HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Naturmuseum
St.Gallen

# maaas

## Museums-App as a Service

# Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2015


| | |
|---|---|
| Autoren: | Daniel Keller |
| | Samuel Müller |
| Betreuer: | Prof. Dr. Markus Stolze |
| Co-Referent: | Prof. Dr. Peter Heinzmann |
| Externer Experte: | Remo Brunschweiler, Namics |
| | |
| Projektpartner: | Naturmuseum St. Gallen |

# 1 Abstract

maaas
Museums-App as a Service

17.12.2015

Das Naturmuseum St. Gallen besitzt eine Mobile-App, welche einen interaktiven Museumsrundgang ermöglicht. Die App erlaubt es, auf Grund der aktuellen Position des Benutzers passende Zusatzinformationen und Quiz anzuzeigen.

Da alle Inhalte der App in Files abgelegt sind, welche mit der App ausgeliefert werden, gibt es keinen einfachen Weg für das Museum, Anpassungen vorzunehmen. Diese werden jeweils von den Entwicklern gemacht und können nur mit einem neuen Release der App veröffentlicht werden.

In dieser Arbeit ging es darum, ein Content Management System zu entwickeln, welches das Editieren dieser Inhalte erlaubt. Es wurde eine Vielzahl von Anforderungen evaluiert und umgesetzt. Dazu gehören ein integrierter Rich Text Editor, Bilder-Uploads, Mehrsprachigkeit der Inhalte, zielgruppengerechte Rundgänge, eine Live-Vorschau der App im Browser sowie das Sammeln von Statistiken.

Mit dem Gedanken, die Gesamtlösung in andere Museen verfügbar zu machen, wurde es als Software as a Service umgesetzt. Die Idee ist, dass sich beliebige Museen auf der Plattform anmelden und individuelle Inhalte für ein App erstellen können.

Für das CMS wurde AngularJS und Ruby on Rails eingesetzt. Die Applikation wurde so konzipiert, dass Sie einfach auf die zukunftsträchtige Nachfolgerversion Angular2 migrierbar ist. Das System ist in der Cloud gehostet, namentlich auf Heroku und Cloudinary.

# Inhaltsverzeichnis I

# Inhaltsverzeichnis II

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 2 Aufgabenstellung



## Aufgabenstellung Bachelorarbeit Abteilung I, HS 2015/16
## Samuel Mueller, Daniel Keller

## *MuseumsApp as a Service*

### 1. Betreuer & Auftraggeber

*Betreuer dieser Arbeit ist*
Prof. Dr. Markus Stolze, Institut für Software mstolze@hsr.ch

*Co-Referent für diese Arbeit ist*
Prof. Dr. Peter Heinzmann, pheinzma@hsr.ch

*Externer Experte für diese Arbeit ist*
Remo Brunschwiler, Namics, remo.brunschwiler@namics.com

*Anwendungspartner dieser Arbeit ist das Naturkunde Museum St. Gallen*
Dr. Toni Bürgin
Museumstrasse 32
CH-9000 St.Gallen
toni.buergin@naturmuseumsg.ch
Tel: 071 242 06 86

### 2. Ausgangslage

Im Rahmen einer Semesterarbeit wurde eine individuelle Mobile-App für das Naturmuseum St. Gallen umgesetzt. Die App erlaubt es, im Museum ortsbezogene Inhalte darzustellen. Ein Benutzer kann sich frei durch das Museum bewegen, wobei ihm je nach Standort entsprechende Inhalte zur Verfügung gestellt werden. Die Ortung basiert auf der iBeacon-Technologie, bei der kleine, batteriebetriebene Sender zum Einsatz kommen. Die dargestellten Inhalte bestehen aus Text-, Bild-, Audio- und Video-Material. Ausserdem kann pro Ausstellungsobjekt jeweils ein Quiz gelöst werden.

Das Ziel der Semesterarbeit war es, das Potential von iBeacons in Museumsausstellungen zu evaluieren. Die App wurde am Museumstag im Naturmuseum St. Gallen mit 48 Museumsbesuchern getestet. Das Feedback war grösstenteils positiv und es zeigte sich, dass das Konzept durchaus auch in anderen Museen zum Einsatz kommen könnte. Aus diesem Grund entschied man sich, die Idee weiter zu entwickeln. Das Naturmuseum St. Gallen hat sich bereit erklärt, weiterhin als Anwendungspartner zu agieren.

Eine wichtige Funktionalität, welche in der bisherigen Arbeit nicht umgesetzt wurde, ist die selbständige Pflege der Inhalte durch das Museum. Die Inhalte werden zur Zeit statisch mit der App ausgeliefert und können nur von den App-Entwicklern editiert werden.

## 3. Ziele der Arbeit

Ziel dieser Bachelorarbeit ist es, ein Content-Management-System (CMS) für Museums-Apps zu entwickeln. Die relevanten Inhalte sollen editierbar gemacht werden. Dies beinhaltet darstellungsspezifische Eigenschaften wie zum Beispiel Hintergrundbilder oder Farbschemen, Text-, Bild-, Audio- und Video-Material so wie Quiz-Fragen und -Antworten. Das CMS soll als Platform as a Service (PaaS) angeboten werden. Das System soll Museen in die Lage versetzen schnell und kosteneffizient eine eigene Museums-App zu erstellen und deren Inhalt aktuell zu halten.

Ausserdem sollen einige Verbesserungen an der aktuellen Museums-App vorgenommen werden. Diese basieren auf den Rückmeldungen des Pilottages. Für die Quiz-Aktivierung sollen QR-Codes statt Beacons zum Einsatz kommen. Zudem sollen die Rundgänge in verschiedenen Sprachen und Schwierigkeitsstufen angeboten werden.

## 4. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollen den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in einem Exemplar abzugeben (Exemplar für das Sekretariat Informatik) Sowie ein Download-Link für Prof. Stolze und weitere Exemplare nach Absprache mit dem Co-Referenten (Heinzmann) und dem Experten (Brunschwiler)

Zudem ist eine kurze Projektresultatdokumentation im öffentlichen Wiki von Prof. M. Stolze zu erstellen.

## 5. Weitere Regeln und Termine

Im Weiteren gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten „Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik" (HSR Intranet)
https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html )

Der Terminplan ist hier ersichtlich (HSR Intranet)
https://www.hsr.ch/Termine-Bachelor-und-Studiena.5142.0.html

## 6. Rechte

Die resultierende Software und Dokumentation darf vom Museum St. Gallen, der HSR und den Studierenden genutzt und weiterentwickelt werden. Kosten für Nutzung des CMS Service werden in diesem Dokument nicht geregelt. Die HSR darf die Software zu Schulungszwecken und zur Werbung demonstrieren und nutzen. Die Arbeit (ohne geheime Anhänge) wird veröffentlich. Titel, Abstract und Auftraggeber der Arbeit dürfen von der HSR und Studierenden schon während der Arbeit kommuniziert werden.

## 7. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 350h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 25h pro Woche (auf 14 Wochen) und damit ca. 3 Tage Arbeit pro Woche pro Student.

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

Die Aufgabenstellung wurde am 16.9.2015 vorbesprochen.
Die definitive Aufgabenstellung wurde am XX.10.2015 beschlossen.

Rapperswil, 14.10.2015

Prof. Dr. Markus Stolze
Institut für Software
Hochschule für Technik Rapperswil

# 3 Vereinbarung

## 3.1 Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit Museums von Daniel Keller und Samuel Müller unter der Betreuung von Dr. Prof. Markus Stolze geregelt.

## 3.2 Urheberrecht

Die Urheberrechte stehen der Studentin / dem Student zu.

## 3.3 Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von der Studentin / dem Student wie von der HSR nach Abschluss der Arbeit verwendet und weiter entwickelt werden

Rapperswil, den 16. Dez 15 .....
Die Studentin/der Student

Rapperswil, den 16. Dez 15 .....
Die Studentin/der Student

Rapperswil, den 16 Dez 15 .....
Der Betreuer / die Betreuerin der Studienarbeit

Rapperswil, den 16 Dez 15 .....
Der Studiengangleiter / die Studiengangleiterin

# 4 Management Summary

## 4.1 Ausgangslage

Im Rahmen einer Semesterarbeit [1] wurde eine App für das Naturmuseum St. Gallen entwickelt. Mit Hilfe der iBeacon-Technologie ermöglicht die App einen interaktiven Museums-Rundgang. Es können lokationsbasierte Zusatzinformationen zu Ausstellungsobjekten geladen sowie Quiz gelöst werden.

Alle Inhalte der App (Zusatzinformationen, Bilder, Quiz) werden direkt mit der App veröffentlicht. Die Inhalte sind in lokalen Files abgelegt. Dies hat zur Folge, dass Anpassungen an den Inhalten nicht direkt vorgenommen werden können.

Ziel dieser Bachelor-Arbeit war es, ein Content Management System (CMS) zu entwickeln, welches das Erstellen und Warten jener Inhalte unterstützen soll. Da der Einsatz dieses Konzeptes (App, Beacons, CMS) auch in anderen Museen denkbar ist, sollte das CMS als Software as a Service (SaaS) entwickelt werden. Damit könnten Museen mit wenig Aufwand eine lokationsbasierte Museums-App einführen.

Das Naturmuseum St. Gallen hat sich bereit erklärt, auch für die Bachelorarbeit als Projektpartner zur Verfügung zu stehen und das CMS von Mitarbeitern testen zu lassen.

**iBeacon**
Bei der iBeacon-Technologie handelt es sich um ein von Apple eingeführtes Protokoll, welches auf Bluetooth Low Energy (BLE) basiert. Ein Beacon ist ein kleiner, batteriebetriebener Sender, welcher in regelmässigen Intervallen BLE-Signale aussendet, welche von mobilen Geräten empfangen werden können. Mit diesen Signalen kann der ungefähre Abstand des Gerätes zum Sender (Beacon) ermittelt werden. Damit wird eine universelle und batterieschonende Indoor-Lokalisierungstechnologie ermöglicht.

### 4.1.1 Abgrenzung Semesterarbeit

Inhalt dieser Bachelorarbeit war es, das CMS zu entwickeln und die Mobile-App mit entsprechenden Anpassungen zu versehen. Eine detaillierte Dokumentation der App kann dem Bericht der Semesterarbeit entnommen werden. [1]

## 4.2 Vorgehen, Technologien

### 4.2.1 Vision

Es sollte ein CMS entwickelt werden, welches den Mitarbeitern eines Museums erlaubt, Inhalte der App zentral zu verwalten. Das CMS soll in der Cloud gehostet werden und mandantenfähig sein. So kann sich ein Museum bei dem Service registrieren und im CMS folgende Aufgaben wahrnehmen:

- Editieren der Museumsinformationen
- Erstellen und Editieren von Inhalten und Quiz
- Verwalten von Beacons und Verknüpfen mit einem Bereich
- Übersetzen sämtlicher Elemente
- Live-Vorschau der App im Browser

Aus der Evaluation der Semesterarbeit hat sich zudem ergeben, dass zielgruppengerechte Inhalte gewünscht sind. Daraufhin wurde das Konzept der Rundgänge erarbeitet. Ein Museum soll einen Rundgang erstellen können und diesem Zusatzinhalte und Quiz zuordnen. So lässt sich zum Beispiel ein Familienrundgang mit einfachen Inhalten und Quiz für Kinder erstellen und ein Rundgang für Oberstufenklassen mit detaillierteren Inhalten und fordernden Quiz.

### 4.2.2 Lösungsansatz und eingesetzte Technologien

Für das User-Interface des CMS wurde auf AngularJS gesetzt. Es wurde darauf geachtet, eine zukunftssichere Lösung zu entwickeln welche neuste Konzepte der Webentwicklung verwendet. Ziel war es, eine mögliche Migration auf die angekündigte Version 2 von AngularJS so reibungslos wie möglich zu gestalten.

Ein Backend mit Datenbankanbindung wurde in Ruby on Rails entwickelt. Das Backend wird vom CMS sowie auch von der App verwendet.

Beide Systeme wurden in der Cloud gehostet. Es wurden zwei etablierte Anbieter verwendet: Heroku für den Web-Service sowie Cloudinary für das Hosten von Bildern.

## 4.3 Ergebnisse

Das CMS wurde entwickelt und von der Museumspädagogin des Naturmuseums St. Gallen auf Benutzbarkeit getestet. Das Feedback war insgesamt positiv, Mängel wurden entsprechend behoben.

Die aktuellen App-Inhalte des Naturmuseum St. Gallen wurden in das CMS übernommen. Diese Inhalte sind nun editierbar und werden von der App geladen.

Das System wurde wie geplant als Software as a Service (SaaS) umgesetzt. Alle Komponenten sind in der Cloud gehostet und dadurch beliebig skalierbar. Benutzer können sich selbständig registrieren und Museen erfassen. An dieser Stelle muss man bedenken, dass für eine erfolgreiche Nutzung des Service die Installation von Beacons in jeweiligen Museen notwendig ist.

Es wurde eine umfangreiche Palette von Features umgesetzt. Inhalte können in einem Rich-Text-Editor im Browser verwaltet werden, es können Bilder hochgeladen werden, es stehen 5

verschiedene Quiz-Typen zur Verfügung und es können Rundgänge zusammengestellt werden. Alle Inhalte sind auf Deutsch, Englisch, Französisch und Italienisch übersetzbar. Die App kann online in einer Preview-Ansicht geladen werden, um die erfassten Inhalte zu testen. In der App werden Statistiken gesammelt, welche in Google Analytics analysiert werden können.


*Abbildung 4-1 Logo maaas*

Die App wurde so erweitert, dass sämtliche Daten aus dem Backend geladen werden und nicht mehr direkt mit der App ausgeliefert werden. Auf das angedachte QR-Code Feature für die Quiz Aktivierung wurde vorerst verzichtet. Grund dafür war, dass man sich auf das CMS und den SaaS Ansatz konzentrieren wollte. Das Konzept für Multimediale Inhalte steht, beschränkt sich jedoch noch auf Bilder.

### 4.3.1 Business Evaluation

Um herauszufinden wie gross das Interesse an der *maaas* Gesamtlösung sein könnte wurde eine Business-Evaluation durchgeführt. Dazu wurde ein Fragebogen zusammengestellt und an diverse Museen der Deutschschweiz gesendet. Ziel war es, die aktuelle Situation und grundsätzliche Einstellung bezüglich Museums-App zu bestimmen. Es hat sich ergeben, dass vor allem mittelgrosse Museen sich aktuell Gedanken über das Einführen einer App machen. Häufig ist aber die Kostenfrage ein Gegenargument. Wir kamen zum Schluss, dass es noch nicht zu spät wäre in diesem Markt einzusteigen. Mit einem kostengünstigen SaaS Angebot könnten vor allem diese mittelgrossen Museen als mögliche Kunden überzeugt werden. Auch könnte man die App in anderen Szenarien als nur Museen einsetzen. Zum Beispiel in Zoos oder anderen, öffentlich zugänglichen Rundgängen wie der Wasservogel-Parcours Rapperswil.

Details zur Business-Analyse können dem Anhang *6.7 Business Evaluation* entnommen werden.

## 4.4 Ausblick

Es ist klar, dass das entwickelte System Potential hat, um kommerziell weitergeführt zu werden. Der Business-Analyse ist zu entnehmen, dass es verschiedene Museen gibt, die sich über die Einführung einer Mobile-App Gedanken machen. Auch das Naturmuseum St. Gallen ist nach wie vor interessiert.

Einen kommerziellen Nutzen zu erzielen war von Anfang an eine Vision der Autoren dieser Arbeit. Dass das System funktionieren könnte, wurde mit dem entwickelten Prototyp bewiesen. Allerdings gibt es einiges zu bedenken, sollte die Arbeit weitergeführt werden.

Zum Beispiel wären Besuche bei einführenden Museen nötig, um Beacons zu installieren. Auch regelmässige Wartungsarbeiten würden einen grossen Aufwand erzeugen. Ausserdem müsste die Sicherheit des Systems auf Herz und Nieren getestet werden. Es müssten Backup-Lösungen erarbeitet werden, um die Archivierung der von Museen erfassten Inhalte zu gewährleisten.

Die Liste der Notwendigkeiten für einen kommerziellen Erfolg dieses Projektes ist noch ein ganzes Stück länger. Wichtig ist, dass mit dem hier entwickelten Prototyp eine gute Grundlage geschaffen wurde und eine Weiterführung durchaus denkbar wäre.

# 4.5 Technischer Bericht

## 4.5.1 Systemübersicht

Die Gesamtlösung kann in vier Komponenten aufgeteilt werden: Backend mit Datenbank, das CMS, Cloudinary [2] und die App.

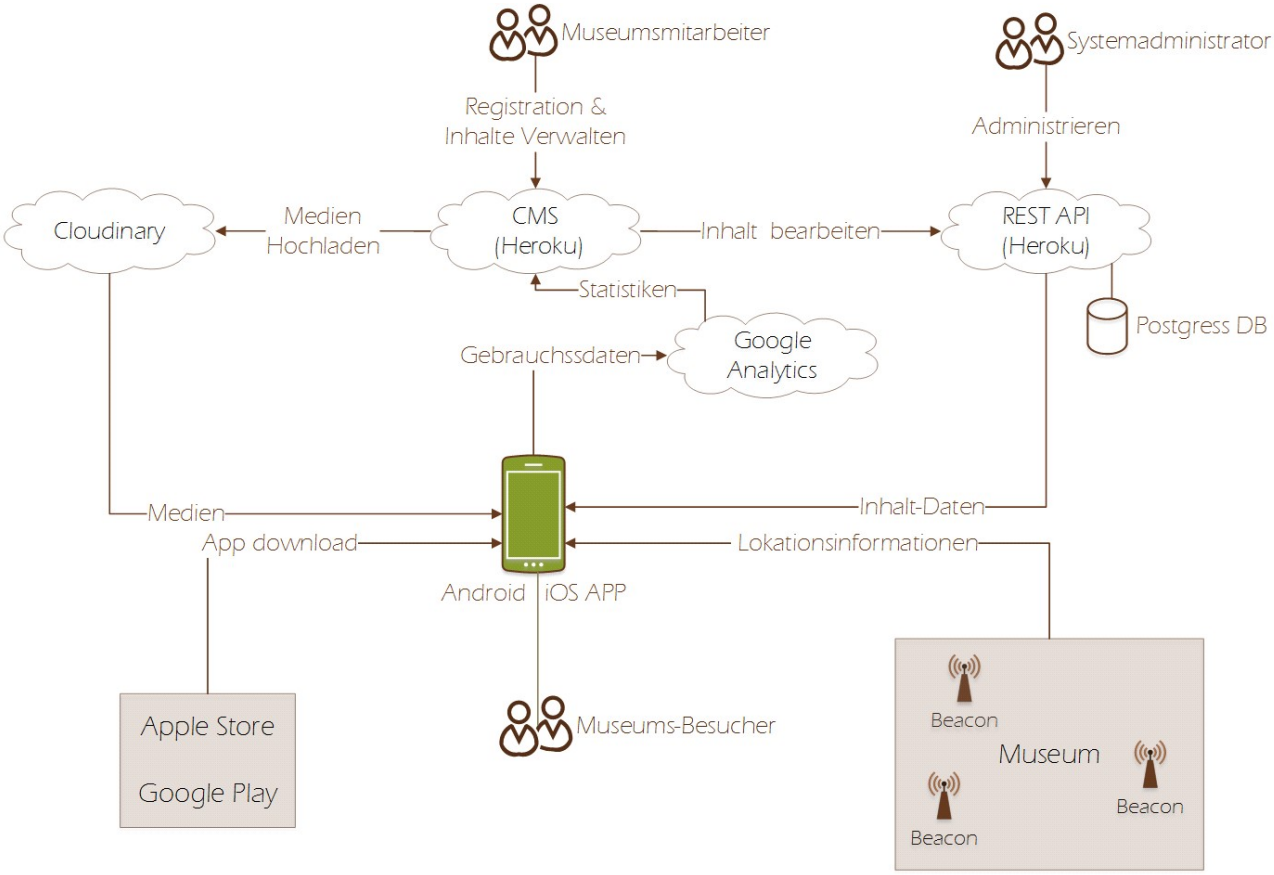



*Abbildung 4-2 Systemübersicht*

### 4.5.1.1 CMS

Das CMS sollte als Web-Applikation entwickelt werden, damit die Benutzer betriebssystemunabhängig darauf zugreifen können und keine lokalen Komponenten vorausgesetzt sind. Durch die Mandantenfähigkeit des CMS können alle Museen, welche den Service beanspruchen, auf das gleiche System zugreifen. Mit einem Login wird sichergestellt, dass man nur die eigenen Daten einsehen und bearbeiten kann.

Damit bei wechselnder Kundenanzahl flexibel skaliert werden kann, wurde entschieden, das CMS in der Cloud zu hosten. Da Heroku ein etablierter Anbieter ist [3], welcher unsere gewählten Technologien unterstützt und einen kostenfreien Start-Kostenplan hat, haben wir uns für diesen Anbieter entschieden. Grundsätzlich könnte das CMS aber auf einen beliebigen Hosting-Anbieter gezügelt werden, da keine anbieterspezifischen Funktionalitäten verwendet wurden.

Das CMS wurde mit JavaScript und HTML umgesetzt. Als JavaScript Framework wurde auf AngularJS in Verbindung mit der Programmiersprache TypeScript gesetzt.

#### 4.5.1.2 Backend

Das Backend ist die Schnittstelle zwischen der App und dem CMS. Sämtliche zu persistierende Daten des CMS werden per HTTPS an die REST-Schnittstelle des Backends gesendet welches diese in einer Datenbank speichert. Die Mobile-App wiederum ruft diese Daten ab um diese für die Besucher darzustellen. Auch das Backend könnte bei einem beliebigen Hosting-Anbieter laufen welcher die nötige Technologie unterstützt. Aus denselben Motivationsgründen wie beim CMS wurde aber für Heroku als Hosting-Anbieter entschieden. Das Backend wurde mit Ruby on Rails umgesetzt.

#### 4.5.1.3 Cloudinary

Cloudinary [2] ist ein Hosting Anbieter welcher auf das Hosten von Medien spezialisiert ist. Der Dienst wird verwendet um sämtliche Bilder, welche im CMS hochgeladen werden können, zentral abzulegen. Im Backend wird lediglich ein Verweis auf die Bilder persistiert.

#### 4.5.1.4 App

Die App wird von den Besuchern in einem Museum genutzt. Die Inhalte der App werden aus dem Backend geladen, die Medien direkt von Cloudinary. Durch im Museum platzierte Beacons kann die App die Position des Besuchers bestimmen und entsprechende Inhalte anzeigen. In einer ersten Version ist eine Internetverbindung erforderlich. Die App könnte man aber so erweitern, dass es nach einer Synchronisation komplett offlinefähig wäre. [4]

Beim App handelt es sich um eine gemeinsame App in dem sich alle Museen laden lassen. Die App kann im App-Store von Apple sowie im Play-Store von Google veröffentlicht werden. Man könnte sich für grössere Museen auch vorstellen, ein dediziertes App in die App-Stores zu stellen.

Es handelt sich um eine Crossplatform-App für Android und iOS welche mit dem Ionic Framework entwickelt wurde. Weitere Details zur App können der Semesterarbeit [1] entnommen werden.

### 4.5.2 Tools & Sprachen

#### 4.5.2.1 AngularJS

AngularJS [5] ist ein von Google entwickeltes JavaScript-Framework. Es dient der Entwicklung von Single-Page-Applications (SPAs). Das besondere an SPAs ist, dass ein grosser Teil der Business-Logik Clientseitig implementiert ist. Im Gegensatz dazu stehen Round-Trip-Applications (RTAs), bei welchen die Logik Server-Seitig implementiert ist, was heisst, dass jeweils HTML vom Server gerendert und an den Client geschickt wird. Ein grosser Vorteil von SPAs ist, dass komplexe User Interfaces im Gegensatz zu RTAs verhältnismässig einfach implementiert werden können.

AngularJS bietet einen grossen Funktions-Umfang. Nennenswert ist hier u.A. das Two-Way-Binding: UI-Elemente wie z.B. Eingabefelder können deklarativ an ein Objekt in der JS-Runtime gekoppelt werden. Ändert man den Wert des Eingabefeldes, ändert sich automatisch der Wert des Objektes, und umgekehrt. Ein weiterer Vorteil von AngularJS ist die Architektur mit Dependency

Injection, was die Strukturierung und Testbarkeit von Angular-Applikationen stark vereinfacht. AngularJS verfügt ausserdem über eine grosse und stetig wachsende Open-Source Community, was in einer guten Dokumentation des Frameworks resultiert.

Natürlich gibt es neben AngularJS ein gutes Duzend weitere professionelle SPA-Frameworks, wie z.B. ReactJS [6], EmberJS [7] oder Polymer [8]. Der Entscheid, AngularJS zu verwenden, ist auch darauf zurückzuführen, dass die Autoren dieser Arbeit beruflich mit AngularJS zu tun haben, und so viel Synergie erzielen konnten.

Ein weiterer Grund, AngularJS einzusetzen, ist der kurz bevorstehende Release des Nachfolgers, Angular2 [10]. Angular2 wird als SPA-Framework der Zukunft gehandelt und aktuelle Berichte und Beispiele von Angular2-Applikationen sind vielversprechend. [11] Angular2 wird in TypeScript entwickelt. Um eine Migration unseres CMS von Angular1 auf Angular2 so einfach wie möglich zu gestalten, wurde TypeScript eingesetzt.

#### 4.5.2.2 TypeScript

TypeScript ist eine von Microsoft entwickelte Programmiersprache. Sie dient dazu, Mängel von JavaScript zu beheben, in dem Konstrukte wie Klassen, Module und Typsicherheit zur Verfügung gestellt werden. Durch diese Features soll auch die Produktivität und Zufriedenheit von Entwicklern verbessert werden, da z.B. bessere Code-Completion ermöglicht wird [9].

#### 4.5.2.3 Jade & SCSS

Wie schon bei der Semesterarbeit wurden auch für HTML und CSS transkompilierte Sprachen verwendet. Ziel dieser Sprachen ist es, die verschiedenen Mängel der Zielsprachen auszubügeln.

##### *Jade*

Um HTML-Ansichten zu generieren wurde Jade eingesetzt. Jade ist indentationsbasiert und arbeitet somit mit Einrückungen. Dies und weitere Funktionen erlauben es, den Overhead von HTML zu vermindern und führen zu kürzerem, übersichtlicherem Code. Nachfolgend ein Beispiel mit der Punktnotation für Style-Klassen und dem Vergeben einer ID:

**Jade**

```
article#main.content
  p Text
```

**HTML**

```
<article class='content' id='main'>
  <p>Text</p>
</article>
```

### *SASS*

Für das Gestalten der Views wurde SASS (Syntactically Awesome Style Sheets) verwendet. Die Syntax von SASS sieht zunächst aus wie normales CSS. Man hat aber zusätzliche Möglichkeiten wie Variablen-Deklarationen oder das verschachteln von Styles.

**SASS**

```
$main: #bc43ec;
div {
 border-color: $main;
 h1{
  background-color : $main;
 }
}
```

**CSS**

```
div {
  border-color: #bc43ec;
}

div h1 {
  background-color: #bc43ec;
}
```

#### 4.5.2.4 Tools

Für die Entwicklung wird keine bestimme IDE vorausgesetzt. Es kann mit einem simplen Text-Editor gearbeitet werden. Der lokale Entwicklungsserver sowie Test- und Buildprozesse wurden aus der Kommandozeile heraus gesteuert. Das Debugging der App wurde im Browser mit den dort vorhandenen Developer-Tools durchgeführt.

#### 4.5.2.5 Kompilierung

Die oben erwähnten Sprachen können von heutigen Browsern nicht interpretiert werden. Dies bedeutet also, dass nach einer Änderung am Source-Code dieser zunächst in die Zielsprache übersetzt werden muss: Man editiert z.B. ein TypeScript File, und muss dieses dann in JavaScript umwandeln, sodass es vom Browser interpretiert werden kann.

Um diesen Prozess zu automatisieren wurde der Task-Runner Gulp verwendet. Per JavaScript kann man verschiedene Tasks konfigurieren und diese ausführen. So wurden Tasks eingeführt welche auf Fileänderungen im Source Ordner reagieren und die Files automatisch in die Zielsprache kompilieren. Auch ist ein Task erstellt worden, der automatisch einen lokalen Webserver aufbaut, welcher den Browser aktualisiert, wenn Files neu kompiliert werden.

Da Gulp auf Node.js basiert können sämtliche Tasks auch direkt auf Heroku ausgeführt werden. Somit ist es möglich den Source-Code via GIT auf Heroku zu pushen und dort automatisch kompilieren zu lassen.

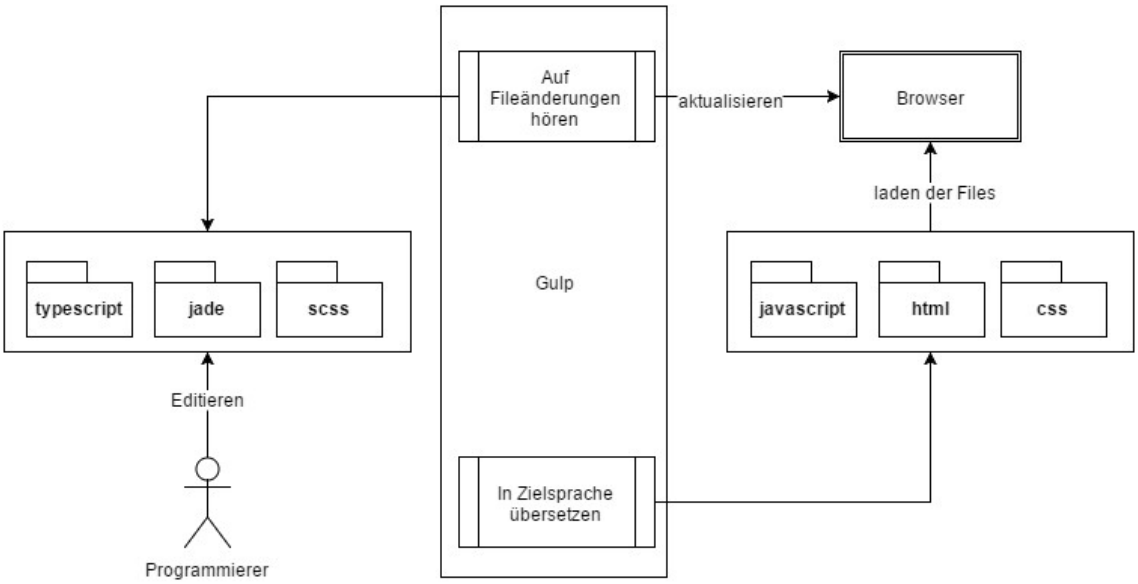



*Abbildung 4-3 Illustration Build-Prozess*

## 4.6 Ergebnisse und Folgerungen

Ein wichtiges Ziel dieser Arbeit war es, den Umgang mit modernen Web- und Cloud-Technologien zu vertiefen. Im Folgenden werden die gesammelten Erfahrungen und Empfehlungen erläutert.

Hinweis: Um folgende Abschnitte besser zu verstehen werden Grundkenntnisse in AngularJS vorausgesetzt.

### 4.6.1 Angular & TypeScript

Der Einsatz von AngularJS mit TypeScript war ein Erfolg. Ein wichtiger Aspekt war die Migrierbarkeit von Angular1 auf Angular2. Es wurden einige Vorkehrungen getroffen, um dies zu erleichtern.

#### 4.6.1.1 Strukturierung in Components

Die Strukturierung einer Applikation mit Components ist ein Trend, der auch in vielen anderen SPA-Frameworks wie Polymer oder ReactJS anzutreffen ist. Weil TypeScript Klassen und Annotations kennt, können einzelne Components sehr elegant strukturiert werden. [12]

Eine typische AngularJS-Applikation ist mit Controllers und Views aufgebaut. Directives werden dazu verwendet, einzelne UI-Elemente wie z.B. einen Button mit speziellen Zusatzfunktionen zu realisieren und wiederverwendbar zu machen. Die Strukturierung in Components bedeutet in Angular, dass das gesamte UI mit Directives implementiert wird, was zu Folge hat, dass keine Controllers mehr benötigt werden.

Genau genommen ist dies eine wichtige Änderung in Angular2, welches das Konzept von Controllers nicht mehr kennt. Eine Angular2-Applikation wird also nur mit verschachtelten Components realisiert, was dem Composite-Pattern entspricht.

Um eine Migration von AngularJS auf Angular2 so leicht wie möglich zu gestalten, sollte also auf Controllers vollständig verzichtet werden und stattdessen sämtliches UI mit Directives implementiert werden. [13]

#### 4.6.1.2 Class statt Factory-Function

Der folgende Code-Ausschnitt zeigt ein in JavaScript geschriebenes Directive wie es üblicherweise anzutreffen ist. Es handelt sich dabei um eine Factory-Function, welche ein JavaScript-Objekt zurückgibt, welches unter anderem eine Property für die Link-Function enthält. Diese Link-Function ist die Schnittstelle zum UI, denn dort wird per Parameter der *$scope* injected mit welchem Objekte mit dem DOM verknüpft werden können.

```
angular.directive('areaViewComponent', ['AreaService',
function(AreaService) {
  return {
    restrict: 'E',
    template: '<div>{{title}}</div>',

    link: function($scope) {
      $scope.title = 'Ant';

      $scope.saveArea = function() {
        ...
      }
    }
  }
}]);
```

*Abbildung 4-4 Directive in Angular1*

Ein wichtiger Vorteil von TypeScript ist die Verfügbarkeit des class-Keywords. Dies kann man sich bei der Implementation eines Directives zu Nutze machen: Statt dass man, wie im vorhergehenden Beispiel gezeigt, eine Link-Funktion implementiert, erstellt man eine Klasse:

```
export class AreaViewComponent{
  private static template = '<div>{{ctrl.title}}</div>';
  private static selector = 'area-view-component';

  private static options = {
    bindToController: {
      title: '='
    }
  };

  constructor(@Inject('AreaService') private AreaService) {
    this.title = 'Ant';
  }
  saveArea() {
    ...
  }
}

app.directive(
    makeSelector(AreaViewComponent),
    makeDirective(AreaViewComponent));
```

*Abbildung 4-5 Directive in Angular1 mit TypeScript*

Das obige Code-Beispiel bedarf einiger Erklärungen:

Interessant ist hier der Einsatz der von AngularJS kürzlich eingeführten *bindToController*-Option: Dadurch wird das *$scope*-Objekt obsolet, denn statt dass Objekte auf dem *$scope* deklariert werden, können diese direkt als Properties der Klasse implementiert werden.

Man sieht ausserdem, wie Services mit einer TypeScript-Annotation injected werden.

Das Directive wird erstellt, in dem die Helper-Funktionen *makeSelector* und *makeDirective* vewendet werden. Diese lesen die statischen Properties *template* sowie *selector* der Klasse.

Dieser Code ist 100% Angular1-tauglich. Betrachtet man bloss die Klassen-Deklaration, sieht er der Component-Deklaration in Angular2 im folgenden Code-Beispiel wesentlich ähnlicher als die klassische Directive-Deklaration in Code-Beispiel Abbildung 4-4 Directive in Angular1.

```
@Component({
  selector: 'area-view-component',
  services: [AreaService]
})
@View({
  inline: '<div>{{title}}</div>';
})
export class AreaViewComponent{
  constructor(AreaService: AreaService) {
    this.title = 'Ant';
  }
  saveArea() {
    ...
  }
}
```

*Abbildung 4-6 Component in Angular2*

#### 4.6.1.3 ES6 Module Syntax.

Je mehr Komponenten eine Applikation umfasst, desto schwieriger wird es, diese sauber zu strukturieren. JavaScript (ES5) selbst bietet keine Konzepte zur Modularisierung von Applikationen, was sehr schnell zu einem Durcheinander führt. TypeScript ermöglicht die Verwendung der Module Syntax des ECMAScript6 Standards, was eine klare Strukturierung ermöglicht. [14]

```
import {IArea} from './IArea';
export class AreaViewComponent {
}
```

*Abbildung 4-7 EcmaScript6 Module Syntax in TypeScript*

#### 4.6.1.4 Kein Two-Way-Binding in Angular2

##### *Two-Way-Binding in Angular1*

Eine wichtige Neuheit in Angular2 ist die Inexistenz des aus Angular1 bekannten Two-Way-Bindings. Two-Way-Binding wurde in Angular1 mit einem rechenintensiven Mechanismus implementiert: Im sogenannten *digest*-Zyklus wurde überprüft, ob sich die Objekte, welche per *ng-model* oder *$scope.$watch* für ein Two-Way-Binding registriert sind, geändert haben. Im Falle von Änderungen wurden entsprechende Anpassungen im DOM automatisch gemacht bzw. die per *$watch* registrierten Callbacks aufgerufen.

Es wird schnell klar, dass dies bei aufwändigen UIs mit vielen Bindings nicht besonders effizient ist. Es ist auch kein Geheimnis, dass AngularJS unter diversen Performance-Problemen leidet. [15] Um dem entgegenzuwirken, wird das Two-Way-Binding in Angular2 durch eine Implementation des Observer-Patterns abgelöst.

##### *Lösung in Angular2*

Das folgende Beispiel demonstriert den Einsatz von *ng-model* in einer Angular1-Applikation:

```
<input ng-model="person.name">
```

*Abbildung 4-8 ng-model in Angular1*

Dieser Code bewirkt, dass sämtliche Eingaben in diesem *input*-Feld auf das *name*-Property des *person*-Objektes automatisch übernommen werden, und umgekehrt. Das Binding ist also Bi-Direktional.

In Angular2 würde die Deklaration dieses Feldes folgendermassen aussehen:

```
<input [ng-model]="person.name" (ng-model-change)="person.name=$event">
```

*Abbildung 4-9 ng-model in Angular2*

Auffällig ist hier der Einsatz von *[ng-model]* und *(ng-model-change)*. *[ng-model]* bedeutet, dass dem *value*-Property des input-Feldes der Wert von *person.name* zugewiesen wird. *(ng-model-change)* heisst in Angular2, dass ein Event-Binding auf den *input*-Event des *input*-Feldes hergestellt wird. Beim Eintreffen des input-Events, also wenn jemand etwas eingibt, wird der entsprechende Code wie oben dargstellt ausgeführt. Mit diesem Code erreicht man also im Prinzip ein manuelles Two-Way-Binding.

Das zweite Code-Beispiel ist deutlich aufwändiger als das erste und spricht nicht für den Einsatz von Angular2. Um dem vorzubeugen kann in Angular2 eine Abkürzung ("Syntactic Sugar") verwendet werden. Aus dem folgenden Code-Beispiel resultiert das gleiche Verhalten wie aus Code-Beispiel 2:

```
<input [(ng-model)]="person.name">
```

*Abbildung 4-10 ng-model abgekürzt in Angular2*

Wie man sieht ist dieser Code beinahe identisch mit Code-Beispiel 1.

#### *Migrierbarkeit: was muss beachtet werden*

Die Schlussfolgerung daraus ist, dass bei der Entwicklung einer auf Angular2 migrierbaren Applikation kein *$scope.$watch* eingesetzt werden sollte, da dies nicht mehr unterstützt wird. Beim Einsatz von *ng-model* hingegen ist weniger Vorsicht geboten, da sich die Syntax nur minimal unterscheidet.

### 4.6.2 Heroku und Cloudinary

Das Angebot an Cloud-Technologien lässt kaum Wünsche offen. Das Hosten des Front- und Backends in der Cloud mit Heroku verlief ohne Probleme. Das Konzept des Deployments per Git ist simpel und effizient. [16]

Durch den Einsatz von Cloudinary als Medienhost wurde das Verwalten von Bildern stark vereinfacht. Die verfügbaren Funktionen wie automatische Konvertierung in verschiedene Datei-Typen, Grössen etc. sind von grossem Vorteil, denn damit kann genau gesteuert werden, in welchem Format die Bilder hoch- und runtergeladen werden. Dies ist z.B. von Vorteil für die Darstellung von Thumbnails.

Sowohl bei Heroku als auch Cloudinary konnten kostenfreie Starter-Pläne eingesetzt werden. [17] [18] In beiden Fällen können die Kapazitäten massiv erweitert werden, was ein problemloses Scale-Out ermöglichen sollte.

### 4.6.3 Problem: Initialaufwand

Der Einsatz von moderner Technologie bringt aber auch Probleme mit sich. So ist vor allem die Handhabung von auf NodeJS bzw. NPM basierenden Projekten nicht ganz einfach. Die gesamte Toolchain musste von Hand aufgesetzt werden. Dies beinhaltet z.B. den Build-Prozess, also die Transpilation der eingesetzten Sprachen TypeScript, Jade und SCSS, welche als Ersatz für JavaScript, HTML und CSS fungieren. Bei der Lancierung eines neuen Projektes, welches auf solchen neuartigen Technologien basiert, muss das Aufsetzen der Arbeitsumgebung sorgfältig in die Zeitplanung mit einbezogen werden. Dies ist ein Gegensatz zu Produkten wie z.B. dem .NET-Framework, welches als Gesamtsystem daherkommt und typischerweise sehr leicht mit entsprechenden IDEs wie z.B. Visual Studio aufgesetzt werden kann.

### 4.6.4 Entwicklung in kurzer Zeit

#### 4.6.4.1 Eigenentwicklung vs. Standardsoftware

Die Entwicklungszeit im Rahmen der Bachelorarbeit ist kurz. Aus diesem Grund musste sich zu Beginn der Arbeit die Frage gestellt werden, ob man ein existierendes CMS wie WordPress oder Typo3 anpassen will oder auf eine Eigenentwicklung setzt.

Wichtigstes Entscheidungskriterium war dabei der Grad an Flexibilität der durch die individuellen Funktionalitäten des CMS gefordert wurde. Das Bilder-Management, die dynamische Quiz Konfiguration oder auch die Übersetzung hätten sich nicht direkt mit einer Out-of-the-box Lösung realisieren lassen. Da von Museumsmitarbeitern keine technische Versiertheit

vorausgesetzt werden kann, sollte das System so benutzerfreundlich wie möglich gestaltet werden.

Die Aufwandanalyse zeigte, dass mit dem Einarbeiten und anschliessenden Erweitern einer Standardsoftware keine signifikanten Zeitersparnisse hätten gewonnen werden können. Auch wurden klassische Funktionalitäten eines CMS, wie das Zusammenstellen einer ganzen Ansichts-Seite aus mehreren Komponenten, nicht benötigt. Aus diesen Gründen haben wir uns für das Entwickeln einer Individualsoftware entschieden und waren somit auch frei in der Technologiewahl.

#### 4.6.4.2 Einsatz von Open-Source-Software

Der Fokus wurde auf Hauptfunktionalitäten gerichtet. Jedoch durften wichtige Aspekte, die zur Benutzerfreundlichkeit und Verständlichkeit des CMS beitragen, nicht vernachlässigt werden. Dies beinhaltet Punkte wie die einheitliche Navigation, das Validieren der Formulare und Feedbacks auf Aktionen des Benutzers.

Wichtigen Anforderungen des CMS konnten in gesetzter Frist umsetzen werden. Die Wahl von Webtechnologien hat sich auch in diesem Aspekt bewährt. Aufgrund der aktiven Open-Source Community konnte man auf bereits realisierte Komponenten zurückgreifen und sich so wichtige Zeit sparen. So konnte zum Beispiel ein bestehendes Breadcrumb-Modul integrieren werden. Sämtliche verwendete Libraries und Module können dem Architekturdokument entnommen werden.

#### 4.6.4.3 Testing

Da das CMS von Grund auf neu entwickelt wurde, war es schwierig, die unterschiedlichen Problemstellungen von Anfang an einzuschätzen. Dies hat auch das Schreiben von Unit-Tests erschwert, da diese lösungsorientiert sind und sich eine Applikation in der Anfangsphase sehr leicht ändern kann.

Die Priorität dieser Arbeit war es, ein fertiges, benutzerfreundliches CMS zu entwickeln, damit es von Mitarbeitern des Museums getestet werden kann. Aus diesen Gründen sind abschliessende Unit-Tests nicht Bestandteil dieser Arbeit.

Sollte das Projekt weitergeführt werden, wäre das Schreiben der Unit-Tests eine der Prioritäten, um eine erfolgreiche Weiterentwicklung zu ermöglichen.

### 4.6.5 App Statistiken mit Google Analytics

Das Sammeln von Statistiken wurde mittels Google Analytics implementiert. Der Vorteil von Google Analytics ist, dass viele Informationen mit wenigen Zeilen Code gewonnen werden können. Google Analytics ist jedoch sehr umfangreich und eine Einarbeitung aufwendig. Da das Interface von Google Analytics ebenfalls sehr komplex ist wurde entschieden, die Daten von Google Analytics in eigenen Graphen und Tabellen direkt im CMS anzuzeigen. Dazu braucht es die Verlinkung eines Google Logins. Dieser Schritt im momentan noch nicht automatisiert. Das heisst ein neues Museum muss durch die Projektmitglieder manuell bei Google Analytics

hinzugefügt werden. Zudem müssen Google Accounts des Museums damit verknüpft werden, um Berechtigungen für das Betrachten der Statistiken zu erhalten. Was die Entwicklung zudem erschwert hat, ist, dass die Daten bei Google Analytics zeitverzögert erst ca. 2 Tage später publiziert werden.

In einem ersten Schritt wurden nur die Zugriffstatistiken im CMS dargestellt. Das Konzept für weitere Graphen und Tabellen steht aber.

### 4.6.6 Umfang

Die Lines of Code beziehen sich auf Code ohne Berücksichtigung von Kommentaren.

#### 4.6.6.1 CMS

| Sprache | Lines of Code | Files |
|---|---|---|
| **TypeScript** | 2830 | 89 |
| **Jade/HTML** | 660 | 50 |
| **SCSS** | 9 | 450 |
| **Javascript (Gulp and Node)** | 200 | 3 |

*Tabelle 1 CMS Lines of Code*

#### 4.6.6.2 Backend

| Sprache | Lines of Code | Files |
|---|---|---|
| **JSON** | 1420 | 16 |
| **Ruby** | 1226 | 82 |

*Tabelle 2 Backend Lines of Code*

# 5 Glossar

| | |
|---|---|
| **Angular/AngularJS** | Ein JavaScript-Framework für die Entwicklung von Rich Client Web-Applikationen |
| **Bug** | Ein Fehler in einem Computer-Programm |
| **Cloud** | Verteilte Rechenzentren die für Hosting und Ausführen von Programmen genutzt werden. Man zahlt meistens nur die Ressourcen welche man braucht und ist sehr skalierbar. |
| **Crossplatform** | Ausführbar auf unterschiedlichen Zielsystemen (z.B Android und iOS) |
| **Device** | Mobiles Gerät, Handy, Mobiltelefon |
| **Dependency Injection,** | Ein Programmier-Pattern welches die Abhängigkeit eines Objektes zur Laufzeit reglementiert. |
| **DOM** | Schnittstelle zwischen HTML und JavaScript. Erlaubt das Verändern, Hinzugefügen und Löschen von HTML Elementen im JavaScript. |
| **HSR** | Hochschule für Technik Rapperswil |
| **HTTPS** | HyperText Transfer Protocol Secure. Ein Protokoll im Web für die abhörsichere Übertragung. |
| **iBeacon/Beacon** | Bluetooth Low Energy Sender. |
| **Ionic** | Ein JavaScript-Framework für die Entwicklung von Crossplatform-Applikationen. Basiert auf Angular |
| **Skalierung** | Die dargestellten Inhalte werden auf die Grösse des Bildschirms angepasst |
| **SaaS** | Software as a Service, ein Cloud-Computing-Konzept, bei welchem eine Software als Dienstleistung betrieben wird |
| **REST** | Ein Programmierparadigma für Webservices, welches das Verhalten und die Struktur des World-Wide-Web abstrahiert |
| **RTA** | Round-Trip-Application, eine Web-Site, deren Markup (HTML) auf dem Server generiert wird. Um die Darstellung auf einer Ansicht zu ändern, wird ein Reload der gesamten Seite benötigt. |
| **Scale-out** | Ein Konzept aus dem Cloud-Computing, bei welchem Rechenleistung durch Hinzufügen von neuen Rechnern erweitert wird. Im Gegensatz dazu steht das Scale-up, bei welchem Rechenleistung durch Upgraden eines einzelnen Rechners erweitert wird |

| | |
|---|---|
| **IDE** | Integrated Development Environment, eine Kollektion von Tools, welche das Arbeiten mit spezifischen Technologien vereinfacht und unterstützt |

## 5.1 Literaturverzeichnis

[1] D. Keller und S. Müller , „eprints.hsr.ch,“ Mai 2015. [Online]. Available: http://eprints.hsr.ch/437/. [Zugriff am Dezember 2015].

[2] „Cloudinary - Image Management In The Cloud,“ [Online]. Available: http://cloudinary.com/.

[3] „Selbie Labs,“ [Online]. Available: http://selbielabs.com/cloud-platforms-compared/.

[4] D. a. B. C. a. F. S. Rogai, „Content-Based Multi-platform App Forge,“ *Mobile Software Engineering and Systems ,* pp. 166-167, Mai 2015.

[5] „AngularJS,“ November 2015. [Online]. Available: https://angularjs.org/.

[6] „ReactJS,“ November 2015. [Online]. Available: http://facebook.github.io/react/.

[7] „EmberJS,“ November 2015. [Online]. Available: http://emberjs.com.

[8] „Polymer,“ November 2015. [Online]. Available: https://www.polymer-project.org/1.0/.

[9] „Angular,“ November 2015. [Online]. Available: https://angular.io/.

[10] „telerik.com,“ November 2015. [Online]. Available: http://developer.telerik.com/featured/will-angular-2-be-a-success-you-bet/.

[11] L. a. H. S. Fischer, „An Empirical Investigation of the Effects of Type Systems and Code Completion on API Usability Using TypeScript and JavaScript in MS Visual Studio,“ in *Proceedings of the 11th Symposium on Dynamic Languages*, ACM, 2015, pp. 154-167.

[12] „Rangle io,“ Dezember 2015. [Online]. Available: http://blog.rangle.io/angular2-components/.

[13] „codelord.net,“ [Online]. Available: http://www.codelord.net/2015/10/07/angular-2-preparation-killing-controllers/. [Zugriff am Dezember 2015].

[14] „2 Ality,“ November 2015. [Online]. Available: http://www.2ality.com/2014/09/es6-modules-final.html.

[15] „airpair.com,“ [Online]. Available: https://www.airpair.com/angularjs/posts/angularjs-performance-large-applications. [Zugriff am Dezember 2015].

[16] „Heroku,“ September 2015. [Online]. Available: https://devcenter.heroku.com/articles/git.

[17] „Heroku Pricing,“ November 2015. [Online]. Available: https://www.heroku.com/pricing.

[18] „Cloudinary Pricing,“ November 2015. [Online]. Available: http://cloudinary.com/pricing.

[19] „Mountain Goat Software,“ [Online]. Available: https://www.mountaingoatsoftware.com/tools/planning-poker. [Zugriff am Dezember 2015].

[20] „Pivotaltracker,“ [Online]. Available: http://www.pivotaltracker.com/community/tracker-blog/public-beta-starting-for-redesigned-new-pivotal-tracker. [Zugriff am Mai 2015].

[21] „Pixabay,“ [Online]. Available: https://pixabay.com/de/frau-portr%C3%A4t-l%C3%A4cheln-462663/. [Zugriff am Oktober 2015].

[22] „Quicknet,“ [Online]. Available: http://members.quicknet.nl/bflassing/images/fotoben.jpg. [Zugriff am Oktober 2015].

[23] „tutorialspoint,“ [Online]. Available: http://www.tutorialspoint.com/ruby-on-rails/rails-directory-structure.htm. [Zugriff am November 2015].

[24] „Ninja Squad,“ Dezember 2015. [Online]. Available: http://blog.ninja-squad.com/2015/07/21/what-is-new-angularjs-1.4/.

[25] „Berner Zeitung,“ [Online]. Available: http://www.bernerzeitung.ch/region/bern/App-bringt-Uebersicht-in-die-Berner-Museumslandschaft/story/26147831. [Zugriff am Dezember 2015].

# 6 Dokumente des Projekts

## 6.1 Projektorganisation

Das Entwicklerteam besteht aus den beiden Mitgliedern Daniel Keller und Samuel Müller. Beide übernehmen dieselben Rollen. Der betreuende Dozent ist Prof. Dr. Markus Stolze.

### 6.1.1 Arbeitstage und Meetings

Es wurden drei volle Tage (8 Stunden) in der Woche für die Arbeit eingeplant. Die Team-Mitglieder waren dabei meistens im gleichen Büro was die Zusammenarbeit und Absprache erleichtert hat.

Jeweils am Mittwoch-Morgen hat das Meeting mit Herrn Prof. Stolze stattgefunden. Ziel war es den aktuellen Stand, das weitere Vorgehen sowie diverse Fragen zu klären.

### 6.1.2 Meilensteine

| Phase | Meilenstein | Bis | Eingehalten |
|---|---|---|---|
| **Evaluation** | Kickoff Meeting | 23.9.2015 | ✓ |
| | Umfang definiert | 24.9.2015 | ✓ |
| **Umsetzung** | Vertikaler Durchstich | 8.10.2015 | ✓ |
| | CMS Fertig | 5.11.2015 | ✓ |
| | Zwischenstand Benutzertests mit NMSG | Mitte Nov. 2015 | ✓ |
| | App Fertig | 26.11.2015 | ✓ |
| **Abschluss** | Abschliessender Benutzertests mit NMSG | Anfangs Dez. 2015 | ✓ |
| | Abgabe Arbeit | 18.12.2015 | ✓ |
| | Abschlussprüfung | 13.1.2016 | - |
| | Abschluss Meeting mit NMSG | Mitte Jan. 2016 | - |

*Tabelle 3 Projektplan*

### 6.1.3 Task-Verwaltung

Zu Beginn der Entwicklungsphase wurde Aufgrund der Anforderungen und gezeichneten Wireframes Arbeitspakete erstellt. Die Pakete wurden dann zu zweit mit dem Planning Poker [19] Ansatz geschätzt. Das heisst beide Personen haben im verdeckten pro Task eine Karte mit Wert 1h, 2h, 4h oder 8h auf den Tisch gelegt. Wichen die beiden Werte voneinander ab wurde diskutiert und sich geeinigt. Tasks über 8 Stunden wurden in Sub-Tasks aufgeteilt. Sämtliche Tasks wurden dann der Priotität nach geordnet und im Online-Tool Pivotal Tracker nachgeführt. Die Tasks wurden nicht fix einem Team-Mitglied zugeteilt. Stattdessen wurde auf agile Entwicklung gesetzt. Die Tasks wurden individuell der Wichtigkeit nach abgearbeitet.

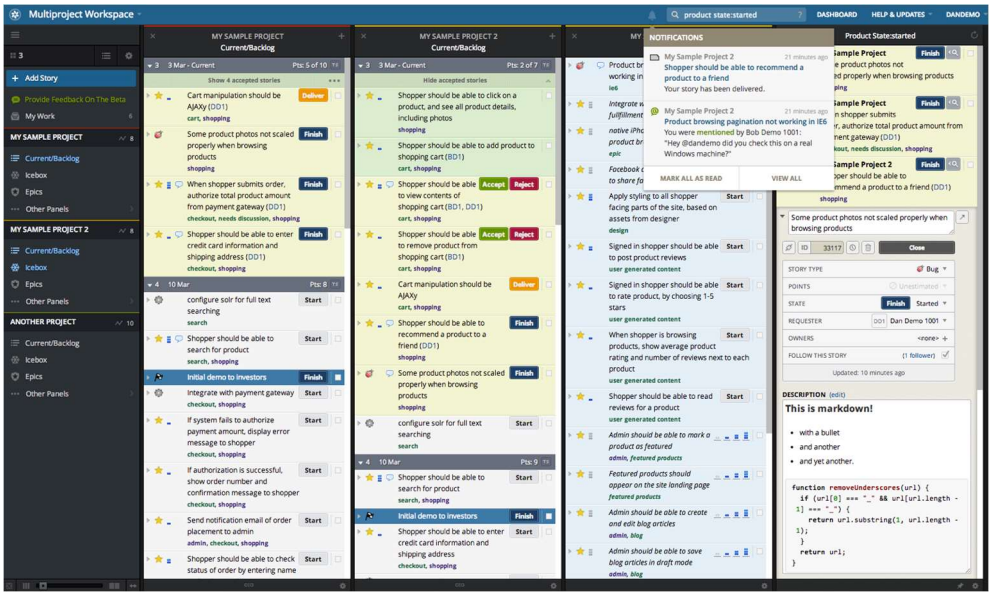



*Abbildung 6-1 Screenshot Pivotal Tracker [20]*

### 6.1.4 Source Code Verwaltung

Der Source Code wurde in privaten Github Repositories verwaltet. Es wurden Pull-Requests verwendet, um Code-Reviews durchzuführen.

### 6.1.5 Dokumentenverwaltung

Sämtliche Dokumente wurden in einem gemeinsamen Ordner auf Google Drive verwaltet. Somit konnte der Zugriff und das gemeinsame Arbeiten an den Dokumenten sichergestellt werden.

Für die Abgabe wurden die Dokumente in einem OneDrive Word Dokument zusammengeführt da die Formatierungsmöglichkeiten in Google Drive begrenzt sind.

## 6.2 Zeitkontrolle

Wir hatten keine Teilung der Aufgabenbereiche. Jedes Teammitglied sollte überall mitwirken und es wurde darauf geachtet, dass alle ungefähr gleich viel Zeit in das Projekt investieren. Die Arbeitszeiten wurden in einem Excel erfasst und später ausgewertet.

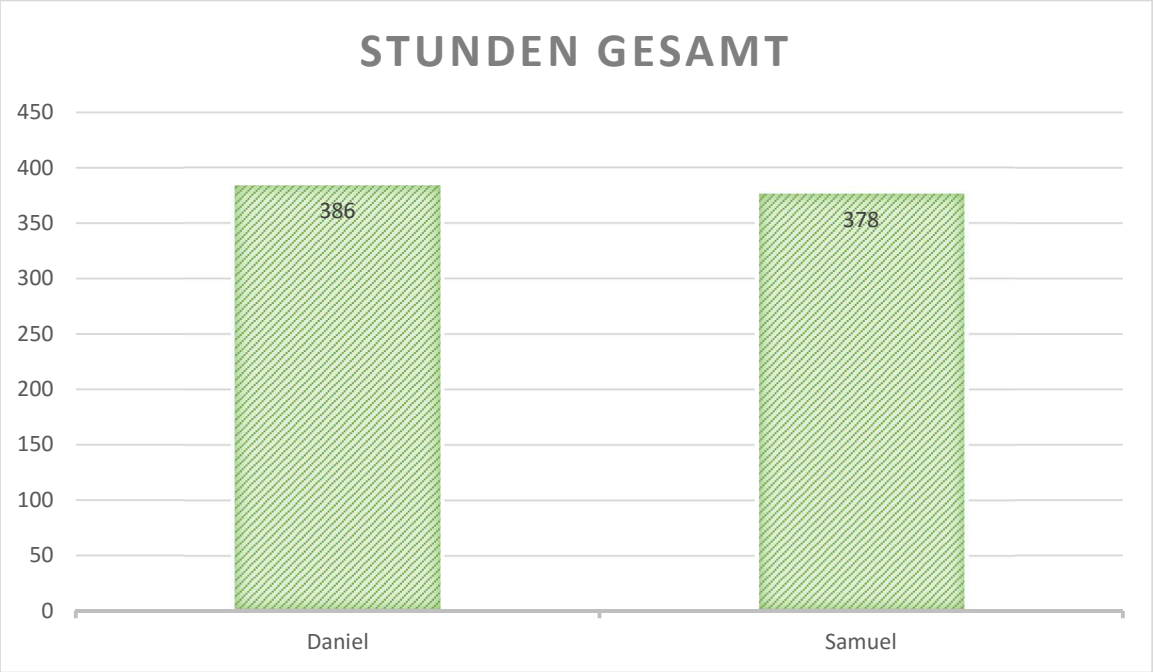



*Abbildung 6-2 Zeitkontrolle Stunden gesamt*

Die Arbeitsverteilung war über alle Monate hinweg etwa gleich (ca. 3*8h pro Person). Im September hatten wir ca. 2 Wochen, im Dezember knapp 3 Wochen zur Verfügung, weshalb dort weniger Arbeitsstunden anfielen.

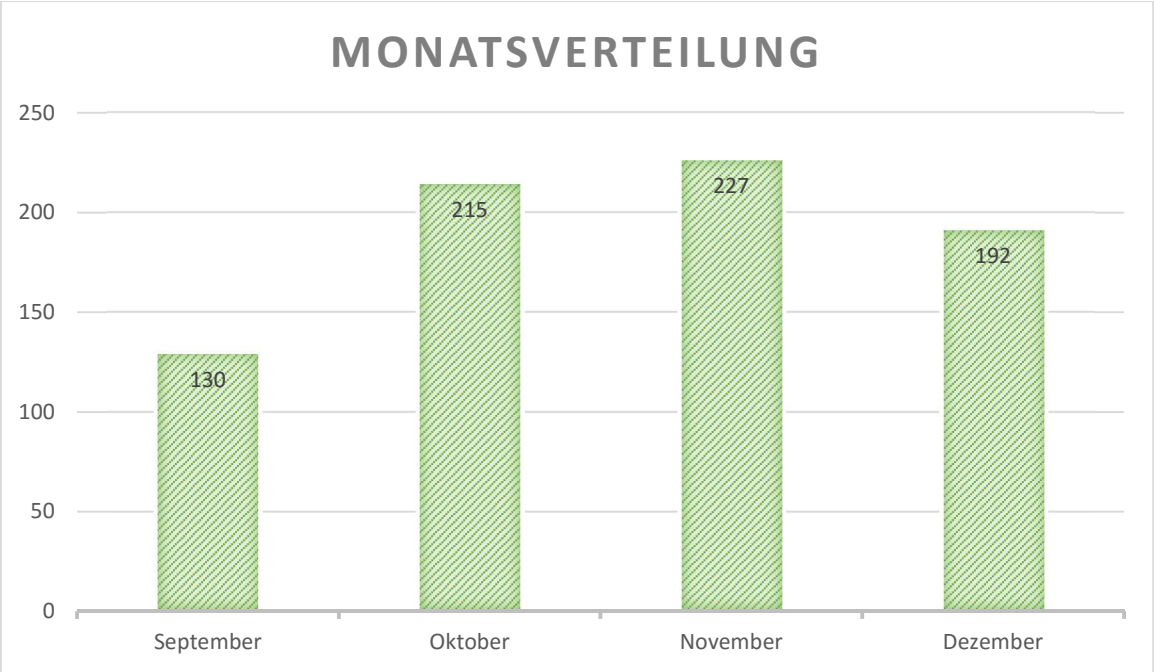



*Abbildung 3 Zeitkontrolle Monatsverteilung*

Mehr als die Hälfte der Arbeitszeit wurde in die Entwicklung investiert. Der Rest war Dokumentation, Meetings und andere administrative Tasks.

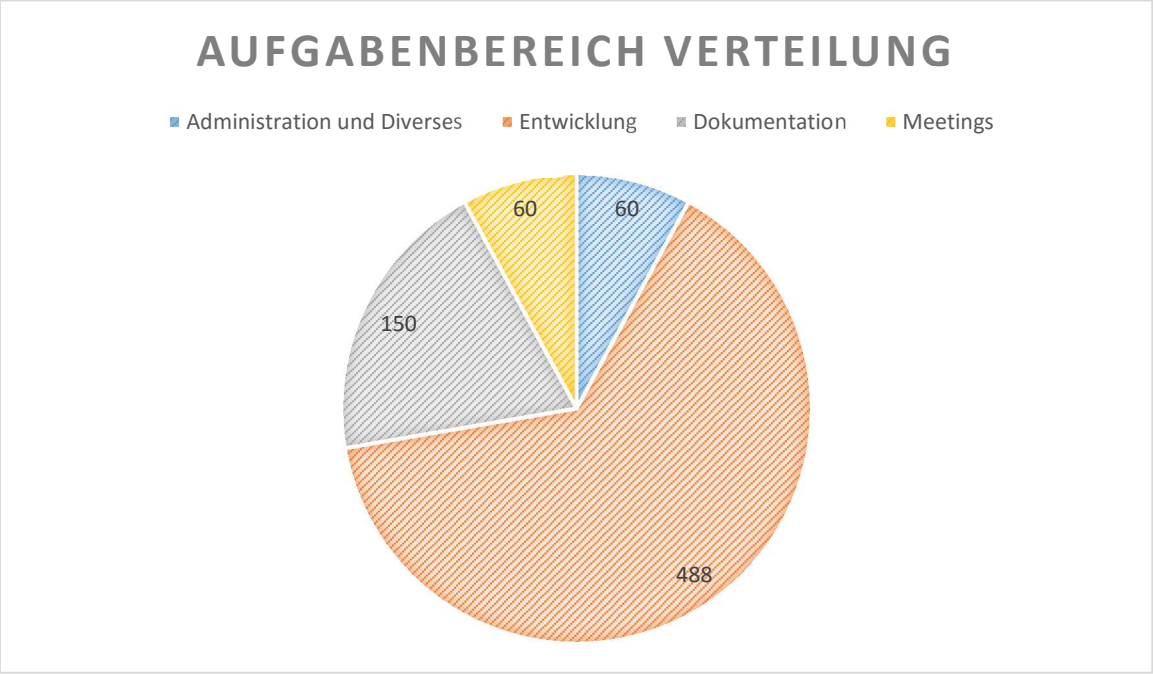



*Abbildung 6-4 Zeitkontrolle Aufgabenbereich Verteilung*

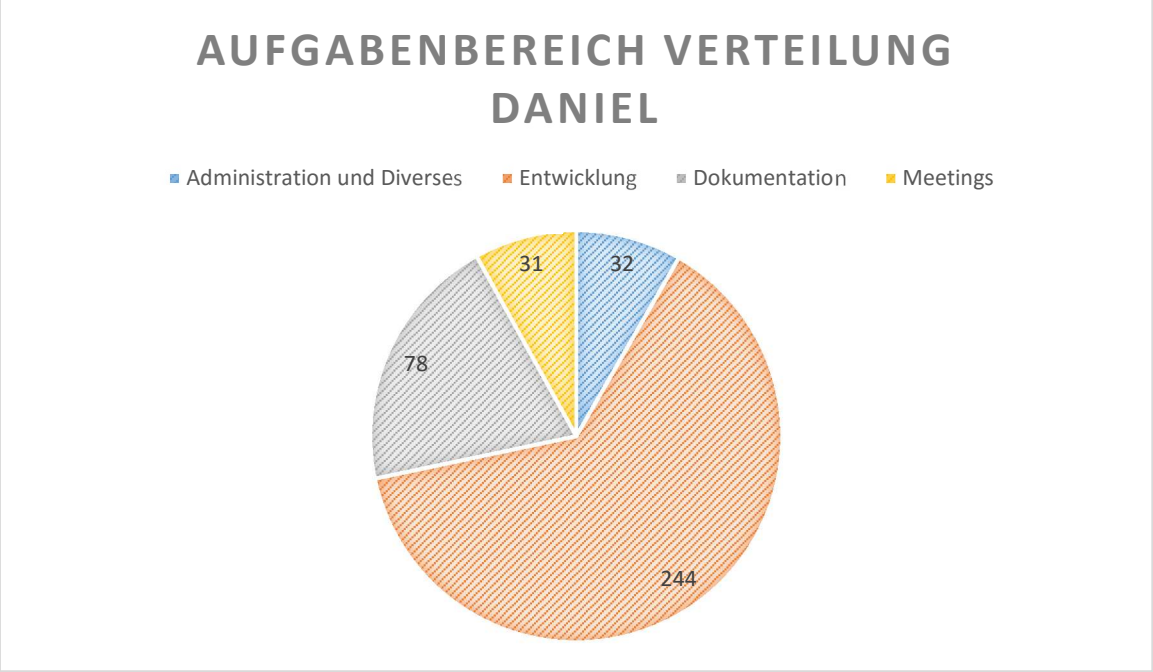



*Abbildung 6-5 Zeitkontrolle Aufgabenbereich Verteilung Daniel*

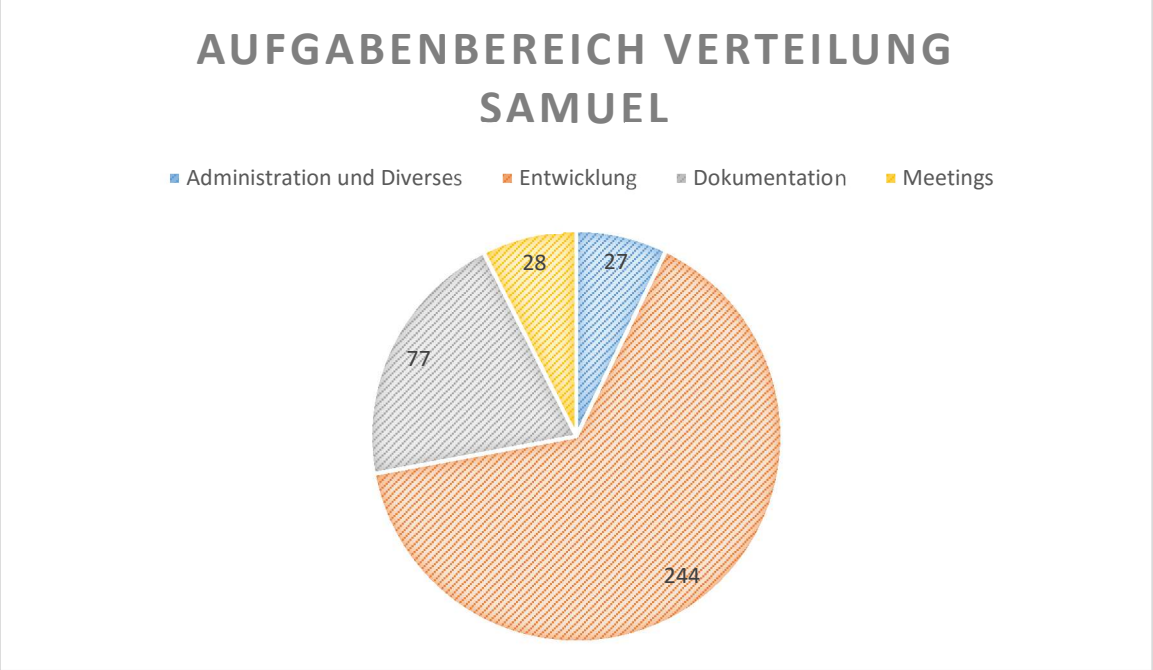



*Abbildung 6-6 Zeitkontrolle Aufgabenbereich Verteilung Samuel*

## 6.3 Anforderungsdokument

### 6.3.1 Einführung

#### 6.3.1.1 Zweck

Dieses Dokument beschreibt die Anforderungsspezifikation und damit die fachliche (nicht technische) Lösung des *maaas* Projektes.

#### 6.3.1.2 Allgemeine Beschreibung

Dieses Dokument summiert die Anforderungen und korrespondierenden Artefakte für die Bachelorarbeit *maaa*s.

### 6.3.2 Benutzer Charakteristik

Das CMS wird von den Mitarbeitern eines Museums bedient. Meistens wird es sich dabei um Museumspädagogen handeln. Es muss davon ausgegangen werden, dass die Benutzer technisch nicht sehr versiert sind und keine CMS Kenntnisse besitzen.

### 6.3.3 Personas

#### 6.3.3.1 Persona “Regina Frey”

| | |
|---|---|
| Name | Regina Frey |
| Alter | 38 |
| Funktion | Museumspädagogin |
| Aufgaben | Zuständig für Museums-Inhalte, Rundgänge durchführen |
| CMS Kenntnisse | Keine |

*Abbildung 6-7 Persona Regina Frey [21]*

Frau Frey wurde vom Museum beauftragt die Museums-App einzuführen. Ihr Ziel ist es, zu allen Bereichen im Museum spannende Zusatz-Inhalte und Quiz zu bieten. Sie hat sich entschieden zwei Zielgruppen damit anzusprechen. Zum einen eine Familie mit Kindern und zum anderen Schulklassen der Oberstufe. Für die Familie sollten die Inhalte hauptsächlich visuell und einfach sein und für die Schulklassen informativ und fordernd.

Sie ist motiviert das CMS zu beherrschen und ist bereit, sich dafür an einem halben Tag Schulen zu lassen. Da Sie sich aber nicht alles notieren kann ist sie auf unterstützende Hilfetexte und Vorschauen im CMS angewiesen. Wichtig ist eine verständliche Navigation und dass Sie immer nachvollziehen kann, was nun im App angezeigt wird.

Zwei Jahre nach der Einführung von *maaas* wird Frau Frey vom Museumdirektor gebeten, das neue Austellungsobjekt in das System zu integrieren. Die Schulung sitzt nicht mehr wirklich und Ihre Notizen sind verloren gegangen. Da sie Angst hat, bestehende Inhalte zu überschreiben, wünscht sie sich eine Auffrischung der Schulung.

#### 6.3.3.2 **Persona "Dieter Direktor"**

| | |
|---|---|
| Name | Dieter Direktor |
| Alter | 61 |
| Funktion | Museumsdirektor |
| Aufgaben | Organisation und Verwaltung des Museums |
| CMS Kenntnisse | Keine |

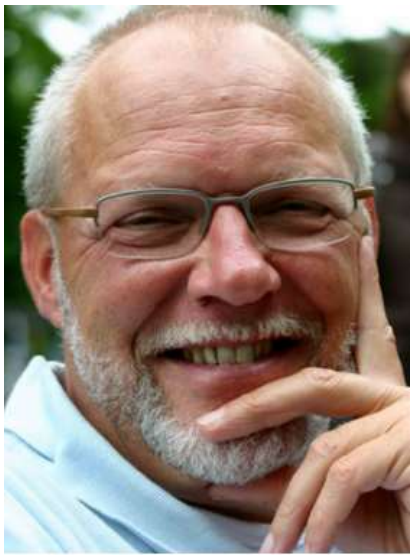


*Abbildung 6-8 Persona Dieter Direktor [22]*

Dieter Direktor hat jeden Tag alle Hände voll zu tun mit verschiedensten Dingen rund um das Museum. Er hat Sitzungen, plant neue Ausstellungen, bezahlt Rechnungen etc. Er ist ein echter Multitasker. Auch ist es ihm wichtig, dass das Museum qualitativ vorne mit dabei ist und setzt viel Wert auf das Detail. So überprüft er auch stets die Arbeit seiner Museums-Pädagogen an den im CMS erfassten Inhalten. Technisch ist er nicht besonders versiert und von Computern versucht er sich fern zu halten. Trotzdem möchte er, wenn es mal nicht anders geht, kleinere Anpassungen an einem Inhalt vornehmen können. Er möchte dabei möglichst direkt zum Ziel gelangen und sich nicht lange mit komplizierten Eingabemasken auseinandersetzen müssen.

### 6.3.4 Use Cases CMS

#### 6.3.4.1 Use Case Diagramm

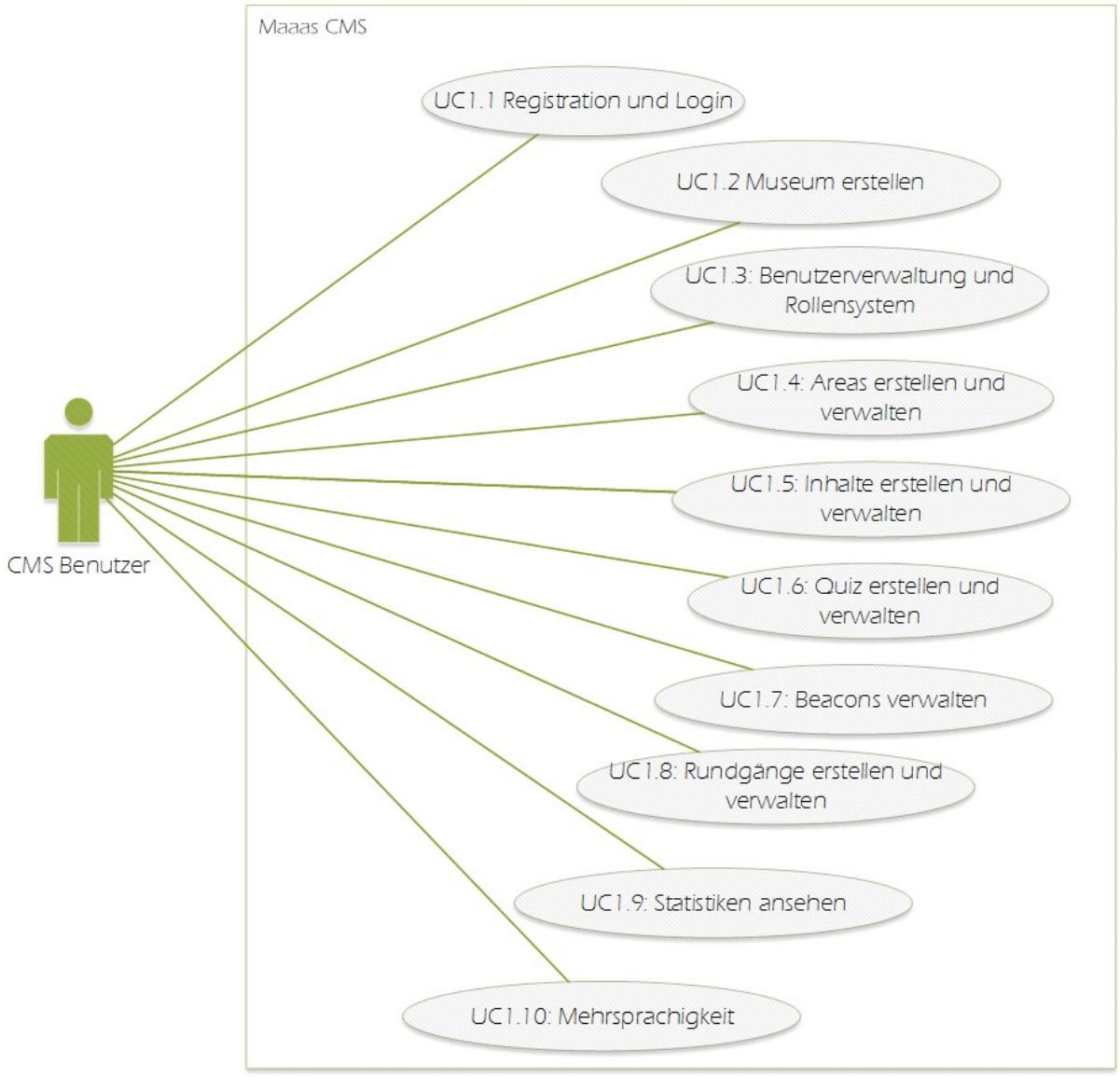



*Abbildung 6-9 Usecase Diagramm*

#### 6.3.4.2 Beschreibungen (Brief)

##### *UC1.1: Registration und Login*

Benutzer können sich unter Angabe von Name, Email und Passwort registrieren. Danach können sich unter Verwendung von Email und Passwort einloggen.

##### *UC1.2: Museum erstellen*

Benutzer können unter Angabe eines Namens ein neues Museum erstellen. Alle Eigenschaften sind nach dem Erstellen editierbar und erscheinen in der App in der Museumsübersicht.

##### *UC1.3: Benutzerverwaltung und Rollensystem*

Benutzer können andere Benutzer zum Museum hinzufügen. Je nach Rolle können diese die Inhalte des Museums verwalten. Die Rollen werden vom System vorgegeben, z.B. Bereich Manager, etc.

##### *UC1.4: Bereiche erstellen und verwalten*

Benutzer können unter Angabe eines Namens einen Bereich erstellen. Die Bereiche können mit einem Beacon verknüpft werden. Visuelle Eigenschaften wie Hintergrundbild, Stickerbild und Farbschema können angepasst werden. Dem Benutzer wird eine Vorschauansicht präsentiert, mit der er die Konfiguration überprüfen kann.

##### *UC1.5: Inhalte erstellen und verwalten*

Benutzer können Inhalte erstellen, in dem sie einen Titel angeben. Der eigentliche Content eines Inhalts kann mit einem Editor angepasst werden. Es kann ausserdem Bild-, Ton- (Audioguide) und Videomaterial hochgeladen werden. Die Bearbeitungsschritte können historisiert werden. Der Benutzer kann so alte Versionen des Inhaltes wieder einsehen.

Dem Benutzer wird eine Vorschauansicht eines einzelnen Inhaltes präsentiert.

##### *UC1.6: Quiz erstellen und verwalten*

Benutzer können zum Erstellen eines Quiz einen Quiz-Typ aus einem Katalog auswählen. Die Konfiguration gestaltet sich je nach Quiz-Typ. Anpassbar sind die Fragen, Antworten sowie der Text, welcher angezeigt wird, wenn das Quiz richtig oder falsch gelöst wurde. Der Benutzer kann sich eine Vorschau ansehen.

Die Quiz-Typen wurden aus der Semesterarbeit übernommen und werden im Kapitel 6.3.4.3 genauer beschrieben.

##### *UC1.7: Beacons verwalten*

Benutzer können die Beacons erfassen und einen Alias für jeden Beacon bestimmen. Zudem kann eine Verlinkung zu dem repräsentierenden Bereich erfasst werden.

##### *UC1.8: Rundgänge erstellen und verwalten*

Benutzer können einen Rundgang erstellen. Ein Rundgang verknüpft die erstellten Bereiche, Inhalte und Quiz.

### *UC1.9: Statistiken ansehen*

Benutzer können Gebrauchsstatistiken der App anschauen.

### *UC1.10: Übersetzen*

Benutzer können die Inhalte in verschiedenen Sprachen verwalten. Quiztypen

#### 6.3.4.3 Quiz-Typen

### *Wahr/Falsch*

Der Benutzer hat eine Reihe von Fragen, welche sich mit Wahr/Falsch bzw. zwei Werten beantworten lassen. Die Frage kann durch ein Bild unterstützt werden. Wurden alle Fragen richtig beantwortet, ist das Quiz abgeschlossen.

Beispiel: Handelt es sich bei dieser Abbildung um einen Schwanz oder einen Froschlurch.

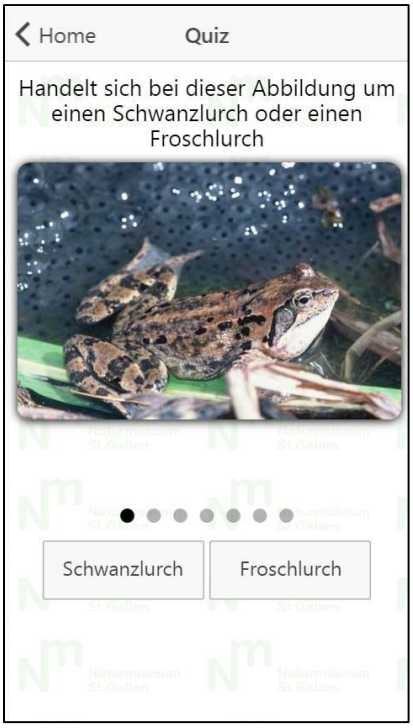



*Abbildung 6-10 Screenshot Wahr/Falsch Quiz*

### *Multiplechoice*

Der Benutzer hat zu einer Frage mehrere Antwortmöglichkeiten. Die Antworten können in Form von Text oder Bild dargestellt werden. Sobald der Benutzer die korrekte Antwort ausgewählt hat, ist das Quiz abgeschlossen

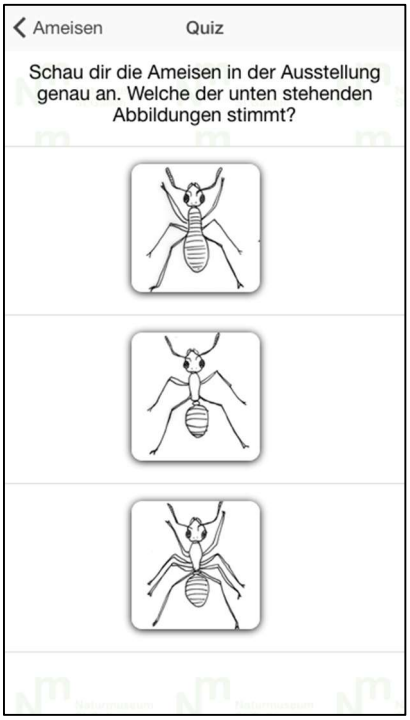



*Abbildung 6-11 Screenshot Multiplechoice Quiz*

### *Bilder in korrekte Reihenfolge bringen*

Der Benutzer muss eine Reihe von Bildern, welche einen Prozess darstellen, in die richtige Reihenfolge bringen. Ist die Reihenfolge korrekt, ist das Quiz abgeschlossen.

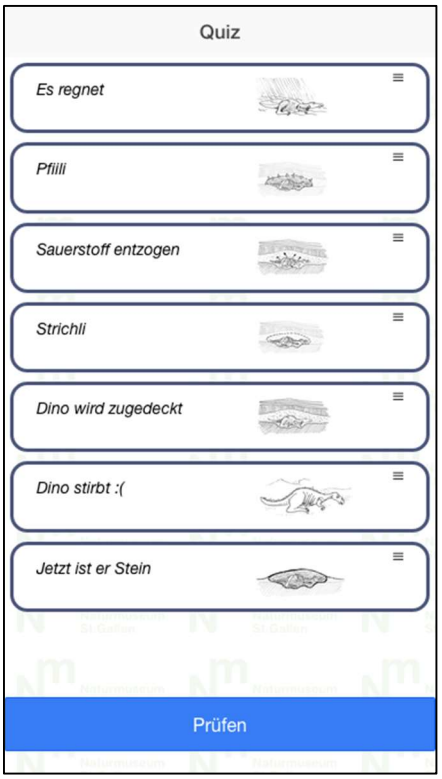



*Abbildung 6-12 Screenshot Reihenfolge Quiz*

### *Bildregion bestimmen*

Dem Benutzer wird ein Bild dargestellt in welchem er eine Region anklicken muss. Hat er die korrekte Stelle angeklickt ist das Quiz abgeschlossen.


*Abbildung 6-13 Screenshot Bildregion Quiz*

### *Bild-Wort Matching*

Der Benutzer hat auf der Linken und Rechten Seite Wörter oder Bilder zur Auswahl, welche er assoziieren muss. Wurden alle Verbindungen zwischen den Seiten korrekt gesetzt, ist das Quiz abgeschlossen.

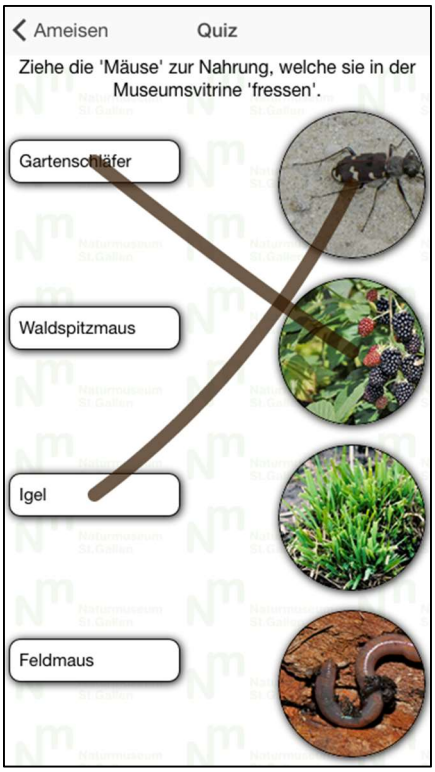



*Abbildung 6-14 Screenshot Matching Quiz*

### 6.3.5 Weitere Anforderungen

#### 6.3.5.1 Qualitätsmerkmale (ISO/IEC 9126)

***Funktionalität***

| | |
|---|---|
| Richtigkeit | Alle Benutzeroberflächen sollen auf unterschiedlichen Bildschirmgrössen funktionieren. |

***Zuverlässigkeit***

| | |
|---|---|
| Reife | Das CMS wird automatisiert und manuell getestet, um mögliche Fehler aufzudecken. |
| Fehlertoleranz | Der Benutzer soll auf Eingabefehler hingewiesen werden. Formulare können nur bei Vollständigkeit gespeichert werden. Fehleingaben führen in keinem Fall zu einem Systemabsturz. |

***Benutzbarkeit***

| | |
|---|---|
| Erlernbarkeit | Es wird davon ausgegangen, dass die Benutzer in einer Schulung in das CMS eingeführt werden. Eine Hilfeseite soll dem Benutzer zusätzliche Informationen geben. |
| Verständlichkeit | Die Software soll leicht verständlich sein. Eingabeformulare und einzelne Eingabefelder werden mit Hilfetexten versehen. Es werden wenige, einfache Navigationsmöglichkeiten zur Verfügung gestellt. Die Navigation wird durch Breadcrumbs unterstützt. |
| Attraktivität | Das CMS soll nach existierenden Standards entworfen werden. Es werden bewährte Open-Source-Libraries eingesetzt. |

***Effizienz***

| | |
|---|---|
| Zeitverhalten | Eine Antwort vom Server sollte unter einer Sekunde sein und im Maximum drei Sekunden nicht überschreiten.<br><br>Der Ladevorgang wird dem Benutzer mit einem Indikator signalisiert. |

#### *Wartbarkeit*

| | |
|---|---|
| Konformität | Es werden bewährte, weit verbreitete Tools (Sprachen, Libraries, Frameworks) eingesetzt. Die Codierung geschieht anhand der Coding Guidelines jener Tools. |
| Analysierbarkeit | Die Prinzipien der hohen Kohäsion und geringen Koppelung sollen eingehalten werden.<br><br>Statische Code-Analyse (Linting) soll dazu dienen, bestimmte Strukturvorschriften einzuhalten (z.B. Spacing zwischen Operatoren, etc.). Dies wird ermöglicht durch existierende Open-Source-Programme. |
| Prüfbarkeit | Der Code soll durch automatisierte Unit-Tests sinnvoll abgedeckt sein. |
| Skalierbarkeit | Das System soll so entworfen werden, dass es auf mehrere hundert bis tausend Anwender ausgeweitet werden kann. |
| Lokalisierung | Das CMS soll in unterschiedlichen Sprachen verfügbar sein. Alle Inhalte können in mehreren Sprachen erfasst werden. |

### 6.3.6 Anforderungen App

Die vorhandene App wird vom Stand der Semesterarbeit übernommen. Aufgrund der neuen Mandantenfähigkeit und der Rundgang Erweiterung muss die App weiterentwickelt werden. Folgend werden die erweiternden Anforderungen an die App aufgezählt:

#### *UC2.1: Museum auswählen*

Benutzer können aus einer Liste von Museen jenes auswählen, in dem sie sich befinden. Als optionale Erweiterung soll dies automatisch aufgrund der Lokalisierung geschehen.

#### *UC2.2: Rundgang starten*

Benutzer können das App öffnen und einen Rundgang in der gewünschten Sprache starten.

### 6.3.7 Benutzer Tests

Die CMS wird vom Naturmuseum St. Gallen getestet. In Zentrum stehen dabei folgende Fragen:

- Ist die der Aufbau und die Navigation verständlich
- Können Inhalte selbständig angepasst werden
- Sind die Masken ansprechend und Benutzerfreundlich
- Allgemeines Feedback

### 6.3.8 Ansichten

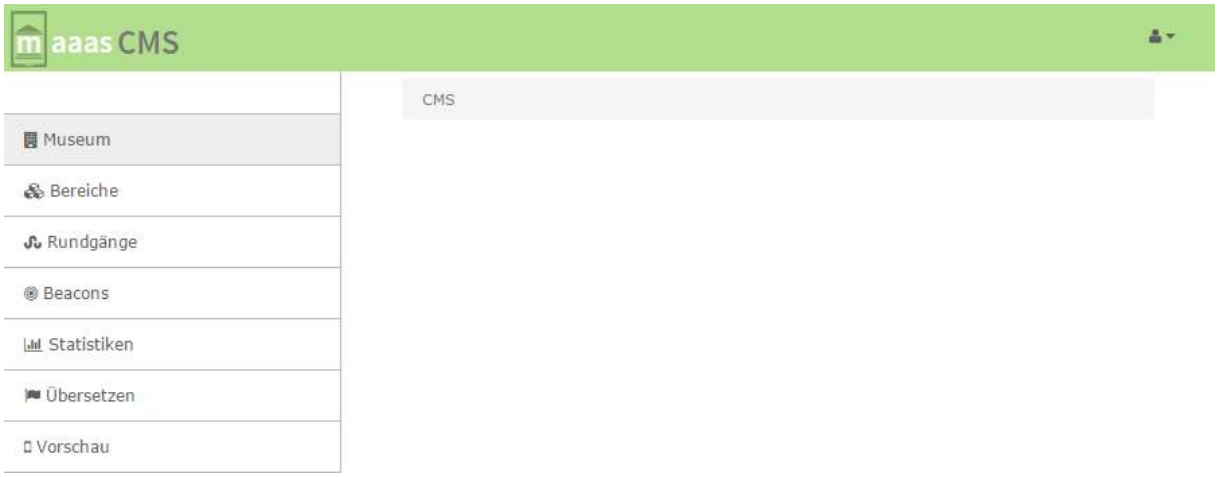



*Abbildung 6-15 Screenshot CMS Gliederung*

Das CMS ist grob gegliedert in Header, Navigationsleiste und Hauptansicht.

Im Header werden ein Logo, Informationen zum Benutzer sowie ein Menü-Punkt zur Bearbeitung dieser Informationen dargestellt.

In der Hauptansicht wird die Ansicht des aktuellen Navigationspunktes dargestellt und eine Breadcrumb Navgationsleiste für die Orientierung und für einfaches Zurücknavigieren.

Die Navigationsleiste ermöglicht die Navigation innerhalb des CMS und umfasst mit vollen Zugriffsrechten sieben Einträge. Diese werden nachfolgend im Detail beschreiben.

#### 6.3.8.1 Museum

Gibt den Mitarbeitern des Museums die Möglichkeit, die Details des Museums zu bearbeiten. Dazu gehören Informationen wie der Name, Adresse und Logo.

#### 6.3.8.2 Bereiche

Die Übersicht-View zeigt die erfassten Bereiche an sowie einen Button für die Erstellung eines neuen Bereiches. Die Auflistung erfolgt gekachelt, eine einzelne Kachel präsentiert jeweils eine Art Vorschau des dahinterliegenden Bereiches. Mit einem Klick auf eine Kachel gelangt man auf die Detail-View des Bereiches.

Die Detail-View enthält Eingabefelder, um Titel und Präsentationsdetails der Bereiche zu editieren. Sie wird simultan für das Erstellen, Anzeigen und Bearbeiten von Bereichen verwendet. Sie enthält ausserdem Navigationsbuttons um Inhalte und Quiz für die Bereiche zu verwalten.

### *Inhalte*

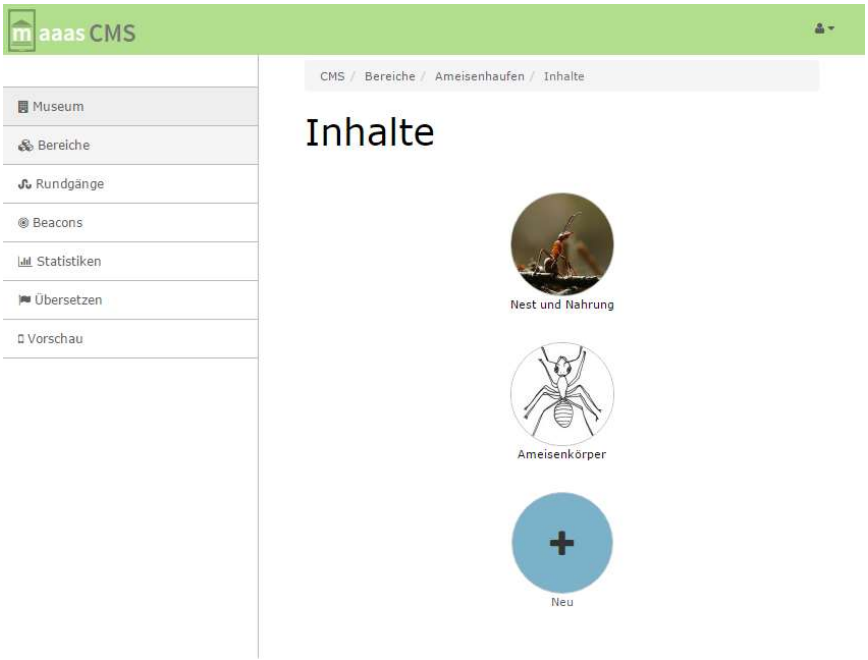



*Abbildung 6-16 Screenshot CMS Inhalt Überisicht*

Auch die Inhalte haben eine Übersicht über bestehende Einträge mit der Möglichkeit neue zu erstellen oder vorhandene zu löschen.

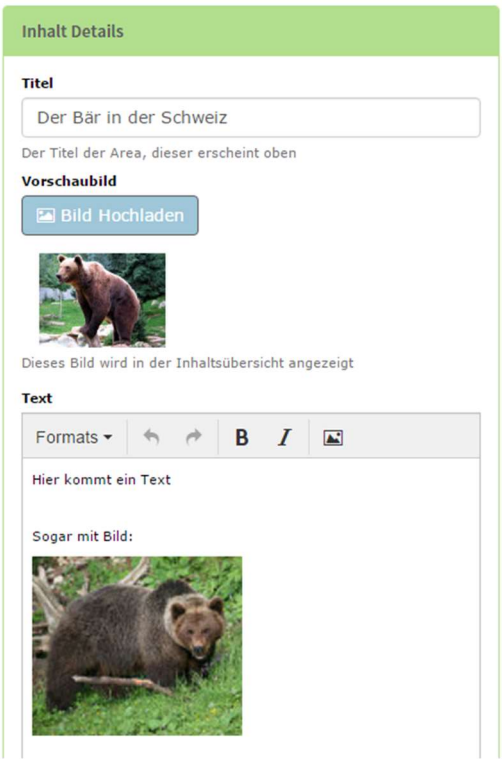



*Abbildung 6-17 Screenshot CMS Inhalt Details*

Die Detail-View enthält Eingabefelder, um Titel und Präsentationsdetails des Inhalts zu editieren. Sie enthält ausserdem einen Rich-Text-Editor, mit welchem der Text des Inhalts bearbeitet werden kann. Die Detail-View wird simultan für das Erstellen, Anzeigen und Bearbeiten von Inhalten verwendet.

#### *Quiz*

Auch bei den Quiz gibt es analog zu den Inhalten und den Bereichen eine Übersicht.

Die Detail-View eines Quiz ist abhängig vom Quiz-Typ. Es werden jeweils entsprechende Eingabefelder für die Bearbeitung der Quiz-Inhalte angezeigt. Die Detail-View wird simultan für das Erstellen, Anzeigen und Bearbeiten von Quiz verwendet.

### 6.3.8.3 Rundgänge

Die Detail-View enthält ein Eingabefeld für den Namen sowie ein Upload-Button für ein optionales Bild. Es können die Bereiche ausgewählt werden, welche in diesem Rundgang vorkommen sollen. Für jeden Bereich können die Inhalte und Quiz selektiert werden, welche in diesem Rundgang vorkommen sollen.

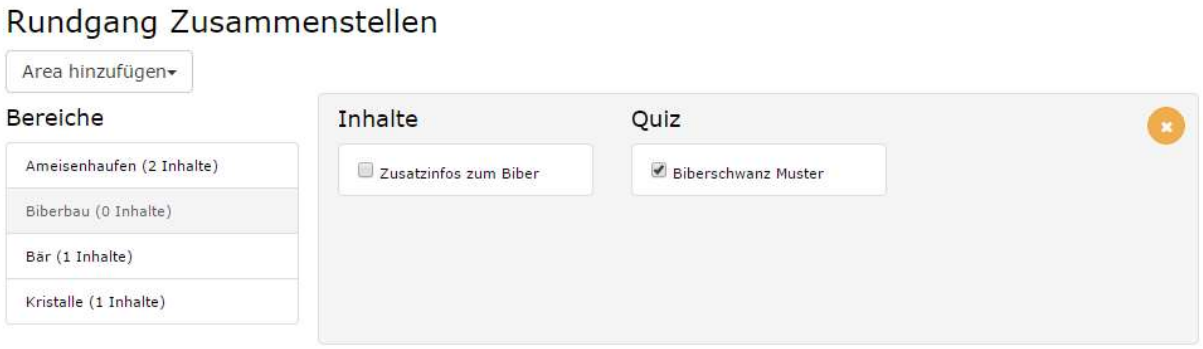



*Abbildung 6-18 Screenshot Rundgang Detail*

#### 6.3.8.4 Beacons

Auf der Beacon-View werden die Beacons aufgelistet, welche im Museum installiert sind/werden. Jedem Beacon kann ein Bereich zugewiesen werden.

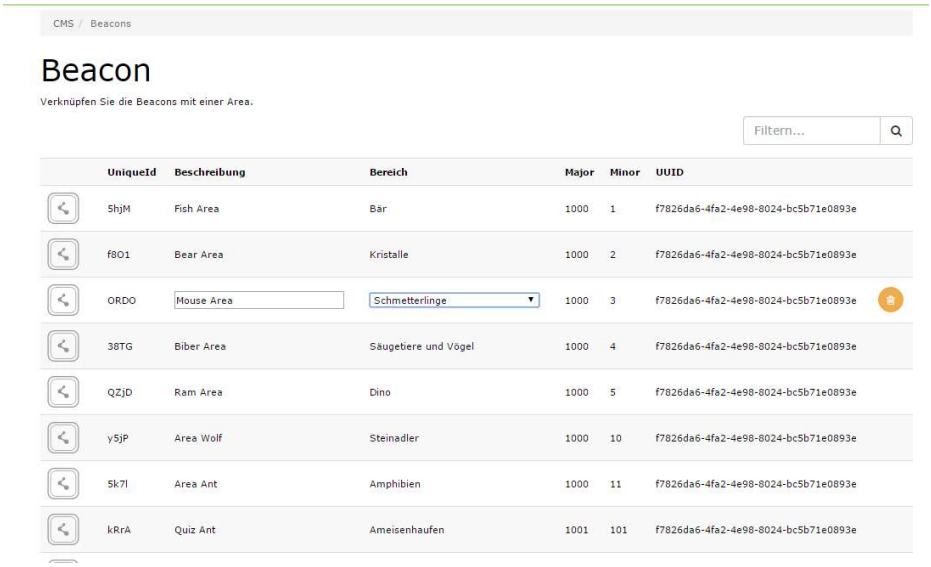



*Abbildung 6-19 Screenshot CMS Beacons*

#### 6.3.8.5 Statistiken

Auf der Statistik-View werden diverse Statistiken angezeigt. Diese umfassen in einem ersten Schritt Verwendungsstatistiken der App.

#### 6.3.8.6 Übersetzen

Auf der Übersetzen-Ansicht hat der Benutzer die Möglichkeit die Inhalte zu übersetzen. In einem Wizard-Modus kann man sich durcharbeiten und hat einen Indikator dafür, wie viel bereits übersetzt ist.

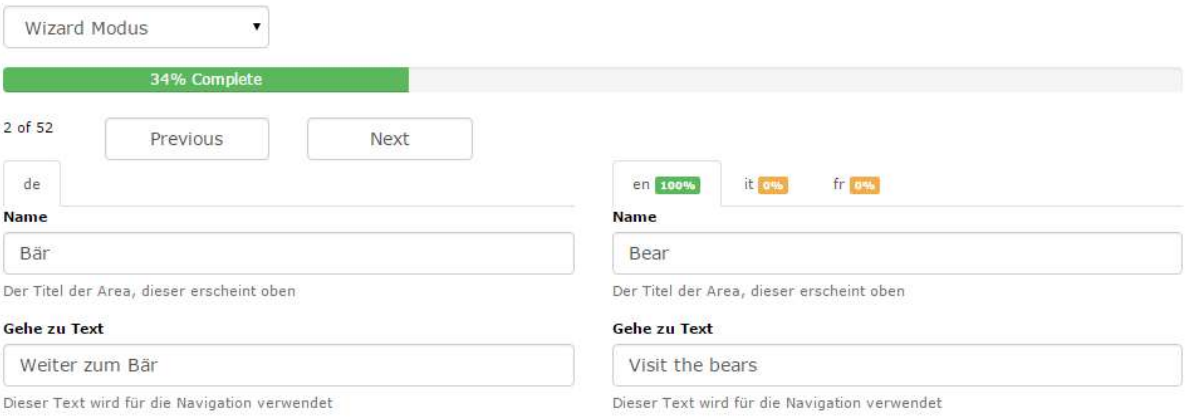



*Abbildung 6-20 Screenshot CMS Übersetzung Wizard*

#### 6.3.8.7 **Vorschau**

Dieser Navigationspunkt Vorschau öffnen ein Popup mit einer Vorschau auf das App. So ist es möglich während des bearbeiten der Elementes jeweils eine Vorschau zu haben bevor man speichert.

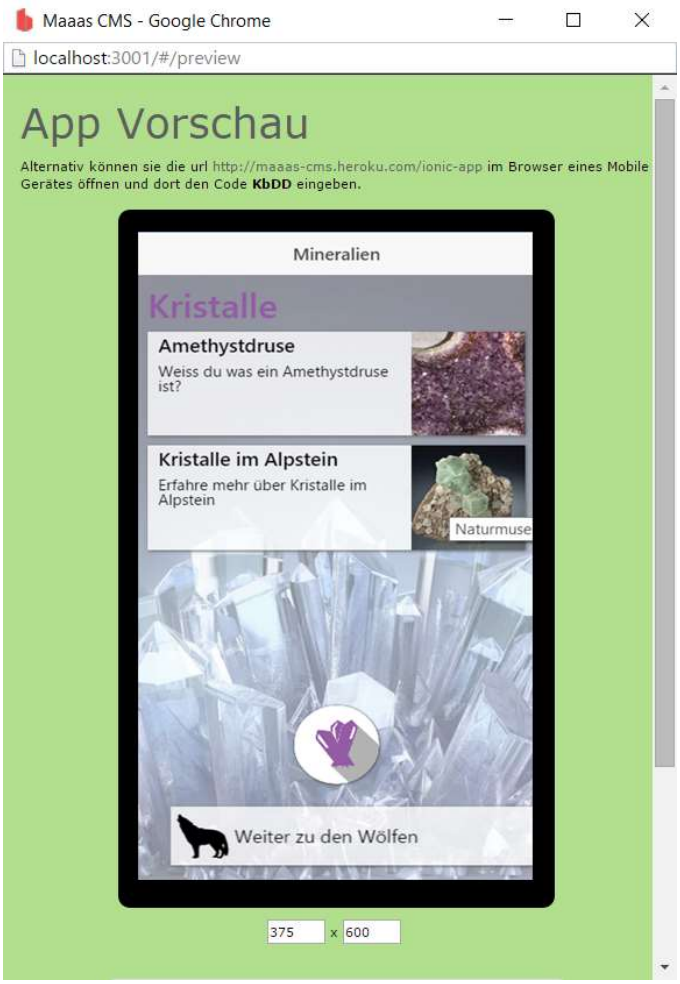



*Abbildung 6-21 Screenshot CMS Privew Popup*

# 6.4 Architekturdokument

## 6.4.1 Systemübersicht

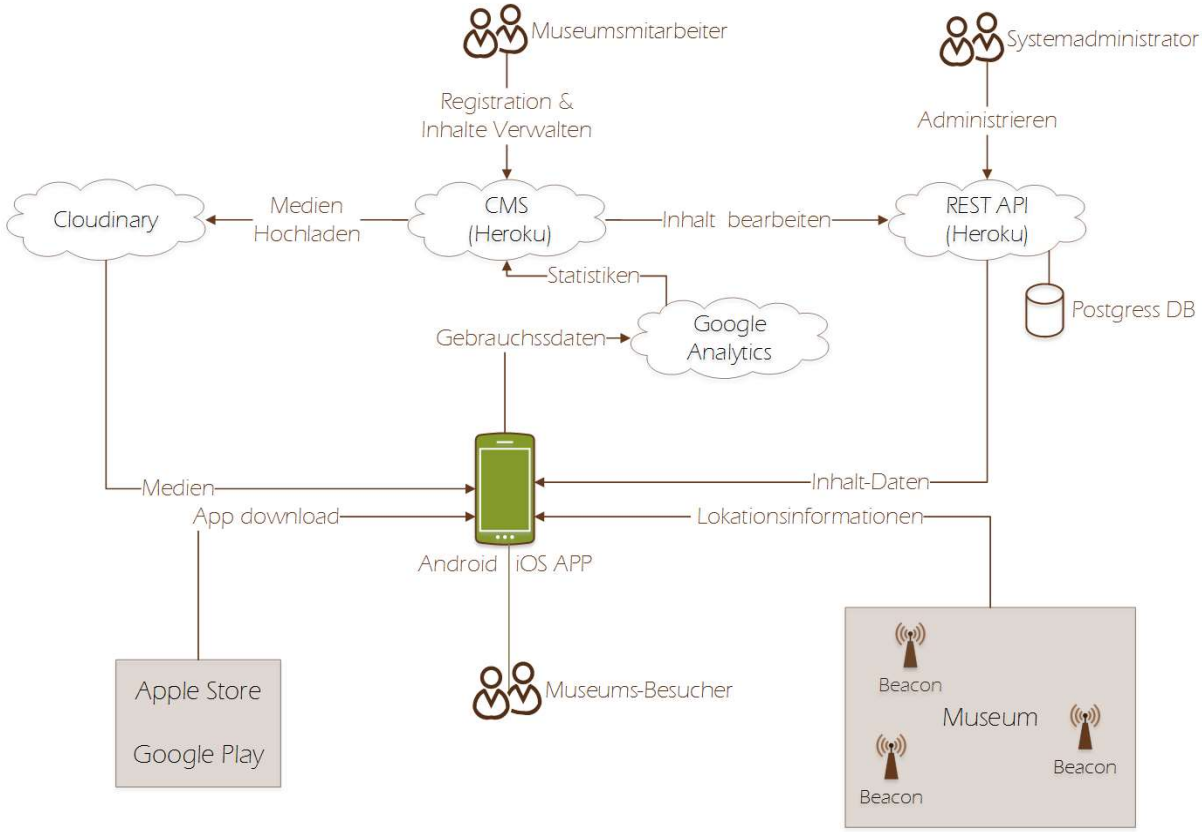



*Abbildung 6-22 Systemübersicht*

### 6.4.2 Entitäten Übersicht

Nachfolgend werden die Haupt-Entitäten des CMS und der App kurzbeschreiben und die Begriffe einzuführen und deren Relationen aufzuzeigen.

| Entität | Intern | Beschreibung |
|---|---|---|
| **Museum** | Museum | Übergeordnete Entität um Mandantenfähigkeit zu ermöglichen |
| **Bereich** | Area | Ein Ort in der Ausstellung welcher ein oder mehrere Ausstellungsobjekte enthalten kann. Die App merkt, wenn sich der Besucher in einen neuen Bereich begibt. Bereich Beispiele vom Naturmuseum: Bär, Wolf. Adler |
| **Quiz** | Challenge | Quiz welche der Benutzer in der App lösen kann. Es gibt verschiedene Quiz-Typen wie z.B. Multiplechoice oder Wahr/Falsch Fragen |
| **Inhalt** | Content | Zusatzinformationen, welche auf dem Bereich dargestellt werden,liefern den Besuchern mehr Informationen zu einem Thema und können auch multimediale Inhalte enthalten. |
| **Rundgang** | Tour | Ein Rundgang besteht aus einer Selektion von Inhalten und Quiz der Bereiche. So kann man Zielgruppen spezifische Inhalte liefern. |
| **Einladung** | Invitation | Ein Museum kann andere User unter der Angabe einer Emailadresse einladen. Die Einladung kann dann vom User angenommen werden, womit er dem Museum hinzugefügt wird. |

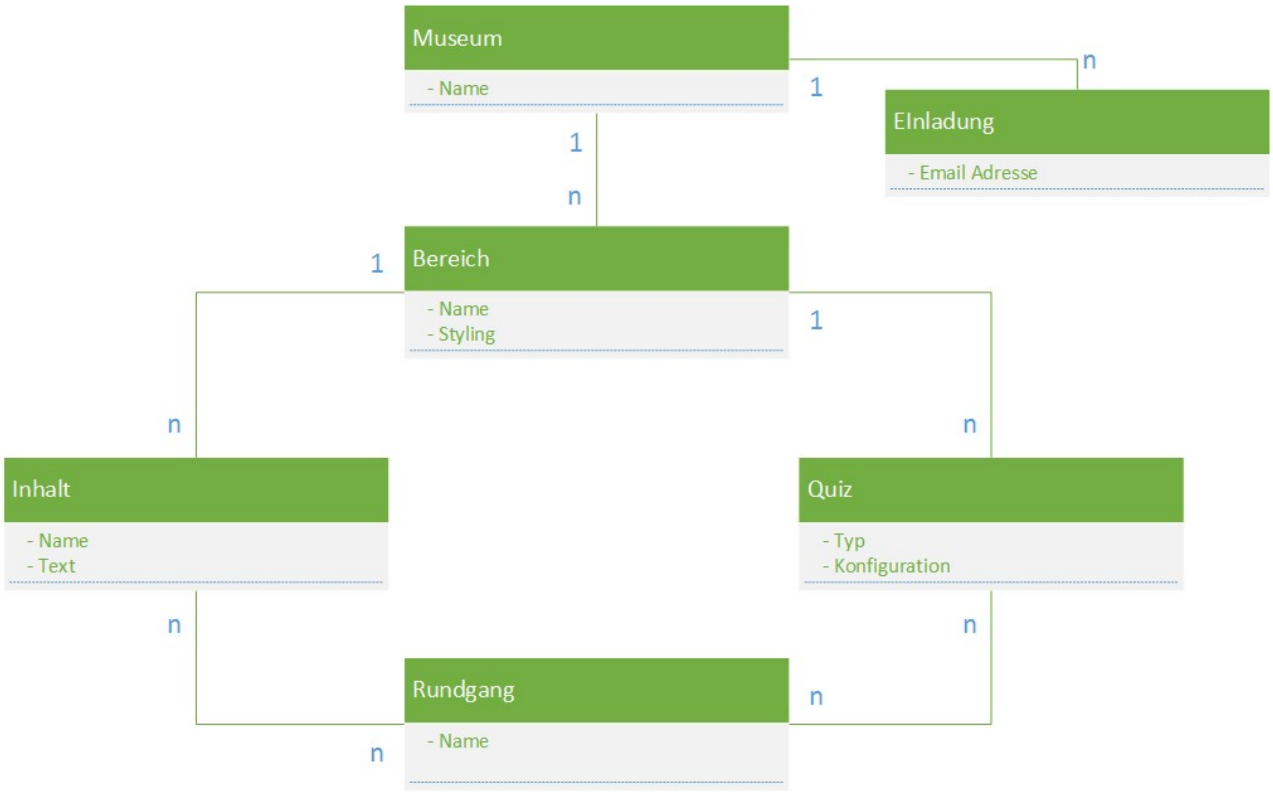



*Abbildung 23 Entitäten Abhängigkeits Übersicht*

### 6.4.3 Eingesetzte Technologien

#### 6.4.3.1 AngularJS 1

Angular ist das wohl bekannteste SPA-Framework. Es bietet gute Konzepte um eine saubere Architektur und Testbarkeit einer Applikation zu gewährleisten. Es ermöglicht Two-Way-Bindings zwischen View und Model.

*Gründe für Einsatz: Sehr populär, beide Projekt-Mitglieder hatten damit am meisten Erfahrung.*

#### 6.4.3.2 TypeScript

TypeScript ist ein Super-Set von JavaScript. Es werden viele Konzepte eingeführt, welche in der ECMAScript5 Spezifikation nicht vorhanden sind. Dazu gehören Dinge wie Klassen und Module. Ein grosser Vorteil ist die optionale Typisierbarkeit von Variablen und Parametern.

*Gründe für Einsatz: Angular2 wird in TypeScript entwickelt und die empfohlene Sprache für Angular2-Applikationen ist TypeScript. Auch sind die oben genannten Konzepte (Klassen, Module, Typisierbarkeit) ein grosser Vorteil gegenüber Alternativen wie z.B. CoffeeScript oder Plain JavaScript.*

#### 6.4.3.3 Jade

Jade ist eine spezielle HTML-Syntax. Ähnlich wie bei CoffeeScript ist die Syntax knapp und elegant gehalten.

*Alternativen*

| Name | Nicht eingesetzt weil... |
|---|---|
| Plain HTML | Aufwändigere Syntax |

#### 6.4.3.4 SCSS

SCSS (bzw. SASS) dient dazu, CSS-Code besser zu organisieren. Es können z.B. Variablen verwendet werden sowie Kontrollstrukturen wie For-Loops.

*Alternativen*

| Name | Nicht eingesetzt weil... |
|---|---|
| Plain CSS | Organisation Umfangreicher Stylesheets ist sehr schwierig, es gibt keine Möglichkeiten für Code Re-Use, keine Scripting-Möglichkeiten, etc. |
| LESS | SCSS wird standardmässig von Ionic eingesetzt |

#### 6.4.3.5 GulpJS

GulpJS ist ein Buildsystem, mit welchem die oben genannten Sprachen (TypeScript, Jade, SCSS) in die Zielsprachen kompiliert werden können.

*Alternativen*

| Name | Nicht eingesetzt weil... |
|---|---|
| BroccoliJS | Das Einrichten des Buildprozesses für TypeScript war sehr aufwändig und unflexibel |
| Grunt | Langsam, kein Caching |

#### 6.4.3.6 KarmaJS

KarmaJS ist ein Testrunner, der Unit Tests automatisch und manuell ausführen kann. KarmaJS wird vom AngularJS-Team gewartet und verwendet.

*Alternativen wurden keine in Betracht gezogen, da KarmaJS von Anfang an hervorragend funktionierte.*

#### 6.4.3.7 Ruby on Rails

Ruby on Rails ist ein Full-Stack MVC-Web-Framework. Für dieses Projekt wurde es für die Implementation des Rest-APIs verwendet.

*Gründe für Einsatz: Einziges Web-Framework, mit dem das Projekt-Team genügend Erfahrung hatte, um produktiv zu arbeiten. Kann ausserdem sehr leicht und gratis auf Heroku deployed werden.*

#### 6.4.3.8 Github

Github ist ein Host für Git-Repositories. Es stellt kollaborative Funktionen wie Issue-Verwaltung oder Pull-Requests zur Verfügung.

*Alternativen*

| Name | Nicht eingesetzt weil... |
|---|---|
| Bitbucket | Projekt-Team hat keine Erfahrung damit und Github ist absolut ausreichend |

#### 6.4.3.9 Travis-CI

Travis ist eine Server-Infrastruktur, auf welcher automatische Builds durchgeführt werden können.

*Gründe für Einsatz: Nahtlose Integration mit Github und KarmaJS. Alternativen wurden nicht in Betracht gezogen, weil Erfahrung mit Travis im Projekt-Team bereits vorhanden war.*

### 6.4.4 Code Guidelines

Für TypeScript und Jade wurde mit Gulp ein Linting-Task eingeführt. Das bedeutet, dass bei jeder entsprechenden Fileänderung definierte Code-Guidlines analysiert werden. Bei nicht einhalten werden Fehler auf der Konsole ausgegeben. Das Linting wird ebenfalls von Build-Server Travis überprüft. Nachstehend werden die wichtigsten Guidlines angegeben.

#### 6.4.4.1 TypeScript

- Keine Leerschläge an Zeilenenden
- Durchgehendes verwenden der einfachen Hochkommas (nicht die doppelten)
- Verwenden von dreifachen Gleichheitszeichen für Vergleiche
- Semikolon am Schluss eines Statements
- Maximale Zeilenlänge von 140 Zeichen

#### 6.4.4.2 Jade

- Durchgehendes verwenden der einfachen Hochkommas (Nicht die doppelten)
- Einrückung mit zwei Leerschlägen
- Keine Leerschläge an Zeilenenden
- Eine einzige Leerzeile am Ende eines Files
- Klassen werden mit der Dot-Notation angegeben

### 6.4.5 Projektstruktur

#### 6.4.5.1 Front-End

Es gibt zwei populäre Arten, wie man eine AngularJS-Applikation gliedern kann: Gliederung nach Typ (also nach Controllers, Directives, Services, etc), oder Gliederung nach Use Case (also z.B. Login, Contents, Challenges, etc). Bei diesem Projekt kam der zweitere Ansatz zum Zug.

```
|-app
|---areas
|---beacons
|---challenges
|---common
|---contents
|---login
|---master
|---media
|---museums
|---preview
|---registration
|---statistics
|---tours
|---translation
|---utils
|-fonts
|-img
|-styles
|-tests
```

***app***

Enthält pro UseCase einen Folder. Jeder dieser Folder enthält die zugehörigen TypeScript, Jade und Übersetzung-Files. Gemeinsam verwendete Komponenten werden im Ordner Common verwaltet. So gibt es zum Beispiel ein Ordne Area. Dort drin enthalten sind alle Ansichten und Services welche mit dem Bearbeiten eines Bereiches zusammenhängen.

***fonts***

Enthält vom CMS benötigte Schriftarten.

***img***

Vom CMS benötigte Bilder.

***styles***

SCSS gegliedert nach UseCase Files.

***tests***

Test Files.

#### 6.4.5.2 Back-End

Das Back-End ist nach den Konventionen von Rails gegliedert und bedarf hier keiner besonderen Bemerkungen. [23]

### 6.4.6 Layers

#### 6.4.6.1 Front-End

Das Front-End wurde mit Angular-JS entwickelt. Die Applikation ist mit der Idee konzipiert, dass jedes UI-Element als sogenannter "Web-Component" realisiert ist. Die Idee von Web-Components ist kein AngularJS-eigenes Konzept, sondern wird von vielen weiteren JS-Libraries verwendet. Ein Web-Component besteht aus Markup-Code (HTML) sowie zugehörigem Controller-Code (JavaScript). In AngularJS werden Web-Components mit Directives realisiert.

Als Schnittstelle zum REST-Api wurde die OpenSource-Library JS-Data verwendet. Dies ist eine JavaScript-Bibliothek, mit welcher man ein Domain-Modell modellieren kann. Man kann also Entities und deren Abhängigkeiten zueinander deklarieren. Unter Einhaltung der REST-Konventionen setzt JS-Data dann entsprechende Requests ab.

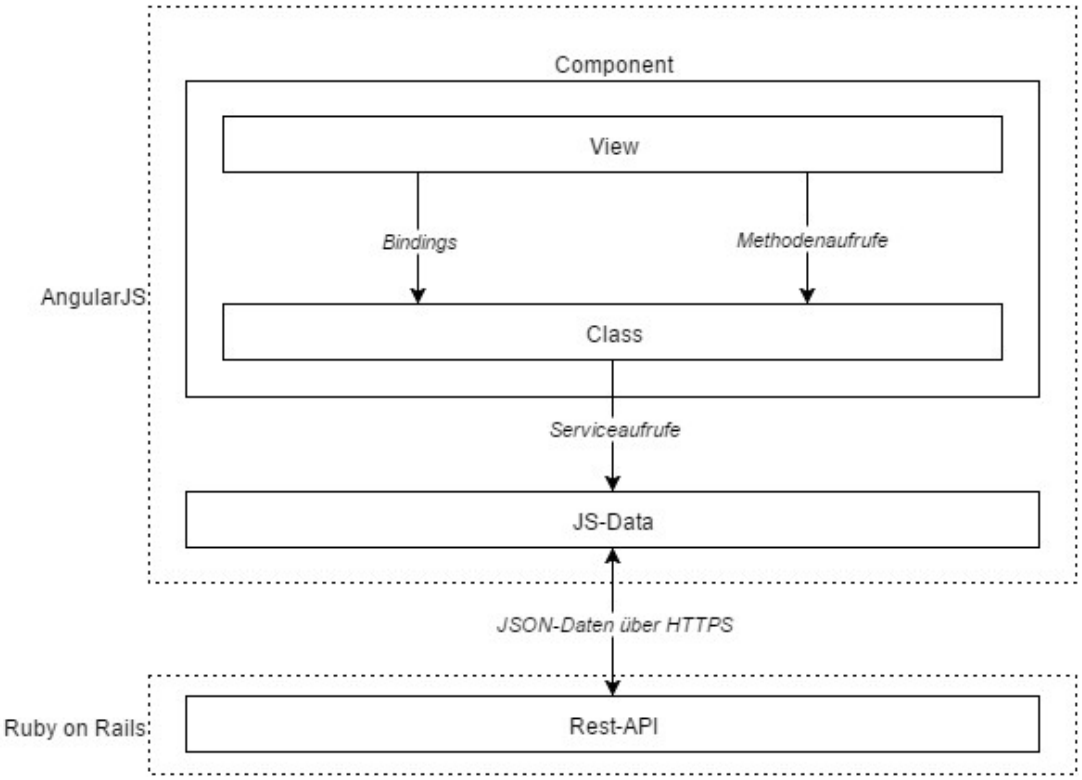



*Abbildung 6-24 Layer Übersicht*

#### 6.4.6.2 Back-End

Im Back-End wird eine Relationale Datenbank verwendet. Jede Entity, welche von einem Museum verwaltet wird, ist mit der entsprechenden Museums-ID referenziert. Das heisst also, dass z.B. ein Content eine Museums-ID besitzt, obwohl die Relation zum Museum über die Zusammengehörigkeit des Contents zur Area hergestellt werden könnte. Die dadurch entstandene Redundanz wurde in Kauf genommen mit der Begründung, dass so die Authorisierungslogik einfacher und effizienter implementiert werden kann.

Rundgänge (Tabelle 'tours') sind so modelliert, dass Contents und Quiz mit einer Tour-Entity verknüpft werden. Das heisst also, dass die Zugehörigkeit von Areas implizit über die Referenzen zu Contents und Quiz modelliert ist.

Die Abhängigkeiten zum Museum wurde nur für die Areas dargestellt, um das Diagramm zu entlasten.

Die Übersetzungstabellen sind auf diesem Diagramm der Einfachheit halber nicht ersichtlich.

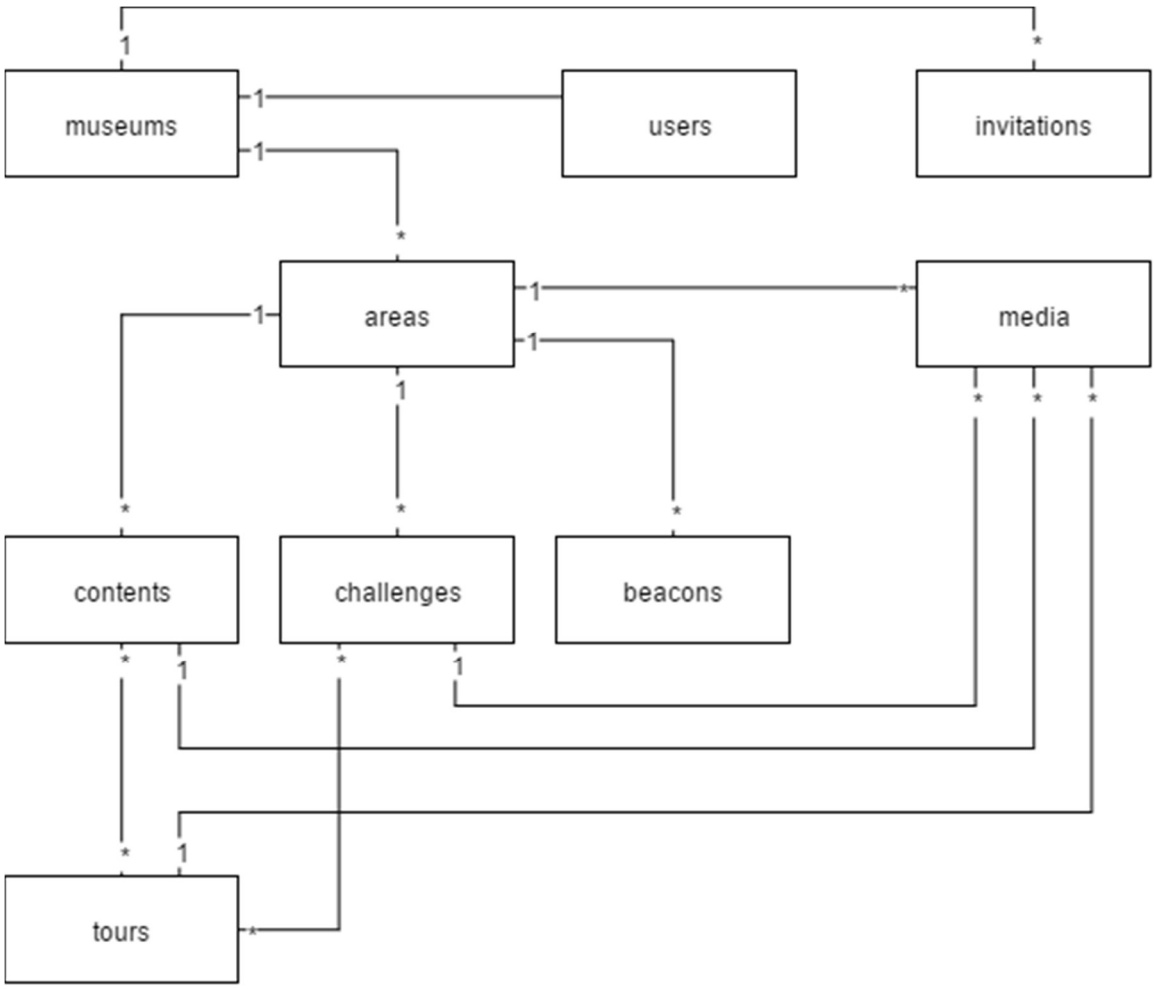



*Abbildung 6-25 Datenbank-Modell*

## 6.4.7 Problemstellungen und Lösungen

### 6.4.7.1 Lokalisierung

Eine Anforderung an das CMS war die Mehrsprachigkeit. Das gesamte Benutzer-Interface sollte in verschiedene Sprachen übersetzt werden können. Dazu wurde das Open-Source-Modul *angular-translate* verwendet, welches sehr verbreitet ist.

Man kann davon ausgehen, dass dieser Teil mit einer allfälligen Migration auf Angular2 angepasst werden muss, da die Internationalisierung fixer Bestandteil von Angular2 sein wird. Ein erster Schritt dafür wurde mit ngMessageFormat in Angular1.4 eingeführt [24].

*angular-translate* wurde so konfiguriert, dass es pro Feature und Sprache ein eigenes Übersetzungsfile gibt, welches separat geladen wird:

```
app.config(function ($translateProvider,
$translatePartialLoaderProvider) {
  $translatePartialLoaderProvider.addPart(„area“);
});
```

*Abbildung 6-26 Konfiguration von angular-translate*

Dieser Code bewirkt, dass die Sprachfiles der Area initialisiert werden. Ein solches Sprachfile muss im „Translations“-Ordner liegen. Der Prefix der Keys ist einzuhalten damit man nicht übergreifende Duplikate erstellt. Die Übersetungsfiles haben ein JSON Format:

```
{
  "areas_title": "Bereiche",
  "areas_details_title": "Area Details"
}
```

*Abbildung 6-27 Beispiel Übersetzungsdirektiven*

Diese Konfiguration lässt die Verwendung von lokalisierten Texten im Template sowie in TypeScript zu mittels der Verwendung des Filters:

```
h1 {{ 'areas_details_title' | translate }}
```

```
this.$filter('translate')('areas_details_title')
```

*Abbildung 6-28 Einsatz des translate filters*

### 6.4.7.2 Formular-Komponenten

Die Formulare im CMS wurden möglichst benutzerfreundlich gestaltet und umfassen folgende Funktionalität:

- Es wird ein Hilfe-Text unterhalb eines Eingabefeldes zur Erläuterung des zugrundeliegenden Feldes angezeigt
- Ein Eingabefeld kann mit einer Validierung versehen werden. So kann z.B. die Mindestlänge einer Eingabe forciert werden
- Die oben genannten Texte sind lokalisiert

Um diese Anforderungen möglichst leicht für jedes Feld umzusetzen, wurde ein Angular-Directive geschrieben, welches die Deklaration stark vereinfacht. Der folgende Code-Ausschnitt illustriert dies:

```
<mas-form-group field-name="name"
    localization-prefix="areas_details_name">
  <input
    ng-model="ctrl.area.name"
    name="name"
    required="required"
    ng-minlength="2"/>
</mas-form-group>
```

*Abbildung 6-29 Beispiel Formular-Komponente*

#### 6.4.7.3 Bild-Upload

Im CMS soll der Upload von Bildern ermöglicht werden.

##### *Cloudinary*

Sämtliche Bilder werden beim Cloud-Anbieter Cloudinary gespeichert. In der Datenbank des CMS werden lediglich Referenzen auf die von Cloudinary generierten IDs abgelegt. Der Upload eines Bildes kann mit folgendem simplen Sequenzdiagramm veranschaulicht werden:

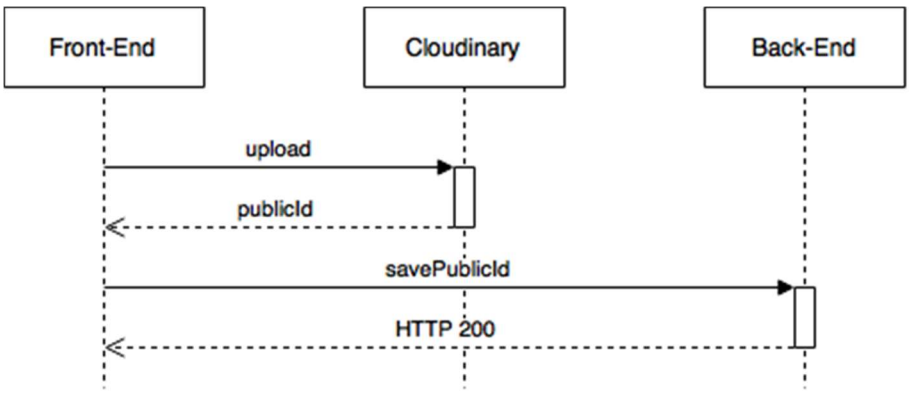



*Abbildung 6-30 Bild-Upload*

Das obige Diagramm gilt für den Fall, bei dem eine Entity, welche Referenzen auf Bilder haben kann (z.B. Area), bereits erstellt ist und eine Id besitzt. Beim Erstellen einer Entity existiert noch keine Id, weshalb auch keine Referenzen zu Bildern abgespeichert werden können. In diesem Falle muss darauf gewartet werden, bis die Entity und somit eine Id erstellt ist, danach können alle zuvor hochgeladenen Bilder mit der Entity verknüpft werden:

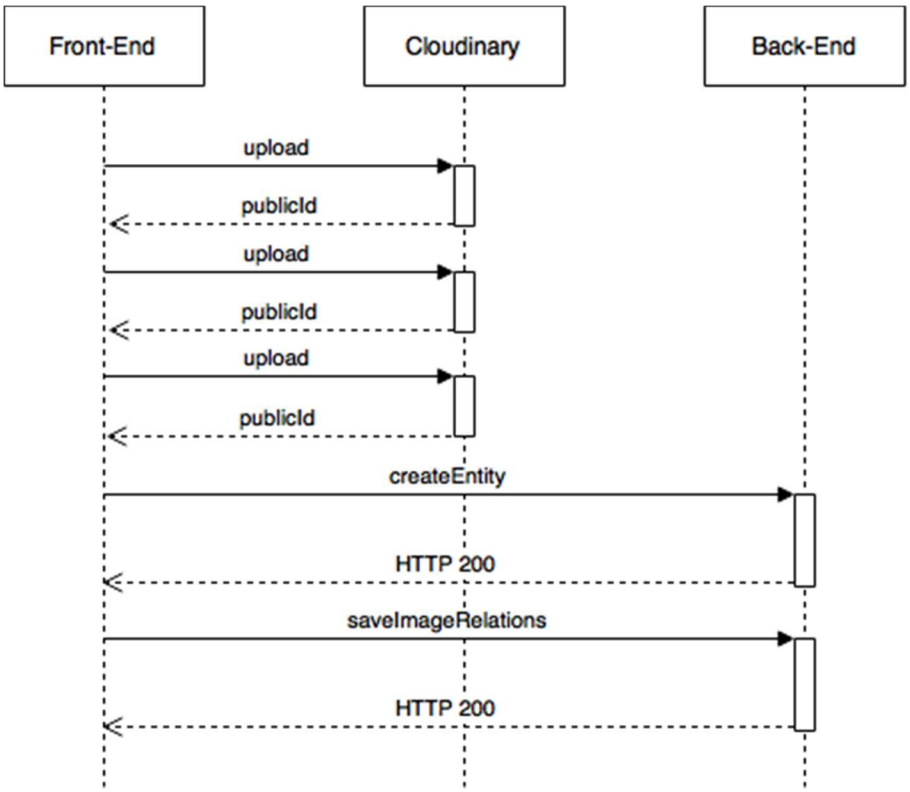



*Abbildung 6-31 Bild-Upload nach dem Erstellen einer Entity*

### Reusability im Front-End

Damit auf Formularen möglichst einfach Bild-Uploads realisiert werden können, wurde ein Angular-Directive entwickelt, mit welchem man auf elegante Weise einen Bild-Upload deklarieren kann:

```
<mas-image-form-wrapper
  image-id='stickerImage'
  model='ctrl.area.stickerImageId'
  height='100'
  is-required='true'
/>
```

*Abbildung 6-32 Bild-Upload-Directive*

Dieses Directive löst die folgenden Probleme:

- Es erkennt anhand des gegebenen Models, ob ein Bild bereits hochgeladen wurde oder nicht und passt dementsprechend die Darstellung an:
  - Ist das gegebene Model undefined, wird ein Placeholder dargestellt
  - Ist das gegebene Model nicht undefined, wird zunächst der Media-Eintrag aus dem Back-End geholt, mit welchem dann die URL des Bildes auf Cloudinary geholt werden kann
- Es stellt einen Upload-Button zur Verfügung, mit welchem ein Bild von der Disk Hochgeladen werden kann. Das Bild wird zunächst an Cloudinary geschickt, dann wird eine Referenz dazu im Backend abgespeichert
- Es verschickt Events nach einem erfolgreichen Image-Upload

- Es nimmt ein height- und width-Property entgegen, mit welchen die Thumbnail-Dimensionen deklariert werden können

#### 6.4.7.4 Verschiedene Quiz-Typen

Das CMS erlaubt die Bearbeitung verschiedener Quiz-Arten. Um diese Quiz zu modellieren, kamen zwei Arten in Frage: Relationale Modellierung in der Datenbank oder Abspeichern als JSON-Struktur.

Es wurde entschieden, die Quiz also JSON-Strukturen abzulegen, was folgende Vorteile mit sich bringt:

- Maximale Flexibilität: die relationale Modellierung wäre sehr aufwändig und würde zu einem komplexeren Datenbankmodell führen
- Übersetzbarkeit: die Übersetzbarkeit zu modellieren wäre umso komplexer geworden
- JSON-Strukturen können in JavaScript-Objekte umgewandelt werden. Auf dem Front-End können die Quiz-Daten also sehr leicht eingebunden und bearbeitet werden

```
"data": {
  "name": "Ameisen Silhouette",
  "kind": "multiple-choice",
  "question": "Welche der unten stehenden Abbildungen stimmt?",
  "answers": [{
    "idx": 1,
    "imageId": 4431
  }, {
    "idx": 2,
    "imageId": 4432
  }, {
    "idx": 3,
    "imageId": 4433
  }],
  "correctAnswer": "3",
  "type": "image",
  "successfullyMessage": "Richtig!"
}
```

*Abbildung 6-33 Daten-Struktur des multiple-choice Quiz*

Ein Nachteil dieses Konzeptes ist, dass für die Modellierung der Fremdschlüsselbeziehung zur Media-Tabelle (Bilder) Redundanzen in Kauf genommen werden müssen: Jedes Bild, welches für ein Quiz hochgeladen wird, wird relational mit der Quiz-Instanz verknüpft, gleichzeitig aber auch in der JSON-Struktur abgespeichert. Auf diese Art kann man also per SQL-Abfragen herausfinden, welche Bilder zu welchen Quiz gehören, ohne dass die JSON-Struktur geparst werden muss. Wenn ein Bild nachträglich aus der JSON-Struktur gelöscht wird, müsste auch die

Referenz zum Media-Objekt aus der Datenbank gelöscht werden. Dies ist aktuell noch nicht implementiert und wäre ein wichtiger Punkt bei einer Weiterführung des Projektes.

#### 6.4.7.5 Übersetzungen

Im CMS soll es ermöglicht werden, alle Areas, Inhalte, Quiz und Rundgänge zu übersetzen.

##### *Datenbankstruktur*

Um Übersetzungen aller Inhalte zu ermöglichen, wurde im Backend eine Library namens Globalize eingesetzt. Diese Library sieht vor, dass pro Tabelle, welche übersetzte Felder haben soll, eine zugehörige Übersetzungstabelle erstellt wird. Diese Übersetzungstabelle speichert pro Eintrag alle zu übersetzenden Felder für eine Entity und einen Locale. Globalize automatisiert dann das Zusammenjoinen der Tabellen.

Auf dem folgenden Diagramm sieht man dies am Beispiel der Tabelle contents. Auf dieser Tabelle gibt es die Felder title, description und data, welches alles Felder sind die übersetzt werden müssen. Zusätzlich gibt es die Tabelle content_translations, welche ebendiese Felder enthält, zusätzlich mit einem Feld für den Locale sowie dem Fremdschlüssel des zugehörigen Contents.

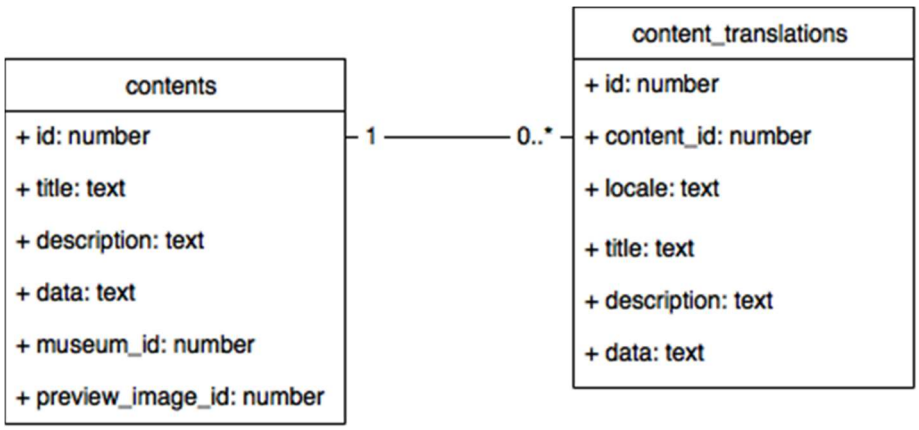



*Abbildung 6-34 Datenbank-Modell für Übersetzungen der Contents*

##### *Front-End*

Wenn Inhalte im Front-End editiert werden, werden diese jeweils unter dem Default-Locale des Users abgespeichert. Auf der Übersetzungs-View hat man dann die Möglichkeit, die Inhalte in weitere Sprachen zu übersetzen.

Damit das Front-End die bereits übersetzten Inhalte vom Back-End holen kann, kann der Request, um den entsprechenden Inhalt zu holen, mit einem Parameter versehen werden, z.B.: /areas?translations=yes. Das Backend schickt dann im JSON-Response zusätzlich ein Feld mit allen Übersetzungen mit. Hier ein Beispiel einer Area mit Übersetzungen:

```
{
  "name": "Ameisenhaufen",
  "gotoText": "Weiter zum Ameisenhaufen",
  "translations": {
    "en": {
      "name": "Anthill",
      "gotoText": "Proceed to the anthill"
    },
    "it": {
      "name": null,
      "gotoText": null
    },
    "fr": {
      "name": null,
      "gotoText": null
    }
  }
}
```

*Abbildung 6-35 Übersetzte Felder eines Area-Eintrags*

Diese JSON-Struktur kann nun in der Angular-Applikation direkt verwendet und entsprechend auf Eingabefelder Two-Way-gebindet werden. Zum Abspeichern wird die genau gleiche Struktur wieder an den Server geschickt, welcher die Werte für jeden einzelnen Locale abspeichert:

```
if translations = params[model_key][:translations]
  allowed_locales.each do |locale|
    Globalize.with_locale(locale) do
      save(...)
    end
  end
end
```

*Abbildung 6-36 Speichern der einzelnen Locales mit Globalize*

### 6.4.7.6 App-Preview mit Web-Socket

Ziel war es den CMS Benutzer eine Vorschau der Elemente anzuzeigen welche er zurzeit editiert. Dies betrifft die Bereiche, die Quiz und die Inhalte.

Die App-Vorschau sollte sowohl in einen Popup auf der gleichen Workstation sowie auch geräteübergreifend funktionieren. Zweiteres bedeutet, dass der CMS Benutzer ein Element bearbeiten kann und auf einem anderen Gerät (z.B iPad) im geöffneten Browser sofort die Änderungen nachvollziehen kann.

#### *Umsetzung*

Damit man sämtliche Ansichten der App nicht im CMS nachprogrammieren muss, haben wir uns entschieden, die App ins CMS zu deployen. Da die App komplett auf HTML und JS basiert, läuft diese auch im Browser. Was nicht funktioniert sind sämtliche Cordova Plugins. Diese werden

aber für die Vorschau nicht benötigt. Die Navigation geschieht automatisch, wenn man im CMS ein Element öffnet, und die Beacons sind somit nicht relevant für die Vorschau.

Für eine saubere und geräteübergreifende Lösung der Kommunikation haben wir uns für Sockets mit Socket.io entschieden. Der Entscheid für socket.io ist mit dessen Verbreitung begründet und beide Autoren haben damit bereits Erfahrungen in anderen Projekten gesammelt.

Der Socket.io Server wurde auf dem Node Webserver des CMS eingerichtet und hört dort auf Nachrichten des CMS und pusht diese weiter an die Vorschau-Clients.

Damit sich ein geräteübergreifender Client einer Session zuordnen kann, wird er in der App aufgefordert, einen vierstelligen Code, welcher im CMS generiert wird, einzugeben. Der Socket Server kann dann die Zuweisung an einen Benutzer aufgrund der Benutzer-ID vornehmen.

Im CMS wird registriert, wenn ein Element bearbeitet wird. Da die Datenstruktur im CMS und in der App leicht abweichen wird es über einen Aufruf auf das Backend in das App-Format konvertiert und danach an den Socket Server gesendet. Die Konvertierung wird im Backend vorgenommen, da dort die Logik bereits implementiert ist weil die App die Daten von dort bezieht.

Die App wurde erweitert und hört direkt auf die Push-Nachrichten des Socket.io Servers. Das Socket-Paket enthält das gesamte Objekt welches angezeigt werden muss. Das App nimmt diese Daten entgegen, lädt diese in die interne Datenverwaltung und navigiert zur passenden Ansicht.

Folgendes Sequenz-Diagramm veranschaulicht den Ablauf mit einer Area. Der Einfachheit halber wird nur ein Vorschau-Client dargestellt und vernachlässigt, dass ein CMS Client auch ein Preview-Client sein kann.

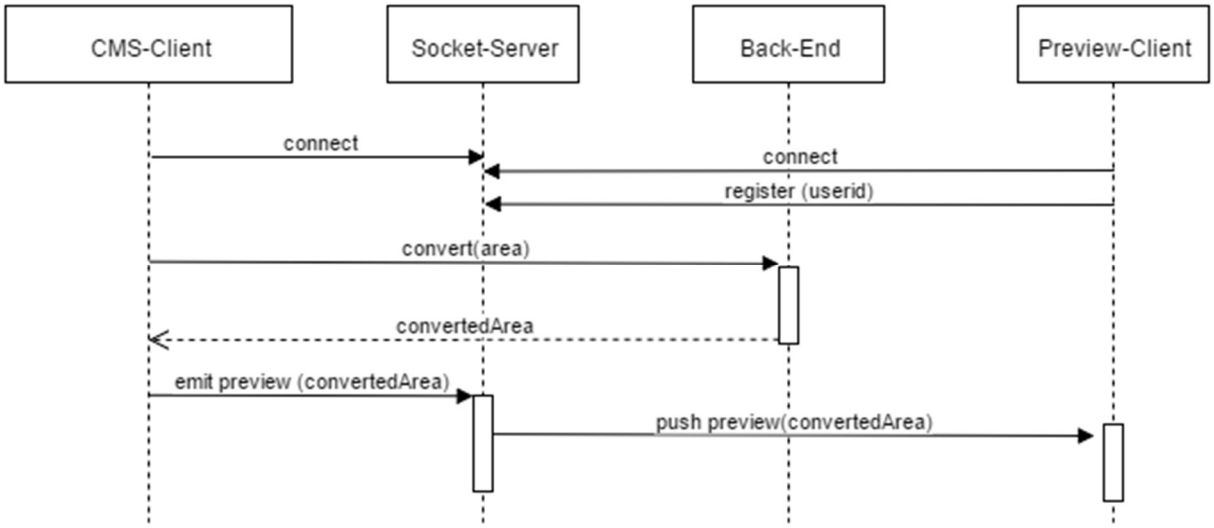



*Abbildung 6-37 Sequenzdiagramm Socket*

#### 6.4.7.7 Statistiken sammeln und anzeigen

Die Statistiken der App werden mit Google Analytics gesammelt. Mit dem Login maaas.dev@gmail.com wird pro Museum ein sogenanntes Property mit Tracking ID im Google Analytics erstellt. Dieser Prozess ist noch nicht automatisiert und muss pro Museum manuell erstellt werden. Die Tracking-ID kann danach im Museum im CMS hinterlegt werden. Im App wird die Tracking ID des Museums geladen und sämtliche Daten darauf getrackt.

Den Benutzern des CMS muss danach Leserechte auf die Google Analytics View des eigenen Museums gegeben werden. Dies kann man machen, indem man bei Analytics in der Nutzerverwaltung der Datenansicht die Google Email Adresse hinterlegt.

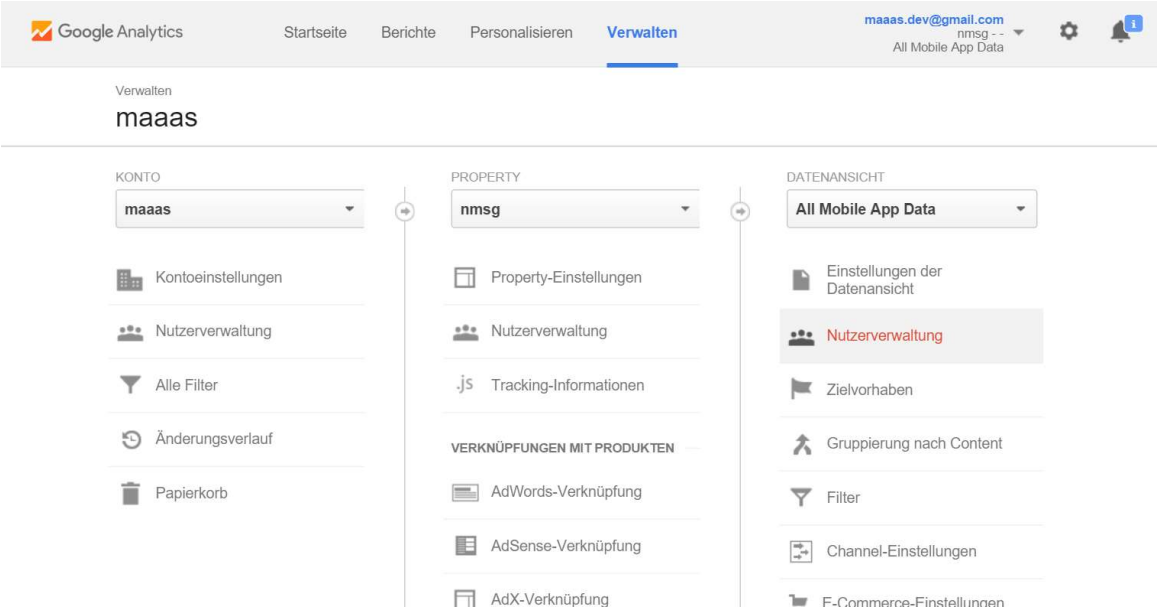



*Abbildung 38 Screenshot Google Analytics*

Die ID der Datenansicht (View) wird ebenfalls im Museum hinterlegt.

Im CMS wurde das Opensource Modul ngAnalytics von Drittanbietern verwendet. Dies erleichtert die Handhabung mit der Google Authentifizierung und liefert einfach einzusetzende Directives.

Sobald sich der Benutzer über das Authentifizierungs-Directive bei Google angemeldet hat können die Statistiken geladen werden.

Im Moment werden die Zugriffstatistiken auf Bereiche, Quiz und Inhalte getrackt und im CMS angezeigt wie viel Mal jede Ansicht geöffnet wurde und wie lange sich Benutzer darauf aufgehalten haben. Dies wird mit dem Chart Directive gemacht bei welchen ein Filter hinterlegt wird. In der App werden entsprechende Befehle abgesetzt, wenn eine Ansicht geladen wird.

```
analytics.trackView("Area:Ameise")
```

*Abbildung 39 Code Tracking einer View im App*

```
'filters':  `ga:screenName=~^Area.*`,
```

*Abbildung 40 Codeauschnitt Laden der Bildschirmaufrufe mit Filter*

### 6.4.8 Security

#### 6.4.8.1 Zugriff auf fremde Daten

Jede Entity (Area, Content, etc) wird mit einem Fremdschlüssel auf das zum User gehörende Museum abgespeichert. Bei allen CRUD-Operationen wird mittels entsprechender SQL-Abfragen und Prüfungen auf die Museum-Id sichergestellt, dass die Daten nur von autorisierten Personen bearbeitet werden.

#### 6.4.8.2 Brute-Force Attacken

Um ein mögliches Erraten der Logins vorzubeugen muss das Passwort eine Mindestlänge von 8 Zeichen aufweisen.

Ein weiterer wichtige Massnahme wäre das sperren einer IP bei übermässig vielen Fehlversuchen des Logins. Dies wurde aber aus Zeitgründen noch nicht umgesetzt.

#### 6.4.8.3 Bot-Registrationen

Das System könnte unnötig ausgelastet werden wenn es von unzähligen automatisieren Registrationen durch Bots überflutet wird. Um sich dagegen zu schützen müsste man im Registrations-Formular Captchas einbauen. Dies wurde aus Zeitgründen noch nicht umgesetzt.

#### 6.4.8.4 SQL-Injection

Ruby on Rails bietet einen sehr einfachen Mechanismus, um SQL-Injection vorzubeugen. Der folgende Code ist SQL-Injection-anfällig:

```
title = params["title"]
Model.where("title = #{title}")
```

*Abbildung 6-41 Auf SQL-Injection anfälliger Code*

Dieser Code ist anfällig, weil ein Parameter “title” entgegengenommen und dieser direkt in die SQL-Abfrage eingebunden wird.

Der folgende Code ist gegen SQL-Injection geschützt:

```
title = params["title"]
Model.where("title = ?", title)
```

*Abbildung 6-42 Code mit Schutz vor SQL-Injection*

Wie man sieht, wird der Parameter “title” nicht direkt in die SQL-Abfrage eingebunden. Statt dessen wird ein SQL-Statement angegeben, welches Platzhalter (durch “?” markiert) enthält. Diese Platzhalter werden dann der Reihe nach mit den gegeben Parametern ersetzt. Der Unterschied zur vorhergehenden Version ist, dass Rails die Parameter zuerst auf bestimmte SQL-Symbole überprüft (z.B. ‘ oder “) und zurückweist, falls sie schädlich sein könnten.

#### 6.4.8.5 Session Hijacking

Die wirkungsvollste Massnahme gegen Session Hijacking ist der Einsatz von SSL, sodass Cookies, in welchen Session-Daten übertragen werden, verschlüsselt sind. Die Verbindung zu Heroku geschieht über HTTPS.

#### 6.4.8.6 Cross-Site Scripting (XSS)

XSS ist eines der grössten um umfangreichsten Sicherheitsprobleme von Web-Applikationen. Aufgrund der Komplexität des Themas und Zeitmangels wurde kein besonderes Augenmerk darauf gelegt. Sollte das System in Produktion gehen, wäre dies eines der ersten Dinge die man genauer testen und evaluieren müsste.

#### 6.4.8.7 DOS Attacken

Da das CMS auf einem Cloudanbieter gehosted wird, muss man sich auf dessen Sicherheitsmassnahmen verlassen.

#### 6.4.8.8 CORS

Das Backend erlaubt CORS-Zugriffe der folgenden Origins:

- http://localhost:3001 (Lokale Entwicklung)
- http://localhost:4200 (Lokale Entwicklung)
- https://maaas-cms.herokuapp.com (Produktiv)

Dadurch ist gewährleistet, dass das Backend nur Requests verarbeitet welche von diesen Domains stammen.

### 6.4.9 Komponenten von Drittanbietern

#### 6.4.9.1 Styling

***Bootstrap***

Sehr verbreitetes Styling Framework von Twitter mit nützlichen CSS-Klassen. Mit Bootstrap ist es einfacher möglich, ein grid-orientiertes, responsives Design zu erstellen welches von allen gängigen Browser unterstützt wird. Alternativen wurden nicht evaluiert da beide Team-Mitglieder bereits Erfahrung mit Bootstrap haben.

Link: http://getbootstrap.com/

Lizenz: MIT

***Sb Amin 2***

Ein kostenloses Template basierend auf Bootstrap. Wurde leicht angepasst und legt das grundlegend Styling des CMS fest. Das Template braucht die JS Library MetisMenu für das Navigationsmenu.

Link: http://startbootstrap.com/template-overviews/sb-admin-2/

Link: https://github.com/onokumus/metisMenu

Lizenz: Apache

#### 6.4.9.2 JavaScript Libraries

***Cloudinary***

Cloudinary stellt ein JQuery Plugin zur Verfügung für einen unkomplizierten Fileupload. Es wurden Komponenten daraus verwendet um ein eigenes Directive zu schreiben.

Link: http://cloudinary.com/documentation/jquery_integration

Lizenz: MIT

#### *tinymce*

Ein webbasierter JavaScript WYSIWYG („What you see is what you get“) Editor. Wurde integriert um die Inhalte der Bereiche möglich benutzerfreundlich zu erfassen.

Link: https://www.tinymce.com/

Lizenz: GNU

Evaluierte Alternativen:

| Name | Nicht eingesetzt weil … |
|---|---|
| **CKEditor** | weniger flexibel |
| **Quill** | relativ neu, wenig Dokumentation |

### 6.4.9.3 AngularJS Module

#### *angular-translate*

Ein Modul welches die Internationalisierung einer Angular-App vereinfacht. Mithilfe von Angular-Translate wurde das CMS mehrsprachig gemacht. Im Moment stehen aber nur die deutschen Texte zur Verfügung. Alternativen wurden nicht evaluiert und AngularJS bietet in diesem Bereich noch zu wenig Funktionalität.

Link: https://github.com/angular-translate/angular-translate

Lizenz: MIT

#### *Ui-Router*

Ein Routing-Framework, welches es erlaubt, die Zustände der App in States zu verwalten. Der UI Router hat im Gegensatz zum Standardmodul ngRoute mehr Möglichkeiten, um mit verschachtelten View zu arbeiten.

Link: https://github.com/angular-ui/ui-router

Lizenz: MIT

#### *JS Data Angular*

Ein Framework welches den Umgang mit dem REST Backend vereinfacht. Es stellt Angular-Services zur Verfügung um Operationen auf den Daten auszuführen und setzt sie in Relationen.

Link: http://www.js-data.io/docs/js-data-angular

Lizenz: MIT

#### *angularjs-color-picker*

Beinhaltet ein einfach zu verwendendes Angular-Directive für die Farbauswahl. Wird für das Bereiche-Formular verwendet. Alternativen wurden nicht evaluiert.

Link: https://github.com/ruhley/angular-color-picker

Lizenz: MIT

#### *Angular Breadcrumb*

Generiert Breadcrumbs für alle Seiten. Wurde verwendet da dieses Modul auf den Zuständen des verwendeten UIRouters basiert.

Link: https://github.com/ncuillery/angular-breadcrumb

Lizenz: MIT

#### *Ui-Bootrstrap*

Beinhaltet UI Komponenten welche auf die Bootstrap Style-Lib basieren. Wurde verwendet für Alert-Boxen und Popvers.

Link: https://angular-ui.github.io/bootstrap/

Lizenz: MIT

#### *Angular Devise*

Angular-Service um mit dem Devise-API des Backends zu kommunizieren.

Link: https://github.com/cloudspace/angular_devise

Lizenz: MIT

#### *ngAnalytics*

Wird benötigt um die App Statistiken anzuzeigen. Alternative hätte man das Google Analytics API direkt verwenden können. Mit dem Modul lassen sich aber vor allem die Google Authentifizierung und Statistiken leichter integrieren.

Link: https://github.com/flyacts/ngAnalytics

Lizenz: MIT

### 6.4.9.4 Ruby-Gems

#### *Devise*

Bibliothek für Authentisierung. Wird verwendet für die Registration, das Login und das Session-Management.

Link: https://github.com/plataformatec/devise

Lizenz: MIT

### *Rack-Cors*

Ermöglicht Cross-Origin-Aufrufe per AJAX. Wird als Rack-Middleware konfiguriert. Es können die Origins angegeben werden für welche CORS erlaubt ist. Im Falle von maaas sind dies diverse localhost-URLs für die lokale Entwicklung sowie die URL zu Heroku für die Produktion.

Link: https://github.com/cyu/rack-cors

Lizenz: MIT

### *Globalize*

Abstraktion für Übersetzungen von Entities. Generiert die Übersetzungstabellen und sorgt implizit für die nötigen Join-Statements.

Links: https://github.com/globalize/globalize

Lizenz: MIT

### *Cancancan*

Bibliothek für Autorisierung. Damit werden die Resourcen vor unautorisierten Zugriffen geschützt, welche über das REST-API zugänglich sind.

Link: https://github.com/CanCanCommunity/cancancan

Lizenz: MIT

### *Nokogiri*

Unterstützt XML-Parsing. Wird eingesetzt um die Quiz-Daten zu parsen.

Link: https://github.com/sparklemotion/nokogiri

Lizenz: MIT

### 6.4.10 Code-Review

Der Code wurde von Michael Gfeller überprüft. Wir erhielten folgende Inputs:

| Input | Folgeschluss |
|---|---|
| Readme nicht aktuell | Readme wurde aktualisiert |
| Keine Unit-Tests | Wurde im Bericht begründet Kapitel 4.6.4.3 |
| Keine Source-Dokumentation | Wir verfolgen den Ansatz, Code so wenig wie möglich zu kommentieren und statt dessen aussagekräftiges Naming und Testing einzusetzen |
| Frage: Sind die Ressourcen Server-Seitig geschützt? | Antwort: Ja |
| Code könnte teilweise in Services extrahiert werden, um Controllers zu entlasten | Kann in einem zukünftigen Refactoring angepasst werden |
| Alle @Injects sollten einen Type besitzen | Kann in einem zukünftigen Refactoring angepasst werden. Bei einer Migration auf Angular2 müssen ohnehin alle Injects entsprechend angepasst werden |
| Gibt es Code-Konventionen? | Ja, siehe Kapitel 6.4.4 |

### 6.4.11 Installationsguides

#### 6.4.11.1 Frontend

Es muss zunächst Node und npm installiert sein. Danach müssen in Ordner des Projektes folgende Befehle in der Kommandozeile mit Administratorrechten ausgeführt werden

```
> npm install -g bower tsd gulp
> npm install
> bower install
> tsd install
```

Dies installiert alle nötigen npm-Packages und andere Abhängigkeiten. tsd beinhaltet die Typen-Definitionen der verwendeten Frameworks welche für Type-Script benötigt werden.

Um den Source-Code zu kompilieren und einen Lokalen Web-Server zu starten muss der Default-Task von gulp aufgerufen werden. Dies geschieht mit folgendem Befehl

```
> gulp --production
```

Ohne das Production flag würde das CMS das Backend lokal adressieren. Dazu müsste man das Backend local hosten. In diesem Falle geht das CMS direkt auf das Heroku Backend.

Der Gulp-Task startet zudem auch sogenannte Watches welche auf Fileänderungen reagieren und das Projekt neu kompilieren.

Benutzer Name: test@gmail.com
Passwort: test@gmail.com

#### 6.4.11.2 Backend

Systemvoraussetzungen:

- Ruby 2.0 oder höher
- Postgres 9.4 oder höher

Im Projektordner folgendes ausführen:

```
> bundle install
> rake db:setup
```

Danach kann der Server gestartet werden:

```
> rails s
```

## 6.5 Benutzertest und Testlogs

Um die Richtigkeit und Benutzerfreundlichkeit des *maaas* Systems zu testen wurde es in verschiedenen Phasen getestet.

### 6.5.1 Papierprototyp

In einem ersten Schritt wurde ein Papierprototyp gefertigt. Prof. Stolze hat sich bereit erklärt als erste Testperson zur Verfügung zu stehen.

Es hat sich ergeben, dass das System auf einen guten Weg ist. Folgende Erkenntnisse wurden aufgrund des Papierprototypens gewonnen:

- Das Angedachte Navigationskonzept ist verständlich
- Die Buttons müssen in den Masken einheitlich beschriftet sein. z.B. „Speichern und Zurück“
- Das System wird ohne jegliche Schulung schwer zu verstehen sein. Ein Museum wird wohl nicht um eine kurze Schulung kommen. Jedoch wird versucht mehr Hilfetexte in das CMS einzubauen.
- Die Maximallänge von Feldern muss beachtet werden. Dieser Punkt wurde durch die Preview Funktionalität aber relativiert. Der CMS Anwender sieht immer direkt wenn ein Text zu lang ist.

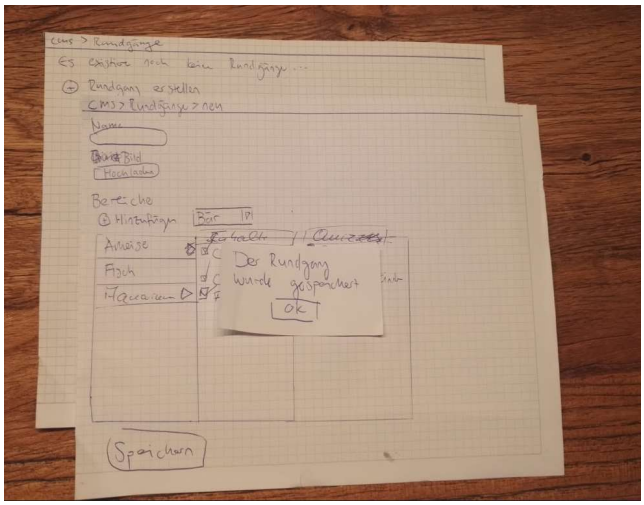



*Abbildung 6-43 Foto Papierprototyp*

### 6.5.2 Zwischenstand Test

Am 19. November wurde in einem zweiten Schritt der aktuelle Zwischenstand des CMS durch die Museumspädagogin Regula Frei vom Naturmuseum St. Gallen, getestet. Das Ziel davon war, herauszufinden ob die angedachten Konzepte benutzerfreundlich umgesetzt wurden und ob Personen, die nicht vom Fach sind, sich ebenfalls zurechtfinden.

Schriftlich wurde eine Ausgangslage und Aufgabenstellung erfasst. Regula Frei musste danach mit möglichst wenig Hilfe der Entwickler die Aufgabe lösen. Der Test verlief positiv und die Aufgaben konnten alle gelöst werden. Aus dem Testing wurden zudem folgende Erkenntnisse gewonnen welche bei der Entwicklung noch nicht aufgefallen sind:

- Es müssen Meldungen angezeigt werden, wenn ein Datensatz erfolgreich erstellt wurde
- Wenn man ein Datensatz erstellt und zurück klickt muss dieser sofort in die Übersichtsliste nachgeladen werden
- Einheitliche Benennung der Bereiche
- Jegliche Löschaktionen brauchen eine Bestätigung des Benutzers

### 6.5.3 Abschluss Test

Regula Frei hat am 14. Dezember das abgeschlossene CMS getestet. Dazu wurden schriftlich Aufgaben erfasst welche gelöst werden mussten. Der Fokus lag dabei auf den neuen Funktionalitäten wie Vorschau und Übersetzen.

Aufgaben:

1. Live-Vorschau der App öffnen und einen Bereich anzeigen lassen
2. Titelfarbe bei der Ameise auf Rot ändern und speichern
3. Den Bereich der Fische löschen
4. Wahr/Falsch Quiz für den Bären erfassen
5. Den Inhalt “Greifvogelmerkmale”
    a. Mit einem Absatz ergänzen
    b. Bild hinzufügen
    c. speichern
6. Einen neuen Rundgang für Schulklassen zusammenstellen
7. Einen Bereich in alle Sprachen übersetzen
8. Die App Live Vorschau auf einem Smartphone öffnen

Die Aufgaben konnten alle gelöst werden. Folgende Punkte sind noch aufgefallen:

- Die Vorschau soll sich schon beim Navigieren aktualisieren und nicht erst bei Änderungen
- Wenn man im Breadcrumb auf das erste Element „CMS“ klickt, navigiert man zu einer weissen Seite
- Inhalt Erfassen: Für das Einfügen per Drag&Drop der Bilder braucht es einen Hilfetext
- Benachrichtigungs-Boxen sollten nach einer kurzen Zeit selber verschwinden
- Noch nicht alle Texte sind lokalisiert (Area Detail)

Regula Frei hat zudem bestätigt, dass das CMS übersichtlich und benutzerfreundlich ist. Vor allem die Vorschau wurde gelobt.

### 6.5.4 Test-Logs

Aus dem Abschluss-Testing und den eigenen Versuchen konnten folgende Logs erstellt werden.

#### 6.5.4.1 Use-Case

| Usecase | Erreicht | Bemerkung |
|---|---|---|
| UC1.1: Registration und Login | Ja | Benutzer können sich selbständig Registrieren und Einloggen. |
| UC1.2: Museum erstellen | Ja | Registrierte Benutzer können ein neues Museum erstellen zu einem bestehenden eingeladen werden. |
| UC1.3: Benutzerverwaltung und Rollensystem | Teils | Auf ein Rollensystem wurde aus Zeitgründen vorerst verzichtet. |
| UC1.4: Bereiche erstellen und verwalten | Ja | Benutzer können Bereiche erfassen, editieren, löschen und eine Vorschau anzeigen lassen. |
| UC1.5: Inhalte erstellen und verwalten | Ja | Benutzer können Inhalte erstellen, editieren löschen und eine Vorschau anzeigen lassen. |
| UC1.6: Quiz erstellen und verwalten | Ja | Benutzer können Quiz erstellen, editieren, löschen und eine Vorschau anzeigen lassen. |
| UC1.7: Beacons verwalten | Ja | Benutzer können die Beacons mit Kontakt.io abgleichen und mit einem Bereich verknüpfen |
| UC1.8: Rundgänge erstellen und verwalten | Ja | Benutzer können Rundgänge zusammenstellen |
| UC1.9: Statistiken ansehen | Ja | Mit Google Analytics werden Statistiken gesammelt welche im CMS angezeigt werden können. |
| UC1.10: Übersetzen | Ja | Die Benutzer können die Bereiche, Inhalte und Quiz in verschiedene Sprachen übersetzen. |

#### 6.5.4.2 **NFR**

| NFR | Erreicht | Bemerkung |
|---|---|---|
| Richtigkeit | Ja | Das CMS ist responsive |
| Reife | Ja | Das CMS wurde von den Entwicklern sowie externen Personen manuell getestet |
| Fehlertoleranz | Ja | Die Formulare haben eine Validierung |
| Erlernbarkeit | Ja | Die Benutzertests haben ergeben, dass nur wenig zusätzliche Hilfe für das Bedienen des CMS notwendig ist. |
| Verständlichkeit | Ja | Die Eingabemasken haben Hilfetexte und die Navigation ist nicht tief hierarchisch und jede Seite besitzt ein Breadcrumb für die Navigation |
| Attraktivität | Ja | Durch das Verwenden von Bootstrap und Einhalten von schlichtem Design hat das CMS eine moderne Optik |
| Zeitverhalten | Ja | Lokal ausgeführt unterschreiten alle Aktionen die Reaktionszeit von 3 Sekunden. Auf der Cloud könnte man mehr Ressourcen kaufen um selbiges Resultat zu erreichen. |
| Konformität | Ja | Es wurden bewährte, weit verbreitete Tools (Sprachen, Libraries, Frameworks) eingesetzt. Die Codierung geschieht anhand der Coding Guidelines jener Tools. |
| Analysierbarkeit | Ja | Die Prinzipien der hohen Kohäsion und geringen Koppelung sollen eingehalten werden.<br><br>Statische Code-Analyse (Linting) für TypeScript und Jade wurde eingerichtet. |
| Prüfbarkeit | Nein | Tests wurden wie beschrieben noch nicht umgesetzt. Die Architektur lässt das Nachliefern aber zu. |
| Skalierbarkeit | Ja | Der gewählte Cloudanbieter Heroku lässt sich nach Belieben skalieren |

| | | |
|---|---|---|
| Lokalisierung | Ja | Sämtliche Texte wurden in ein Sprach-File ausgelagert, welches man übersetzen könnte. |

## 6.6 Risikomanagement

### 6.6.1 Risiken

#### *R1 Arbeitsumgebung defekt*

Arbeitsumgebung wird unbrauchbar, z.B. kaputter Laptop oder fehlerhafte Software

| | |
|---|---|
| **Vorbeugung:** | Diverse Workstations vorbereiten |
| **Verhalten beim Eintreten:** | Weiterarbeiten auf anderen Workstations oder Neuinstallation |

#### *R2 Mangelndes Interesse des Museums*

Das Naturmuseum hat wenig Interesse am CMS oder kann keine Zeit dafür aufwenden, was zu einer schlechten Kooperation führt

| | |
|---|---|
| **Vorbeugung:** | Dem Museum die Möglichkeit geben, dort zu helfen und mitzubestimmen, wo sie wollen. Jedoch auch selbständig sein wo es möglich ist. |
| **Verhalten beim Eintreten:** | Andere Tester anfragen |

#### *R3 Technologie ungenügend*

Die ausgewählten Technologien und Frameworks können die Anforderungen nicht erfüllen

| | |
|---|---|
| **Vorbeugung:** | Möglichst schnelle Entwicklung des vertikalen Durchstichs |
| **Verhalten beim Eintreten:** | Workarounds anwenden, andere Technologien einsetzen |

#### *R4 Browserinkompatibilität*

Das CMS läuft nur in bestimmten Browsern

| | |
|---|---|
| **Vorbeugung:** | Kontinuierliche manuelle Tests in verschiedenen Browsern |
| **Verhalten beim Eintreten:** | Browser-spezifischer Code, oder Browser explizit nicht unterstützen |

### *R5 Schlechte Architektur*

Code wird schlecht aufgebaut was Wartbarkeit und Erweiterbarkeit erschwert

| | |
|---|---|
| **Vorbeugung:** | Gängige Entwurfsmuster verwenden. Genügend Zeit nehmen |
| **Verhalten beim Eintreten:** | Bei genügend Zeit: Redesign |

### *R6 Unintuitiv*

Das CMS ist für Benutzer nicht verständlich

| | |
|---|---|
| **Vorbeugung:** | App von echten Benutzern testen lassen |
| **Verhalten beim Eintreten:** | Auf Feedback der Benutzertests eingehen |

### *R7 Schlechte Testabdeckung*

Automatisierte Unit- und E2E-Tests sind nicht umfangreich genug um ein schnelles Vorankommen zu ermöglichen

| | |
|---|---|
| **Vorbeugung:** | Bei Abschluss eines Use Cases Tests mitliefern |
| **Verhalten beim Eintreten:** | Bei genügend Zeit Tests nachliefern |

### *R8 Unzureichende Sicherheit*

Die Web-App ist nicht genügend vor Hacking-Attacken geschützt

| | |
|---|---|
| **Vorbeugung:** | Libraries einsetzen, die Security-Aspekte unterstützen |
| **Verhalten beim Eintreten:** | Sicherheitslücken schliessen |

### *R9 Ändernde Cloudangebote*

Cloudanbieter ändern Ihr Angebot zu unseren Ungunsten oder stellen Service ein

| | |
|---|---|
| **Vorbeugung:** | Bewährte Cloudanbieter einsetzen |
| **Verhalten beim Eintreten:** | Andere Cloudanbieter verwenden |

### *R10 Veröffentlichung im Appstore macht Probleme*

Veröffentlichung der App auf den Appstores der verschiedenen Plattformen macht Probleme

| | |
|---|---|
| **Vorbeugung:** | Möglichst früher Release der App in den Stores |
| **Verhalten beim Eintreten:** | Bei genügend Zeit redesign |

### *R11 App Performance Probleme*

Das App läuft nicht auf allen Geräten flüssig oder mit zu hohem Akkuverbrauch

| | |
|---|---|
| **Vorbeugung:** | Altes Android Phone organisieren und von Anfang an darauf testen |
| **Verhalten beim Eintreten:** | App Software optimieren |

### 6.6.2 Risikoeinschätzung und Verlauf

| Risiko | Maximaler Schaden [h] | EW (Projektbeginn) | EW 14.10.15 | EW 27.10.15 | EW 12.11.15 | EW ( | Gewichteter Schaden [h] Projektbeginn | Gewichteter Schaden Projektende [h] |
|---|---|---|---|---|---|---|---|---|
| **R1** | 3 | 40% | 30% | 30% | 20% | 0% | 1.2 | 0 |
| **R2** | 10 | 20% | 20% | 20% | 20% | 0% | 2 | 0 |
| **R3** | 50 | 30% | 20% | 20% | 10% | 0% | 15 | 0 |
| **R4** | 10 | 30% | 30% | 30% | 20% | 0% | 3 | 0 |
| **R5** | 40 | 40% | 40% | 40% | 10% | 0% | 16 | 0 |
| **R6** | 50 | 50% | 50% | 40% | 40% | 0% | 25 | 0 |
| **R7** | 25 | 40% | 40% | 40% | 60% | 60% | 10 | 0 |
| **R8** | 50 | 30% | 30% | 30% | 30% | 15% | 15 | 15 |
| **R9** | 30 | 20% | 20% | 20% | 10% | 0% | 6 | 0 |
| **R10** | 20 | 40% | 40% | 40% | 40% | 10% | 8 | 2 |
| **R11** | 20 | 20% | 20% | 20% | 20% | 20% | 4 | 4 |

EW= Eintrittswahrscheinlichkeit

### 6.6.3 Analyse

#### 6.6.3.1 Unzureichende Sicherheit

Ruby on Rails bietet viele gute Ansätze, um Sicherheitslücken zu vermeiden. Das Back-End wurde nach den Ruby on Rails-Standards entwickelt. Allerdings kann man ohne genaue Tests nie ganz sicher sein, ob genügende Sicherheit gewährleistet ist.

#### 6.6.3.2 Veröffentlichung im App-Store macht Probleme

Da Apple sehr strikte Richtlinien betreffend App-Veröffentlichung hat und ein Deployment einer App jeweils mindestens 7 Tage dauert, muss die Veröffentlichung im Apple-Store stets gut geplant sein. Die App kann wegen kleinen Fehlern z.B. in den Metadaten rejected werden.

#### 6.6.3.3 Performance Probleme

Aufgrund knapper Zeit konnten die Performance-Probleme in der App auf manchen Geräten nicht gelöst werden.

#### 6.6.3.4 Schlechte Testabdeckung

Weil das Projekt von Grund auf entwickelt wurde und die Zeit knapp war, wurden keine automatisierten Unit-Tests geschrieben. Die Grundvoraussetzungen (z.B. der Testrunner) wurden aber eingerichtet. Bei einer Weiterführung des Projektes sollten den Tests eine höhere Priorität beigemessen werden.

## 6.7 Business Evaluation

Ziel der Business Evaluation war es, herauszufinden ob sich *maaas* in der Markwirtschaft durchsetzen könnte. Dafür wurde nationale Konkurrenz analysiert und eine Umfrage durchgeführt.

### 6.7.1 Konkurrenz

#### 6.7.1.1 Xponia

Xponia ist eine AG aus St. Gallen. Die Informationen des Internetauftritts lassen keine Schlüsse zu, wie weit der Entwicklungstand ist oder ob bereits irgendwo eingesetzt wird. Da das Unternehmen erst Ende 2014 gegründet wurde kann man annehmen, dass sie im Markt noch nicht etabliert sind. Die Versprochenen Features entsprechen ungefähr den unseren. Zusätzlich wird eine Navigationskarte angeboten.

### 6.7.2 Umfrage

Um eine Idee zu gewinnen, ob auch andere Museen ein solches System begrüssen, wurde eine Umfrage durchgeführt. Dazu hat man verschiedene Museen angeschrieben. Es wurde ein Fragebogen erstellt, um herauszufinden ob die Museen Interesse an moderner Technologie haben und eventuell bereits ein App im Einsatz ist. Konkret wurde auch nachgefragt den Einsatz welcher Features von *maaas* sie sich vorstellen könnten.

#### 6.7.2.1 Empfänger

Es wurden insgesamt 35 Museen aus der Deutschschweiz angeschrieben wobei auf die vielfallt geachtet wurde. So wurden kleinere Museen (25 Besucher pro Woche) aber auch grössere Museen (2000 Besucher pro Woche) angeschrieben.

#### 6.7.2.2 Fragebogen

Der Fragebogen wurde als Google Form an die Museen versendet. Folgende Fragen waren darin enthalten:

**Name des Museums**

_____

**Wie viele Besucher haben sie durchschnittlich pro Woche?**

_____

**Wie stellen Sie Informationen zur Verfügung?**

- ☐ Informationstafeln
- ☐ Audio-Guide
- ☐ Bildschirme
- ☐ Interaktive Bildschirme
- ☐ Mobile-App
- ☐ IPad/Tablets
- ☐ Andere: _____

**Sind sie interessiert an moderner Technik fürs Museum?**

[Ja/Nein]

**Falls Sie ein App für den Museumsbesuch haben, wer ist der Anbieter?**

_____

**Falls Sie kein App haben, gibt es dafür bestimmte Gründe?**

_____

**Wären Sie bereit in ein Museums-App zu investieren und wenn ja, welche der folgenden Funktionalitäten halten Sie für sinnvoll?** (Unser System bietet all diese Funktionalitäten an)

- ☐ Multimediale Inhalte anzeigen
- ☐ Ortsbezogene Inhalte (anhand von Indoor Ortung)
- ☐ Quiz lösen
- ☐ Mehrsprachige Inhalte
- ☐ Statistiken sammeln (Wer hält sich wo auf, welche Ausstellungsstücke sind am beliebtesten, etc.)
- ☐ Auf Zielgruppen angepasste Inhalte
- ☐ Inhalte selbständig verwalten

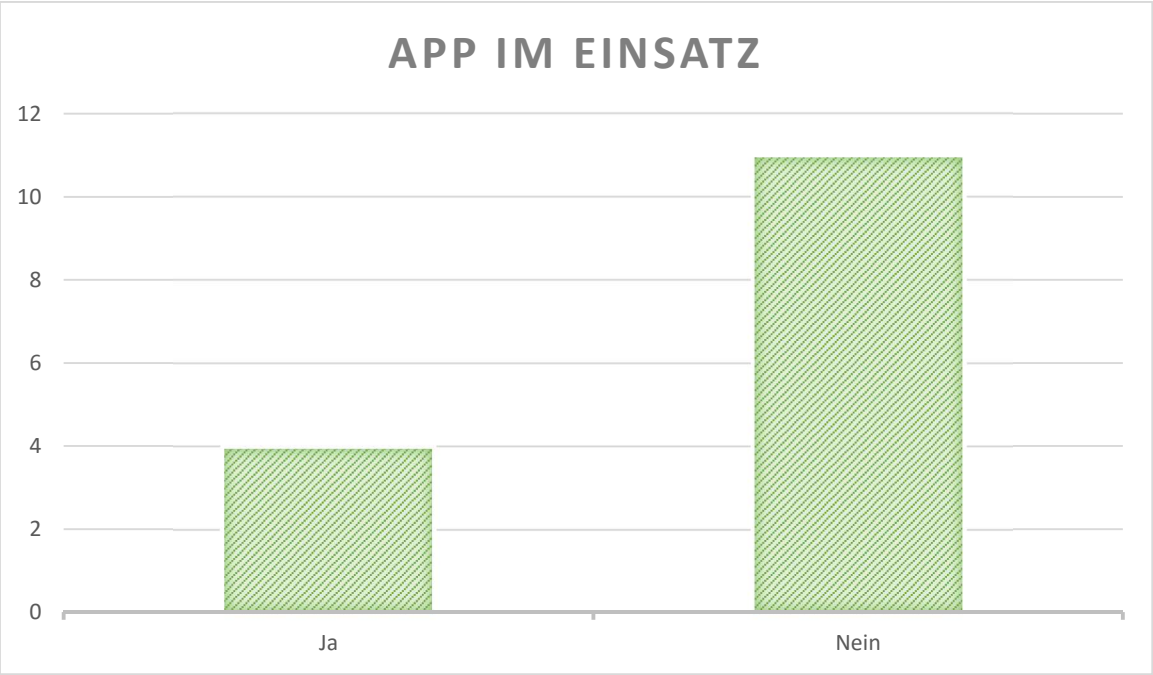
**Bemerkungen**

_____

_____

#### 6.7.2.3 Auswertung

Total haben 15 Museen die Umfrage ausgefüllt. Folgende Auswertung bezieht sich auf deren Feedback. 4 Museen haben sich komplett gegen die Einführung einer App geäussert. Die Ausstellung sei dafür zu klein oder eine App wird als unpersönlich empfunden. Dies sind Museen welche eher auf Führungen setzen und nicht auf Individualbesucher.




Vor allem die grösseren Museen haben ein App im Einsatz. Es handelt sich dabei um Eigenproduktionen und nicht um einen Anbieter. Der Kanton Bern und Kanton Aargau hat zudem ein App, in dem man sich einen Überblick über die Museen verschaffen kann. Dies sind aber keine interaktiven Apps um eine Ausstellung zu ergänzen. [25]

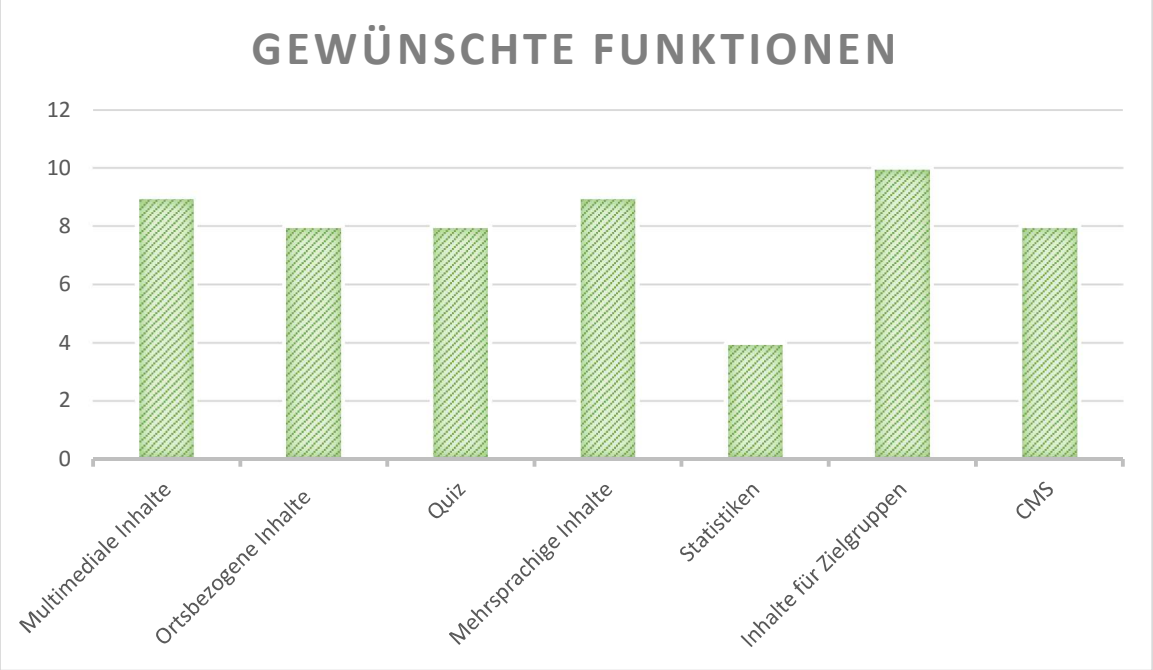



*Abbildung 6-44 Pilottag Auswertung 1*

Man kann herauslesen, dass wenn ein App gewünscht ist, alle Funktionalitäten in etwa gleich präferiert wurden. Nur die Statistiken werden nicht als ganz so wichtig empfunden.

### 6.7.3 Vogelparcours

Nach der Vermittlung durch Professor Heinzmann wurde das System dem Macher der Seite http://www.wasservoegel.ch vorgestellt. Diese Seite enthält unter anderem einen grossflächigen Vogelparcours, der sich anbieten würde, mit der App umgesetzt zu werden.

An einem kurzen Meeting wurden die Möglichkeiten besprochen und es stellte sich heraus, dass der Einsatz der App und des CMS durchaus interessant sein könnte. Die Bereiche wären Outdoor und die App-Benutzer könnten nachlesen, welche Vögel am aktuellen Standort auffindbar sind und zusätzliche Informationen zu den Vögeln abrufen. Das Sammeln aller Sticker könnte zudem mit einem Kaffee in der HSR belohnt werden. Als Beacons könnten die *"Tough"*-Beacons von kontakt.io eingesetzt werden, welche speziell für Outdoor-Anwendungen gemacht wurden. Allerdings kann die Umsetzung des Vogelparcours in der App aus terminlichen Gründen nicht Gegenstand dieser Arbeit sein. Weitere Schritte werden deshalb unabhängig von dieser Arbeit unternommen.

### 6.7.4 Fazit

Die Umfrage hat ergeben, dass es Museen gibt welche sich aktuell Gedanken um die Einführung einer App machen und noch keine Lösung haben. Dies sind vor allem mittelgrosse Museen. Da diese um die 400 Besucher in der Woche haben ist auch das Budget nicht ganz so gross und deshalb die Kostenfrage zentral. Mit einem preiswerten Modell, welches für das Museum wenig Aufwand bis zur Einführung bedeutet, könnte man diese vielleicht überzeugen.

Dass es eine lokale Konkurrenz gibt, welche ähnliche Gedanken verfolgt, bestätigt auch, dass es einen Markt gibt.