure 5.13: Desktop mock-up: Event edit page

Figure 5.14 displays the new section on the event single page, which shows all images of the event. The images are displayed inside a carousel, and the user can navigate through the images by clicking on the arrows on the left and right side of the carousel. At the top of the page, the primary image of the event is displayed in a larger size than the other images.

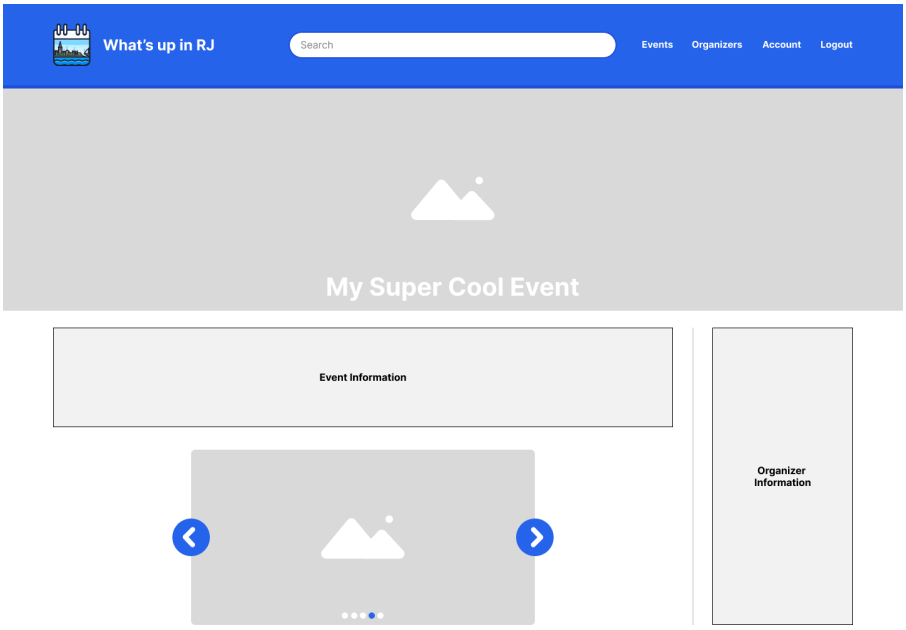


Figure 5.14: Desktop mock-up: Event single page

Admin Panel

A new admin panel is created in order to manage the verification status of the organizers as well as view all users of the system, see figure 5.15. To get to the admin panel, a red bar is shown below the header of the page, and it is only visible to administrators. The admin panel overview contains four tabs: 'Unverified Organizers', 'All Organizers', 'All Users', and 'Administrators'. The 'Unverified Organizers' tab contains a list of all unverified organizers, and the admin can verify them by clicking on the toggle button. The 'All Organizers', 'All Users' and 'Administrators' tabs contain a list of all organizers, users, and admins, respectively, with basic information.

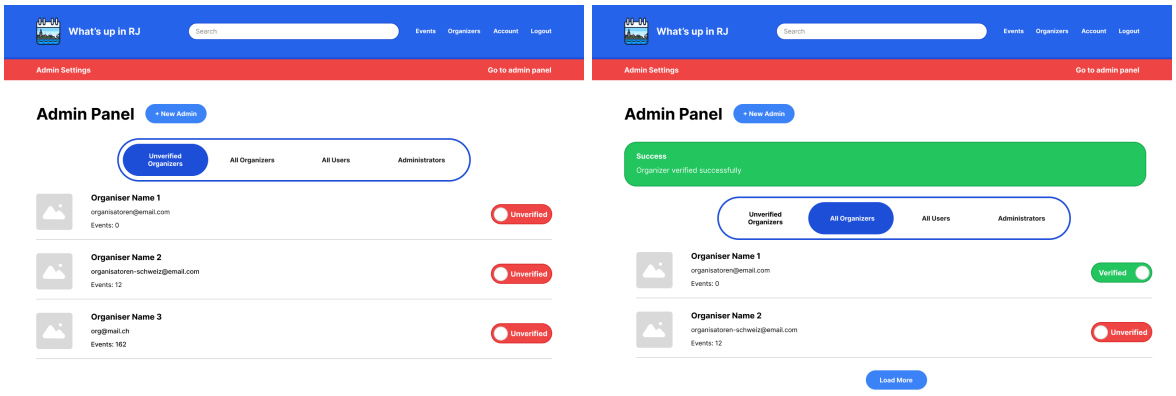


Figure 5.15: Desktop mock-ups: Admin panel overview

The admin panel overview also contains a ‘New Admin’ button, which allows the admin to create a new admin account, see figure 5.16. When the button is clicked, a new page is shown to the admin, and they can enter the information of the new admin. Parallel to the user sign up page, the admin can enter the name, email, password, and password confirmation of the new admin.

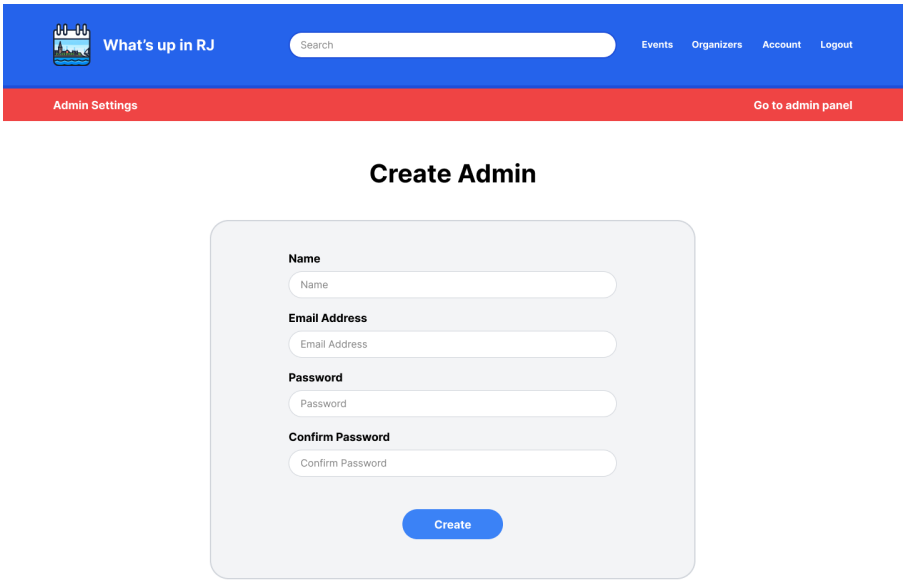


Figure 5.16: Desktop mock-up: Admin panel create admin page

5.6.2 Mobile Mock-ups

The following sections and figures show the responsive design of the entire web application. They do not contain any new functionality that is not already shown in the desktop mock-ups or the existing application. However, they show how the existing functionality is displayed on smaller screens.

General

The general layout of the mobile mock-ups is shown in figure 5.17, including the header, footer, and home page. A hamburger menu is used to display the menu items on smaller screens, and the menu items are displayed in a list. The menu covers the entire screen when opened. The ‘Admin Panel’ item inside the menu is only visible to administrators. The footer is the same as the existing footer, but it has been scaled down to fit the smaller screen.

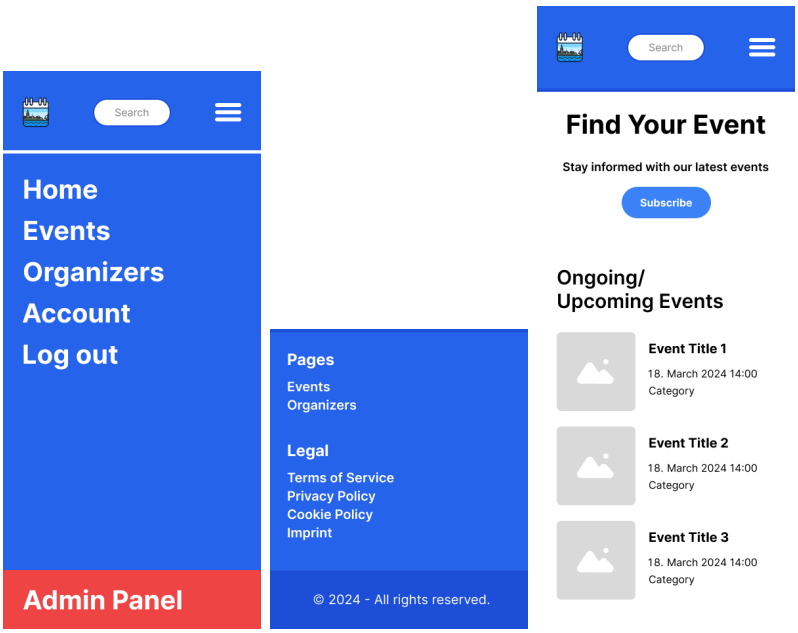


Figure 5.17: Mobile mock-ups: Header, footer, and home page (left to right)

Account

The account page of the registered user is shown in figure 5.18. It only includes the preferences section, where the user can change their event interests and newsletter subscription. The save button is fixed to the bottom of the screen, and it is only visible when the user has made changes.

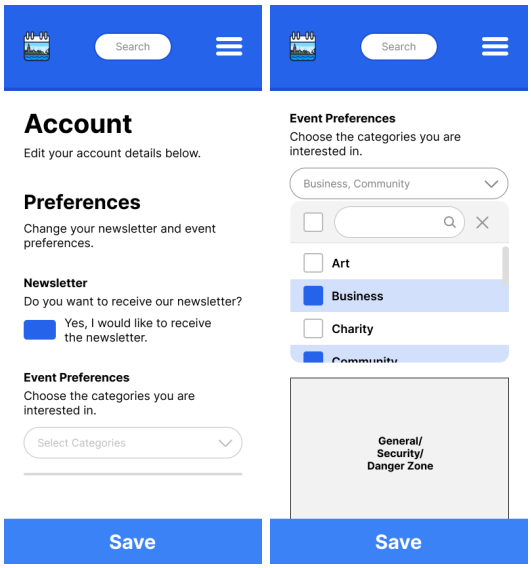


Figure 5.18: Mobile mock-ups: Account page (registered user)

The account page of the organizer is shown in figure 5.19. It includes the events and the profile editor sections.

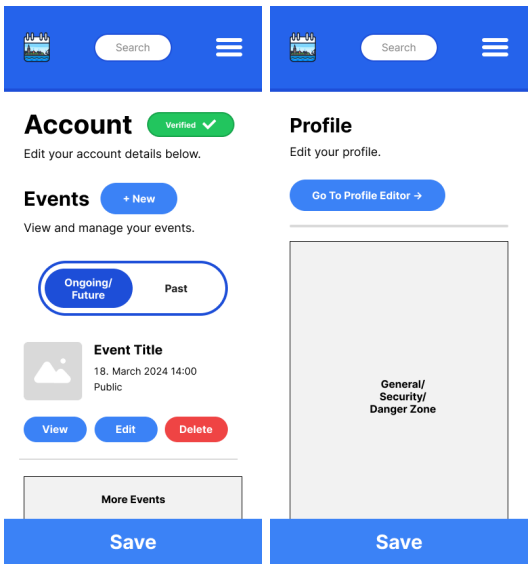


Figure 5.19: Mobile mock-ups: Account page (organizer)

The remaining sections of the account page are shown in figure 5.20. These sections are available to all users, and they include the general, security, and danger zone sections.

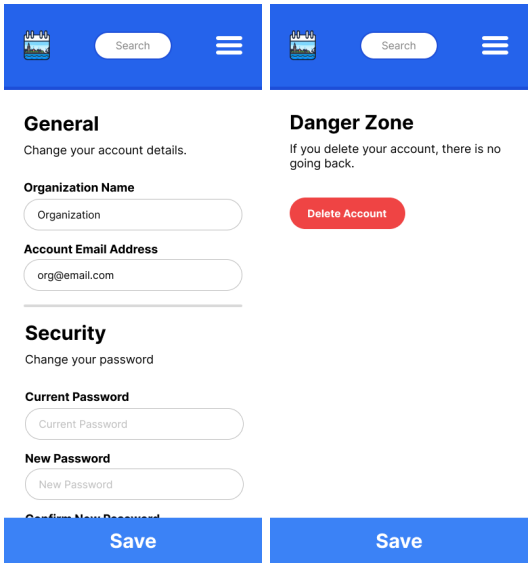


Figure 5.20: Mobile mock-ups: Account page (general)

An example of the organizer profile editor page is shown in figure 5.21. The page contains the same input fields as the desktop version, but they are displayed in a single column.

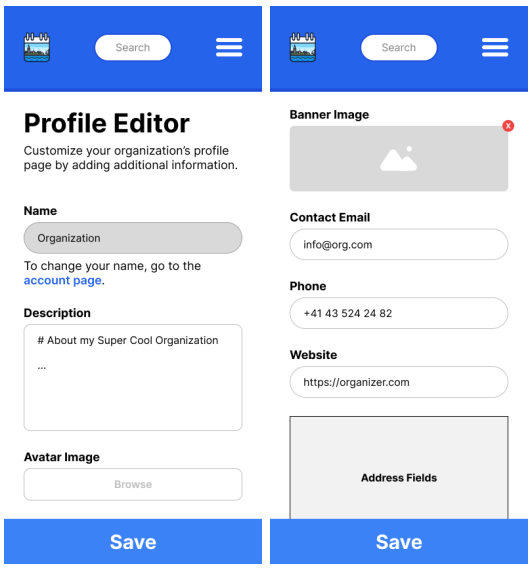


Figure 5.21: Mobile mock-ups: Organizer profile editor

Authentication

The sign-up and login pages are shown in figure 5.22.

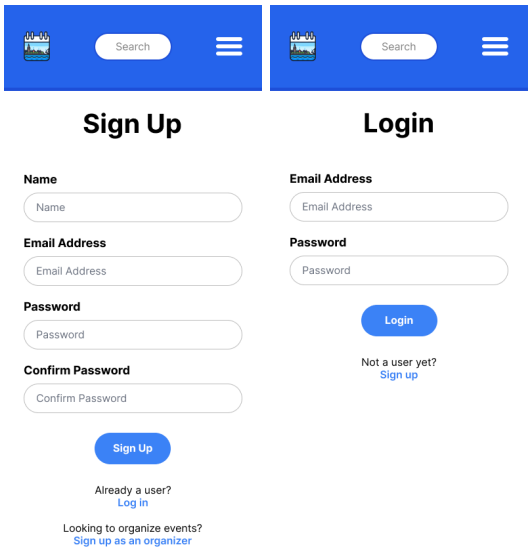


Figure 5.22: Mobile mock-ups: Sign up and login pages

Events

The event overview page is shown in figure 5.23. The event filters are packed into a new menu, which is opened by clicking on the filter button at the bottom of the screen.

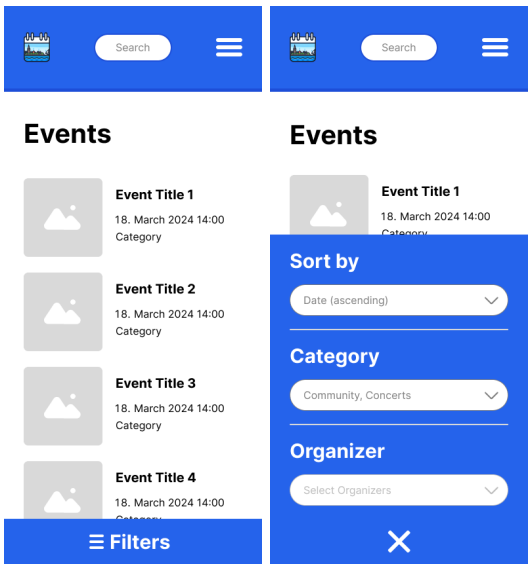


Figure 5.23: Mobile mock-ups: Event overview page

The event creation page is shown in figure 5.24. Same as on the account page, the save button is fixed to the bottom of the screen, and it is only visible when the user has made any changes.

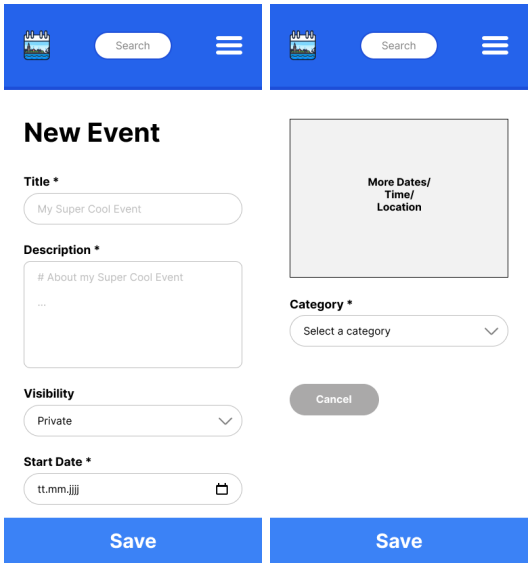


Figure 5.24: Mobile mock-ups: Event creation page

The event edit page is shown in figure 5.25.

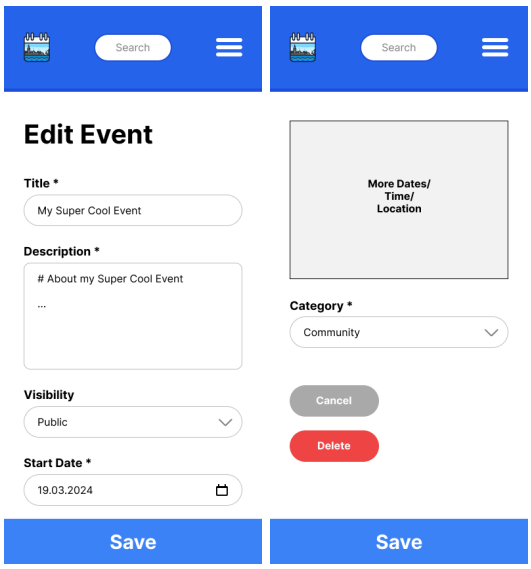


Figure 5.25: Mobile mock-ups: Event edit page

The event single page is shown in figure 5.26.

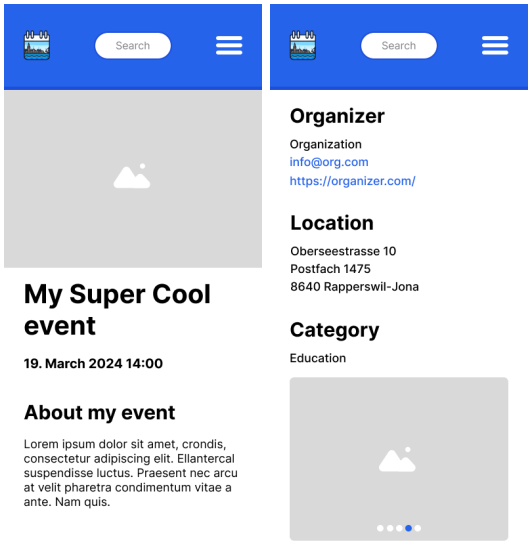


Figure 5.26: Mobile mock-ups: Event single page

Organizers

The organizer overview page is shown in figure 5.27. Same as on the event overview page, the organizer filters are also packed into a new menu, for consistency.

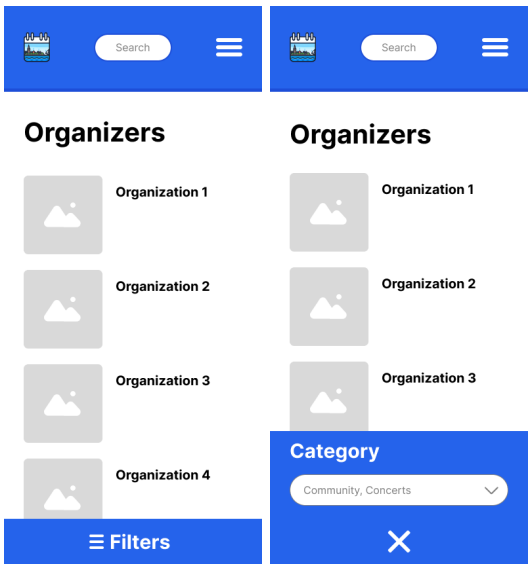


Figure 5.27: Mobile mock-ups: Organizer overview page

The organizer single page is shown in figure 5.28.

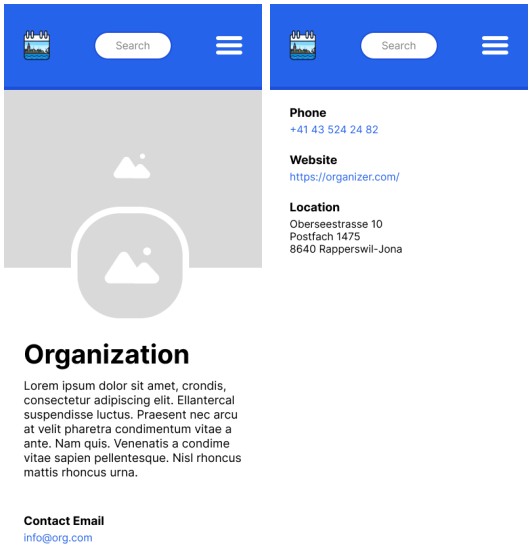


Figure 5.28: Mobile mock-ups: Organizer single page

Search

The responsive search bar requires a slight design change, as shown in figure 5.29. Once the search input is focused, the search bar expands to the full width of the screen. The search popover is then laid over the rest of the content below the header, containing minimal search results. To close the popover, the user can click on the 'X' button, also known as the close button. The search page remains the same as on the desktop version, and also incorporates the new menu for the filters.

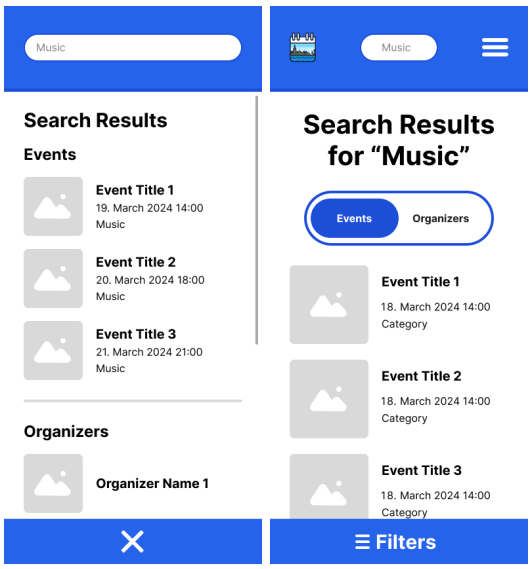


Figure 5.29: Mobile mock-ups: Search bar and search page

Admin Panel

The admin panel is shown in figure 5.30. It includes the overview and the create admin page. The overview page displays the tabs at the bottom of the screen, similar to how a tab bar is displayed in a mobile application.

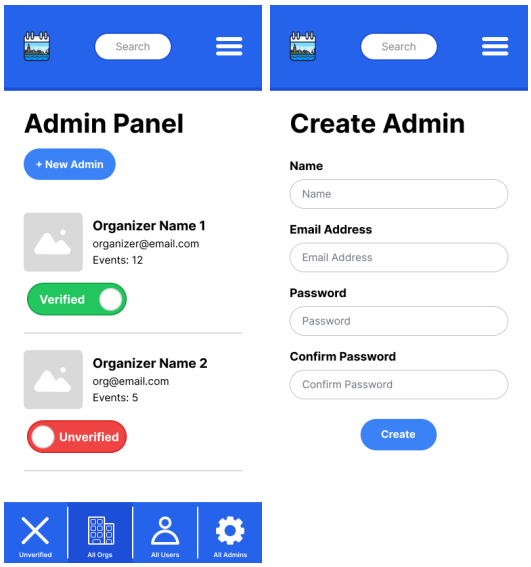


Figure 5.30: Mobile mock-ups: Admin panel overview and create admin page

5.7 Technologies

The technologies used in the project have mostly been defined in the ‘Studienarbeit’. Additional technologies are needed to be defined for the new features and improvements that are planned for this iteration. A requirement for the new technologies is that they should be compatible with the existing technologies and should not require a complete overhaul of the existing system. Furthermore,

technologies shall be easily maintainable, have a large community and be JavaScript based, as agreed upon in the 'Studienarbeit'. A list of existing technologies is available in the chapter 2 under the section 2.3. All the existing technologies will be used in this project iteration as well. The following sections will describe the new technologies that are planned to be used in this project iteration.

DigitalOcean Spaces

DigitalOcean Spaces is an object storage service that makes it easy and cost-effective to store and serve large amounts of data. It is compatible with the existing DigitalOcean infrastructure and can be used to store user-uploaded images for this project, and various other files in the future. The main advantage of using DigitalOcean Spaces is that it is cheaper than storing files on the main server, and it is also more scalable. The primary objective of using an object store is to have a dedicated storage solution for user-uploaded files, such as organization logos, event images, and other files that are not part of the core application. Another benefit of using DigitalOcean Spaces is that it offers a Content Delivery Network (CDN), which can be used to serve the files to the users faster, at no additional cost.

Chapter 6

Quality Measures

6.1 Quality Assurance

Definition of Done

Code review: Every code change must be reviewed by the other team member to ensure quality and consistency.
Automated testing: All automated tests such as linting and building the project must pass before merging.
Manual testing: The defined tests in chapter 7 have passed successfully.
Test documentation: The test protocols are documented and up-to-date.
Documentation: The documentation needs to be up-to-date, and every change needs to be reviewed by the other team member.

Continuous Integration Pipeline

The CI/CD pipeline is used to automate the development process and ensure that the software is tested consistently. The pipeline is integrated into the GitLab repository and is triggered by every push to the repository. All automated tests are executed within the pipeline. This includes linter checks, code quality check and building the project. The pipeline only issues a warning when the linter fails but returns an error when the build fails. Therefore, only buildable code can be merged into the 'develop' branch. However, an exception is made for merges into the 'main' branch, where lint tests must pass, to ensure the quality of the code in the 'main' branch.

Code Metrics

The code quality check is made with Code Quality, which is integrated into the CI/CD pipeline. It creates a report with the code quality and complexity of the source code. The following metrics are measured:

- **Complexity:** The Cognitive Complexity metric is used to measure the complexity of the code. If the complexity is high, the code is difficult to understand and maintain. A number of conditions and nesting levels are used to calculate the complexity.
- **Bug Risk:** The bug risk is calculated with the maintainability and keywords that indicate a high bug risk.

- **TODOs:** Leftover TODOs are highlighted to keep the code clean.

Quality Measurement Tools

The following table 6.1 shows technologies used to ensure code quality.

Technology	Objective	Statement
Code Quality	Code analysis	Code Climate, a tool for assessing code quality, is seamlessly integrated into GitLab's CI/CD pipeline. It analyses the quality and complexity of the source code directly.
ESLint	Project code quality	The linter is employed to analyse the code syntax of both the frontend and backend code. It's integrated into the CI/CD pipeline to uphold code quality standards.

Table 6.1: Technologies for quality measurement

Unit Tests

The unit tests are a proper method for testing the functionality of the code and ensuring that it works as expected. In the initial state of the project, no unit tests were implemented due to time constraints, as was the case in the previous project, the 'Studienarbeit'. The technologies mentioned in Table 6.2 are implemented and ready to use. However, since the focus of this project iteration is on implementing new features, unit tests are still not implemented. The project itself is still in its early phases.
Table 6.2 shows the technologies that were planned to be used for unit testing.

Technology	Objective	Statement
—	Backend unit testing	A backend unit testing framework is not yet implemented. To simplify the process, Jasmine could be used, but other frameworks are also possible.
Jasmine + Karma	Frontend unit testing	Jasmine and Karma are used to test the frontend code. Karma is a test runner that is used to execute the tests. It's also the default testing suite of Angular.

Table 6.2: Additional technologies for quality measurement

Chapter 7

Test Plan

7.1 Introduction

This chapter provides a comprehensive test plan to ensure the quality of the product and test all functionalities based on the requirements defined for this project. Only the implemented optional requirements are included in this test plan as all other optional requirements are not mandatory to successfully complete the project. As this is only a test plan, the actual test results will be documented in the test reports located in the appendix section 13.

7.2 Test Objectives

The following objectives are defined for the test plan:

- Verify that system administrators are able to manage users and organizers.
- Verify that the frontend is responsive and therefore usable on different screen sizes.
- Verify that the system allows organizers to upload images to their events and profile pages.
- Verify that the system is capable of creating, editing and displaying profile pages for organizers.

7.3 Test Strategy

The test strategy uses a combination of automated and manual testing to ensure the quality of the product. Automated testing is used to lint the code and build the project. On the other hand, manual testing is used to verify functionalities of the product. The automated tests are to be performed on every push to a remote branch on GitLab using GitLab CI/CD. The manual tests described in following sections are to be performed by the developers (students) and selected test users.

7.4 Test Environment

The following test environment will be used for manual testing:

- Operating System: Windows 11, Apple macOS Sonoma 14.5
- Browsers: Google Chrome 125, Mozilla Firefox 126, Apple Safari 17.5

- Testing Tool: Postman[1]

Additionally, to test the responsiveness of the system, the following tools and devices will be used:

- Browser developer tools
- iPhone 13 Pro (iOS 17.5, Safari 17.5)
- OnePlus Nord (Android 12, Chrome 125)

7.5 Test Specifications Functional Requirements

The table 7.1 shows the test cases for the functional requirements. The implemented optional requirements that are tested are indicated with an asterisk (*) in the 'No.' column of the table. The test report is located in the appendix section 13.1.

No.	Description	Precondition	Input	Expected Output
1	Administrator views users, organizers and administrators	The administrator is logged in and views the user management section of the admin panel	Choose the preferred user type in the menu	A list of users, organizers and administrators will be displayed respectively
2	Administrator verifies an organizer	The administrator is logged in and views the user management section of the admin panel	Use the 'Verify' toggle	The organizer is verified, the verified badge on their account page is visible, and they can publish events
3	View responsive frontend	Any system actor on the website	Change the screen size to a smaller size	The website will adjust to the new screen size
4	Organizer uploads images to or deletes images of their profile or events	The organizer is logged in and views the event create, event edit or profile editor page	Upload an image using the corresponding field or delete an image	The image will be uploaded and displayed, or deleted
5	Organizer edits profile page	The organizer is logged in and views the profile editor	Edit the form and submit it	The profile page will be edited

Continued on next page

[1]https://www.postman.com/

No.	Description	Precondition	Input	Expected Output
6*	Onboarding for users and organizers	A user or organizer has never done the onboarding process	Sign up for a new account, or log in with an existing account	The user or organizer enters the onboarding process
7*	Email verification for users and organizers	A user or organizer has created an account or requested an email verification	Click on the verification link in the email sent	The user's or organizer's email address will be verified
8*	Administrator manages categories	The administrator is logged in and views the category management section of the admin panel	Add, edit or delete a category	The category will be added, edited or deleted

Table 7.1: Test specifications for the functional requirements

7.6 Test Specifications Non-Functional Requirements

The table 7.2 shows the test cases for the non-functional requirements. The test report is located in the appendix section 13.2.

No.	Description	Precondition	Input	Expected Output
1	Collaborative: Required features are implemented	—	System actor uses the implemented features	All use cases are successfully executed
2	Performance: Backend can handle up to 1000 requests per minute	—	Backend requests	The system keeps running without errors
3	Response Time: All pages are loaded in under 200ms	The user has a stable internet connection	Pages are loaded	All pages are loaded faster than 200ms
4	Responsiveness: Already tested in the functional requirements	—	—	—

Continued on next page

No.	Description	Precondition	Input	Expected Output
5	Browser Compatibility: The product is compatible with the latest versions of Google Chrome, Mozilla Firefox and Apple Safari	The latest stable version of an aforementioned browser is used	Pages are loaded	Pages are loaded without errors
6	Availability: The system is available using the customer-provided domain	The system is deployed on the customer-provided domain	The domain is accessed through the web browser	The website is loaded and displayed
7	User Satisfaction: Users rate the UI	Test users are selected and can provide ratings	Test users navigate the page and provide ratings	3 out of 4 users rate the UI with a minimum score of 8/10
8	Scalability: Database is capable of handling up to 10'000 events and 1'000 users	Database is running	10'000 events and 1'000 users are created	The system keeps running without errors
9	Error Handling: Errors do not cause the system to crash	The system is running	Errors are triggered	The system keeps running without getting into an error state and displays a meaningful error message if necessary
10	Security: All communication within and to the system is encrypted	The system is running in a production environment	The system is accessed	All communication is encrypted
11	Security: Input data is validated and sanitized	—	Invalid or malicious input is entered	The system does not crash, can not be exploited, and may display a meaningful error message
12	Data Privacy: Data protection regulations are met	—	User data is stored	Data is stored according to the data protection regulations
13	Password Security: User passwords are stored securely	The user is registered	User creates or changes password	The password is stored securely and not in plain text

Continued on next page

No.	Description	Precondition	Input	Expected Output
14	User Data Isolation: A logged-in user can only access their own data	The user is logged-in	User tries to access other users' data	The data can not be accessed
15	Modularity: The system is built in a modular way	—	New features are implemented	New features are implemented without the need to change or revamp existing features
16	Testing: API testing	The API testing tool is configured	Tests have been executed	All tests successfully pass
17	Deployment: The system and its whole functionality is deployed	The system is deployed on DigitalOcean, configured with the customer-provided domain	The domain is accessed	The system is available, and all functional requirements are working

Table 7.2: Test specifications for the non-functional requirements

7.7 End-User Tests

Four different test users will be selected to rate the UI and test the use cases of the system. Each test user will receive a questionnaire to guide them through the system and test all defined use cases, as described in the functional requirements' section 7.5. In addition, the test user is asked to provide feedback and rate the UI on a scale from 1 to 10 with 10 being the best. The end-user test reports are located in the appendix section 13.3.

7.8 Test Schedule

The following test schedule is planned for the test plan:

7.8.1 CI/CD Tests

All automated tests using GitLab CI/CD will be executed on every commit or push to a branch. The CI/CD pipelines have to pass before a merge request can be merged. Some jobs within the pipeline do not need to pass successfully, as long as the branch the commit or push is made to is not named 'main'.

7.8.2 Integration Tests

The integration tests will be executed once the corresponding feature is implemented and the code is reviewed. All functional and non-functional tests will be executed before 30.05.2024.

7.8.3 End-User Tests

The end-user tests will also take place before 30.05.2024.

Chapter 8

Implementation

This chapter covers the implementation of the project that was done during the construction phase of this project. It includes the improvements that were made to features that existed prior to this project, the new features that were added during the current project iteration, and additional information about code documentation, deployment, security and testing.

8.1 Improvements

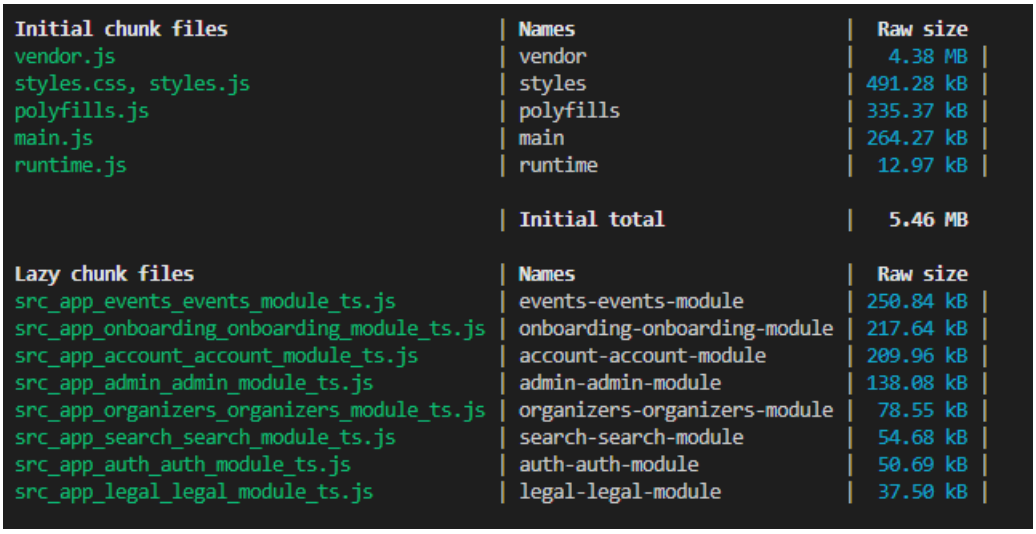
This section describes the most important improvements that were made to the existing features of the project. Those improvements were implemented to either fix bugs, improve the user experience or to make the code more maintainable.

8.1.1 Frontend

This section describes the improvements that were made to the frontend of the project.

Angular Modules

As described in chapter 5.3, the frontend was restructured to use Angular modules. Each feature of the frontend was separated into its own feature module. Additionally, two new modules 'core' and 'shared' were added to the application. Every feature module has its own routing functionality to load the related components. The routing of the root module now imports the feature modules and no longer includes the components directly. Using this structure, each feature module gets lazy loaded when the user navigates to the related page, meaning the module only gets loaded when needed. The figure 8.1 displays the output of the CLI after the frontend gets started in development mode. It shows the lazy loading of the different feature modules.



Initial chunk files	Names	Raw size
vendor.js	vendor	4.38 MB
styles.css, styles.js	styles	491.28 kB
polyfills.js	polyfills	335.37 kB
main.js	main	264.27 kB
runtime.js	runtime	12.97 kB
	Initial total	**5.46 MB**

Lazy chunk files	Names	Raw size
src_app_events_events_module_ts.js	events-events-module	250.84 kB
src_app_onboarding_onboarding_module_ts.js	onboarding-onboarding-module	217.64 kB
src_app_account_account_module_ts.js	account-account-module	209.96 kB
src_app_admin_admin_module_ts.js	admin-admin-module	138.08 kB
src_app_organizers_organizers_module_ts.js	organizers-organizers-module	78.55 kB
src_app_search_search_module_ts.js	search-search-module	54.68 kB
src_app_auth_auth_module_ts.js	auth-auth-module	50.69 kB
src_app_legal_legal_module_ts.js	legal-legal-module	37.50 kB

Figure 8.1: Lazy loading of feature modules

The ‘core’ and ‘shared’ modules are eagerly loaded as they are needed throughout the application. In every feature module the related components and models are added to separate folders. As services are only instantiated once and used in a wide array of different modules, they were added to the ‘core’ module, as it only gets included once as well. Guards and helpers, such as the HTTP interceptor, also take their place inside the ‘core’ module. The ‘shared’ module includes all components that are not considered ‘core’ and are used in various feature modules. An example is the filter component, which is used on the overview pages and the search page. The components, helpers, models and pipes of the ‘shared’ module are available in every feature module simply by importing the ‘shared’ module.

Routing inside a feature module is handled in a separate file, which is imported in the feature module. It defines the routes on the path of the current feature module, and includes the components that are loaded when the user navigates to the path. The routing of the root module is now only used to define the routes of the feature modules, the home page, and to show the 404 page when a route is not found.

Dependencies

A new dependency PrimeNG was added to the project to improve the user interface. PrimeNG is a collection of rich UI components for Angular, which are used to create a more visually appealing and user-friendly interface. The components are used in select locations where default HTML elements would not be sufficient. It unifies the design of the application on different browsers and devices, and provides a better overall user experience. The PrimeNG components used are:

- Carousel: Displays multiple items in a slide show.
- Dropdown: A dropdown menu to select an option from a list.
- MultiSelect: A dropdown menu to select multiple options from a list. Includes a search bar to filter the options and a select all button.
- ToggleButton: A button that can be toggled on and off.

8.1.2 Backend

This section describes the improvements that were made to the backend of the project.

Maintainable Code

The backend was refactored to use overall better typings. The types of the entities were split into multiple interfaces for different purposes to accommodate the presence of different fields in different contexts. For example, the event entity has 'EventCreate', 'EventUpdate' and 'EventPublic' interfaces, each extending from the default 'Event' interface. They are used for creating, updating and returning a public event respectively. The same was done for all other entities, making the code more maintainable and easier to understand. Additionally, the attributes that are returned by the endpoints were unified. The attributes have been split into private and public attributes, where the private attributes are only returned in a context where the user is allowed to see them. This was done to prevent sensitive data from being returned to the user, e.g. the account email address of an organizer. The public attributes are returned in every context, e.g. the name of an event. Overall, this change makes it easier for the frontend to handle the data returned by the backend. A similar change was made for the fields that are saved to the database. The fields were unified to make it easier to understand which attributes will be set when creating an entity. Using the 'fields' option also ensures that only the fields that are allowed to be set are actually set.

UUIDs

The IDs of the entities were initially incremental numbers. This was changed to UUIDs, version 4, to prevent users from guessing the IDs of other entities. The UUIDs are generated when an entity is created and are unique across all entities. Other than changing the data type of the ID and removing the obsolete conversion of strings to integers, no other changes were necessary as the UUIDs are handled the same way as the incremental numbers. The following listing 8.1 shows how the UUID is used as the primary key for the base user entity.

```
1  export const initBaseUser = (sequelize: Sequelize) => {
2      return sequelize.define(name, {
3          id: {
4              type: DataTypes.UUID,
5              defaultValue: DataTypes.UUIDV4,
6              primaryKey: true
7          },
8          // More fields
9      });
10 };
```

Listing 8.1: Using UUID as type for the primary key

Logging

The logging to the console was improved to include the timestamp of the log entry. It should now be easier to track when exactly an event occurred. This is especially useful in the development environment, where console does not show the exact time of the log entry. On DigitalOcean, the logs are already timestamped, so this change is not as important there.

8.1.3 Existing Feature Changes

General Changes

This section describes changes that cannot be assigned to a specific feature. This may be because they change the base of the application, or impact a range of features.

To prevent the application from slowing down or even crashing by loading too many entries from the database, load more buttons were already implemented. However, they were shown even if there were no more entries to load. To prevent this, a new property 'hasMore' was added to the events, organizers, categories and search endpoint results. This indicates whether there are more results to load after the current query, and thus shows or hides the load more button based on its value.

For different entities like organizers, users or events, the frontend already implemented a validation process, but it was distributed across different files, making it very error-prone. Therefore, the whole validation was refactored to use a unified approach for all pages, provided by the validation service. Additionally, the backend now enforces much better validation for all entities, restricting users from entering invalid or unreasonable data that would have otherwise generated an error or lead to unwanted behaviour.

Another change was made to events in the frontend. It was possible to view events that were already in the past. This could be achieved by searching for an event name, which links to an event that has already passed. This has been fixed so that only events that are ongoing or in the future are shown.

Event Handling

The events date and time were initially saved in the user's local time, which causes problems when a user has a different time zone set on their device. To mitigate this issue, the dates and times are now converted to UTC before being sent to the backend and saved in the database. When an event is queried, the date first gets converted to the user's local time zone before being displayed. All conversions are done on the user's device, as it is the only place where the local time zone is known. This change is indicated with a new text below the date and time fields on the create event and edit event pages. Listing 8.2 shows how the date and time are converted to local time before being displayed to the user.

```
public async get(id: UUID): Promise<EventFromBackendPublic> {
    // Get the event from the backend
    const data = this.http.get<EventFromBackendPublic>(
        EVENT_API + id,
        httpOptions
    );

    // Get the event from the observable once it completes
    const event = await lastValueFrom(data);

    // Convert the dates and times of the event to the local time of the user
    setEventDates(event);

    return event;
}
```

Listing 8.2: Converting the date and time to local time

Similarly to converting UTC dates and times to the user's local time, when creating or editing an event, the date and time are converted to UTC before being sent to the backend. In listing 8.3, the newly created event is converted to UTC before being sent to the backend.

```
public async create(event: EventToBackendCreate | EventToBackendCreatePublic |
    FormData, useMultipart = false): Promise<UUID> {
    // Send the event to the backend
    const data = this.http.post<UUID>(
        // Use the correct API endpoint
        EVENT_API,
        // Convert the event to UTC before sending it to the backend
        convertEventToUTC(event),
        // Use multipart if the event may include images
        useMultipart ? {} : httpOptions
    );

    // Get the UUID of the event from the observable once it completes
    return lastValueFrom(data);
}
```

Listing 8.3: Converting the date and time to UTC

This ensures consistency across all users, no matter where they are located.

Create/Edit Event: An event was not able to be edited or created if it is already ongoing. The reason behind this was a check that would not allow an organizer to save an event with a start date in the past. This check was changed to the end date, so that ongoing events can be edited and even created if need be.

The edit event page now includes a button to the single page of the event to improve the navigation for the user. It was not possible to directly navigate to the single page to see the result of the changes before.

In addition, the visibility field now uses PrimeNG's toggle-button component, which is more visually appealing and easier to use than a standard dropdown, requiring only a single click to change the

visibility of the event. The field also gets disabled if the organizer is unverified, as unverified organizers are not allowed to create public events.

Event Single: The event and organizer single pages was initially only viewable if the event was public or the organizer was verified. While creating events or editing their profile, the organizer may want to preview their changes before publishing. Thus, this was slightly changed, also in order to accommodate the button on the edit page and the profile editor, navigating the user to the single page. The event single page is now viewable by the organizer of the event and any administrator at all times even if the event is private or the organizer is unverified. To make it visible that the event is not public, a text was added below the header which indicates that this event is not publicly viewable.

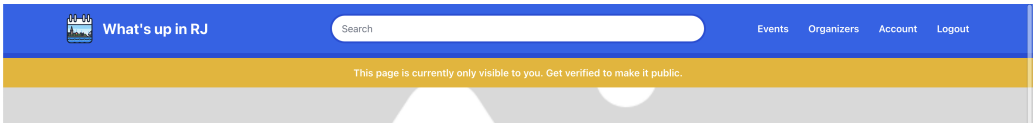


Figure 8.2: Notice on event single page for unverified organizers

The event single page now also includes a section at the bottom of the page with up to six events of the same category, and a button to the event overview with the filter set to the current event category. This was added to display more events of the same category to users and to improve the overall user experience.

Filter

Various minor issues occurred with the filters that have been fixed. The order filter for start dates was not working correctly as it sorted events by the start date but not the start time, leading to an unwanted order of events. This was adjusted, so events are sorted by their start date and time.

Filters that were capable of selecting multiple options were changed to use PrimeNG's multi-select component. This change was made to improve the user experience and to make the filters more visually appealing. Especially on mobile devices, the multi-select component is easier to use than the standard checkboxes. Thanks to the new component, the user can now search for a specific option, select all options at once, or deselect all options at once. The 'order' filter was also changed to use PrimeNG's dropdown component, which unifies the design of the field for all browsers and devices.

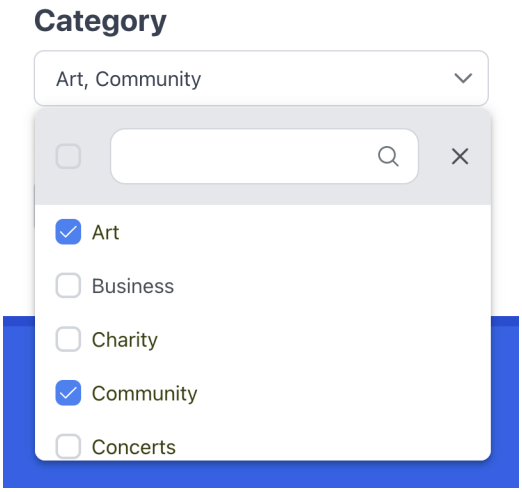


Figure 8.3: Multi-select filter

Furthermore, a new order filter option 'Relevance' was added. This option replaces the standard option for the order of entities when the user navigates to a page with filters such as the event overview or the search page. It shows the events that start in the future first, which can be today or at a later date, and then the ongoing events. The idea of this is to highlight the next upcoming events and not the already ongoing events. If there are many ongoing events, they would all be shown first and the events that start next would lose visibility. Now, ongoing events are only shown first if the order filter option 'Date (Ascending)' is selected.

Home

The problem with the events that are ongoing or in the future was also present on the home page. To prevent this, the upcoming and ongoing events are now separated into their own sections respectively, giving all events the same level of importance.

Additionally, each section now includes a button to view more events. This button leads to the event overview page with the correct filter applied. The filter chosen is based on the section the button is clicked. With this, the initial user experience is improved as the home page is the entry point for almost any new user.

Account

The account page of the organizer initially loaded and displayed all events they created. But if this organizer created a lot of events, the page could take a long time to load and the account page would seem overloaded. A load more button was added to the events, making the process much more manageable if the organizer has many events. To achieve this, two new endpoints were added to the backend, one for the ongoing/future and one for the past events. Only five ongoing/future and five past events are loaded and shown under their respective tabs. If an organizer wants to see more events, they can click the load more button to load five more events inside the current tab. Though, this change does not come without a trade-off. It may take the organizer a longer time to find the event they would like to edit, as they would need to press the load more button repeatedly until the event they are looking for is visible. Considering the amount of events an organizer would have to create to reach this point, this trade-off was deemed acceptable.

As the delete function of an account is a very critical function, it was decided to make it harder to accidentally delete an account. Hence, the user has to confirm their password in a new dialogue before they can delete their account.

In the testing process with test users in the 'Studienarbeit', one feedback was the bad positioning of the save button on the account page. It was not entirely clear if the save button was meant only for the password section or the whole account page. To solve this problem, the save button is repositioned and no longer included in any section, as indicated by new borders around the button.

Authentication

The authentication was only valid for the duration of the session. This means that opening a new tab or closing and reopening the browser logged the user out of the system. A minor change in how detection of a logged-in user works resulted in longer authentication, across multiple sessions. Instead of using the session storage, the check now uses the local storage of the browser, where data can be saved persistently across sessions. Therefore, authentication is now valid for 24 hours or until the user logs out. The token is still saved as a 'HttpOnly' cookie, but some user data is also saved in the local storage to detect a logged-in user.

The initial password requirements were very low and only required a minimum length. To make the accounts more secure, the complexity was increased to require at least one lowercase letter, one uppercase letter, one number, and one special character, besides having a minimum password length of 8 characters.

Newsletter

The newsletter was initially sent out at midnight (12AM UTC). This causes low opening rates as most users are not usually online at this time. To increase the opening rates of the newsletter the time was changed to midday (12PM UTC), which should lead to more interaction between the user and the newsletter email.

In addition, the newsletter was only sent out to users who have selected at least one category as a preference. Now, users who have not selected any categories but are subscribed to the newsletter will also receive the newsletter. The reasoning behind this is that the user may not have found a specific category they are interested in. By default, all categories are included in the newsletter if the user has not selected any categories as preferences.

8.1.4 Other Improvements

Some smaller improvements that were made are listed below:

- Throughout the whole application, events and organizers are displayed in a card view. These cards were centralized in separate components and are used in multiple locations. They also got some minor redesigns and improvements, e.g. the event card now also shows the organizer avatar. When hovering over the avatar, the organizer name is displayed.
- The legal pages are now displaying dummy texts as an example for the user to get an idea of what the page will look like. These texts need to be replaced with actual legal texts before deploying the application to production.

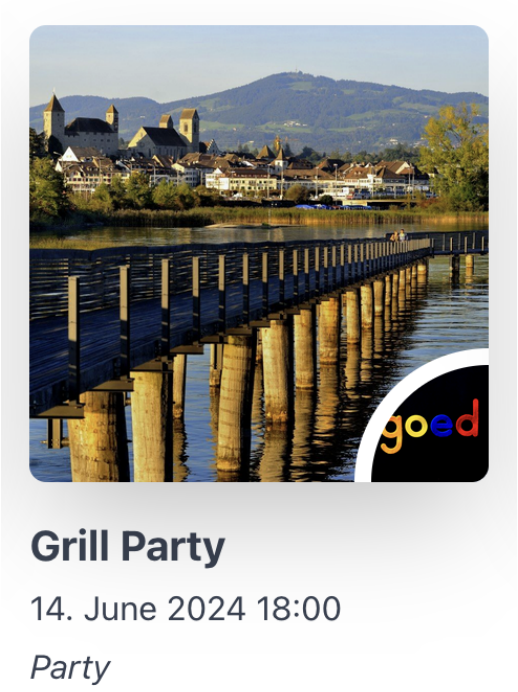


Figure 8.4: Event card

8.2 Design

The base design of the application was already defined by the 'Studienarbeit'. However, some minor changes were made to improve the overall user experience and to make the application more visually appealing. The current implementation of the design may differ from the design that was defined in the section 5.6 during the planning phase of the project. This is due to the fact that some design decisions were made during the construction phase to improve the user experience or to make the application more visually appealing. Regarding the responsive design, some elements were styled differently from the mock-ups to improve the user experience on smaller screens. The responsive design is described in detail in the section 8.4.8.

8.3 Accessibility

The application was designed with accessibility in mind. The main focus was on the user experience and the functionality of the application. However, accessibility is an important aspect of web development and should not be neglected. To provide a better experience for users with disabilities, a few steps were taken to secure a minimum level of accessibility.

8.3.1 Semantic HTML

The application was built with semantic HTML rules in mind where possible to ensure that the content is accessible to screen readers and other assistive technologies. While not all elements are used as they should be, the main content of the application is structured using the correct HTML elements. For example, the body of the application uses a 'main' element to wrap the main content of the page, the header uses a 'header' element to wrap the navigation bar, and the footer uses a 'footer' element to

wrap the footer content. While the footer itself uses ‘h5’ elements to title the different sections, thus skipping the ‘h2’ to ‘h4’ elements, the headings are still used in a hierarchical order, though not all levels are used. Due to design reasons, a smaller heading was chosen to make the footer less dominant on the page.

8.3.2 Colour Contrast

The colour contrast of the application was chosen accordingly. The text colour and background colour have a high contrast ratio on most elements, making it easier for users with visual impairments to read the content. However, the contrast ratio was not checked for every element, and some elements may not have a high enough contrast ratio. To improve the accessibility of the application, the contrast ratio should be checked for all elements and adjusted if necessary.

8.3.3 Keyboard Navigation

The application supports navigation using the keyboard. All interactive elements are accessible using the tab key, and the active element is highlighted to make it easier for the user to see where they are. To activate a focusable element, the user can press the enter key.

One exception to the design is the button element, which does not show a focus ring when focused. This is due to the design of the application, where the focus ring would show even if the user is not using the keyboard to navigate, making it less visually appealing. However, the button still works and shows a slight effect when focused, indicating that it is active.

8.3.4 Screen Reader

Screen readers are partially supported by the application. Labels are used for all form fields, however they are not always visible to the normal user, as placeholders are used to display the labels instead. Screen readers are able to read the labels and help the user to understand the purpose of the form field. Images are also provided with an ‘alt’ attribute to describe the image to the user, although the text is not always helpful to the user, as it may only show that it is an image of the event or organizer. To add more context to the images, the ‘alt’ attribute could be defined by the user when uploading an image, or a detection algorithm could be used to automatically generate a description of the image.

8.3.5 Form Validation

Form validation is done on the client side to prevent the user from entering invalid data. The user is informed about the error and how to fix it. Improving the form validation to also include attributes like ‘aria-invalid’ and ‘aria-errormessage’ would help screen readers to understand the error and provide a better user experience.

8.3.6 Testing

The application was not tested with screen readers or other assistive technologies.

8.4 Features

This section describes the new features that were added to the project during the construction phase. All features are described in desktop view, except for the responsive design, which is described in the section 8.4.8.

8.4.1 Admin Panel

Administering the application was initially only possible by directly accessing the database. To make it easier to manage the application, an admin panel was added. The admin panel is only accessible by administrators and includes important features to manage the application. Functionalities are divided into their own sub-pages, namely the user management, described in section 8.4.1 and the category management, described in section 8.4.1. Access to the admin panel is possible by using the admin bar described in section 8.4.1.

Admin Bar

The admin bar resides in the header, sticking to the bottom of the navigation bar. It is only visible to logged-in administrators and includes two links, one to the user management and one to the category management. As the admin bar is fixed together with the navigation bar, it is always visible to the administrator, no matter where they are on the page. This also introduces a new issue, as the admin bar may overlap content on the page. In order to prevent this, the content of the page is additionally pushed down by the height of the admin bar, if it is visible.

User Management

Managing users, especially organizers, is a crucial part of the application. The admin panel sub-page for user management consists of two parts, the creation of new administrators and the management of existing users.

Create Admin: When clicking on the 'Create Admin' button, a form is shown on a new page to create a new administrator. The form is structurally the same as the registration form for users, requiring a name, email address, and password. The same checks and verifications apply to all fields, ensuring that the name is of sufficient length, the email address is unique and valid, and the password is secure. After submitting the form, a new administrator is created and can log in with the provided email address and password.

Tabs: At the top of the page, four tabs are shown, one for each user type ('organizer', 'user', 'admin'), and one extra tab for unverified organizers. Each tab lists the users of the corresponding type, including their name, email address, and email verification status. Organizers also include the amount of events they have created, and a toggle to verify or unverify the organizer. An organizer can be verified by clicking the toggle, which prompts the user to confirm the action. The organizer can be verified even if the email address is not verified yet, however, the administrator is warned in the confirmation dialogue. When an organizer is verified, an email is sent to the organizer to inform them about their verification. The existing SendGrid service with a new dynamic template is used to send the email, which includes the name of the organizer and a short text about the verification. The email is purely informative and does not require any action from the organizer. The verification status of the organizer is updated in the database, and the organizer is now visible to the public and is able to publish events. Unverifying an organizer works the same way, but in reverse, and no email is sent to the organizer.

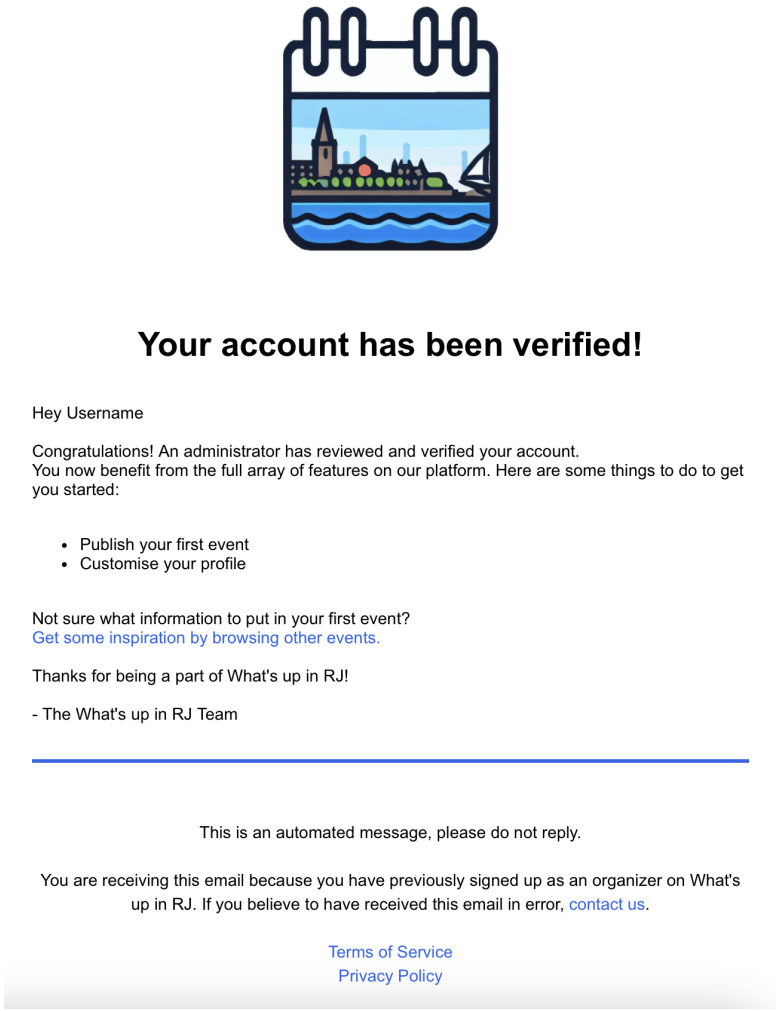


Figure 8.5: Organizer account verification email

A load more button is included at the bottom of the lists if the total amount of users for that list exceeds 50. The 'Unverified Organizers' tab lists all organizers that are not yet verified, making it easier for administrators to go through pending verifications. Every organizer, verified or not, is listed in the 'All Organizers' tab. This opens up the possibility to also unverify an organizer after they have been verified, if necessary. The third tab 'All Users' lists all registered users. With no additional actions possible, this tab only serves as information. The last tab 'All Admins' lists all administrators with a delete button. Administrators can delete other administrators, but not themselves. Thus, the delete button for the currently logged-in administrator is not visible. In addition, the 'Delete Account' section on the account page is hidden for administrators. This prevents the last administrator from deleting themselves and locking everyone out of the admin panel. The default administrator function still exists, and is only executed when the application is started for the very first time with an empty database. It creates a new administrator with a predefined email address and password, which can be manually set through environment variables, enabling the first administrator to log in and create more administrators.

Category Management

With the addition of the category management, administrators can now create, edit, and delete categories on their own. Categories are used to group events together, making it easier for users to find events they are interested in, and receive a newsletter based on their selected categories as preferences. The category management sub-page includes a list of all categories, and buttons to create, edit, and delete categories.

Overview: The sub-page includes a list of all categories, displaying their name and buttons to edit or delete the category. The list is ordered alphabetically by the category name, and automatically updates when a category is created, edited, or deleted.

Create Category: Creating a new category is done by clicking the 'Create Category' button, which opens a dialogue to enter the name of the new category. The name is checked for its length and uniqueness, and the category is then created and added to the list of categories.

Edit Category: The edit dialogue is displayed inline when clicking the edit button, allowing the administrator to change the name of the category. The buttons are then replaced with a save and a cancel button for the duration of the edit. The name is also checked for validity after saving, as described in the 'Create Category' paragraph above.

Delete Category: The delete button opens a confirmation dialogue to prevent accidental deletion of a category. A category cannot be deleted if it is assigned to any event, and the administrator is informed about this when trying to delete a category that is still in use. The reasoning behind this is to prevent the deletion of a category that is still in use, as this would lead to public events without a category, which is not allowed. A possible improvement would be to allow the deletion of the category, but to assign a default category to the events that are using the category to be deleted. Another approach would be to also allow the deletion of the category, but to inform the administrator about the events that are using the category to be deleted, so they can decide if they want to delete the category or not.

8.4.2 Profile Editor

The profile editor is a new feature that allows organizers to customize their profile. Accessible from the account page, the profile editor includes various fields to extend the organizer's profile. Each field is optional, and the organizer can choose which information they want to share with the public. Additionally, each field is validated to ensure that the data entered conforms to the expected format. The following fields are available in the profile editor:

- Description: A description of the organizer with Markdown support, allowing the organizer to add links or lists, emphasize text, etc.
- Avatar Image: An image to represent the organizer. The image is shown with every public event the organizer creates.
- Banner Image: A banner image to represent the organizer. The image is only shown on the organizer's profile page.
- Contact Email: An email address to contact the organizer. Is displayed on the organizer's profile page and the event single page.

- Phone: A phone number to contact the organizer. Is displayed on the organizer's profile page and the event single page.
- Website: A website link to the organizer's website. Is displayed on the organizer's profile page and the event single page.
- Location: The location of the organizer. Can be entered by filling out the fields 'Street', 'Street 2', 'Zip' and 'City'.

In addition, the name of the organizer is displayed at the top of the profile editor. The name is read-only and can only be changed on the account page, which is noted in a short text below the name. The layout uses a two-column design on desktop screen sizes to group related fields together, making it easier for the organizer to find the field they want to edit. The name, description and image fields are displayed on the left side of the profile editor. They have more space horizontally, as the description can be quite long and the images are displayed side by side to avoid a long vertical scroll. On the right side, the contact information and location fields are shown. An organizer can easily view the changes they made by clicking the 'View Profile' button, which opens the organizer's profile page. If an image has been uploaded, it is shown in the profile editor as a preview. To delete the image, the organizer can click the 'X' button in the top right corner of the image. Optionally, a new image can be uploaded to replace the old one. The image upload functionality is explained in detail in section 8.4.4.

8.4.3 Organizer Profile Page

An organizer now has their own profile page, displaying the information they have entered on the profile editor. The profile page is accessible to all users, if the organizer is verified. If the organizer is not verified, the profile page is only accessible to administrators and the organizer themselves, implementing the same logic as the event single page. The profile page includes the name of the organizer, all fields of the profile editor, and a list of up to six public events the organizer has created, with a button to view more events by the organizer. The layout of the profile page is similar to the profile editor, with the description on the left side, and the contact information and location on the right side in the sidebar. The banner image is displayed at the top of the page, followed by the avatar image and the name of the organizer. The sidebar on the right is sticky, meaning it stays in place when the user scrolls down the page. This allows the user to always see the contact information and location of the organizer, even when scrolling through the description. All the contact information items are clickable, leading the user to the respective contact method. For example, clicking the email address opens the default email client, clicking the phone number opens the phone app, and clicking the website opens the website in a new tab.

8.4.4 Image Upload

Organizers can now upload images to represent themselves and their events. The image upload functionality is available in the profile editor and the create/edit event pages. Special cases for both the profile editor and the event pages are described in their respective sections.

User Interface

The file upload field is designed as a drop zone, allowing the organizer to drag and drop an image file onto the field. Alternatively, the organizer can click the field or the 'Browse' button inside the zone to open a file picker dialogue. The file name and the file size are displayed inside the field after a file has been selected. A selected file can be removed by clicking the 'Clear' button, which is only

visible when a file is selected. The image upload field is designed to only accept image files, and the file size is limited to 5 MB. If the file size exceeds the limit, an error message is displayed below the field. An image preview is available once the form has been saved and the image has been uploaded. Each image can be deleted by clicking the 'X' button in the top right corner of the image. The exact behaviour of the image upload field for different pages is described in the sections 8.4.4 and 8.4.4.

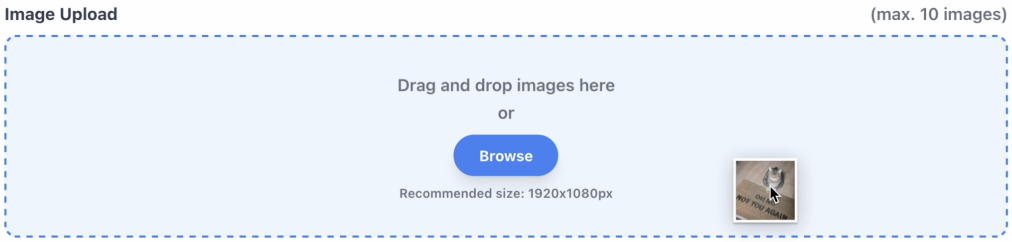


Figure 8.6: An image being dropped onto the image upload field

Frontend

Whenever a form includes an image upload field, it is submitted as a 'multipart/form-data' form. By default, forms are set to use the 'application/json' content type, which does not support file uploads. To accommodate file uploads, the content type is changed to 'multipart/form-data' by telling the Angular service responsible for the HTTP requests to derive the content type from the form data. In order to correctly derive the content type, the form data is created using the 'FormData' class, which is a built-in JavaScript class that allows the creation of 'multipart/form-data' forms. The image is then uploaded to the backend using the 'HttpClient' service, which sends the form data to the backend.

Backend

The image upload functionality is implemented in the backend using the 'Multer' middleware for Express. 'Multer' is a middleware for handling 'multipart/form-data' forms, which is used to upload files. The middleware is added to the routes that handle an image upload, where it checks for specific field names that are used to upload images. If an image is uploaded, 'Multer' checks the file for file size and MIME type, and saves the image temporarily to the 'uploads' directory if it passes the checks. The name of the file is adjusted to prevent collisions with other files and for privacy reasons. It is renamed to a UUID with the original file extension. Uploaded image files stored in the 'uploads' directory are deleted after the image has been uploaded to the object storage or an error occurred during the whole process. If the image is successfully uploaded to the object storage, the file is deleted at the end of the request. If an error occurs, the file objects are passed to the global error handler, where the files are deleted. All images are then available on the 'Request' object inside 'req.files', which is an array of files.

Due to the different content type of the form, the request body itself, which includes all other form fields, is not available in the same way as for a 'application/json' form. Therefore, the request body must be converted back to its original form, which is done by parsing the form data one field at a time. This process is necessary to not break the existing form handling in the backend. It converts field values to the correct data type, e.g. a numeric string to a number, or the string 'true' to the boolean 'true'. Once this is done, the images are then extracted from the 'req.files' array.

Each image is being prepared for upload by generating the key used to access the image on the object storage. This is done by adding the previously generated UUID file name to the path of the

image in the object storage, e.g. 'avatars/organizers/organizerId/fileName' for an avatar image, or 'events/eventId/fileName' for an event image. For development environments, the key is prefixed with 'test/' to prevent collisions with production images and separate the images from the images in the production environment. As a last preparation step, 'sharp' is used to rotate the image based on its orientation set in EXIF, and to remove all EXIF metadata from the image. This is done to prevent the image from being displayed incorrectly due to the orientation, and to remove any sensitive information that may be stored in the metadata. Sensitive information could be the location where the image was taken, the device used to take the image, or the date and time the image was taken, which could be used to track the organizer. Because the image is uploaded to a public object storage, it is important to remove any sensitive information from the image.

An issue affecting '.jpeg' images was discovered during the late stages of the project. The issue revolves around the 'sharp' package, which processes and returns the image as a buffer. A memory leak occurs when a '.jpeg' image is processed by 'sharp', which causes the server to crash due to memory exhaustion. DigitalOcean automatically detects the component as unhealthy and restarts it, which leads to a short downtime of the application. The short downtime can be explained by the database connection being unexpectedly dropped, resulting in locked tables. The database requests need to time out before the tables are unlocked, and the application can continue to run. The issue is not present when processing other image types, like '.png'. It only affects the production environment, more specifically Linux systems, as the development architecture is different and does not show the same behaviour. The reason behind the memory leak can be traced back to the 'sharp' package, which is used to process the image. The issue lies in the package's dependencies, and not in the implementation of the image upload functionality. Under the hood, 'sharp' uses the 'libvips' library, which is a low-level image processing library. A bug was discovered in the 'libvips' library, which causes the memory leak. The bug has been patched since, but the issue still persists in the application. The backend server runs 'Ubuntu 22.04.3 LTS' and uses the 'glibc' memory allocator. The memory leak is caused by the 'glibc' memory allocator, which does not release the memory back to the system after processing the image. The memory is still allocated to the process, but is not used, which causes the memory leak. Using a different memory allocator like 'jemalloc', as advised by the 'sharp' package maintainers, would normally prevent the memory leak, as it releases the memory back to the system after processing the image. However, switching to 'jemalloc' did not prevent the memory leak in this case due to an unknown reason. More steps were tested to prevent the memory leak, but did not resolve the issue:

- Creating a new child process with worker threads to process the image, and killing the process after the image has been processed.
- Writing the image to disk after processing, and reading it back to memory before uploading it to the object storage.

Using 'Alpine Linux' as the base image for the backend component would be another possible solution, as it uses the 'musl' memory allocator, which does not show the same behaviour as the 'glibc' memory allocator. However, due to the nature of App Platform, the base image cannot be easily changed. The backend would need to be containerized as a Docker container, which is not feasible at this point.

To prevent the server from crashing, the image processing is skipped for '.jpeg' images, and the image is uploaded as is. In addition, the 'jemalloc' memory allocator is used in production to possibly prevent higher memory usage over time. The issue is not resolved, but the server no longer crashes due to the memory leak. This temporary fix is not ideal, as the images are not processed and may contain sensitive information in the metadata. However, this is the best solution for now, as the server needs

to be stable, and the application needs to be fully available to the users. The following listing 8.4 shows the code snippet where the image processing is skipped for '.jpeg' images.

```
// Remove metadata and rotate image according to EXIF orientation
let buffer;

// Processing JPEGs is currently not possible due to a memory allocation issue
if (file.mimetype !== 'image/jpeg') {
    buffer = await sharp(file.path)
        .rotate()
        .toBuffer();
} else {
    buffer = await fs.readFile(file.path);
}
```

Listing 8.4: Removing EXIF Metadata from Non-JPEG Images

Finally, the image is uploaded to the object storage, and the key used to access the image on the object storage is saved to the database. The external dependency '@aws-sdk/client-s3' is used to interact with the object storage, which is DigitalOcean Spaces in this case. DigitalOcean Spaces officially supports the AWS SDK, which is why this package is used to interact with the object storage. To connect to the object storage, the package requires the Access Key ID and the Secret Access Key, which are stored in environment variables. The image is uploaded to the object storage using the 'putObject' method, which takes the bucket name, the key, the body of the file (image buffer), the content type and the Access Control List (ACL) as parameters. The ACL is set to 'public-read', which allows the image to be accessed by anyone who has the link to the image. Listing 8.5 shows the image upload functionality in the backend.

```
// Import and configuration of the S3 client

// Upload an image to DigitalOcean Spaces
export const uploadS3Object = async (key: string, contentType: string, body:
    Buffer): Promise<true> => {
    const response = await s3Client.putObject({
        // The name of the bucket
        Bucket: process.env.SPACES_BUCKET,
        // The key used to access the image on the object storage
        Key: key,
        // The image content
        Body: body,
        // The content type of the image
        ContentType: contentType,
        // The Access Control List, allowing public read access to the image
        ACL: 'public-read'
    });

    // ...
};
```

Listing 8.5: Configuring the S3 Client and Uploading an Image

Should an error occur during the request to the backend, the image is not uploaded to the object storage, and the request is aborted. An error may occur because of the form body, which holds all

other form fields, a network error while uploading an image, or an error in the backend itself. If the image has already been uploaded to the object storage when an error occurs, the image is deleted from the object storage to prevent orphaned images from being stored. It is important to clean up the object storage to prevent unnecessary storage costs and to not save images without the user's consent, as the image key is not saved to the database until the request is successful in its entirety. The error is then returned to the frontend, where it is displayed to the user.

Image Deletion: The deletion of an event image is done by sending a request to a separate endpoint in the backend. Other images, like the avatar and banner images of organizers, are deleted via normal form submission and do not require a separate endpoint. The endpoint is protected by an authentication middleware, ensuring that only the organizer of the event can delete an image. Only a single image can be deleted at a time, as the endpoint only accepts the ID of the event image to be deleted. The image is deleted from the object storage by using the 'deleteObject' method of the AWS SDK, which takes the bucket name and the key of the image as parameters. The key is retrieved from the database, where it is saved when the image is uploaded. After the image is deleted from the object storage, the key is removed from the database, and the image is no longer accessible. Listing 8.6 shows the deletion of an image from the object storage.

```
1  // routes/events.ts
2  // Register the endpoint to delete an event image
3  router.delete('/image/:id', [midVerifyToken, midIsOrganizer], (req: Request,
       res: Response, next: NextFunction) => {
4      // Call the controller, which will run the deletion logic
5      void EventController.deleteImage(req, res, next);
6  });
7
8  // helpers/s3.ts
9  // Delete an image from DigitalOcean Spaces
10 export const deleteS3Object = async (key: string): Promise<true> => {
11     const response = await s3Client.deleteObject({
12         // The name of the bucket
13         Bucket: process.env.SPACES_BUCKET,
14         // The key used to access the image on the object storage
15         Key: key
16     });
17
18     // ...
19 };
```

Listing 8.6: Deleting an Image from DigitalOcean Spaces

DigitalOcean Spaces and CDN

The images are stored in DigitalOcean Spaces, which is an object storage service provided by DigitalOcean. DigitalOcean Spaces is compatible with the AWS SDK, making it easy to interact with the object storage programmatically. The images are stored in a bucket, which is a container for objects, and are accessible via a unique URL. By combining the base URL of the bucket with the key of the image, the image can be accessed from the object storage. The images are served to the user via a Content Delivery Network (CDN), which is also provided by DigitalOcean Spaces. The CDN caches the images and serves them to the user from the edge location closest to the user, reducing the latency

and improving the loading time of the images. All images are configured to be cached for one week, the maximum value allowed by DigitalOcean, before they are fetched from the object storage again. This is done to reduce the load on the object storage and to improve the loading time of the images. Because each image is unique and has a unique URL, even if updated, the cache setting can be set to a high value without affecting the functionality of the application. The following figure 8.7 shows the root level of the bucket in DigitalOcean Spaces, where the images are stored.

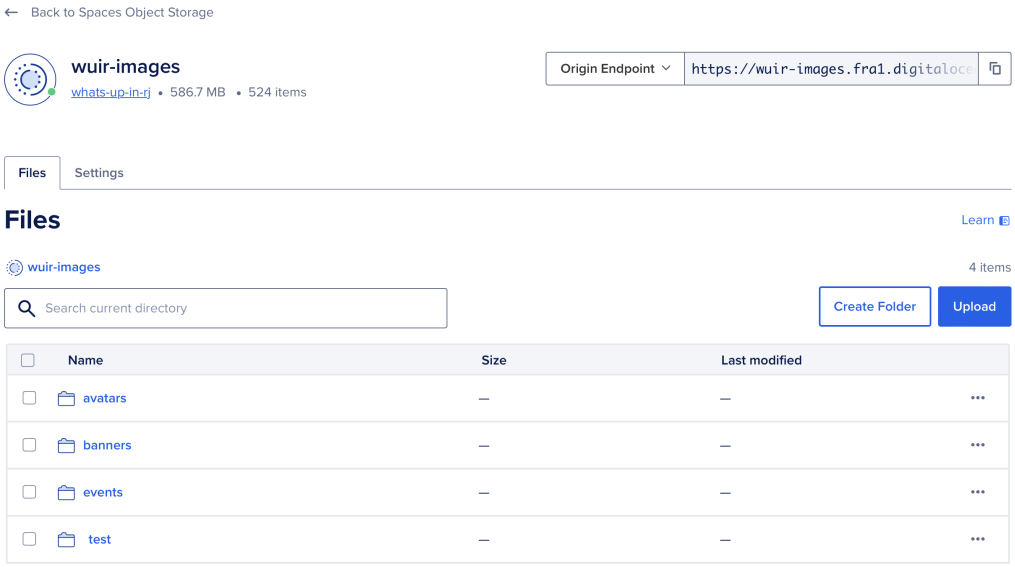


Figure 8.7: DigitalOcean Spaces bucket file list on the root level

Event Images

The number of images that can be uploaded for an event is limited to ten. This is to prevent the event page from becoming too cluttered with images. The upload field disappears as soon as ten images are uploaded. If more than ten images are uploaded, the user is informed that only ten images can be uploaded, and only the first ten images are saved. Any uploaded images are displayed above the drop zone. The images are shown in a grid layout, with a maximum of five images per row. As mentioned in the user interface section, the images can be deleted by clicking the 'X' button in the top right corner of the image. The deletion of an image is confirmed by a dialogue to prevent accidental deletion, as it takes place immediately after clicking the button and is irreversible. This only applies to the event images, due to technical limitations. Because the backend endpoint only supports the upload of images, the deletion of an image is not possible. The additional complexity of an array of images would have made the implementation on the same endpoint more difficult. Hence, an extra endpoint in the backend has been added to delete a single event image. Furthermore, a primary image can be set by clicking on an image. The primary image will be used as the main image for the event, and will be displayed in the event card, the event single page and the account page as the first image. A blue border is added to the image to indicate that it is the primary image. It can be removed by clicking on the blue 'X' button in the top right corner of the image. If no primary image is selected, the first image gets used as the main image.

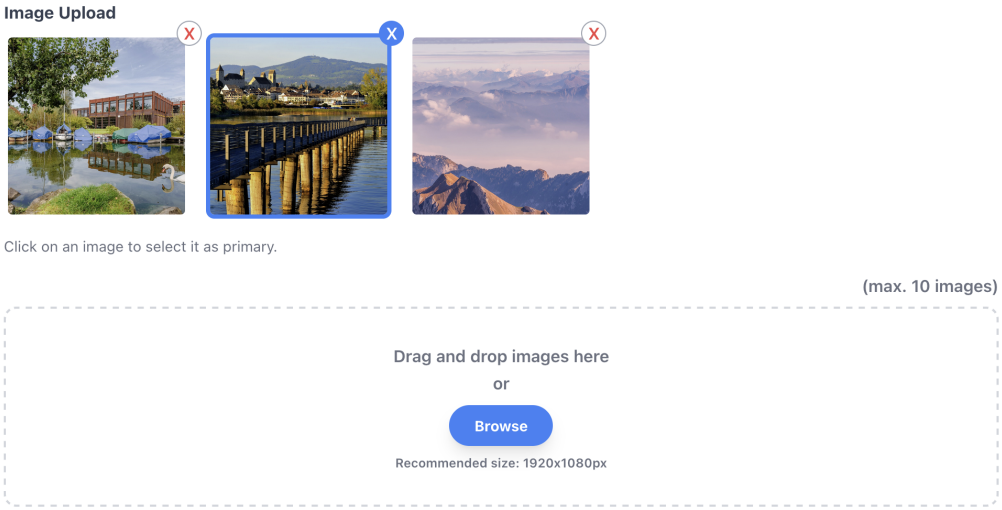


Figure 8.8: Primary event image selection

Organizer Profile Images

Only one image can be uploaded per field in the profile editor. If e.g. an avatar image exists, the avatar image preview replaces the drop zone of the avatar image field. The same applies to the banner image. An image can be deleted by clicking the 'X' button in the top right corner of the image. In contrast to the event images, the deletion of an image is not immediate, and will only take place after the form is saved. Once the 'X' button is clicked, the drop zone is shown again, letting the user upload a new image, or keep the field empty. The avatar image is visible across the application, for example in the event card and the event single page. The banner image is only visible on the organizer's profile page and is used for decoration purposes.

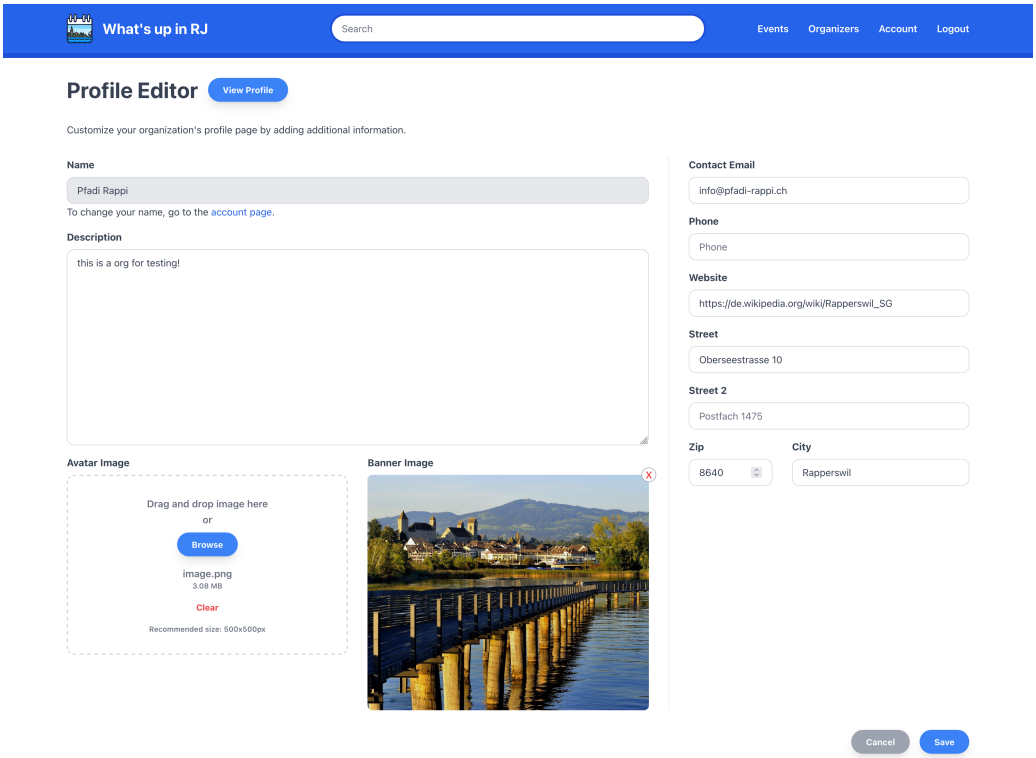


Figure 8.9: Profile editor with selected avatar and uploaded banner image

Handle NSFW Images

Currently, there is no check for Not Safe For Work (NSFW) images. This could be implemented by using a third-party service like the 'NSFWJS' library, which uses a neural network to detect NSFW images. The image is then checked before it is uploaded to the object storage. If the image is detected as NSFW, the upload is aborted, and the user is informed about the issue. This would prevent NSFW images from being uploaded to the application and being displayed to other users. Optionally, the image could be uploaded, but marked as NSFW, and be blurred or hidden by default. The user could then decide if they want to view the image or not. Implementing it in this manner ensures that the user can still upload the image while not displaying the image to other users by default, providing a more user-friendly approach. This would be especially useful for false positives, where the image is not NSFW, but is detected as such.

8.4.5 Placeholders

If no image is uploaded by the user, a placeholder image is used instead. The placeholder image is a generic image showing mountains and a sun, which is used to represent the missing image. It is used for the avatar image, the banner image, and the primary event image. Using a placeholder image ensures that the user interface does not break when no image is uploaded, and that the user can still see where the image would be displayed.

8.4.6 Email Verification

To prevent spam and fake accounts, email verification was added to the registration process. The verification applies to all users, except for administrators. After registering, the user receives an email

with a verification link. In the background, an email verification token as well as an expiration date are generated and saved to the database. The token is a random 64 hexadecimal string that is unique to the user and is used to verify the email address. The expiration date is set to 24 hours after the token was generated, after which the token is no longer valid. A new verification email can always be requested on the account page if the token has expired or if the email was not received. The verification link is a unique URL that includes the token. The email is sent using the SendGrid service, which has previously been used for the newsletter. A new dynamic template is used to send the email, which includes the name of the user, a short text about the purpose of the email, and the verification link. When the user clicks the link, the token is extracted from the URL and checked against the database. If the token is valid and has not expired, the email verification status is set to true. The user is then redirected to the home page and a success message is displayed. If the token is invalid or has expired, an error message is displayed to the user and the email verification status is not changed. The email verification status limits the delivery of the newsletter to registered users with a verified email address. Additionally, administrators can see if the account email address is still unverified before verifying an organizer.

Verify your email address

Hey Username!

Please click the link below to verify your email address. This link is valid for 24 hours.
You can always request a new email verify link on your account page.

https://whatsup.rapperswil-jona.city/auth/verify-email/3acb4017492be824fda610cef842492c3acb4017492be824fda610cef842492c

- The What's up in RJ Team

This is an automated message, please do not reply.

You are receiving this email because you have previously signed up on What's up in RJ. If you believe to have received this email in error, contact us.

Terms of Service
Privacy Policy

Figure 8.10: Email verification email

Account Page

Should the user not receive the verification email, or is in need of a new one, they can request a new verification email below the email address field on the account page. It also shows the current verification status of the email address, informing the user if the email address is not verified yet. The

newsletter field now also includes a notice that the newsletter is only sent to verified email addresses.

8.4.7 Onboarding Process

An initial onboarding process was added to get new users started with the application. The onboarding process is divided into two parts, one for users and one for organizers. Administrators do not have an onboarding process, as they are already familiar with the application. It is displayed after registration and can be skipped at any time by the user. If the user skips the onboarding process, they will be prompted to complete it the next time they log in. Alternatively, they can access it by explicitly navigating to the onboarding page. If the user has already completed the onboarding process, they will not see it again. Completion of the onboarding process can be achieved by completing one of the following flows. For users, this is achieved by either selecting 'No, thanks' on the newsletter subscription step, or by clicking on the 'Skip' or 'Next' button on the interest selection step. Organizers have to either click any button on the resource overview step, which is not labelled with 'Skip', or complete the event creation flow on the step-by-step guide after clicking on 'Create Event' on the resource overview step.

User Onboarding

The user onboarding is divided into a welcome message, a newsletter subscription, and an interest selection. In a first step, the user is welcomed to the application and informed about the purpose of the onboarding process. If the account email is not verified, a notice is shown to let the user know that they should verify their email address. On the second step, the user can subscribe to the newsletter by clicking on the appropriately labelled button. Should the user decide to subscribe, the newsletter subscription status is set to true. In the last step, which is only shown if the user decided to subscribe to the newsletter, the user can select their newsletter preferences by choosing from a list of interests, displayed as a dropdown menu. The interests are the categories of the events, which are used to group events together. The user can select multiple interests, and the selection is saved to the database after the 'Next' button is clicked. A final message is displayed to the user, informing them that the onboarding process is complete. After clicking the 'Get Started' button, the user is redirected to the home page.

Organizer Onboarding

The organizer onboarding is divided into a welcome message, a resource overview, and a step-by-step guide to create the first event. In the first step, the organizer is shown a welcome screen, similar to the user onboarding. The account email verification notice is also present if the email is not verified. Additionally, another notice regarding the verification of the organizer themselves is shown, should the organizer not be verified yet.

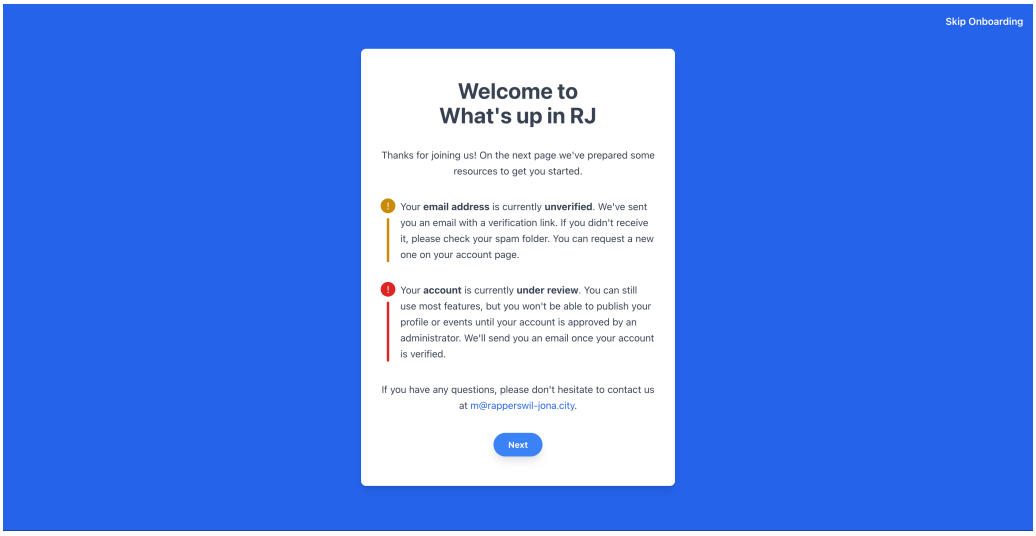


Figure 8.11: Organizer onboarding welcome message

Next, the organizer is presented with an overview, which includes a list of resources to help the organizer get started. The resources are links to the profile editor, the event overview and the organizer overview. A button to create an event is also included in the list, leading the organizer to the step-by-step guide. This guide includes additional steps in the onboarding process, which asks the organizer for information about the event they would like to create. It is meant to guide the organizer through the process of creating an event and to inform them about the different fields they can fill out. The first step is to enter the title of the event, followed by the description, the start and end date, the category, and the location. The title is required, while the other fields are optional and can be skipped. Each step is displayed on its own page, and the organizer can navigate back and forth between the steps. When clicking on the 'Next' button on a step, the data entered is validated, and the organizer is informed about any errors. If the data is valid, the organizer is taken to the next step.

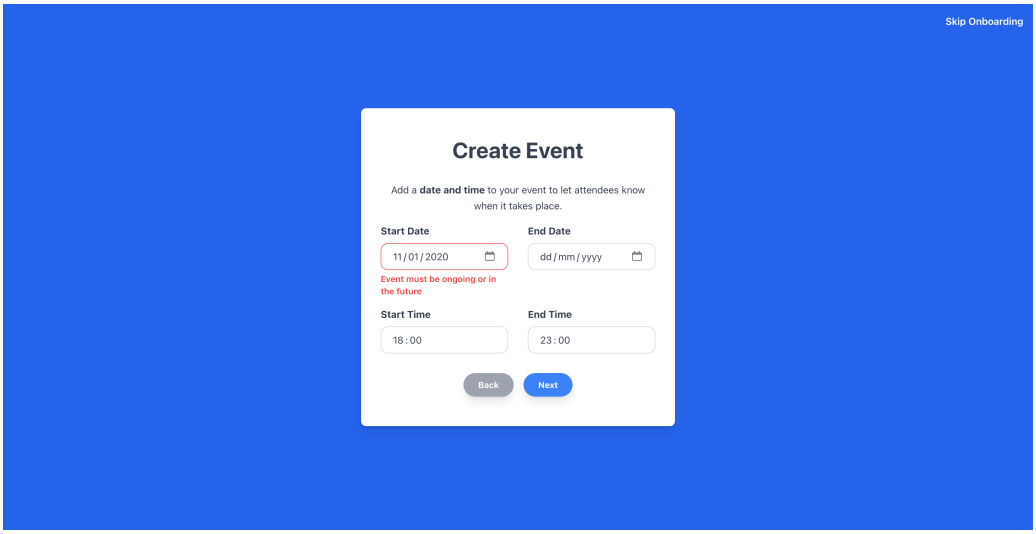


Figure 8.12: Organizer onboarding step-by-step guide with invalid dates

The last step is a summary of the entered data, where the organizer can review the information before creating the event. The layout of the summary is similar to the event single page, with the event title

at the top, followed by the description on the left side, and other event details on the right side. The organizer can go back to any step to change the data, or click the 'Save' button to create the event. After the event is created, a message is displayed to the organizer, giving them the option to view the just created event, edit the event, or create another event. The organizer is then redirected to the respective page, depending on the action they chose.

8.4.8 Responsive Design

To make the application responsive on different screen sizes, different breakpoints were defined. Tailwind already includes five breakpoints by default at '640px', '768px', '1024px', '1280px', and '1536px'. Additionally, two custom breakpoints were added at '400px' and '450px' to improve the responsiveness on smaller screens. As Tailwind is mobile first, the default breakpoint is the smallest one, and for each bigger breakpoint the breakpoint-specific classes are added to the HTML elements. The smallest screen size supported is '300px', which is the size of a small smartphone. Additionally, the viewport meta tag is added to the HTML head to ensure that the application is displayed correctly on mobile devices. The tag includes the 'width=device-width' attribute, which sets the width of the viewport to the width of the device, and the 'initial-scale=1' attribute, which sets the initial zoom level to 1. This is important to prevent the application from being zoomed in or out when it is loaded on a mobile device. The following screenshots show the home page of the application on each breakpoint.

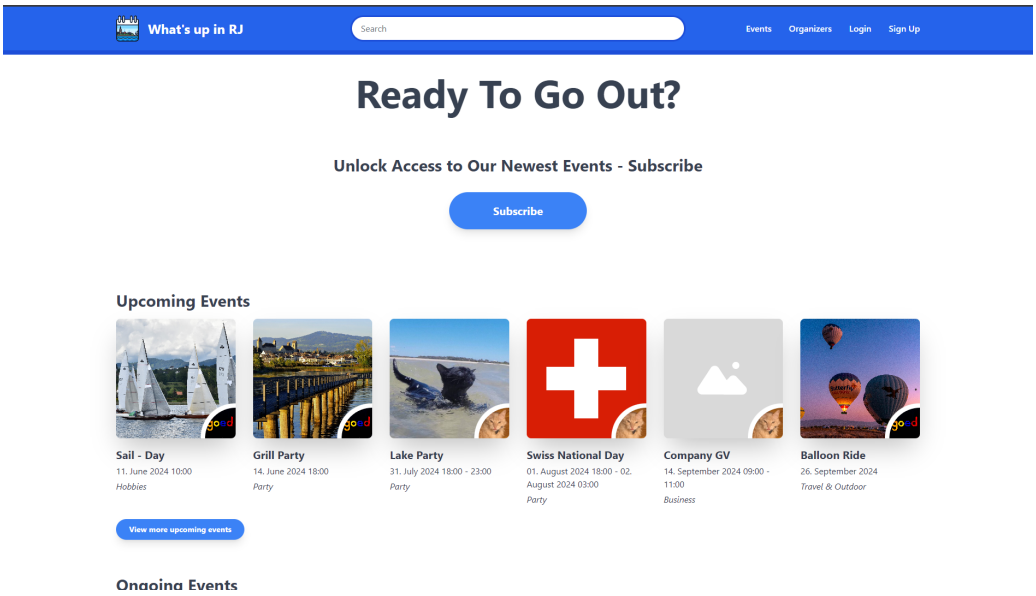


Figure 8.13: Home page on desktop screen size (1920px)

Figure 8.13 shows the home page on a desktop screen size, which is the first design created for the application.

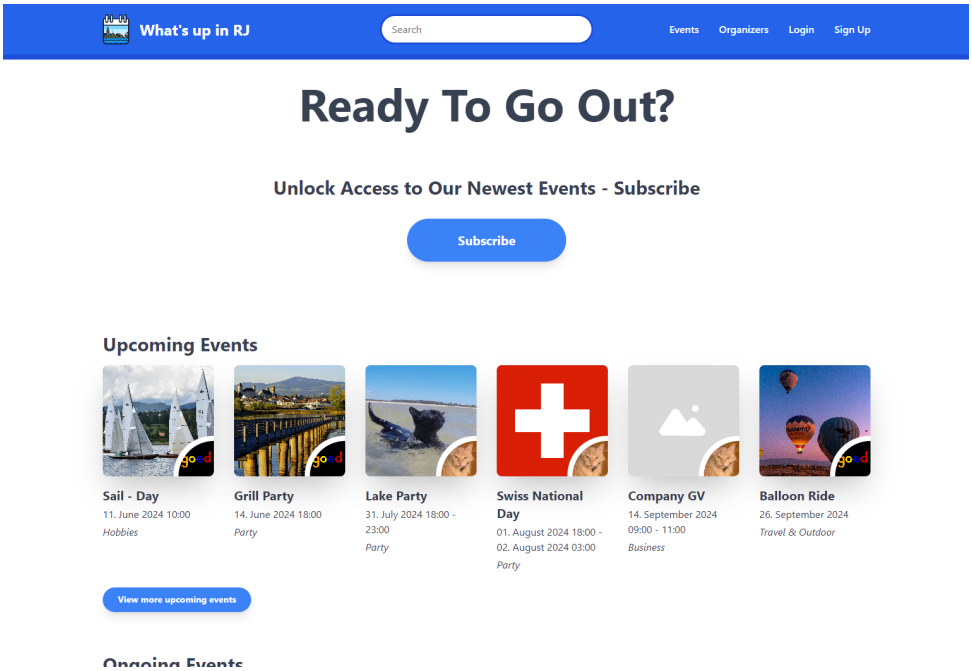


Figure 8.14: Home page on 1536px screen size

With figure 8.14, the first breakpoint is reached. The images and the search field are getting marginally smaller to fit the screen size, but there is no major layout change.

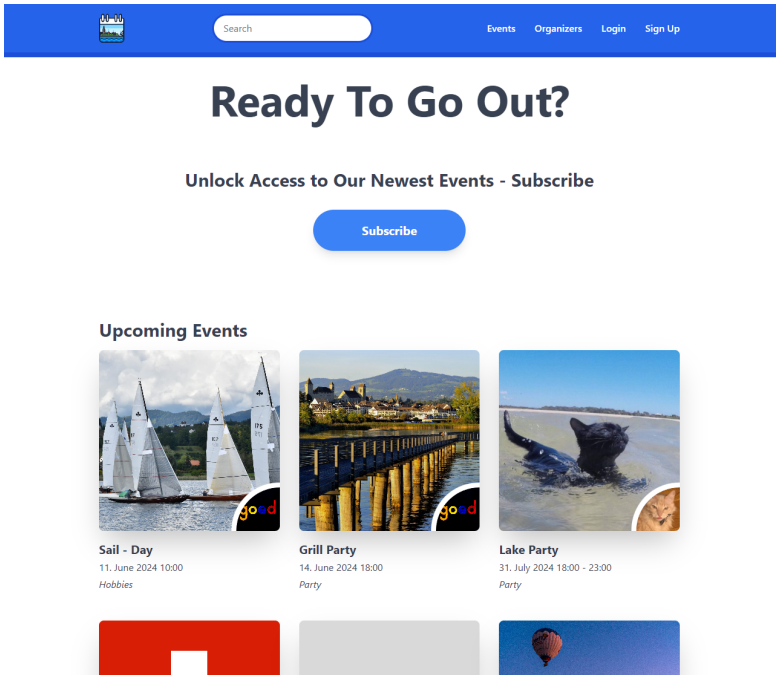


Figure 8.15: Home page on 1280px screen size

At the size of figure 8.15, the layout changes to two rows of events to give more space to the text and images. Also, the header changes and does no longer include the 'What's up in RJ' text to keep the search field and the navigation on the same line.

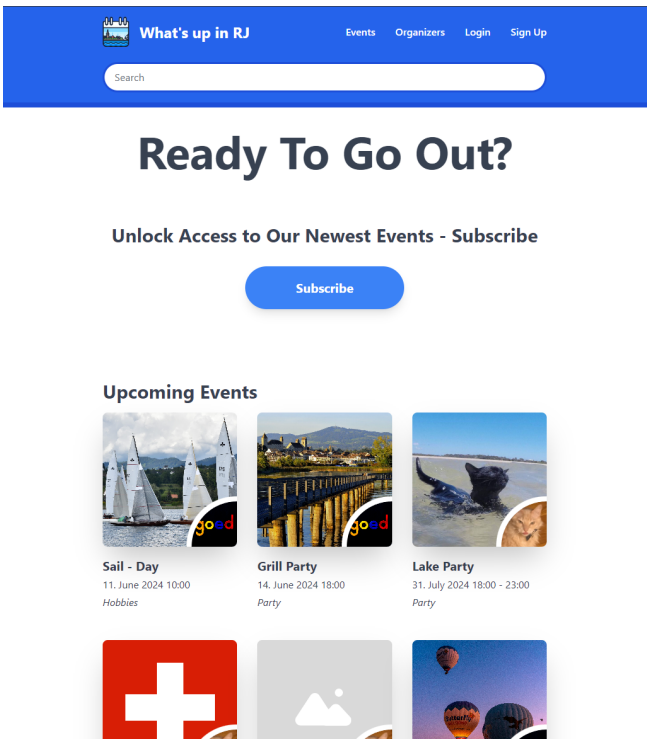


Figure 8.16: Home page on 1024px screen size

Figure 8.16 shows the home page on a '1024px' screen size. Again, the images are just getting smaller without changing the layout. The header, however, changes by breaking the search field to a new line to give more space to the navigation.

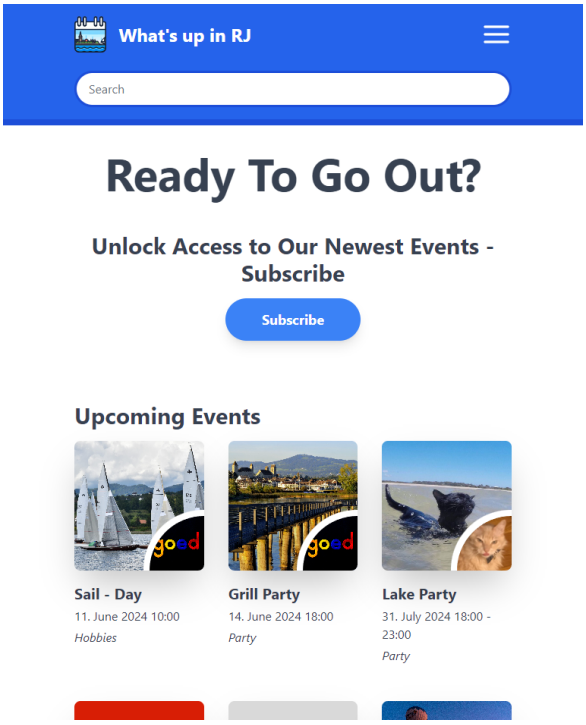


Figure 8.17: Home page on 768px screen size

At the size of figure 8.17, the images are getting smaller again to not break the layout. The header now changes to the mobile version, by having a hamburger menu to access the navigation.

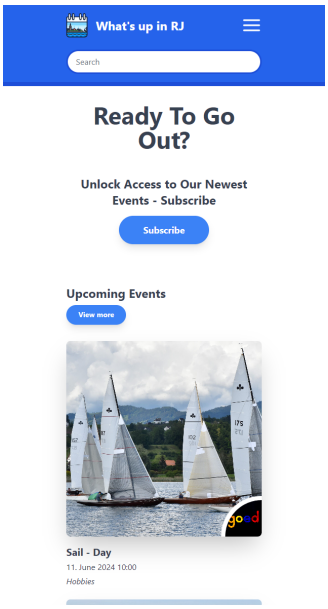


Figure 8.18: Home page on 640px screen size

With figure 8.18, the last layout change is reached. Here, the events are displayed in a single column to give more space to the images and the event information. Some buttons are moved to the title of the section to make them more visible.

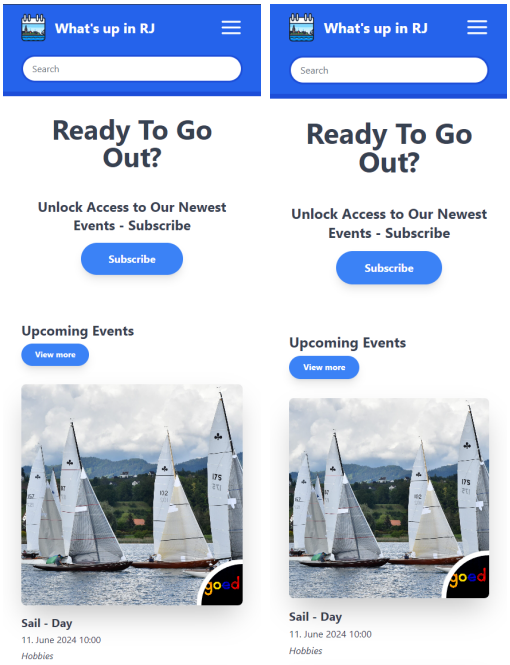


Figure 8.19: Home page on 450px and 400px screen sizes

The last two images in figure 8.19 show the home page on the smallest screen sizes. No layout changes take effect, only the images and texts are getting smaller to fit the screen size.

The breakpoints are not limited to the home page, but are used across the whole application. To keep the documentation concise, only the mobile version with a size of '450px' is described in the following sections. To view every breakpoint, access the application and resize the browser window to see the changes. The design was kept as close as possible to the mobile mock-ups, but some changes were made to improve the usability and the responsiveness of the application.

Header and Footer

The mobile header relocates the search field to a new line and implements a hamburger menu to access the navigation. By clicking on the hamburger menu, the navigation is displayed full screen, and the user can navigate to the different pages. The content of the navigation itself remains the same as on the desktop version. The following figure 8.20 shows the mobile header and the navigation with both a logged-in and a logged-out user.

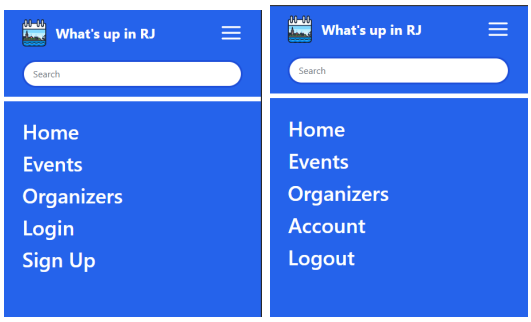


Figure 8.20: Mobile: Header

The footer on the mobile version is displayed as a single column and aligned to the left. This is done to give more space to the content and to make it easier to click on the links, as shown in the figure 8.21.

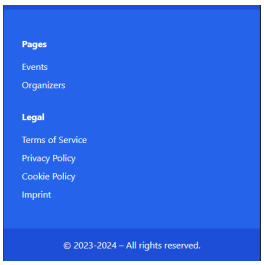


Figure 8.21: Mobile: Footer

An invisible overlay is added to the page to detect clicks outside the filter container. Clicking on the overlay closes the filters. However, the header has a higher z-index than the overlay, and blocks the click event from reaching the overlay. To mitigate this, a new service called 'HeaderClickService' was created. It listens for clicks on the header, and emits an event when it detects activity. The filter component listens for this event and closes the filters when it is emitted. This way, the filters can be closed when the user clicks on the header.

In addition, another new service called 'BottomBarService' was added to handle pages that have content fixed to the bottom of the screen. An example for this is the event overview page, where the filter container is visible at the bottom of the screen, overlapping the content. When scrolling all the way down, the bottom part of the footer would be hidden behind the filter container. The service

can be used to define a bottom margin on the footer, which is equal to the height of the bottom bar. The size of the margin is set on the page component, and is then applied to the footer through a CSS variable, which is first defined on the root element of the application. The page component sets a new value for the respective CSS variable, depending on the breakpoint, which is then applied to the footer. Each breakpoint is allowed to set a different margin size, as the bottom bar may have a different height on each breakpoint. This way, the footer is always fully visible and not hidden behind the bottom bar.

Search

Search contains two pages, the search popover and the search results. The search popover is displayed when the search field includes at least three characters. On the mobile version, the popover is displayed full screen to give the search results enough space. This popover can be closed by clicking the 'Close' button that is fixed to the bottom. Figure 8.22 shows the search popover.

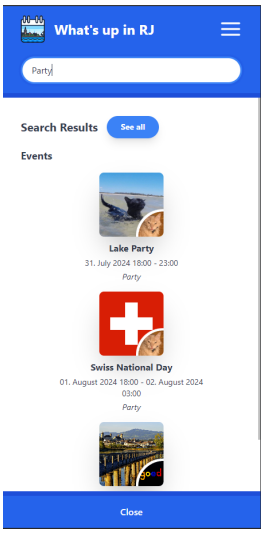


Figure 8.22: Mobile: Search popover

By clicking on the 'See All' button, the search results are displayed. This page then includes all search results including the tab navigation to switch between event and organizer results. The design of the event or organizer tab is the same as on the overview pages.

Event Overview

The following figure 8.23 shows the event overview on the mobile version.

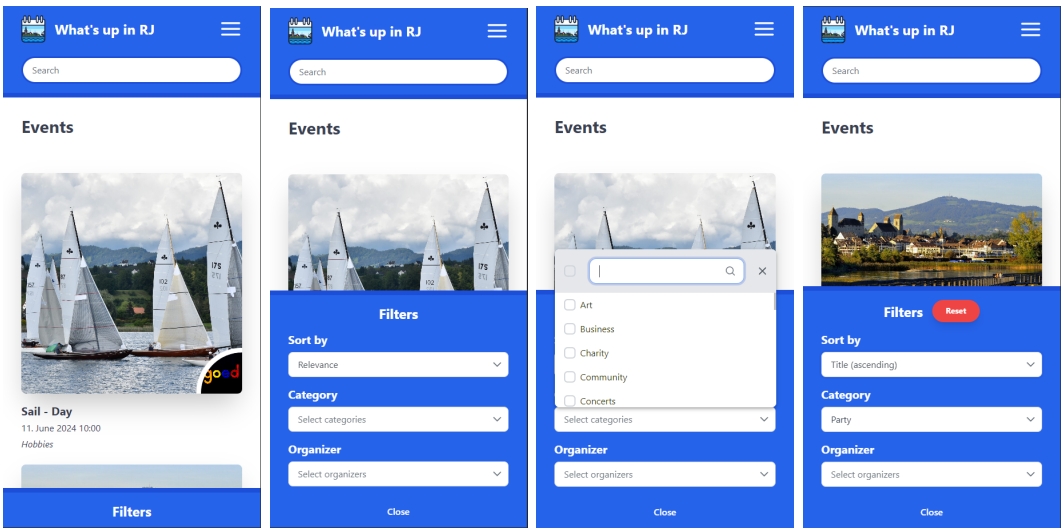


Figure 8.23: Mobile: Event overview

As visible on the first screenshot, the events are displayed in a single column to give more space to the images and the event information. A fixed filter button is displayed at the bottom of the page, which opens the filter popover that includes the same filters as on the desktop version. By clicking on each filter field, either a dropdown menu or a select field is displayed to select the filter value, for example the category. The filters can be cleared by clicking the 'Reset' button that is only visible when at least one filter is selected.

Event Single

The event single page on mobile devices moves all the information and images to a single column. Therefore, the details are moved into its own collapse element below the date and time. By default, the details are open, showing information about the organizer, location and the category. The collapse element can be closed by clicking on the header of the element. Furthermore, the image carousel does no longer include the indicators below the images, as the usability on mobile devices is limited. The previous and next buttons of the carousel are still available, but are displayed on the image itself to save space. The 'Similar Events' section displayed at the bottom of the page includes the same event cards as i.e. the event overview. The following figure 8.24 shows the event single page on the mobile version.

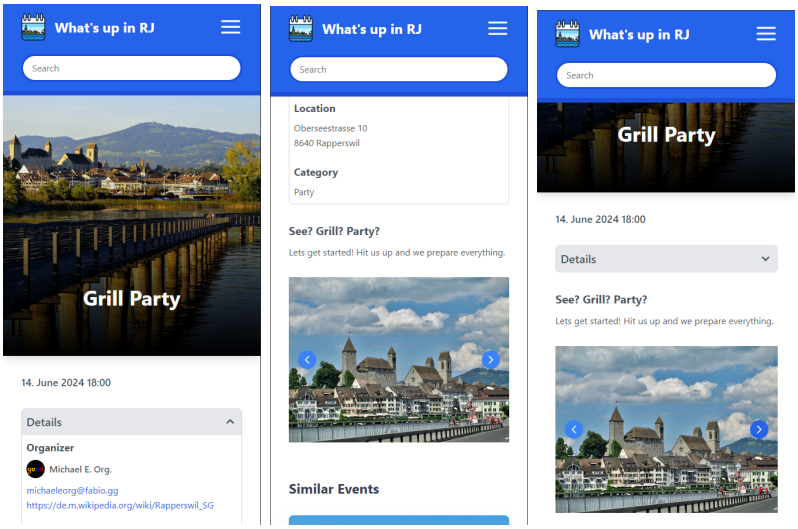


Figure 8.24: Mobile: Event single

Event Create/Edit

The event create and edit pages on mobile screens are also aligned in a single column. Otherwise, there are no significant changes to the layout, except for some scaling of the elements. As the edit and the create page are very similar, only the edit page is shown in figure 8.25.

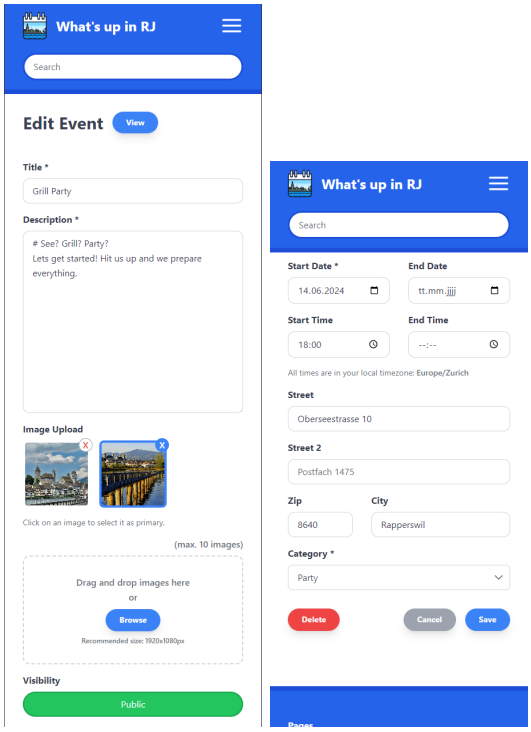


Figure 8.25: Mobile: Event edit

Organizer Overview

The organizer overview has the same layout as the event overview. Each organizer with its avatar image, name and location is displayed in a single column similar to the event cards. The filters have the same design as the event overview and the same functionality as the desktop version.

Organizer Single

The figure 8.26 shows the organizer single page on the mobile version.

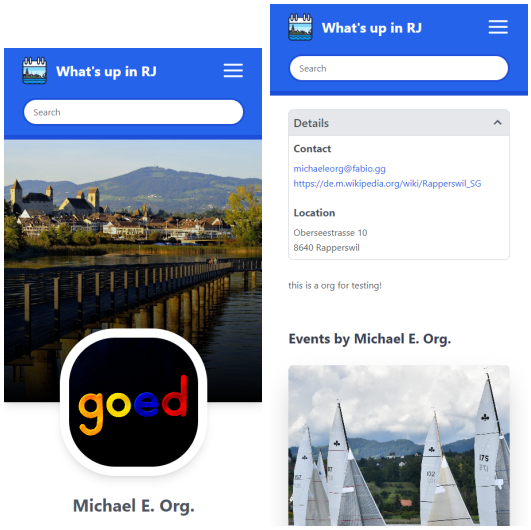


Figure 8.26: Mobile: Organizer single

The first screenshot shows that the avatar image and the name of the organizer are centred. Similar to the single page of an event, the details are displayed in a container that can be opened and closed. Below the collapse element, the description of the organizer is displayed. The content is again aligned in a single column as on the other pages. At the end of the page, a section of events created by the organizer is visible, which has the same layout as the events on other pages, effectively reusing the event card component.

Profile Editor

The profile editor implements the same style as the event edit page. The elements from the browser layout are aligned in a single column. No significant changes are made to the layout, only the scaling of the elements is adjusted to fit the screen size.

Account

The account page has three different layouts for each user type. Each user type has options that are shared between the different user types, such as the email address, the username or the password. The following figure 8.27 shows the account page with the options most users have. Only the 'Delete Account' section is not visible to an administrator, else the layout is the same for all user types.

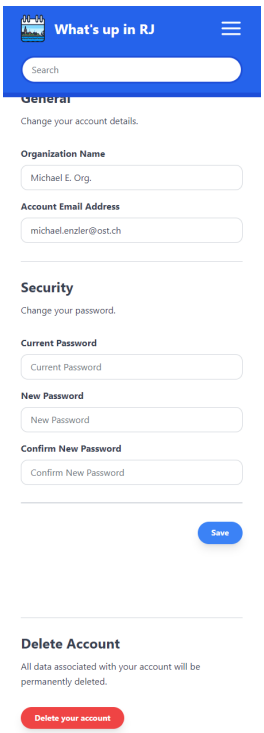


Figure 8.27: Mobile: Account page

The design of the overall account page is the same as the desktop version. However, the input fields are getting smaller and the left navigation disappears to save space. For organizers, the account page includes some more sections, like the event management, as shown in the figure 8.28.

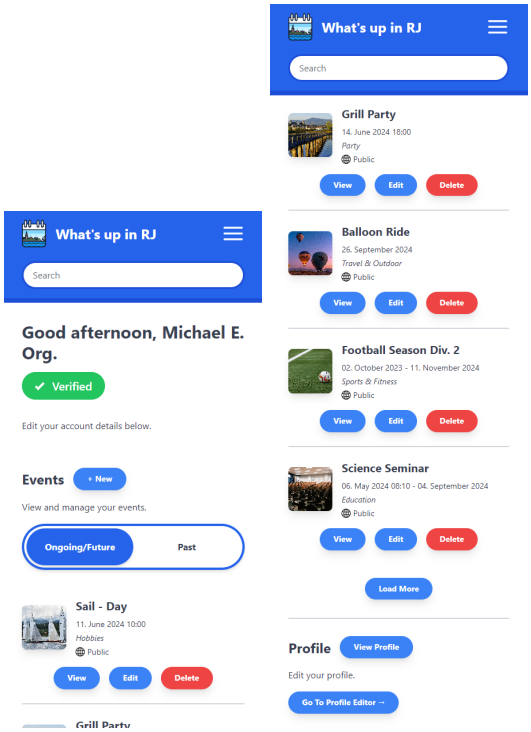


Figure 8.28: Mobile: Account page for organizers

As visible in the first picture above, the tab navigation between the ongoing and past events remains the same design and is only scaled down. The events itself now have the buttons to view, edit or delete an event below the event information as they would not have space anywhere else. The profile section does not change from the desktop version. On the other hand, the 'Preferences' section, which is only visible to users, including the button and the selection field to choose the preferred categories as shown in the figure 8.29 is only scaled down to fit the screen size.

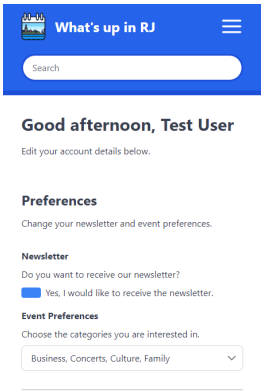


Figure 8.29: Mobile: Account page for users

Onboarding

The onboarding pages are keeping the same overall design as the desktop version. Some minor changes were made, such as the second step of the organizer onboarding, which is now displayed in a single column. The following figure 8.30 shows some screenshots of the onboarding process on the mobile version to give an overview of the design.

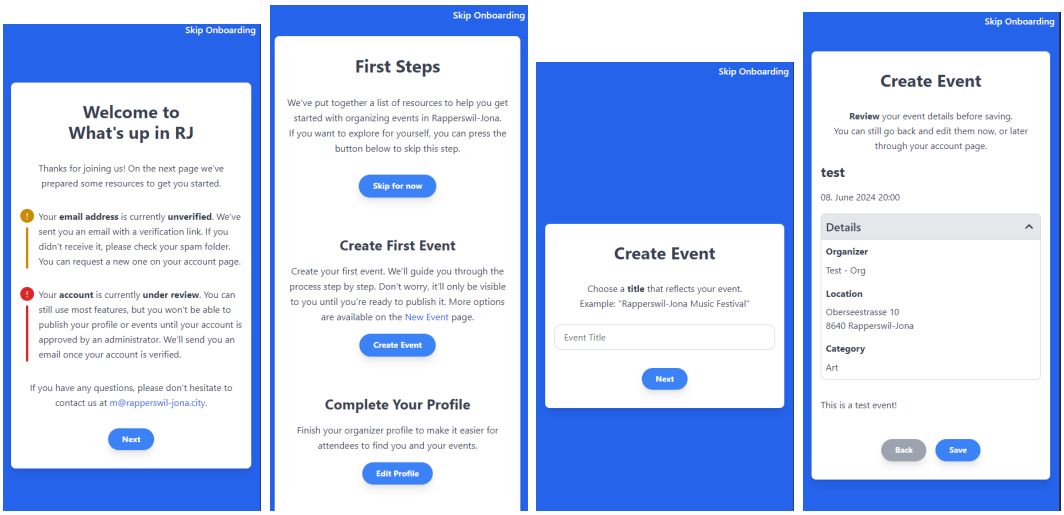


Figure 8.30: Mobile: Organizer onboarding

The provided screenshots show the onboarding process for an organizer. The user onboarding has the same design albeit with different content.

Admin Panel

The admin panel consists of two pages, the user and the category management. To access the different pages, the hamburger menu has to be opened to reveal the navigation. This navigation includes the tabs for the user and the category management if an admin is logged in, as shown in the figure 8.31.

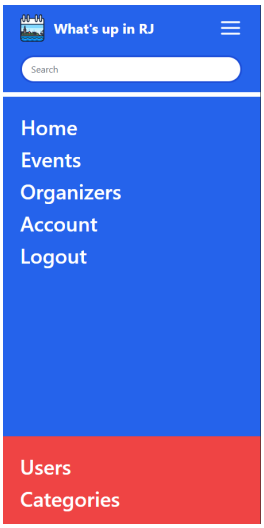


Figure 8.31: Mobile: Admin panel navigation

The following figure 8.32 shows the user management page on the mobile version.

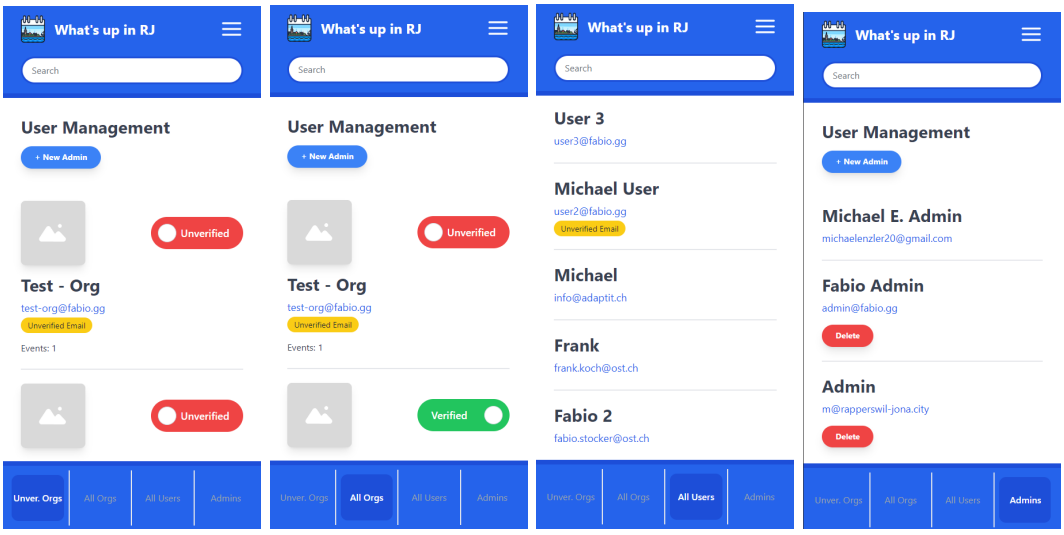


Figure 8.32: Mobile: User management

The tab navigation is now displayed at the bottom of the page to enhance the usability on mobile devices. On this screen size, the organizer cards have their information section below the avatar image. This was done to keep a good layout and to ensure long names or email addresses are displayed correctly. On even smaller screens, the 'verify' toggle will break down below the organizations' information as there is no space left to display it on the right-hand side of the image. The user tab only had some minor changes, such as the line break of the 'Unverified Email' badge. The same changes were made to the admin tab with the 'Delete' button that is now displayed below the admin's information.

To create a new admin, the ‘New Admin’ button leads to a new page with a form that has the same styling as the sign-up page described in section 8.4.8.

The figure 8.33 shows the category management page on mobile screens.

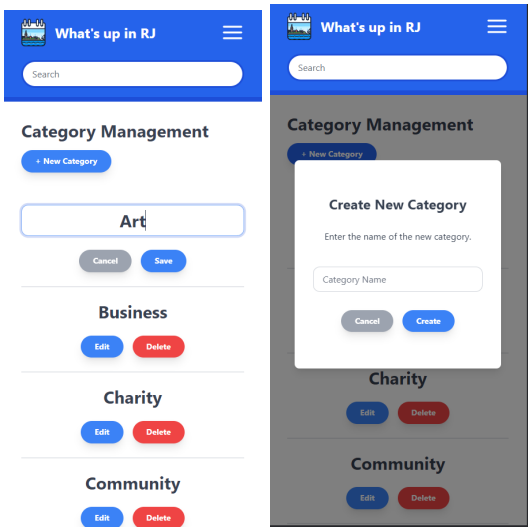


Figure 8.33: Mobile: Category management

The buttons to edit and delete a category are now displayed below the category as there is not enough space to display them elsewhere. Also, the category card is centred to ensure the layout is clean and easy to read. The new category form is still displayed in a highlighted tab, only the size of this element is scaled down to fit the screen size.

Login and Sign Up

The log-in and sign-up pages only lose their grey background, which is visible on the desktop version to give the user a better feeling of the form. As the design is not applicable to smaller screens, the grey background is removed. Except for some scaling and spacing of text elements, the design remains largely the same, as visible in the figure 8.34.

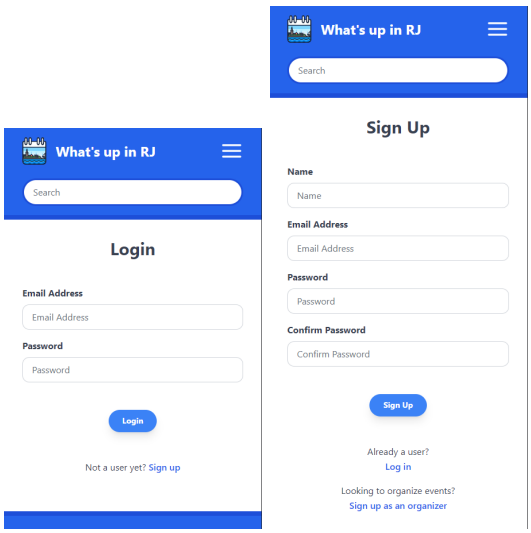


Figure 8.34: Mobile: Login and sign up

8.5 Deployment

The application is deployed to DigitalOcean App Platform, a Platform-as-a-Service (PaaS) offering by DigitalOcean. App Platform allows for easy deployment of web applications without the need to manage servers or infrastructure. It automatically deploys the application when changes are pushed to the 'main' branch of the GitLab repository. The deployment process is fully automated and usually requires no manual intervention. If new environment variables are added, or changes are made to the deployment configuration, the deployment process may require manual review. The application is accessible at the following URL: `https://whatsup.rapperswil-jona.city/`. The base configuration of the deployment has already been done in a previous phase, and only minor changes were necessary to deploy the application. New alert policies were added to monitor the application in case of deployment failures or domain configuration issues. If an alert is triggered, the responsible person is notified via email. The majority of the deployment has been described in the 'Studienarbeit' document, and is not repeated here. This section focuses on the changes made to the deployment process during the construction phase.

8.5.1 New Plans

Since the last project iteration, DigitalOcean has introduced new plans for the App Platform. The new plans offer the same pricing and features as the legacy plans, but with higher egress bandwidth. The current deployment uses the new plans, which offer 10 GiB more egress bandwidth compared to the legacy plans.

In addition, the development database now uses Postgres 14 by default, compared to Postgres 12 before, and can be upgraded to Postgres 16 if necessary. The new database versions were released in the late stages of the construction phase, thus only Postgres 14 is used for the database. Switching to Postgres 16 is not necessary at the moment, as the application has already been tested with Postgres 14, and it runs without issues. Updating the database version to the latest version is usually recommended to benefit from new features and security updates. However, as the testing phase is already completed, and the application is running smoothly, it is not a priority to update, as it may introduce new issues that need to be resolved.

8.5.2 Environment Variables

Environment variables are used to set configuration values for the application. This includes sensitive information like the database URL, the object storage credentials, and the email service credentials. Environment variables are set in the DigitalOcean App Platform dashboard, where they are securely stored and can be accessed by the application. Each component has its own variables. The frontend component includes variables for the API URL and the object storage CDN URL. The backend component defines variables for the database connection, SendGrid related variables, DigitalOcean Spaces settings, and many more. The database component only exports its connection settings, and does not require any other environment variables. Some variables are shared between the components, like the current environment, which is set to 'production' for all components. Using app-level environment variables makes it possible to only define the variables once, and use them in all components. Variables that contain sensitive values are encrypted, ensuring that nobody gaining access to the dashboard can read the values. The environment variables are loaded into the application at runtime, allowing for easy configuration of the application without changing the code.

The frontend uses a special script to load the environment variables into the application. The script is executed before the Angular frontend is built, and creates a new file with the environment variables. This file is then used by the Angular application to access the environment variables. In this project iteration, the script has been updated to also include object storage related variables, which are used to display images in the application. The backend does not require special configuration to load the environment variables, as they are automatically loaded by the Node.js runtime. Using the 'process.env' object, which is available in Node.js, the backend can access the environment variables and use them in the application. For local development, a '.env' file is used to define the environment variables, and the 'dotenv' package is used to load the variables into the application. This is not necessary in the production environment, as the environment variables are directly set in the DigitalOcean App Platform dashboard.

8.5.3 Memory Allocation

The memory allocator for the backend component has been switched to 'jemalloc'. In order to use 'jemalloc', a new 'Aptfile' was added to the backend component, which contains the package name of 'jemalloc'. The 'Aptfile' is a file that specifies additional packages that should be installed on the build image. App Platform automatically looks for the 'Aptfile' during the build process, uses the 'heroku-buildpack-apt' buildpack, and installs the packages specified in the file. This allows for the use of 'jemalloc' in the application, which may improve memory allocation and performance. To enable 'jemalloc', the 'LD_PRELOAD' environment variable is set to the path of the 'jemalloc' library. This tells the operating system to use 'jemalloc' as the memory allocator for the application.

8.5.4 Scaling

As the application grows, it may be necessary to scale the application to handle more traffic. DigitalOcean App Platform allows for easy scaling of the application by adding more instances of the components. Scaling the application has already been considered and described in the 'Studienarbeit' document. However, with the new object storage, it may be necessary to address the scaling of the object storage. Currently, the object storage is not scaled, and only one bucket is used to store all images. This may lead to performance issues as the application grows, and more images are uploaded. To address this issue, the object storage can be scaled by adding more buckets and distributing the

images across the buckets. DigitalOcean Spaces supports 800 Requests per second per bucket[1], which should be sufficient for the current application. With the addition of the CDN, the performance of the object storage is improved, as the images are cached at the edge locations. This reduces the load on the object storage and improves the loading time of the images.

8.6 Security

Security is an important aspect of web development, especially when dealing with user data. The application was designed with security in mind, and several measures were taken to protect the application from common security threats. This section describes the security measures that were implemented in the application. It also mentions recommendations for further security improvements.

8.6.1 Attack Vectors

Multiple attack vectors were considered during the development of the application. The most common attacks are SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). More attacks like Brute Force Attacks are also considered, but are less common and are not as easy to exploit. The following sections describe the measures taken to prevent these attacks.

SQL Injection

SQL injection is a common attack vector that allows an attacker to execute arbitrary SQL queries on the database. To prevent SQL injection, all SQL queries are parameterized, meaning that user input is never directly concatenated into the query string. Instead, user input is passed as parameters to the query, which are then sanitized by Sequelize, the Object-Relational Mapping (ORM) library used in the application. Sequelize automatically escapes user input to prevent SQL injection attacks.

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is another common attack vector that allows an attacker to inject malicious scripts into web pages viewed by other users. To prevent XSS attacks, all user input is sanitized before being displayed on the page. This includes escaping special characters and validating input to prevent the execution of malicious scripts. Angular by default escapes all user input, making it difficult for an attacker to inject malicious scripts into the page. The ‘description’ field of the event and organizer is an exception, as it supports Markdown, which allows the user to format the text. To prevent XSS attacks in the ‘description’ field, the ‘DOMPurify’ library is used to sanitize the input before rendering it on the page. ‘DOMPurify’ is a library that removes potentially dangerous HTML and JavaScript from user input, making it safe to display on the page. Listing 8.7 shows how the ‘marked’ package is used in combination with the ‘DOMPurify’ library to sanitize user input in the backend. The same is done on the frontend for the review of the created event on the onboarding process.

[1] https://www.digitalocean.com/blog/scale-kubernetes-spaces-object-storage#scalability-and-performance

```
// Convert Markdown to HTML and sanitize the output for XSS protection
export const markdownToHtml = (markdown: string): string => {
    // Use marked to convert Markdown to HTML
    const html = marked.parse(markdown) as string;

    // ...

    // Use DOMPurify to sanitize the generated HTML
    return DOMPurify.sanitize(html);
};
```

Listing 8.7: Sanitizing user input with DOMPurify

Additional protection against XSS attacks could be achieved by using Content Security Policy (CSP) headers. CSP headers allow the server to specify which content is allowed to be loaded on the page, preventing the execution of malicious scripts. CSP headers can be configured to only allow scripts from trusted sources, such as the application's domain, and to block all other scripts. This would provide an additional layer of security against XSS attacks and prevent attackers from injecting malicious scripts into the page. Due to time constraints and other priorities, CSP headers were not implemented in the application during the construction phase. However, it is recommended to add CSP headers before deploying the application to production to protect against XSS attacks.

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is an attack vector that allows an attacker to execute unauthorized actions on behalf of a user. Currently, the application is partially protected against CSRF attacks. Some efforts have been made to prevent CSRF attacks, such as:

- Using the 'SameSite' attribute for cookies to prevent CSRF attacks via cookies. This value is set to 'strict' to prevent cookies from being sent in cross-site requests.
- Enabling the 'HttpOnly' flag for cookies to prevent client-side scripts from accessing cookies.
- Cookies are set to be 'Secure' to prevent them from being sent over an insecure connection.
- Using the 'keys' option in the cookie session configuration to sign and validate the session cookie value, preventing tampering with the session cookie.
- Expiring the session cookie after 24 hours to not keep the user logged in indefinitely.
- Using Cross-Origin Resource Sharing (CORS) to mitigate CSRF attacks by restricting the domains that can make requests to the server.

To completely prevent CSRF attacks, a CSRF token should be generated for each request and validated on the server. The token should be unique for each user session and should be included in all requests that modify the state of the application. This would prevent attackers from executing unauthorized actions on behalf of a user. Because of other priorities and time constraints, this feature was not implemented, but it is recommended to add it in the future before going live.

Brute Force Attacks

Brute force attacks are another common attack vector that allows an attacker to guess a user's password by trying different combinations. To prevent brute force attacks, rate limiting is used to limit the number of requests a user can make within a certain time frame. If a user exceeds the rate limit, they are temporarily blocked from making further requests. This prevents attackers from guessing a user's password by trying different combinations. Currently, the rate limit is set to 100 requests per minute, but this value can be adjusted based on the application's needs.

Passwords

Passwords are stored securely in the database by using the 'bcrypt' library to hash and salt the passwords. When a user registers or changes their password, the password is hashed using the 'bcrypt' library, which generates a random salt and hashes the password with a secure algorithm. In total, the password is hashed 15 times to increase the security of the password. The hashed password is then stored in the database, along with the salt used to hash the password. When a user logs in, the password they enter is hashed with the same salt and compared to the hashed password in the database. This ensures that even if the database is compromised, the passwords are not easily decrypted. Additionally, a stronger password policy is enforced to ensure that users choose secure passwords. The password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, one number, and one special character.

Content Moderation

Organizers are able to generate content for their profile and events, which is displayed to the public. To prevent malicious content from being displayed, only verified organizers are allowed to publish any content. A manual verification process is in place to verify organizers, ensuring that only legitimate organizers can publish content. Administrators can also unverify an organizer after they have already been verified, making the organizer's content no longer appear on the site. For example, an organizer could add a malicious link to their profile or event description, which could lead to a phishing website. An administrator can either not verify the organizer in the first place, or unverify the organizer if they suspect malicious behaviour.

In the future, additional measures could be taken to prevent malicious content from being displayed. For example, links inside descriptions could be checked for malicious content using a third-party service. The service could scan the link for malware, phishing, or other malicious content, and block the link if it is found to be malicious. Additionally, a report button could be added to allow users to report malicious content or behaviour. When a user reports content, the content is reviewed by an administrator, who can take action to remove the content or unverify the organizer. Using reCAPTCHA to prevent bots from creating fake accounts or spamming the site could also be considered.

8.6.2 Data Protection

Sensitive data is protected in several ways to ensure the security and privacy of user data. The following measures are taken to protect sensitive data:

- HTTPS: All communication between the client and the server is encrypted using HTTPS. This prevents attackers from intercepting sensitive data, such as passwords or personal information. The traffic between the backend and the database is also encrypted using SSL/TLS to prevent eavesdropping.

- Environment Variables: Environment variables are used to store sensitive information, such as API keys and database credentials. Environment variables are not stored in the codebase and are only accessible to the server. This prevents sensitive information from being exposed in the codebase or in version control.

8.6.3 Third-Party Libraries

Multiple third-party libraries are used in the application to add functionality and improve security. To ensure the security of the application, third-party libraries are to be checked for vulnerabilities regularly and updated to the latest version. The 'npm audit' command is used to check for vulnerabilities in the dependencies of the application. If any vulnerabilities are found, the dependencies are updated to the version that fixes the vulnerability. Additionally, the 'npm outdated' command is used to check for outdated dependencies and update them to the latest version. This ensures that the application is up-to-date and protected against known vulnerabilities. Sometimes, updates may include breaking changes, which require changes in the codebase. In such cases, the codebase is updated to be compatible with the latest version of the dependency.

8.6.4 Security Recommendations

To summarize, the following security recommendations are made to improve the security of the application:

- Implement Content Security Policy (CSP) headers to prevent Cross-Site Scripting (XSS) attacks.
- Add Cross-Site Request Forgery (CSRF) tokens to prevent CSRF attacks.
- Regularly check for security vulnerabilities in the application and fix them as soon as possible. This includes checking for vulnerabilities in third-party libraries and updating them to the latest version.
- Use strong password policies to ensure that users choose secure passwords.
- Rate limit requests to prevent brute force attacks.
- Moderate content to prevent malicious content from being displayed. Adding a report button to allow users to report malicious content or behaviour can help with this.
- Regularly review and update security measures to ensure the security of the application.
- Educate users about security best practices, such as choosing secure passwords and not sharing sensitive information.

8.7 Code Documentation

8.7.1 JSDoc Comments

Documenting the code lets other developers understand the codebase and maintain it more easily. Some core functions and helpers are documented using JSDoc comments, which are a standard for documenting JavaScript code. JSDoc comments are written above the function definition and describe the purpose of the function, the parameters it takes, and the return value. For example, all functions inside the 'stores' directory are documented using JSDoc comments. A specific example is the 'getAllByEventId' method inside the 'EventImageStore' class, which is used to get all event images by an

event ID. The code editor can display the JSDoc comments as tooltips when hovering over a function, making it easier to understand the purpose of the function. This is especially useful for developers who are new to the codebase and need to understand how a function works. Listing 8.8 shows the JSDoc comments for the 'getAllByEventId' method in the 'EventImageStore' class.

```
/**
 * Gets all event images by event ID
 *
 * @param eventId The ID of the event to get event images for
 * @param options The options to use
 * @param transaction The transaction to use
 * @returns All event images for the event with the given ID
 * @throws {Error} if an error occurred
 */
static async getAllByEventId(eventId: UUID, options?: EventImageFilterOptions,
    transaction?: Transaction): Promise<EventImagePublic[]> {
    // Implementation
}
```

Listing 8.8: EventImageStore Comments

8.7.2 Type Annotations

Type annotations are used throughout the codebase to improve code quality and readability. They are added to variables, function parameters, and return values to specify the type of data that is expected. Using the example in listing 8.8, the 'eventId' parameter is annotated with the type 'UUID', which specifies that the parameter should be a UUID. Furthermore, the 'options' parameter is of type 'EventImageFilterOptions', which can be used to pass additional options to the function. Lastly, an optional parameter 'transaction' is annotated with the type 'Transaction', which specifies that the parameter should be an instance of the 'Transaction' class. The return value of the function is of type 'Promise<EventImagePublic[]>', specifying that the function should return a promise that resolves to an array of 'EventImagePublic' objects. Type annotations help prevent bugs and make the code easier to understand by providing information about the expected data types. In addition to type annotations, TypeScript is used to enforce type safety and catch type errors at compile time. This ensures that the code is free of type errors and runs more reliably.

8.7.3 Backend API Endpoints

The backend API endpoints are documented in a separate file, 'backend/README.md', which is available in the code repository. It lists all available endpoints, their purpose, who has access to them, the parameters they take, and the response they return. The file can also be found in the appendix chapter 15.

8.8 Testing

The application was tested manually during development to ensure that it works as expected. The testing process involved creating test cases for each feature and verifying that the feature works correctly. Chapter 7 includes the test plan, and the test reports are available in the appendix chapter 13. Testing the application manually helped identify bugs and issues early in the development process, allowing them to be fixed before deployment. Besides manual testing by using the application, Postman

was used to test the backend API endpoints. Postman is a tool that allows developers to test API endpoints by sending requests and receiving responses. By sending specific requests to the backend and observing the response, API endpoints can be tested to ensure that they work as anticipated.

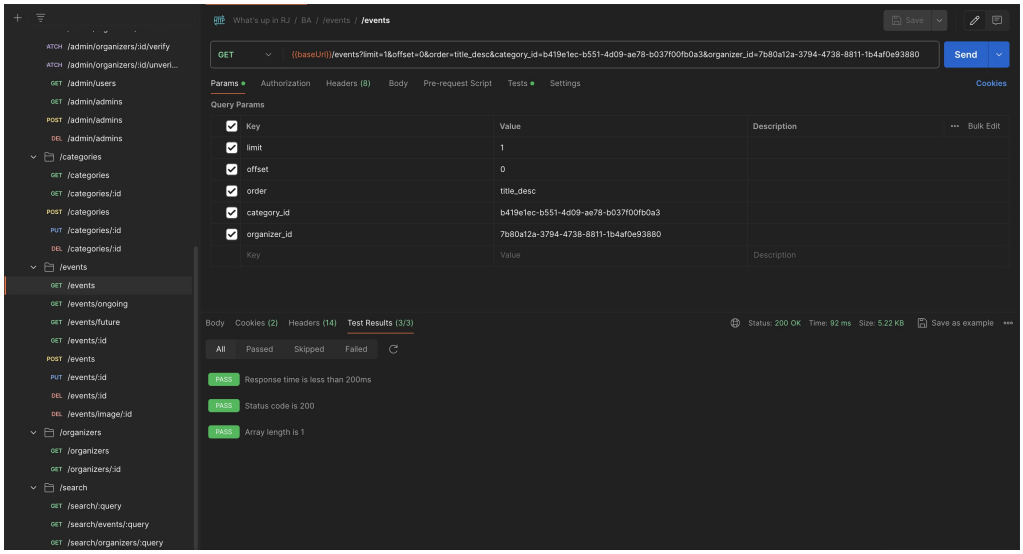


Figure 8.35: Postman screenshot of the API endpoint 'GET /events'

8.8.1 Future Improvements

To improve the testability of the application, functions with more than 50 lines should be split into smaller functions. This makes the code easier to test and maintain, as smaller functions are easier to understand and test. Additionally, unit tests and integration tests can be written to automate the testing process and ensure that the application works as expected. Unit tests can be written for individual functions to verify that they work correctly, while integration tests can be written to test the interaction between different components of the application. Automated tests can be run automatically whenever changes are made to the codebase, ensuring that new changes do not break existing functionality.

The frontend of the application can be tested using Jasmine and Karma, which are already set up with Angular. Jasmine is a testing framework for JavaScript that allows developers to write unit tests for their code. Karma is a test runner that runs e.g. Jasmine tests in a browser-like environment, allowing developers to test their code in a realistic environment. Specifically, the frontend can be tested by writing tests for individual components and services. The tests can verify that the components render correctly, that the services return the expected data, and that the application behaves as expected.

The backend can be tested by integrating a testing framework like Jest or Mocha with Chai. Another option would be to use Jasmine for testing the backend, as it is already used for the frontend, eliminating the need to learn another testing framework. API endpoints can be tested by sending requests to the server and verifying that the response is correct. To test the backend API endpoints, a test database can be set up that is separate from the production database. This ensures that the tests do not affect the production data and can be run independently of the production environment. A test coverage of at least 80% is recommended to ensure that the application is thoroughly tested and that most of the code is covered by tests. In optimal cases, the test coverage should be above 90% to ensure that the application is well-tested and reliable.

Chapter 9

Results

In this chapter, the results of the project are presented. The achieved functional and non-functional requirements are listed and the minimum viable product is described. The project is based on an initial state, which is described in chapter 2.

9.1 Functional Requirements

The functional requirements are defined as use cases in chapter 3.1. The following required use case have been implemented:

- **UC1: Manage organizers and users**
- **UC2: View responsive frontend**
- **UC3: Upload pictures**
- **UC4: Create organizer profile page**

All required use cases have been achieved. Furthermore, optional use cases were defined. As there was enough time to implement some of them, the following list contains the implemented optional use cases:

- **UC9*: View onboarding pages**
- **UC11*: Email verification**
- **UC13*: Category management**

All listed required and optional use cases have been tested. The test results are presented in the functional test protocol in appendix section 13.1.

9.2 Non-Functional Requirements

The non-functional requirements (NFRs) were defined in section 3.2. All NFRs were achieved and are shown in the following list:

- **NFR1: Collaborative**
- **NFR2: Performance**

- **NFR3: Response Time:**
- **NFR4: Responsiveness**
- **NFR5: Browser Compatibility**
- **NFR6: Availability**
- **NFR7: User Satisfaction**
- **NFR8: Scalability**
- **NFR9: Error Handling**
- **NFR10: Security**
- **NFR11: Data Privacy**
- **NFR12: Password Security**
- **NFR13: User Data Isolation**
- **NFR14: Modularity**
- **NFR15: API Testing**
- **NFR16: Deployment**

The results of the non-functional tests are presented in the non-functional test protocol in appendix section 13.2.

9.3 Minimum Viable Product

The already available web application was extended with the following features:

- Administrators can access a user management portal, where they are able to verify and unverify organizers, view all users, organizers and administrators, and create and delete other administrators.
- Everyone can view the whole application on different screen sizes as the application is responsive.
- Organizers can upload images to their profile page and events.
- Organizers can create and update their profile page that is visible to other users on the platform.
- Registered users and organizers can view onboarding pages to get started with the application after signing up.
- Registered users and organizers can verify their email address by clicking on a link sent to their email address.
- Administrators can create, update and delete categories using the category management portal.

The minimum viable product defined in section 3.3 is achieved with the first four features listed above. The other three features were not part of the minimum viable product but were implemented as optional features. Therefore, the MVP is not only achieved but also extended with optional features.

Chapter 10

Conclusion

The conclusion compares the results to the goals of the project, and outlines the future vision.

10.1 Result Reflection

The results from chapter 9 describe the achieved functional and non-functional requirements. All requirements, functional and non-functional, have been met. The project successfully delivered more requirements than originally planned, including three additional optional requirements. Therefore, the project was not only able to deliver the minimum viable product, but also additional features.

10.2 Goal Achievement

The goal of this project was to further extend the existing system with new features. An end-user is able to use new features such as uploading their own images. Furthermore, the user experience should be improved by implementing a responsive design, enabling the use of the website on phones and tablets. The overall idea was to give the system the needed features to be used in a real-world scenario.

The final product is in a state where users are able to effortlessly upload images and view the page through their smaller devices, thanks to the new responsive design. Besides the main focus of the project, even more features were added to the system described in previous chapters. The system is many steps closer to real-world usage in its current state. The project was able to overachieve its goals and is therefore considered a success.

10.3 Future Vision

As already mentioned, the project is getting closer to being published. However, there are some changes and improvements to be done. The following list gives an overview of the most important improvements that could be done in the future:

- Add a feature that lets users reset their passwords without manual intervention of an administrator.
- Add a more descriptive error message in some cases, unifying the experience and showing exactly where verification has failed.

- Add a search field and filters to the admin panel for both user and category management.
- Add a search field and filters to the organizer account page to easily find an event they have created.
- Add an edit button to the organizer single and event single pages to let the organizer easily get to their profile editor and event edit page.
- Add a dropdown menu in the header that includes items like 'Account', 'Log out', 'My Profile' for organizers, and more, creating more space for additional items in the header navigation bar.
- Automatically load the previously selected tab on page reload by adding a query parameter to the URL for all pages with tabs (search, account, admin panel, etc.).
- Show the current password strength and requirements on password fields when signing up or changing the password.
- Add the ability to show or hide the text inside a password field.
- Add a 'Create Event' or 'Go to own Events' button on the event overview page for logged-in organizers to easily add new events or view their existing events.
- Add the not implemented optional requirements defined in chapter 3.
- Add the security recommendations defined in section 8.6.4.
- Include the testing improvements defined in section 8.8.1.

The next list gives an overview of some ideas for future features:

- Add capability to search for ongoing events (e.g. event that runs from 2024-01-01 to 2024-01-31 should show up if the specified date is 2024-01-15).
- Add a new tag on the organizer single page, showcasing the organizer's join date, e.g. 'Organizer since May 2024'.
- Create a file upload component to easily reuse across the frontend.
- Add more filters to organizers on the overview and search pages, e.g. order, etc.
- Add animations to the onboarding step changes (fade or slide in/out), making it visually more appealing.
- Use PrimeNG components for more features on the frontend such as toast messages, calendar view, pagination, confirm dialogues, etc.

With these improvements and new features the system is further enhanced, and the usability is improved.

Part III

Project Documentation

Chapter 11

Project Plan

11.1 Resources

Time

The project lasts from February 19th to June 14th, 2024, with a total of 17 weeks. Each team member is expected to work around 21 hours per week, resulting in a total of 360 hours per team member. For the entire team, a total of 720 hours is allocated to the project.

Costs

The costs of the project are to be kept as low as possible. No one-time costs are expected, as the project is based on open-source software. Recurring costs include the domain, server hosting and object storage. The domain is provided by the industry partner and has already been paid for, therefore no additional costs are expected for the domain. The previous project has already set up the server hosting necessary for this project, which is also provided by the industry partner through DigitalOcean. Total costs for the server hosting, including static site hosting (free)[1], a single low performance node for the backend ($5/mo)[2], and a Postgres development database ($7/mo)[3], are $12 per month. The object storage, DigitalOcean Spaces, is a new expense, which is set to be $5 per month at current usage levels[4]. Therefore, the total costs for the project are $17 per month. Regarding bandwidth limits, only egress traffic to external Points of Presence (PoPs) counts towards the application's bandwidth allocation and may be charged[5]. No costs occur for the following items, as they are free of charge:

- GitLab Instance — Provided by the university
- SendGrid — Limited to 100 emails per day[6]
- Technology stack — Open-source software
- DigitalOcean CDN — Provided by DigitalOcean for hosting the static site component
- DigitalOcean Spaces CDN — Provided by DigitalOcean Spaces as part of the object storage[7]

[1] https://www.digitalocean.com/pricing/app-platform
[2] https://docs.digitalocean.com/products/app-platform/details/pricing/#current-plans
[3] https://docs.digitalocean.com/products/app-platform/details/pricing/#development-database
[4] https://docs.digitalocean.com/products/spaces/details/pricing/
[5] https://docs.digitalocean.com/products/app-platform/details/pricing/#bandwidth-pricing
[6] https://sendgrid.com/en-us/pricing
[7] https://docs.digitalocean.com/products/spaces/details/pricing/#content-delivery-network-cdn

The following sections provide more detailed information about the specifications and costs of the components used in the project.

Frontend

The frontend is hosted as a static site and served via edge caching over DigitalOcean's CDN to end-users. DigitalOcean offers this service for free, and no additional costs are expected.

Backend Server

The backend server is hosted on a shared instance on DigitalOcean. The server has the following specifications:

- 1 vCPU (shared, fixed)
- 512 MB RAM
- 50 GiB bandwidth
- 2 GiB disk[8]
- Base plan: $5 per month

Additional bandwidth will be billed at $0.02 per GiB if the 50 GiB limit is exceeded. The bandwidth limit is prorated across the entire 28 days[9]. This means that on the first day of the month, the application receives 1/28th of its monthly bandwidth allocation. The application will be billed for the additional bandwidth used if the application usage exceeds this limit.

Database

The database is hosted on a shared infrastructure on DigitalOcean, using their development database plan. The database has the following specifications:

- Shared CPU
- 512 MB RAM
- 1 GB Disk
- Postgres v14
- Base plan: $7 per month

Additional outbound data transfer is $0.02 per GiB extra in a month. This, however, should not be a problem as the database is only accessed by the internal backend server.

[8]https://docs.digitalocean.com/products/app-platform/details/limits/#storage-limits
[9]https://docs.digitalocean.com/platform/billing/bandwidth/#app-platform

DigitalOcean Spaces

The object storage is hosted on DigitalOcean Spaces and has the following specifications:

- 250 GiB storage
- 1,024 GiB (1 TiB) outbound transfer
- Base plan: $5 per month

It utilizes the Spaces CDN for free, which is included in the base plan. The CDN caches the content and serves it from the nearest point of presence to the user. Additional storage is $0.02 per GiB and additional transfer is $0.01 per GiB. However, these costs are not expected to occur as the base plan is sufficient for the project's needs.

11.2 Processes and Meetings

Processes

Scrum+ is used as the project management methodology, combining Scrum and Rational Unified Process (RUP) for a more comprehensive approach. Leveraging the Scrum+ methodology, the project is divided into four phases: Inception, Elaboration, Construction, and Transition. The project is further divided into sprints, each lasting two weeks, making it very agile and adaptable to change. The project's requirements are gathered and documented in the form of use cases, which are then prioritized and assigned to sprints. To ensure the project's success, the team will hold sprint planning meetings, sprint review meetings, and sprint retrospective meetings. In support of handling the project's management, Atlassian's Jira is used to organize and track the project's backlog, sprints, and issues. The Git version control system is used in combination with the GitLab platform to manage the project's source code and documentation, ensuring that the project is well-documented and that the team can work on the project simultaneously.

Meetings

Sprint planning meetings are held at the beginning of each sprint to plan the work for the upcoming sprint. Sprint review and retrospective meetings are held at the end of each sprint to review the work completed during the sprint and to discuss what went well and what could be improved for the next sprint. Stand-up meetings will be held occasionally to discuss the work completed the previous day, the work planned for the current day, and any obstacles that need to be addressed.
The team will hold regular meetings with the industry partner and the project supervisor to discuss the project's progress and to receive feedback. These meetings will be held weekly or bi-weekly on Thursday mornings at 10:00 AM, and will be held virtually using Microsoft Teams.
An interim meeting will be held with all parties involved, including experts and proofreaders, to showcase the current state of the project, and allow for feedback and suggestions. An additional meeting with all parties involved will be held at the end of the project to present the project's results, and includes a complete demonstration of the application's functionality as well as a technical discussion.

11.3 Schedule

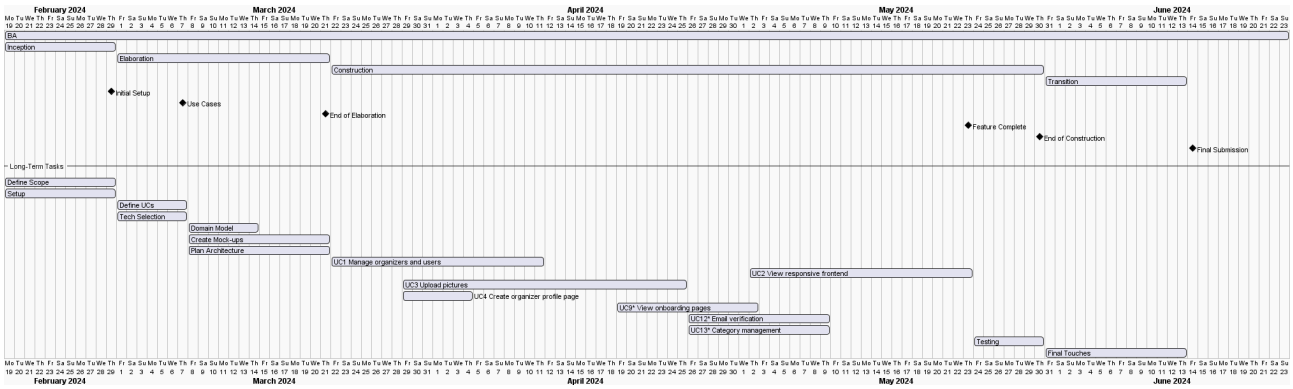


Figure 11.1: Project schedule

The remaining optional tasks UC5 to UC8, and UC11, were omitted to prioritize refining the remaining parts of the application.

Phases

Four phases are planned for the project, each lasting a different amount of weeks.

- **Inception**: The first phase of a project involves defining the project's scope and setting up the necessary environment, tools, and resources for the team to begin working. This includes the setup of Jira, the creation of a new GitLab repository for the documentation, and assigning roles to the team members. This phase is completed once the 'Initial Setup' milestone has been reached.
- **Elaboration**: In this phase, the project's requirements are converted into use cases. The domain model is created, and the architecture is defined. Technology selection is also performed in this phase. Additionally, the required mock-ups are created for the frontend. The elaboration phase is completed with the 'End of Elaboration' milestone.
- **Construction**: The construction phase is the primary stage, where the development work takes place. During this phase, the team implements the business logic and the user interface. Testing is also performed to ensure the backend and frontend work together as expected. The goal of this phase is to produce a fully functional and deployable software system. This phase is completed once the 'End of Construction' milestone is reached.
- **Transition**: The final phase will be used to finish the documentation and clean up any remaining issues. This project and phase will be finished with the milestone 'Final Submission'.

Milestones

The project is divided into several milestones, each marking the completion of a phase or a significant part of the project. Milestones are to be reached by the end of a sprint, and are used to track the project's overall progress.

- **Initial Setup**: The scope of the project has been successfully defined and the project's environment, tools, and resources are set up. This marks the completion of the inception phase.

- **Use Cases**: The project's requirements are converted into use cases.
- **End of Elaboration**: The software architecture plan is completed, and the technology stack is selected. This marks the completion of the elaboration phase.
- **Feature Complete**: The product is feature complete, and the software system is fully functional. All required features have been implemented, the product is ready for testing and additional optional features.
- **End of Construction**: The product is finished, tested, and the software system is fully functional and deployable. This marks the completion of the construction phase.
- **Final Submission**: The project is finished, and the final submission is made. This marks the completion of the transition phase.

11.4 Organization and Roles

Roles

The project is divided into several roles, each with its own responsibilities.

Role	Member
Project Owner	Michael Enzler
Scrum Master	Fabio Stocker
Backend Developer	Fabio Stocker
Frontend Developer	Michael Enzler
Deployment Manager	Michael Enzler
Architect	Fabio Stocker

Table 11.1: All roles with the assigned members

Project Owner

The project owner is responsible for defining the requirements of the project. They verify the fulfilment of all requirements, manage the processing and updating of the backlog, as well as communicate with the project advisor. They also work with the Scrum Master to ensure that the project is delivered on time.

Scrum Master

The scrum master is in charge of weekly meetings, sprint, review and retrospective meetings. They are responsible for ensuring that the development team follows the Scrum process. They also support the team's progression and remove any obstacles that may be blocking the team from completing their tasks.

Backend Developer

The backend developer is responsible for building the server-side of the software. They write the code that processes requests from the user interface and interacts with the database. They are also responsible for the communication between the frontend and the backend by exposing the needed information through an API.

Frontend Developer

The frontend developer is responsible for building the user interface and user experience of the software. They create a visually appealing and intuitive interface that is easy to use. They also ensure that the interface is responsive and works on different devices and platforms. Additionally, they are responsible for the communication between the frontend and the backend by sending requests to the backend and displaying the received information.

Deployment Manager

The deployment manager oversees the deployment process of the application. They ensure that the software is deployed to the production environment without any issues and that it is available to users. They also monitor the performance of the software and ensure that it is scalable and reliable. Additionally, the deployment manager takes care of the CI/CD pipeline during development and release.

Architect

The architect is responsible for the software architecture of the project. They ensure that the software is designed in a way that is easy to maintain and extend. Additionally, they ensure that the software is scalable, reliable and secure.

11.5 Risk Management

The following table 11.2 shows the risks identified either at the beginning of the project or during the project's lifecycle.

No.	Risk	Mitigation	Probability	Severity	Exposure
R1	Server hosting is unavailable	As a reputable cloud server provider is already in use and the existing setup will not be changed, this risk can be accepted.	Unlikely	Catastrophic	High
R2	Object storage is unavailable	Use a reputable object storage provider with low downtimes.	Unlikely	Critical	Medium
R3	Object storage charges high fees	Evaluate the costs of different object storage providers and choose the one with the best price-performance ratio.	Possible	Critical	High
R4	Existing infrastructure is not compatible with the extensions	Evaluate the compatibility of the existing infrastructure with the extensions.	Possible	Catastrophic	Very High

Continued on next page

No.	Risk	Mitigation	Probability	Severity	Exposure
R5	Team member is absent	Ensure that the team members are available and communicate any absences in advance.	Possible	Marginal	Medium
R6	Missing knowledge	Do research in advance of the construction phase to ensure that the team has the necessary knowledge.	Possible	Minor	Low
R7	Scope changes	Ensure that the requirements are well-defined during the elaboration phase. If changes occur, re-evaluate the project scope and adjust the plan accordingly.	Likely	Marginal	High
R8	Project plan is flawed	After every sprint review, evaluate the project plan and adjust it if necessary.	Possible	Marginal	Medium

Table 11.2: Initially identified risks

The risk matrix[10] was utilized to evaluate the identified risks. Additionally, the same matrix was used to display the risks in the following figure 11.2.

[10]https://en.wikipedia.org/wiki/Risk_matrix

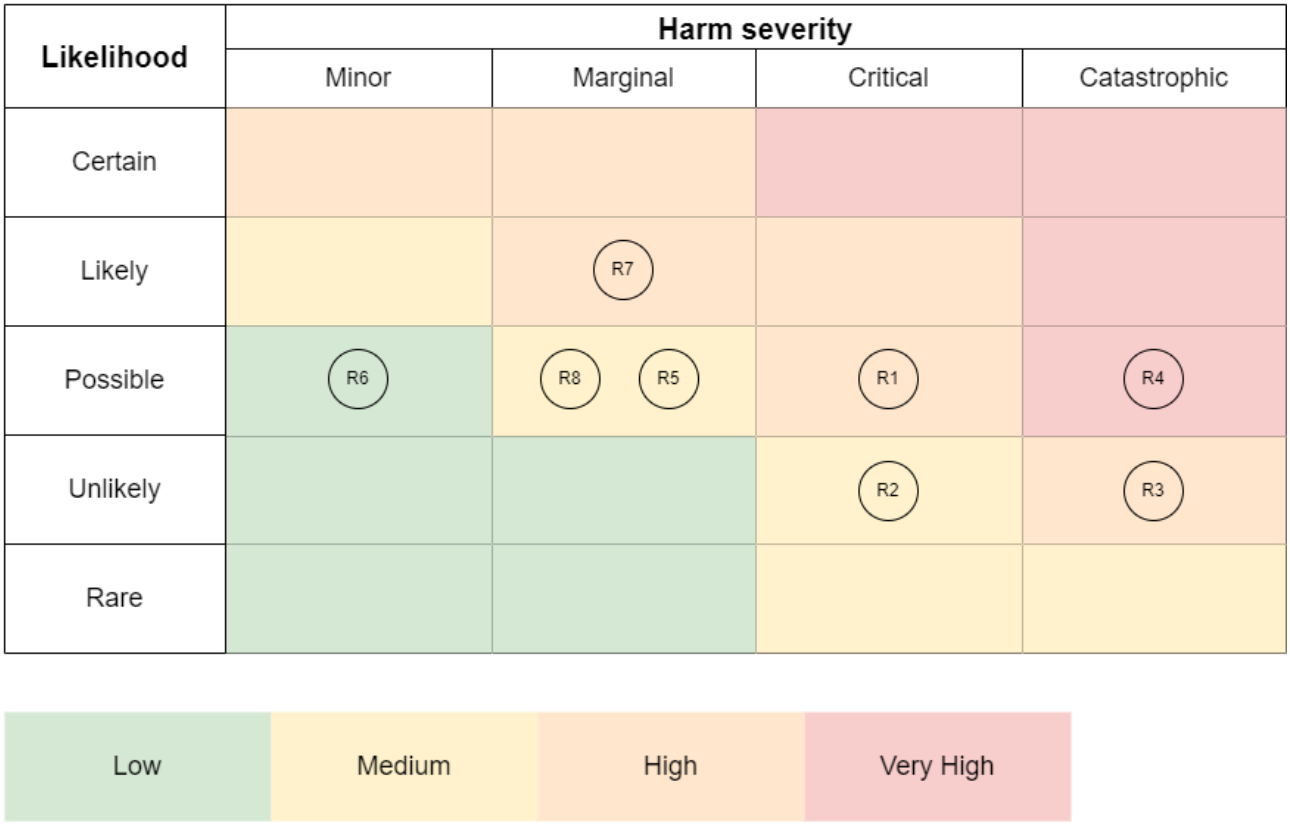


Figure 11.2: Risk matrix

Risk Handling

This section describes how the risks were handled and whether they occurred.

R1: Server hosting is unavailable

This risk was accepted with the decision to use the same reputable cloud server provider as in the existing setup. In the previous project, the 'Studienarbeit', DigitalOcean has been chosen as reputable and reliable cloud server provider. Therefore, it can be considered reputable and reliable for this project as well.

Status: Accepted
Occurred: No

R2: Object storage is unavailable

As already mentioned in the risk mitigation, a reputable object storage provider with low downtimes should be used. Therefore, this risk needs research and evaluation to find the best object storage provider. This was done in the elaboration phase of the project and the decision to use DigitalOcean Spaces is described in the technologies section 5.7. With this decision, the risk cannot be further mitigated and is accepted.

Status: Accepted
Occurred: No

R3: Object storage charges high fees

The mitigation suggests using the object storage provider with the best price-performance ratio. This risk as well as other risks rely on the decision, which object storage provider to use. In the elaboration phase of the project, DigitalOcean Spaces was chosen described in the technologies section 5.7. This decision mentions that DigitalOcean Spaces has a cost-effective pricing model. It may not be the cheapest object storage provider, but it suits the project's needs and helps to mitigate the other risks. Therefore, the costs are predictable as described in the cost section 11.1. With this decision, the risk is very unlikely to occur and is considered mitigated.

Status: Mitigated
Occurred: No

R4: Existing infrastructure is not compatible with the extensions

As most of the infrastructure is already in place and only a few extensions are planned, the risk can be mitigated by evaluating the compatibility of the existing infrastructure with the extensions. This was done in the elaboration phase of the project and the decision to use DigitalOcean Spaces is described in the technologies section 5.7. No incompatibilities have been found, and the risk is considered mitigated. But as issues can still occur during the construction phase, the risk status is set to accepted.

Status: Accepted
Occurred: Yes

The dependency used in the image upload feature had some compatibility issues with the existing infrastructure. The issue is described in detail in the section 8.4.4. To solve the issue, a temporary fix was implemented. However, to fully mitigate the issue, more research is needed to provide a solution. As the issue occurred at the end of the construction phase and time was running low, the temporary fix was considered sufficient. Still, the feature is working as intended with this fix. The risk mitigation was therefore not fully successful as incompatibilities were planned to be evaluated in the elaboration phase. This issue could not have been foreseen as it is too specific to evaluate in an early project phase. Therefore, the risk was handled successfully as the feature is working as intended.

R5: Team member is absent

To mitigate this risk, the team members are required to communicate any absences in advance. Then, it will be evaluated if the absence has a big impact, depending on how long the team member is absent. If the absence has a big impact, the project plan needs to be adjusted, and it should be discussed with the advisor. Should the absence have a small impact, the team member can catch up on the work after returning, or put in more work before the absence. The risk can be considered accepted as there is no further mitigation possible.

Status: Accepted
Occurred: Yes

Both team members had absences for a few days, either due to illness or military service. As the absences were only for 1-2 days, they had no significant impact on the project. The work was caught up on after returning and the project plan was not adjusted. Therefore, the risk mitigation was successful.

R6: Missing knowledge

As most of the technologies used are already well-known by the team members, the risk is considered minor. If there are any knowledge gaps because of new technologies that were introduced, the team members will do research during the elaboration phase. Still, it is possible that some problems occur due to missing knowledge. The problem will then be discussed in the team and the knowledge gap will be closed. Therefore, some additional time is planned for each implementation task to handle such problems. The risk cannot be further mitigated and has to be accepted.

Status: Accepted
Occurred: Yes

There were some minor cases of missing knowledge, but they were quickly solved in the team. The additional time planned for each implementation task was sufficient to handle these problems. Therefore, the risk mitigation was successful.

R7: Scope changes

It is very likely that scope changes occur during the project as there is a great possibility of new ideas and requirements popping up. For this reason, the requirements have to be well-defined during the elaboration phase, which was done in chapter 3. If there are new requirements coming up, they have to be evaluated and may be added to the project. This will be discussed with the project advisor and the industry partner. If the change gets approved, the project plan has to be adjusted. There is nothing more to mitigate this risk, and it has to be accepted.

Status: Accepted
Occurred: Yes

There were some scope changes during the project. The new requirements were defined as optional use cases as described in section 3.1. Some of these optional use cases were considered to be important and were additionally implemented. Hence, the project plan was changed after a discussion with the project advisor and the industry partner. The risk mitigation was successful.

R8: Project plan is flawed

Every two weeks, there is a sprint review where the project plan is evaluated and adjusted if necessary. With this approach, the risk cannot be mitigated fully, but minimized. The status is set to accepted as it is not fully mitigated.

Status: Accepted
Occurred: Yes

The project plan did not have any major flaws, but some task estimations were too low or too high. Additionally, there were new optional requirements, which had to be added to the project plan. With the regular sprint reviews as described in the risk mitigation, the risk was handled successfully.

11.6 Planning Tools

Jira

Jira is used to manage the project's backlog, sprints, and issues. Time tracking is also performed in Jira, allowing the team to track the time spent on each task. The plugin 'Clockwork Free' is used to summarize the recorded times per issue. The meetings and documentation efforts are also tracked by separate issues to ensure that the time spent on these activities is accounted for.

GitLab

GitLab is used to manage the project's source code and documentation. The team uses GitLab to collaborate on the project and to ensure that the project is well-documented.

11.7 Git

The team uses Git as the version control system for both the source code and the documentation.

Branching Model

The GitFlow branching model is used to manage the project's branches on the source code repository. The `main` branch is used to store the production-ready code, while the `develop` branch is used to store the latest development code. Feature branches are created from the `develop` branch and are used to develop new features. Once a feature is complete, a pull request is created to merge the feature branch into the `develop` branch. When a release is ready, a release branch is created from the `develop` branch and is used to prepare the release. Once the release is complete, a pull request is created to merge the release branch into the `main` branch and back into the `develop` branch. Finally, a hotfix branch is created from the `main` branch to fix any issues that may arise in the production code.

A similar branching model is used for the documentation repository, with the minor difference that the `main` branch is used to store the latest documentation code. The `develop` branch is left out, as the documentation is not developed in the same way as the source code. With the absence of a `develop` branch, the feature branches are created from the `main` branch and are used to document new parts of the project. Additionally, the release and hotfix branches are not used in the documentation repository, as the documentation does not get released nor does it have a production environment.

Repositories

The project is divided into two repositories: one for the source code and one for the documentation. All pull requests are reviewed by at least one other team member before being merged.
The GitLab repository for the documentation is located at
`https://gitlab.ost.ch/wuir/ba-documentation/`.
The GitLab repository for the source code is located at
`https://gitlab.com/mguenten/whats-up-in-rj/`.

Chapter 12

Time Tracking Report

Procedure

The tasks for the project were defined in the Jira backlog. At the start of each sprint, the tasks were assigned to the new sprint, and a team member was selected for each task. The ongoing sprint with its tasks were then visible on the Jira board with five columns: 'To Do', 'In Progress', 'Awaiting Review', 'Time Tracking', and 'Done'. The 'Time Tracking' column was only used for overall tasks like meetings or documentation. With the 'Clockwork Free' plugin for Jira, the time spent on each task was tracked. This data can be used to estimate if the submission is on schedule or not.

Statistics

With the tracked data from Jira, the following statistics were generated. The numbers in the figures 12.1, 12.2, 12.3 and 12.4 are in hours and rounded to one decimal place. The phase 'Other' was used to track time spent on meetings and overall documentation. Tasks that were not assigned to a specific phase are included in this phase.

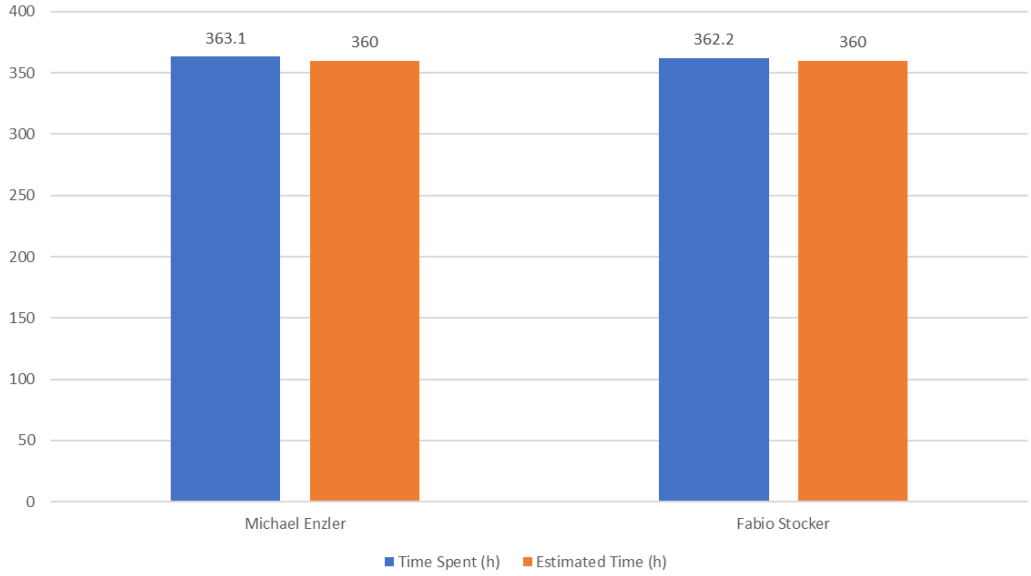


Figure 12.1: Time spent per person

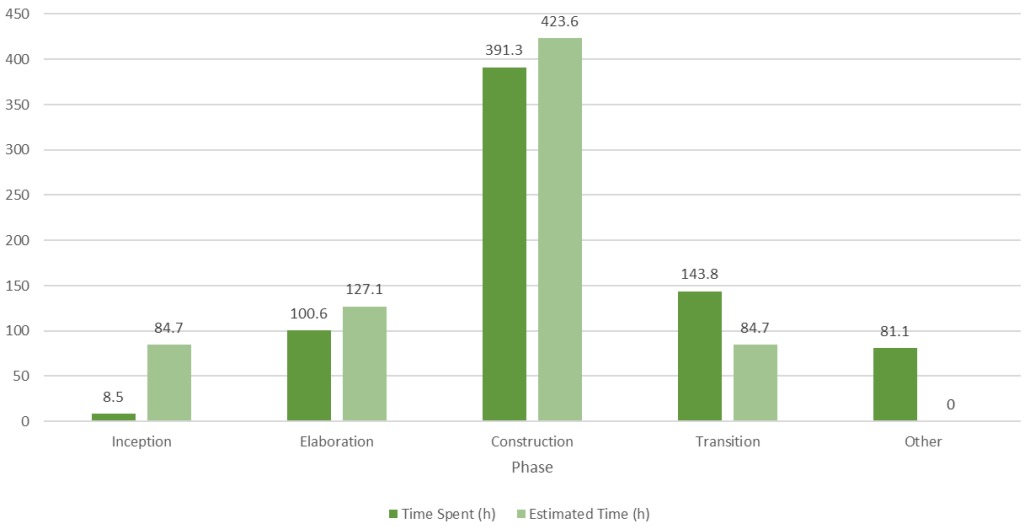


Figure 12.2: Time spent per phase

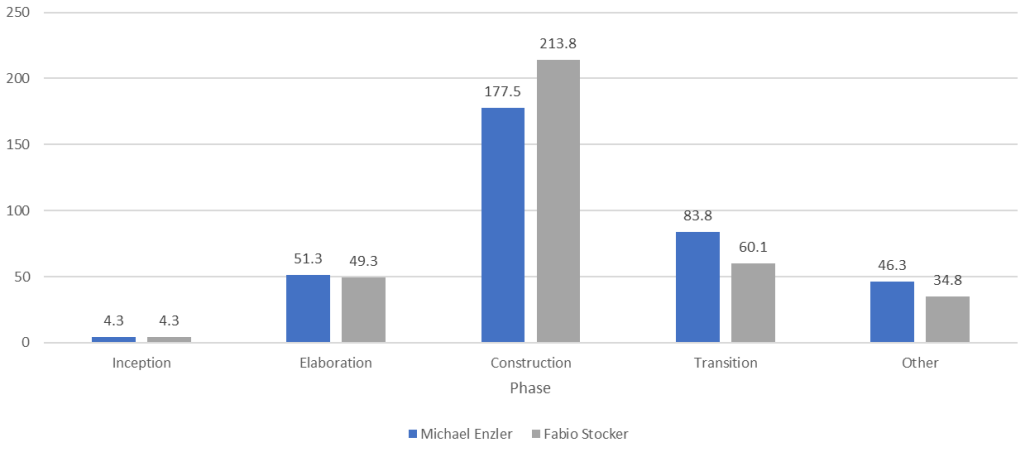


Figure 12.3: Time spent per phase and person

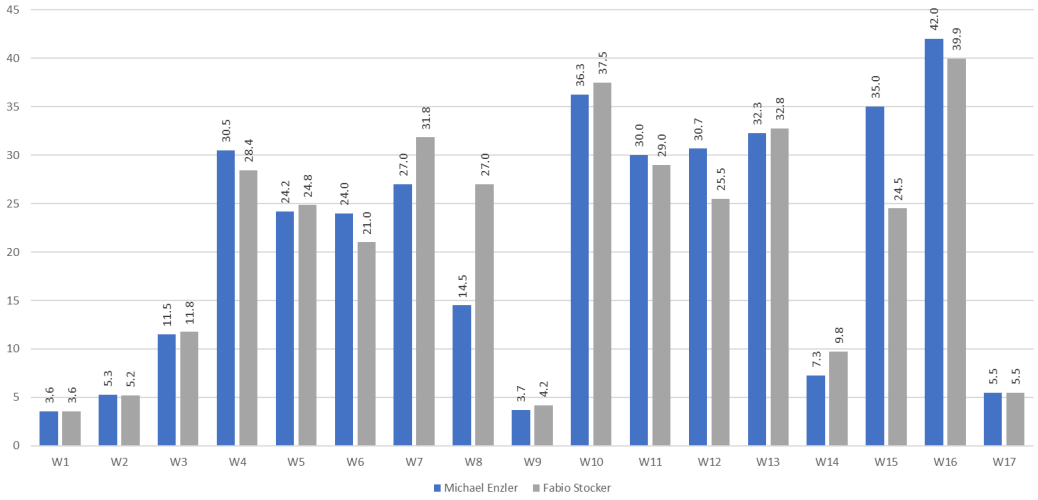


Figure 12.4: Time spent per week

Bibliography

[1] Enzler, Michael and Stocker, Fabio. *What's up in RJ.* Other thesis - Studienarbeit, OST - Ostschweizer Fachhochschule, Oberseestrasse 10, 8640 Rapperswil-Jona, 2023. Available at eprints.ost.ch/1173.

Glossary

ACL Access Control List: A list of permissions attached to an object that specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects.

AI Artificial Intelligence: The simulation of human intelligence processes by computer systems.

API Application Programming Interface: A set of rules and protocols for building and interacting with software applications.

AWS SDK Amazon Web Services Software Development Kit: A collection of tools and libraries that allow developers to create and manage resources through a unified interface. While mainly used for AWS, it can also be applied to other technologies such as DigitalOcean Spaces, which uses the same underlying S3 technology and protocols.

CD Continuous Deployment: A software development practice where code changes are automatically built, tested, and deployed to production.

CDN Content Delivery Network: A geographically distributed network of proxy servers and their data centers.

CI Continuous Integration: A software development practice where members of a team integrate their work frequently.

CI/CD Continuous Integration/Continuous Deployment: A software development practice that combines continuous integration and continuous deployment.

CLI Command Line Interface: A text-based interface used to interact with a computer program.

CORS Cross-Origin Resource Sharing: A mechanism that allows many resources on a web page to be requested from another domain outside the domain from which the resource originated.

CPU Central Processing Unit: The component of a computer system that performs the basic operations of the system.

CSP Content Security Policy: A computer security standard introduced to prevent cross-site scripting (XSS) or other code injection attacks.

CSRF Cross-Site Request Forgery: A type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.

CSS Cascading Style Sheets: A style sheet language used for describing the presentation of a document written in a markup language like HTML.

C4 Context, Containers, Components, Classes: A software architecture model for visualizing the system architecture.

DOM Document Object Model: A cross-platform and language-independent interface that treats an XML or HTML document as a tree structure.

DOMPurify A DOM-only XSS sanitizer for HTML.

ECTS European Credit Transfer and Accumulation System: A standard for comparing the study attainment and performance of students of higher education across the European Union and other collaborating European countries.

EXIF Exchangeable Image File Format: A standard that specifies the formats for images and other media.

GiB Gibibyte: A unit of digital information storage.

GPT Generative Pre-trained Transformer: A type of deep learning model that is trained on a large set of data.

HTML HyperText Markup Language: The standard markup language for documents designed to be displayed in a web browser.

HTTP HyperText Transfer Protocol: An application layer protocol to transfer data between a client and a server.

HTTPS HyperText Transfer Protocol Secure: An extension of HTTP for secure communication over a computer network.

JPEG Joint Photographic Experts Group: A file format for compressing images.

JS JavaScript: A high-level, interpreted programming language for web development.

JSON JavaScript Object Notation: A lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

JSDoc JavaScript Documentation: A markup language used to annotate JavaScript source code files.

JWT JSON Web Token: An open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

MB Megabyte: A unit of digital information storage.

MIME Multipurpose Internet Mail Extensions: A standard that extends the format of email messages to support content types other than text.

MVP Minimum Viable Product: A product with just enough features to satisfy early customers and provide feedback for future product development.

NFR Non-Functional Requirement: A requirement that specifies criteria that can be used to judge the operation of a system.

NSFW Not Safe For Work: Content that is not suitable for viewing in a work environment.

npm Node Package Manager: A package manager for the JavaScript programming language.

ORM Object-Relational Mapping: A programming technique to convert data between incompatible type systems using object-oriented programming languages.

PNG Portable Network Graphics: A format for storing graphical images.

PoP Point of Presence: A location where a network or internet service provider connects with other networks and deliver services to end-users.

PX Pixel: A single point in a graphic image.

RAM Random Access Memory: The short-term memory of a computer system.

ReCaptcha A CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) system designed to determine whether a user is a human or a bot.

RJ Rapperswil-Jona: A municipality in the canton of St. Gallen in Switzerland.

REST Representational State Transfer: A software architectural style that defines a set of constraints to be used for creating Web services.

RUP Rational Unified Process: An iterative software development process framework.

SCSS Sassy CSS: A preprocessor scripting language that is interpreted or compiled into CSS.

SQL Structured Query Language: A domain-specific language used in programming and designed for managing data held in a relational database management system.

SSL Secure Sockets Layer: A standard security technology for establishing an encrypted link between a server and a client.

SSL/TLS Secure Sockets Layer/Transport Layer Security: A standard security technology for establishing an encrypted link between a server and a client.

TS TypeScript: A strict syntactical superset of JavaScript that adds optional static typing.

TiB Tebibyte: A unit of digital information storage.

UI User Interface: The space where interactions between humans and machines occur.

URL Uniform Resource Locator: A reference to a web resource that specifies its location on a computer network.

UTC Coordinated Universal Time: The primary time standard by which the world regulates clocks and time.

UUID Universally Unique Identifier: A 128-bit number used to identify information in computer systems.

UX User Experience: The overall experience of a person using a product such as web applications.

vCPU Virtual Central Processing Unit: A CPU that is not physically present but is made up of resources from a physical CPU.

XSS Cross-Site Scripting: A security vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users.

List of Figures

List of Tables

Listings

Part IV

Appendix

Chapter 13

Test Reports

13.1 Functional Test Protocol 29.05.2024

No.	Description	Precondition	Input	Expected Output	Actual Output	Pass/Fail
1	Administrator views users, organizers and administrators	The administrator is logged in and views the user management section of the admin panel	Choose the preferred user type in the menu	A list of users, organizers and administrators will be displayed respectively	On each tab (Unverified Organizers, All Organizers, All Users and Admins) the corresponding accounts and their details are displayed	Pass
2	Administrator verifies an organizer	The administrator is logged in and views the user management section of the admin panel	Use the 'Verify' toggle	The organizer is verified, the verified badge on their account page is visible, and they can publish events	The 'Verify' toggle of the edited organizer is set to 'verified'. The organizer now has the verified badge and can publish events	Pass
3	View responsive frontend	Any system actor on the website	Change the screen size to a smaller size	The website will adjust to the new screen size	On screen size change, the website adjusts accordingly and keeps the content readable and usable	Pass

Continued on next page

No.	Description	Precondition	Input	Expected Output	Actual Output	Pass/Fail
4	Organizer uploads images to or deletes images of their profile or events	The organizer is logged in and views the event create, event edit or profile editor page	Upload an image using the corresponding field or delete an image	The image will be uploaded and displayed, or deleted	The image is uploaded and displayed on the different pages. On deletion, the image gets correctly removed	Pass
5	Organizer edits its profile page	The organizer is logged in and views the profile editor	Edit the form and submit it	The profile page will be edited	The profile page can be edited through the editor form and the changes are displayed	Pass
6*	Onboarding for users and organizers	A user or organizer has never done the onboarding process	Sign up for a new account, or log in with an existing account	The user or organizer enters the onboarding process	After signing up, the onboarding process is started and the user or organizer is guided through the process	Pass
7*	Email verification for users and organizers	A user or organizer has created an account or requested an email verification	Click on the verification link in the email sent	The user's or organizer's email address will be verified	The email with the verification link is delivered to the inbox, and the email address is verified after clicking the link	Pass
8*	Administrator manages categories	The administrator is logged in and views the category management section of the admin panel	Add, edit or delete a category	The category will be added, edited or deleted	All categories are shown in the category management section. A single category is successfully added, edited or deleted	Pass

Test report for the functional requirements

13.2 Non-Functional Test Protocol 29.05.2024

No.	Description	Precondition	Input	Expected Output	Actual Output	Pass/Fail
1	Collaborative: Required features are implemented	—	System actor uses the implemented features	All use cases are successfully executed	The use cases can successfully be executed via the implemented features on the user interface	Pass
2	Performance: Backend can handle up to 1000 requests per minute	—	Backend requests	The system keeps running without errors	No errors occur, and the system keeps running without getting into an error state	Pass
3	Response Time: All pages are loaded in under 200ms	The user has a stable internet connection	Pages are loaded	All pages are loaded faster than 200ms	All new pages are loaded in under 200ms	Pass
4	Responsiveness: Already tested in the functional requirements	—	—	—	—	Pass
5	Browser Compatibility: The product is compatible with the latest versions of Google Chrome, Mozilla Firefox and Apple Safari	The latest stable version of an aforementioned browser is used	Pages are loaded	Pages are loaded without errors	The pages are loaded without errors and are displayed correctly on each browser	Pass

Continued on next page

No.	Description	Precondition	Input	Expected Output	Actual Output	Pass/Fail
6	Availability: The system is available using the customer-provided domain	The system is deployed on the customer-provided domain	The domain is accessed through the web browser	The website is loaded and displayed	The web application is successfully available under the customer-provided domain	Pass
7	User Satisfaction: Users rate the UI	Test users are selected and can provide ratings	Test users navigate the page and provide ratings	3 out of 4 users rate the UI with a minimum score of 8/10	All 4 test users have rated the UI with a minimum score of 8/10	Pass
8	Scalability: Database is capable of handling up to 10'000 events and 1'000 users	Database is running	10'000 events and 1'000 users are created	The system keeps running without errors	The system stays in a stable state and does not crash or show any errors	Pass
9	Error Handling: Errors do not cause the system to crash	The system is running	Errors are triggered	The system keeps running without getting into an error state and displays a meaningful error message if necessary	When an error is triggered, the system does not crash and a meaningful error is displayed to the user	Pass
10	Security: All communication within and to the system is encrypted	The system is running in a production environment	The system is accessed	All communication is encrypted	All communication is encrypted and secure	Pass
11	Security: Input data is validated and sanitized	—	Invalid or malicious input is entered	The system does not crash, can not be exploited, and may display a meaningful error message	The input gets validated, sanitized and the system does not crash	Pass

Continued on next page

No.	Description	Precondition	Input	Expected Output	Actual Output	Pass/Fail
12	Data Privacy: Data protection regulations are met	—	User data is stored	Data is stored according to the data protection regulations	User data is stored in accordance with the data protection regulations	Pass
13	Password Security: User passwords are stored securely	The user is registered	User creates or changes password	The password is stored securely and not in plain text	The passwords of all accounts are stored securely as hashes only and not in plain text	Pass
14	User Data Isolation: A logged-in user can only access their own data	The user is logged in	User tries to access other users' data	The data can not be accessed	The data of other users can not be accessed	Pass
15	Modularity: The system is built in a modular way	—	New features are implemented	New features are implemented without the need to change or revamp existing features	New features can be implemented without changing existing features	Pass
16	Testing: API testing	The API testing tool is configured	Tests have been executed	All tests successfully pass	The API testing tool (Postman) does not show any errors while testing	Pass
17	Deployment: The system and its whole functionality is deployed	The system is deployed on DigitalOcean, configured with the customer-provided domain	The domain is accessed	The system is available, and all functional requirements are working	The system is successfully deployed, and the application is running on and reachable via DigitalOcean	Pass

Test specifications for the non-functional requirements

Notes

Test case #3 was set to 'Pass' even though not all pages of the application are loaded in the expected time. The reason for this is that the authentication pages developed during the 'Studienarbeit' are not loading in under 200ms. This is due to the hashing of the passwords, which is a time-consuming process

if done securely. The authentication pages are not the main focus of this project and therefore the test case is set to 'Pass'. Test case #4 was set to 'Pass' because the responsiveness of the application has already been tested in the functional requirements, where it passed. Test case #7 was tested with 4 end-users, where all of them were asked to rate the UI. The reports of their test process is documented in the section 13.3.

13.3 Reports from Test Users

This section contains the reports from the test users. Four independent test users have been asked to work through a questionnaire to test all the features of the application and report any issues they encounter. Additionally, they were asked to rate the user interface and provide some constructive feedback. To ensure the test users are not biased, they were not given any information about the application in advance. The names of the test users were blacked out to ensure their privacy.

What's up in RJ Testing / Questionnaire

Name: [redacted]

Date: 24.05.2024

Use the following link to access the web application: https://whatsup.rapperswil-jona.city/

Please use your own email addresses, as the system will send you emails! If you run out of email addresses to use, contact us.

Feel free to look around and check out everything. If you have any feedback or comments, write them into the feedback/comment section at the end of the questionnaire. Perform the actions as described in the following points. Carefully read through them first before you do anything. If you encounter any problems, use the lines underneath to elaborate.

1. Create an **organizer** account via the sign up. Go through the **onboarding process** by creating your first event using the **Create Event** button on step 2.

---

2. Check your email inbox, where a verification email should have arrived, and **verify your email address** using the link inside the email.

---

3. **Log out** from this new account and **use the provided admin credentials** to log in as administrator. Visit the **user management** of the admin panel and try to find your organizer account in the **unverified organizers** tab. **Verify** your organizer account using the respective toggle. You should now receive an email for your organizer account.

---

4. Additionally, create a **new admin** and try to log in with the credentials you created. Now, delete the **provided admin** using the **admins** tab of the user management.

---

5. Visit the **category management** of the admin panel and **create a new category**. Try to **edit** and **delete** this category but you should have at least **one** own category before you move on.

---

6. If you successfully verified your organizer in **step 3**, you now should be able to create public events. Try this by logging in with your organizer account and **create** a new event (or **edit** your already created event) via the **account page**. Add some **images** to your event by using the **image upload** field. Also choose the **category** you created in **step 5**. Now, look for your event on the different pages (home, overviews, search) and open your event to see your entered details, including the images.

---

7. Go back to the account page and visit the **profile editor**. Add your details including an **avatar** and a **banner** image. Try to find your organizer on the different pages (overview, search). Open the organizer **single page** and view your entered details including the avatar and banner image. Furthermore, your created event should be visible at the bottom of the page.

8. Now, access the website from your phone or tablet, or resize your browser window to a smaller screen size. Create a new **user** account, go through the **onboarding process**, and choose your preferences. You will also receive an email to verify your account email. **Do not** verify it (if you accidentally did, change your email to a different one via the account page).

9. Try all available pages (home, overviews, search and account) on the size of your device. After you have completed this, log out and log back in with **your created admin** and find your user on the **users** tab of the **user management** page. Your user should have an indicator for the **unverified** email. You can now verify the email of the user account and this indicator should **not** be visible anymore after a page reload.

10. Visit the **all organizers** tab of the admin panel's user management and **unverify** your organizer account. Now you should **not** be able to find your previously created **events** or **organizer** throughout the website.

11. Next, **log in** again with your **organizer** account. You should see an indicator on the account page that the account is **unverified**. Try viewing an event you created, it should have a notice at the top, letting you that the event is only viewable by you. If you do not see this, you are probably still verified. Finally, **delete** the account.

12. Try all **other** functionalities you come across on the website, for example: the filters on the overview and search pages, the input fields on the account page (name, email, password), etc.

At the very end, you can leave us some feedback. Please **fill out all** of them and help us improve the application.

13. From 1-10 (with 10 being the highest), how would you rate the UI?

9

14. Have you encountered any problems, bugs or weird behaviour throughout your exploration session?

15. Additional feedback or comments:

Did not realise that there is a public and private option for the event

Looks really good, i like the UI and responsiveness

Thanks for filling out this questionnaire and testing our web application!

What's up in RJ Testing / Questionnaire

Name: [redacted]

Date: 24.05.2024

Use the following link to access the web application: https://whatsup.rapperswil-jona.city/

Please use your own email addresses, as the system will send you emails! If you run out of email addresses to use, contact us.

Feel free to look around and check out everything. If you have any feedback or comments, write them into the feedback/comment section at the end of the questionnaire. Perform the actions as described in the following points. Carefully read through them first before you do anything. If you encounter any problems, use the lines underneath to elaborate.

1. Create an **organizer** account via the sign up. Go through the **onboarding process** by creating your first event using the **Create Event** button on step 2.

Closed tab of the onboarding process because of the verification E-Mail.

2. Check your email inbox, where a verification email should have arrived, and **verify your email address** using the link inside the email.

3. **Log out** from this new account and **use the provided admin credentials** to log in as administrator. Visit the **user management** of the admin panel and try to find your organizer account in the **unverified organizers** tab. **Verify** your organizer account using the respective toggle. You should now receive an email for your organizer account.

4. Additionally, create a **new admin** and try to log in with the credentials you created. Now, delete the **provided admin** using the **admins** tab of the user management.

5. Visit the **category management** of the admin panel and **create a new category**. Try to **edit** and **delete** this category but you should have at least **one** own category before you move on.

6. If you successfully verified your organizer in **step 3**, you now should be able to create public events. Try this by logging in with your organizer account and **create** a new event (or **edit** your already created event) via the **account page**. Add some **images** to your event by using the **image upload** field. Also choose the **category** you created in **step 5**. Now, look for your event on the different pages (home, overviews, search) and open your event to see your entered details, including the images.

7. Go back to the account page and visit the **profile editor**. Add your details including an **avatar** and a **banner** image. Try to find your organizer on the different pages (overview, search). Open the organizer **single page** and view your entered details including the avatar and banner image. Furthermore, your created event should be visible at the bottom of the page.

8. Now, access the website from your phone or tablet, or resize your browser window to a smaller screen size. Create a new **user** account, go through the **onboarding process**, and choose your preferences. You will also receive an email to verify your account email. **Do not** verify it (if you accidentally did, change your email to a different one via the account page).

9. Try all available pages (home, overviews, search and account) on the size of your device. After you have completed this, log out and log back in with **your created admin** and find your user on the **users** tab of the **user management** page. Your user should have an indicator for the **unverified** email. You can now verify the email of the user account and this indicator should **not** be visible anymore after a page reload.

10. Visit the **all organizers** tab of the admin panel's user management and **unverify** your organizer account. Now you should **not** be able to find your previously created **events** or **organizer** throughout the website.

11. Next, **log in** again with your **organizer** account. You should see an indicator on the account page that the account is **unverified**. Try viewing an event you created, it should have a notice at the top, letting you that the event is only viewable by you. If you do not see this, you are probably still verified. Finally, **delete** the account.

12. Try all **other** functionalities you come across on the website, for example: the filters on the overview and search pages, the input fields on the account page (name, email, password), etc.

At the very end, you can leave us some feedback. Please **fill out all** of them and help us improve the application.

13. From 1-10 (with 10 being the highest), how would you rate the UI?

9/10

14. Have you encountered any problems, bugs or weird behaviour throughout your exploration session?

No, I have not.

15. Additional feedback or comments:

I like the design when you resize the window so it fits into the respective size.

The website looks very professional and it feels very intuitive to use.

Thanks for filling out this questionnaire and testing our web application!

What's up in RJ Testing / Questionnaire

Name: [redacted]

Date: 27.05.2024

Use the following link to access the web application: https://whatsup.rapperswil-jona.city/

Please use your own email addresses, as the system will send you emails! If you run out of email addresses to use, contact us.

Feel free to look around and check out everything. If you have any feedback or comments, write them into the feedback/comment section at the end of the questionnaire. Perform the actions as described in the following points. Carefully read through them first before you do anything. If you encounter any problems, use the lines underneath to elaborate.

1. Create an **organizer** account via the sign up. Go through the **onboarding process** by creating your first event using the **Create Event** button on step 2.

---

2. Check your email inbox, where a verification email should have arrived, and **verify your email address** using the link inside the email.

---

3. **Log out** from this new account and **use the provided admin credentials** to log in as administrator. Visit the **user management** of the admin panel and try to find your organizer account in the **unverified organizers** tab. **Verify** your organizer account using the respective toggle. You should now receive an email for your organizer account.

---

4. Additionally, create a **new admin** and try to log in with the credentials you created. Now, delete the **provided admin** using the **admins** tab of the user management.

---

5. Visit the **category management** of the admin panel and **create a new category**. Try to **edit** and **delete** this category but you should have at least **one** own category before you move on.

---

6. If you successfully verified your organizer in **step 3**, you now should be able to create public events. Try this by logging in with your organizer account and **create** a new event (or **edit** your already created event) via the **account page**. Add some **images** to your event by using the **image upload** field. Also choose the **category** you created in **step 5**. Now, look for your event on the different pages (home, overviews, search) and open your event to see your entered details, including the images.

---

7. Go back to the account page and visit the **profile editor**. Add your details including an **avatar** and a **banner** image. Try to find your organizer on the different pages (overview, search). Open the organizer **single page** and view your entered details including the avatar and banner image. Furthermore, your created event should be visible at the bottom of the page.

---

8. Now, access the website from your phone or tablet, or resize your browser window to a smaller screen size. Create a new **user** account, go through the **onboarding process**, and choose your preferences. You will also receive an email to verify your account email. **Do not** verify it (if you accidentally did, change your email to a different one via the account page).

---

9. Try all available pages (home, overviews, search and account) on the size of your device. After you have completed this, log out and log back in with **your created admin** and find your user on the **users** tab of the **user management** page. Your user should have an indicator for the **unverified** email. You can now verify the email of the user account and this indicator should **not** be visible anymore after a page reload.

---

10. Visit the **all organizers** tab of the admin panel's user management and **unverify** your organizer account. Now you should **not** be able to find your previously created **events** or **organizer** throughout the website.

---

11. Next, **log in** again with your **organizer** account. You should see an indicator on the account page that the account is **unverified**. Try viewing an event you created, it should have a notice at the top, letting you that the event is only viewable by you. If you do not see this, you are probably still verified. Finally, **delete** the account.

---

12. Try all **other** functionalities you come across on the website, for example: the filters on the overview and search pages, the input fields on the account page (name, email, password), etc.

At the very end, you can leave us some feedback. Please **fill out all** of them and help us improve the application.

13. From 1-10 (with 10 being the highest), how would you rate the UI?

10

---

14. Have you encountered any problems, bugs or weird behaviour throughout your exploration session?

length of character is limited to 255, but the application struggles to properly display a very long name

---

15. Additional feedback or comments:

after verifying return to onboarding

Thanks for filling out this questionnaire and testing our web application!

What's up in RJ Testing / Questionnaire

Name: [redacted]

Date: 30.05.2024

Use the following link to access the web application: https://whatsup.rapperswil-jona.city/

Please use your own email addresses, as the system will send you emails! If you run out of email addresses to use, contact us.

Feel free to look around and check out everything. If you have any feedback or comments, write them into the feedback/comment section at the end of the questionnaire. Perform the actions as described in the following points. Carefully read through them first before you do anything. If you encounter any problems, use the lines underneath to elaborate.

1. Create an **organizer** account via the sign up. Go through the **onboarding process** by creating your first event using the **Create Event** button on step 2.

sign up as organizer can get missed quickly

2. Check your email inbox, where a verification email should have arrived, and **verify your email address** using the link inside the email.

3. **Log out** from this new account and **use the provided admin credentials** to log in as administrator. Visit the **user management** of the admin panel and try to find your organizer account in the **unverified organizers** tab. **Verify** your organizer account using the respective toggle. You should now receive an email for your organizer account.

4. Additionally, create a **new admin** and try to log in with the credentials you created. Now, delete the **provided admin** using the **admins** tab of the user management.

5. Visit the **category management** of the admin panel and **create a new category**. Try to **edit** and **delete** this category but you should have at least **one** own category before you move on.

6. If you successfully verified your organizer in **step 3**, you now should be able to create public events. Try this by logging in with your organizer account and **create** a new event (or **edit** your already created event) via the **account page**. Add some **images** to your event by using the **image upload** field. Also choose the **category** you created in **step 5**. Now, look for your event on the different pages (home, overviews, search) and open your event to see your entered details, including the images.

did not see the image upload when creating the event from the onboarding,
a little too many categories for a dropdown

7. Go back to the account page and visit the **profile editor**. Add your details including an **avatar** and a **banner** image. Try to find your organizer on the different pages (overview, search). Open the organizer **single page** and view your entered details including the avatar and banner image. Furthermore, your created event should be visible at the bottom of the page.

8. Now, access the website from your phone or tablet, or resize your browser window to a smaller screen size. Create a new **user** account, go through the **onboarding process**, and choose your preferences. You will also receive an email to verify your account email. **Do not** verify it (if you accidentally did, change your email to a different one via the account page).

9. Try all available pages (home, overviews, search and account) on the size of your device. After you have completed this, log out and log back in with **your created admin** and find your user on the **users** tab of the **user management** page. Your user should have an indicator for the **unverified** email. You can now verify the email of the user account and this indicator should **not** be visible anymore after a page reload.

10. Visit the **all organizers** tab of the admin panel's user management and **unverify** your organizer account. Now you should **not** be able to find your previously created **events** or **organizer** throughout the website.

11. Next, **log in** again with your **organizer** account. You should see an indicator on the account page that the account is **unverified**. Try viewing an event you created, it should have a notice at the top, letting you that the event is only viewable by you. If you do not see this, you are probably still verified. Finally, **delete** the account.

To the organizer its not instantly clear when you get unverified

12. Try all **other** functionalities you come across on the website, for example: the filters on the overview and search pages, the input fields on the account page (name, email, password), etc.

At the very end, you can leave us some feedback. Please **fill out all** of them and help us improve the application.

13. From 1-10 (with 10 being the highest), how would you rate the UI?

8

14. Have you encountered any problems, bugs or weird behaviour throughout your exploration session?

Sometimes the loading looks like it froze for a few seconds and is not spinning

15. Additional feedback or comments:

The color palette is a bit to bright for me, i would choose a little more opaque colors, but thats preference. The category management page should have a search function.

Thanks for filling out this questionnaire and testing our web application!

Notes

Test user #1 reported that they did not realize the existence of a public/private option for the event. This seems to be a problem with the button design of the visibility option. As this is the only test user that reported this issue, it is not considered a critical issue that needs to be fixed immediately. However, this should be kept in mind for future improvements.

Test users #2 and #3 reported that they closed the tab of the onboarding process after freshly signing up. When they received the verification email in their inbox, which was almost instant, they clicked on the link. As this link opens a new tab on the home page, the onboarding process is not visible any more. Nevertheless, after the next login of the user the onboarding process is opened again, if they have not completed it yet. This should still be improved in the future, e.g. by redirecting the user to the onboarding process again after clicking the verification link.

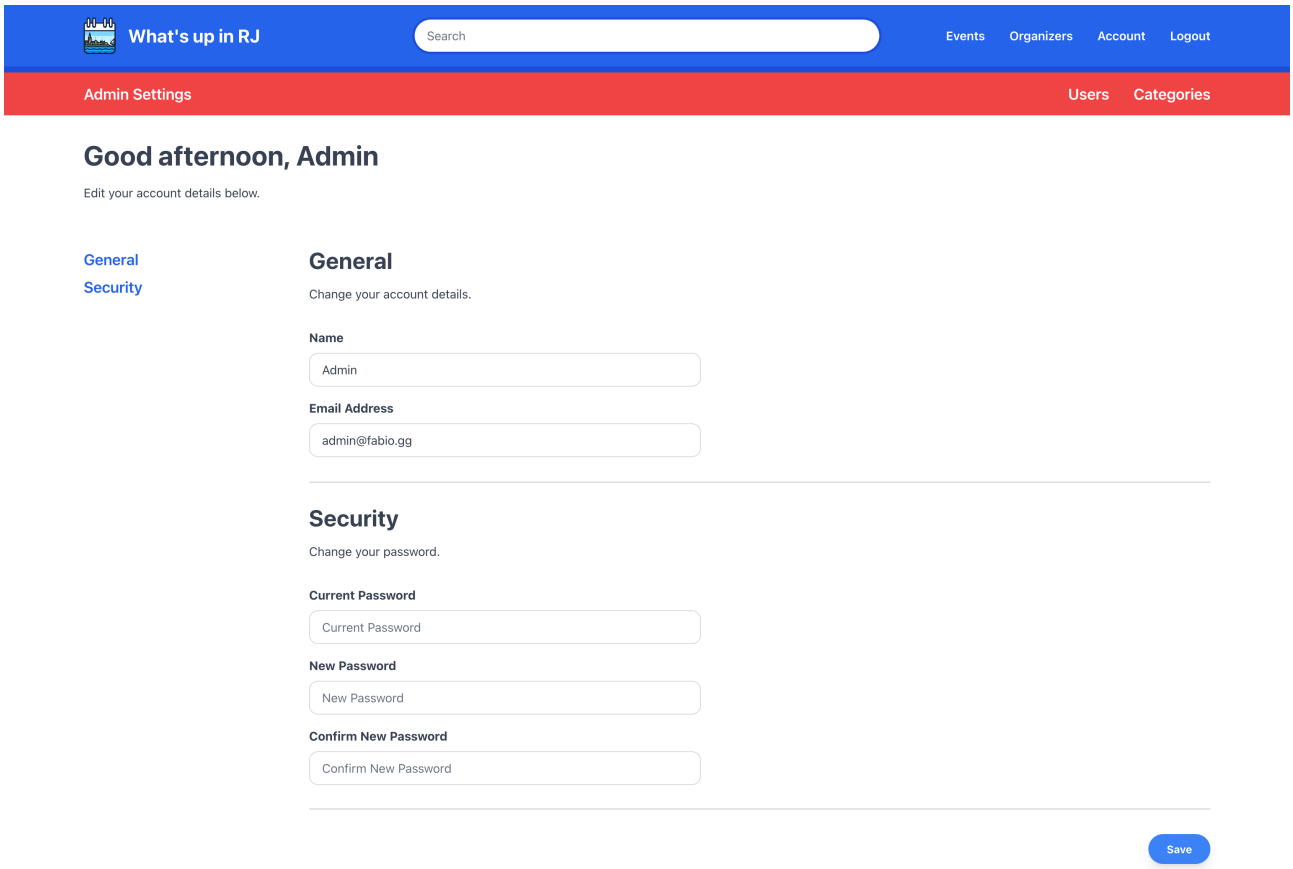
Test user #3 reported that the application struggles to display very long names that use the maximum allowed characters. The application does allow very long names and also displays them, but the design is not perfect for very long names. Especially on smaller screens very long names may not appear with a perfect design. In future iterations, the design for some elements should be improved to handle very long names better, or limit the input length to have shorter names. Since content on the website is only generated by organizers, which need to be verified first in order to publish any content, this should not be an issue though. It can be assumed that verified organizers are not using any text lengths that would lead to unwanted overlapping of content.

Test user #4 reported several feedback points. Firstly, they mentioned that the 'Sign up as organizer' option can be easily missed. This is a design question of the sign-up page, which should be reevaluated in future mock-ups. Secondly, they reported that they did not see the image upload field in the onboarding process. This is true, as the image upload is not currently part of the onboarding process. Even though this event is not public and can be edited later by adding images, it should be considered to add the image upload field to the onboarding process in the future. Additionally, they reported that there are too many categories for a dropdown and its getting hard to find the right one. As the design of the categories in the onboarding process may already get improved in the future, the whole design of the category selection on all pages could be reevaluated. Furthermore, they reported that it is not instantly clear to an organizer when they get unverified. This can be improved in a future iteration by adding a notification on the front page, or by sending an email to the organizer. As overall bugs, they reported that the loading animation of the spinner sometimes looks frozen and does not spin. This issue may be related to the browser or the system of the test user, as it was not reported by any other test user, nor was it observed during the development process. While this is the only report of this issue, it is not considered a critical issue but should be kept in mind for future fixes. Finally, they reported that the category management in the admin panel should have a search function. This was not planned in the scope of this project, but has already been considered for future improvements in section 10.3.

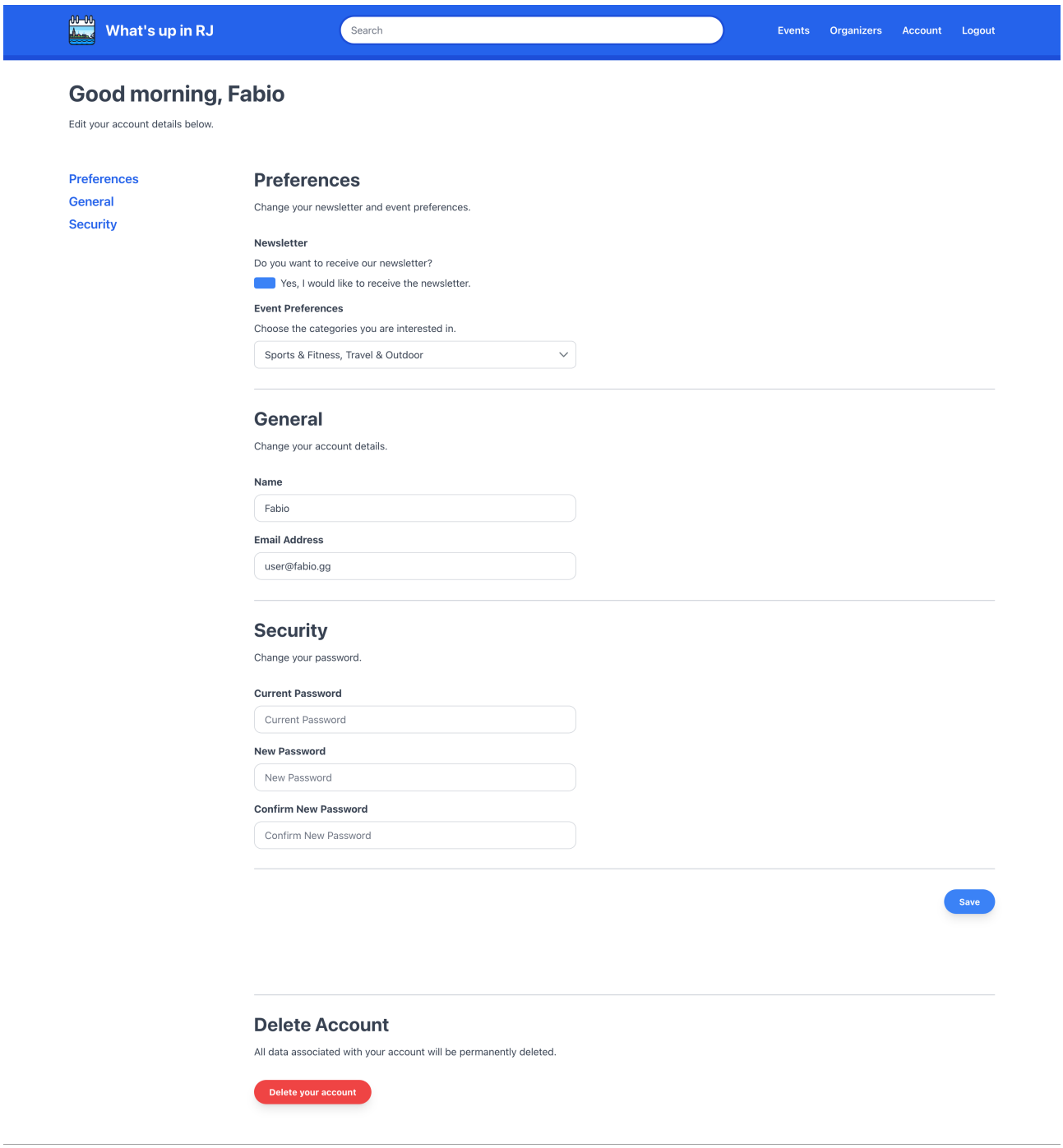
Chapter 14

Application Screenshots

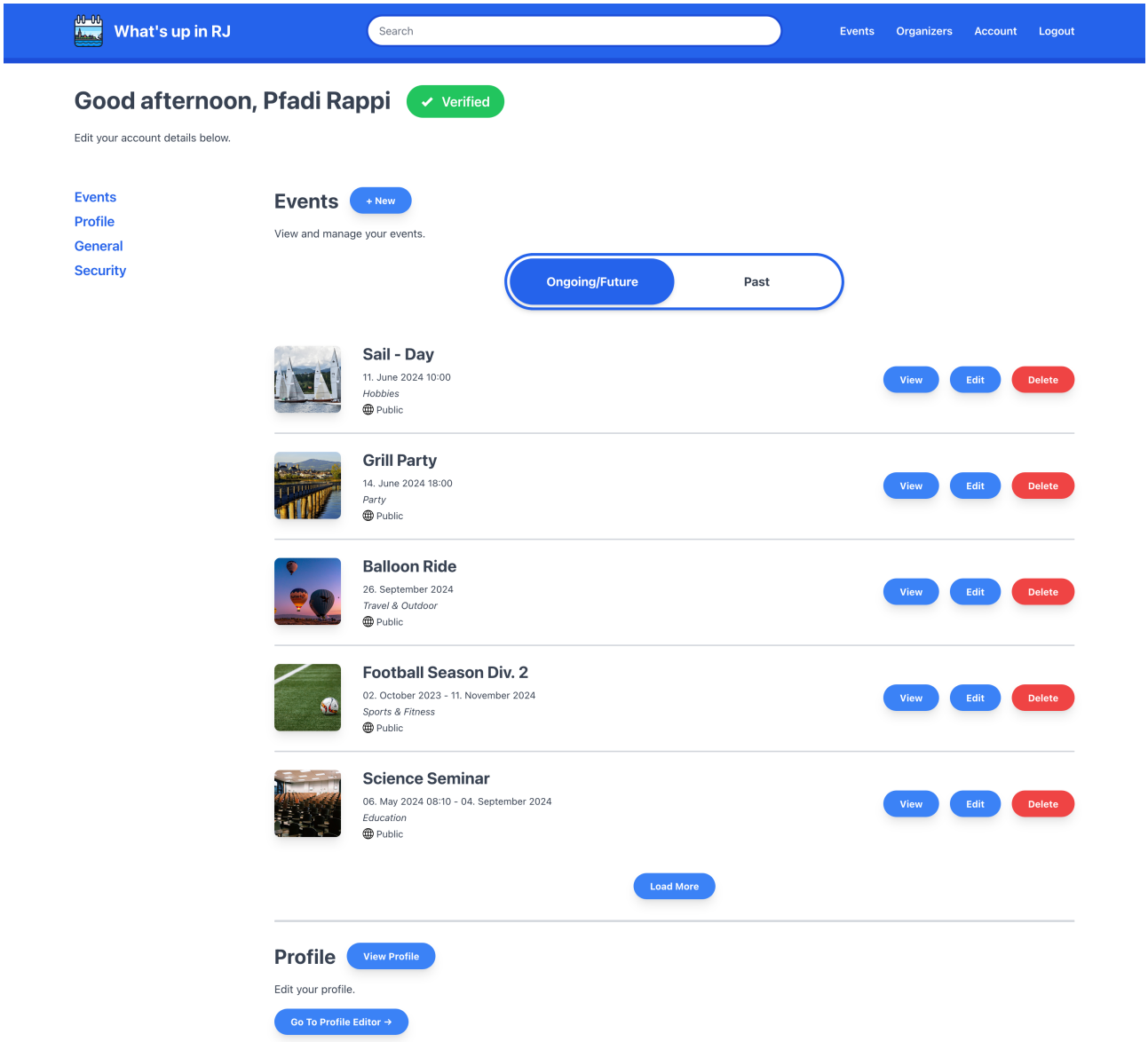
14.1 Desktop



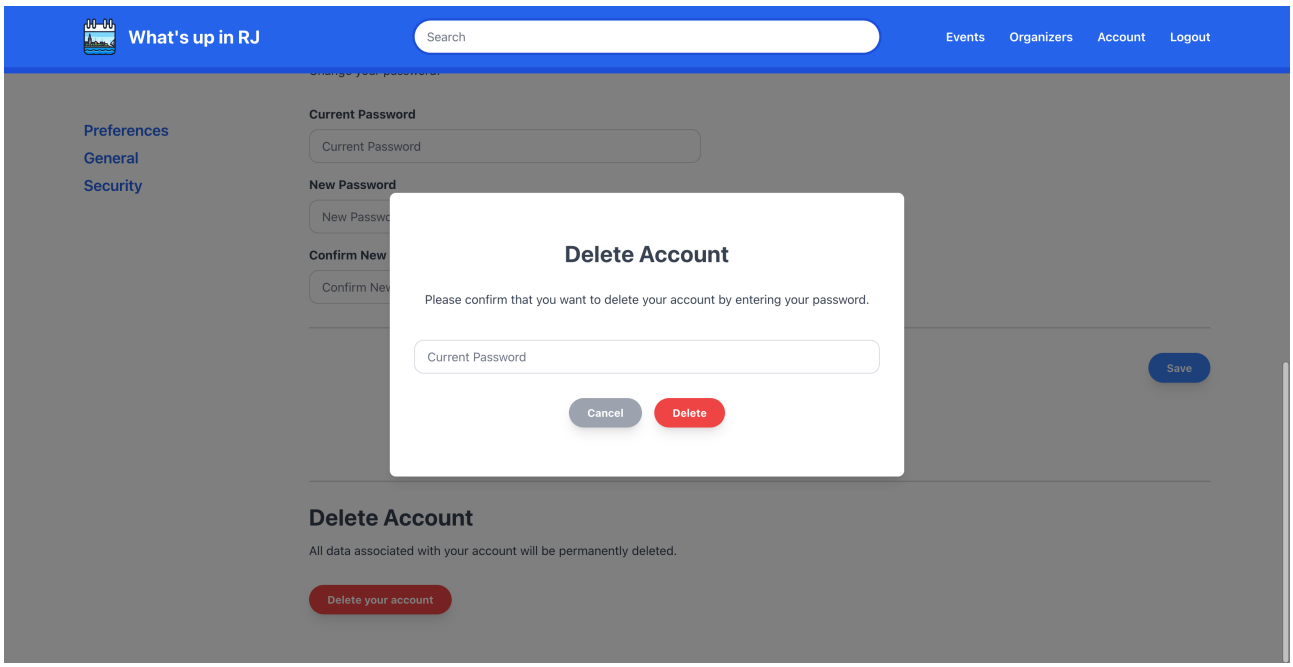
Desktop: Account page of an administrator



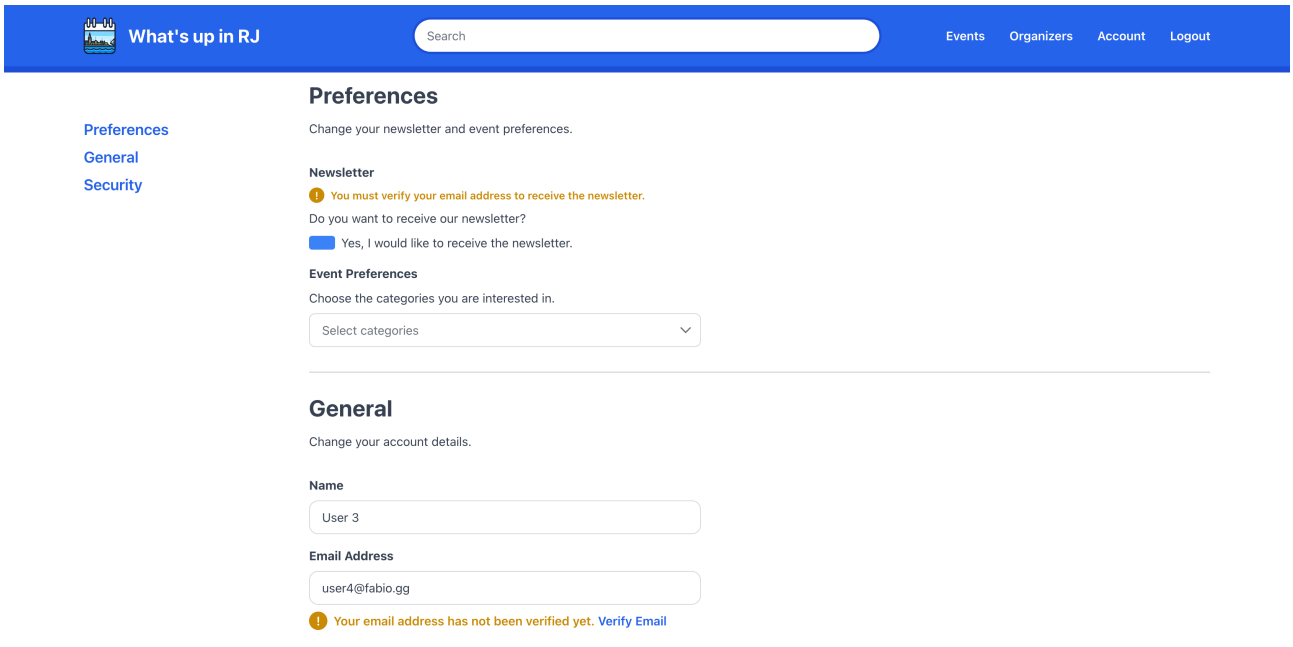
Desktop: Account page of a user



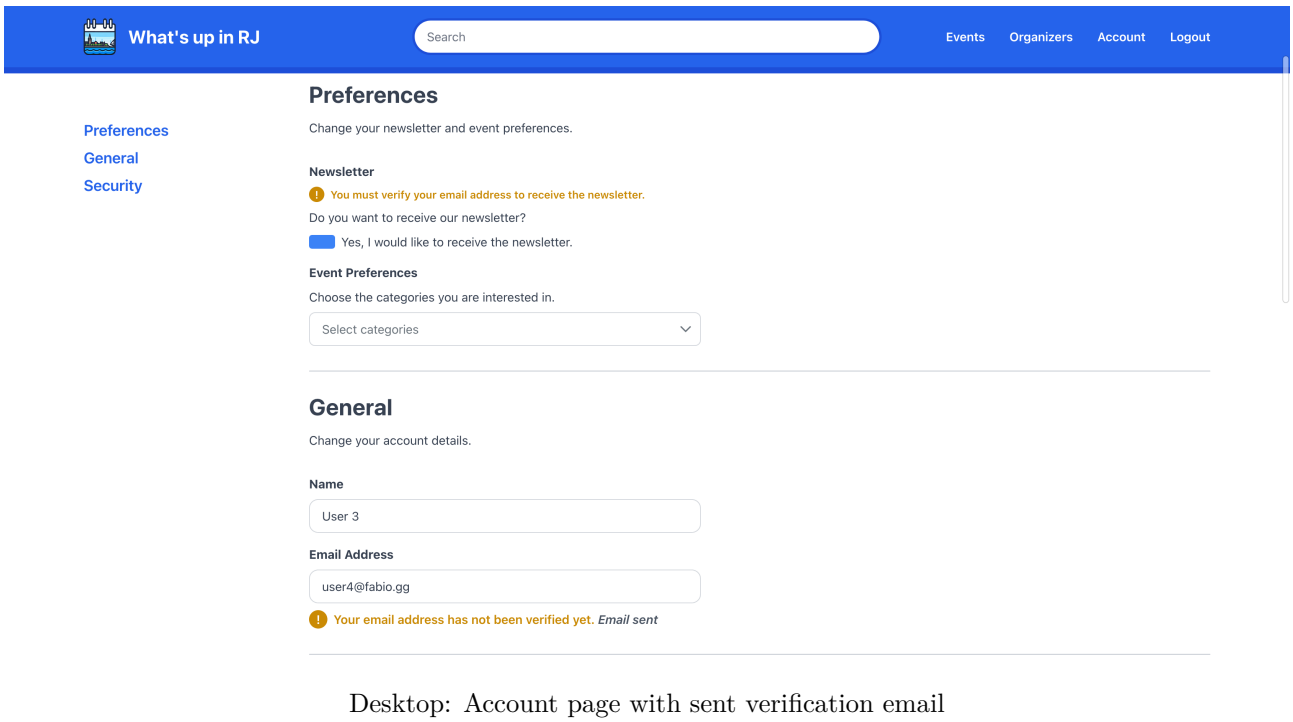
Desktop: Account page of an organizer



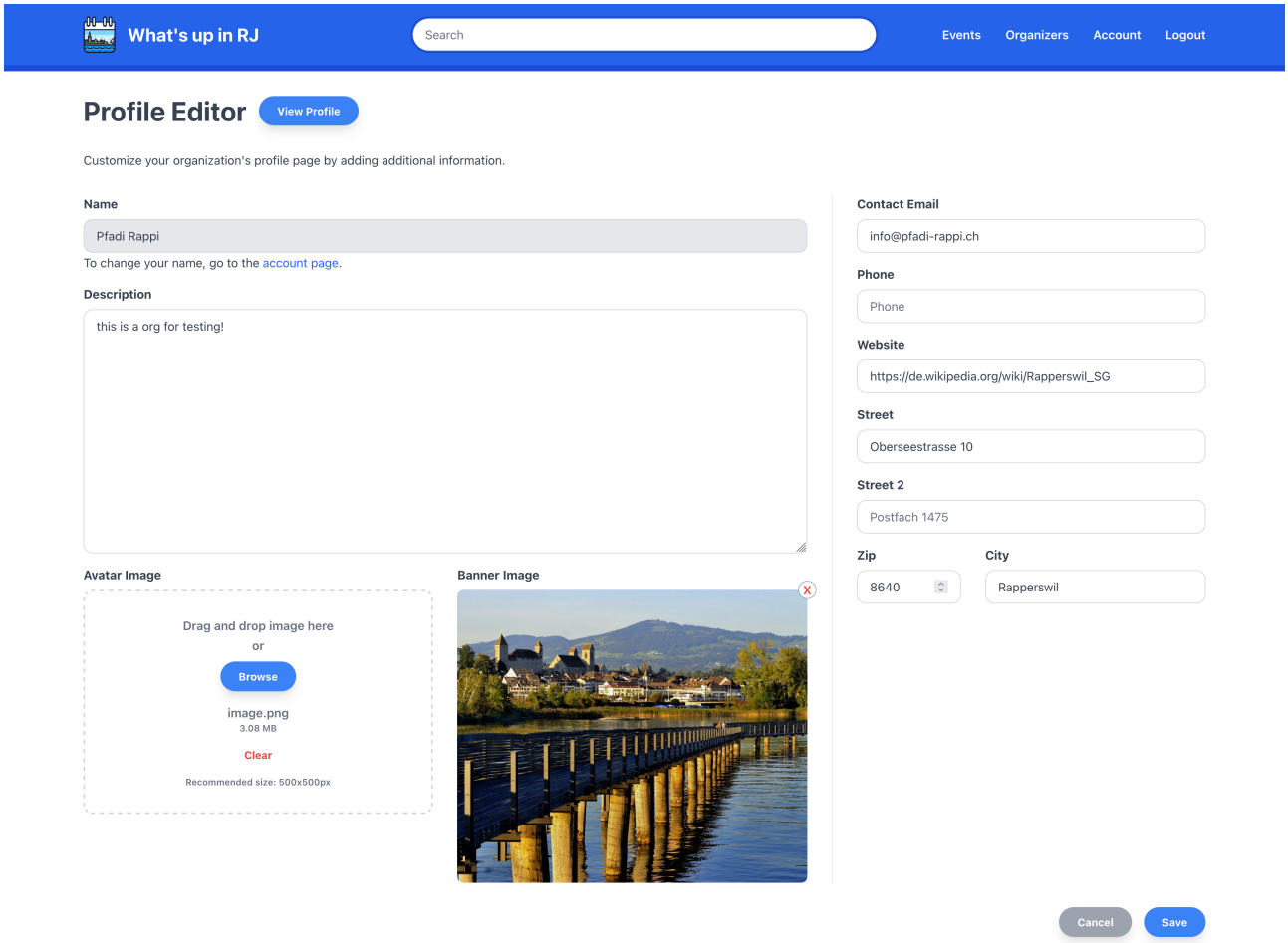
Desktop: Delete account dialogue



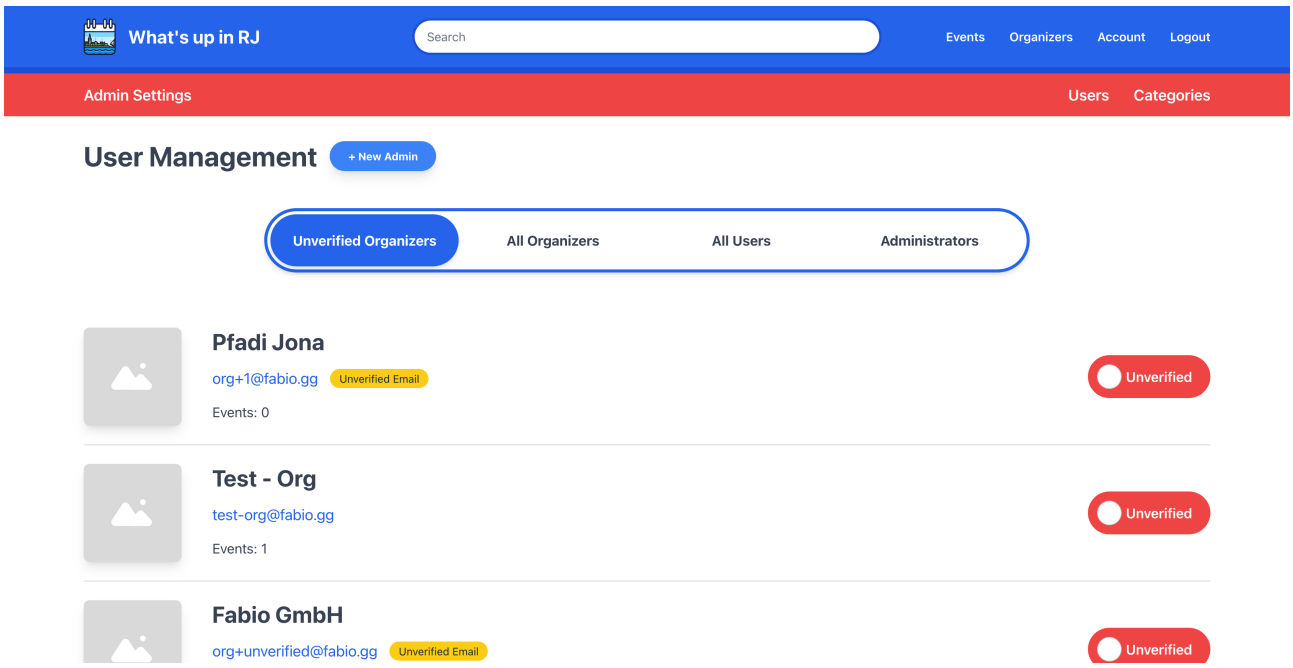
Desktop: Account page with unverified email



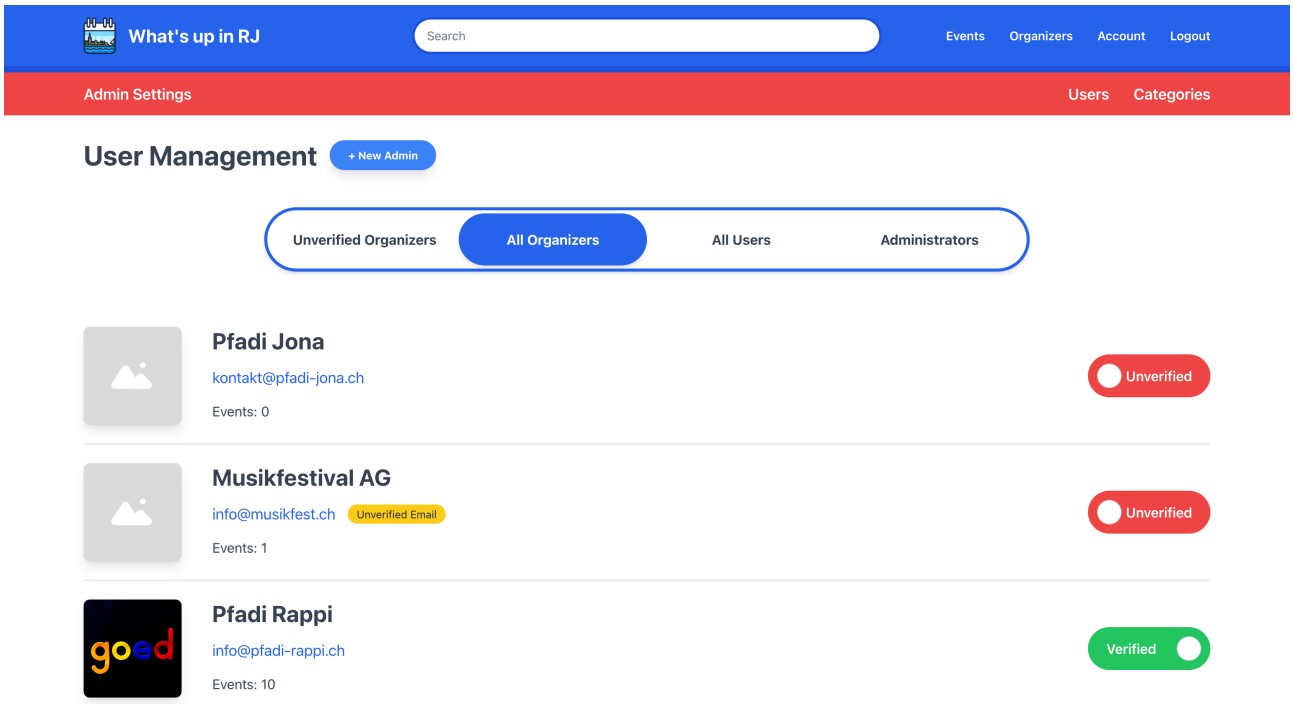
Desktop: Account page with sent verification email



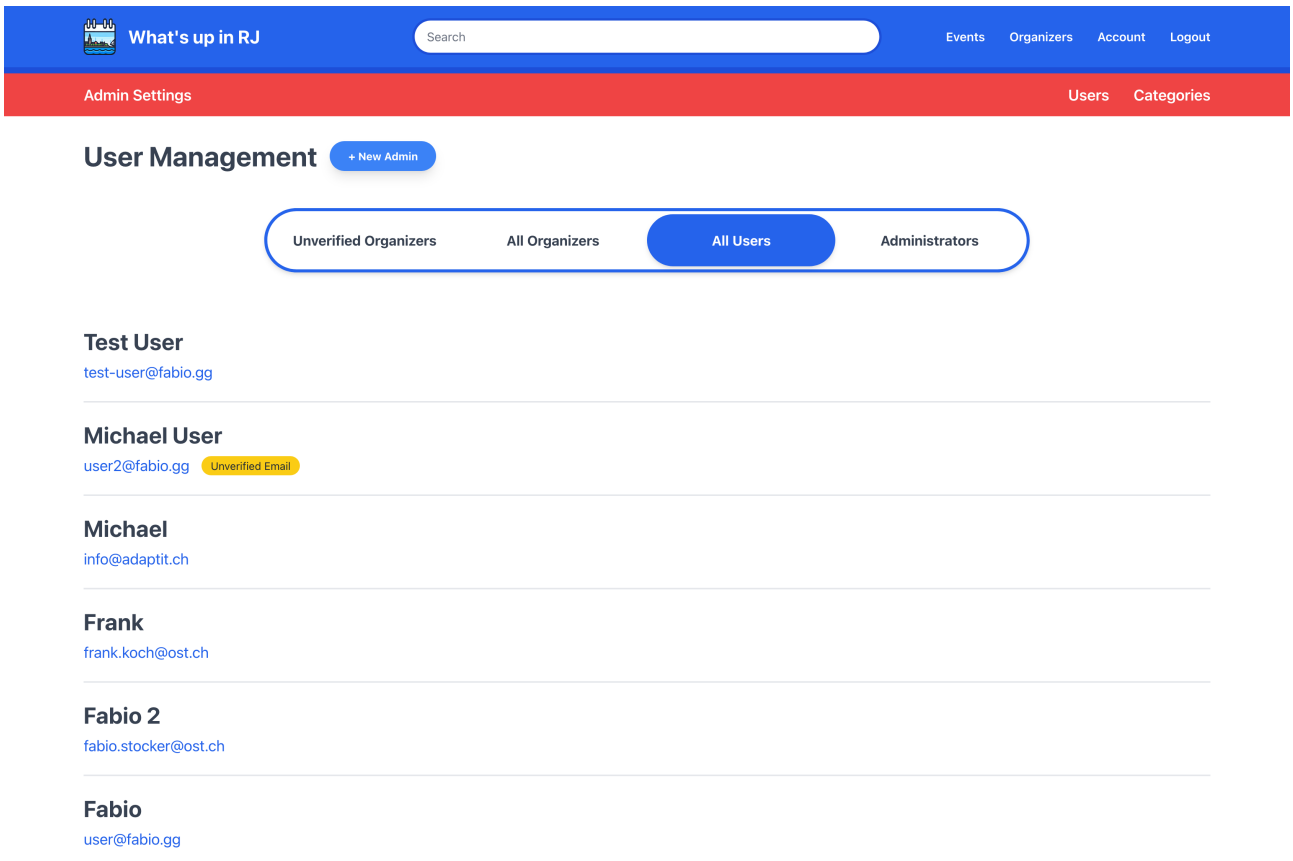
Desktop: Profile editor



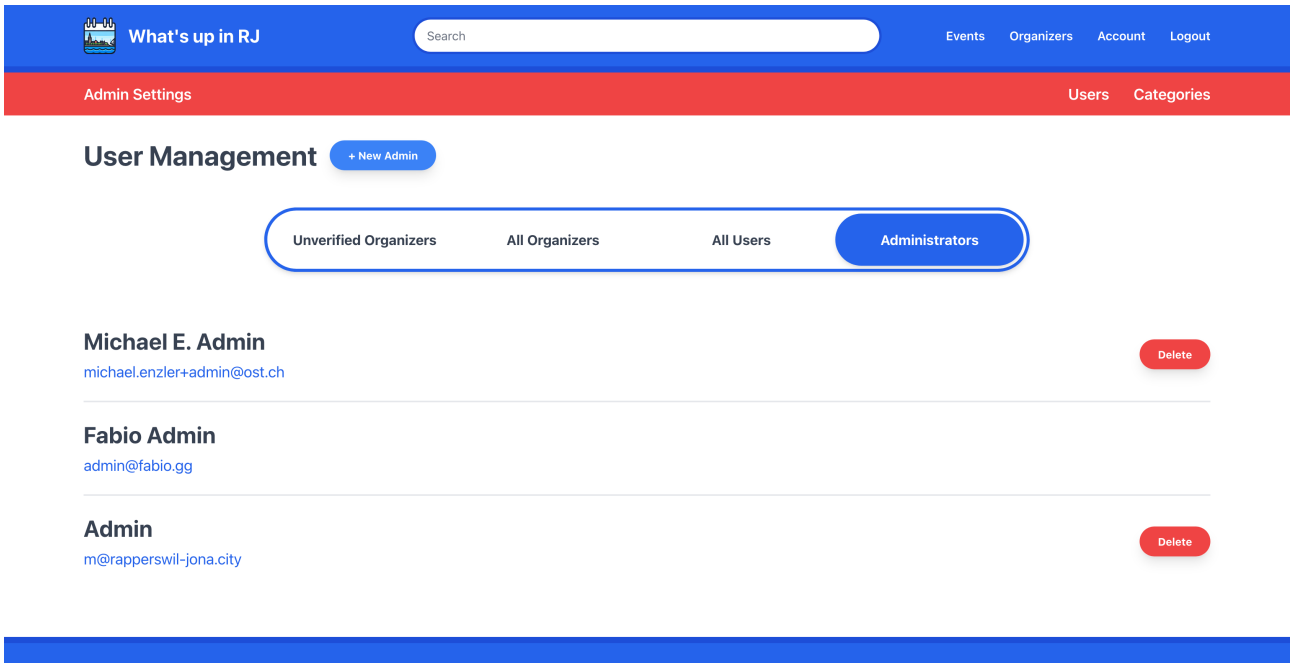
Desktop: Unverified organizers in the admin panel



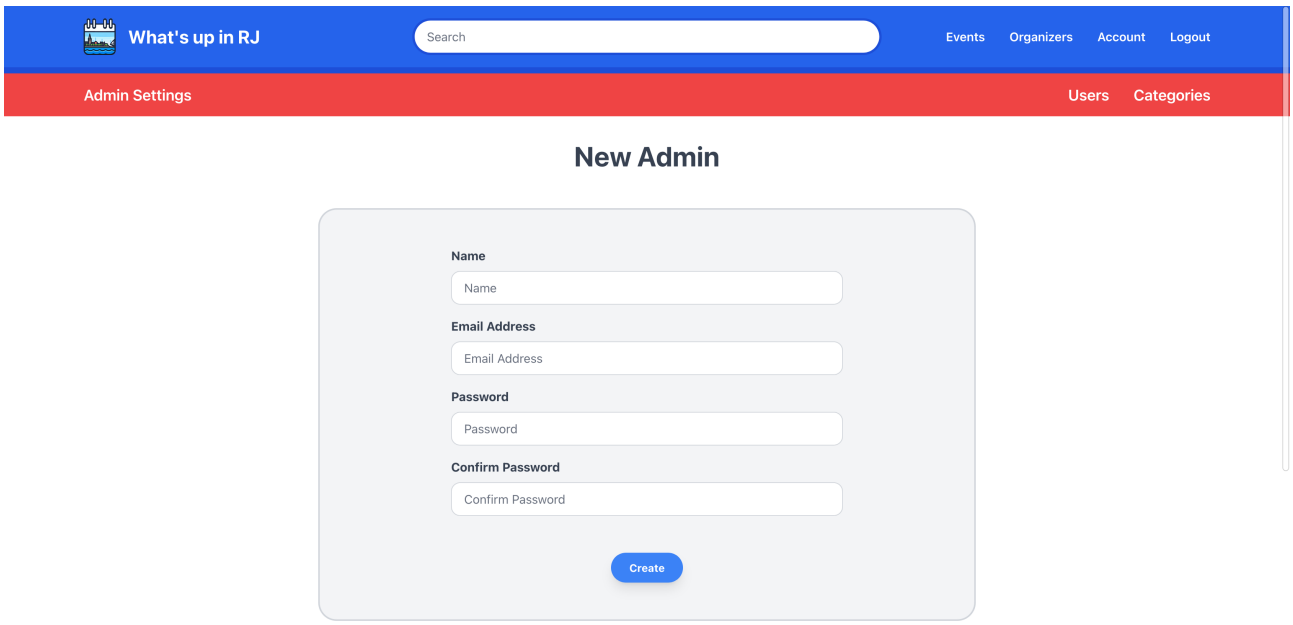
Desktop: All organizers in the admin panel



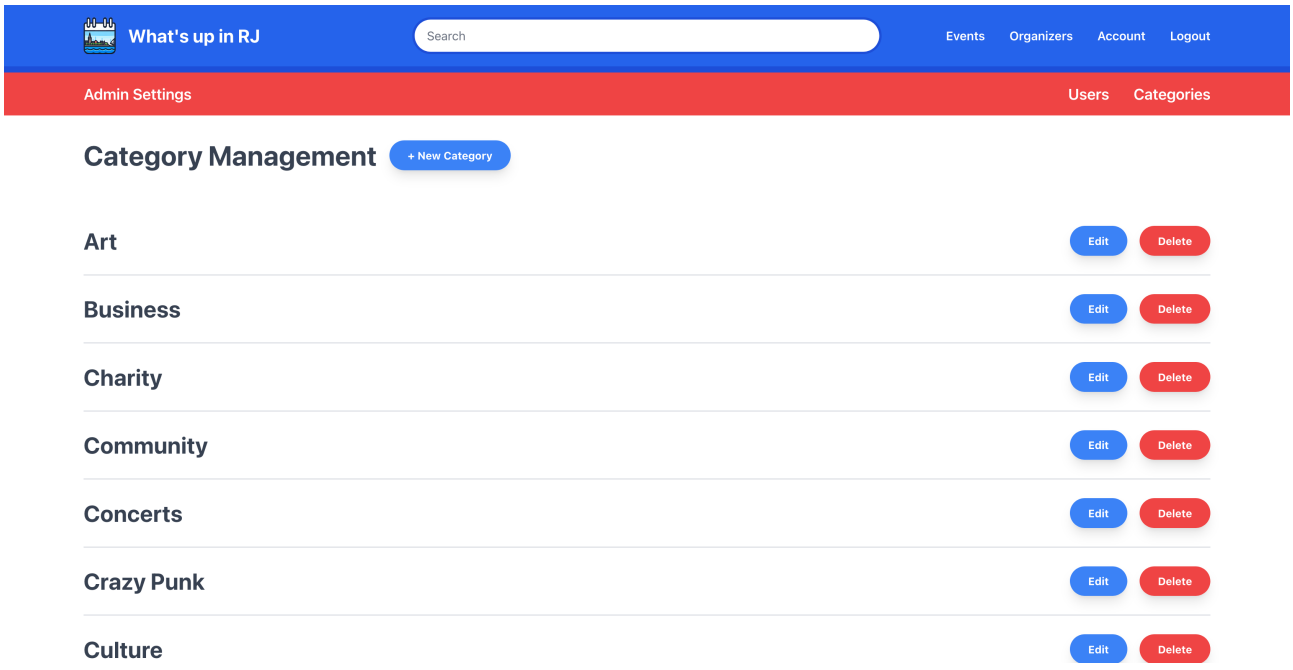
Desktop: All users in the admin panel



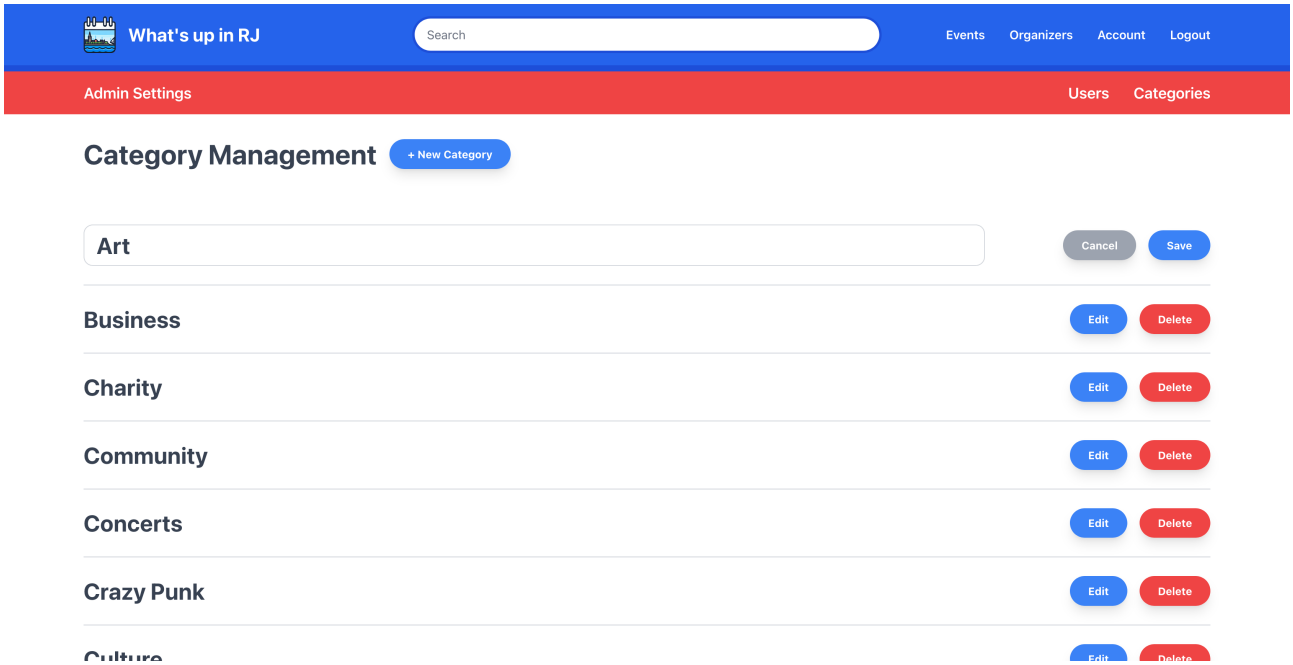
Desktop: Admins in the admin panel



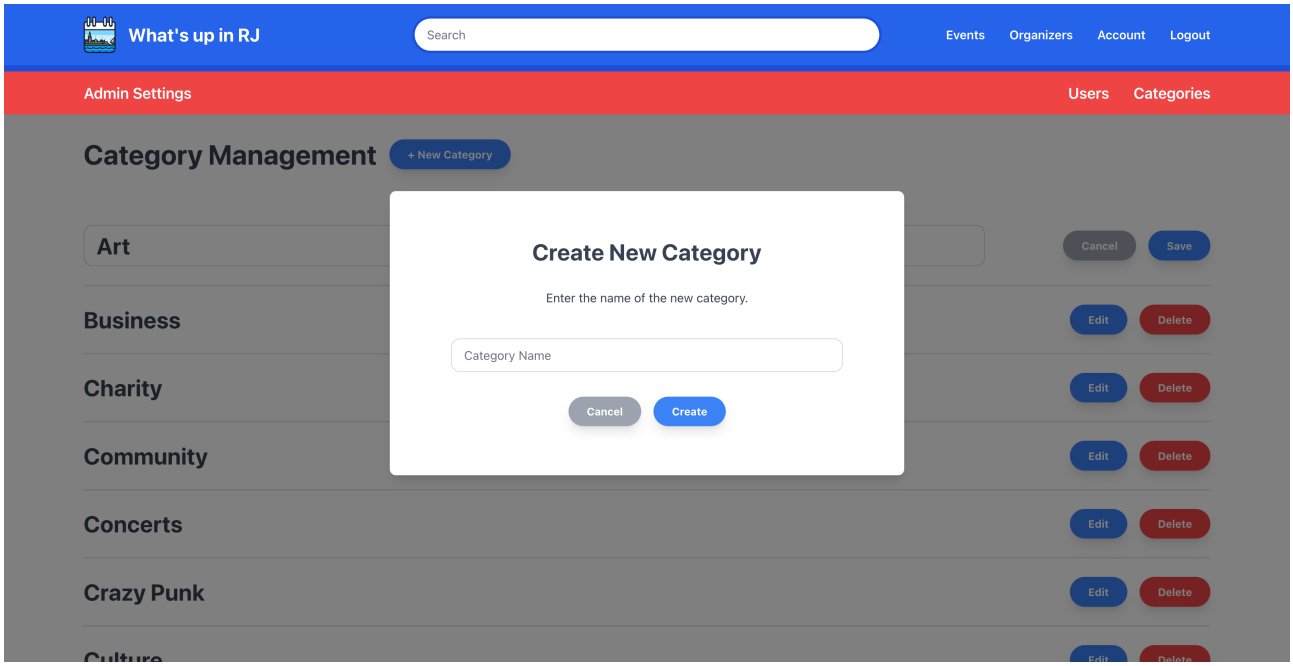
Desktop: New admin page in the admin panel



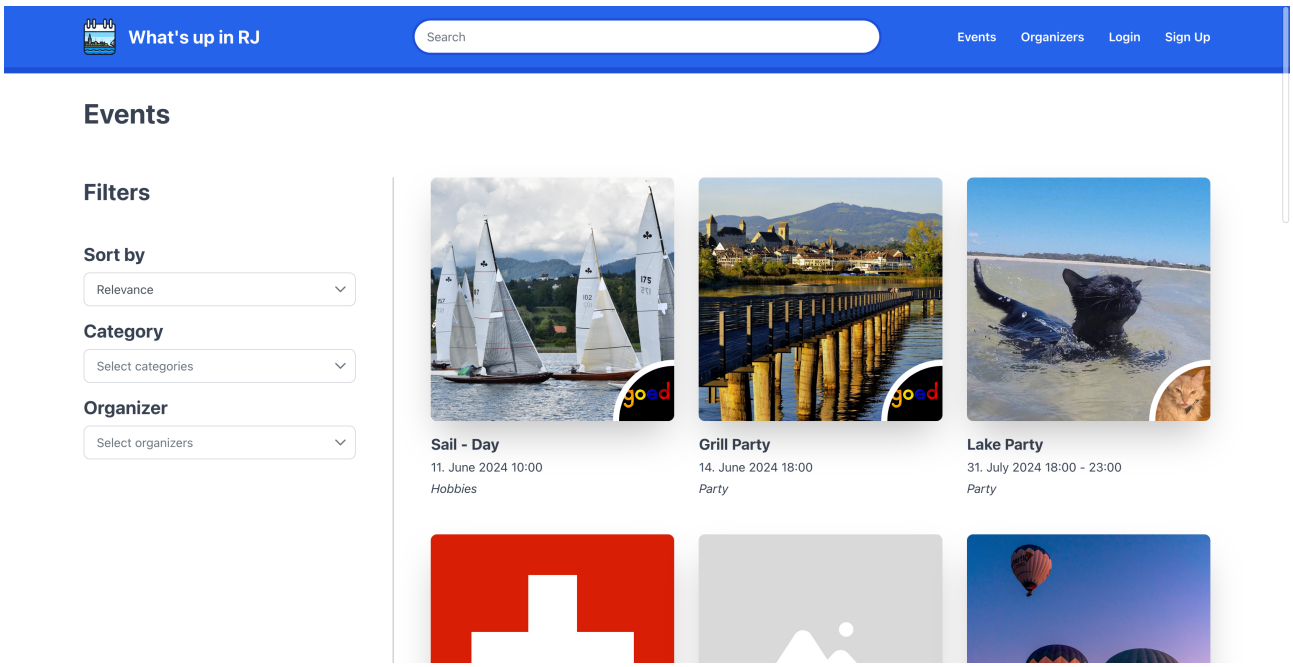
Desktop: Category management in the admin panel



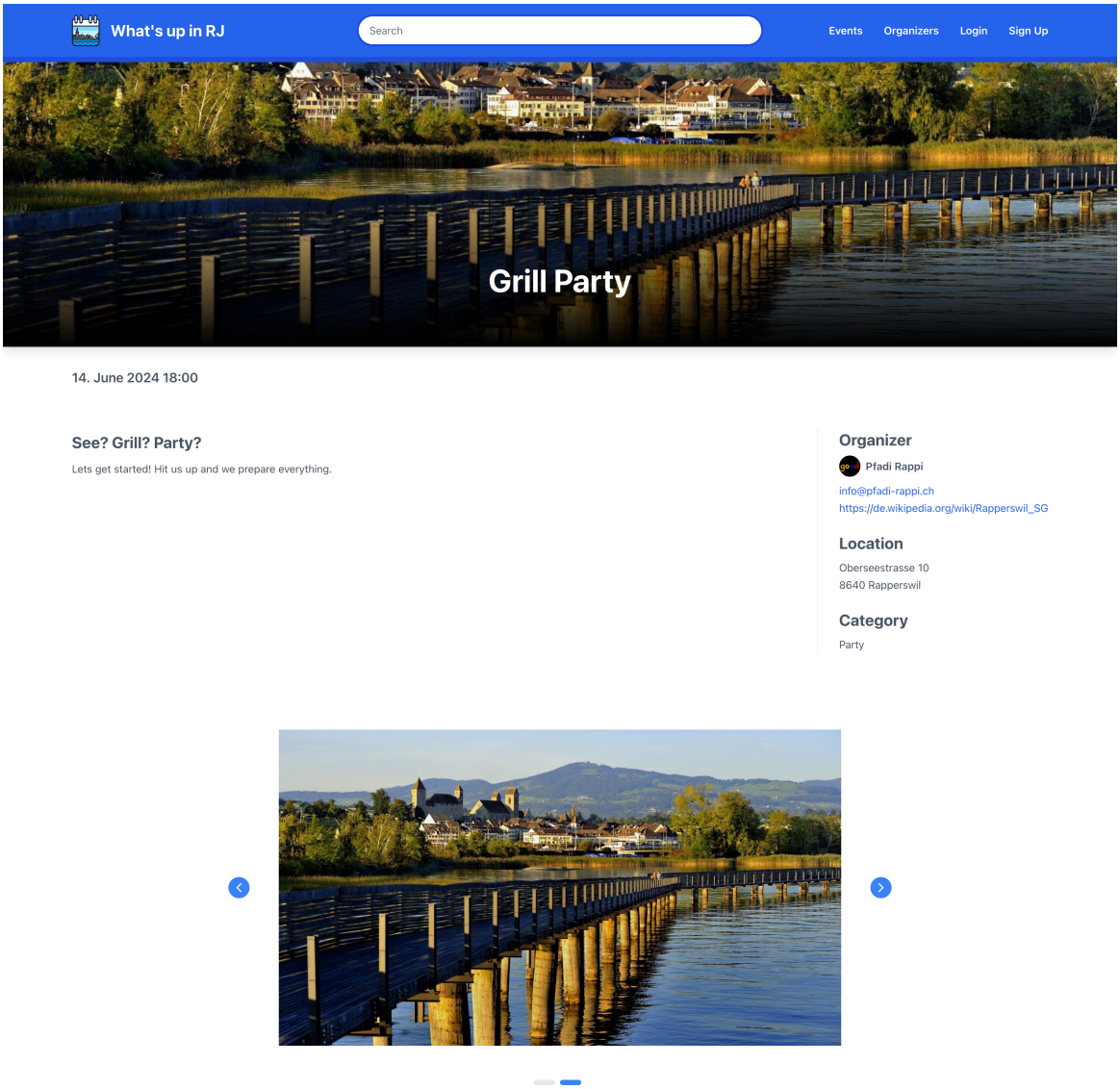
Desktop: Edit category in the admin panel



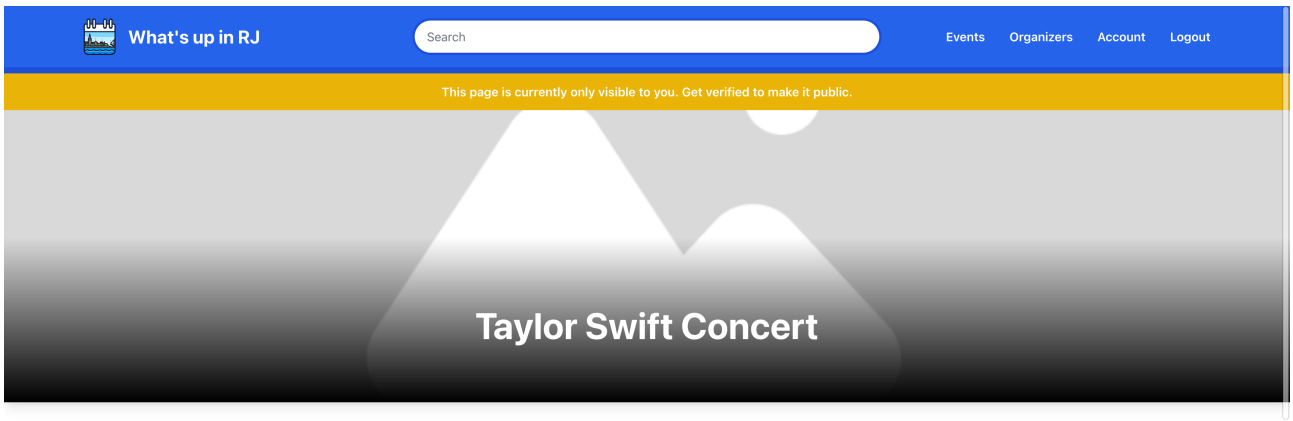
Desktop: Create category dialogue in the admin panel



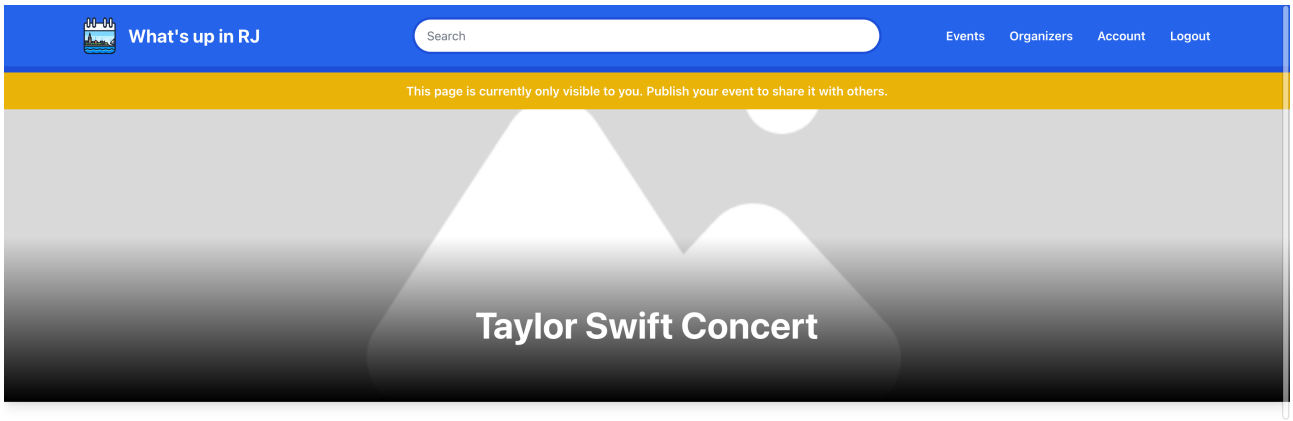
Desktop: Event overview



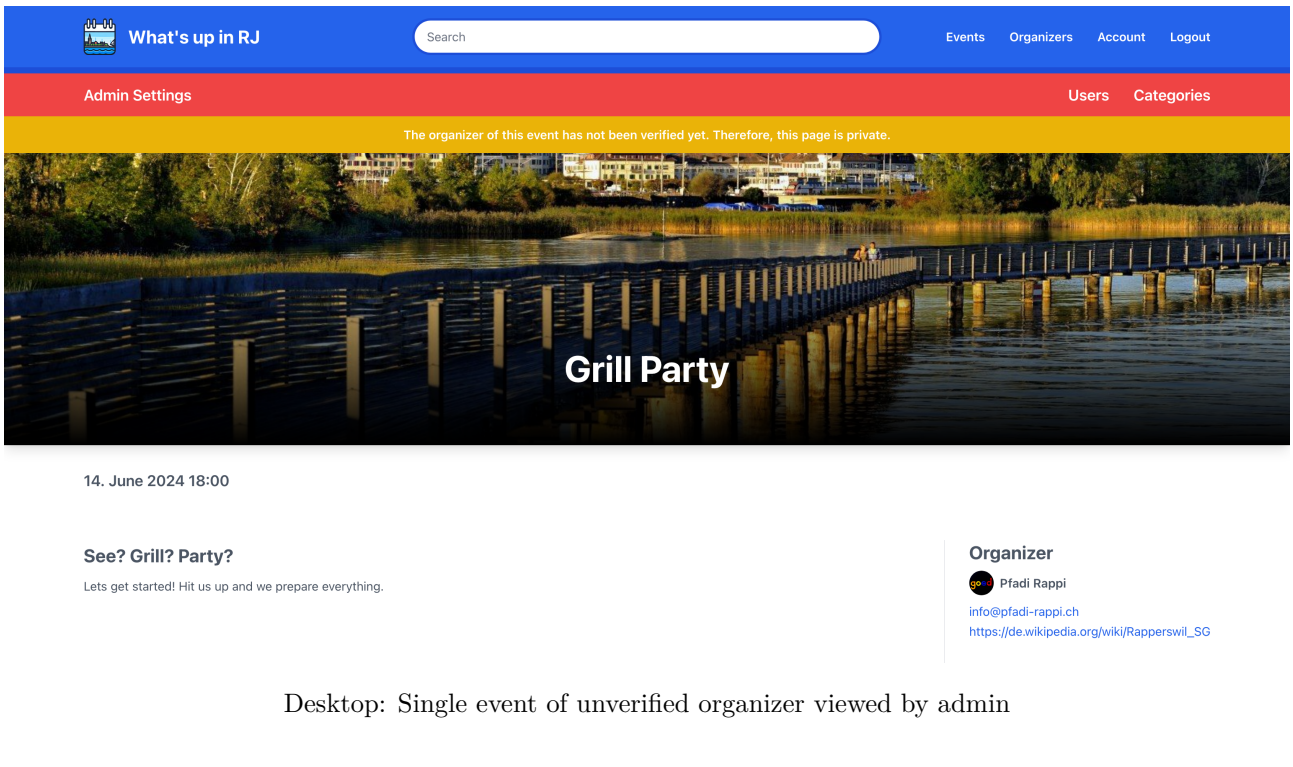
Desktop: Single event



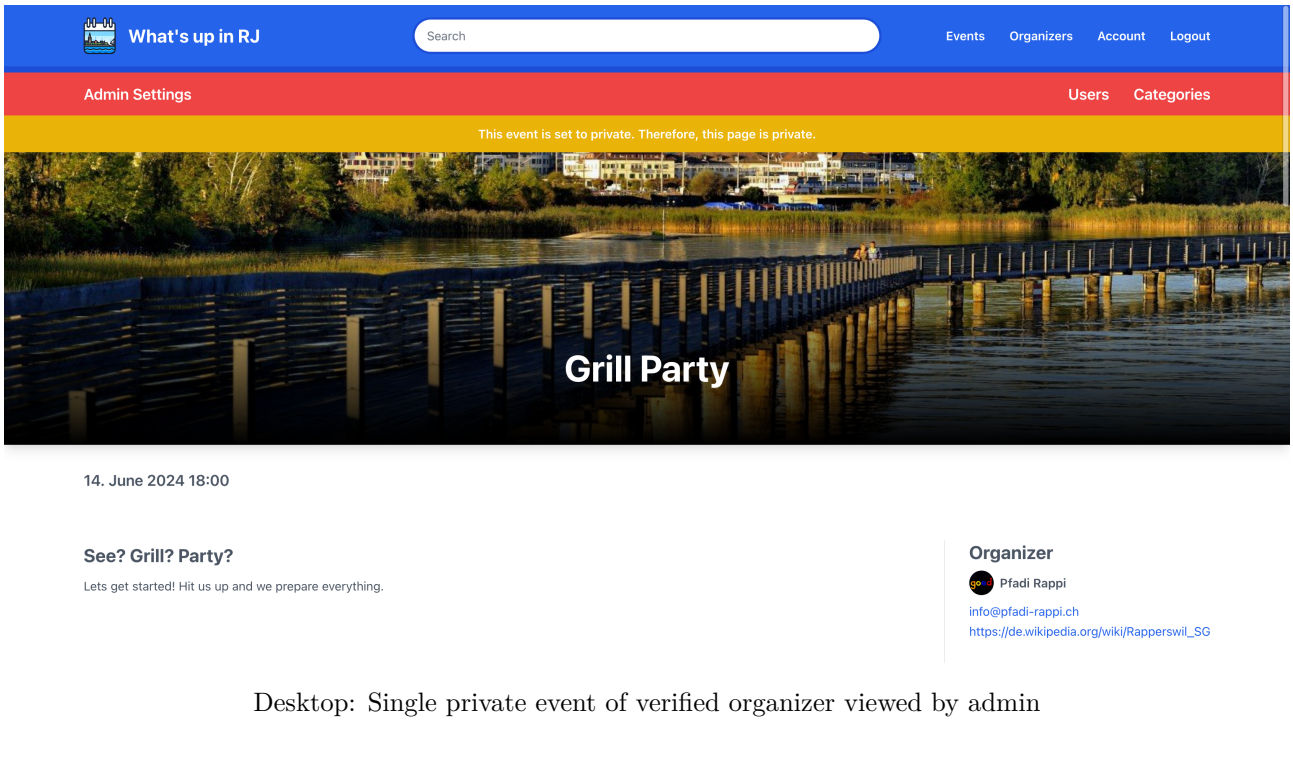
Desktop: Single event of unverified organizer



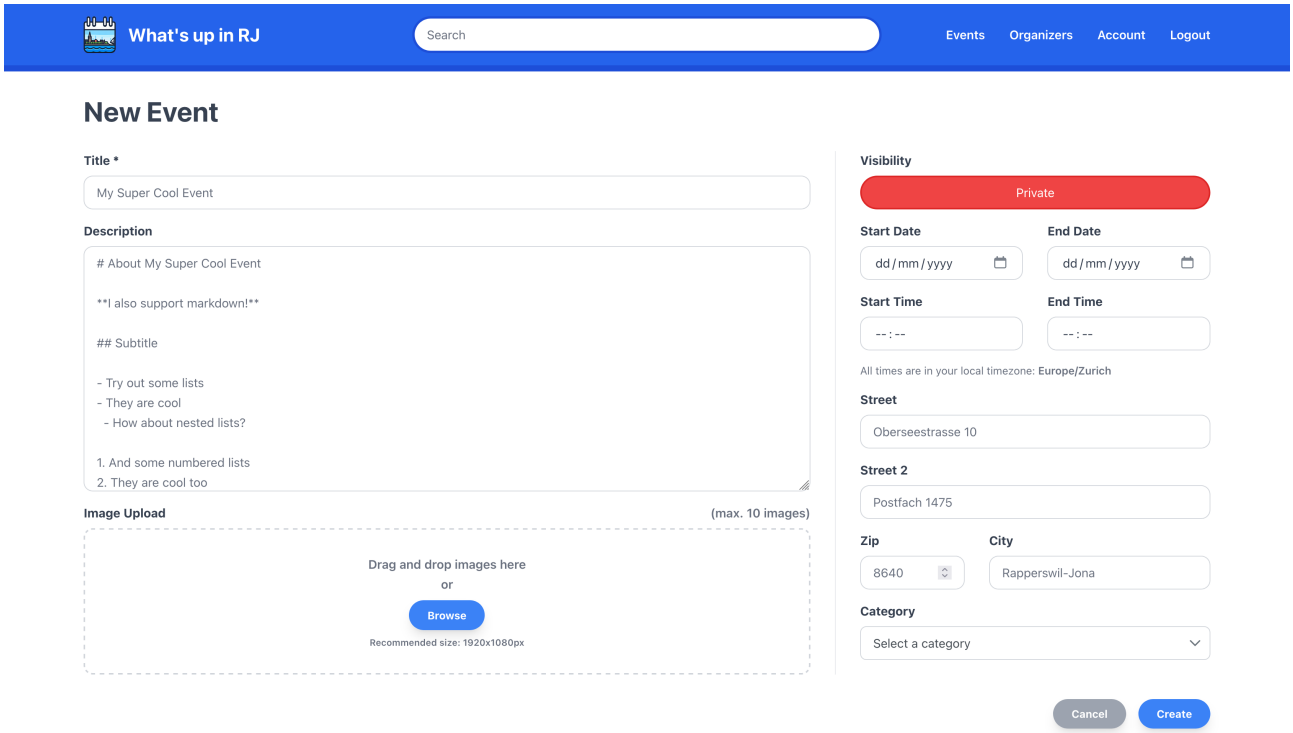
Desktop: Single private event of verified organizer



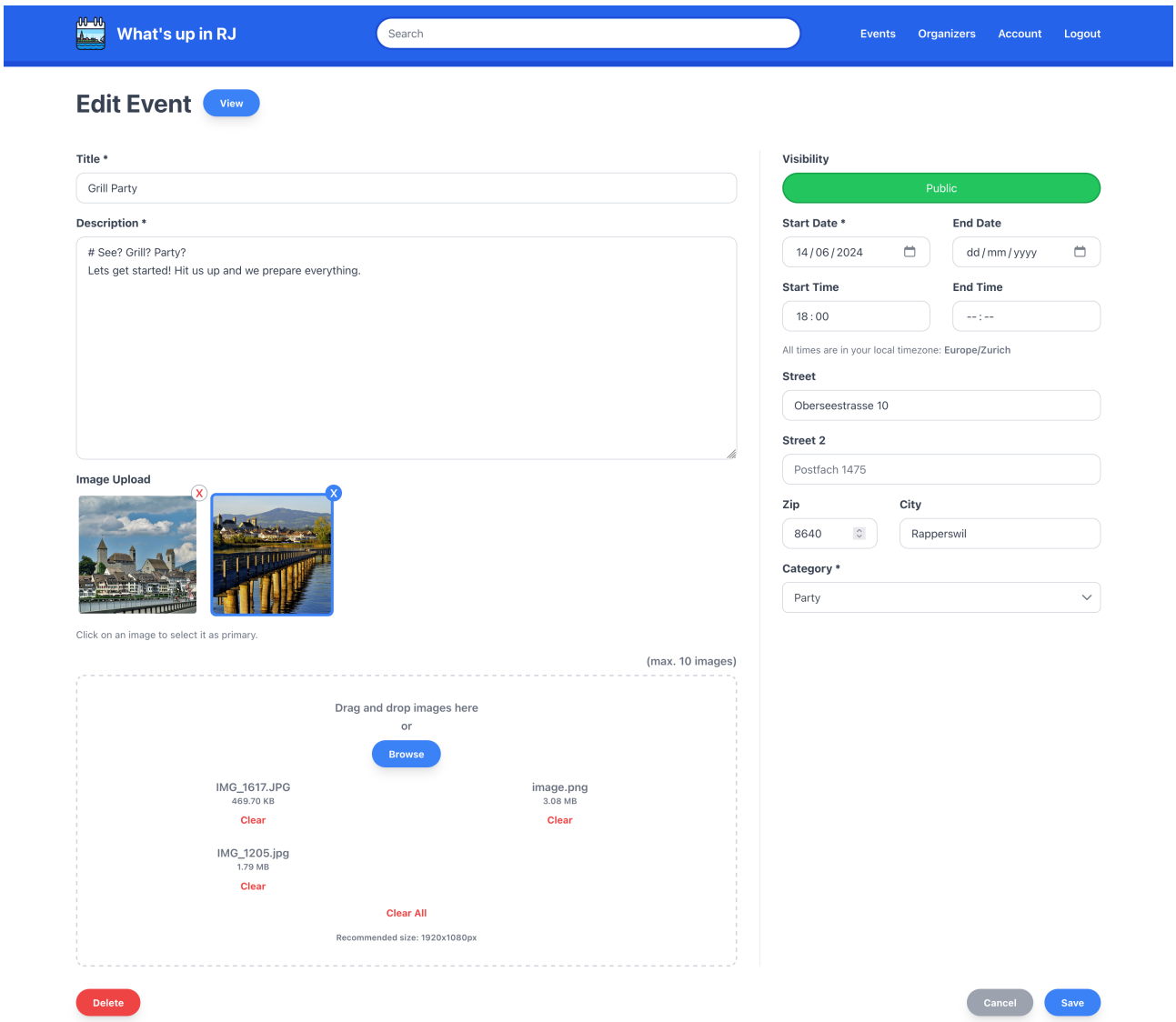
Desktop: Single event of unverified organizer viewed by admin



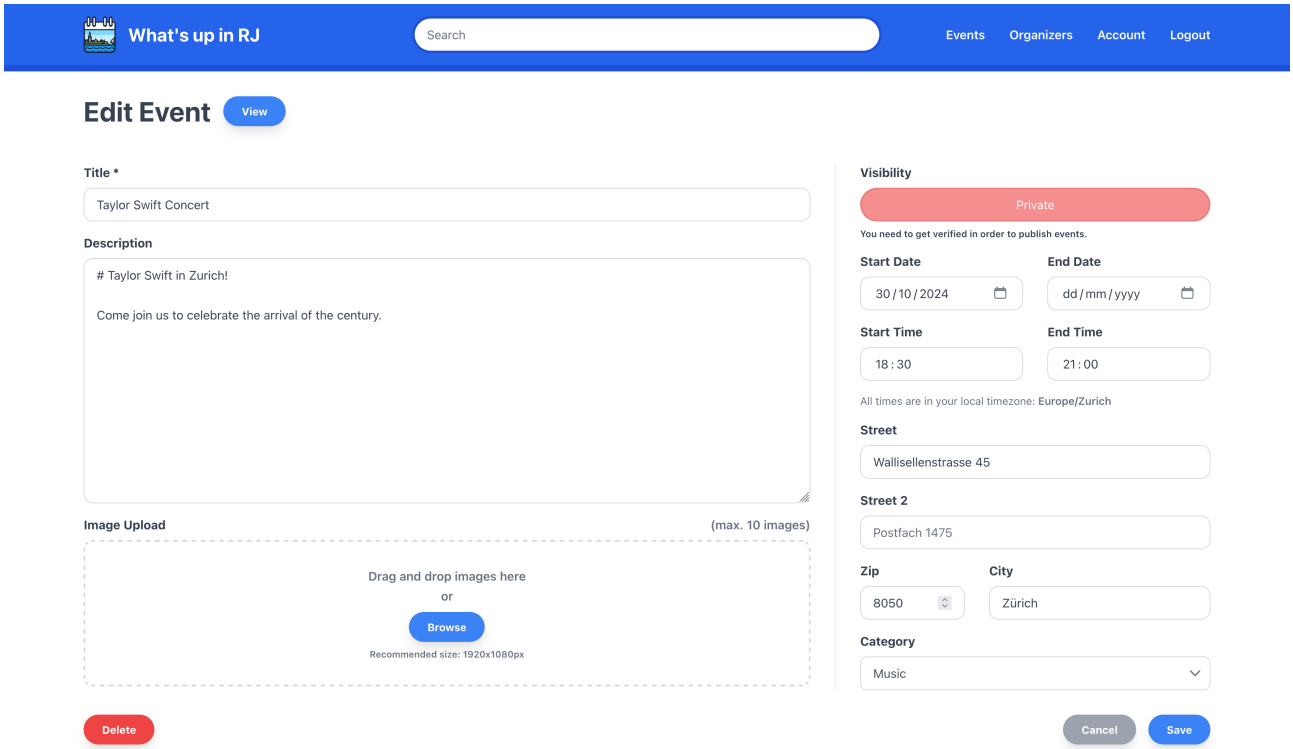
Desktop: Single private event of verified organizer viewed by admin



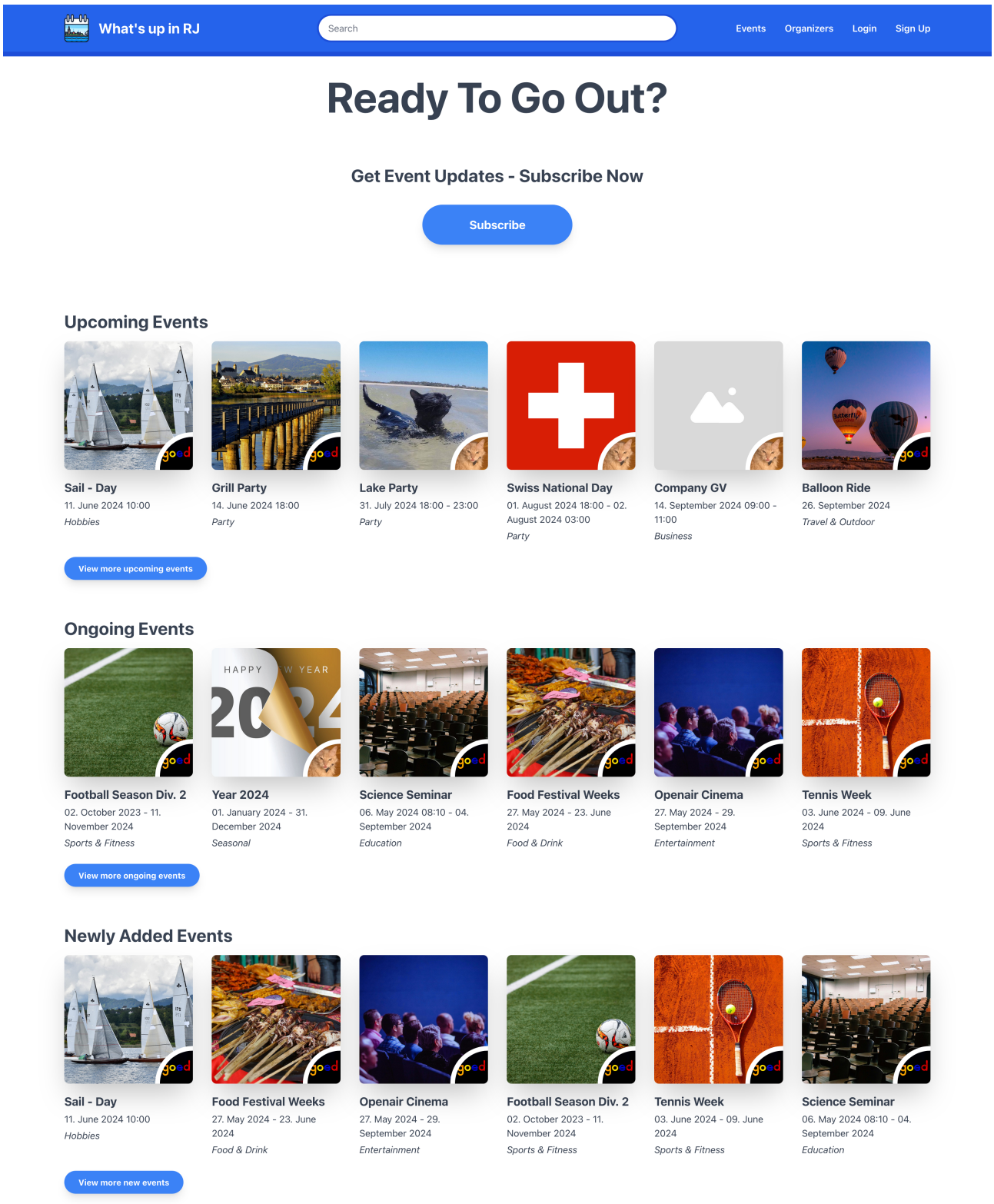
Desktop: Create event



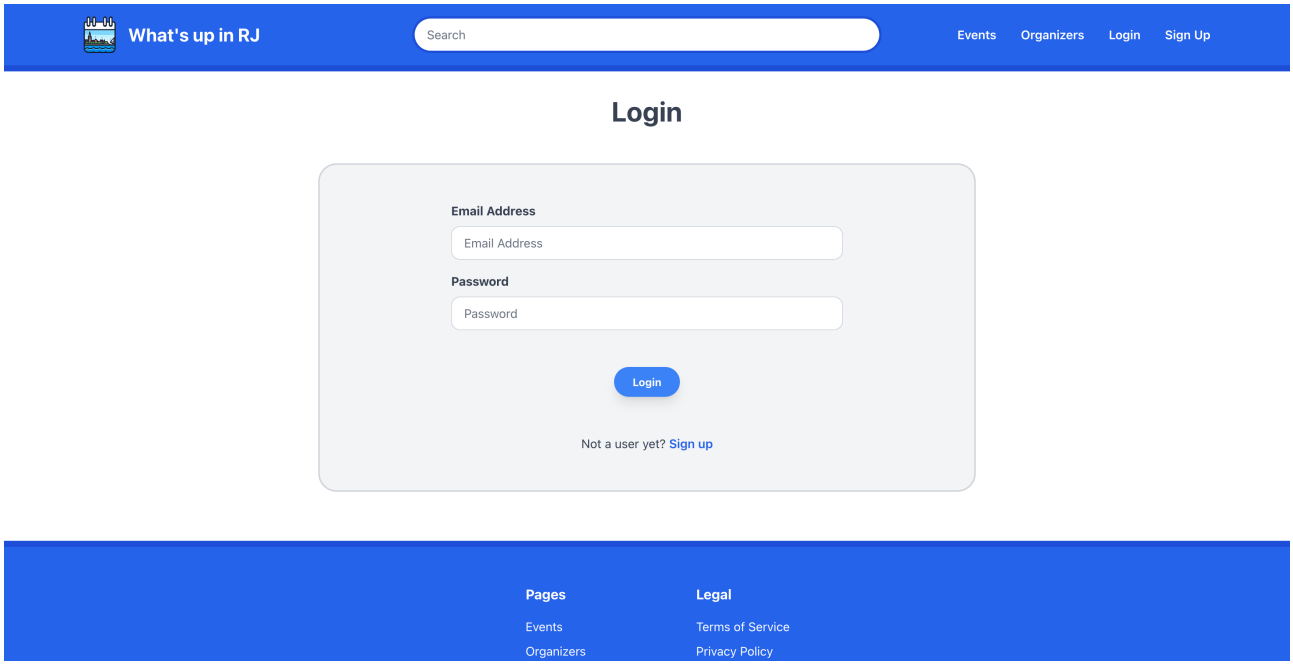
Desktop: Edit event



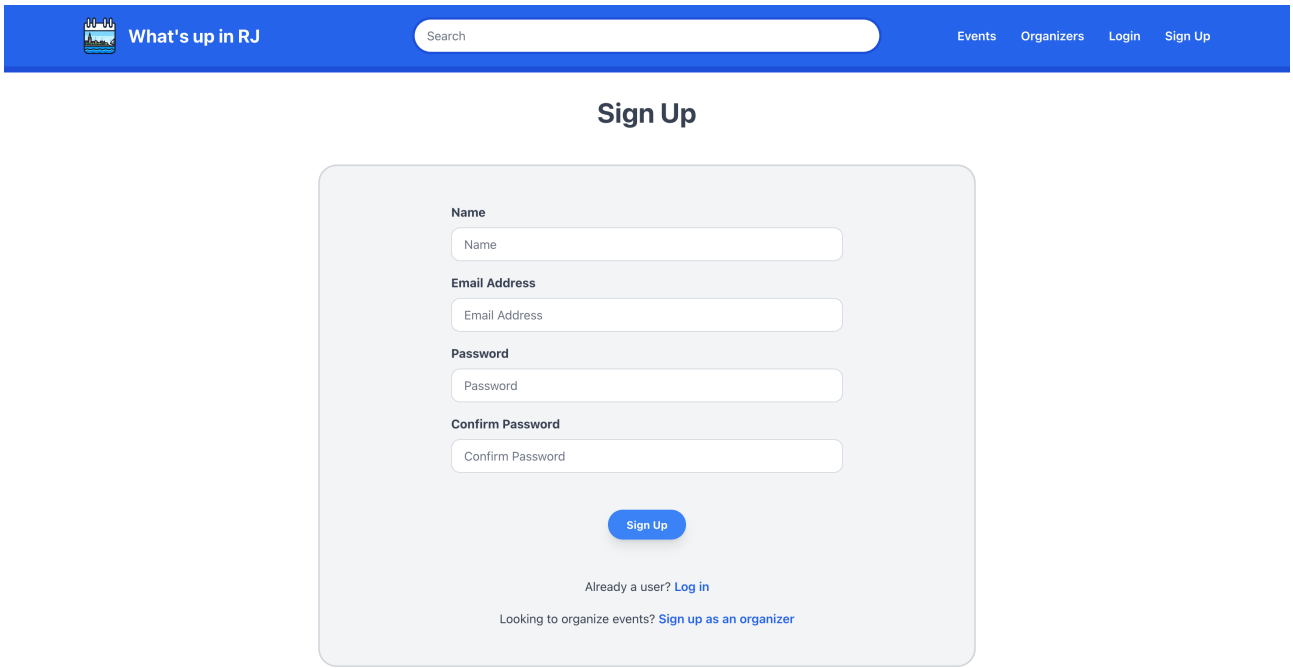
Desktop: Edit event of unverified organizer



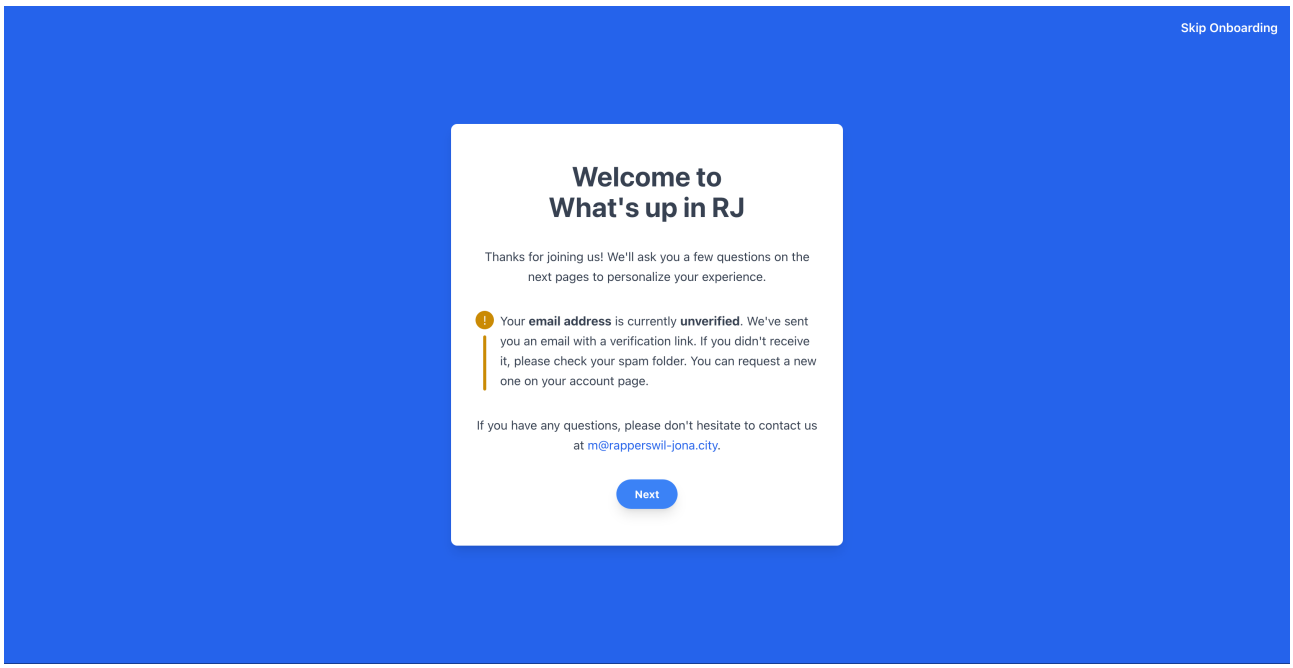
Desktop: Home page



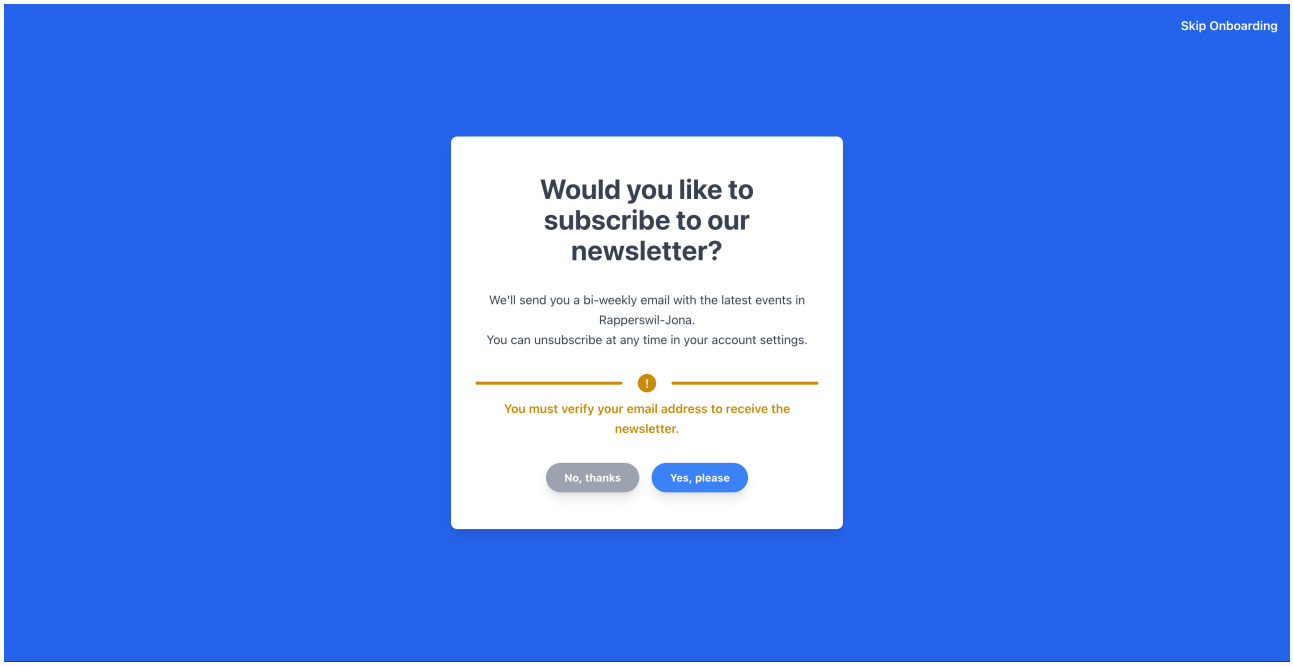
Desktop: Login page



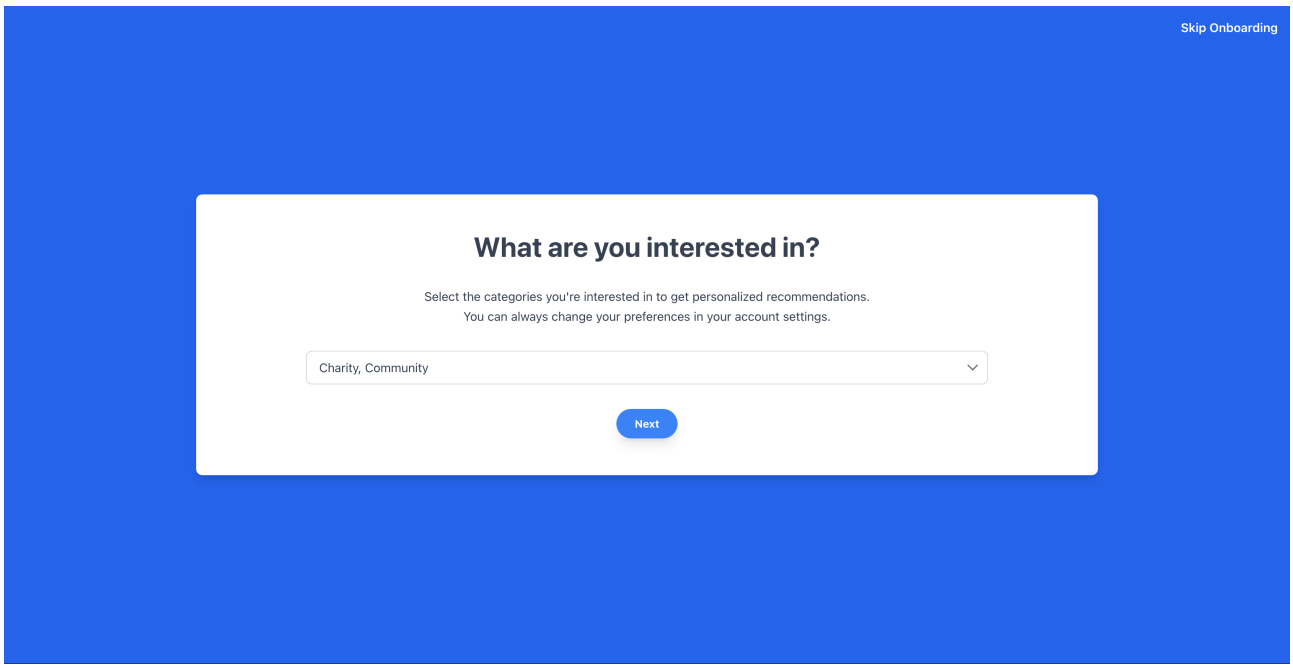
Desktop: Sign up page



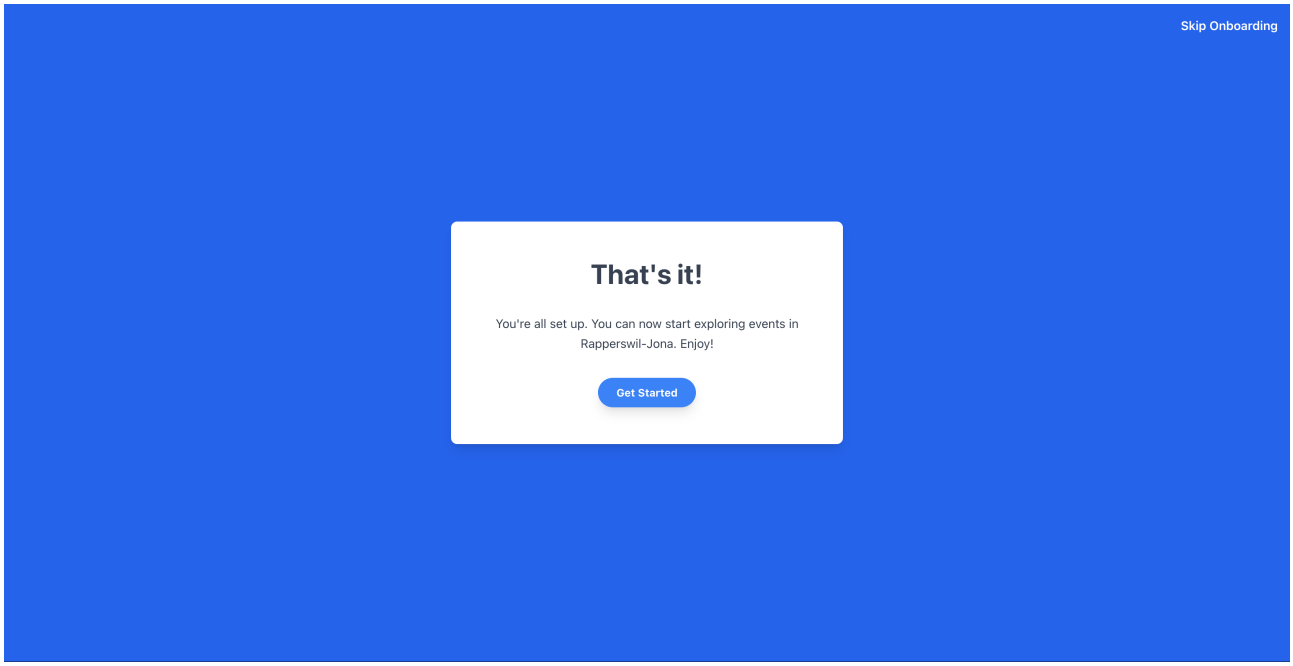
Desktop: Onboarding step 1 for registered users



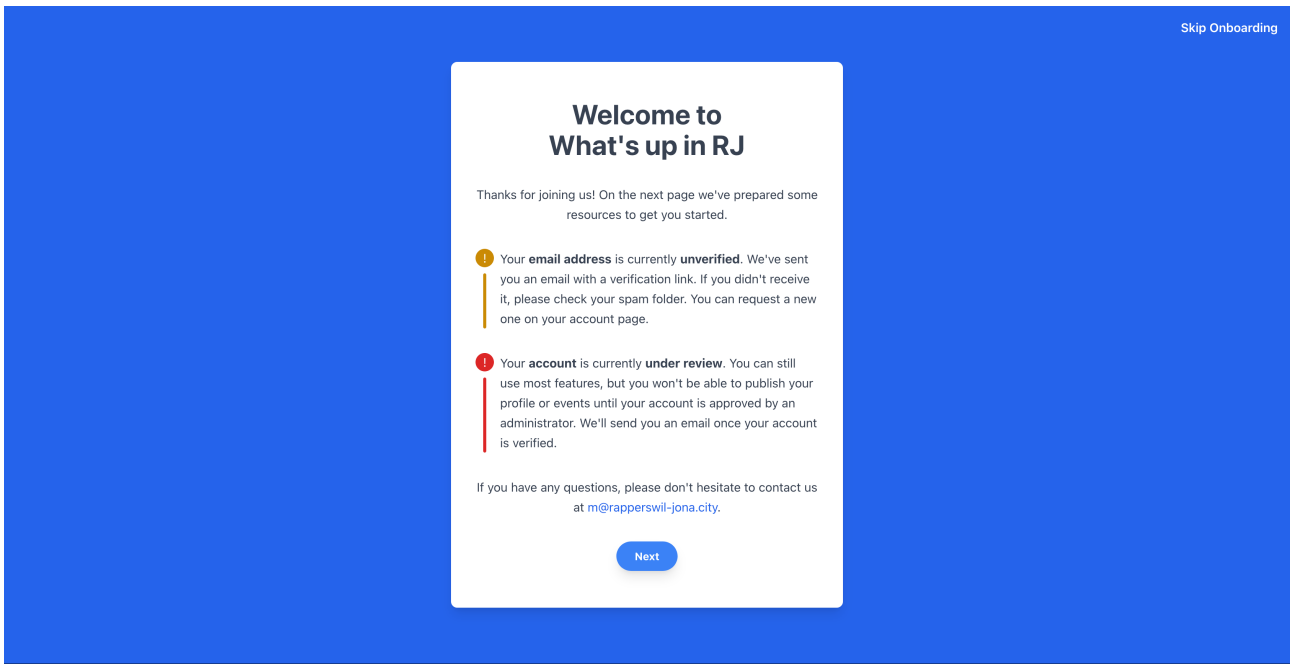
Desktop: Onboarding step 2 for registered users



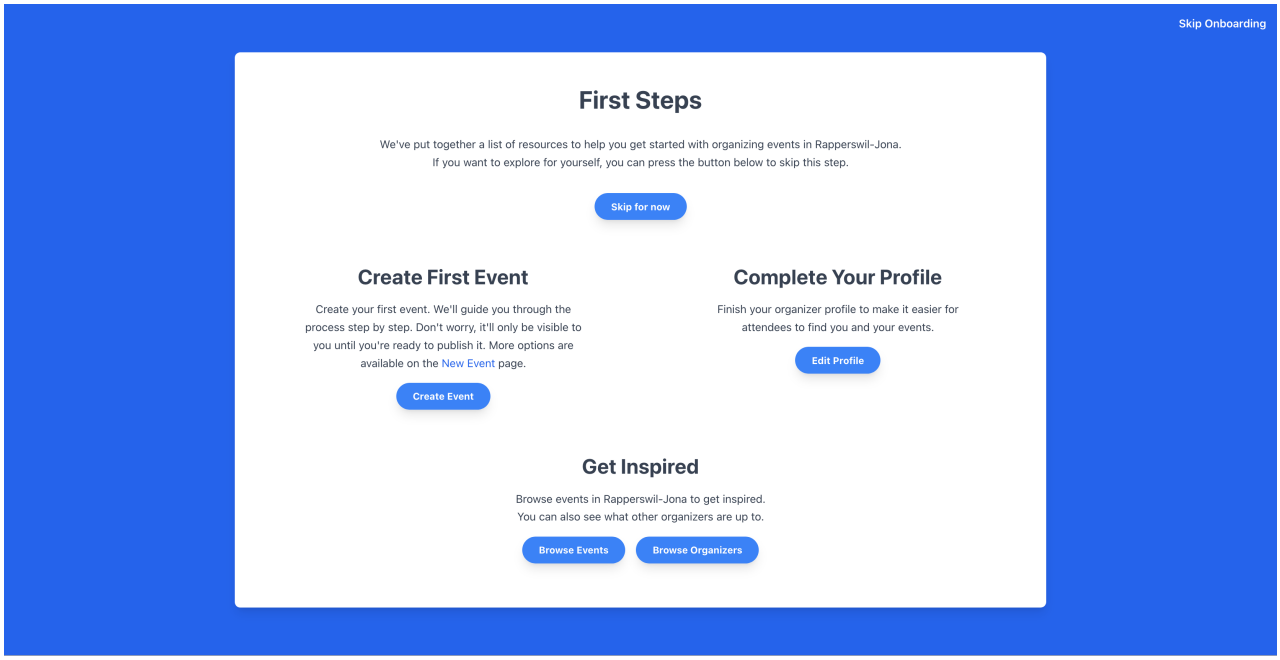
Desktop: Onboarding step 3 for registered users



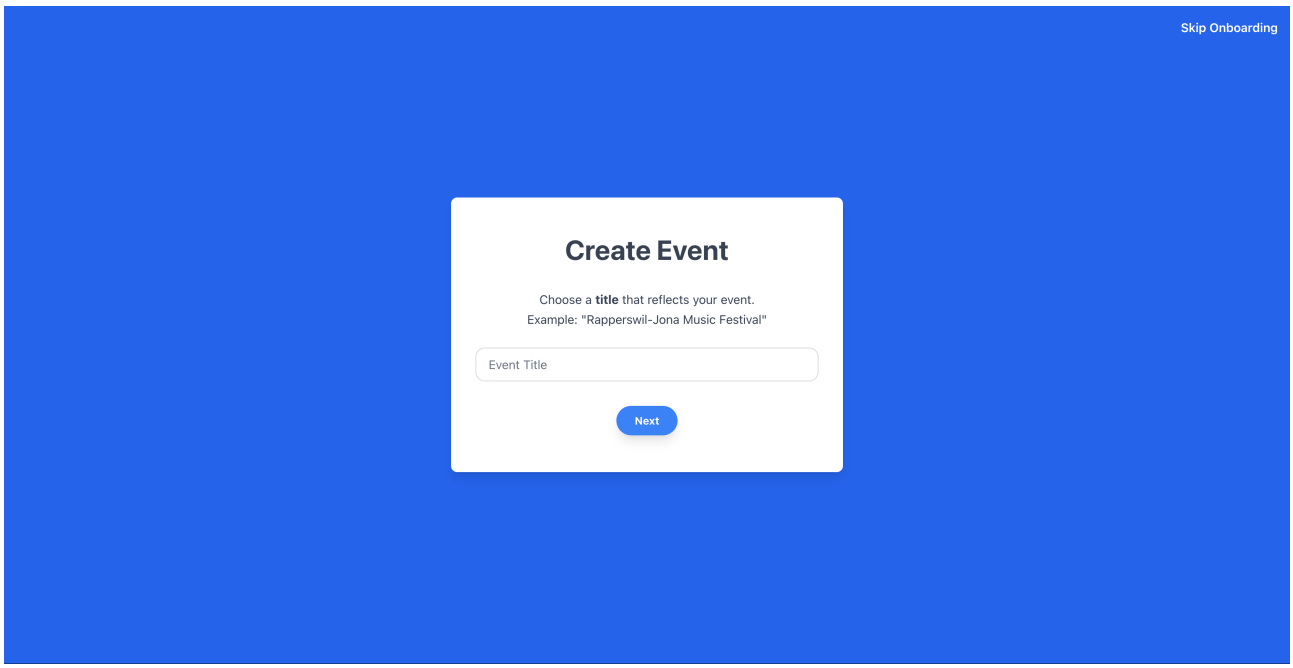
Desktop: Onboarding step 4 for registered users



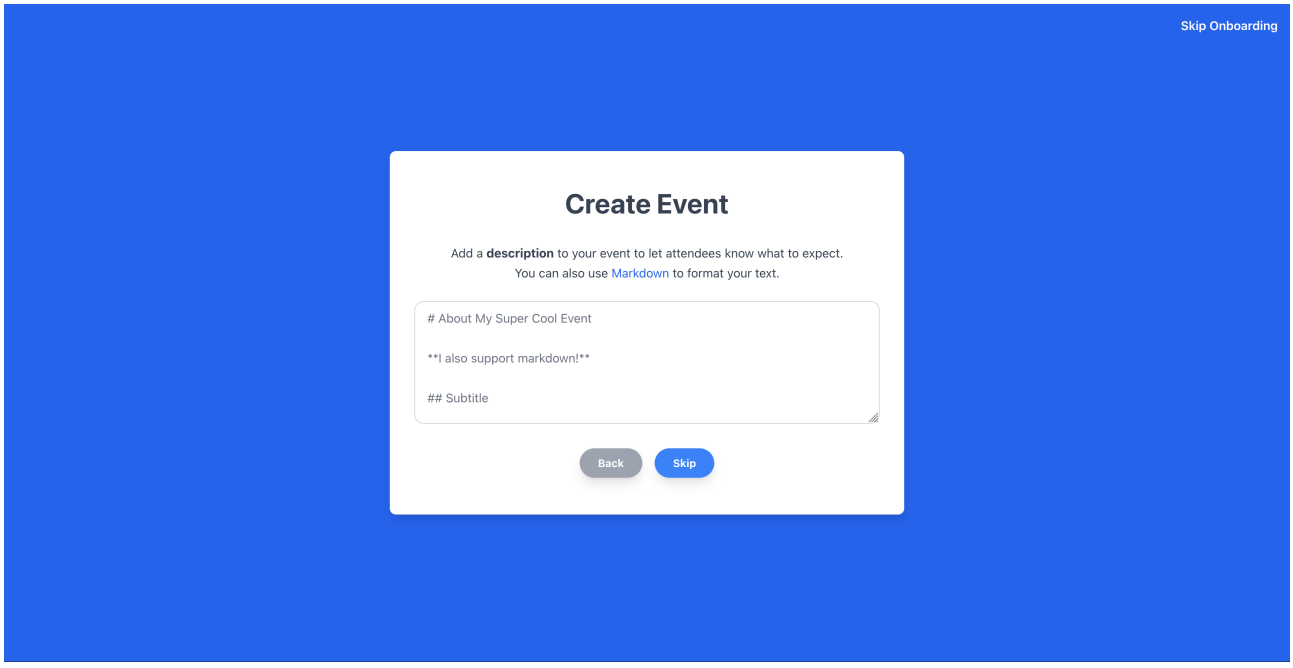
Desktop: Onboarding step 1 for organizers



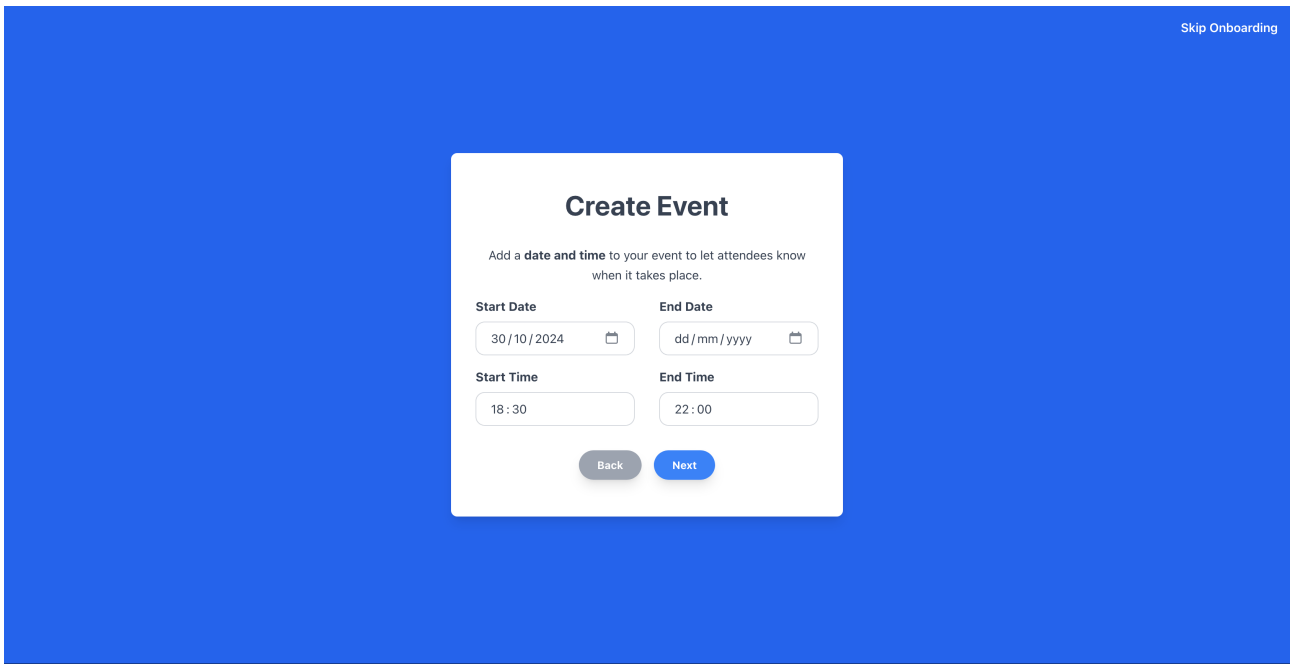
Desktop: Onboarding step 2 for organizers



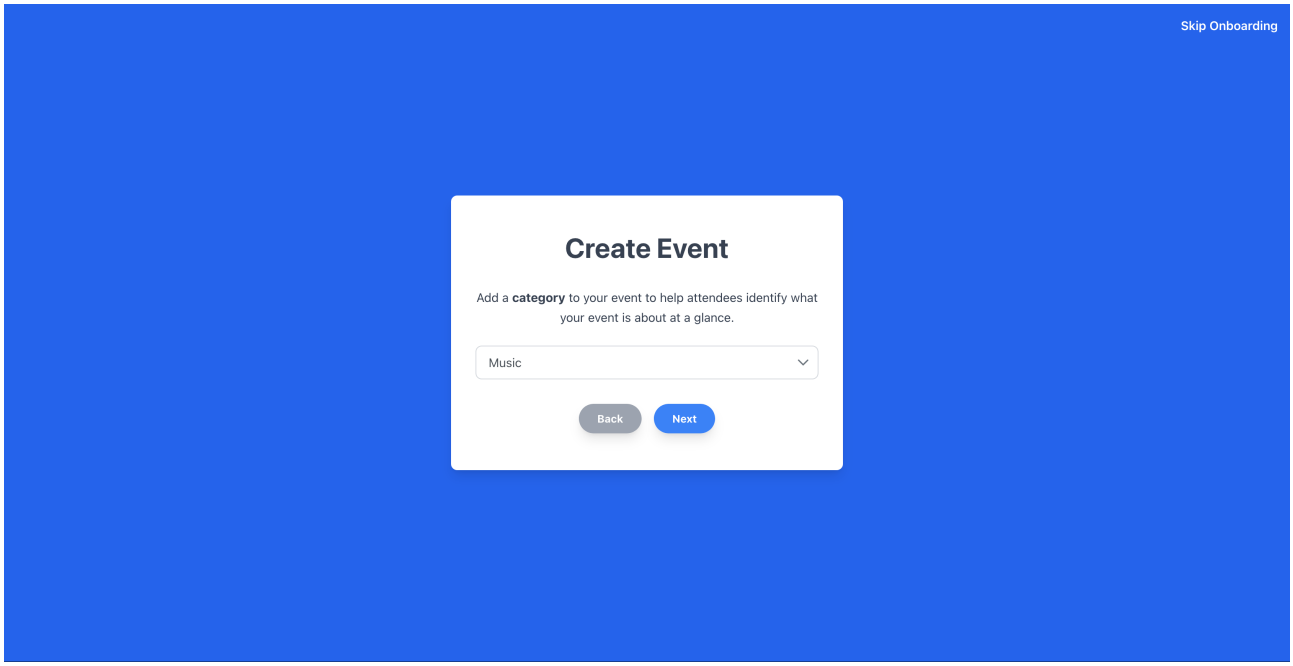
Desktop: Onboarding step 3 for organizers (create event)



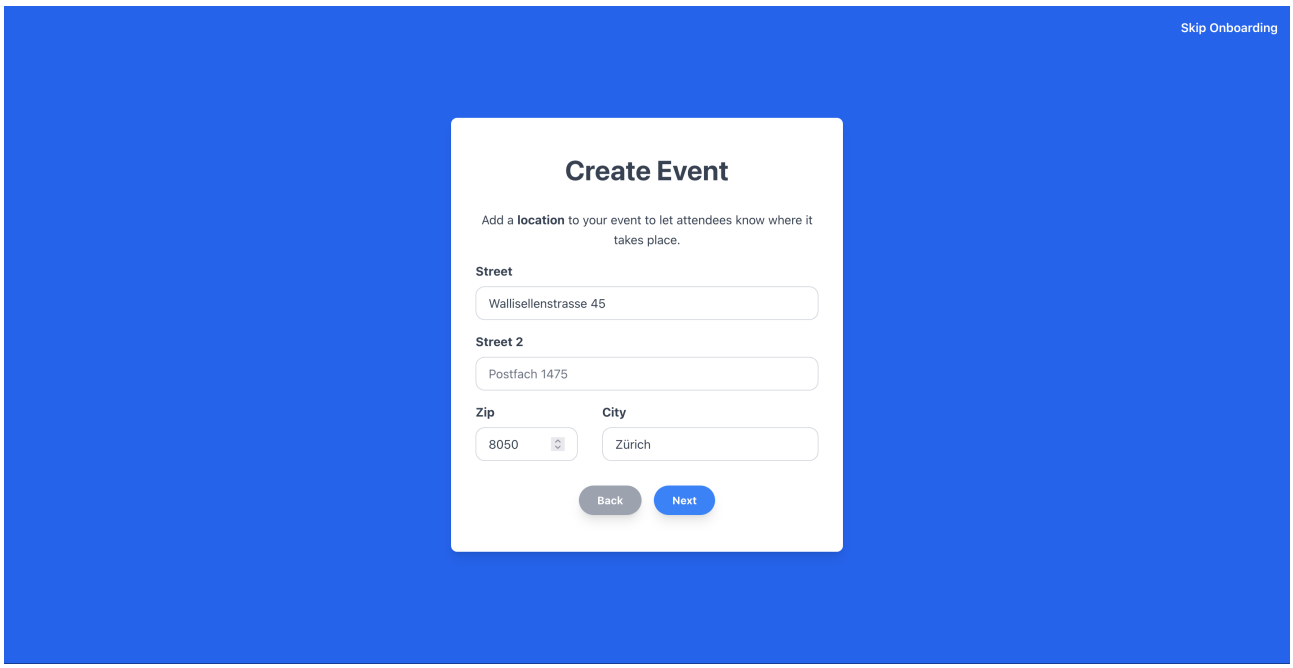
Desktop: Onboarding step 4 for organizers (create event)



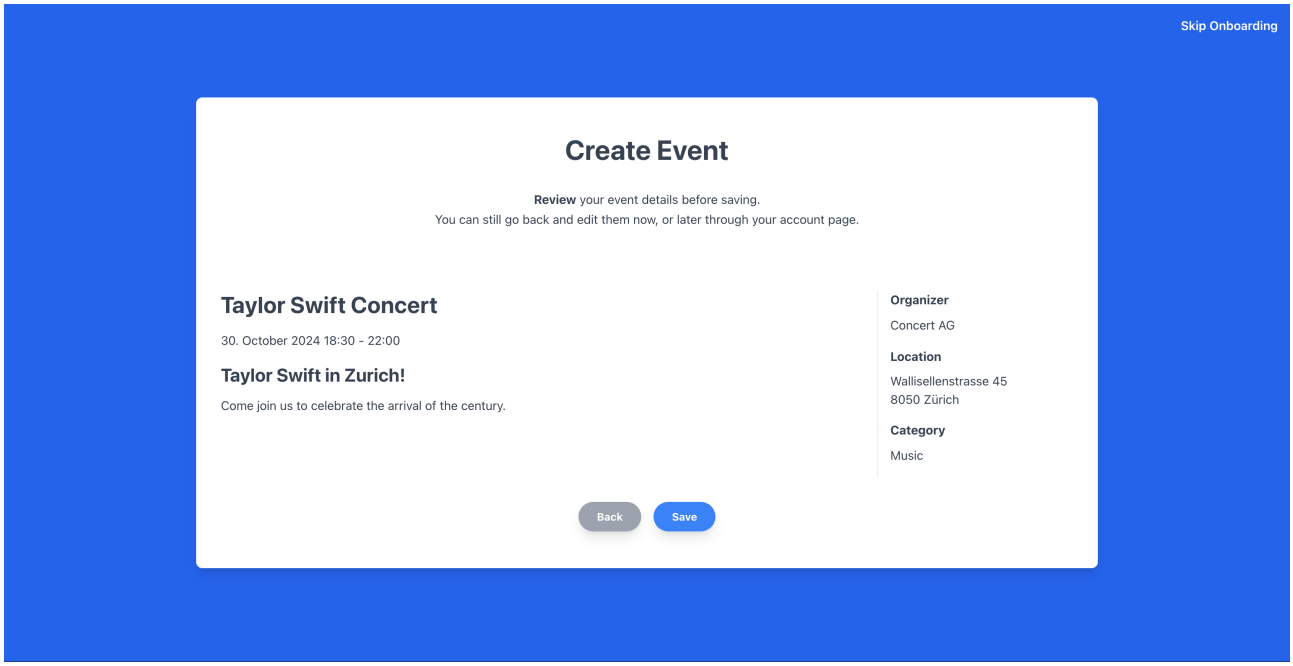
Desktop: Onboarding step 5 for organizers (create event)



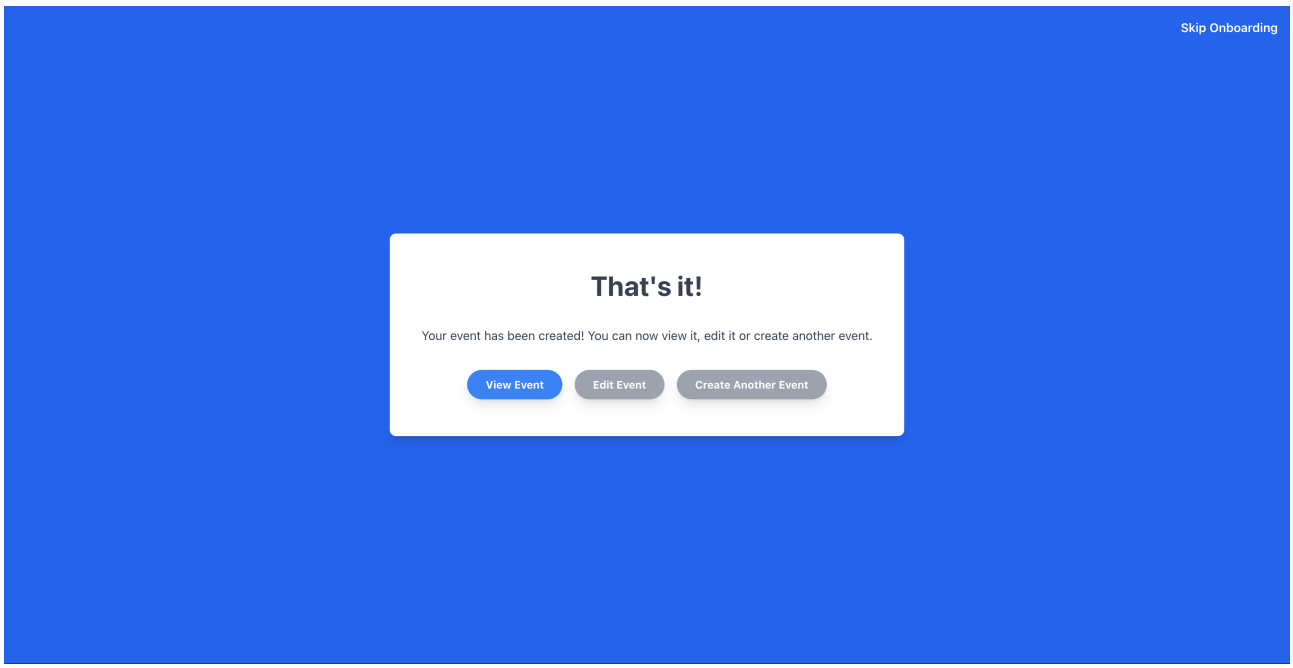
Desktop: Onboarding step 6 for organizers (create event)



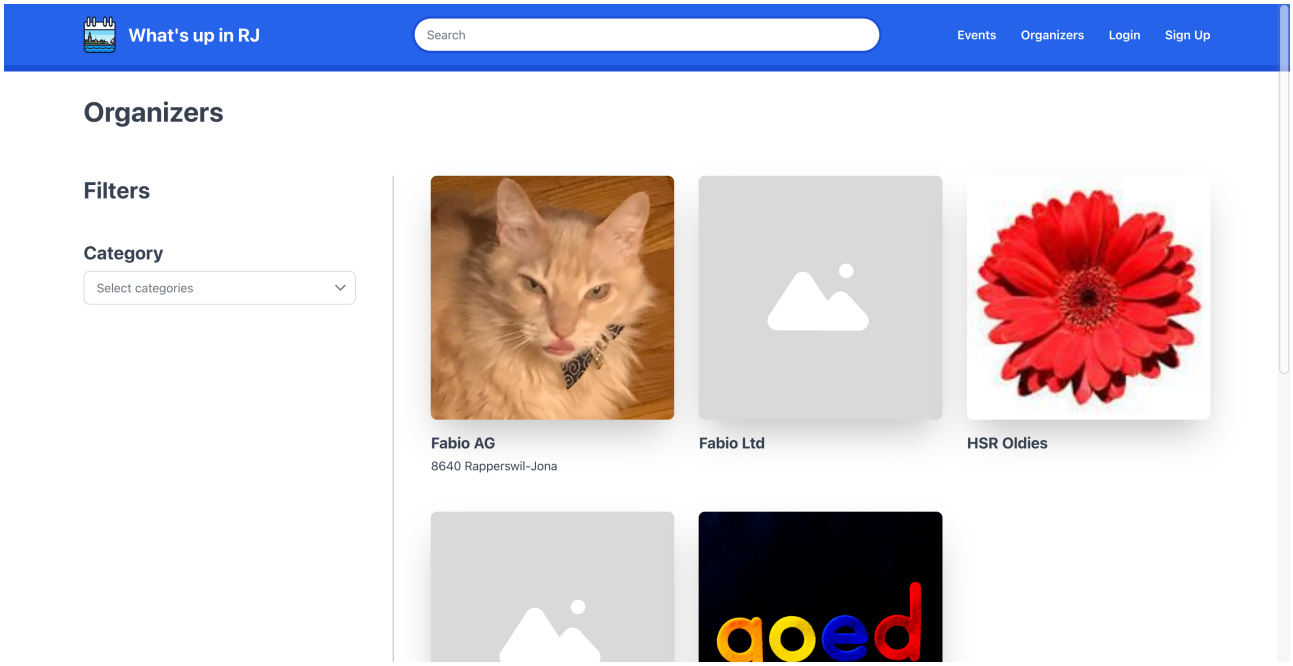
Desktop: Onboarding step 7 for organizers (create event)



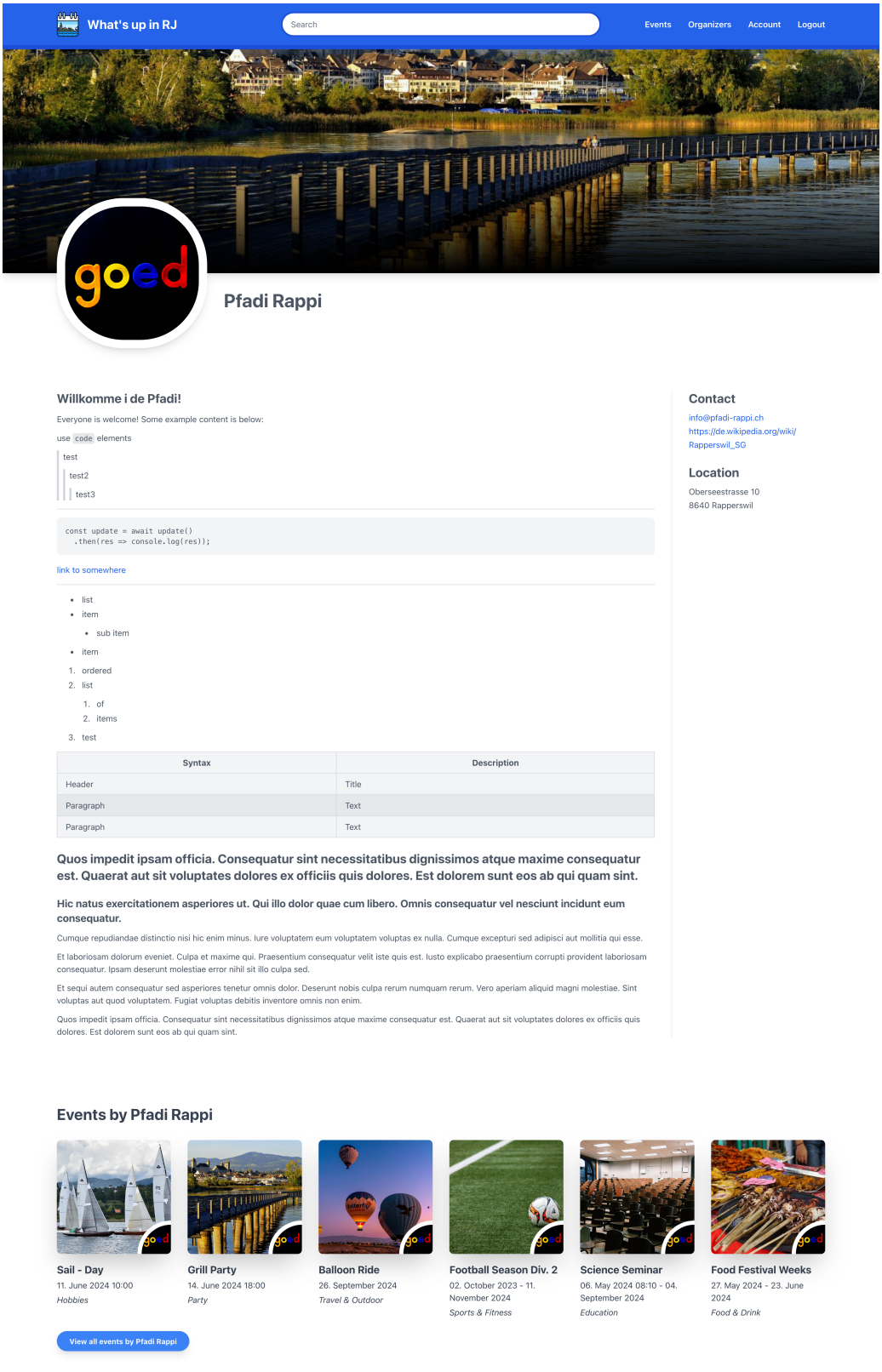
Desktop: Onboarding step 8 for organizers (create event)



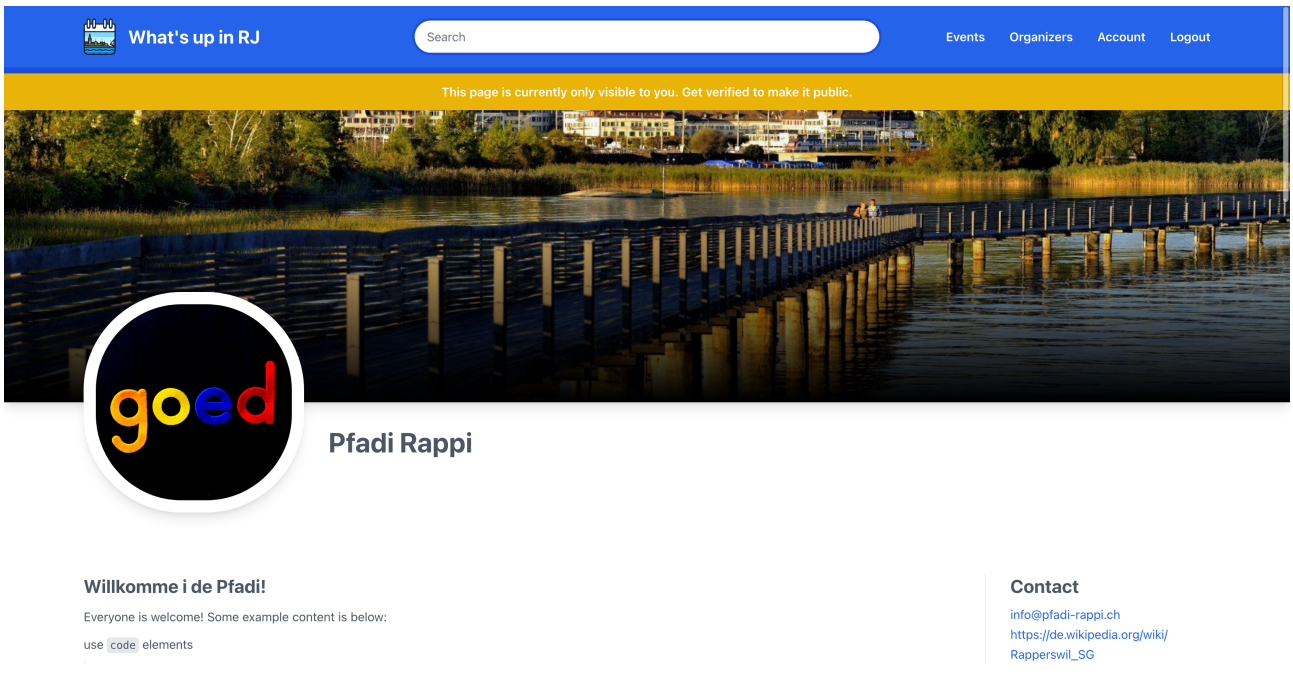
Desktop: Onboarding step 9 for organizers (create event)



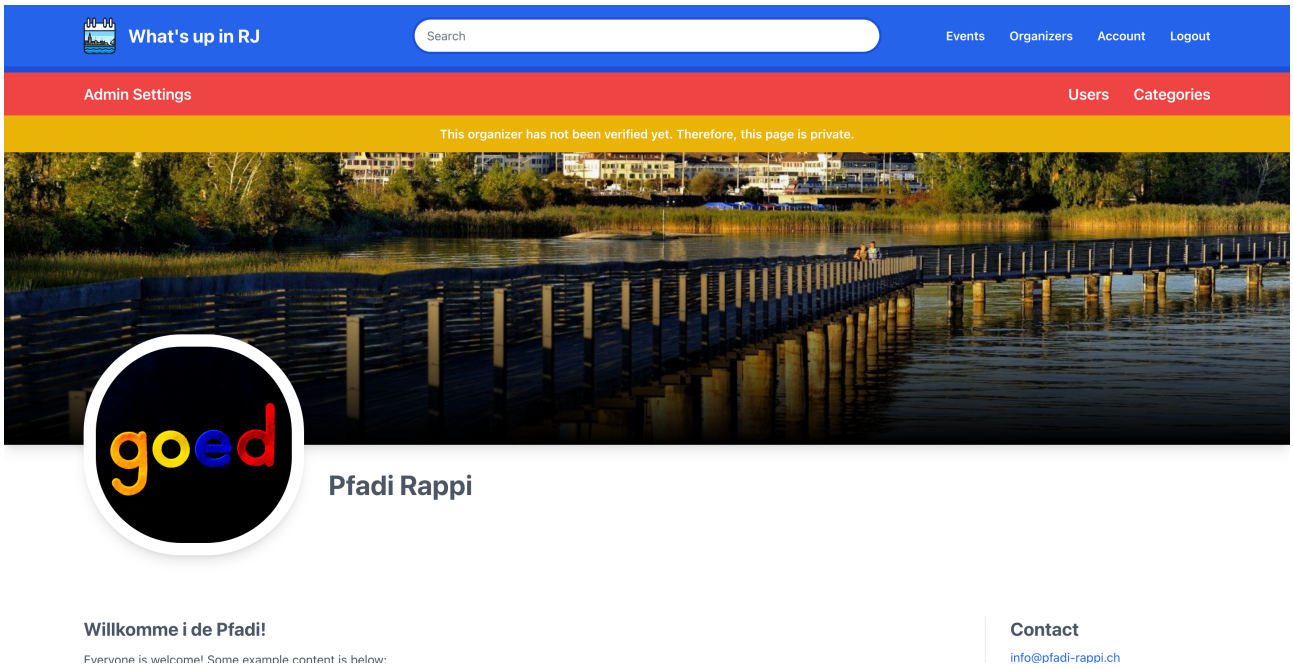
Desktop: Organizer overview



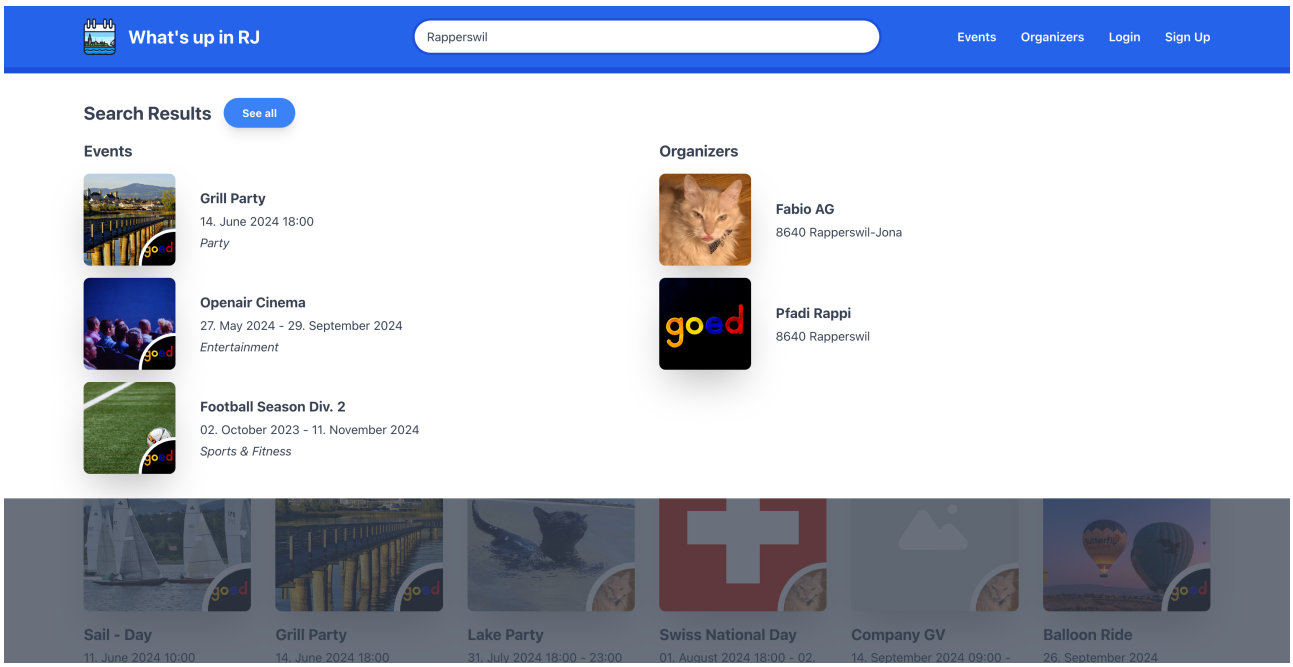
Desktop: Single organizer



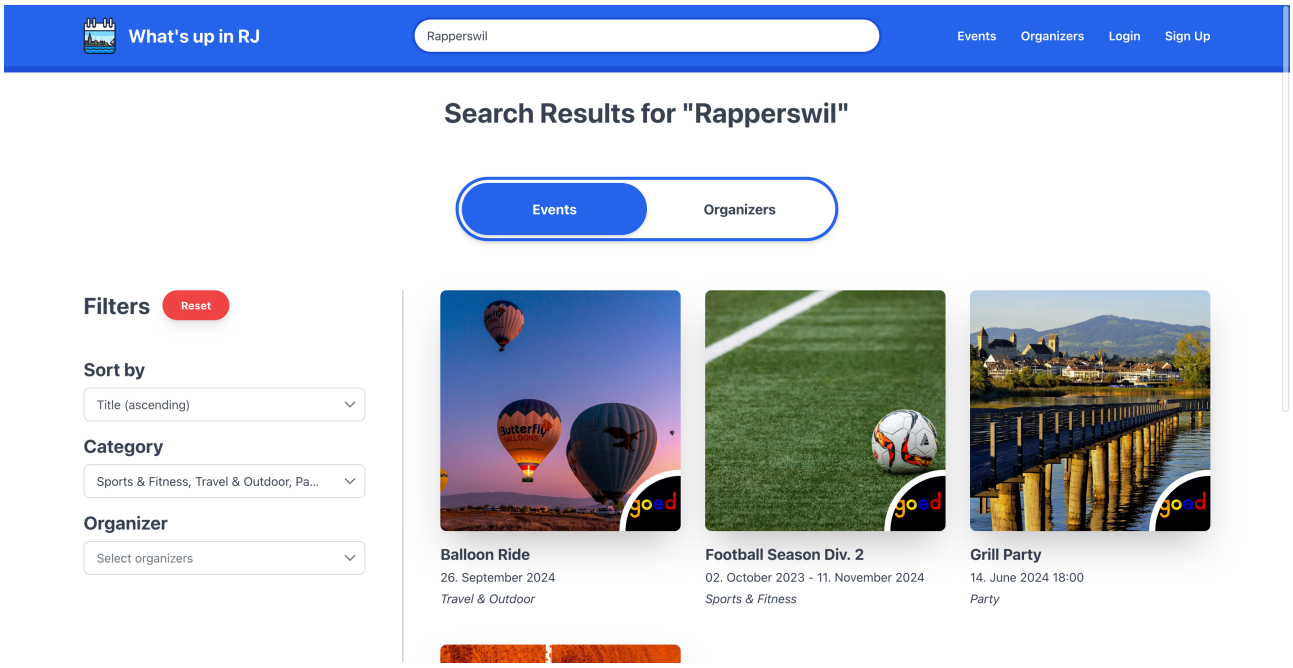
Desktop: Single unverified organizer



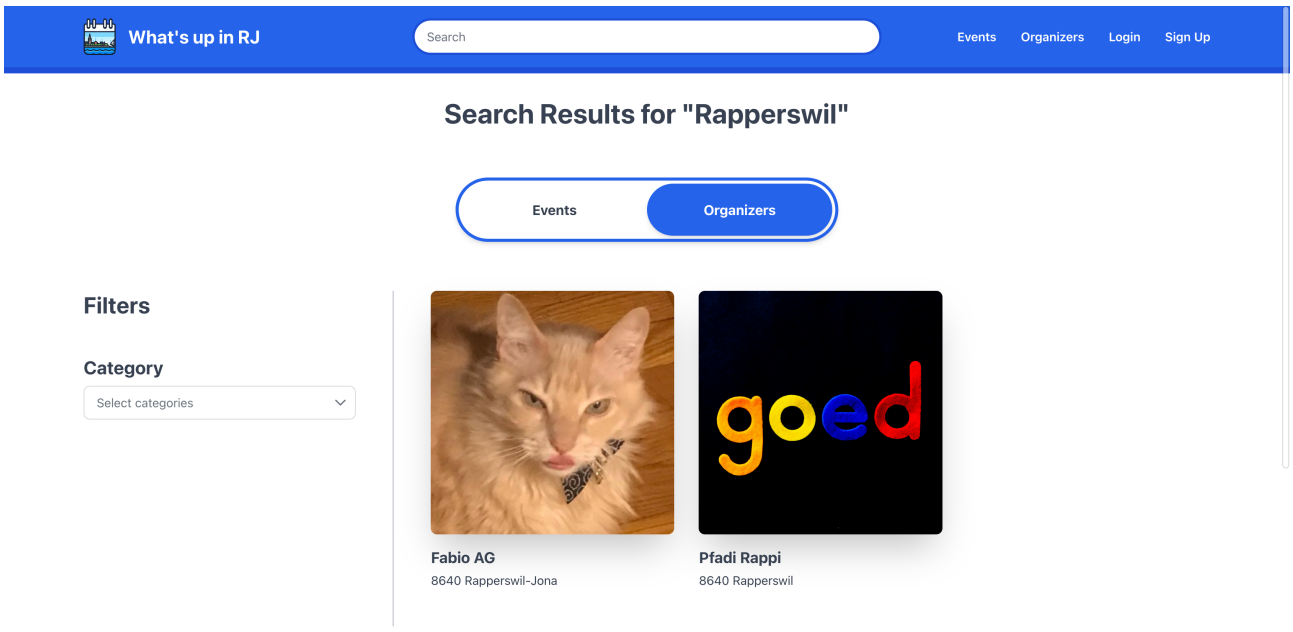
Desktop: Single unverified organizer viewed by admin



Desktop: Search popover



Desktop: Search page for events



Desktop: Search page for organizers

New Events just dropped in Rapperswil!

Christmas Dance

by Let's Dance

20 December 2024 18:00 - 20:00

Music

Eminem Concert

by Ticketmaster AG

22 August 2024 20:00 - 23:00

Concerts

Desktop: Newsletter email

Verify your email address

Hey Username!

Please click the link below to verify your email address. This link is valid for 24 hours.
You can always request a new email verify link on your account page.

https://whatsup.rapperswil-jona.city/auth/verify-email/3acb4017492be824fda610cef842492c3acb4017492be824fda610cef842492c

- The What's up in RJ Team

This is an automated message, please do not reply.

You are receiving this email because you have previously signed up on What's up in RJ. If you believe to have received this email in error, contact us.

Terms of Service
Privacy Policy

Desktop: Email verification email

Your account has been verified!

Hey Username

Congratulations! An administrator has reviewed and verified your account.
You now benefit from the full array of features on our platform. Here are some things to do to get you started:

- Publish your first event
- Customise your profile

Not sure what information to put in your first event?
Get some inspiration by browsing other events.

Thanks for being a part of What's up in RJ!

- The What's up in RJ Team

This is an automated message, please do not reply.

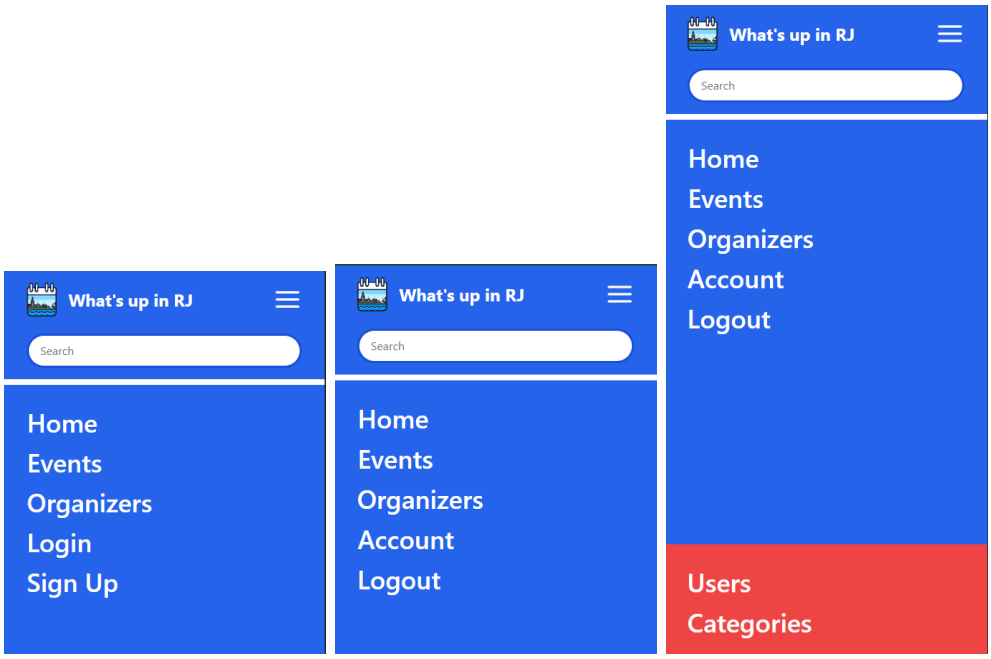
You are receiving this email because you have previously signed up as an organizer on What's up in RJ. If you believe to have received this email in error, contact us.

Terms of Service
Privacy Policy

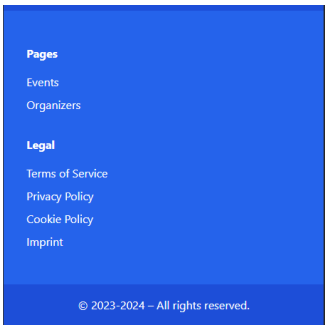
Desktop: Organizer account verification email

14.2 Mobile

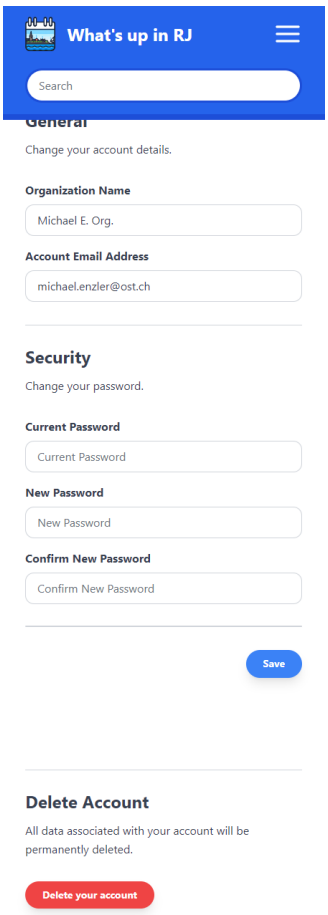
Mobile screenshots use a screen width of 450px.



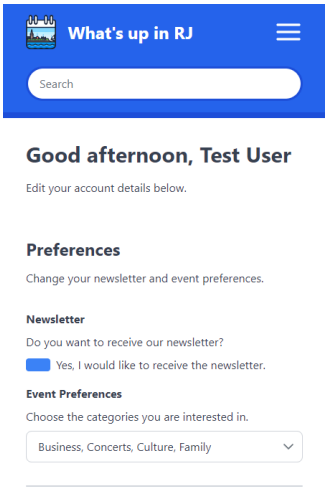
Mobile: Header for logged-out user, logged-in user, and admin



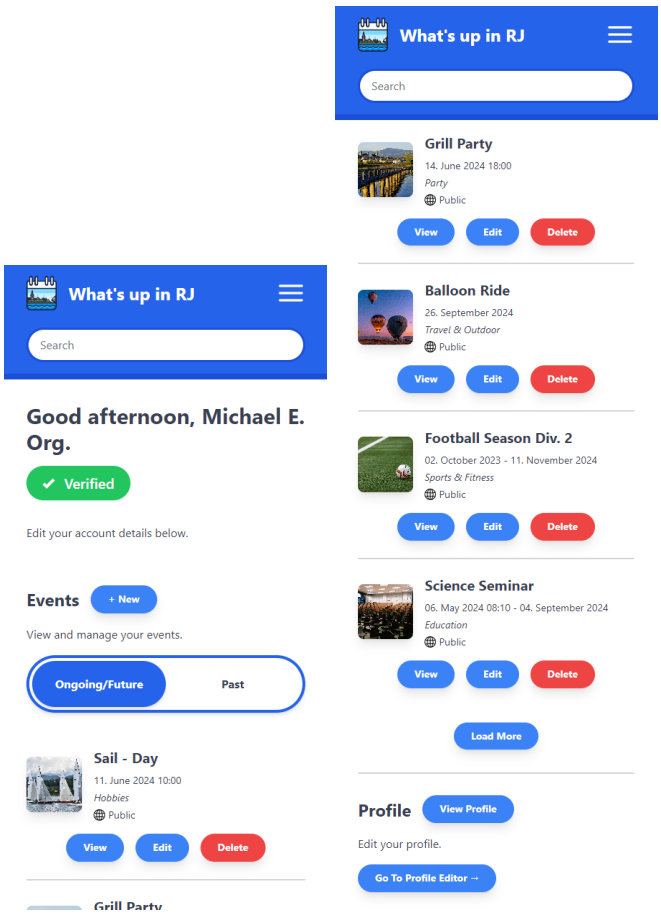
Mobile: Footer



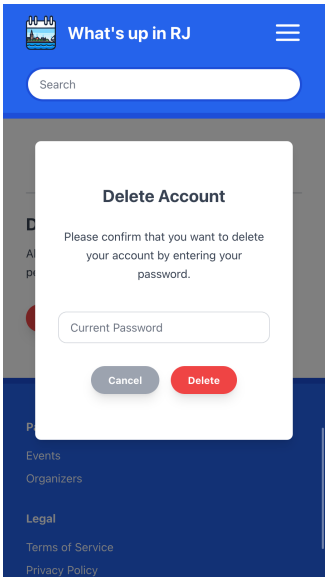
Mobile: Account page displaying settings for all user types



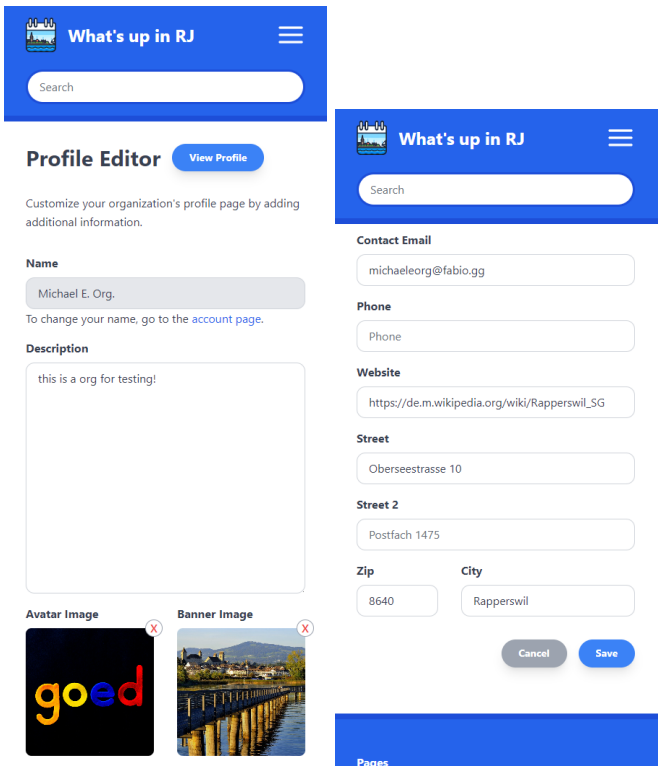
Mobile: Account page of a user



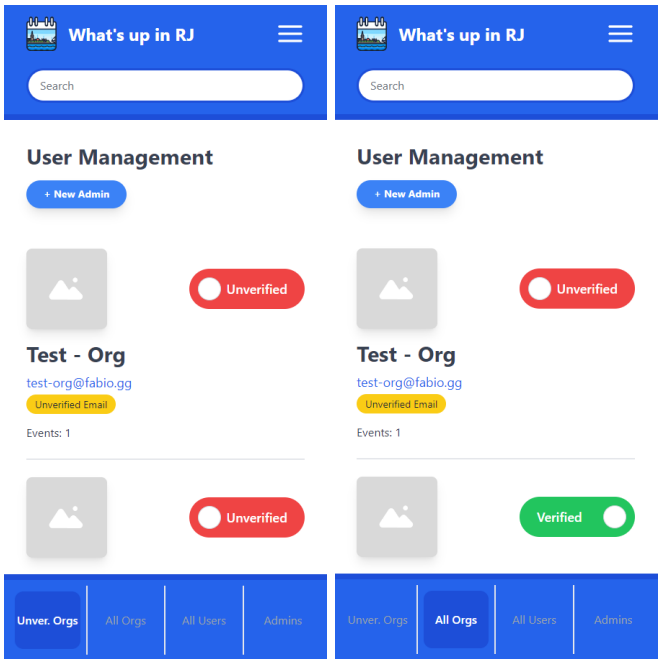
Mobile: Account page of an organizer



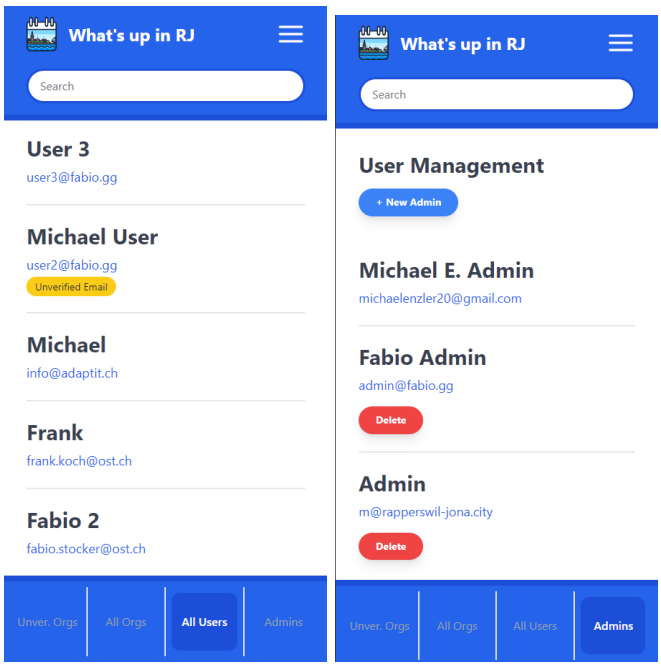
Mobile: Delete account dialogue



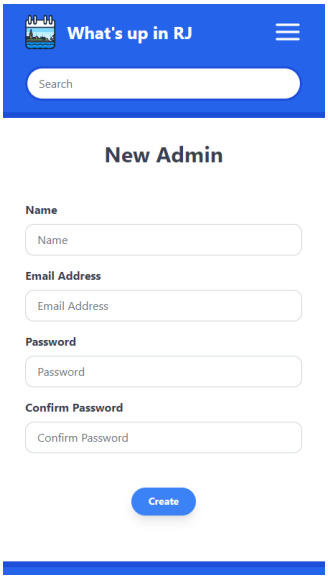
Mobile: Profile editor



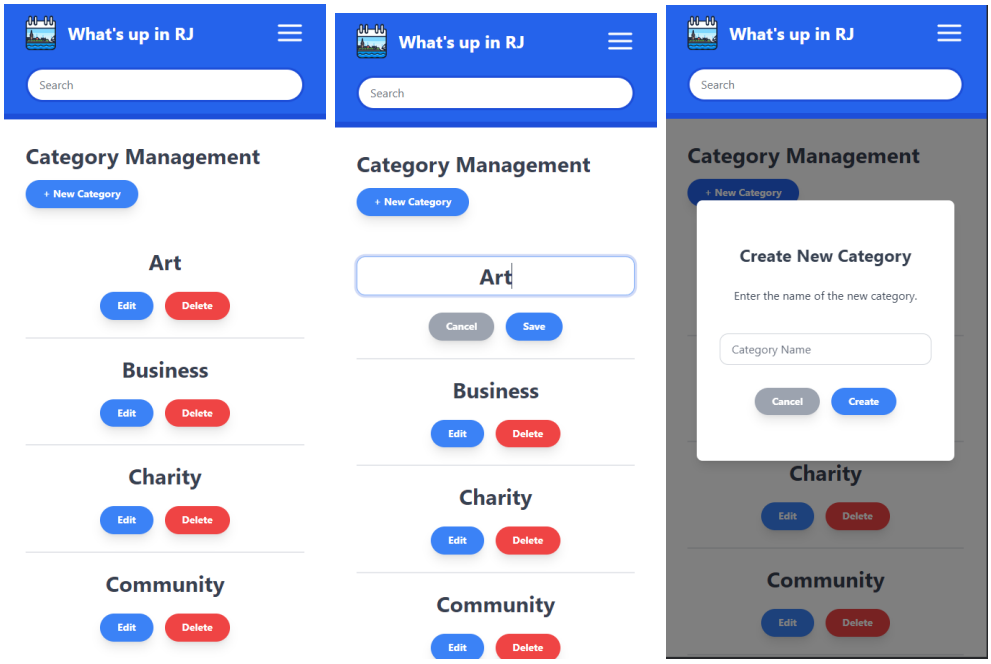
Mobile: Unverified and all organizers in the admin panel



Mobile: Users and admins in the admin panel

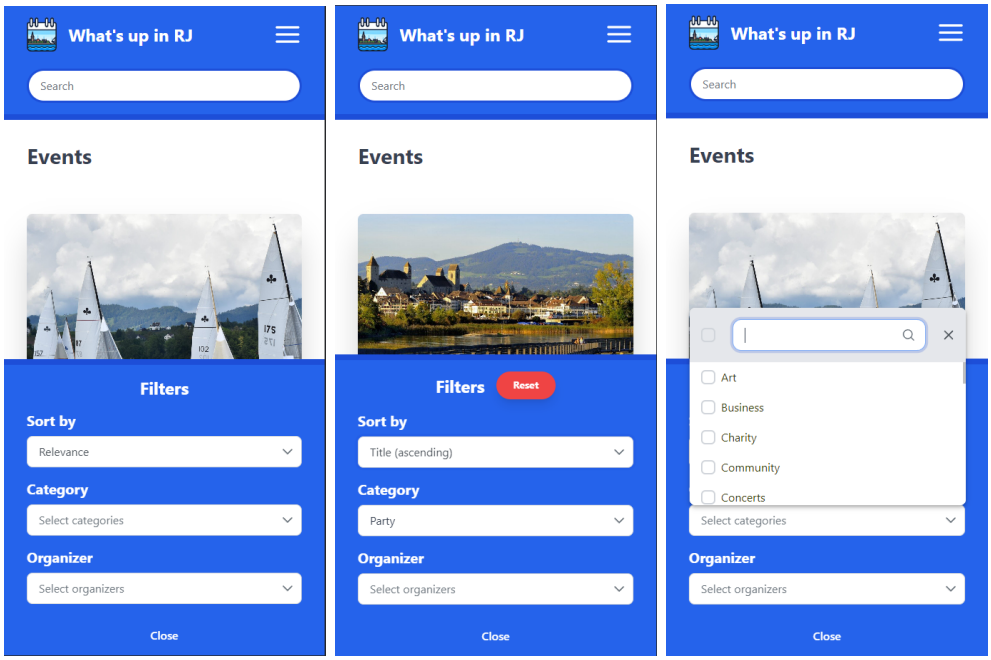


Mobile: New admin page in the admin panel

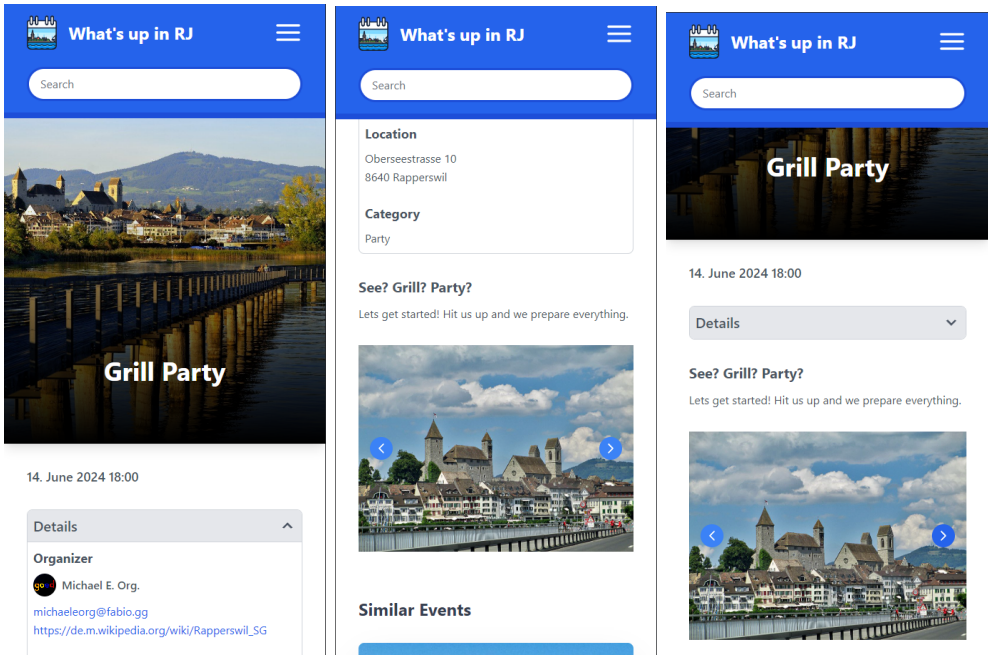


Mobile: Category management in the admin panel

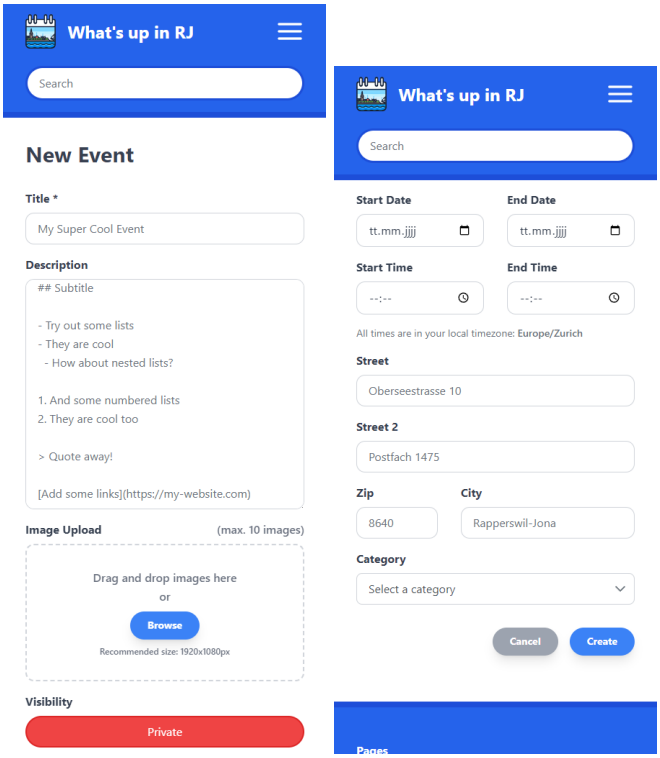
Mobile: Event overview



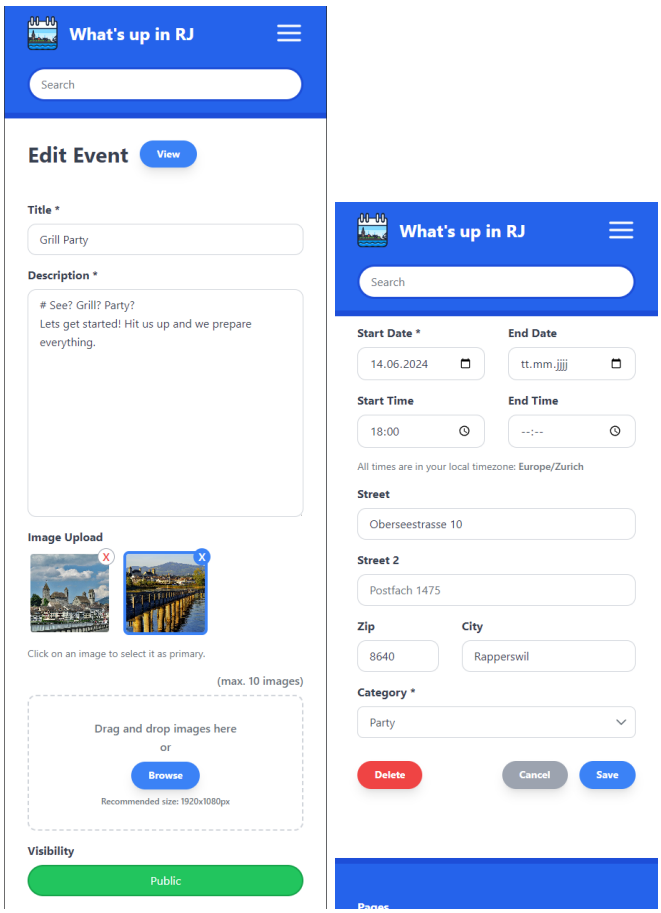
Mobile: Filters on the event overview



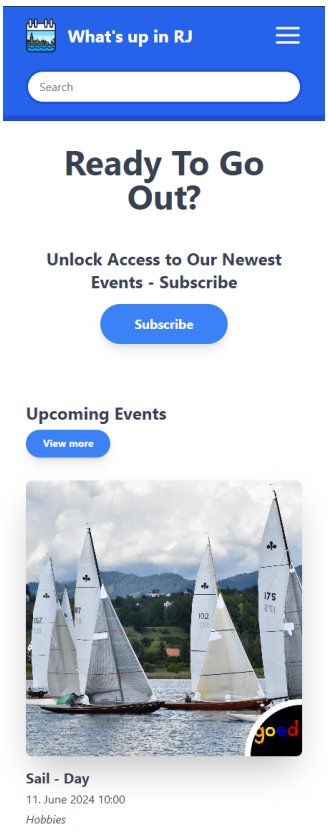
Mobile: Single event



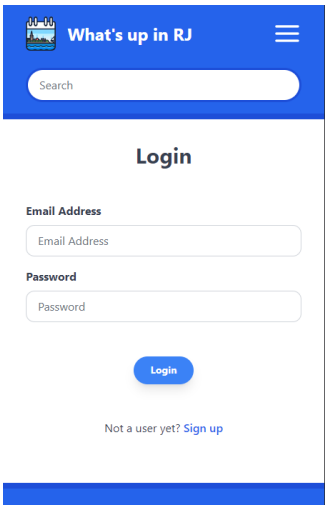
Mobile: Create event



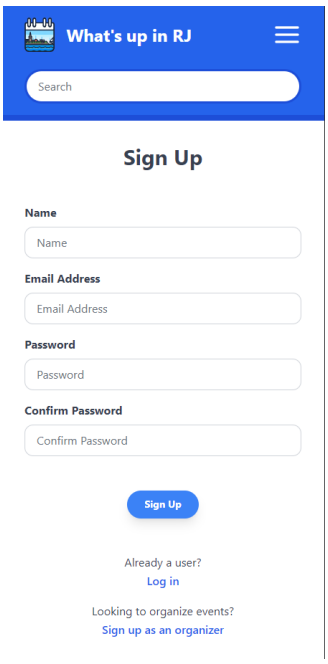
Mobile: Edit event



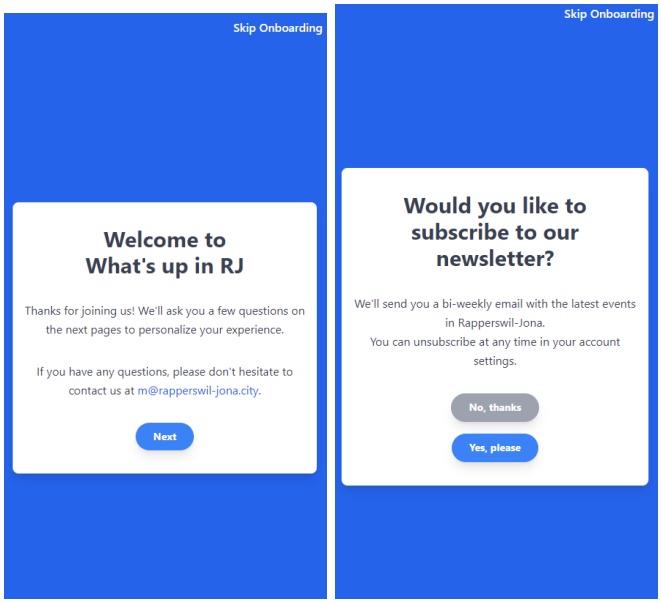
Mobile: Home page



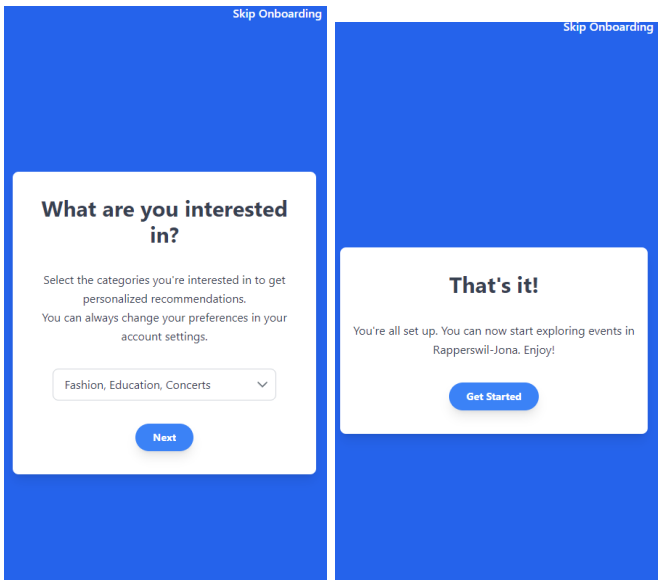
Mobile: Login page



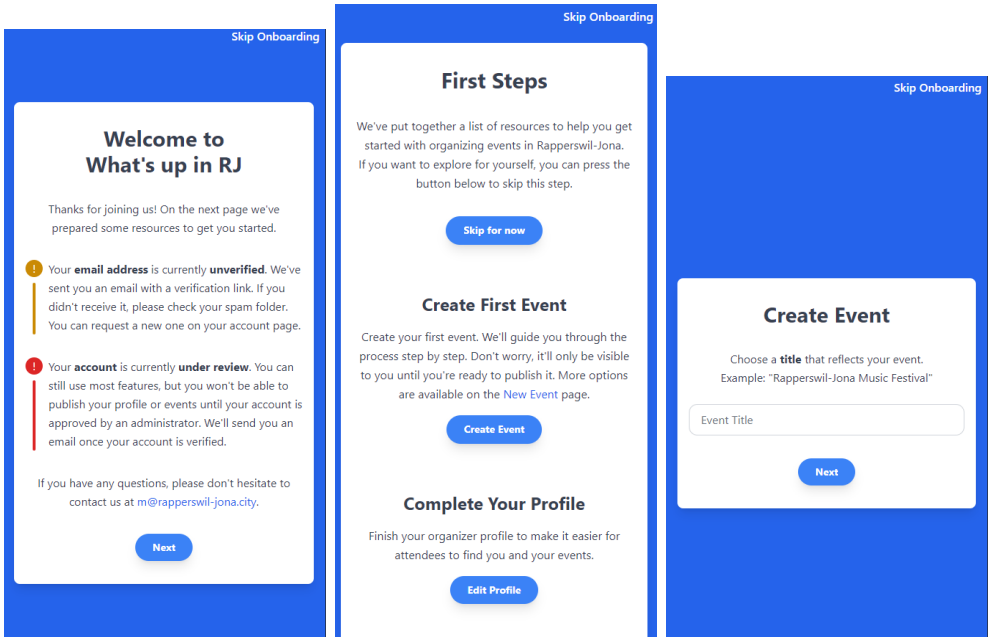
Mobile: Sign up page



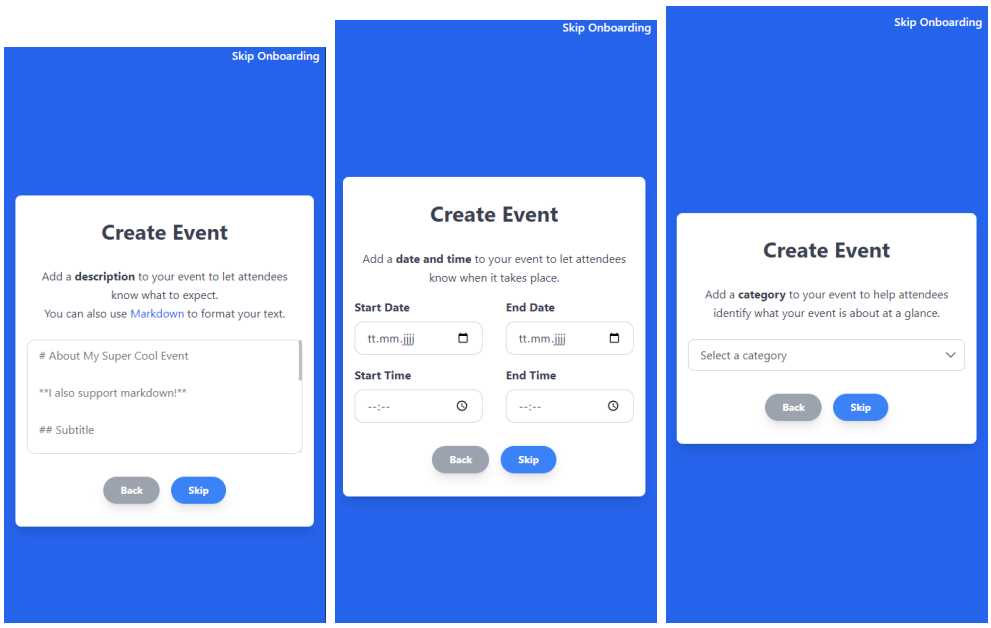
Mobile: Onboarding steps 1 & 2 for registered users



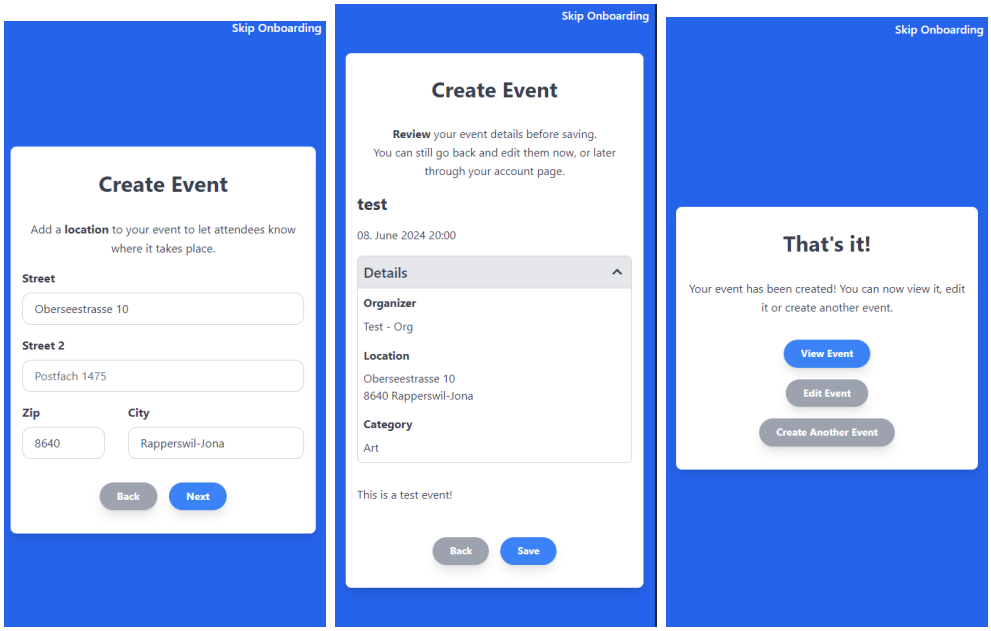
Mobile: Onboarding steps 3 & 4 for registered users



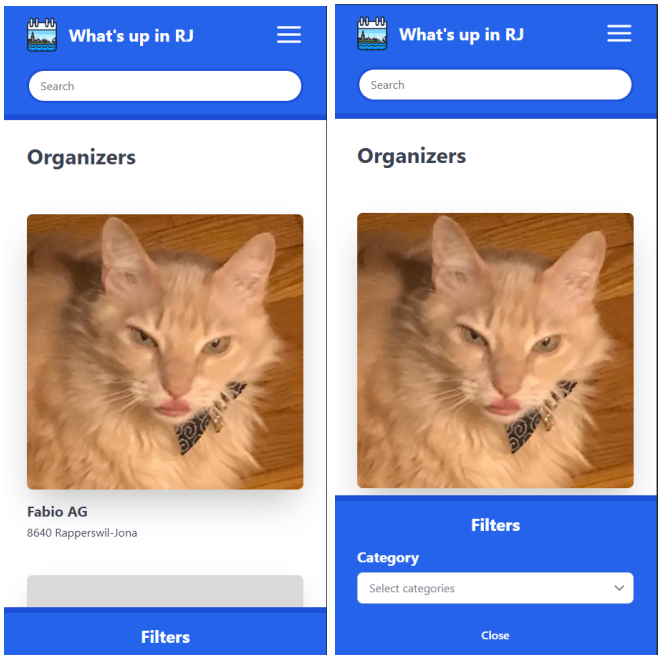
Mobile: Onboarding steps 1, 2, 3 for organizers



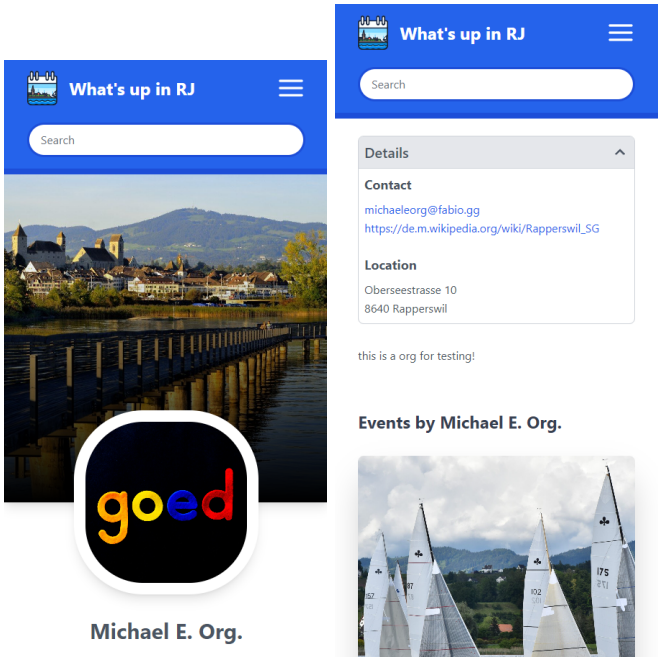
Mobile: Onboarding steps 4, 5, 6 for organizers



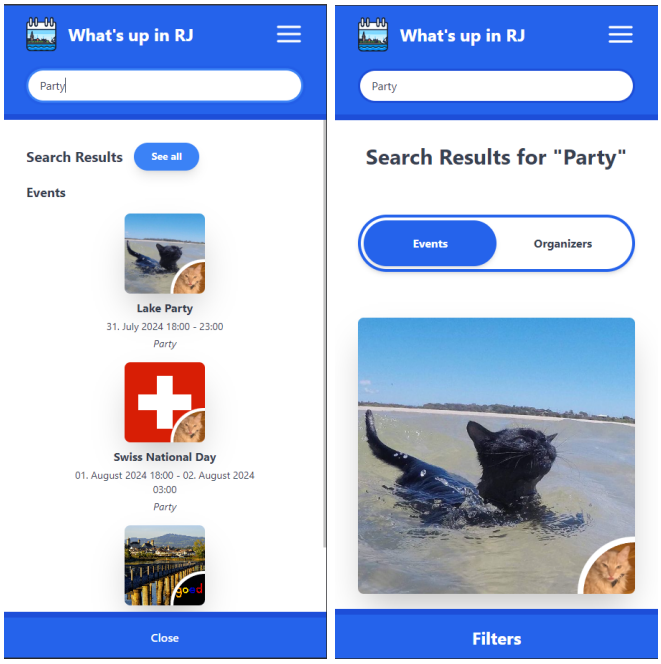
Mobile: Onboarding steps 7, 8, 9 for organizers



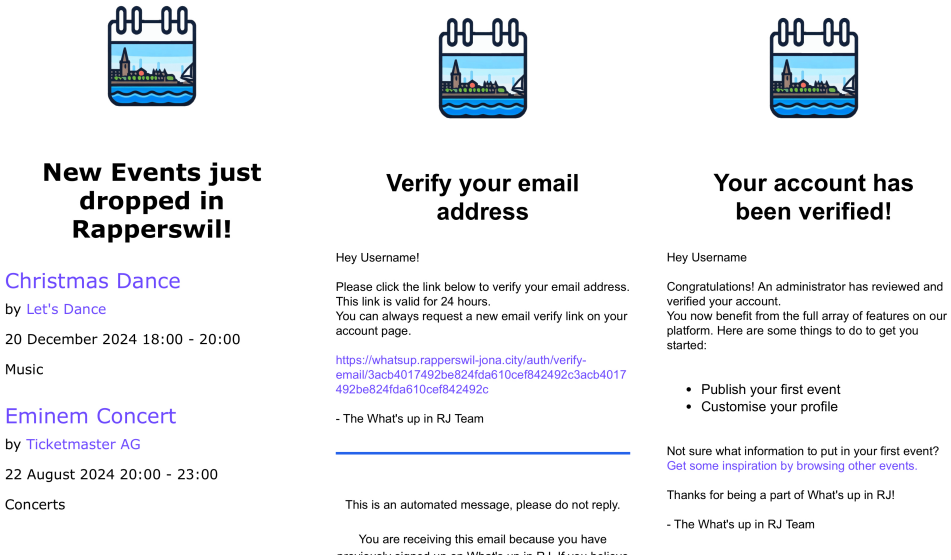
Mobile: Organizer overview and filters



Mobile: Single organizer



Mobile: Search popover and search results



Mobile: Newsletter, email verification, and organizer account verification emails

Chapter 15

Backend API Endpoints

What's up in RJ - Backend

This is the backend part of the project. The frontend part can be found here. The backend is a Node.js application that runs an Express server. It handles the API and the interaction with the database.

Table of Contents

        - GET /admin/admins
        - POST /admin/admins
        - DELETE /admin/admins/:id
    - /categories
        - GET /categories
        - GET /categories/:id
        - POST /categories
        - PUT /categories/:id
        - DELETE /categories/:id
    - /events
        - GET /events
        - GET /events/ongoing
        - GET /events/future
        - GET /events/:id
        - POST /events
        - PUT /events/:id
        - DELETE /events/:id
        - DELETE /events/image/:id
    - /organizers
        - GET /organizers
        - GET /organizers/:id
    - /search
        - GET /search/:query
        - GET /search/events/:query
        - GET /search/organizers/:query

Rate Limiting

The API is rate limited using `express-rate-limit`. The rate limit is `100` requests per `60` seconds per IP address. If the rate limit is exceeded, the server will respond with a `429 Too Many Requests` status code. The rate limit can be configured in the `main.ts` file.

Enums

UUID

The UUID type is used to represent a UUID. It is a string that is 36 characters long and follows the UUID format. Example: `550e8400-e29b-41d4-a716-446655440000`. Behind the scenes, it is handled as a normal string.

UserType

The user type enum is used to represent the type of a user. It has the following values:

- `user`: The user is a registered user.
- `organizer`: The user is a registered organizer.
- `admin`: The user is a registered administrator.

A user who is not authenticated will not have a type.

Order

The order type is used to represent the order of a list. It has the following values:

- `date_asc`: Orders the list by date in ascending order. Only available for events.
- `date_desc`: Orders the list by date in descending order. Only available for events.
- `title_asc`: Orders the list by title in ascending order. Only available for events.
- `title_desc`: Orders the list by title in descending order. Only available for events.
- `created_asc`: Orders the list by creation date in ascending order. Available for most entities.
- `created_desc`: Orders the list by creation date in descending order. Available for most entities.

Objects

Error

The error object is used to represent an error during an API request. The status code of the response will be set according to the error. It has the following structure:

```
{
  "error": {
    "message": "The error message"
  }
}
```

BaseUser

The base user object is used to represent a user in the database. It has the following properties:

- `id`: (`UUID`) The ID of the user.
- `name`: (`string`) The name of the user.
- `email`: (`string`) The email address of the user. This attribute is **not** available in a public context.
- `password`: (`string`) The password of the user in hashed form. This attribute is **never** returned but can be set.
- `type`: (`UserType`) The type of the user.
- `email_verified`: (`boolean`) Whether the email address of the user is verified or not. This attribute is **not** available in a public context.
- `onboarding`: (`boolean`) Whether the user has completed the onboarding process or not. This attribute is **not** available in a public context.

Admin

The admin object is used to represent an admin in the database. The following properties are available:

- `id`: (`UUID`) The ID of the admin.
- `base_user` (`BaseUser`) The base user object.

User

The user object is used to represent a user in the database. The following properties are available:

- `id`: (`UUID`) The ID of the user.
- `newsletter`: (`boolean`) Whether the user is subscribed to the newsletter or not. This attribute is **not** available in a public context.
- `base_user` (`BaseUser`) The base user object.

Organizer

The organizer object is used to represent an organizer in the database. The following properties are available:

- `id`: (`UUID`) The ID of the organizer.
- `description`: (`string` | `null`) The description of the organizer.
- `contact_email`: (`string` | `null`) The contact email address of the organizer.
- `phone`: (`string` | `null`) The phone number of the organizer.
- `website`: (`string` | `null`) The website of the organizer.
- `verified`: (`boolean`) Whether the organizer is verified or not.
- `avatar_image`: (`string` | `null`) The avatar image URI of the organizer.
- `banner_image`: (`string` | `null`) The banner image URI of the organizer.
- `base_user`: (`BaseUser`) The base user object.
- `location`: (`Location` | `null`) The location of the organizer.

Location

The location object is used to represent a location in the database. It has the following properties:

- `id`: (`UUID`) The ID of the location.
- `street`: (`string`) The street of the location.
- `street2`: (`string` | `null`) Additional street information of the location.
- `zip`: (`number`) The ZIP code of the location.
- `city`: (`string`) The city of the location.

Category

The category object is used to represent a category in the database. It has the following properties:

- `id`: (`UUID`) The ID of the category.
- `name`: (`string`) The name of the category.

Event

The event object is used to represent an event in the database. All event dates and times are in UTC, and the format is `YYYY-MM-DD` for dates and `HH:MM:SS` for times. It has the following properties:

- `id`: (`UUID`) The ID of the event.
- `title`: (`string`) The title of the event.
- `description`: (`string` | `null`) The description of the event.
- `start_date`: (`string` | `null`) The start date of the event. The format is `YYYY-MM-DD`.
- `end_date`: (`string` | `null`) The end date of the event. The format is `YYYY-MM-DD`.
- `start_time`: (`string` | `null`) The start time of the event. The format is `HH:MM:SS`.
- `end_time`: (`string` | `null`) The end time of the event. The format is `HH:MM:SS`.

- `public`: (`boolean`) Whether the event is public or not.
- `organizer`: (`Organizer`) The organizer of the event.
- `category`: (`Category` | `null`) The category of the event.
- `location`: (`Location` | `null`) The location of the event.
- `primary_image`: (`EventImage` | `null`) The primary image of the event.
- `event_images`: (`EventImage[]`) An array of event images of the event.

EventImage

The event image object is used to represent an image of an event in the database. It has the following properties:

- `id`: (`UUID`) The ID of the image.
- `image`: (`string`) The URI of the image.
- `event`: (`Event`) The event of the image.

Preference

The preference object is used to represent a preference in the database. It has the following properties:

- `user`: (`User`) The user of the preference.
- `category`: (`Category`) The category of the preference.

Endpoints

/auth

Handles authentication.

POST /auth/register

Registers a new user.

Access

- Everyone

Parameters

Body Parameters

- `name`: (`string`) Required. The name of the user.
- `email`: (`string`) Required. The email address of the user.
- `password`: (`string`) Required. The password of the user in plain text.
- `type`: (`UserType`) Required. The type of the user. Can be either `user` or `organizer` in this context.

Response

- `201 Created`
  - (`User` | `Organizer`) The user object if the user was created successfully.

- `400 Bad Request`
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an `Error` object.
- `409 Conflict`
  - If the email address is already in use.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the user could not be created.
  - Returns an `Error` object.

POST /auth/login

Logs in a user.

Access

- Everyone

Parameters

Body Parameters

- `email`: (`string`) Required. The email address of the user.
- `password`: (`string`) Required. The password of the user in plain text.

Response

- `200 OK`
  - (`Admin` | `User` | `Organizer`) The user object if the user was logged in successfully.
- `400 Bad Request`
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - If the specified user does not exist.
  - If the password is incorrect.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the user could not be logged in.
  - Returns an `Error` object.

POST /auth/logout

Logs out a user.

Access

- Everyone

Parameters

This endpoint does not accept any parameters.

Response

- `204 No Content`
  - (`null`) If the user was logged out successfully.
- `500 Internal Server Error`
  - If the user could not be logged out.
  - Returns an `Error` object.

PATCH /auth/send-email-verification

Sends an email verification email to the user.

Access

- Authenticated `User` and `Organizer` users

Parameters

This endpoint does not accept any parameters.

Response

- `204 No Content`
  - (`null`) If the email verification email was sent successfully.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `404 Not Found`
  - If the user does not exist.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the email verification email could not be sent.
  - Returns an `Error` object.

PATCH /auth/verify-email/:token

Verifies the email address of the user.

Access

- Everyone

Parameters

Path Parameters

- `token`: (`string`) Required. The email verification token to verify the email address.

Response

- `204 No Content`
  - (`null`) If the email address was verified successfully.
  - (`null`) If the email address was already verified.
- `400 Bad Request`
  - If the email verification token is not supplied.
  - If the email verification token is invalid.
  - If the email verification token has expired.
  - Returns an `Error` object.
- `404 Not Found`
  - If no user was found for the email verification token.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the email address could not be verified.
  - Returns an `Error` object.

/me

Handles the current user. All endpoints in this section require authentication, meaning that the session token in the current session must be valid.

GET /me

Gets the current user.

Access

- Authenticated `Admin`, `User`, and `Organizer` users

Parameters

This endpoint does not accept any parameters.

Response

- `200 OK`
  - (`Admin` | `User` | `Organizer`) The user object if the user was retrieved successfully.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `404 Not Found`
  - If the user does not exist.
  - Returns an `Error` object.

- `500 Internal Server Error`
  - If the user could not be retrieved.
  - Returns an `Error` object.

PUT /me

Updates the current user.

Access

- Authenticated `Admin`, `User`, and `Organizer` users

Parameters

Body Parameters

Note: The `BaseUser` attributes are inside a `base_user` object, including the `currentPassword` attribute.

- `name`: (`string`) Optional. The name of the user.
- `email`: (`string`) Optional. The email address of the user.
- `password`: (`string`) Optional. The password of the user in plain text.
- `currentPassword`: (`string`) Optional. The current password of the user in plain text. Required if `password` is specified, otherwise it will not update the password.
- `User` only:
  - `newsletter`: (`boolean`) Optional. Whether the user is subscribed to the newsletter or not.
  - `preferences`: (`{ id: UUID; value: boolean }[]`) Optional. The preferences of the user. The `id` is the ID of the category and the `value` is whether the user wants to enable the category or not.
- `Organizer` only:
  - `description`: (`string`) Optional. The description of the organizer.
  - `contact_email`: (`string`) Optional. The contact email address of the organizer.
  - `phone`: (`string`) Optional. The phone number of the organizer.
  - `website`: (`string`) Optional. The website of the organizer.
  - `location`: (`Location` | `null`) Optional. The location of the organizer.
  - `avatar_image`: (File | `null`) Optional. A file of the avatar image of the organizer. Requires a `Content-Type` of `multipart/form-data`.
  - `banner_image`: (File | `null`) Optional. A file of the banner image of the organizer. Requires a `Content-Type` of `multipart/form-data`.

Response

- `200 OK`
  - (`Admin` | `User` | `Organizer`) The user object if the user was updated successfully.
- `400 Bad Request`
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - If the current password is incorrect.
  - Returns an `Error` object.

- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `404 Not Found`
  - If the user does not exist.
  - Returns an `Error` object.
- `409 Conflict`
  - If the updated email address is already in use.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the user could not be updated.
  - Returns an `Error` object.

DELETE /me

Deletes the current user.

Access

- Authenticated `User`, and `Organizer` users

Parameters

Body Parameters

- `password`: (`string`) Required. The password of the user in plain text.

Response

- `204 No Content`
  - (`null`) If the user was deleted successfully.
- `400 Bad Request`
  - If the request body is missing any of the required parameters.
  - If the password is incorrect.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is an `Admin`.
  - Returns an `Error` object.
- `404 Not Found`
  - If the user does not exist.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the user could not be deleted.
  - Returns an `Error` object.

GET /me/preferences

Gets the preferences of the current user.

Access

- Authenticated `User` users

Parameters

This endpoint does not accept any parameters.

Response

- `200 OK`
  - (`Preference[]`) An array of preference objects if the preferences were retrieved successfully.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not a `User`.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the preferences could not be retrieved.
  - Returns an `Error` object.

GET /me/events

Gets all events, including private events, owned by the current user.

Access

- Authenticated `Organizer` users

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of events to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of events to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the events. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`

    - (`object`) An object containing the events if they were retrieved successfully. It has the following properties:
        - `events`: (`Event[]`) An array of event objects.
        - `hasMore`: (`boolean`) Whether there are more events to load.
- `401 Unauthorized`
    - If the user is not authenticated.
    - Returns an `Error` object.
- `403 Forbidden`
    - If the user is not an `Organizer`.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the events could not be retrieved.
    - Returns an `Error` object.

GET /me/events/past

Gets all past events, including private events, owned by the current user.

Access

- Authenticated `Organizer` users

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of events to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of events to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the events. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`
    - (`object`) An object containing the past events if they were retrieved successfully. It has the following properties:
        - `events`: (`Event[]`) An array of event objects.
        - `hasMore`: (`boolean`) Whether there are more events to load.
- `401 Unauthorized`
    - If the user is not authenticated.
    - Returns an `Error` object.
- `403 Forbidden`
    - If the user is not an `Organizer`.
    - Returns an `Error` object.
- `500 Internal Server Error`

    - If the events could not be retrieved.
    - Returns an `Error` object.

GET /me/events/future

Gets all ongoing and future events, including private events, owned by the current user.

Access

- Authenticated `Organizer` users

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of events to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of events to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the events. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`
    - (`object`) An object containing the ongoing and future events if they were retrieved successfully. It has the following properties:
        - `events`: (`Event[]`) An array of event objects.
        - `hasMore`: (`boolean`) Whether there are more events to load.
- `401 Unauthorized`
    - If the user is not authenticated.
    - Returns an `Error` object.
- `403 Forbidden`
    - If the user is not an `Organizer`.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the events could not be retrieved.
    - Returns an `Error` object.

GET /me/events/:id

Gets an event owned by the current user.

Access

- Authenticated `Organizer` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the event.

Response

- `200 OK`
  - (`Event`) The event object if the event was retrieved successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an `Organizer`.
  - If the event does not belong to the current user.
  - Returns an `Error` object.
- `404 Not Found`
  - If the event does not exist.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the event could not be retrieved.
  - Returns an `Error` object.

/admin

Handles admin actions.

GET /admin/organizers

Gets all organizers.

Access

- Authenticated `Admin` users

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of organizers to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of organizers to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the organizers. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.

Response

- `200 OK`
  - (`object`) An object containing the organizers if they were retrieved successfully. It has the following properties:
    - `organizers`: (`Organizer[]`) An array of organizer objects. The objects include an additional `event_count` property.
    - `hasMore`: (`boolean`) Whether there are more organizers to load.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an `Admin`.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the organizers could not be retrieved.
  - Returns an `Error` object.

GET /admin/organizers/unverified

Gets all unverified organizers.

Access

- Authenticated `Admin` users

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of organizers to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of organizers to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the organizers. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.

Response

- `200 OK`
  - (`object`) An object containing the unverified organizers if they were retrieved successfully. It has the following properties:
    - `organizers`: (`Organizer[]`) An array of organizer objects. The objects include an additional `event_count` property.
    - `hasMore`: (`boolean`) Whether there are more organizers to load.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an `Admin`.

    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the organizers could not be retrieved.
    - Returns an `Error` object.

PATCH /admin/organizers/:id/verify

Verifies an organizer.

Access

- Authenticated `Admin` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the organizer.

Response

- `204 No Content`
    - (`null`) If the organizer was verified successfully.
- `400 Bad Request`
    - If the ID is not supplied.
    - Returns an `Error` object.
- `401 Unauthorized`
    - If the user is not authenticated.
    - Returns an `Error` object.
- `403 Forbidden`
    - If the user is not an `Admin`.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the organizer could not be verified.
    - Returns an `Error` object.

PATCH /admin/organizers/:id/unverify

Unverifies an organizer.

Access

- Authenticated `Admin` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the organizer.

Response

- `204 No Content`
  - (`null`) If the organizer was unverified successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an `Admin`.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the organizer could not be unverified.
  - Returns an `Error` object.

GET /admin/users

Gets all users.

Access

- Authenticated `Admin` users

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of users to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of users to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the users. Defaults to a custom order specified by the backend.

Response

- `200 OK`
  - (`object`) An object containing the users if they were retrieved successfully. It has the following properties:
    - `users`: (`User[]`) An array of user objects.
    - `hasMore`: (`boolean`) Whether there are more users to load.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`

    - If the user is not an `Admin`.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the users could not be retrieved.
    - Returns an `Error` object.

GET /admin/admins

Gets all admins.

Access

- Authenticated `Admin` users

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of admins to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of admins to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the admins. Defaults to a custom order specified by the backend.

Response

- `200 OK`
    - (`object`) An object containing the admins if they were retrieved successfully. It has the following properties:
        - `admins`: (`Admin[]`) An array of admin objects.
        - `hasMore`: (`boolean`) Whether there are more admins to load.
- `401 Unauthorized`
    - If the user is not authenticated.
    - Returns an `Error` object.
- `403 Forbidden`
    - If the user is not an `Admin`.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the admins could not be retrieved.
    - Returns an `Error` object.

POST /admin/admins

Creates a new admin.

Access

- Authenticated `Admin` users

Parameters

Body Parameters

- `name`: (`string`) Required. The name of the admin.
- `email`: (`string`) Required. The email address of the admin.
- `password`: (`string`) Required. The password of the admin in plain text.

Response

- `201 Created`
  - (`Admin`) The admin object if it was created successfully.
- `400 Bad Request`
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an `Error` object.
- `409 Conflict`
  - If the email address is already in use.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the admin could not be created.
  - Returns an `Error` object.

DELETE /admin/admins/:id

Deletes an admin.

Access

- Authenticated `Admin` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the admin.

Response

- `204 No Content`
  - (`null`) If the admin was deleted successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.

- `403 Forbidden`
  - If the user is not an `Admin`.
  - If the ID is the same as the current user's ID.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the admin could not be deleted.
  - Returns an `Error` object.

/categories

Handles categories.

GET /categories

Gets all categories.

Access

- Everyone

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of categories to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of categories to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the categories. Defaults to a custom order specified by the backend.

Response

- `200 OK`
  - (`object`) An object containing the categories if they were retrieved successfully. It has the following properties:
    - `categories`: (`Category[]`) An array of category objects.
    - `hasMore`: (`boolean`) Whether there are more categories to load.
- `500 Internal Server Error`
  - If the categories could not be retrieved.
  - Returns an `Error` object.

GET /categories/:id

Gets a category.

Access

- Everyone

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the category.

Response

- `200 OK`
  - (`Category`) The category object if the category was retrieved successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `404 Not Found`
  - If the category does not exist.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the category could not be retrieved.
  - Returns an `Error` object.

POST /categories

Creates a new category.

Access

- Authenticated `Admin` users

Parameters

Body Parameters

- `name`: (string) Required. The name of the category.

Response

- `201 Created`
  - (`UUID`) The ID of the category if the category was created successfully.
- `400 Bad Request`
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an administrator.

    - Returns an `Error` object.
- `409 Conflict`
    - If the category already exists.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the category could not be created.
    - Returns an `Error` object.

PUT /categories/:id

Updates a category.

Access

- Authenticated `Admin` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the category.

Body Parameters

- `name`: (string) Required. The name of the category.

Response

- `200 OK`
    - (`Category`) The category object if the category was updated successfully.
- `400 Bad Request`
    - If the ID is not supplied.
    - If the request body is missing any of the required parameters.
    - If the request body contains invalid parameters.
    - Returns an `Error` object.
- `401 Unauthorized`
    - Returns an `Error` object.
- `403 Forbidden`
    - If the user is not an administrator.
    - Returns an `Error` object.
- `409 Conflict`
    - If the category already exists.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the category could not be updated.
    - Returns an `Error` object.

DELETE /categories/:id

Deletes a category.

Access

- Authenticated `Admin` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the category.

Response

- `204 No Content`
  - (`null`) If the category was deleted successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an administrator.
  - Returns an `Error` object.
- `409 Conflict`
  - If the category is in use by an event.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the category could not be deleted.
  - Returns an `Error` object.

/events

Handles events.

GET /events

Gets all public events, which are ongoing or in the future.

Access

- Everyone

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of events to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of events to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the events. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`
  - (`object`) An object containing the events if they were retrieved successfully. It has the following properties:
    - `events`: (`Event[]`) An array of event objects.
    - `hasMore`: (`boolean`) Whether there are more events to load.
- `500 Internal Server Error`
  - If the events could not be retrieved.
  - Returns an `Error` object.

GET /events/ongoing

Gets all public events, which are ongoing.

Access

- Everyone

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of events to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of events to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the events. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`
  - (`object`) An object containing the events if they were retrieved successfully. It has the following properties:
    - `events`: (`Event[]`) An array of event objects.
    - `hasMore`: (`boolean`) Whether there are more events to load.
- `500 Internal Server Error`

    - If the events could not be retrieved.
    - Returns an `Error` object.

GET /events/future

Gets all public events, which are in the future.

Access

- Everyone

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of events to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of events to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the events. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`
    - (`object`) An object containing the events if they were retrieved successfully. It has the following properties:
        - `events`: (`Event[]`) An array of event objects.
        - `hasMore`: (`boolean`) Whether there are more events to load.
- `500 Internal Server Error`
    - If the events could not be retrieved.
    - Returns an `Error` object.

GET /events/:id

Gets an event.

Access

- Everyone

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the event.

Response

- `200 OK`
  - (`Event`) The event object if the event was retrieved successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `404 Not Found`
  - If the event does not exist, is private or the organizer is not verified.
    - The `Organizer` of the event and `Admin` users can view the event even if it is private or the organizer is not verified.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the event could not be retrieved.
  - Returns an `Error` object.

POST /events

Creates a new event.

Access

- Authenticated `Organizer` users

Parameters

Body Parameters

- `title`: (`string`) Required. The title of the event.
- `description`: (`string` | `null`) Optional. The description of the event. Required if `public` is set to `true`.
- `start_date`: (`string` | `null`) Optional. The start date of the event. Required if `public` is set to `true`.
- `end_date`: (`string` | `null`) Optional. The end date of the event.
- `start_time`: (`string` | `null`) Optional. The start time of the event.
- `end_time`: (`string` | `null`) Optional. The end time of the event.
- `public`: (`boolean`) Optional. Whether the event is public or not. If set to `true`, `description`, `start_date`, and `category_id` are required.
- `category_id`: (`UUID` | `null`) Optional. The ID of the category of the event. Required if `public` is set to `true`.
- `location`: (`Location` | `null`) Optional. The location of the event.
- `event_images`: (`File[]` | `null`) Optional. An array of files including the images of the event. Requires a `Content-Type` of `multipart/form-data`.

Response

- `201 Created`
  - (`UUID`) The ID of the event if the event was created successfully.
- `400 Bad Request`

    - If the request body is missing any of the required parameters.
    - If the request body contains invalid parameters.
    - If the organizer is not verified and the event is set to public.
    - If the total number of event images exceeds the maximum allowed.
    - Returns an `Error` object.
- `401 Unauthorized`
    - If the user is not authenticated.
    - Returns an `Error` object.
- `403 Forbidden`
    - If the user is not an `Organizer`.
    - Returns an `Error` object.
- `404 Not Found`
    - If the organizer does not exist.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the event could not be created.
    - Returns an `Error` object.

PUT /events/:id

Updates an event.

Access

- Authenticated `Organizer` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the event.

Body Parameters

- `title`: (`string`) Optional. The title of the event.
- `description`: (`string` | `null`) Optional. The description of the event. Required if `public` is set to `true` and not already set.
- `start_date`: (`string` | `null`) Optional. The start date of the event. Required if `public` is set to `true` and not already set.
- `end_date`: (`string` | `null`) Optional. The end date of the event.
- `start_time`: (`string` | `null`) Optional. The start time of the event.
- `end_time`: (`string` | `null`) Optional. The end time of the event.
- `public`: (boolean) Optional. Whether the event is public or not. If set to `true`, `description`, `start_date`, and `category_id` must either be already set or provided.
- `category_id`: (`UUID`) Optional. The ID of the category of the event. Required if `public` is set to `true` and not already set.
- `primary_image_id`: (`UUID` | `null`) Optional. The ID of the primary image of the event.

- `location`: (`Location` | `null`) Optional. The location of the event.
- `event_images`: (`File[]`) Optional. An array of files including the images of the event. Requires a `Content-Type` of `multipart/form-data`.

Response

- `200 OK`
  - (`Event`) The event object if the event was updated successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - If the organizer is not verified and the event is set to public.
  - If the total number of event images exceeds the maximum allowed.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an `Organizer`.
  - If the owner of the event is not the current user.
  - Returns an `Error` object.
- `404 Not Found`
  - If the event does not exist.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the event could not be updated.
  - Returns an `Error` object.

DELETE /events/:id

Deletes an event.

Access

- Authenticated `Organizer` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the event.

Response

- `204 No Content`
  - (`null`) If the event was deleted successfully.

- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an `Organizer`.
  - If the owner of the event is not the current user.
  - Returns an `Error` object.
- `404 Not Found`
  - If the event does not exist.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the event could not be deleted.
  - Returns an `Error` object.

DELETE /events/image/:id

Deletes an image of an event.

Access

- Authenticated `Organizer` users

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the event image.

Response

- `204 No Content`
  - (`null`) If the event image was deleted successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `401 Unauthorized`
  - If the user is not authenticated.
  - Returns an `Error` object.
- `403 Forbidden`
  - If the user is not an `Organizer`.
  - If the owner of the associated event is not the current user.
  - Returns an `Error` object.
- `404 Not Found`
  - If the event image does not exist.

    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the event image could not be deleted.
    - Returns an `Error` object.

/organizers

Handles organizers.

GET /organizers

Gets all verified organizers.

Access

- Everyone

Parameters

Query Parameters

- `limit`: (`number`) Optional. The maximum number of organizers to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of organizers to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the organizers. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.

Response

- `200 OK`
    - (`object`) An object containing the organizers if they were retrieved successfully. It has the following properties:
        - `organizers`: (`Organizer[]`) An array of organizer objects.
        - `hasMore`: (`boolean`) Whether there are more organizers to load.
- `500 Internal Server Error`
    - If the organizers could not be retrieved.
    - Returns an `Error` object.

GET /organizers/:id

Gets a verified organizer.

Access

- Everyone

Parameters

Path Parameters

- `id`: (`UUID`) Required. The ID of the organizer.

Response

- `200 OK`
  - (`Organizer`) The organizer object if the organizer was retrieved successfully.
- `400 Bad Request`
  - If the ID is not supplied.
  - Returns an `Error` object.
- `404 Not Found`
  - If the organizer does not exist or is not verified.
    - The `Organizer` themselves and `Admin` users can view the organizer even if the organizer is not verified.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the organizer could not be retrieved.
  - Returns an `Error` object.

/search

Handles search.

GET /search/:query

Searches for events and organizers.

Access

- Everyone

Parameters

Path Parameters

- `query`: (`string`) Required. The search query.

Query Parameters

- `limit`: (`number`) Optional. The maximum number of results to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of results to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the results. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`
  - (`object`) An object containing the results. It has the following properties:
    - `events`: (`object`) An object containing the events. It has the following properties:
      - `events`: (`Event[]`) An array of event objects.
      - `hasMore`: (`boolean`) Whether there are more events to load.
    - `organizers`: (`object`) An object containing the organizers. It has the following properties:
      - `organizers`: (`Organizer[]`) An array of organizer objects.
      - `hasMore`: (`boolean`) Whether there are more organizers to load.
- `400 Bad Request`
  - If the query is not supplied.
  - Returns an `Error` object.
- `500 Internal Server Error`
  - If the results could not be retrieved.
  - Returns an `Error` object.

GET /search/events/:query

Searches for events.

Access

- Everyone

Parameters

Path Parameters

- `query`: (`string`) Required. The search query.

Query Parameters

- `limit`: (`number`) Optional. The maximum number of results to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of results to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the results. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.
- `organizer_id`: (`UUID[]`) Optional. The IDs of the organizers to filter by.

Response

- `200 OK`
  - (`object`) An object containing the results if the events were retrieved successfully. It has the following properties:
    - `events`: (`Event[]`) An array of event objects.
    - `hasMore`: (`boolean`) Whether there are more events to load.
- `400 Bad Request`

    - If the query is not supplied.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the events could not be retrieved.
    - Returns an `Error` object.

GET /search/organizers/:query

Searches for organizers.

Access

- Everyone

Parameters

Path Parameters

- `query`: (`string`) Required. The search query.

Query Parameters

- `limit`: (`number`) Optional. The maximum number of results to return. Defaults to unlimited.
- `offset`: (`number`) Optional. The number of results to skip. Defaults to `0`.
- `order`: (`Order`) Optional. The order of the results. Defaults to a custom order specified by the backend.
- `category_id`: (`UUID[]`) Optional. The IDs of the categories to filter by.

Response

- `200 OK`
    - (`object`) An object containing the results if the organizers were retrieved successfully. It has the following properties:
        - `organizers`: (`Organizer[]`) An array of organizer objects.
        - `hasMore`: (`boolean`) Whether there are more organizers to load.
- `400 Bad Request`
    - If the query is not supplied.
    - Returns an `Error` object.
- `500 Internal Server Error`
    - If the organizers could not be retrieved.
    - Returns an `Error` object.

Chapter 16

Task

OST
Ostschweizer
Fachhochschule

Aufgabenstellung Bachelorarbeit „What's up in RJ“

Autor: Frank Koch

Version: 2.0 – vereinbarte Version

Anpasst am: 29.02.24

1. Beteiligte Personen

- Studierende: Michael Enzler, Fabio Stocker
- Industriepartner: AdaptIT GmbH, Michael Güntensperger
- Experte: Hansjörg Huser
- Gegenleser: Cyrill Brunschwiler
- Betreuer: Frank Koch

2. Problembeschrieb

In Rapperswil-Jona gibt es ein vielfältiges Kulturprogramm. Um nichts zu verpassen, muss man regelmässig die Zeitung durchsuchen und sich bei den Newslettern der jeweiligen Veranstalter anmelden. Schnell geht die Übersicht verloren und viele Personen in der Region klagen, dass es kein rechtes Abendprogramm gibt.

Es wurde bereits eine Applikation mit ersten Funktionen wie der Signup, Login, das Veröffentlichen von Event und einer Suche implementiert.

Ziel ist, diese Applikation um weitere Module und Funktionen zu erweitern.

3. Aufgabenstellung

Der Fokus bei der Weiterentwicklung liegt auf den noch nicht existierenden Admin-Funktion für die Verwaltung aller Benutzer und Organisationen, wie auch eine komplette Neuentwicklung des Frontends. Neu soll auch der Upload von Bildern in einen Object Store ermöglicht werden. Organisationen sollen in Zukunft zusätzlich die Möglichkeit haben, ein eigenes Profil zu erstellen, welches auf der Hauptseite angezeigt wird.

Gerne werden die Interessen und Ideen der Studierenden berücksichtigt.

Technische Umgebung

Für die Umsetzung wird mit Web-Technologien gearbeitet.

- Frontend: Angular
- Backend: Node.js
- Datenbank: Postgres
- Hoster: DigitalOcean

Funktionale Anforderungen

- Admin-Bereich um die Informationen von Benutzern und Organisationen zu bearbeiten
- Entwicklung eines responsive Frontends
- Upload von Bildern in einen Object Store
- Profil-Seite pro Organisation

Optionale Anforderungen

- Freigabeprozess von Events durch Admin
- Dashboard für Veranstaltende mit Auswertungen z.B. Buttonklicks
- AI um auf Basis von gewählten Interessen, Newsletter mit Events zusammenzustellen
- GPT-API um Eventbeschreitungen zu generieren
- Onboarding-Prozess mit Interessensauswahl
- Kartenansicht mit allen Events
- Button für Ticketkauf

Nicht-Funktionale Anforderungen

- Das Entwicklerteam implementiert die Features gemäss den mit dem Kunden vereinbarten Prioritäten.
- Das Backend sollte 1’000 Requests pro Minute verarbeiten können
- Jede Seite sollte nicht länger als 200ms für das Laden benötigen

- Die Seite soll sowohl auf dem Mobile, Tablet und Desktop gut aussehen (Responsive)
- Die Web-Applikation sollte auf Firefox, Chrome und Safari laufen.
- Via Internet sollte auf eine vom Kunden zur Verfügung gestellte Domain zugegriffen werden können.
- Drei von vier Test-Usern sollten das UI (Kategorien: Layout, Responsiveness, Colour, Content) der Applikation mit einem Tablet mit einer Note von mindestens 8 von 10 bewerten, mit 10 als bester Note.
- Die Datenbank soll bis zu 10'000 Dokumente und 100 Benutzende managen können.
- Errors sollen keine Systemfehler erzeugen, stattdessen eine Fehlermeldung anzeigen und das System auf den vorherigen Zustand zurücksetzen.
- Jeder Error soll im System geloggt werden
- Jede Kommunikation zwischen Front- und Backend soll mit einem SSL-Zertifikat verschlüsselt werden.
- Daten welche in Eingabefelder abgefüllt werden, sollen zuerst validiert werden, bevor diese durch das System verarbeitet werden. SQL Injection Tests der Eingabefelder dürfen keine Verletzlichkeiten zeigen.
- Die Webapplikation soll Datenschutz-konform umgesetzt werden.
- User-Passwörter werden nicht in Plain-Text in der Datenbank gespeichert.
- Wenn sich ein User in die Web-Applikation einloggt, werden ihm nur Daten angezeigt, auf die er Zugriff haben soll.
- Businesslogik im Backend soll modular aufgebaut werden, so dass sie erweitert werden kann.
- Das Backend-API soll durch ein API-Testing Tool überprüft werden.
- Implementierte Funktionalitäten (Datenbank, Backend, Frontend,…) sollen deployed werden.

4. Zur Durchführung

Mit dem Betreuer finden Besprechungen gemäss Absprache statt. Die Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren. Die Kommunikation mit dem Betreuer findet primär über E-Mail statt.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. Abweichungen vom Projektplan sind rechtzeitig mit dem Betreuer zu besprechen.

5. Dokumentation und Abgabe

Siehe «Leitfaden für Bachelor- und Studienarbeiten Version 1.2» Abschnitt 5.5 "Umfang und Form der Abgabe".

6. Termine

Siehe veröffentlichte «Termine BA FS24».

7. Bewertung

Siehe «Leitfaden für Bachelor- und Studienarbeiten Version 1.2» Abschnitt 6 "Bewertung".

Rapperswil, den 12.02.24

Frank Koch