

11.06.2026 - Bachelorarbeit Frühjahrssemester 2026

Event-Suchroboter-Cockpit (ESC)

Bachelorarbeit FS 2026

Studierende:

Nicola Aebi
nicola.aebi@ost.ch

Martyn Foreman
martyn.foreman@ost.ch

Betreuer:

Prof. Stefan Keller
stefan.keller@ost.ch

Joël Schwab
joel.schwab@ost.ch

Industriepartner:

Patrick Steinmann
Kantonspolizei
St.Gallen
patrick.steinmann@kapo.sg.ch

Abstract

Die Kantonspolizei St.Gallen benötigt zur vorausschauenden Einsatzplanung ein möglichst vollständiges Bild aller sicherheitsrelevanten Veranstaltungen im Kanton. Bisher werden Gemeinde- und Vereinswebseiten, kantonale Eventportale sowie strukturierte Schnittstellen manuell gesichtet. Dieser Prozess ist zeitintensiv, fehleranfällig und skaliert nicht mit der Vielfalt und Heterogenität der Quellen.

Die vorliegende Arbeit überführt einen bestehenden Prototyp in einen erweiterten Proof-of-Concept eines Event-Suchroboter-Cockpits. Dabei verfolgt sie die Leitfrage: *Lassen sich im Web veröffentlichte Veranstaltungen im Kanton St.Gallen automatisiert so erfassen, kategorisieren und visualisieren, dass zeitlich-räumliche Koinzidenzen erkennbar werden?*

Kern der Arbeit ist eine konfigurationsgetriebene, asynchrone Pipeline. Ein generischer Crawler erschliesst HTML- und PDF-Quellen anhand deklarativer Quellkonfigurationen. Anschliessend extrahiert ein Large Language Model daraus strukturierte Event-Objekte. Eine deterministische Geo-Komponente validiert und ergänzt die Ortsangaben gegen einen kuratierten Schweizer Adressdatensatz, bevor eine vektorbasierte Kategorisierung mittels pgvector die Ereignisse klassifiziert. Abschliessend identifiziert eine Deduplizierungs- und Merge-Komponente potenzielle Mehrfacherfassungen desselben Events über verschiedene Quellen hinweg. Dabei werden Duplikate konsolidiert, ohne legitime Aktualisierungen oder unterschiedliche Informationsstände eines Events fälschlicherweise zu verwerfen.

Die persistierten Daten werden über einen FastAPI-Dienst an ein SvelteKit-Cockpit ausgeliefert, das die Veranstaltungen auf einer interaktiven Karte mit Filter-, Such- und Exportfunktionen zugänglich macht.

Die Lösung wurde mit über 80 produktiven Quellen aus dem Kanton St.Gallen validiert. Darunter befinden sich die schweizweiten Eventportale Guidle und Eventfrog, regionale Portale, die Webseiten sämtlicher Gemeinden des Kantons sowie ausgewählte Social-Media- und Forumsquellen. Die Pipeline verarbeitet sowohl klassische HTML-Inhalte als auch PDF-Dokumente und zeigt, dass neue Datenquellen durch deklarative Konfigurationen ohne Codeanpassungen integriert werden können. Darüber hinaus gewährleisteten Stage-Artefakte und Trace-IDs eine durchgängige Nachvollziehbarkeit aller Verarbeitungsschritte.

Als zentrale Herausforderungen zeigen sich die LLM-bedingte Verarbeitungszeit sowie die stochastische Varianz der semantischen Extraktion. Diese werden durch Guardrails, mehrstufige Validierungsmechanismen und deterministische Verarbeitungsschritte begrenzt.

Die Ergebnisse belegen die technische Machbarkeit eines automatisierten Eventaggregators für die Kantonspolizei St.Gallen. Gleichzeitig ergeben sich weiterführende Perspektiven hinsichtlich eines produktiven Lage- und Informationscockpits, der Nutzung durch weitere Bedarfsträger wie Gemeinden oder Rettungsdienste sowie der möglichen Erweiterung um Verfahren zur automatisierten Risikoeinstufung und Gefährdungsbeurteilung von Veranstaltungen.

Management Summary

Ausgangslage

Das Lage- und Nachrichtenzentrum der Kantonspolizei St.Gallen muss jederzeit ein aktuelles Bild der Sicherheitslage im Kanton führen. Ein wichtiger Bestandteil dieser Arbeit ist die frühzeitige Erkennung im Web veröffentlichter Veranstaltungen wie Konzerten, Sportanlässen oder Demonstrationen, die einen erhöhten Personaleinsatz oder koordinierte polizeiliche Massnahmen erfordern.

Heute geschieht diese Erfassung weitgehend von Hand: Mitarbeitende durchsuchen Gemeinde- und Vereinswebseiten, kantonale Eventportale, Social-Media-Plattformen und strukturierte Schnittstellen. Die relevanten Informationen liegen in unterschiedlichsten Formaten vor, von HTML-Seiten über PDF-Dokumente bis hin zu API-Antworten. Der Prozess ist zeitintensiv, fehleranfällig und skaliert nicht mit der wachsenden Zahl an Quellen.

Die vorliegende Bachelorarbeit baut auf einer vorausgegangenen Studienarbeit auf und überführt den dort entstandenen Prototyp in einen erweiterten Proof-of-Concept eines **Event-Suchroboter-Cockpits** (ESC) [1]. Ziel ist eine automatisierte, durchgängige Lösung von der Datenerfassung bis zur Visualisierung. Die zentrale Leitfrage lautet: *Lassen sich im Web veröffentlichte Veranstaltungen im Kanton St.Gallen automatisiert so erfassen, kategorisieren und visualisieren, dass zeitlich-räumliche Koinzidenzen erkennbar werden?*

Vorgehen und Technologien

Das System besteht aus einer mehrstufigen Verarbeitungsplattform mit Crawler-Pipeline, Datenbank, API-Service und Frontend. Gemeinsam bilden diese Komponenten den gesamten Verarbeitungsweg eines Events von der Quelle bis zur Visualisierung ab, welche in Abbildung 1 dargestellt ist.

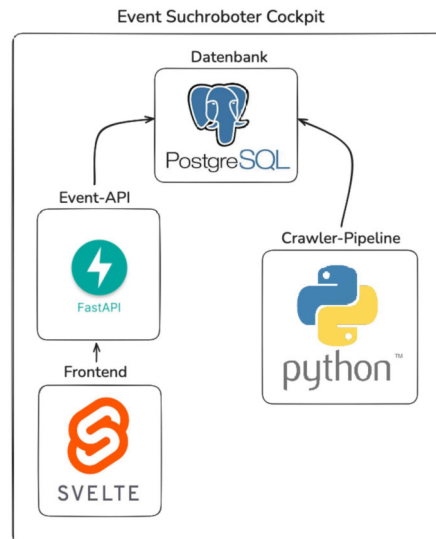


Abbildung 1: Gesamtarchitektur des Event-Suchroboter-Cockpits

- **Crawler-Pipeline:** Erschliesst die Quellwebseiten und überführt deren Inhalte in strukturierte Eventdaten.
- **Datenbank:** Speichert die konsolidierten Events relational in PostgreSQL.
- **API-Service:** Stellt die Daten dem Frontend lesend zur Verfügung.
- **Frontend:** Visualisiert die Events auf einer interaktiven Karte.

Der Schwerpunkt der Arbeit liegt auf der **Crawler-Pipeline**. Sie ist als asynchrone Verarbeitungskette aufgebaut, in der jede Stufe eine klar abgegrenzte Aufgabe übernimmt:

1. Ein **generischer Crawler** lädt die konfigurierten Webseiten und PDF-Dokumente und überführt sie in ein einheitliches Markdown-Format.
2. Ein **Large Language Model** extrahiert daraus die fachlichen Eventattribute wie Titel, Datum, Ort und Beschreibung.
3. Eine **Geo-Komponente** validiert die Ortsangaben gegen einen Schweizer Adressdatensatz und ergänzt fehlende Koordinaten.
4. Eine **vektorbasierte Kategorisierung** ordnet jedes Event automatisch einer Kategorie zu.
5. Eine **Persistenz- und Deduplizierungsschicht** erkennt identische oder ähnliche Veranstaltungen über mehrere Quellen hinweg, konsolidiert diese zu einem gemeinsamen Eventdatensatz und verarbeitet gleichzeitig legitime Aktualisierungen bestehender Events.

Eine zentrale konzeptionelle Entscheidung war der Wechsel zu einer **konfigurationsgetriebenen Architektur**: Neue Quellen werden nicht mehr durch zusätzlichen Programmcode, sondern durch eine Konfigurationsdatei angebunden. Dadurch lässt sich das System mit deutlich geringerem Aufwand auf zusätzliche Gemeinden, Vereine oder Plattformen ausweiten.

Ergebnisse

Das im Rahmen der Arbeit entwickelte Cockpit erfüllt die wichtigsten gesteckten Ziele:

- **Automatisierte Erfassung:** Über 80 produktive Quellen aus dem Kanton St.Gallen werden vollautomatisch erschlossen. Dazu zählen die schweizweiten Eventportale Guidele und Eventfrog, regionale Portale, die Webseiten sämtlicher Gemeinden des Kantons sowie ausgewählte Social-Media- und Forumsquellen. Die Lösung verarbeitet sowohl HTML-Inhalte als auch PDF-Dokumente.
- **Strukturierte Datenbasis:** Die heterogenen Quellinhalte werden zu einem einheitlichen, durchsuchbaren Datenbestand zusammengeführt und in einer relationalen Datenbank gespeichert.
- **Deduplizierung und Konsolidierung:** Veranstaltungen, die über mehrere Quellen gefunden werden, werden automatisch zusammengeführt. Gleichzeitig erkennt das System Aktualisierungen bestehender Events und verhindert, dass diese fälschlicherweise als Duplikate behandelt werden.
- **Interaktive Visualisierung:** Das Frontend stellt alle erfassten Events auf einer Karte dar und erlaubt eine schnelle Filterung nach Zeitraum, Uhrzeit, Kategorie und Quelle sowie eine Volltextsuche.

Die resultierende Oberfläche ist in Abbildung 2 dargestellt. Sie verbindet Kartenansicht, Filterbereich und Eventliste in einer kompakten Ansicht.

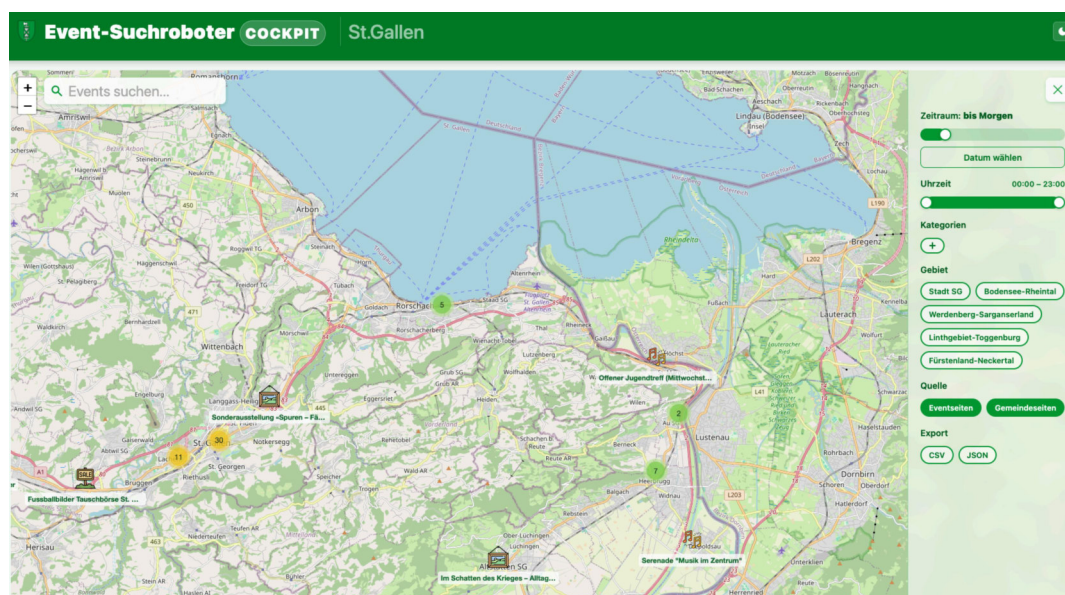


Abbildung 2: Cockpit mit Kartenansicht, Filtern und Eventliste

- **Nachvollziehbarkeit:** Jeder Verarbeitungsschritt eines Events ist über Zwischenartefakte und eindeutige Event-Kennungen lückenlos nachvollziehbar. Dies ist eine zentrale Voraussetzung für den Einsatz in einem polizeilichen Umfeld.
- **Erweiterbarkeit:** Neue Datenquellen lassen sich durch eine Konfigurationsdatei integrieren, ohne den Kern der Pipeline zu verändern.

Die Deduplizierung erfolgt bewusst nicht nur über einfache Datenbankregeln, sondern in der Anwendungsschicht. Dadurch kann das System unterscheiden, ob ein neuer Fund ein bereits bekanntes Event, ein Event aus einer zusätzlichen Quelle, eine unveränderte Wiederholung oder eine echte Aktualisierung ist. Tabelle 1 fasst diese Persistenzentscheidung zusammen.

Entscheidung	Bedeutung	Wirkung
deduped	Gleicher Event aus derselben Quelle erneut gefunden	Keine neue Zeile
merged	Gleicher Event über eine andere Quelle gefunden	Event wird nicht doppelt angezeigt
unchanged	Gleiche Quelle, unveränderte stabile Felder	Keine Änderung
updated	Gleiche Quelle, aber fachliche Änderung	Bestehender Datensatz wird aktualisiert
inserted	Kein bestehender Event passt	Neuer Event wird gespeichert

Tabelle 1: Vereinfachte Persistenzentscheidung bei der Deduplizierung.

Die Deduplizierung basiert auf einem fachlichen Fingerprint aus stabilen Eventfeldern wie Titel, Datum, Uhrzeit, Ort, Koordinaten und Status sowie auf einer kanonisierten Quell-URL. Dadurch werden exakte oder nahezu identische Mehrfachfunde zuverlässig unterdrückt. Gleichzeitig bleiben mehrere echte Termine mit gleicher Quellseite erhalten, etwa wenn eine Veranstaltungsseite mehrere Daten enthält.

Zur Bewertung der Extraktionsqualität wurde ein separater LLM-Evaluationstest mit einem manuell annotierten Gold-Datensatz durchgeführt. Die wichtigsten Kennzahlen sind in Tabelle 2 zusammengefasst.

Kennzahl	Wert
Testfälle	35
Erwartete Events	30
Korrekt erkannte Events	28
Precision der Event-Erkennung	0.800
Recall der Event-Erkennung	0.933
F1-Score der Event-Erkennung	0.862
Genauigkeit Datum und Adresse	1.000
Genauigkeit Titel	0.964
Genauigkeit Startzeit	0.926
Genauigkeit Kategorie	0.429

Tabelle 2: Zentrale Qualitätskennzahlen des LLM-Evaluationstests vom 09.06.2026.

Die Evaluation zeigt, dass das System Veranstaltungen mit hohem Recall erkennt: Von 30 erwarteten Events wurden 28 korrekt gefunden. Besonders zuverlässig sind strukturierte Felder wie Datum, Ort und Adresse. Die wichtigste bekannte Schwachstelle ist die automatische Kategorisierung, die im Test nur eine Genauigkeit von 42.9% erreicht. Dieser Wert ist auch deshalb vorsichtig zu interpretieren, weil der Gold-Datensatz mit 35 Testfällen noch begrenzt ist und die Kategorieverteilung nicht alle realen Varianten gleichmässig abdeckt. Für die Kernaufgabe der Event-Erkennung liefert der Test dennoch ein positives erstes Qualitätsbild.

Demgegenüber stehen drei wesentliche Einschränkungen, die für den späteren Produktivbetrieb relevant sind: Erstens wird die Laufzeit der Pipeline massgeblich durch das Sprachmodell

bestimmt, was bei steigender Quellenzahl zum Engpass werden kann. Zweitens erzeugen Sprachmodelle prinzipiell schwankende Ergebnisse, weshalb der Pipeline mehrere Validierungs- und Plausibilitätsstufen nachgelagert sind. Drittens wurde die Deduplizierung im Rahmen dieser Arbeit funktional und anhand konkreter Testfälle überprüft, jedoch noch nicht mit einem eigenen Gold-Datensatz quantitativ als Precision-/Recall-Metrik ausgewertet.

Ausblick

Der entstandene Proof-of-Concept bildet eine tragfähige Grundlage für den weiteren Ausbau. Neben der Anbindung zusätzlicher Datenquellen und offizieller Schnittstellen bietet sich insbesondere die Nutzung durch weitere Bedarfsträger wie Gemeinden, Tourismusorganisationen, Veranstaltende oder Gesundheits- und Rettungsdienste an.

Aus technischer Sicht stehen drei Weiterentwicklungen im Vordergrund: Erstens sollte die Deduplizierung um eine explizite Quellenverwaltung erweitert werden, damit alle Quellen eines konsolidierten Events dauerhaft am Event selbst nachvollziehbar bleiben. Zweitens wäre ein eigener Gold-Datensatz für die Deduplizierung sinnvoll, um den Anteil korrekt zusammengeführter und korrekt getrennt gehaltener Events quantitativ zu messen. Drittens sollte die automatische Kategorisierung verbessert und auf einem grösseren, ausgewogeneren Testdatensatz validiert werden.

Darüber hinaus eröffnet der konsolidierte Eventdatenbestand neue fachliche Anwendungsmöglichkeiten. So könnte das System künftig um Verfahren zur automatisierten Risikoeinstufung und Gefährdungsbeurteilung von Veranstaltungen erweitert werden, um den manuellen Prüfaufwand der Sicherheitsbehörden weiter zu reduzieren.

Eine weitere Fragestellung betrifft die Überführung des Proof-of-Concepts in ein produktives Lage- und Informationscockpit. Hierfür wären technische, organisatorische und rechtliche Aspekte im Rahmen einer weiterführenden Machbarkeitsstudie zu untersuchen.

Inhalt

Abstract	I
Management Summary	II
Teil I – Technischer Bericht	1
1 Einführung	2
1.1 Problemstellung und Vision	2
1.2 Ausgangslage	3
1.3 Industriepartner	4
1.4 Aufgabenstellung	5
1.4.1 Ziele	5
1.4.2 Herausforderungen	6
1.5 Vorarbeiten und Abgrenzung	7
1.6 Rahmenbedingung	8
1.7 Leitfrage	9
1.8 Vorgehen und Aufbau der Arbeit	10
2 Stand der Technik	11
2.1 Evaluation und Bestätigung des Technologie-Stacks	11
2.1.1 Crawler-Pipeline	11
2.1.2 API	12
2.1.3 Frontend	12
2.1.4 Datenbank	13
2.1.5 Infrastruktur und DevOps	14
2.2 Bestehende Lösungsansätze	15
2.2.1 Fragmentierung der Quellen	15
2.2.2 Eingeschränkte zeitliche Filterung	15
2.2.3 Fehlende Exportierbarkeit und Weiterverarbeitbarkeit	15
2.2.4 Fazit	15
3 Evaluation	16
3.1 Evaluationskriterien	16
3.1.1 Kriterienkatalog	16
3.1.2 Bewertungsskala	18
3.1.3 Berechnung des Nutzwerts	18
3.2 Anpassung der Crawler-Architektur	19
3.3 Evaluation der Adressauflösung und Geokodierung	20
3.4 Workflow der Event-Kategorisierung	22
3.5 Monitoring und Fehlertracking	24
3.6 Evaluation der Datenbeschaffung	26
3.7 Schlussfolgerungen	28
4 Umsetzungskonzept	30
4.1 Architektur	30

4.2 Crawler-Pipeline	32
4.2.1 Konfigurationsgetriebene Quellen	33
4.2.2 Pipeline-Stufen	33
4.2.3 Data Verzeichnis	38
4.2.4 Logging-Architektur	38
4.2.5 Abgrenzung zur API-Schicht	39
4.3 Frontend	40
4.4 API	42
4.5 Datenbankkonzept	43
4.5.1 Event-Tabelle§	43
4.5.2 Kategorie-Tabelle	44
4.5.3 Swiss-Addresses-Tabelle	44
5 Resultate	45
5.1 Ergebnisse	45
5.1.1 Offene Punkte und Limitationen	46
5.2 Zielerreichung	47
5.2.1 Harvester für Gemeindeseiten	47
5.2.2 Integration von Event-Quellen	47
5.2.3 Erschliessung von Social-Media- und Foren-Daten	47
5.2.4 Dynamisches Aktualisierungsmanagement	48
5.2.5 Datenpipeline-Orchestrierung	48
5.2.6 Automatisierte Kategorisierung	48
5.2.7 Optimierung des Frontends	49
5.3 Schlussfolgerungen und Ausblick	50
Teil II – SW-Projektdokumentation	51
6 Anforderungsspezifikation	52
6.1 Anforderungen	52
6.1.1 Personas	52
6.1.2 Use-Case-Diagramm	53
6.1.3 Funktionale Anforderungen	54
6.1.4 Non Functional Requirements (NFR)	55
7 Implementation	61
7.1 Crawler-Konfiguration	61
7.1.1 Aufbau der Konfiguration	62
7.1.2 Grundaufbau	62
7.1.3 Preparation und Pagination	62
7.1.4 Fallback-Location	63
7.1.5 Splitter	63
7.1.6 Besondere Herausforderungen	63
7.1.7 Fazit	63
7.2 URL-Crawler	64
7.2.1 Phase 1: Seitenvorbereitung	64
7.2.2 Phase 2: Link-Sammlung	64
7.2.3 Phase 3: Navigations-Aktion	64
7.2.4 Phase 4: Fortschrittserkennung	65

7.2.5 Phase 5: Abbruchkriterien	65
7.2.6 Fazit	66
7.3 Markdown-Crawler	67
7.3.1 Laden und Inhaltsauswahl	67
7.3.2 Frontmatter als Metadatenenträger	67
7.3.3 PDF-Extraktion	68
7.3.4 Fazit	68
7.4 Markdown-Splitter	69
7.4.1 Strategieauswahl	69
7.4.2 Automatische Strategien	69
7.4.3 Konfigurierte Strategien	70
7.4.4 Fazit	70
7.5 LLM-Extraktion	71
7.5.1 Markdown-Vorverarbeitung	71
7.5.2 Locked Facts	72
7.5.3 Prompt Engineering	72
7.5.4 Deterministische Guardrails	73
7.5.5 Evidence Hints und Modellwahl	74
7.5.6 Fazit	74
7.6 Geo-Anreicherung	75
7.6.1 Erweiterungen gegenüber der Semesterarbeit	75
7.6.2 Fazit	76
7.7 Event Kategorisierung	77
7.7.1 Einordnung und Abgrenzung zur Studienarbeit	77
7.7.2 Kategorienschema	78
7.7.3 Initialisierung der Kategorie-Embeddings	79
7.7.4 Zuweisung zur Laufzeit	80
7.7.5 Fallback-Kategorie	81
7.7.6 Anbindung an Persistierung	81
7.7.7 Fazit	81
7.8 Deduplizierung und Persistenz	82
7.8.1 Persistenzentscheidung	82
7.8.2 Anwendungsbeispiele	83
7.8.3 Verbleibende Herausforderungen	84
7.8.4 Fazit	85
7.9 Datenbank	86
7.9.1 Aufbau und Initialisierung	86
7.9.2 Schemaerweiterungen gegenüber der Studienarbeit	87
7.9.3 Geografische Einschränkung auf den Kanton St.Gallen	87
7.9.4 Fazit	89
7.10 API und Datenbereitstellung	90
7.10.1 Rolle und Technologie-Stack	90
7.10.2 Datenbankanbindung	90
7.10.3 Gefilterter Hauptendpunkt: GET /events/	91
7.10.4 Volltextsuche	92
7.10.5 Exportfunktion	92
7.10.6 Fazit	93

7.11 Frontend-Umsetzungen	94
7.11.1 Aufbau der Komponentenhierarchie	95
7.11.2 Store-Schichten und Datenfluss	99
7.11.3 Fazit	100
7.12 Zusätzliche Quellen	101
7.12.1 Guidle	101
7.12.2 Rheintal Kalender	102
7.12.3 Mobilizon	102
7.12.4 FCSG-Forum	102
7.12.5 Facebook	104
8 Qualitätssicherung	106
8.1 Testkonzept	106
8.1.1 Unit Tests	106
8.1.2 LLM-Evaluationstest	107
8.1.3 Linting und CI/CD	109
8.2 Test Plan mit Endanwendern	110
8.2.1 Usability-Test	110
8.2.2 Akzeptanztest	111
8.3 Validierung von NFRs	113
8.4 Deployment Test	118
8.5 Prozess-Logging	119
8.5.1 Pipeline Runtime Log	119
8.5.2 Crawl-Zusammenfassung	120
8.5.3 LLM-Prozess-Logging	121
8.5.4 Prompt- und Hint-Logging	121
8.5.5 Monitoring	122
9 Projektmanagement	123
9.1 Planung	123
9.1.1 Rollenverteilung	123
9.1.2 Sitzungsformate	124
9.2 Vorgehen	125
9.2.1 Sprintstruktur	125
9.2.2 Aufgabenverwaltung	125
9.2.3 Zeiterfassung	125
9.3 Risikomanagement	126
9.4 Zeitmanagement	128
9.5 Fazit	129
10 Softwaredokumentation	130
10.1 Installation	130
10.2 Benutzerhandbuch	131
10.2.1 Kopfzeile und Darstellung	131
10.2.2 Karte und Event-Marker	131
10.2.3 Suchfunktion	131
10.2.4 Burgermenü und Filter	131
10.3 Weiterentwicklung	132
10.3.1 Datenquellen	132

10.3.2 Extraktion und Klassifizierung	132
10.3.3 Deduplizierung und Aktualisierung	132
10.3.4 Geo-Komponente	132
10.3.5 Frontend	132
10.3.6 Betrieb und Wartung	133
10.3.7 Fachliche Erweiterungen	133
10.3.8 Produktivsetzung	133
A Glossar	134
A.1 Projektbegriffe	134
A.2 Anforderungs- und Vorgehensbegriffe	134
A.3 Pipeline und Verarbeitung	134
A.4 KI und Sprachmodelle	135
A.5 Geodaten	136
A.6 Datenbank und Speicher	136
A.7 Backend und API	136
A.8 Frontend	136
A.9 Werkzeuge und Standards	136
A.10 Datenformate	137
A.11 Externe Quellen und Dienste	137
A.12 Organisation	138
B Tabellenverzeichnis	139
C Abbildungsverzeichnis	141
D Codeverzeichnis	142
E Bibliographie	143
F Anhang	145
F.1 Rückblick	146
F.1.1 Martyn Foreman	146
F.1.2 Fazit Nicola Aebi	147
F.1.3 Aufgabenstellung	149
F.2 Modulbeschreibung Bachelor-Arbeit Informatik	151
F.2.1 Metadaten	151
F.2.2 Lernziele	151
F.2.3 Plan und Lerninhalt	152
F.2.4 Meilensteine	152
F.2.5 README – Installation & Setup	153
F.3 Meeting-Protokolle	155
F.3.1 Sitzungsprotokolle	155

Teil I – Technischer Bericht

Dieser Teil etabliert den organisatorischen und konzeptionellen Rahmen der Bachelorarbeit. Er beschreibt Projektplanung, Vorgehensweise, Risikomanagement und Ressourceneinsatz, beleuchtet den aktuellen Stand der Technik und definiert die Bewertungskriterien für die entwickelte Lösung. Anschliessend wird das architektonische Gesamtkonzept erläutert, bevor eine kritische Einschätzung der erzielten Ergebnisse und ein Ausblick den Teil abschliessen. Als Fortsetzungsprojekt grenzt dieser Teil explizit ab, welche Grundlagen aus der Studienarbeit übernommen wurden und welche Weiterentwicklungen im Rahmen dieser Bachelorarbeit entstanden sind.

1 Einführung

In diesem Kapitel wird das Projekt „Event-Suchroboter-Cockpit“ (ESC) eingeführt und der inhaltliche Rahmen der Arbeit festgelegt. Ausgehend von der Problemstellung und Vision werden die Ausgangslage sowie der Industriepartner vorgestellt. Im weiteren Verlauf erfolgt die Definition von Zielen, Unterzielen und Abgrenzungen sowie die Beschreibung der organisatorischen Rahmenbedingungen. Aus dem vorliegenden Kontext wird die zentrale Leitfrage abgeleitet, bevor abschließend das methodische Vorgehen und der Aufbau der Arbeit erläutert werden.

1.1 Problemstellung und Vision

Das Projekt „Event-Suchroboter-Cockpit“, (ESC) geht auf eine Initiative der Kantonspolizei St.Gallen (KaPo SG) zurück. Ziel der ursprünglichen Beauftragung war es, die vorausschauende polizeiliche Einsatzplanung und Lagebeurteilung durch eine automatisierte, zentrale Erfassung sicherheitsrelevanter öffentlicher Ereignisse massgeblich zu unterstützen. Bisherige manuelle und zeitintensive Rechercheprozesse sollten durch eine systemische Lösung abgelöst werden, um die Informationsqualität zu steigern und den Informationsfluss zu optimieren [1, p. 1].

Die vorliegende Bachelorarbeit baut direkt auf der in diesem Kontext entstandenen Studienarbeit auf. Während die Vorarbeit das Fundament in Form eines ersten funktionalen Prototyps legte, fokussiert sich dieses Nachfolgeprojekt auf die methodische und technologische Überführung in einen umfassenden, robusten Proof-of-Concept. Dabei werden identifizierte Ausbaupotenziale der bisherigen Lösung gezielt adressiert und das System um weitreichende Kernfunktionalitäten erweitert.

Die zentrale Leitfrage der Arbeit lautet:

Lassen sich im Web veröffentlichte Veranstaltungen im Kanton St.Gallen automatisiert so erfassen, kategorisieren und visualisieren, dass zeitlich-räumliche Koinzidenzen erkennbar werden?

Diese Frage motiviert zwei technische Kernanforderungen: Das System muss Veranstaltungsdaten mit validierten Koordinaten und Zeitangaben so strukturieren, dass zeitlich-räumliche Überschneidungen direkt erkennbar werden. Zudem muss jedes Event einer Kategorie zugeordnet werden, damit Art und Kontext einer Veranstaltung auf einen Blick einschätzbar sind.

1.2 Ausgangslage

Das Lage- und Nachrichtenzentrum (LNZ) der Kantonspolizei St.Gallen ist dafür zuständig, jederzeit ein aktuelles Bild der Sicherheitslage im Kanton zu führen. Ein wesentlicher Bestandteil dieser Arbeit ist die proaktive und frühzeitige Erkennung sicherheitsrelevanter Ereignisse. Darunter fallen im Web veröffentlichte Veranstaltungen wie Konzerte, Sportanlässe oder Demonstrationen, die einen erhöhten Personaleinsatz oder koordinierte polizeiliche Massnahmen erfordern. Eine besondere Herausforderung für die vorausschauende Einsatzplanung entsteht, wenn mehrere solcher Veranstaltungen zeitgleich und räumlich konzentriert stattfinden.

Bislang erfolgt die Erfassung solcher Ereignisse überwiegend manuell. Mitarbeitende sichten Gemeinde- und Vereinswebseiten, kantonale Veranstaltungsportale, Social-Media-Plattformen sowie strukturierte Event-APIs. Dieser Prozess ist zeitintensiv, schwer skalierbar und fehleranfällig, da die relevanten Informationen verstreut und in unterschiedlichen Formaten vorliegen – von strukturierten API-Antworten über HTML-Webseiten bis hin zu PDF-Dokumenten.

Um diese informationelle Lücke zu schliessen und die operationelle Lageerfassung technologisch zu unterstützen, wurde durch die Kantonspolizei St.Gallen das Projekt „Event-Suchroboter-Cockpit“ (ESC) initiiert. Durch den Einsatz eines automatisierten Systems sollen relevante offene Datenquellen effizienter erschlossen und die Informationsqualität gesteigert werden. Das vorliegende Projekt baut auf einer vorausgegangenen Studienarbeit auf und überführt den bestehenden Prototyp in einen erweiterten Proof-of-Concept.

1.3 Industriepartner

Die Kantonspolizei St.Gallen (KaPo SG) fungiert als externer Auftraggeber und Partner für dieses Projekt. Als kantonale Sicherheitsbehörde ist sie für die Aufrechterhaltung von Ruhe, Sicherheit und Ordnung sowie für die Kriminalitätsbekämpfung auf dem gesamten Kantonsgebiet zuständig. Mit rund 980 Mitarbeitenden und 21 Polizeistationen stellt die Organisation die polizeiliche Grundversorgung sicher und betreibt zudem die kantonale Notrufzentrale [2].

Die Konzeption und Umsetzung des Event-Suchroboter-Cockpits (ESC) erfolgen in enger Zusammenarbeit mit dem Lage- und Nachrichtenzentrum (LNZ) der Kantonspolizei unter der fachlichen Begleitung von Patrick Steinmann. Das LNZ bildet die operative Informationsdrehscheibe der Kantonspolizei. Als aktive polizeiliche Nachrichtenzelle ist das LNZ für die permanente Überwachung und Analyse der Sicherheitslage zuständig und erarbeitet auf Basis der gesammelten Erkenntnisse Handlungsempfehlungen für Einsätze [3]. Die Einbindung von Fachexperten aus dem LNZ stellt sicher, dass das automatisierte Cockpit optimal auf die spezifischen Bedürfnisse der polizeilichen Informationsbeschaffung und Lagebeurteilung ausgerichtet ist.

1.4 Aufgabenstellung

Ausgehend von den Bedürfnissen des Industriepartners ist das primäre Ziel dieser Bachelorarbeit die Konzeption sowie die Realisierung eines funktionsfähigen Proof-of-Concepts eines Event-Suchroboter-Cockpits. Das System dient der automatisierten Erfassung, Analyse, Kategorisierung, Validierung, Deduplizierung und benutzerzentrierten Visualisierung von Veranstaltungen für frei definierbare Lokalitäten und Zeiträume. Die Erschliessung und Verarbeitung der Datenquellen erfolgt über einen modular erweiterbaren Wertstrom (Value Stream).

Detaillierte Spezifikationen sowie die formale Projektbeschreibung aus dem AVT-Portal [4] sind im Anhang Abschnitt F.1.3 hinterlegt.

1.4.1 Ziele

Zur Erreichung des übergeordneten Projektziels werden folgende technische und konzeptionelle Teilbereiche bearbeitet:

- **Harvester für Gemeindeseiten:** Entwicklung eines Crawlers zur Extraktion von Ereignisdaten aus HTML-Strukturen und PDF-Dokumenten. Mittels KI-gestützter Analyse werden relevante Attribute (Titel, Zeitstempel, Ort, Kategorie) identifiziert und in einer PostgreSQL-Datenbank persistiert.
- **Integration von Event-Quellen:** Anbindung strukturierter Schnittstellen, namentlich Eventfrog und Guidle. Optional wird die Integration dezentraler Plattformen (z. B. Mobilizon) geprüft.
- **Erschliessung von Social-Media- und Foren-Daten:** Evaluation und Anbindung offener Plattformen zur Prüfung ihrer Eignung als Ereignisquellen unter Berücksichtigung technischer Restriktionen.
- **Dynamisches Aktualisierungsmanagement:** Implementierung von Mechanismen zur Erkennung von Mutationen an bereits erfassten Events, einschliesslich zeitlicher Verschiebungen oder Absagen.
- **Datenpipeline-Orchestrierung:** Aufbau einer robusten Pipeline zur Bereinigung, Validierung und Deduplizierung der Datenströme, ergänzt durch ein systemweites Logging und Monitoring.
- **Automatisierte Kategorisierung:** Klassifizierung der Ereignisse basierend auf Standards wie schema.org oder tourismdata.ch. Bei fehlenden Metadaten erfolgt eine KI-basierte Zuordnung, deren Präzision anhand definierter Kriterien evaluiert wird.
- **Optimierung des Frontends:** Weiterentwicklung der Benutzeroberfläche hinsichtlich Usability und Informationsdichte, insbesondere durch eine interaktive Web-Karte sowie effiziente Such- und Filterfunktionen. Zur Erreichung des Projektziels werden folgende Teilbereiche bearbeitet:

1.4.2 Herausforderungen

Die Umsetzung des ESC ist mit folgenden zentralen Herausforderungen verbunden:

- **Inter-Source-Deduplizierung:** Identifikation und Zusammenführung identischer Ereignisse, die redundant über verschiedene, unabhängige Quellen publiziert werden.
- **Normalisierung heterogener Daten:** Einheitliche Extraktion von Kerninformationen trotz massiv variierender Datenstrukturen in Webseiten, Dokumenten und Schnittstellen.
- **Qualität der KI-Klassifizierung:** Sicherstellung einer hohen Zuverlässigkeit bei der automatisierten Kategorisierung, insbesondere bei unvollständiger Datenbasis.
- **Zustandsverfolgung unstrukturierter Quellen:** Detektion von Event-Absagen oder Änderungen bei Quellen, die über keine strukturierten Update-Signale verfügen.
- **Rechtliche und technische Hürden bei Drittplattformen:** Stabiler Datenbezug von Social-Media-Diensten unter Einhaltung von Zugriffsbeschränkungen und rechtlichen Rahmenbedingungen.

1.5 Vorarbeiten und Abgrenzung

Die vorliegende Bachelorarbeit ist die technologische und konzeptionelle Fortsetzung einer im vorangegangenen Semester durchgeführten Studienarbeit [1]. In der Vorarbeit wurde ein erster funktionsfähiger Prototyp eines Event-Suchroboters entwickelt, der Veranstaltungen automatisiert aus Webquellen erfasst, verarbeitet und in einer Webapplikation visualisiert.

Dabei wurde eine modulare Datenpipeline aufgebaut und die Ereignisextraktion durch eine Kombination aus deterministischer Vorverarbeitung und LLM-basierter Analyse (Large Language Models) umgesetzt. Die Events wurden adressbasiert geokodiert und auf einer interaktiven Karte dargestellt. Die Resultate der Studienarbeit zeigten das grosse Potenzial des Systems auf, identifizierten jedoch auch klare Grenzen insbesondere bei der Verarbeitung unstrukturierter Daten, bei Event-Mutationen sowie bei der Geokodierung von Flächenereignissen [1, p. 44-46].

Um den technischen und konzeptionellen Ausgangszustand für das vorliegende Projekt zu definieren, fasst die nachfolgende Übersicht (Tabelle 3) die im Rahmen der Studienarbeit erzielten Resultate zusammen. Diese Ergebnisse bilden das Fundament der Bachelorarbeit.

Themenbereich	Erreichte Resultate der Studienarbeit (Status quo)
• Gesamtarchitektur	Entwicklung eines funktionsfähigen, modularen Event-Suchroboters zur automatisierten Erfassung, Verarbeitung und strukturierten Speicherung von Veranstaltungen.
• Event-Erfassung und Extraktion	Gewinnung konsistenter Kerndaten durch eine Kombination aus deterministischer Vorverarbeitung und LLM-basierter Extraktion.
• Datenqualität und Normalisierung	Implementierung initialer Normalisierungs- und Bereinigungsschritte zum Ausgleich inkonsistenter Quellinformationen (z. B. Zeiten und Adressen).
• Geokodierung	Anreicherung der Events mit präzisen Koordinaten (WGS84) über das schweizweite Gebäudeadressverzeichnis; mit bekannten Limitationen bei grossflächigen Spezialevents.
• Frontend und Visualisierung	Bereitstellung einer Webapplikation zur räumlichen Exploration der aufbereiteten Event-Daten mittels Kartenansicht und zeitlichen Filtern.

Tabelle 3: Übersicht der in der Studienarbeit realisierten Systemkomponenten [1, p. 44-46]

1.6 Rahmenbedingung

Gemäss der offiziellen Modulbeschreibung [5] des Studiengangs Informatik an der OST dient die Bachelorarbeit dem Nachweis der Fähigkeit, komplexe Informatik-Problemstellungen unter Anwendung ingenieurmässiger Methoden selbstständig zu lösen. Die vorliegende Arbeit erfüllt diese Anforderungen durch eine systematische Gliederung in konzeptionelle, theoretische und praktische Teilbereiche. Der Umfang der Arbeit ist auf 12 ECTS-Punkte festgesetzt, was einem kalkulierten Arbeitsaufwand von circa 360 Stunden pro Person entspricht. Der Bearbeitungszeitraum erstreckt sich vom 16. Februar 2026 bis zum definitiven Abgabetermin am 12. Juni 2026. Als verbindliche Grundlage für die Durchführung und die formalen Anforderungen dient der „Leitfaden für Bachelor- und Studienarbeiten“ [6] in der aktuell gültigen Version.

1.7 Leitfrage

Das Projekt verfolgt die folgende Leitfrage:

Lassen sich im Web veröffentlichte Veranstaltungen im Kanton St.Gallen automatisiert so erfassen, kategorisieren und visualisieren, dass zeitlich-räumliche Koinzidenzen erkennbar werden?

Die Frage ist motiviert durch das zentrale Einsatzszenario der Kantonspolizei St.Gallen: Parallel stattfindende Veranstaltungen sind das primäre polizeiliche Risiko, da sie die verfügbaren Ressourcen gleichzeitig beanspruchen. Um dieses Risiko frühzeitig zu erkennen, muss ein System nicht nur Veranstaltungen erfassen, sondern diese so strukturieren und darstellen, dass zeitliche und räumliche Überschneidungen direkt sichtbar werden.

Die Leitfrage lässt sich in drei Teilprobleme aufgliedern:

Erfassung: Lassen sich heterogene, öffentlich zugängliche Veranstaltungsquellen automatisiert und flächendeckend erschliessen?

Kategorisierung: Können erfasste Events automatisch einer definierten Kategorie zugeordnet werden, sodass Art und Kontext einer Veranstaltung unmittelbar einschätzbar sind?

Visualisierung: Lässt sich eine kombinierte Filterung nach Zeit und Region so realisieren, dass zeitlich-räumliche Koinzidenzen auf einen Blick erkennbar sind?

Das Projekt beantwortet diese Frage im Rahmen eines Proof-of-Concept, der auf der Studienarbeit [1] aufbaut und alle drei Teilaspekte in einem integrierten System umsetzt.

1.8 Vorgehen und Aufbau der Arbeit

Das Projekt folgte einem iterativen, sprintbasierten Ansatz mit acht zweiwöchigen Sprints. Planung, Aufgabenverwaltung und Fortschrittsdokumentation erfolgten über YouTrack. Die Sprints wurden gemeinsam mit den Betreuern in regelmässigen Reviews abgenommen, was eine kontinuierliche Rückkopplung zwischen Entwicklung und Anforderung sicherstellte.

Die Arbeit gliedert sich in zwei Hauptteile: Teil I (Technischer Bericht) dokumentiert den fachlichen und konzeptionellen Rahmen der Arbeit – von Problemstellung und Stand der Technik über Architektur und Resultate bis zu Schlussfolgerungen. Teil II (SW-Projektdokumentation) hält den Engineering-Lebenszyklus fest und umfasst Anforderungsanalyse, Systemdesign, Implementationsdokumentation, Qualitätssicherung, Projektmanagement und Deployment.

2 Stand der Technik

Dieses Kapitel ordnet das Projekt technologisch und fachlich ein. Es beleuchtet die eingesetzten Tools und Technologien und grenzt das System gegenüber bestehenden Lösungsansätzen ab.

2.1 Evaluation und Bestätigung des Technologie-Stacks

Obwohl die grundsätzliche Technologieevaluation bereits im Rahmen der Studienarbeit durchgeführt wurde, erfordert die Weiterentwicklung des Event-Suchroboter-Cockpits eine kurze Reflexion der eingesetzten Kernkomponenten. Die Studienarbeit bildete die technische Grundlage für die Bachelorarbeit und setzte bereits auf Python als Backend-Sprache sowie PostgreSQL als persistente Datenbanklösung [1, S. 9-10].

Auf Basis der bisherigen Projektergebnisse wurde entschieden, diesen etablierten Stack weiterhin zu verwenden. Python bietet durch sein umfangreiches Ökosystem an Bibliotheken für Web-Scraping sowie durch seine Unterstützung asynchroner Verarbeitungsmechanismen weiterhin eine geeignete Grundlage für die Datenextraktion und die Anbindung der Large Language Models. PostgreSQL, welches über Psycpg3 integriert ist, deckt parallel dazu die Anforderungen an die strukturierte Speicherung der aggregierten Eventdaten ab.

Folgende Technologien und Tools wurden für die Weiterentwicklung des Systems evaluiert und/oder bestätigt:

2.1.1 Crawler-Pipeline

Python: Als bewährte Backend-Sprache mit umfangreichen Bibliotheken für Web-Scraping, Datenverarbeitung und KI-Integration bleibt Python die erste Wahl für die Backend-Entwicklung.

Playwright: Das Playwright-Framework wird weiterhin für die Implementierung des Crawlers genutzt, da es eine robuste und flexible Lösung für die Interaktion mit dynamischen Webseiten bietet. Alternative Tools wie Crawl4AI wurden zwar in Betracht gezogen, konnten jedoch nicht die gleiche Stabilität und Anpassungsfähigkeit gewährleisten. [1, S. 9-10]

BeautifulSoup: Für die HTML-Parsing-Aufgaben wird weiterhin auf BeautifulSoup zurückgegriffen, da es eine einfache und effektive Möglichkeit bietet, die gesammelten Webdaten zu strukturieren.

markdownify: Konvertiert HTML-Fragmente in Markdown – das kompakte Eingabeformat für die LLM-Extraktion.

GitHub-Flavored Markdown (GFM): Verbreitetes Speicherformat für Markdown-Dateien, die als Zwischenoutput der Pipeline dienen. Standardausgabe vom markdownify-Tool, kompatibel mit gängigen Markdown-Renderern.

OpenAI API: OpenAI-kompatibler Client für die OST interne LLMHub-API.

Pydantic v2: Validierung und Typisierung der extrahierten Eventdaten (`EventInfo`); erzwingt ein konsistentes Datenschema durch die gesamte Pipeline.

pyproj: Koordinatentransformation von Schweizer Landeskoordinaten (LV95/EPSG:2056) nach WGS84 (EPSG:4326) beim Import des amtlichen Adressverzeichnisses.

pandas: Tabellarische Aufbereitung beim Import der Geo-/Adress-Datensätze.

aiohttp / requests: Asynchrone bzw. synchrone HTTP-Aufrufe (Datei-Downloads, API-Calls ausserhalb des Browsers).

psycopg v3: PostgreSQL-Treiber für die Persistierung der Events; synchron in Threads ausgeführt, um Plattformprobleme der asyncio-Variante zu vermeiden.

sentry-sdk: Einfache Integration für Fehler- und Performance-Tracking der Pipeline-Läufe (Transaktionen, Exception-Capturing pro Worker).

python-json-logger / logging: Strukturiertes JSON-Logging für die spätere automatisierte Log-Auswertung.

tabulate: Formatierte Tabellenausgabe in der Konsolen-/Log-Auswertung.

asyncio: Trägt die queue-basierte Multi-Stage-Pipeline; ermöglicht parallele Worker ohne Thread-Overhead.

uv: Schneller Python-Paket- und Umgebungsmanager (ersetzt pip/venv), liefert reproduzierbare Installationen via Lockfile und bietet integrierte Befehle für Testing, Linting und Code-Formatierung.

go-task: Task-Runner für die Orchestrierung von Entwicklungs- und Wartungsaufgaben (z.B. Datenbank-Migrationen, Testläufe, etc.).

2.1.2 API

FastAPI: Performantes, typannotiertes Web-Framework; liefert automatische Request-Validierung und interaktive OpenAPI-/Swagger-Dokumentation (`/docs`) ohne Zusatzaufwand.

Uvicorn: ASGI-Server, der die FastAPI-App ausführt (`fastapi run`).

psycopg v3 + psycopg-pool: Direkter PostgreSQL-Zugriff ohne ORM-Overhead; der Connection-Pool vermeidet teures Neuaufbauen von Verbindungen pro Request.

2.1.3 Frontend

TypeScript: Typsichere Frontend-Entwicklung; verhindert Laufzeitfehler bei der Verarbeitung.

SvelteKit (Svelte 5, Runes): Als schlankes, kompilierendes UI-Framework erzeugt SvelteKit bereits zur Build-Zeit hochoptimierten JavaScript-Code, wodurch das ausgelieferte Bundle deutlich kleiner ausfällt als bei Virtual-DOM-basierten Alternativen wie React oder Vue. Mit der neuen Runes-API (`$state` , `$effect` , `$props`) bietet Svelte 5 zudem eine feingranulare, signalbasierte Reaktivität, die für ein datenintensives Cockpit mit häufig wechselnden Filterzuständen besonders geeignet ist.

Vite: Build-Tool und Dev-Server (via SvelteKit). Vite ermöglicht durch natives ES-Module-Handling extrem schnelles Hot Module Replacement, was die Entwicklungszyklen verkürzt und das Testen von UI-Änderungen erheblich beschleunigt.

@sveltejs/adaptor-node: Erzeugt aus dem SvelteKit-Projekt einen eigenständigen Node-Server, der innerhalb des Docker-Containers betrieben werden kann. Dies ist gegenüber

statischen Adaptern erforderlich, da Endpoint-Handler und serverseitige Konfiguration zur Laufzeit verfügbar sein müssen.

Node.js 22: JavaScript-Laufzeitumgebung für den Build-Prozess und den produktiven Serverbetrieb des SvelteKit-Frontends. Die Version 22 ist eine LTS-Release (Long-Term Support) und garantiert damit über mehrere Jahre hinweg Sicherheits- und Stabilitäts-Updates.

Leaflet: Etablierte, leichtgewichtige Open-Source-Kartenbibliothek mit einer breiten Community und umfangreicher Plugin-Landschaft. Sie stellt die interaktive Karte auf Basis frei verfügbarer OpenStreetMap-Tiles dar und vermeidet so Abhängigkeiten von kommerziellen Anbietern wie Google Maps oder Mapbox.

leaflet.markercluster: Plugin für Leaflet, das räumlich nahe beieinanderliegende Event-Marker dynamisch zu Clustern zusammenfasst. Dies verhindert visuelle Überladung bei hoher Eventdichte und verbessert die Performance der Karte spürbar, da nicht jeder Marker einzeln gerendert werden muss.

bits-ui: Bibliothek zugänglicher, ungestylter UI-Primitive für Svelte. Liefert die Slider-Komponenten für die Zeitraum- und Uhrzeit-Filter und stellt dabei Tastaturnavigation und ARIA-Konformität ohne zusätzlichen Implementierungsaufwand sicher.

TypeScript: Statische Typisierung für Stores, API-Antworttypen und Komponenten-Props. Da die API-Antworten der FastAPI über generierte oder manuell gepflegte Typen abgebildet werden, lassen sich Schemaänderungen bereits zur Build-Zeit erkennen, was Laufzeitfehler im Frontend reduziert.

ESLint und Prettier: Linting bzw. einheitliche Code-Formatierung. Diese Werkzeuge sichern eine konsistente Code-Qualität und vermeiden stilistische Diskussionen, was insbesondere in einer Bachelorarbeit mit beschränkter Bearbeitungszeit wertvoll ist.

2.1.4 Datenbank

PostgreSQL 18: Zentrale relationale Datenbank für Events, Kategorien und Geo-Lookups. PostgreSQL überzeugt durch ausgereifte Transaktionssicherheit, einen ausgereiften Query-Planner und ein besonders breites Ökosystem an Erweiterungen, das den Funktionsumfang gezielt um Volltextsuche, Vektorähnlichkeit und unscharfe Textvergleiche ergänzt. Damit lassen sich relationale Eventdaten, semantische Kategoriezuordnungen und fehlertolerantes Geocoding in einer einzigen Datenbank vereinen, ohne zusätzliche Spezial-Dienste betreiben zu müssen.

pgvector: Stellt den `vector`-Datentyp sowie passende Ähnlichkeitsoperatoren bereit und ermöglicht so die semantische Kategoriezuordnung per Embedding-Nähe (Cosinus-Distanz). Dadurch können neue Events Kategorien zugeordnet werden, ohne dass diese exakt textlich übereinstimmen müssen. Das Basis-Image `pgvector/pgvector:pg18-trixie` liefert die Extension bereits mit, sodass kein eigener Build-Schritt nötig ist.

pg_textsearch: Erweiterung, die einen BM25-Volltextindex bereitstellt und damit eine relevanzbasierte Titelsuche ermöglicht. BM25 liefert in Vergleichen zu klassischen `tsvector`-Suchen deutlich brauchbarere Treffer-Reihenfolgen bei kurzen Eventtiteln.

2.1.5 Infrastruktur und DevOps

Docker und Docker Compose: Containerisierung und Orchestrierung der drei Hauptdienste (`db` , `api` , `frontend`). Durch die deklarative Konfiguration in `docker-compose.yml` wird die gesamte Umgebung inklusive Datenbank-Initialisierung mit einem einzigen Befehl (`docker compose up -d`) reproduzierbar gestartet. Dies eliminiert klassische „works on my machine“-Probleme, vereinfacht das Onboarding neuer Entwickelnder erheblich und stellt sicher, dass Entwicklungs-, Test- und Produktivumgebungen identisch konfiguriert sind. Zudem erlaubt die Container-Trennung eine unabhängige Skalierung und einen einfachen Austausch einzelner Komponenten.

go-task (Taskfile): `make` -ähnlicher, plattformunabhängiger Task-Runner, der über eine YAML-Konfiguration (`Taskfile.yml`) wiederkehrende Backend- und Wartungsaufgaben kapselt zum Beispiel die Generierung der Kategorie-Embeddings oder den Import des amtlichen Adressverzeichnisses. Im Gegensatz zu `make` ist `go-task` ohne zusätzliche Abhängigkeiten unter Windows, macOS und Linux einsetzbar und bietet eine klar lesbare Syntax mit deklarativen Abhängigkeiten zwischen Tasks.

Git: Verteilte Versionskontrolle als Industriestandard. Sie ermöglicht nachvollziehbare Änderungen über die gesamte Projektdauer, parallele Entwicklung an Features über Branches sowie eine zuverlässige Wiederherstellung früherer Zustände. In Kombination mit GitLab oder GitHub wird zudem Code-Review und Issue-Tracking abgedeckt.

Sentry: Gehostete Plattform für Error- und Performance-Monitoring der Crawler-Pipeline. Sentry erfasst aufgetretene Exceptions inklusive Stacktrace, betroffenem Worker und Pipeline-Kontext und gruppiert wiederkehrende Fehler automatisch. So lassen sich Probleme im produktiven Betrieb frühzeitig erkennen und gezielt analysieren, ohne dass manuell Logs durchsucht werden müssen.

2.2 Bestehende Lösungsansätze

Bevor ein eigenes System entwickelt wird, stellt sich die Frage, warum bestehende Event-Plattformen den Anforderungen der Kantonspolizei St.Gallen nicht genügen. Die folgende Analyse zeigt, dass die zentralen Lücken weniger in der technischen Umsetzung der Plattformen liegen als in deren Ausrichtung: Bestehende Lösungen sind auf die Vermarktung von Veranstaltungen für Endnutzer ausgelegt und nicht auf den behördlichen Anwendungsfall einer systematischen Lagebeurteilung.

2.2.1 Fragmentierung der Quellen

Der Kanton St.Gallen umfasst gemäss der Fachstelle für Statistik 75 politische Gemeinden [7]. Da diese Gemeinden ihre Daten und Veranstaltungshinweise dezentral auf ihren jeweiligen Webauftritten publizieren, variieren die Formate in der Praxis stark – von klassischen HTML-Seiten über PDF-Kalender bis hin zu unstrukturiertem Fliesstext. Eine Erfassung auf zentralen Plattformen wie Eventfrog oder Guidle setzt eine aktive, manuelle Registrierung der jeweiligen Veranstalter voraus [8]. Daraus lässt sich schliessen, dass rein kommunale oder vereinsinterne Termine, die ausschliesslich lokal auf den Gemeinde-Websites angekündigt werden, auf solchen überregionalen Plattformen systembedingt oft unvollständig bleiben.

2.2.2 Eingeschränkte zeitliche Filterung

Guidle bietet eine Kartenansicht mit Filterung nach Kategorie und Datum – eine Filterung nach Uhrzeit fehlt jedoch [9]. Für die polizeiliche Lagebeurteilung ist diese zeitliche Differenzierung potenziell entscheidend: Zwei Veranstaltungen am selben Tag wären in der Regel unproblematisch, sofern eine morgens und die andere abends stattfindet. Kritisch könnte es hingegen dann werden, wenn beide Ereignisse gleichzeitig ablaufen und dadurch Sicherheitsressourcen parallel binden. Ohne Uhrzeitfilter müsste die Polizei jeden gefundenen Event manuell auf seine genaue Startzeit prüfen.

2.2.3 Fehlende Exportierbarkeit und Weiterverarbeitbarkeit

Kommerzielle Event-Plattformen sind auf die Präsentation von Veranstaltungen für Endnutzer ausgelegt – nicht auf die maschinelle Weiterverarbeitung der Daten. Ein strukturierter Export der Eventliste als CSV, JSON oder in ein anderes maschinenlesbares Format ist auf Guidle und Eventfrog nicht vorgesehen [9], [10]. Wer dennoch mit den Daten arbeiten möchte, ist auf manuelles Kopieren oder auf inoffizielle Scraping-Methoden angewiesen, die bei Layout-Änderungen der Plattform jederzeit brechen können. Für einen polizeilichen Anwendungsfall, der eine Integration in eigene Systeme, eine Geocodierung der Adressen oder eine automatisierte Kategorisierung erfordert, sind diese Plattformen damit strukturell ungeeignet. Das ESC exportiert Eventdaten direkt über eine dokumentierte API und stellt sicher, dass alle Felder – Titel, Beschreibung, Zeit, Koordinaten, Kategorie – maschinenlesbar und weiterverarbeitbar vorliegen.

2.2.4 Fazit

Die beschriebenen Lücken begründen die Eigenentwicklung: Ein spezialisiertes System, das heterogene Quellen automatisiert erschliesst, Events kategorisiert und eine geographisch-zeitliche Filteransicht bietet, lässt sich mit bestehenden Plattformen nicht abbilden. Das Event-Suchroboter-Cockpit ist damit keine Konkurrenz zu Eventfrog oder Guidle, sondern eine Ergänzung, die genau jene Daten sichtbar macht, die auf diesen Plattformen fehlen.

3 Evaluation

Die Evaluation prüft systematisch die für den Proof-of-Concept zentralen Architektur- und Technologieentscheide. Die Erkenntnisse bilden das Fundament für das nachfolgende Umsetzungskonzept (Kapitel 4).

3.1 Evaluationskriterien

Die in diesem Kapitel dokumentierten Architektur- und Technologieentscheide betreffen unterschiedliche Teilbereiche des Systems, folgen jedoch einer vergleichbaren Fragestellung: Aus mehreren möglichen Optionen ist diejenige zu bestimmen, die den Anforderungen des Proof-of-Concepts am besten entspricht. Eine ausschliesslich qualitative Gegenüberstellung von Vor- und Nachteilen macht zwar die relevanten Argumente sichtbar, lässt jedoch offen, wie stark die einzelnen Aspekte gegeneinander gewichtet wurden. Die Nachvollziehbarkeit der finalen Auswahl bleibt dadurch eingeschränkt. Um die Entscheidungen vergleichbar, überprüfbar und konsistent zu gestalten, werden sie auf ein einheitliches, quantifizierbares Bewertungsverfahren gestützt.

Als Methode dient eine Nutzwertanalyse. Dabei wird jede Option entlang eines gewichteten Kriterienkatalogs bewertet und zu einem Gesamtnutzwert verdichtet. Die Option mit dem höchsten Nutzwert gilt als die für das Projekt geeignetste Lösung. Der Katalog wird einheitlich auf alle nachfolgenden Evaluationen angewendet. Die qualitative Abwägung der Vor- und Nachteile entfällt damit nicht, sondern bildet die inhaltliche Begründung für die Vergabe der einzelnen Bewertungen.

3.1.1 Kriterienkatalog

Der Kriterienkatalog umfasst fünf Kriterien, welche die im Projekt wiederkehrenden Entscheidungstreiber abbilden. Die Gewichtung spiegelt die Prioritäten des Proof-of-Concepts wider: Das höchste Gewicht erhält die funktionale Eignung, da in jeder Einzelentscheidung die fachliche Zwecktauglichkeit der Lösung den Ausschlag gibt. Da das System langfristig betrieben und erweitert werden soll, sind zudem die Wartbarkeit sowie der Integrations- und Implementierungsaufwand hoch gewichtet, gefolgt von der Performance und Skalierbarkeit. Die Betriebskosten und der Ressourcenbedarf erhalten ein geringes Gewicht, da kostenkritische Ausprägungen primär über Ausschlusskriterien und nicht über die abgestufte Bewertung berücksichtigt werden. Die Kriterien und ihre Gewichtung sind in Tabelle 4 zusammengefasst; die in Klammern genannten Kurzbezeichnungen werden in den nachfolgenden Bewertungstabellen verwendet.

Kriterium	Beschreibung	Gewichtung
Funktionale Eignung (<i>Eignung</i>)	Grad, in dem die Option die fachliche Aufgabe erfüllt, etwa hinsichtlich Erfolgsquote, Ergebnisqualität und Abdeckung der Anforderungen.	30%
Performance und Skalierbarkeit (<i>Performance</i>)	Verarbeitungsgeschwindigkeit sowie Verhalten der Option bei steigendem Daten- und Quellenvolumen.	20%
Betriebskosten und Ressourcenbedarf (<i>Kosten</i>)	Laufende Kosten sowie Speicher- und Rechenbedarf im Betrieb. Eine hohe Bewertung entspricht einem geringen Bedarf.	5%
Integrations- und Implementierungsaufwand (<i>Aufwand</i>)	Einmaliger Aufwand zur Einbindung der Option in die bestehende Architektur. Eine hohe Bewertung entspricht einem geringen Aufwand.	20%
Wartbarkeit und Komplexität (<i>Wartbarkeit</i>)	Laufender Pflegeaufwand und resultierende Systemkomplexität. Eine hohe Bewertung entspricht einer wartungsarmen und wenig komplexen Lösung.	25%

Tabelle 4: Kriterienkatalog der Nutzwertanalyse mit Gewichtung

Sämtliche Kriterien sind so formuliert, dass eine höhere Bewertung stets eine vorteilhaftere Ausprägung beschreibt. Dies gilt insbesondere für die aufwands- und kostenbezogenen Kriterien, bei denen ein geringer Aufwand beziehungsweise geringe Kosten mit einer hohen Punktzahl bewertet werden. Dadurch lassen sich die gewichteten Bewertungen direkt zu einem Gesamtnutzwert aufsummieren.

3.1.2 Bewertungsskala

Die Bewertung der Optionen je Kriterium erfolgt auf einer fünfstufigen ordinalen Skala. Tabelle 5 definiert die Bedeutung der einzelnen Stufen, um eine möglichst einheitliche und nachvollziehbare Punktvergabe sicherzustellen.

Wert	Bezeichnung	Bedeutung
1	ungenügend	Das Kriterium wird nicht oder nur in geringem Mass erfüllt.
2	gering	Das Kriterium wird unterdurchschnittlich erfüllt.
3	ausreichend	Das Kriterium wird teilweise beziehungsweise zufriedenstellend erfüllt.
4	gut	Das Kriterium wird überdurchschnittlich erfüllt.
5	sehr gut	Das Kriterium wird vollständig erfüllt.

Tabelle 5: Bewertungsskala der Nutzwertanalyse

3.1.3 Berechnung des Nutzwerts

Der Gesamtnutzwert einer Option ergibt sich aus der Summe der mit der jeweiligen Gewichtung multiplizierten Einzelbewertungen. Die Berechnung ist in Gleichung 1 dargestellt, wobei g_i die Gewichtung des Kriteriums i und b_i die zugehörige Bewertung auf der fünfstufigen Skala bezeichnet.

$$N = \sum_{i=1}^n g_i \cdot b_i \quad (1)$$

Die Option mit dem höchsten Nutzwert N wird als die für das Projekt geeignetste Lösung gewählt. Erfüllt eine Option eine zwingende Rahmenbedingung nicht, etwa durch das Überschreiten des verfügbaren Budgets oder der vorhandenen Serverkapazität, so wird sie unabhängig von ihrem Nutzwert als Ausschlusskriterium behandelt und nicht weiter berücksichtigt. Auf Grundlage dieses methodischen Rahmens werden die einzelnen Architektur- und Technologieentscheide in den folgenden Abschnitten bewertet.

3.2 Anpassung der Crawler-Architektur

In der vorangegangenen Projektphase zeigte sich, dass für jede neue Datenquelle ein individueller Crawler implementiert werden musste. Um diese Ineffizienz zu beheben, wurde die Konzeption eines verallgemeinerten Crawlers evaluiert. Dieser Ansatz sieht vor, quellenabhängige Parameter und Extraktionslogiken in separaten Konfigurationsdateien zu kapseln. Diese Konfigurationen werden von einem zentralen Crawler dynamisch eingelesen, anstatt für jede Datenquelle redundanten Programmcode zu generieren.

Für die Entscheidung bezüglich dieser Architekturänderung wurden die folgenden Vor- und Nachteile abgewogen:

Vorteile	Nachteile
• Verschlankter und modularer Scraping-Prozess • Hohe Skalierbarkeit durch einfache Konfigurationsdateien • Langfristige Zeitersparnis bei der Quellenanbindung	• Hoher initialer Entwicklungsaufwand für die Kernkomponente • Erhöhte Systemkomplexität der verallgemeinerten Logik • Erhöhter Testaufwand zur Vermeidung potenzieller Fehlerquellen

Tabelle 6: Vor- und Nachteile der generalisierten Crawler-Architektur

Zur quantitativen Absicherung dieser Abwägung werden die generalisierte Architektur und die bisherige Umsetzung mit quellenspezifischen Crawlern anhand des in Kapitel 3.1 definierten Kriterienkatalogs bewertet. Die Einzelbewertungen leiten sich aus den vorstehend genannten Vor- und Nachteilen ab. Tabelle 7 stellt die Bewertung je Kriterium sowie den resultierenden Nutzwert gegenüber.

Kriterium	Gew.	Generalisiert	Spezifisch
Eignung	30%	5	3
Performance	20%	5	2
Kosten	5%	3	3
Aufwand	20%	2	4
Wartbarkeit	25%	3	2
Nutzwert N		3.8	2.75

Tabelle 7: Nutzwertanalyse Crawler-Architektur: generalisierter gegenüber quellenspezifischem Ansatz

Wie Tabelle 7 zeigt, fällt die Bewertung zugunsten der generalisierten Architektur aus. Ausschlaggebend sind die funktionale Eignung und die Skalierbarkeit, die den höheren Integrationsaufwand überwiegen.

Daher wird die Architektur restrukturiert. Da die Datenquellen mindestens verdoppelt und künftig strukturell heterogene Quellen wie Foren, Social-Media-Plattformen und Event-Websites integriert werden sollen, überwiegen die langfristigen Vorteile den initialen Mehraufwand deutlich.

3.3 Evaluation der Adressauflösung und Geokodierung

In der Studienarbeit wurde das „Amtliche Verzeichnis der Gebäudeadressen“ von swisstopo als alleinige Referenz (Single Source of Truth) für die Adressvalidierung implementiert [1, p. 20]. Dafür wird der gesamte Datensatz lokal vorgehalten, projektspezifisch transformiert und für Adressabfragen genutzt. [11]

In der Praxis hat sich jedoch gezeigt, dass extrahierte Veranstaltungsorte häufig keine eindeutigen Gebäudeadressen aufweisen, sondern oft unpräzise benannt sind (beispielsweise allgemeine Ortsangaben oder Gebäudebezeichnungen). Die strikte Struktur des Gebäudeadressverzeichnisses stösst bei der Auflösung derartiger Angaben an ihre Grenzen. Zudem erfordert die bisherige Implementierung einen manuellen Prozess zur Aktualisierung des lokalen Datenbestands. Aus diesen Gründen wurde die Nutzung alternativer Dienste – wie externe Geokodierungs-APIs oder umfangreichere lokale Datenbanken (beispielsweise Open-StreetMap) – evaluiert.

Für die Entscheidungsfindung wurden die Vor- und Nachteile dieser Alternativen gegenüber der bestehenden Lösung abgewogen:

Vorteile alternativer Lösungsansätze	Nachteile alternativer Lösungsansätze
• Höhere Erfolgsquote bei der Identifikation unpräziser Event-Standorte • Automatisierung des Datenabgleichs (Wegfall manueller Datenbank-Updates)	• Erhebliche laufende Kosten bei API-Nutzung durch hohes Abfragevolumen (ca. 10'000 Events/Tag) • Kritischer Speicherbedarf bei alternativen lokalen Datenbanken (ca. 13 GB, rund 33 % der verfügbaren Serverkapazität)

Tabelle 8: Vor- und Nachteile externer APIs und grosser lokaler Datenbanken für die Adressauflösung

Zur quantitativen Absicherung dieser Abwägung werden die bestehende swisstopo-Lösung und die alternativen Ansätze anhand des in Kapitel 3.1 definierten Kriterienkatalogs bewertet. Tabelle 9 stellt die Bewertung je Kriterium sowie den resultierenden Nutzwert gegenüber.

Kriterium	Gew.	swisstopo	Alternativen
Eignung	30%	4	5
Performance	20%	5	3
Kosten	5%	5	1
Aufwand	20%	5	2
Wartbarkeit	25%	3	4
Nutzwert N		4.2	3.55

Tabelle 9: Nutzwertanalyse Adressauflösung: lokale swisstopo-Datenbank gegenüber alternativen Diensten

Wie Tabelle 9 zeigt, fällt die Bewertung zugunsten der swisstopo-Lösung aus. Erschwerend kommt hinzu, dass die alternativen Ansätze ein Ausschlusskriterium verletzen: Sowohl die laufenden Kosten der API-Nutzung als auch der Speicherbedarf von rund 13 GB überschreiten

die verfügbaren Projektressourcen und schliessen diese Optionen unabhängig von ihrem Nutzwert aus.

Auf Basis dieser Gegenüberstellung wurde entschieden, die etablierte Lösung mittels der lokalen swisstopo-Datenbank beizubehalten. Das Verzeichnis stellt eine hochaktuelle und verlässliche Referenzquelle dar, die mit einem Speicherbedarf von lediglich rund 600 Megabyte äusserst ressourcenschonend arbeitet. Gleichzeitig werden die signifikanten laufenden Kosten vermieden, die bei der Nutzung externer APIs für das hohe tägliche Abfragevolumen anfallen würden.

Um dennoch die Herausforderung unpräziser Ortsangaben zu bewältigen, wurde ein spezialisierter Mechanismus für diese Sonderfälle konzipiert, anstatt das gesamte Adressauflösungssystem auszutauschen. Die konkrete Behandlung von Veranstaltungen ohne eindeutige Adresse wird im Rahmen der technischen Umsetzung im Kapitel 7.9.3 erläutert.

3.4 Workflow der Event-Kategorisierung

Für die systematische Auswertung und Filterung der gesammelten Veranstaltungen im Cockpit ist eine präzise Kategorisierung der Daten unerlässlich. Um diesen Workflow in die bestehende Pipeline zu integrieren, wurden zwei primäre Lösungsansätze evaluiert: die direkte Kategorisierung durch das eingesetzte Large Language Model (LLM) und ein vektorbasierter Ansatz unter der Verwendung von pgvector als Erweiterung der bestehenden PostgreSQL-Datenbank.

Die Integration der Kategorisierungslogik in den bestehenden LLM-Prompt stellt konzeptionell den einfachsten Ansatz dar. Da das LLM bereits für die Extraktion und Strukturierung der Daten zuständig ist, könnte die Zuweisung einer Kategorie in denselben Arbeitsschritt integriert werden, ohne die Systemarchitektur zu verändern. Dem gegenüber steht der Einsatz von pgvector, bei dem Text-Embeddings der Events generiert und über eine Ähnlichkeitssuche (Vector Similarity Search) mit vordefinierten Kategorien abgeglichen werden.

Für die Entscheidungsfindung wurden die Vor- und Nachteile des pgvector-Ansatzes gegenüber der reinen LLM-Lösung abgewogen:

Vorteile (pgvector)	Nachteile (pgvector)
• Deutlich höhere Verarbeitungsgeschwindigkeit • Signifikante Entlastung des LLMs • Hohe Skalierbarkeit bei wachsenden Datenmengen	• Erhöhte allgemeine Systemkomplexität • Umfassende architektonische Anpassungen der Pipeline notwendig

Tabelle 10: Vor- und Nachteile der vektorbasierten Kategorisierung mittels pgvector

Zur quantitativen Absicherung dieser Abwägung werden der pgvector-Ansatz und die reine LLM-Kategorisierung anhand des in Kapitel 3.1 definierten Kriterienkatalogs bewertet. Tabelle 11 stellt die Bewertung je Kriterium sowie den resultierenden Nutzwert gegenüber.

Kriterium	Gew.	pgvector	Reines LLM
Eignung	30%	5	3
Performance	20%	5	2
Kosten	5%	4	3
Aufwand	20%	2	4
Wartbarkeit	25%	3	3
Nutzwert N		3.85	3

Tabelle 11: Nutzwertanalyse Event-Kategorisierung: pgvector gegenüber reiner LLM-Lösung

Die Bewertung in Tabelle 11 bildet einen ausgeprägten Zielkonflikt ab: Die reine LLM-Lösung überzeugt beim geringen Integrations- und Wartungsaufwand, während pgvector seine Stärken bei Performance und Skalierbarkeit ausspielt. Der folgende Abschnitt begründet, weshalb dieser Aspekt für das Projekt den Ausschlag gibt.

Die Analyse des bisherigen Systems zeigte, dass das LLM aufgrund der grossen Mengen an zu verarbeitendem Text bereits den primären Flaschenhals (Bottleneck) der Pipeline darstellt und

die Verarbeitungsgeschwindigkeit des Gesamtsystems stark limitiert. Eine zusätzliche Belastung durch Kategorisierungsaufgaben würde diese Latenzproblematik weiter verschärfen. Aus diesem Grund fiel die Entscheidung zugunsten der Implementierung von pgvector. Obwohl dieser Ansatz einen höheren architektonischen Aufwand und eine gesteigerte Komplexität mit sich bringt, überwiegen die massiven Performance-Gewinne und die Entlastung des LLMs deutlich, insbesondere im Hinblick auf die geplante Skalierung der Datenquellen.

3.5 Monitoring und Fehlertracking

Für den langfristigen Betrieb der Event-Pipeline ist ein zuverlässiges Monitoring unverzichtbar. Da die Pipeline aus mehreren parallelen Workern besteht und stark von externen Quellen (Websites, LLM-API, Geocoding-Daten) abhängt, treten Fehler typischerweise asynchron, kontextabhängig und teils unregelmässig auf. Ohne dediziertes Monitoring lassen sich solche Probleme nur durch das manuelle Durchsuchen von Logdateien rekonstruieren, was bei produktivem Betrieb nicht praktikabel ist.

Zur Auswahl einer geeigneten Lösung wurden zwei etablierte Ansätze evaluiert: der gehostete Dienst **Sentry** sowie der selbst betriebene **Grafana-Stack** (Grafana, Prometheus, Loki und optional Tempo).

Für die Entscheidungsfindung wurden die Vor- und Nachteile von Sentry gegenüber dem Grafana-Stack im Kontext des Projekts abgewogen:

Vorteile (Sentry)	Nachteile (Sentry)
• Sofort einsatzbereit, kein Self-Hosting nötig • Automatisches Exception-Tracking inkl. Stacktrace und Kontext • Natives Python-SDK mit minimalem Integrationsaufwand • Gruppierung wiederkehrender Fehler und Performance-Transaktionen • Kostenloser Tarif deckt erwartetes Volumen des Projekts ab	• Externer Dienst, Daten verlassen das eigene System • Eingeschränkte Anpassbarkeit von Dashboards und Alerts • Bei hohem Event-Volumen kostenpflichtig • Fokus auf Errors/Traces, Log-Aggregation nicht primäres Feature

Tabelle 12: Vor- und Nachteile von Sentry gegenüber einem selbst betriebenen Grafana-Stack

Zur quantitativen Absicherung dieser Abwägung werden beide Optionen anhand des in Kapitel 3.1 definierten Kriterienkatalogs bewertet. Die Einzelbewertungen leiten sich unmittelbar aus den vorstehend genannten Vor- und Nachteilen ab. Tabelle 13 stellt die Bewertung je Kriterium sowie den resultierenden Nutzwert gegenüber.

Kriterium	Gew.	Sentry	Grafana-Stack
Eignung	30%	5	3
Performance	20%	4	4
Kosten	5%	4	2
Aufwand	20%	5	2
Wartbarkeit	25%	5	2
Nutzwert N		4.75	2.7

Tabelle 13: Nutzwertanalyse Monitoring: Sentry gegenüber Grafana-Stack

Wie Tabelle 13 zeigt, fällt das Ergebnis klar zugunsten von Sentry aus. Die folgenden Ausführungen begründen die vergebenen Bewertungen.

Die Analyse zeigt, dass für das Projekt primär ein verlässliches Exception-Tracking benötigt wird: Fehler in der Crawler-Pipeline, fehlgeschlagene LLM-Aufrufe oder unerwartete API-Antworten sollen schnell auffindbar und nachvollziehbar sein. Genau in diesem Bereich liegt die Stärke von Sentry, während der Grafana-Stack seine Vorteile vor allem bei umfangreicher Metrik-Visualisierung und Log-Aggregation ausspielt – Anforderungen, die im aktuellen Projektumfang von untergeordneter Bedeutung sind. Hinzu kommt der zeitliche Rahmen der Bachelorarbeit: Aufbau, Konfiguration und Wartung eines vollständigen Grafana-Stacks würden einen erheblichen Mehraufwand bedeuten, ohne im Gegenzug einen entsprechenden Mehrwert zu liefern.

Aus diesen Gründen fiel die Entscheidung zugunsten von Sentry. Sollte sich der Bedarf an detaillierten Metriken oder zentraler Log-Aggregation zu einem späteren Zeitpunkt ergeben, lässt sich Sentry problemlos um einen Grafana-Stack erweitern, ohne die bestehende Integration zu ersetzen.

3.6 Evaluation der Datenbeschaffung

Die Erfassung der Veranstaltungsdaten stützt sich seit der Studienarbeit auf eine quellenspezifische Crawling-Pipeline, in die unter anderem die überregionale Plattform Eventfrog bereits vollständig integriert ist [1]. Für die vorliegende Arbeit stellt sich die Frage, wie zusätzliche Quellen erschlossen werden sollen. Geprüft wurden insbesondere die Plattformen Guidle und Mobilizon. Mobilizon weist zum aktuellen Zeitpunkt keine Veranstaltungen im Kanton St.Gallen auf, weshalb eine Anbindung über deren Schnittstelle keinen Mehrwert bietet. Damit verbleibt Guidle als kostenpflichtige Quelle [12], für die eine API-Anbindung überhaupt sinnvoll erscheint.

Entsprechend evaluiert dieser Abschnitt die Nutzung der offiziellen Programmierschnittstelle (API) von Guidle als Alternative zur bestehenden, quellenspezifischen Crawling-Pipeline. Während die API einen direkten, strukturierten Datenzugriff verspricht, schränkt das restriktive Lizenzmodell der Plattform die Nutzbarkeit für Drittsysteme ein. Die wesentlichen Vor- und Nachteile sind in Tabelle 14 gegenübergestellt.

Vorteile (Guidle-API)	Nachteile (Guidle-API)
• Homogene Datenqualität durch konsistente, vorstrukturierte JSON-Datensätze ohne syntaktische Fragmente • Entlastung der Pipeline, da das Überspringen der Extraktionsphase Rechenlast und Token-Kosten der LLM-Pipeline reduziert • Geringer Parsing-Aufwand, da das Normalisieren heterogener HTML-Strukturen und damit verbundene Laufzeitfehler entfallen	• Finanzielle Barriere durch ein kostenpflichtiges Abonnement, im Widerspruch zum Open-Source-Ansatz des Projekts • Architektonischer Bruch durch dedizierte API-Konnektoren parallel zur etablierten Inhouse-Infrastruktur • Abhängigkeit von Spezifikationen, Ausfallzeiten und Tarifänderungen eines einzelnen kommerziellen Anbieters (Vendor Lock-in)

Tabelle 14: Vor- und Nachteile der Guidle-API-Integration gegenüber der bestehenden Inhouse-Pipeline

Zur quantitativen Absicherung werden die API-Anbindung und die Weiterführung der eigenen Crawling-Logik anhand des in Kapitel 3.1 definierten Kriterienkatalogs bewertet; Tabelle 15 stellt das Ergebnis dar. Bei der funktionalen Eignung schneidet die Inhouse-Pipeline besser ab, da die API systembedingt nur Veranstaltungen abdeckt, die über kostenpflichtige Media Services vertrieben werden, während lokale Kleinstvereine fehlen. Bei der Performance liegt die API vorne, weil strukturierte JSON-Payloads die nachgelagerte LLM-Verarbeitung minimieren. Hinsichtlich der Kosten fällt die API aufgrund der Lizenzgebühren von rund CHF 100 pro Monat deutlich ab. Beim Aufwand erzeugt der Umbau der bestehenden, funktionierenden Architektur auf ein neues API-Paradigma einen hohen Zusatzaufwand. Bei der Wartbarkeit schützt die API zwar vor Layout-Änderungen der Quellwebsites, erzeugt jedoch eine externe Abhängigkeit, sodass die Kontrolle über den eigenen Code der Inhouse-Pipeline den Vorteil sichert.

Kriterium	Gew.	Guidle-API	Inhouse-Pipeline
Eignung	30%	3	5
Performance	20%	5	2
Kosten	5%	1	5
Aufwand	20%	2	4
Wartbarkeit	25%	3	4
Nutzwert <i>N</i>		3.1	3.95

Tabelle 15: Nutzwertanalyse Datenbeschaffung: Guidle-API gegenüber quellspezifischem Inhouse-Ansatz

Wie Tabelle 15 zeigt, fällt die Bewertung deutlich zugunsten der Beibehaltung der quellspezifischen Inhouse-Pipeline aus. Den Ausschlag geben die funktionale Eignung, die vollständige Kostenfreiheit sowie der geringere Anpassungsaufwand der bereits vorhandenen Systemarchitektur. Da das primäre Projektziel die lückenlose Erfassung des heterogenen, kommunalen Veranstaltungsmarktes im Kanton St.Gallen ist, wiegen die inhaltliche Unabhängigkeit und die Kostenstruktur der eigenen Pipeline schwerer als die punktuellen Performance-Vorteile einer externen Schnittstelle. Auf die Integration der Guidle-API wird daher verzichtet.

3.7 Schlussfolgerungen

Die vorangegangenen Evaluationen wurden einheitlich mit der in Kapitel 3.1 beschriebenen Nutzwertanalyse durchgeführt. Dadurch beruhen sämtliche Architektur- und Technologieentscheide auf einem konsistenten, quantifizierbaren Verfahren und sind untereinander vergleichbar. Tabelle 16 fasst die getroffenen Entscheide und ihre wesentlichen Gründe zusammen.

Evaluation	Gewählte Option	Entscheidungsgrund
Crawler-Architektur	Generalisierter, konfigurierbarer Crawler	Skalierbarkeit und effiziente Anbindung heterogener Quellen
Datenbeschaffung	Quellenspezifische Inhouse-Pipeline	Inhaltliche Vollständigkeit und Kostenfreiheit; Guidle-API hinter Paywall
Adressauflösung	Lokale swisstopo-Datenbank	Ressourcenschonung und Vermeidung laufender Kosten; Alternativen am Ausschlusskriterium gescheitert
Event-Kategorisierung	Vektorbasierter Ansatz mit pgvector	Entlastung des LLM-Engpasses trotz höheren Integrationsaufwands
Monitoring	Gehosteter Dienst Sentry	Eignung für Exception-Tracking bei geringem Aufwand

Tabelle 16: Übersicht der getroffenen Architektur- und Technologieentscheide mit ihren wesentlichen Begründungen

Bei der Crawler-Architektur überwiegen die langfristige Skalierbarkeit und die effiziente Anbindung heterogener Quellen den höheren initialen Implementierungs- und Testaufwand, weshalb der generalisierte Ansatz gewählt wurde. Aus denselben Erwägungen wird bei der Datenbeschaffung auf die Anbindung der Guidle-API verzichtet: Die inhaltliche Vollständigkeit und die Kostenfreiheit der eigenen Pipeline überwiegen die Performance-Vorteile der kostenpflichtigen Schnittstelle. Bei der Adressauflösung bestätigt die Bewertung die bestehende swisstopo-Lösung; die Alternativen scheiden zusätzlich am Ausschlusskriterium der Betriebskosten und des Ressourcenbedarfs aus, sodass die verbleibende Herausforderung unpräziser Ortsangaben über einen spezialisierten Sondermechanismus statt über einen Systemwechsel adressiert wird.

Die Event-Kategorisierung bildet den am stärksten ausgeprägten Zielkonflikt ab: Die reine LLM-Lösung schneidet beim geringen Integrations- und Wartungsaufwand vorteilhaft ab, während der vektorbasierte Ansatz seine Stärke bei der Verarbeitungsleistung ausspielt. Da das Sprachmodell bereits den Engpass der Pipeline bildet, gibt dieser Aspekt den Ausschlag, weshalb pgvector implementiert wurde. Beim Monitoring erweist sich Sentry sowohl bei der

fachlichen Eignung für das benötigte Exception-Tracking als auch beim geringen Integrations- und Wartungsaufwand als überlegen.

Über alle Evaluationen hinweg zeigt sich, dass die fachliche Eignung der Lösung das ausschlaggebende Kriterium bildet, während Betriebskosten und Ressourcenbedarf in einzelnen Fällen als harte Grenze wirken. Die hier getroffenen Entscheide bilden die Grundlage für das im folgenden Kapitel beschriebene Umsetzungskonzept (Kapitel 4), in dem ihre konkrete technische Realisierung dokumentiert wird.

4 Umsetzungskonzept

Das Umsetzungskonzept überführt die Erkenntnisse aus der Evaluation in eine konkrete Lösungsarchitektur. Es beschreibt die Gesamtarchitektur sowie die Konzepte der einzelnen Systemkomponenten.

4.1 Architektur

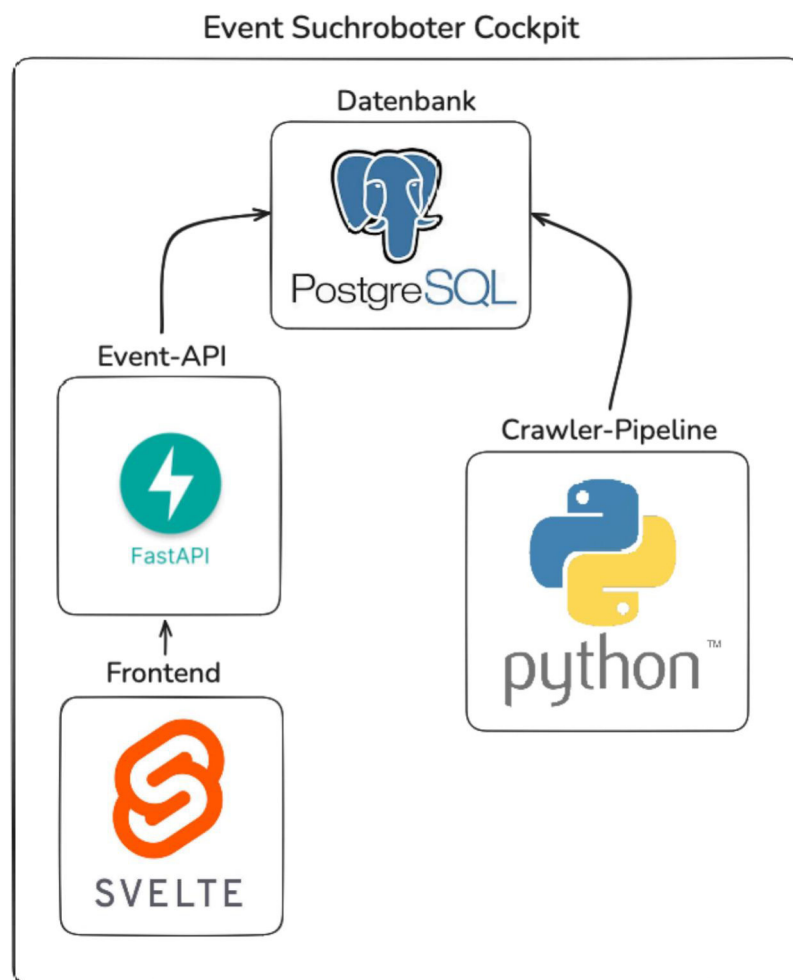


Abbildung 3: Event-Suchroboter-Cockpit Architektur

Die Abbildung 3 gibt einen Überblick über die Architektur des Event-Suchroboters. Diese baut auf der in der Studienarbeit erarbeiteten Grundstruktur auf [1, p. 16] und wird in der vorliegenden Arbeit gezielt weiterentwickelt. Sie ist modular aufgebaut und gliedert sich in vier klar voneinander getrennte Komponenten, die jeweils eine eigene Verantwortlichkeit übernehmen.

- **Crawler-Pipeline:** Umfasst die gesamte Logik für das Crawlen, die Datenverarbeitung und die Persistierung. Sämtliche Skripte, Module und Funktionen zur Ausführung der Pipeline sind hier gebündelt. Kapitel 2.1.1

- **Datenbank:** Sämtliche im Rahmen der Pipeline erzeugten und verarbeiteten Daten werden in einer relationalen PostgreSQL-Datenbank persistiert. Kapitel 4.5
- **API-Service:** Stellt die Schnittstelle zur Abfrage der gespeicherten Events und ihrer Metadaten bereit und vermittelt zwischen Datenbank und Frontend. Kapitel 4.4
- **Frontend:** Bildet die Benutzeroberfläche zur Visualisierung der Events und bezieht die darzustellenden Daten über den API-Service. Kapitel 4.3

Durch diese klare Trennung der Verantwortlichkeiten wird eine Vermischung von Anwendungscode, generierten Daten und Laufzeitinformationen vermieden. Dies ermöglicht reproduzierbare Pipeline-Läufe und eine nachvollziehbare Abbildung aller Verarbeitungsschritte.

4.2 Crawler-Pipeline

Die Crawler-Pipeline bildet die zentrale Verarbeitungsschicht des Event-Suchroboters. Sie koordiniert die Datenerfassung, die Aufbereitung der extrahierten Inhalte, die semantische Analyse von Eventdaten, die geografische sowie kategorielle Anreicherung und die abschließende Persistierung in der Datenbank. In diesem Kapitel wird die Architektur dieser Verarbeitungsschicht beschrieben. Die Konzeption der nachfolgend beschriebenen Pipeline-Stufen sowie der Datei- und Logging-Architektur baut auf den Vorarbeiten der Semesterarbeit auf. [1] [p. 17-19]

Architektonisch ist die Crawler-Pipeline als mehrstufiges, asynchrones Queue-System konzipiert. Anstelle eines monolithischen Blocks operieren die Stufen als parallel laufende Worker-Pools, die ihre Ergebnisse über entkoppelte Warteschlangen an die jeweils nachgelagerte Stufe übergeben. Die fachliche Verarbeitungsreihenfolge bleibt dadurch deterministisch, während die einzelnen Stufen unabhängig voneinander skaliert und ausgelastet werden können. Diese modulare Struktur gewährleistet die Erweiterbarkeit des Gesamtsystems: Neue Datenquellen, spezifische Aufbereitungsstrategien oder zusätzliche Anreicherungsschritte lassen sich ohne strukturelle Anpassungen der bestehenden Pipeline integrieren.

Der konzeptionelle Datenfluss und die Organisation der Pipeline ist in der folgenden Übersicht (Abbildung 4) dargestellt:

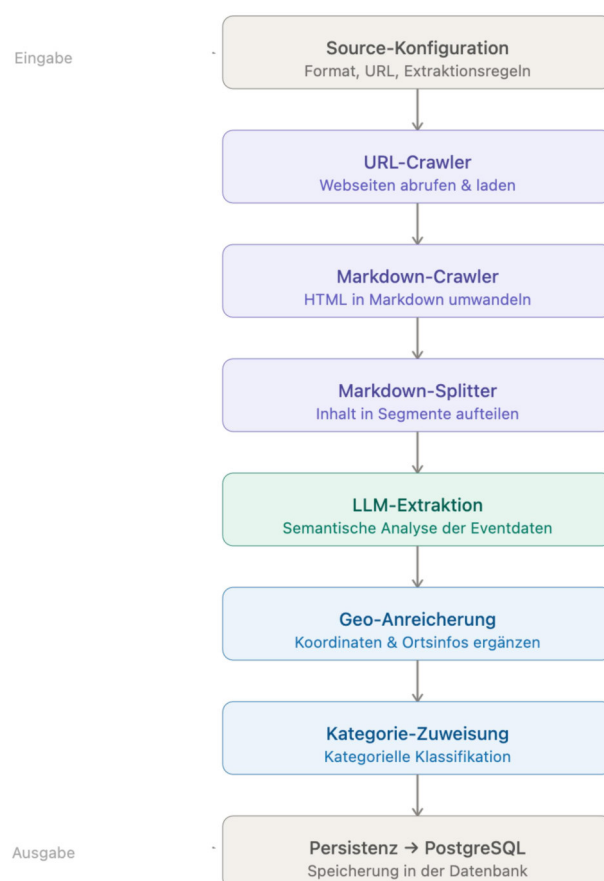


Abbildung 4: Pipeline-Architektur und Datenflüsse

Beigleitende Artefakte die während der Pipeline-Ausführung generiert werden wie Zwischenstände, Logs und Hilfsdaten sind in Tabelle 17 dargestellt. Sie ermöglichen die Nachvollziehbarkeit der Verarbeitungsschritte, die Fehlerdiagnose und die Optimierung der Pipeline-Performance. Sie werden in den Verzeichnissen `data` und `logs` abgelegt, die als zentrale Speicherorte für alle nicht-relationalen Daten und Laufzeitinformationen des Backends dienen.

data	Zwischenstände, Markdown, JSONL, Hilfsdaten
logs	Pipeline-Logs, LLM-Prozesslogs, Analyseberichte

Tabelle 17: Begleitende Pipeline-Artefakte

4.2.1 Konfigurationsgetriebene Quellen

Die Integration externer Eventquellen erfolgt deklarativ über Konfigurationsdateien. Diese Quellkonfigurationen definieren die Parameter für den Einstiegspunkt (Start-URL), die syntaktischen Regeln zur Link-Erkennung, die Navigationseigenschaften der Zielwebseite, quellen-spezifische Metadaten sowie optionale Vorgaben für die Segmentierung der Inhalte. Dieser Ansatz gewährleistet, dass die Kernlogik der Pipeline vollständig von den Spezifikationen einzelner Webseiten entkoppelt ist. Quellenspezifische Besonderheiten werden nicht im Programmiercode abgebildet, sondern in den Konfigurationsdateien gekapselt.

Aufgrund der unterschiedlichen Struktur der Quellsysteme, die eigene Event-Detailseiten, Tabellen, Akkordeon-Elemente und PDF-Dokumente umfasst, ist eine strikte Trennung aus architektonischer Sicht notwendig. Die Konfiguration legt fest, wie jedes Quellformat in den einheitlichen Pipeline-Fluss überführt wird, sodass alle nachgelagerten Komponenten nach demselben Prinzip arbeiten können.

4.2.2 Pipeline-Stufen

Der URL-Crawler stellt den Einstiegspunkt der Pipeline dar. Er identifiziert die relevanten Event-URLs oder übergeordneten Verzeichnisse. In dieser Phase findet noch keine Extraktion fachlicher Event-Attribute statt; die Komponente dient ausschliesslich der Eingrenzung des Verarbeitungsumfangs für die nachfolgenden Stufen.

Der Markdown-Crawler lädt die identifizierten Webseiten und transformiert den extrahierten Kerninhalt in das Markdown-Format. Markdown fungiert hierbei als primäres, normalisiertes Zwischenformat, welches die strukturellen Unterschiede der Quellmedien nivelliert. In dieser Stufe werden auch binäre Inhalte wie PDF-Dokumente in Text überführt und konsolidiert.

Der Markdown-Splitter segmentiert Markdown-Dateien, die Informationen zu mehreren Veranstaltungen beinhalten (beispielsweise Event-Tabellen oder aggregierte PDF-Terminkalender). Die Komponente erzeugt isolierte Markdown-Artefakte pro Event oder Sinneinheit. Sofern in der Quellkonfiguration hinterlegt, greift die Stufe auf spezialisierte Segmentierungsregeln zurück; andernfalls kommt eine generische Fallback-Logik zur Anwendung.

Die LLM-Extraktion überführt die unstrukturierten oder semistrukturierten Markdown-Texte in stark strukturierte Event-Objekte. Das Sprachmodell analysiert den Text semantisch und extrahiert die Kernattribute wie Titel, Datums- und Zeitangaben, Veranstaltungsort, Beschreibung, Verweis zur Quelle sowie den aktuellen Status des Events. Im Gegensatz zu klassischen, rigiden Parsing-Verfahren erlaubt dieser Ansatz die sinngemässe Interpretation des Textkör-

pers. Der Ablauf der Vorverarbeitung und des Prompt-Aufbaus ist in Abbildung 5 dargestellt; der LLM-Aufruf samt deterministischer Guardrails in Abbildung 6.

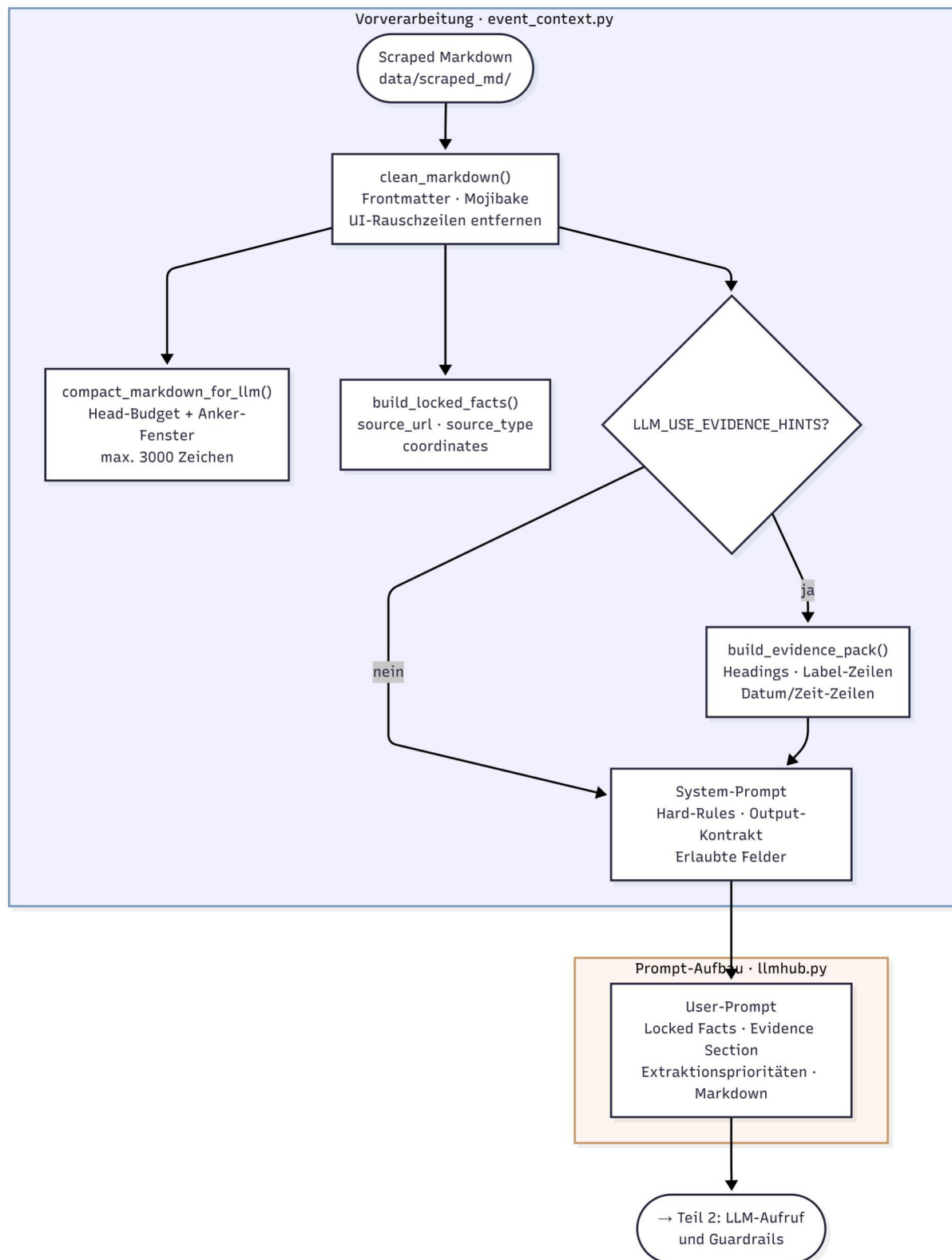


Abbildung 5: LLM-Extraktion Teil 1: Markdown-Vorverarbeitung und Prompt-Aufbau.

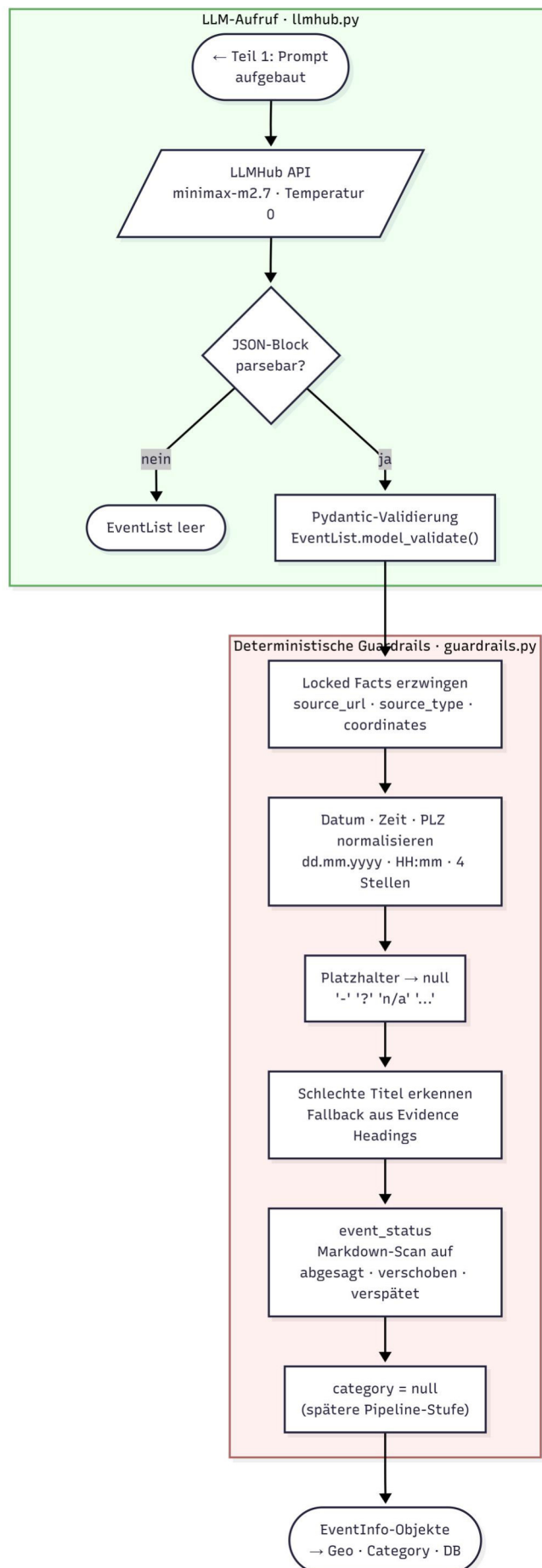


Abbildung 6: LLM-Extraktion Teil 2: LLM-Aufruf, JSON-Parsing und deterministische Guardrails.

Die Geo-Anreicherung ist als eigenständige Komponente realisiert. Sie validiert die extrahierten Ortsangaben, reichert diese durch Geokodierung mit Koordinaten an oder leitet komplementär Adressdaten aus vorhandenen Geokoordinaten ab.

Die Kategorie-Zuweisung klassifiziert die Veranstaltungen anhand eines vordefinierten fachlichen Kategoriensystems. Die funktionale Trennung zwischen der reinen Datenextraktion und der semantischen Klassifizierung ermöglicht eine isolierte Weiterentwicklung der Klassifikationslogik, ohne die vorgelagerten Extraktionsprozesse zu beeinflussen.

Die Persistenzschicht bildet den finalen Schritt der Pipeline. An diesem Punkt erfolgt die logische Konsolidierung der Daten: Das System gleicht die angereicherten Event-Objekte mit dem Datenbestand ab und entscheidet über das Einfügen neuer Datensätze, die Identifikation von Duplikaten, die Zusammenführung (Merge) komplementärer Informationen oder die Aktualisierung bestehender Entitäten. Diese späte Entscheidungsfindung stellt sicher, dass die Deduplizierung und Qualitätsprüfung auf Basis vollständig angereicherter Daten erfolgt. Abbildung 7 zeigt die Vorbereitungsphase des Entscheidungsablaufs; Abbildung 8 den Dedup-Entscheid und die abschliessende Datenbankpersistenz.

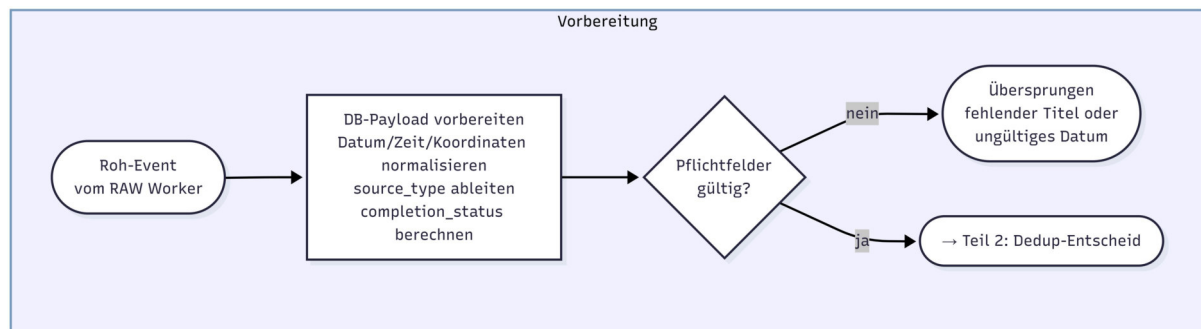


Abbildung 7: Persistenzschicht Teil 1: Payload-Vorbereitung und Pflichtfeldprüfung.

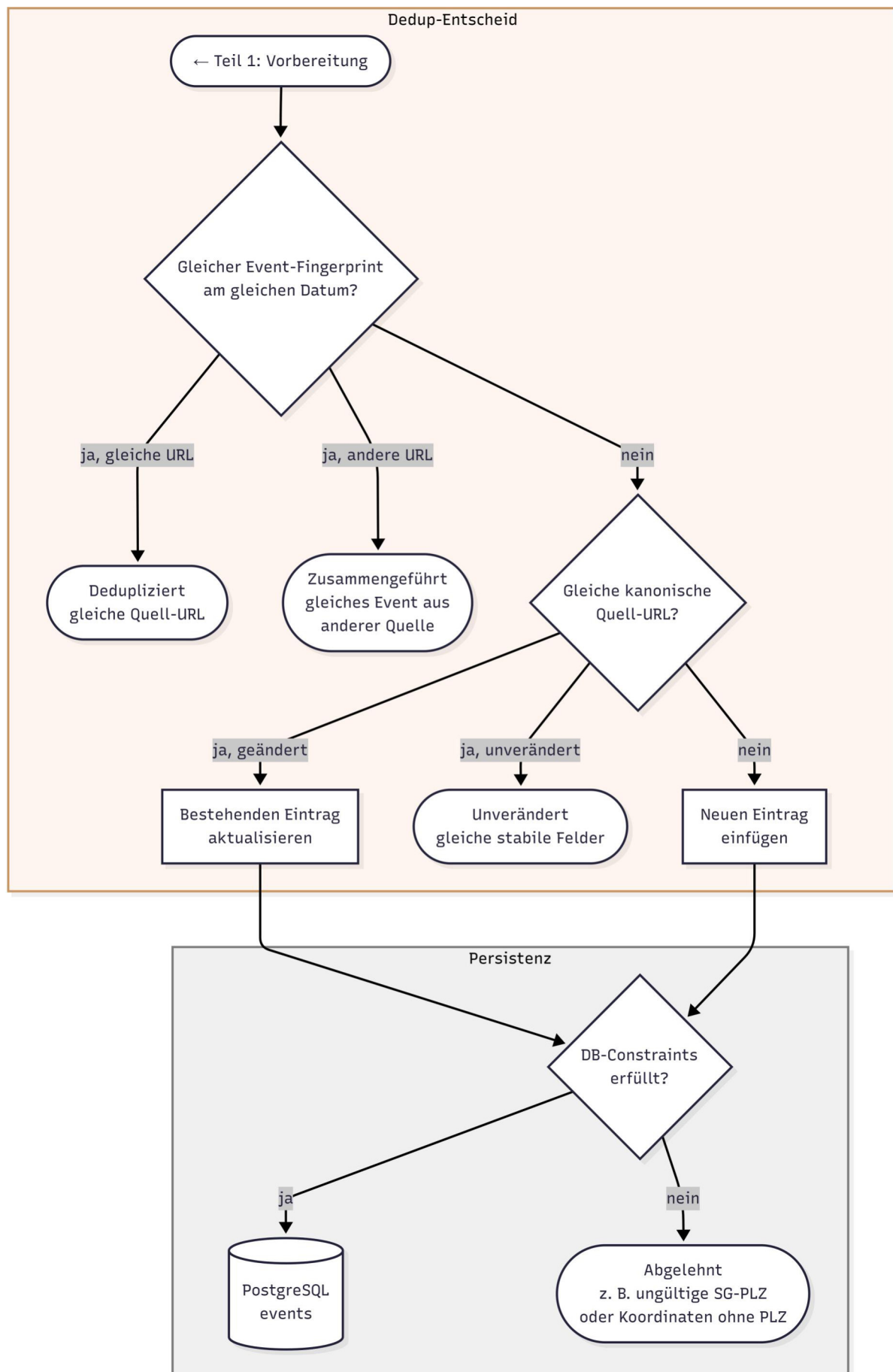


Abbildung 8: Persistenzschicht Teil 2: Dedup-Entscheid, Merge, Update und DB-Constraints.

4.2.3 Data Verzeichnis

Das `data`-Verzeichnis fungiert als primärer Datenbereich des Backends für vorübergehende und teilweise anhaltende Artefakte. Es beinhaltet keine ausführbaren Code-Elemente, sondern die während des Pipeline-Durchlaufs generierten physischen Dateien. Hierzu zählen extrahierte URL-Listen, normalisierte Markdown-Dokumente, JSONL-Zwischenstände der einzelnen Pipeline-Stufen sowie statistische Hilfsdaten für das Geocoding und die Systemanalyse. Aus architektonischer Sicht erfüllt das `data`-Verzeichnis drei Kernfunktionen:

- **Entkopplung:** Es dient als transienter Speicherplatz, wodurch die einzelnen Pipeline-Stufen entkoppelt und isoliert analysiert werden können.
- **Reproduzierbarkeit:** Die physische Speicherung der Zwischenstände erlaubt eine nachträgliche Auditierung und Fehleranalyse der Crawling- und Extraktionsergebnisse.
- **Daten-Separation:** Es etabliert eine klare Trennung zwischen unstrukturierten Rohdaten, normalisierten Zwischenformaten und dem finalen, relationalen Zustand in der Datenbank.

Diese dateibasierte Datenarchitektur ergänzt die relationale PostgreSQL-Datenbank. Während die Datenbank den konsolidierten, operativen Zielzustand repräsentiert, macht das Verzeichnis `data` den historischen Entstehungsprozess transparent. Es ermöglicht die Nachvollziehbarkeit der Datenherkunft, die Analyse von Fehlern und die Optimierung der Pipeline-Performance auf Basis konkreter Artefakte.

4.2.4 Logging-Architektur

Das Verzeichnis `logs` bildet die Grundlage für das Laufzeit-Monitoring und die Systemdiagnose des Backends. Es akkumuliert sämtliche Protokolle der Pipeline-Ausführung und dient der Identifikation von Laufzeitfehlern. Das Logging unterteilt sich in verschiedene Granularitätsstufen, um eine differenzierte Beobachtbarkeit zu gewährleisten:

- **Allgemeine Pipeline-Logs:** Dokumentieren den Lebenszyklus der Worker-Komponenten, den Fortschritt des Crawling-Prozesses, Fehlermeldungen, Systemwarnungen sowie den Zustand der internen Warteschlangen (Queues).
- **LLM-Prozess-Logs:** Protokollieren die spezifischen Interaktionen mit dem Sprachmodell. Hierzu gehören Zuordnungen verarbeiteter Dateien, Modifikationen einzelner Datenfelder sowie die Ergebnisse nachgeschalteter Guardrail-Überprüfungen.
- **LLM-Payload-Logs:** Erlauben bei Aktivierung die detaillierte Aufzeichnung der an die API des Sprachmodells übermittelten Datenstrukturen (Prompts und Kontext). Dies dient der Optimierung der Prompt-Struktur und der Analyse von Kontextüberschreitungen.
- **Pipeline-Analyse-Logs:** Bieten aggregierte Analysen eines Gesamtlaufs. Diese Berichte verdichten komplexe Logströme auf die wesentlichen Ereignisse wie kritische Fehler, Deduplizierungsentscheidungen und Daten-Merges.

Ergänzend zur Protokollierung wird **Sentry** als zentraler Exception- und Performance-Tracking-Layer eingesetzt. Während die lokalen Logs eine vollständige, zeitlich geordnete Aufzeichnung der Verarbeitung im Verzeichnis `logs` bereitstellen, aggregiert Sentry asynchron auftretende Ausnahmen aus allen Worker-Stufen, gruppiert wiederkehrende Fehler und verknüpft sie mit Kontextinformationen wie der jeweiligen Quelle oder der LLM-Transaktion. Beide Ebenen ergänzen sich: Die dateibasierten Logs dienen der detaillierten Nachvollziehbarkeit

eines konkreten Pipeline-Laufs, Sentry der quellenübergreifenden Beobachtung wiederkehrender Fehlerbilder über mehrere Läufe hinweg.

4.2.5 Abgrenzung zur API-Schicht

Die Crawler-Pipeline und die API-Schicht sind klar voneinander getrennt. Die Pipeline schreibt und transformiert die Daten in die PostgreSQL-Datenbank. Der API-Dienst stellt diese Daten anschliessend dem Frontend zur Verfügung. Das Frontend hat keinen direkten Zugriff auf die Dateisysteme (data, logs) oder die Worker. Diese Trennung hält die Kopplung gering und erlaubt es, Datenerfassung und Datenbereitstellung unabhängig zu betreiben und zu skalieren.

4.3 Frontend

Das Frontend soll als Single-Page Application mit SvelteKit (Tabelle 18) umgesetzt werden und einer komponentenbasierten Architektur folgen. Die Anwendung ist als einzelne Ansicht konzipiert, in der Veranstaltungsdaten über die Event-API abgerufen und dargestellt werden.

Technologie-Stack:

Komponente	Technologie
Framework	SvelteKit (Svelte 5, Runes-Modus)
Sprache	TypeScript
Karte	Leaflet + leaflet.markercluster
Slider-Komponenten	bits-ui (Slider-Komponente)
Build-Tool	Vite (via SvelteKit)
Node-Adapter	sveltejs/adapters-node
Package Manager	npm
Deployment	Docker Container

Tabelle 18: Frontend Technologie-Stack

Folgende Features (Tabelle 19) wurden für die überarbeitete Version des Frontends definiert:

Feature	Beschreibung	Priorität
Filterfunktion	Filterung der angezeigten Events nach Kriterien wie Datum, Zeitintervall, Kategorie oder Source Type.	1
UI-Verbesserungen	Optimierung der Benutzeroberfläche für eine bessere Übersicht, insbesondere bei der Darstellung von Event-Details und der Kartenansicht. Events sollen mit Labels versehen werden.	2
Kategorie-Icons	Visuelle Kennzeichnung der Event-Kategorien in der Kartenansicht durch Icons.	3
Cluster-Visualisierung	Clustering räumlich naher Events in der Kartenansicht für eine bessere Übersicht bei hoher Datendichte.	4
Suchfunktion	Suche nach Events anhand von Stichwörtern in Titel oder Beschreibung.	5
Exportfunktion	Export der angezeigten Events als CSV oder JSON.	6

Tabelle 19: Frontend Feature

Feature-Priorisierung: Die Priorisierung erfolgt nach Nutzerbedürfnissen und technischer Komplexität. Die Filterfunktion ist dabei von entscheidender Bedeutung, da sie die grundlegendste Interaktion mit den Daten ermöglicht. UI-Verbesserungen folgen direkt, da sie die Übersichtlichkeit erhöhen und unmittelbar zur Nutzererfahrung beitragen. Die Implementierung von Kategorie-Icons und Cluster-Visualisierungen ist sinnvoll, erfordert jedoch zusätzlichen Aufwand und wird daher nachgelagert umgesetzt. Die Suchfunktion erhöht zwar die Flexibilität, steigert jedoch auch die Komplexität. Aus diesem Grund wird sie erst gegen Ende priorisiert. Die Exportfunktion stellt den Abschluss der Liste dar, da ein Datenexport bereits über die API möglich ist und die Frontend-Variante lediglich einen zusätzlichen Komfort bietet.

Um agil auf sich ändernde Anforderungen reagieren zu können, wird die Umsetzung in iterativen Sprints erfolgen. In jedem Sprint wird ein oder mehrere Features implementiert, getestet und in enger Abstimmung mit der Kantonspolizei St. Gallen validiert, bevor zum nächsten Feature übergegangen wird. Darum kann es zu Anpassungen in der Priorisierung kommen, wenn sich die Bedürfnisse der Nutzer oder die technischen Rahmenbedingungen ändern.

4.4 API

Um eine klare Separation of Concerns zu gewährleisten, wird die API als eigenständiger Dienst konzipiert. Crawler und API lassen sich dadurch unabhängig voneinander betreiben, und das Frontend bezieht die Eventdaten jederzeit ohne direkte Kopplung an die Crawler-Pipeline.

Über die Versorgung des Cockpit-Frontends hinaus dient die API als offene Schnittstelle für weitere Dienste der Kantonspolizei St. Gallen. Diese können die aggregierten Veranstaltungsdaten direkt beziehen und in eigenen Anwendungen mit eigener Filter- und Integrationslogik weiterverarbeiten, ohne den Umweg über das Cockpit-Frontend. Die Kommunikation erfolgt über RESTful Endpunkte mit JSON als Datenformat. Tabelle 20 fasst die eingesetzten Komponenten zusammen.

Komponente	Technologie
Framework	FastAPI
Datenbankzugriff	psycopg v3.0 , psycopg-pool
Dependency Manager	uv
Deployment	Docker-Container mit Image python:3.12-slim

Tabelle 20: Technologie-Stack der API.

Die API-Dokumentation wird automatisch über Swagger UI bereitgestellt. Entwickler, auch auf Seiten der Kantonspolizei, können die verfügbaren Endpunkte direkt einsehen, testen und in eigene Systeme integrieren, ohne vorgängig Rücksprache halten zu müssen.

Tabelle 21 fasst die vorgesehenen Endpunkte und ihre Zielgruppen zusammen.

Endpoint	Beschreibung	Zielgruppe
GET /events/	Gefilterter Abruf von Veranstaltungen. Hauptendpunkt des Cockpits: kombiniert Zeitraumfilterung mit optionalen Zusatzfiltern (Stadt, Kategorie, Status, Source Type) sowie einer Volltextsuche.	Frontend
GET /events/timespan/	Abruf von Veranstaltungen ausschliesslich nach Zeitraum, ohne weitere Filteroptionen.	Extern
GET /events/search	Dedizierte Volltextsuche über Veranstaltungstitel.	Extern
GET /events/status/{status}	Liefert alle Veranstaltungen mit einem bestimmten Status.	Extern
GET /events/city/{city_name}	Liefert alle Veranstaltungen für eine bestimmte Stadt. Gross-/Kleinschreibung wird ignoriert.	Extern
GET /events/category/{category}	Liefert alle Veranstaltungen einer bestimmten Kategorie. Gross-/Kleinschreibung wird ignoriert.	Extern

Tabelle 21: Übersicht der vorgesehenen API-Endpunkte.

4.5 Datenbankkonzept

Die persistente Datenhaltung des Systems basiert auf PostgreSQL und gliedert sich in drei thematisch klar getrennte Tabellen.

4.5.1 Event-Tabelle§

Die zentrale Tabelle **events** (Tabelle 22) nimmt alle gesammelten und verarbeiteten Veranstaltungen auf. Gegenüber der SA wurde das Schema gezielt erweitert [1, p. 20], um strukturierte Adress- und Metadatenfelder sowie zusätzliche Statusinformationen aufzunehmen. Neben inhaltlichen Attributen wie Titel, Beschreibung, Datum und Kategorie werden strukturelle Metadaten gespeichert darunter Quellenart, Veranstaltungsstatus sowie geografische Angaben.

Spalte	Typ	Beschreibung
id	SERIAL PRIMARY KEY	Interne Event-ID
title	TEXT NOT NULL	Eventtitel
date	DATE NOT NULL	Eventdatum
full_location	TEXT	Vollständige Adresse als String
country	TEXT	Land
country_short_id	VARCHAR(2)	ISO-Ländercode (z. B. CH)
city_name	TEXT	Gemeinde-/Stadtname (Basis für Geo-Lookup)
city_zip	NUMERIC(4,0)	PLZ – durch Constraint auf den Kanton SG eingeschränkt
street	TEXT	Strassenname
street_number	NUMERIC	Hausnummer
start_time	TIME	Startzeit
end_time	TIME	Endzeit
source_url	TEXT	Ursprüngliche Quell-URL
source_type	TEXT	Quellenart, z. B. eventfrog, eventseiten
coordinates	TEXT	Koordinaten als "lat, lon"-String
category	TEXT	Schema.org-Kategorie, z. B. MusicEvent
description	TEXT	Beschreibungstext
event_status	VARCHAR(20) NOT NULL DEFAULT 'scheduled'	Veranstaltungsstatus, z. B. scheduled, cancelled, postponed
completion_status	VARCHAR(50) NOT NULL DEFAULT 'complete_event'	Datenvollständigkeits-Status, z. B. complete_event, partial_event
updated_on	TIMESTAMP NOT NULL DEFAULT now()	Zeitpunkt der letzten Aktualisierung

Tabelle 22: Schema der Tabelle `events`.

4.5.2 Kategorie-Tabelle

Die Tabelle **categories** (Tabelle 23) dient als Referenztable für die semantische Kategorisierung.

Spalte	Typ	Beschreibung
id	SERIAL PRIMARY KEY	Interne Kategorie-ID
name	VARCHAR(255) NOT NULL UNIQUE	Kategorienname
description	TEXT	Beschreibung, Basis für die Embedding-Generierung
embedding	vector(1024)	Vektor-Embedding der Kategoriebeschreibung

Tabelle 23: Schema der Tabelle `categories`.

4.5.3 Swiss-Addresses-Tabelle

Die Tabelle **swiss_addresses** dient als lokale Referenz für die Geokodierung extrahierter Veranstaltungen. Sie basiert auf dem amtlichen Gebäudeadressverzeichnis von swisstopo [11] und umfasst rund 3.5 Millionen Datensätze mit Strassennamen, Hausnummern, Postleitzahlen, Gemeinden sowie Koordinaten in den Bezugssystemen LV95 und WGS84. Aufbau und Verwendung folgen der technischen Dokumentation von swisstopo [13].

Die Entscheidung für eine lokal vorgehaltene Adressdatenbank wurde bereits in der Studienarbeit getroffen [1, p. 20] und gewährleistet eine datenschutzfreundliche, reproduzierbare und performante Geokodierung ohne Abhängigkeit von externen APIs. Im Rahmen dieser Arbeit wurde diese Wahl gegen externe Geokodierungsdienste sowie umfangreichere lokale Alternativen wie OpenStreetMap geprüft und bestätigt (Kapitel 3.3). Während Datenquelle und Schema gegenüber der Studienarbeit unverändert sind, wird die Tabelle in der vorliegenden Arbeit über eine erweiterte Lookup-Strategie genutzt: Sie validiert die vom Sprachmodell gelieferten Adressfelder gezielt gegen Halluzinationen, arbeitet bidirektional und erlaubt eine konfigurierte Fallback-Adresse (Kapitel 7.6).

5 Resultate

Dieses Kapitel fasst die erzielten Resultate der Arbeit zusammen und vergleicht sie mit den ursprünglich definierten Zielen.

5.1 Ergebnisse

Die Leitfrage der Arbeit – ob sich im Web veröffentlichte Veranstaltungen im Kanton St.Gallen so automatisiert erfassen, kategorisieren und visualisieren lassen, dass zeitlich-räumliche Koinzidenzen erkennbar werden – kann auf Basis der erzielten Resultate bejaht werden. Der in der Studienarbeit [1] entwickelte Prototyp wurde in einen umfassenden Proof-of-Concept überführt, der die ursprünglichen Limitationen gezielt adressiert und das System an mehreren zentralen Stellen substanziell weiterentwickelt.

Die wesentliche Neuerung gegenüber der Vorarbeit liegt in der konfigurationsgetriebenen, asynchron orchestrierten Multi-Stage-Pipeline (Kapitel 4.2.2). An die Stelle einer prototypischen, weitgehend monolithischen Verarbeitungsstrecke tritt ein modulares Queue-System mit klar getrennten Stufen für URL-Crawling, Markdown-Normalisierung, Segmentierung, LLM-Extraktion, Geo-Anreicherung, Kategorisierung und Persistenz. Neue Quellen lassen sich deklarativ über Konfigurationsdateien anbinden, ohne dass der Kerncode angepasst werden muss (Kapitel 4.2.1). Diese Entkopplung wurde im Verlauf des Projekts auf über 80 produktive Quellen aus dem Kanton St.Gallen skaliert, darunter Gemeinde- und Vereinswebseiten sowie die kantonal relevanten Eventplattformen Eventfrog und Guidle.

Darüber hinaus wurden mehrere in der Studienarbeit identifizierte Lücken gezielt geschlossen. Die Kategorisierung ist als eigenständige, vektorbasierte Pipeline-Stufe mit pgvector realisiert und vom LLM entkoppelt, womit sich Klassifikationslogik und Extraktion unabhängig weiterentwickeln lassen (Kapitel 3.4). Die Geo-Anreicherung validiert Ortsangaben deterministisch gegenüber einem kuratierten Schweizer Adressdatensatz, statt auf externe Dienste angewiesen zu sein. Die Persistenzschicht entscheidet auf Basis vollständig angereicherter Daten über Insert, Merge oder Update und erfasst dadurch auch Verschiebungen und Absagen über wiederholte Pipeline-Läufe. Ergänzend gewährleisten eine mehrstufige Logging-Architektur (Kapitel 4.2.4) und die Sentry-Integration (Kapitel 3.5) eine durchgängige Nachvollziehbarkeit und quellenübergreifende Fehleraggregation.

Im Frontend wurde das Cockpit um eine interaktive Karte mit Marker-Clustering, kombinierte Filter nach Zeit, Region, Kategorie und Quelle sowie um eine Volltextsuche und eine Exportfunktion erweitert (Abbildung 16). Damit ist die zentrale fachliche Anforderung erfüllt: Für jeden frei gewählten Zeitraum und jede Region werden zeitlich-räumliche Koinzidenzen direkt sichtbar – die Grundlage für die vorausschauende polizeiliche Lagebeurteilung, von der die ursprüngliche Beauftragung der Kantonspolizei St.Gallen ausging.

5.1.1 Offene Punkte und Limitationen

Trotz dieser Resultate sind drei Limitationen offen geblieben, die ehrlich zu benennen sind.

Die **Erschliessung von Social-Media- und Forenquellen** konnte nur als Proof-of-Concept umgesetzt werden, wie in Kapitel 7.12 im Detail beschrieben. Die Gründe unterscheiden sich je nach Quellenklasse: Das FCSG-Forum liess sich technisch einwandfrei crawlen, doch die fehlende Inhaltsstruktur der Beiträge verhinderte eine zuverlässige Extraktion – selbst ein klar erkennbarer Event scheiterte an einer nicht ableitbaren Datumsangabe. Facebook scheiterte an der Login-Wall und an serverseitig obfusziertem HTML. Mobilizon funktioniert technisch vollständig, liefert jedoch auf keiner der sechs Schweizer Instanzen Veranstaltungen mit Bezug zum Kanton St. Gallen. Diese Quellenklasse bleibt damit ausserhalb des aktuellen Datenkanons, obwohl die technische Basis für eine spätere Integration vorhanden ist.

Die **Latenz der LLM-Aufrufe** ist im Endausbau der dominierende Performance-Faktor der Pipeline. Da die semantische Extraktion jedes Event-Kandidaten einen Modellaufruf erfordert, skaliert der Gesamtdurchsatz primär entlang der Modell-Latenz und der parallel zugelassenen Anfragen. Für den aktuellen Anwendungsfall mit periodischer, nicht zeitkritischer Aktualisierung ist dies tragbar; eine echtzeitnahe Verarbeitung würde architektonische Anpassungen erfordern. Das Laufzeitverhalten ist nicht Teil der automatisierten Testbasis (`backend/tests/`) und wurde ausschliesslich empirisch durch Pipeline-Läufe beobachtet.

Die **automatisierte Kategorisierung** erreicht insgesamt eine brauchbare Trefferquote, ist bei mehrdeutigen oder sehr generischen Eventbeschreibungen aber nicht in jedem Fall passend. Der LLM-Evaluationstest (Kapitel 8.1.2) quantifiziert dies: Mit einer Feldgenauigkeit von 42.9% ist `category` der schwächste extrahierte Wert – deutlich hinter allen anderen Feldern (Tabelle 45). Die getrennte Pipeline-Stufe erlaubt zwar eine isolierte Weiterentwicklung der Klassifikationslogik, das Kategoriensystem selbst und die zugrunde liegenden Embedding-Repräsentationen lassen jedoch Raum für gezielte Verfeinerung.

Ergänzend sei angemerkt: Die strukturierten Plattformen Eventfrog und Guidle wurden über das generische Crawling-Verfahren erschlossen und nicht über ihre offiziellen Schnittstellen angebunden. Funktional liefert dies vergleichbare Ergebnisse, verzichtet aber auf die Vorteile (Stabilität, Datenqualität, Rate-Limits) einer strukturierten API-Integration.

5.2 Zielerreichung

Gegenüberstellung der definierten Ziele (siehe Kapitel 1.4) mit den tatsächlich erreichten Resultaten. Bewertung, inwieweit die Ziele erreicht wurden, und Reflexion über eventuelle Abweichungen oder unerwartete Erkenntnisse.

5.2.1 Harvester für Gemeindeseiten

Ziel: Entwicklung eines Crawlers zur Extraktion von Ereignisdaten aus HTML-Strukturen und PDF-Dokumenten. Mittels KI-gestützter Analyse werden relevante Attribute (Titel, Zeitstempel, Ort, Kategorie) identifiziert und in einer PostgreSQL-Datenbank persistiert.

✓ Vollständig erreicht

Begründung: Die Pipeline umfasst einen URL- und Markdown-Crawler, der HTML-Seiten und PDF-Dokumente in ein einheitliches Zwischenformat überführt (Kapitel 4.2.2). Die nachgelagerte LLM-Extraktion identifiziert Titel, Zeitstempel, Ort und Beschreibung. Die finale Persistierung erfolgt in der PostgreSQL-Datenbank (Kapitel 4.5).

5.2.2 Integration von Event-Quellen

Ziel: Anbindung strukturierter Schnittstellen, namentlich Eventfrog und Guide. Optional wird die Integration dezentraler Plattformen (z. B. Mobilizon) geprüft.

✓ Vollständig erreicht

Begründung: Durch die konfigurationsgetriebene Crawler-Architektur lassen sich neue Quellen deklarativ und ohne Anpassung des Kerncodes anbinden (Kapitel 4.2.1).

5.2.3 Erschliessung von Social-Media- und Foren-Daten

Ziel: Evaluation und Anbindung offener Plattformen zur Prüfung ihrer Eignung als Ereignisquellen unter Berücksichtigung technischer Restriktionen.

✓ Vollständig erreicht

Begründung: FCSG-Forum und FCSG-Facebook wurden evaluiert und als Proof-of-Concept angebunden. Beide Quellen konnten technisch gecrawlt werden, liefern jedoch keinen verwertbaren Mehrwert für die Erfassung von Events im Kanton St. Gallen. Die detaillierte Evaluation ist in Kapitel 7.12 beschrieben.

5.2.4 Dynamisches Aktualisierungsmanagement

Ziel: Implementierung von Mechanismen zur Erkennung von Mutationen an bereits erfassten Events, einschliesslich zeitlicher Verschiebungen oder Absagen.

✓ Vollständig erreicht

Begründung: Die Persistenzschicht gleicht angereicherte Event-Objekte mit dem Bestand ab und entscheidet über Insert, Merge oder Update bestehender Datensätze (Kapitel 4.2.2). Dadurch werden Verschiebungen und Statusänderungen wie Absagen beim wiederholten Pipeline-Lauf erfasst.

5.2.5 Datenpipeline-Orchestrierung

Ziel: Aufbau einer robusten Pipeline zur Bereinigung, Validierung und Deduplizierung der Datenströme, ergänzt durch ein systemweites Logging und Monitoring.

✓ Vollständig erreicht

Begründung: Die mehrstufige Pipeline trennt Crawling, Aufbereitung, Anreicherung und Persistierung in dedizierte Komponenten mit eigenen Validierungs- und Deduplizierungslogiken (Kapitel 4.2.2). Eine mehrstufige Logging-Architektur (Kapitel 4.2.4) sowie die Sentry-Integration (Kapitel 3.5) gewährleisten ein durchgängiges Monitoring.

5.2.6 Automatisierte Kategorisierung

Ziel: Klassifizierung der Ereignisse basierend auf Standards wie schema.org oder tourismdata.ch. Bei fehlenden Metadaten erfolgt eine KI-basierte Zuordnung, deren Präzision anhand definierter Kriterien evaluiert wird.

✓ Vollständig erreicht

Begründung: Die Kategorisierung wurde als eigenständige Pipeline-Stufe realisiert und nutzt eine vektorbasierte Ähnlichkeitssuche mit pgvector, die Events gegen ein vordefiniertes Kategoriensystem abgleicht (Kapitel 3.4). Diese Entkopplung von der reinen Extraktion erlaubt eine isolierte Weiterentwicklung der Klassifikationslogik und entlastet das LLM.

5.2.7 Optimierung des Frontends

Ziel: Weiterentwicklung der Benutzeroberfläche hinsichtlich Usability und Informationsdichte, insbesondere durch eine interaktive Web-Karte sowie effiziente Such- und Filterfunktionen.

✓ Vollständig erreicht

Begründung: Das Cockpit verfügt über eine interaktive Karte, auf der die Events räumlich verortet und kategorisiert dargestellt werden (Abbildung 16). Für die zielgerichtete Exploration stehen Filter nach Zeitraum, Uhrzeit, Kategorie und Quelle sowie eine Volltextsuche zur Verfügung (Abbildung 18). Informationsdichte und Usability wurden durch das Zusammenspiel aus Karte, kompakter Filtersidebar und Suchfunktion erreicht. Ergänzend wurde eine Exportfunktion bereitgestellt, die über das ursprüngliche Ziel hinausgeht.

5.3 Schlussfolgerungen und Ausblick

Aus den offenen Punkten und den im Projekt gewonnenen Erkenntnissen ergeben sich mehrere Stossrichtungen für eine Weiterentwicklung.

Funktional kann das System durch zusätzliche Datenquellen erweitert werden. Neben weiteren Eventplattformen bietet sich insbesondere die Anbindung offizieller Schnittstellen von Eventfrog oder Guidle an. Ebenso erscheint eine geografische Ausweitung über den Kanton St.Gallen hinaus möglich.

Weiteres Potenzial besteht bei der automatisierten Analyse der erfassten Veranstaltungen. Aufbauend auf den vorhandenen Eventdaten könnte das System künftig um Verfahren zur automatischen Risikoeinstufung oder Gefährdungsbeurteilung erweitert werden. Dadurch liesse sich der manuelle Prüfaufwand reduzieren und sicherheitsrelevante Veranstaltungen könnten frühzeitig identifiziert werden.

Auch die Nutzung durch weitere Bedarfsträger erscheint denkbar. Neben der Kantonspolizei könnten Gemeinden, Tourismusorganisationen, Veranstalter sowie Gesundheits- und Rettungsdienste von einem konsolidierten und aktuellen Eventdatenbestand profitieren.

Aus technischer Sicht stehen drei Weiterentwicklungen im Vordergrund: Erstens sollte die Deduplizierung um eine explizite Quellenverwaltung erweitert werden, damit alle Quellen eines konsolidierten Events dauerhaft am Event selbst nachvollziehbar bleiben. Zweitens wäre ein eigener Gold-Datensatz für die Deduplizierung sinnvoll, um den Anteil korrekt zusammengeführter und korrekt getrennt gehaltener Events quantitativ zu messen. Drittens sollte die automatische Kategorisierung verbessert und auf einem grösseren, ausgewogeneren Testdatensatz validiert werden.

Auf nicht-funktionaler Ebene stehen insbesondere Skalierbarkeit, Performance und Betriebsstabilität im Vordergrund. Dazu gehören die Reduktion der LLM-bedingten Verarbeitungszeiten, die weitere Verbesserung der Datenqualität sowie die Vorbereitung des Systems auf einen langfristigen produktiven Betrieb.

Darüber hinaus stellt sich die Frage, unter welchen technischen, organisatorischen und rechtlichen Rahmenbedingungen das entwickelte Proof-of-Concept in ein produktiv betriebenes Lage- und Informationscockpit überführt werden kann. Eine weiterführende Machbarkeitsstudie könnte diese Fragestellung systematisch untersuchen und die Grundlage für eine spätere Produktivsetzung schaffen.

Teil II – SW-Projektdokumentation

Dieser Teil dokumentiert den Engineering-Lebenszyklus und die technische Umsetzung des Systems. Ausgehend von der Projektidee werden zunächst Anforderungen, Use Cases und eine Domänenanalyse erarbeitet. Die Systemarchitektur wird anhand dokumentierter Designentscheide, Komponentenverantwortlichkeiten und Strukturdiagrammen festgehalten. Im Implementationskapitel werden die einzelnen Komponenten der Verarbeitungspipeline detailliert beschrieben. Das Qualitätssicherungskapitel umfasst automatisierte und manuelle Teststrategien sowie CI/CD. Das Projektmanagement dokumentiert Planung, Vorgehen, Risiken und Zeitmanagement. Den Abschluss bildet eine umfassende Deployment- und Benutzerdokumentation.

6 Anforderungsspezifikation

Dieses Kapitel definiert die funktionalen und nicht-funktionalen Anforderungen an den Proof-of-Concept. Die Validierung der NFRs erfolgt in Kapitel 8.3.

6.1 Anforderungen

Die Anforderungen an das System leiten sich direkt aus der Aufgabenstellung und den identifizierten Herausforderungen ab. Um diese anwenderzentriert zu formulieren, wurden zunächst repräsentative Nutzerprofile (Personas) erstellt.

6.1.1 Personas

Zur Strukturierung der Systemanforderungen und zur Sicherstellung einer hohen Usability wurden drei Personas definiert, welche die primären Nutzergruppen des Event-Suchroboter-Cockpits repräsentieren:

- **Die Lageanalystin (Repräsentativ: Sandra Brunner, 38):** Diese Persona vertritt die Endanwender im Lage- und Nachrichtenzentrum. Der Fokus dieser Nutzergruppe liegt auf der kontinuierlichen Überwachung der Sicherheitslage und der Sicherstellung der polizeilichen Handlungsfähigkeit. Das System muss dieser Gruppe eine übersichtliche, kartenbasierte Darstellung aller relevanten Events bieten, welche effizient nach Ort, Zeit und Kategorie filterbar ist, um zeitintensive manuelle Recherchen zu substituieren.
- **Der IT-Betreiber (Repräsentativ: Thomas Zimmermann, 45):** Diese Persona repräsentiert die technischen Administratoren der internen IT-Systeme. Zu den Hauptaufgaben gehören die Konfiguration neuer Datenquellen, die Überwachung der Pipelines sowie das Eingreifen bei Fehlfunktionen der Crawler oder abfallender Datenqualität. Daraus resultiert die Notwendigkeit einer wartbaren Systemarchitektur, klaren Loggings und eines umfassenden Monitorings.
- **Die Veranstaltungsorganisatorin / Zivilperson (Repräsentativ: Lea Kessler, 29):** Diese Persona vertritt die zivile Nutzerschaft, welche das Cockpit für allgemeine Recherchen verwendet. Im Vordergrund stehen die einfache Bedienbarkeit der Applikation sowie die Überprüfung der korrekten Erfassung eigener Veranstaltungen im Kontext anderer regionaler Ereignisse.

6.1.2 Use-Case-Diagramm

Das nachfolgende Diagramm (Abbildung 9) visualisiert die grundlegenden Anwendungsfälle der identifizierten Personas in Interaktion mit dem System.

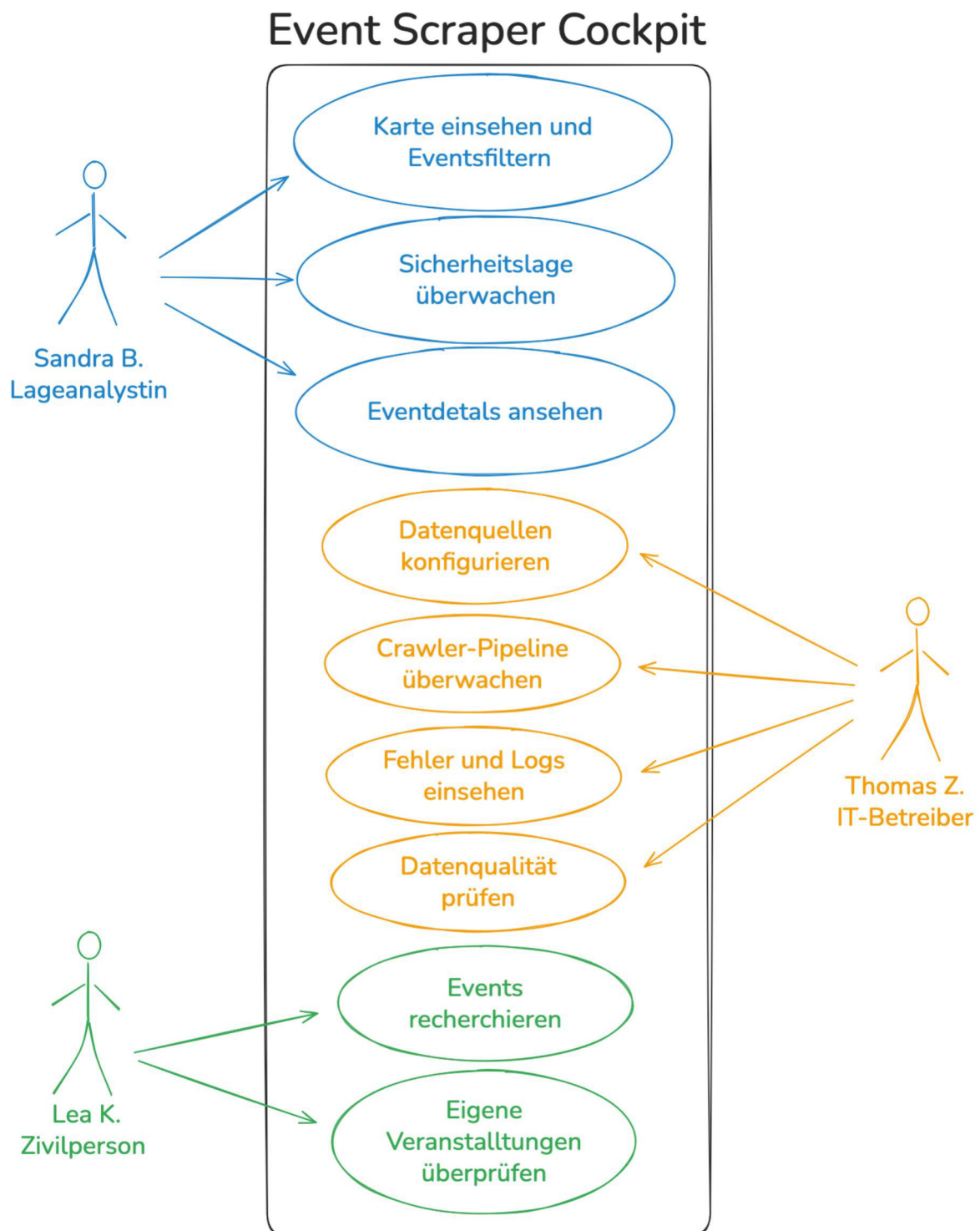


Abbildung 9: Use Case Diagramm

6.1.3 Funktionale Anforderungen

Die folgende Tabelle (Tabelle 24) gibt eine Übersicht über die funktionalen Anforderungen (Functional Requirements, FR) an den Proof-of-Concept. Da die Bachelorarbeit auf einer bestehenden Studienarbeit aufbaut, erweitern diese Anforderungen das bisherige System gezielt.

ID	Anforderung	Priorität
FR-01	Das System soll HTML-Webseiten und PDF-Dokumente von Gemeindeseiten KI-gestützt analysieren und relevante Felder (Titel, Datum/Zeit, Ort, Kategorie, Quelle/URL) extrahieren und in der Datenbank speichern.	Hoch
FR-02	Das System soll strukturierte Event-Quellen (Guidle und Mobilizon) als zusätzliche Datenquellen anbinden.	Mittel
FR-03	Das System soll Events nach Standardkategorien (schema.org oder tourismdata.ch) klassifizieren; fehlt eine Kategorie, soll eine KI-gestützte Kategorisierung erfolgen.	Hoch
FR-04	Das System soll Datenpipelines zur Sammlung, Bereinigung, Validierung und Deduplizierung von Ereignisdaten betreiben, inklusive Logging und Monitoring.	Hoch
FR-05	Das Frontend soll eine Web-Karte, Filter, Suche und Detailansicht bieten, um relevante Events schnell zu erfassen.	Hoch
FR-06	Das System soll Änderungen an bestehenden Events (Zeitverschiebung, Ortsänderung, Absage) erkennen, sofern die Quelle dies erlaubt.	Niedrig
FR-07	Das System soll offene Social-Media-Plattformen und Foren als zusätzliche Ereignisquellen erproben.	Niedrig

Tabelle 24: Funktionale Anforderungen

6.1.4 Non Functional Requirements (NFR)

Die nicht-funktionalen Anforderungen bauen auf den Spezifikationen der vorausgegangenen Studienarbeit auf. Sie wurden für die Bachelorarbeit anhand der erweiterten Aufgabenstellung sowie der offengebliebenen Umsetzungstickets aktualisiert [1, p. 46].

ID	NFR-01 Geografische Einschränkung auf den Kanton St. Gallen
Anforderung	Das System soll Ereignisse geografisch auf den Kanton St. Gallen einschränken können.
Szenario	Das System soll Ereignisse für die Standardansicht geografisch auf den Kanton St. Gallen einschränken können.
Akzeptanzkriterien	• Ereignisse mit eindeutigem Ort ausserhalb des Kantons St. Gallen werden nicht in der Standardansicht angezeigt. • Ereignisse ohne eindeutig auflösbaren Ort werden als unvollständig markiert und nicht automatisch verworfen. • Die geografische Einschränkung kann anhand gespeicherter Orts- oder Koordinateninformationen nachvollzogen werden.
Verifizierung	Test mit gemischten Ereignisdaten innerhalb und ausserhalb des Kantons St. Gallen. Kontrolle der gespeicherten und angezeigten Ereignisse in Datenbank und Frontend.
Priorität	Mittel

Tabelle 25: NFR-01 Datenumfang und Geofencing

ID	NFR-02 Konsistente Sprache und Terminologie
Anforderung	Die Benutzeroberfläche und die Dokumentation sollen konsistent in Deutsch (CH) verfasst sein.
Szenario	Eine Benutzerin oder ein Benutzer verwendet das Cockpit und liest Beschriftungen, Filter, Detailinformationen oder die begleitende Dokumentation.
Akzeptanzkriterien	• Alle sichtbaren UI-Texte sind in Deutsch (CH) formuliert. • Es wird kein «ß» verwendet. • Zentrale Fachbegriffe wie Ereignis, Quelle, Kategorie, Ort und Zeitraum werden konsistent verwendet. • Abkürzungen werden nur verwendet, wenn sie eingeführt oder allgemein verständlich sind.
Verifizierung	Sichtprüfung der Benutzeroberfläche und der Dokumentation anhand einer Terminologieliste.
Priorität	Mittel

Tabelle 26: NFR-02 Benutzbarkeit und Verständlichkeit

ID	NFR-03 Fehlertoleranz bei externen Quellen
Anforderung	Fehler einzelner Datenquellen sollen die Verarbeitung anderer Quellen nicht verhindern.
Szenario	Eine Gemeindeseite, ein PDF-Dokument oder eine Event-API ist nicht erreichbar, liefert ungültige Daten oder überschreitet ein Timeout.
Akzeptanzkriterien	• Ein Fehler in einer Quelle bricht die gesamte Pipeline nicht ab. • Fehlerhafte Quellen werden protokolliert. • Erfolgreich verarbeitbare Quellen werden weiterhin verarbeitet. • Der Pipeline-Lauf erzeugt am Ende eine Zusammenfassung mit Anzahl erfolgreicher, fehlgeschlagener und übersprungener Quellen.
Verifizierung	Testlauf mit mindestens einer absichtlich nicht erreichbaren Quelle und einer gültigen Quelle. Kontrolle, ob die gültige Quelle weiterhin verarbeitet und der Fehler protokolliert wird.
Priorität	Hoch

Tabelle 27: NFR-03 Robustheit der Datenpipeline

ID	NFR-04 Nachvollziehbarkeit durch Logging und Monitoring
Anforderung	Die Verarbeitungsschritte der Datenpipeline sollen nachvollziehbar protokolliert werden.
Szenario	Nach einem Pipeline-Lauf soll überprüft werden, welche Quellen verarbeitet wurden, welche Ereignisse extrahiert wurden und wo Fehler aufgetreten sind.
Akzeptanzkriterien	• Für jeden Pipeline-Lauf wird ein Zeitstempel gespeichert. • Pro Quelle werden Status, Anzahl gefundener Ereignisse und allfällige Fehlermeldungen protokolliert. • Fehler enthalten genügend Kontext, um die betroffene Quelle zu identifizieren. • Die Logs erlauben eine Unterscheidung zwischen technischen Fehlern, leeren Quellen und ungültigen Ereignisdaten.
Verifizierung	Manuelle Kontrolle der Log-Ausgaben nach einem Pipeline-Lauf mit erfolgreichen, leeren und fehlerhaften Quellen.
Priorität	Hoch

Tabelle 28: NFR-04 Beobachtbarkeit

ID	NFR-05 Wiederherstellbarkeit nach Fehlern
Anforderung	Das System soll nach einem technischen Fehler mit geringem Aufwand wiederhergestellt werden können.
Szenario	Ein Container oder ein Systembestandteil stürzt während des Betriebs oder während eines Pipeline-Laufs ab.
Akzeptanzkriterien	• Die betroffenen Container starten automatisch neu. • Das System ist nach einem Container-Absturz innerhalb von 5 Minuten wieder erreichbar. • Bereits persistierte Ereignisdaten bleiben erhalten. • Der Fehler ist in den Logs nachvollziehbar.
Verifizierung	Manueller Neustart oder Abbruch eines Containers während des Betriebs. Kontrolle von Wiederanlaufzeit, Datenpersistenz und Log-Ausgabe.
Priorität	Hoch

Tabelle 29: NFR-05 Recovery

ID	NFR-06 Einfache Installation und lokale Ausführbarkeit
Anforderung	Das System soll auf einem neuen Host-System mit geringem Aufwand lokal gestartet werden können.
Szenario	Eine Fachperson möchte den Prototyp auf einem neuen Entwicklungs- oder Testsystem starten.
Akzeptanzkriterien	• Frontend, Backend und Datenbank können über Docker Compose gestartet werden. • Die notwendigen Konfigurationsschritte sind dokumentiert. • Sensible Konfigurationswerte werden nicht im Quellcode gespeichert. • Ein initialer Start ist mit höchstens einem dokumentierten Startkommando möglich, sofern die Konfiguration vorhanden ist.
Verifizierung	Manueller Installationstest auf einem frischen Host-System oder in einer bereinigten Entwicklungsumgebung.
Priorität	Mittel

Tabelle 30: NFR-06 Installierbarkeit und Portabilität

ID	NFR-07 Erweiterbarkeit der Datenquellen
Anforderung	Neue Datenquellen sollen mit begrenztem Aufwand in die bestehende Pipeline integriert werden können.
Szenario	Neben bestehenden Quellen wie Gemeindeseiten, Eventfrog oder Guidle soll eine weitere Quelle, beispielsweise ein Forum oder eine offene Plattform, angebunden werden.
Akzeptanzkriterien	• Quellspezifische Logik ist gekapselt und nicht direkt mit der gesamten Pipeline vermischt. • Neue Quellen können über eine bestehende Schnittstelle oder ein klar definiertes Adaptermuster integriert werden. • Für eine neue Quelle müssen keine grundlegenden Änderungen an Datenbankmodell, Frontend oder Pipeline-Orchestrierung notwendig sein. • Die minimal erwarteten Event-Felder sind dokumentiert.
Verifizierung	Integration oder prototypische Anbindung einer zusätzlichen Quelle und Review der notwendigen Codeänderungen.
Priorität	Hoch

Tabelle 31: NFR-07 Wartbarkeit und Erweiterbarkeit

ID	NFR-08 Datenqualität bei extrahierten Events
Anforderung	Extrahierte Events sollen vor der Speicherung auf minimale Vollständigkeit und Plausibilität geprüft werden.
Szenario	Ein Harvester oder eine KI-Extraktion liefert unvollständige, widersprüchliche oder unsichere Event-Daten.
Akzeptanzkriterien	• Events enthalten mindestens Titel, Datum oder Zeitraum, Quelle und eine Ortsangabe oder einen Hinweis auf fehlende Ortsinformationen. • Ungültige oder unvollständige Events werden markiert und nicht unbemerkt als vollständig behandelt. • Die Quelle eines Events bleibt jederzeit nachvollziehbar. • Validierungsfehler werden protokolliert.
Verifizierung	Test mit gültigen, unvollständigen und widersprüchlichen Event-Daten. Kontrolle der Validierungsergebnisse in Datenbank und Logs.
Priorität	Hoch

Tabelle 32: NFR-08 Datenqualität und Validierung

ID	NFR-09 Vermeidung unnötiger Duplikate
Anforderung	Das System soll mehrfach gefundene Events erkennen und die Anzeige unnötiger Duplikate reduzieren.
Szenario	Dasselbe Event erscheint auf mehreren Quellen, beispielsweise auf einer Gemeindeseite und in einer Event-API.
Akzeptanzkriterien	• Potenzielle Duplikate werden anhand von Titel, Datum/Zeit, Ort und Quelle erkannt. • Erkannte Duplikate werden zusammengeführt oder als mögliche Duplikate markiert. • Die ursprünglichen Quellen bleiben nachvollziehbar. • Die Deduplizierung entfernt keine unterschiedlichen Events mit ähnlichem Titel ohne nachvollziehbare Begründung.
Verifizierung	Testdatensatz mit bekannten Duplikaten und ähnlichen, aber unterschiedlichen Events. Vergleich der erwarteten und tatsächlichen Deduplizierungsergebnisse.
Priorität	Hoch

Tabelle 33: NFR-09 Datenqualität und Deduplizierung

ID	NFR-010 Benutzbarkeit der Event-Übersicht
Anforderung	Relevante Events sollen im Frontend schnell auffindbar und verständlich dargestellt werden.
Szenario	Eine Benutzerin oder ein Benutzer möchte für einen bestimmten Zeitraum und Ort relevante Events auf der Karte oder in der Detailansicht prüfen.
Akzeptanzkriterien	• Events sind auf der Karte sichtbar und über eine Detailansicht abrufbar. • Filter für zentrale Kriterien wie Zeitraum, Ort oder Kategorie sind verständlich beschriftet. • Die Detailansicht zeigt mindestens Titel, Datum/Zeit, Ort, Kategorie und Quelle. • Die Quelle eines Events ist für Benutzerinnen und Benutzer ersichtlich.
Verifizierung	Manueller Usability-Test anhand typischer Suchaufgaben, zum Beispiel Finden eines Events in einer Gemeinde für einen bestimmten Zeitraum.
Priorität	Hoch

Tabelle 34: NFR-010 Usability

ID	NFR-11 Performance bei typischer Nutzung
Anforderung	Das Frontend soll bei typischen Datenmengen ohne spürbare Verzögerungen bedienbar bleiben.
Szenario	Eine Benutzerin oder ein Benutzer lädt die Event-Karte, filtert Ereignisse oder öffnet eine Detailansicht.
Akzeptanzkriterien	• Die Karte und Eventliste bleiben bei einem repräsentativen Testdatensatz bedienbar. • Filteraktionen liefern innerhalb von 2 Sekunden eine sichtbare Reaktion. • Das Öffnen einer Event-Detailansicht erfolgt innerhalb von 1 Sekunde. • Längere Ladezeiten werden durch einen sichtbaren Ladezustand angezeigt.
Verifizierung	Manueller Performancetest mit einem repräsentativen Testdatensatz und Messung der Reaktionszeiten im Browser.
Priorität	Mittel

Tabelle 35: NFR-11 Antwortzeit

ID	NFR-12 Schutz sensibler Konfigurationsdaten
Anforderung	Sensible Konfigurationsdaten wie API-Schlüssel oder Zugangsdaten dürfen nicht im Quellcode gespeichert werden.
Szenario	Das System wird lokal gestartet oder in einem Repository veröffentlicht.
Akzeptanzkriterien	• API-Schlüssel, Passwörter und Tokens werden über Umgebungsvariablen oder lokale Konfigurationsdateien bereitgestellt. • Beispielkonfigurationen enthalten keine echten Geheimnisse. • Das Repository enthält keine produktiven Zugangsdaten. • Die notwendigen Konfigurationswerte sind dokumentiert.
Verifizierung	Code-Review und Kontrolle der Repository-Dateien sowie der Beispielkonfiguration.
Priorität	Hoch

Tabelle 36: NFR-12 Sicherheit und Konfiguration

ID	NFR-13 Respektvoller Umgang mit öffentlichen Quellen
Anforderung	Das System soll öffentliche Quellen technisch und organisatorisch schonend nutzen.
Szenario	Der Harvester ruft Webseiten, PDFs oder APIs öffentlicher Anbieter ab.
Akzeptanzkriterien	• Quellen werden mit begrenzter Frequenz abgefragt. • Technische oder organisatorische Zugriffsbeschränkungen werden berücksichtigt. • Es werden nur öffentlich zugängliche Inhalte verarbeitet. • Personenbezogene Daten werden nicht als eigenständiges Ziel gesammelt oder ausgewertet.
Verifizierung	Review der Harvester-Konfiguration, Quellenauswahl und Dokumentation der getroffenen Einschränkungen.
Priorität	Hoch

Tabelle 37: NFR-13 Quellennutzung und Datenschutz

7 Implementation

In diesem Kapitel wird die konkrete Realisierung und softwaretechnische Umsetzung des Event-Suchroboter-Cockpits (ESC) detailliert dargelegt. Aufbauend auf dem in Kapitel 3 entworfenen Architekturkonzept dokumentiert dieser Teil der Arbeit die ingenieurmässige Ausführung des Proof-of-Concepts. Neben der Beschreibung, was und wie implementiert wurde, wird dabei auch auf die im Entwicklungsprozess aufgetretenen technischen Hürden, die Grenzen der eingesetzten Technologien sowie die daraus gewonnenen Erkenntnisse eingegangen. Ein Teil der nachfolgend beschriebenen Implementierungen überschneidet sich mit den Vorarbeiten aus der Studienarbeit [1], während andere Komponenten im Rahmen dieser Bachelorarbeit neu entwickelt oder grundlegend überarbeitet wurden. Für eine detaillierte Übersicht und systematische Differenzierung der Vorarbeiten gegenüber den Neuerungen wird auf Kapitel 1.5 verwiesen.

Die Kapitelstruktur orientiert sich konsequent am Lebenszyklus einer Veranstaltung innerhalb des Systems. Die sequenzielle Abfolge der nachfolgenden Abschnitte bildet den chronologischen Datenfluss der Pipeline präzise ab – angefangen bei der initialen Erfassung einer Quelle bis hin zur persistenten Speicherung und visuellen Aufbereitung im Frontend. Durch diese prozessuale Gliederung wird die Transformation von unstrukturierten, volatilen Webinhalten in ein konsolidiertes, relationales Datenmodell lückenlos nachvollziehbar.

Der erste und grundlegende Schritt dieses Prozesses ist die Crawler-Konfiguration, welche als deklarative Schnittstelle zu den heterogenen Webquellen fungiert. Darauf aufbauend behandeln die nachfolgenden Abschnitte die asynchrone Akquisition der Event-URLs, die Transformation der Inhalte in normalisierte Markdown-Dokumente sowie deren anschliessende Segmentierung. Im Zentrum der Datenaufbereitung stehen danach die semantische Extraktion mittels grosser Sprachmodelle (LLM) sowie die deterministischen Schritte zur geographischen Verortung und Klassifizierung. Die darauffolgenden Abschnitte widmen sich den komplexen Mechanismen der datenbankseitigen Deduplizierung, der relationalen Datenhaltung, der API-Bereitstellung und schliesslich der Benutzeroberfläche des Cockpits.

7.1 Crawler-Konfiguration

In der Semesterarbeit wurde für jede Crawl-Quelle eine eigene Python-Datei erstellt [1, p. 22-23]. Bei einer überschaubaren Anzahl Quellen war dieses Vorgehen noch handhabbar. Diese Bachelorarbeit hatte jedoch zum Ziel, alle Gemeinden des Kantons St. Gallen sowie weitere Event-Portale abzudecken – darunter Guidle, Mobilizon, Rheintal Events, Social-Media-Seiten und Foren. Damit wuchs die Zahl der Quellen auf über 80 (75 politische Gemeinden [7], sowie Eventfrog, Guidle, Rheintal Events, Mobilizon, Facebook und das FCSG-Forum). Ein quellspezifischer Python-File pro Quelle wäre nicht mehr wartbar gewesen. Die Lösung war eine vollständig datengetriebene Crawler-Architektur: Alle Quell-Konfigurationen werden als JSON erfasst und zur Laufzeit in typisierte `SourceConfig`-Dataclasses geladen. Eine vollständige Übersicht aller verfügbaren Einstellungsmöglichkeiten ist in `backend/crawlers/config/config_readme.md` dokumentiert.

7.1.1 Aufbau der Konfiguration

Jede Quelle wird durch ein JSON-Objekt beschrieben, das in mehrere thematische Abschnitte gegliedert ist. Code 1 zeigt eine typische Gemeinde-Konfiguration.


backend/crawlers/config/source_configs_gemeinden.json
JSON

```

{
  "identity": { "slug": "steinach", "display_name": "Steinach" },
  "listing": {
    "url": "https://steinach.ch/agenda-uebersicht/",
    "root_url": "https://steinach.ch",
    "wait_until": "domcontentloaded"
  },
  "href_collection": {
    "event_href_selector": "a[href*='/anlass/']",
    "event_href_prefix": "",
    "href_filter": ["/anlass/"]
  },
  "pipeline": {
    "enabled": true,
    "feed_pipeline": true,
    "concurrent_workers": 2,
    "default_source_field": "gemeinden_sg"
  },
  "fallback_location": {
    "full_address": "Schulstrasse 5, 9323 Steinach",
    "city_zip": "9323",
    "city_name": "Steinach"
  },
  "tags": {
    "source_type": "gemeinde",
    "kanton": "SG",
    "gemeinde": "Steinach"
  }
}

```

Code 1: Konfigurationseintrag für die Gemeinde Steinach.

7.1.2 Grundaufbau

Der Abschnitt `identity` legt Slug und Anzeigename fest. `listing` definiert die Einstiegs-URL sowie die Bedingung, unter der die Seite als geladen gilt (`wait_until`). `href_collection` steuert, welche Links auf der Listing-Seite als Event-URLs gelten – über einen CSS-Selektor, ein optionales URL-Präfix und einen Filterpfad.

Der `pipeline`-Abschnitt ist ein zentrales Steuerungsinstrument: Mit `enabled: false` lässt sich eine Quelle vollständig deaktivieren, ohne sie aus der Config zu entfernen. `feed_pipeline` trennt das Crawlen vom LLM-Verarbeitungsschritt – nützlich, wenn nur Rohdaten gesammelt werden sollen. `concurrent_workers` erlaubt eine quellenspezifische Parallelitätskontrolle, da manche Websites bei zu vielen gleichzeitigen Anfragen Fehler produzieren.

7.1.3 Preparation und Pagination

Viele moderne Websites erfordern Interaktionen, bevor Event-Links sichtbar werden. Der `preparation`-Abschnitt beschreibt initiale Scroll-Schritte und Cookie-Banner-Behandlung. Für die Pagination unterstützt die Config drei Strategien: `click` (Weiterblättern per Klick auf ein Paginationselement), `scroll` (Infinite-Scroll-Seiten wie Guidle) und `none`. Zur Fortschrittserkennung dienen zwei Strategien: `href_set_changed` prüft, ob neue Links hinzugekommen sind,

`active_page_changed` liest den aktiven Seitenzähler aus dem DOM aus. Diese Kombination deckte den Grossteil der vorgefundenen Pagination-Muster ab.

7.1.4 Fallback-Location

Im Abschnitt `fallback_location` wird eine Basisadresse für die Quelle hinterlegt. Fehlen in einer Event-Beschreibung explizite Ortsangaben, greift der Geocoding-Schritt auf diesen Fallback zurück. Damit lassen sich spärlich ausgezeichnete Event-Seiten trotzdem auf der Karte verorten.

7.1.5 Splitter

Einige Quellen publizieren alle Veranstaltungen auf einer einzigen Seite, ohne individuelle Detail-URLs. Für diese Fälle existiert der `splitter`-Abschnitt, der mit verschiedenen Strategien (`line_blocks`, `calendar_list`, `inline_event_list`, `markdown_table_rows`) die Seite in Einzel-Events zerlegt. Das Splitting-Verfahren wird in (Kapitel 7.4) detailliert beschrieben.

7.1.6 Besondere Herausforderungen

Die grösste Schwierigkeit lag nicht im technischen Aufbau der Config-Struktur selbst, sondern im Mapping der realen Website-Vielfalt auf ein einheitliches Schema. Pagination allein erwies sich als überraschend heterogen: Einige Seiten verwenden unsichtbare Buttons, andere laden Inhalte via Infinite Scroll, manche erfordern ein initiales Scrollen, bevor der Weiter-Button sichtbar wird – was das Zusammenspiel von `scroll_before_action`, `scroll_target_into_view` und `visibility_wait_ms` nötig machte. Jede neu hinzugefügte Gemeinde brachte mindestens eine Eigenheit mit.

Ein weiterer Aufwand war das initiale Befüllen der 75 Gemeinde-Konfigurationen. Da die Gemeinde-Websites keine einheitliche CMS-Basis teilen, musste jede URL, jeder CSS-Selektor und jede Fallback-Adresse manuell recherchiert und getestet werden.

7.1.7 Fazit

Das JSON-basierte Konfigurationssystem erlaubt es, neue Quellen ohne Code-Änderungen hinzuzufügen und bestehende Quellen granular zu steuern. Die klare Trennung der Abschnitte – von der URL-Sammlung über die Pagination bis zur Pipeline-Steuerung – macht die Konfiguration auch für Dritte lesbar und anpassbar. Im folgenden Abschnitt wird beschrieben, wie der Crawler die konfigurierten Quellen tatsächlich abarbeitet.

7.2 URL-Crawler

In der Semesterarbeit war jede Quelle als eigene Python-Klasse implementiert, was bei wachsender Quellanzahl zu erheblichem Wartungsaufwand führte. Die Umstellung auf eine konfigurationsgetriebene Architektur – beschrieben in Kapitel 7.1 – ermöglicht es, mit einem einzigen generischen Crawler alle Quellen abzudecken. Der URL-Crawler bildet damit die erste operative Stufe der Pipeline: Er navigiert die konfigurierten Zielseiten mit Playwright, sammelt Veranstaltungs-Links und übergibt diese an den nachgelagerten Markdown-Crawler. Inhaltliche Extraktion findet hier bewusst nicht statt; die Trennung hält die fehleranfällige Browser-Navigation von den rechenintensiven LLM-Stufen isoliert.

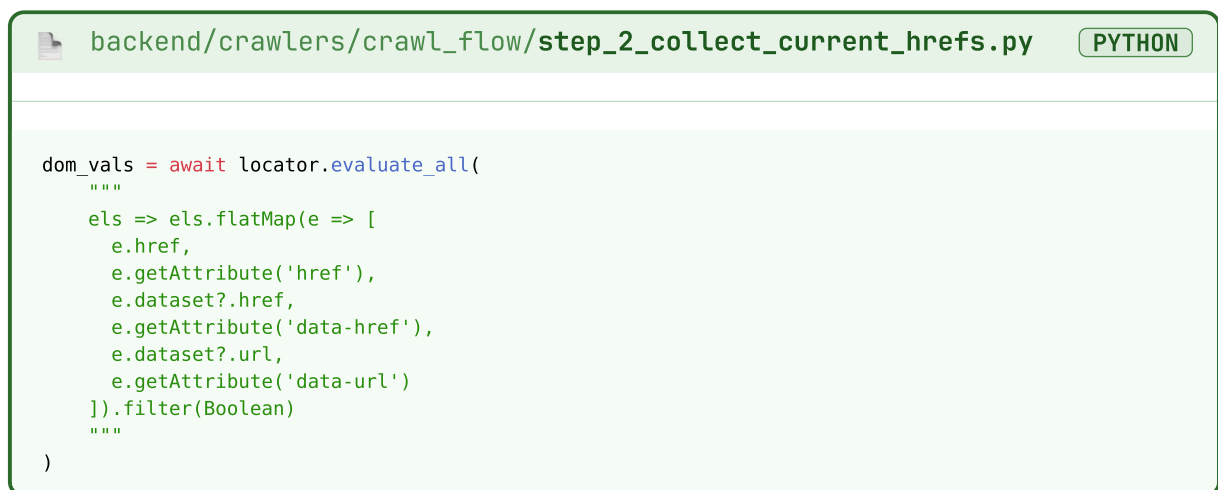
Der Ablauf ist in fünf klar getrennte Phasen unterteilt, die für jede Quelle sequenziell durchlaufen werden.

7.2.1 Phase 1: Seitenvorbereitung

Playwright öffnet eine Browser-Instanz mit standardisiertem Viewport (1366×900 Pixel) und optionalem User-Agent. Anschliessend wird die Listing-Seite geladen, Cookie-Zustimmungsdialoge werden automatisch geschlossen und konfigurierte Wartezeiten werden eingehalten, damit dynamisch nachgeladene Inhalte verfügbar sind, bevor die Link-Sammlung beginnt.

7.2.2 Phase 2: Link-Sammlung

Der Crawler extrahiert alle potenziellen Event-URLs aus dem DOM. Neben Standard-`href`-Attributen werden auch moderne HTML5-Datenattribute (`data-href`, `data-url`) ausgewertet, um Kompatibilität mit Single-Page-Anwendungen sicherzustellen (Code 2). Anschliessend werden irrelevante URI-Schemata (Telefon, E-Mail, JavaScript) verworfen, relative Pfade kanonisiert und die URLs gegen die konfigurierten Filterregeln der Quelle geprüft.



```

backend/crawlers/crawl_flow/step_2_collect_current_hrefs.py PYTHON

dom_vals = await locator.evaluate_all(
    """
    els => els.flatMap(e => [
      e.href,
      e.getAttribute('href'),
      e.dataset?.href,
      e.getAttribute('data-href'),
      e.dataset?.url,
      e.getAttribute('data-url')
    ]).filter(Boolean)
    """
)
  
```

Code 2: Extraktion von Links aus Standard- und SPA-spezifischen DOM-Attributen.

7.2.3 Phase 3: Navigations-Aktion

Abhängig von der Quellkonfiguration (`pagination.action`) wechselt der Crawler zur nächsten Seite entweder durch einen Klick auf ein Pagination-Element oder durch Scrollen. Bei Infinite-Scroll-Layouts wird das Scrollen über echte Maus-Rad-Events simuliert (Code 3), da einfache DOM-Manipulation von modernen Lazy-Loading-Bibliotheken oft nicht als Benutzerinteraktion erkannt wird.

```

backend/crawlers/crawl_flow/step_3_advance_action.py PYTHON

await page.evaluate("window.scrollTo(0, document.body.scrollHeight - 2000)")
await asyncio.sleep(0.1)
for _ in range(4):
    await page.mouse.wheel(0, 600)
    await asyncio.sleep(0.15)

```

Code 3: Hybrides Scroll-Verhalten: Teleport ans Seitenende gefolgt von Maus-Rad-Events zur Auslösung von IntersectionObservern.

7.2.4 Phase 4: Fortschrittserkennung

Nach jeder Navigation prüft der Crawler, ob tatsächlich Fortschritt erzielt wurde. Dazu wird vor und nach der Aktion ein Zustandsvektor (`ListingStateSnapshot`) erfasst, der Elementanzahl, gefundene Links, aktiven Seitenzähler und einen HTML-Hash des Listencontainers enthält (Code 4). Bleibt der Zustand unverändert, wird der Crawl-Zyklus abgebrochen, statt endlos auf einer steckengebliebenen Seite zu verharren.

```

backend/crawlers/crawl_flow/step_4_progress_detection.py PYTHON

@dataclass
class ListingStateSnapshot:
    item_count: int
    hrefs: List[str]
    active_page_value: str
    container_html_hash: str
    document_height: int

```

Code 4: Zustandsvektor zur Erkennung von Fortschritt nach einer Navigations-Aktion.

7.2.5 Phase 5: Abbruchkriterien

Nach jeder Sammel- und Navigationsrunde prüft die Stop-Logik konfigurierbare Sicherheitsgrenzen (`max_pages` , `max_urls`). Meldet die Fortschrittserkennung Stillstand, bricht der Crawler nicht sofort ab: Er versucht bis zu dreimal, die Seite durch einen «Screen Shake» zu reaktivieren – 1500px nach oben scrollen, kurz warten, dann per `window.scrollTo` ans Seitenende springen (Code 5). Dieser Mechanismus ist insbesondere für Guidle relevant, wo Infinite-Scroll-Quellen gelegentlich ins Stocken geraten; Details dazu in Kapitel 7.12. Erst nach drei erfolglosen Versuchen oder wenn kein Navigationsziel mehr existiert, wird der Crawl kontrolliert beendet und die gesammelten URLs werden in die `eventlink_queue` eingespeist.

backend/pipeline/run_crawlers.py PYTHON

```
if "no_progress" in stop_decision.reason:
    consecutive_no_progress += 1
    if consecutive_no_progress <= 3:
        # "Shake" the browser: pull up 1500px, wait, slam to the bottom
        await page.mouse.wheel(0, -1500)
        await asyncio.sleep(0.5)
        await page.evaluate(
            "window.scrollTo(0, document.body.scrollHeight)"
        )
    continue
```

Code 5: Screen Shake bei ausbleibendem Fortschritt: bis zu drei Wiederholungsversuche vor dem Abbruch.

7.2.6 Fazit

Der generische, konfigurationsgetriebene URL-Crawler reduziert den Wartungsaufwand gegenüber der Semesterarbeit erheblich: Neue Quellen werden durch einen Konfigurationseintrag ergänzt, ohne Code zu schreiben. Die fünfstufige Struktur – Vorbereitung, Sammlung, Navigation, Fortschrittsprüfung, Abbruchentscheid – macht jeden Schritt einzeln testbar und austauschbar. Die gesammelten Event-URLs werden anschliessend vom Markdown-Crawler abgerufen und in ein einheitliches Zwischenformat überführt.

7.3 Markdown-Crawler

In der Semesterarbeit umfasste der Detail-Crawler vier Schritte: Playwright-Abruf mit `networkidle`-Warten, Boilerplate-Entfernung mit BeautifulSoup, Konvertierung nach Markdown mit `markdownify` und ein minimales Frontmatter mit `source_url`, `crawled_at` und `domain` – primär als Debugging-Artefakt [1, p. 27-28]. Parallelisierung war bewusst gedrosselt, weil URLs quellweise gesammelt wurden und aggressive Parallelität das Risiko von Domain-Sperren erhöht hätte.

In der vorliegenden Arbeit wurde dieser Crawler an mehreren Stellen erweitert. Die Inhaltsauswahl arbeitet nicht mehr mit pauschaler Boilerplate-Entfernung, sondern wählt den Seitenbereich heuristisch anhand von Textlänge, Datumsangaben und eventtypischen Begriffen. Das Frontmatter ist von einem Debugging-Artefakt zu einem aktiven Metadatenträger geworden: Es überträgt Splitter-Konfiguration, Fallback-Adresse und Quelltyp in alle nachfolgenden Stufen. Hinzu kommen ein statischer `aihttp`-Fallback für Seiten, bei denen Playwright scheitert, sowie per Quellkonfiguration steuerbare Parallelitätslimits. Die grösste Neuerung ist die integrierte PDF-Extraktion, auf die dieses Kapitel schwerpunktmässig eingeht.

7.3.1 Laden und Inhaltsauswahl

Viele Eventseiten bauen ihren Inhalt erst über JavaScript auf, weshalb der Crawler standardmässig eine Playwright-gesteuerte Chromium-Instanz verwendet. Liefert der Browser keinen verwertbaren Text oder einen HTTP-Fehler, fällt er auf einen statischen `aihttp`-Request zurück. Nach dem Laden bestimmt `_pick_content_root()` den relevanten Seitenbereich heuristisch: Kandidaten wie `main`, `article` oder `div.container` werden anhand von Textlänge, erkannten Datumsangaben und eventtypischen Begriffen bewertet und der Score-höchste Bereich gewählt. Ist in der Quellkonfiguration ein expliziter Selector hinterlegt, überstimmt er diese Heuristik vollständig.

7.3.2 Frontmatter als Metadatenträger

Vor dem eigentlichen Inhalt schreibt der Crawler einen YAML-Frontmatter-Block in die Markdown-Datei (Code 6). Dieser überträgt Verarbeitungsparameter wie Splitter-Konfiguration, Fallback-Adresse und Quelltyp aus dem URL-Crawler in alle nachgelagerten Stufen – ohne dass diese auf die ursprüngliche Quellkonfiguration zugreifen müssen.



The screenshot shows a code editor window with a title bar that reads "data/scraped_md/beispiel.md" and a "YAML" icon on the right. The editor contains a YAML frontmatter block enclosed in three dashes. The content of the block is as follows:

```

---
source_url: https://www.altstaetten.ch/_rte/anlass/...
source_slug: altstaetten
source_name: Altstätten
listing_url: https://www.altstaetten.ch/anlaesseaktuelles
source_type: gemeinde
kanton: SG
gemeinde: Altstätten
fallback_location: {"full_address": "Rathausplatz 2,
9450 Altstätten", "city_zip": "9450", "city_name": "Altstätten"}
---

```

Code 6: Frontmatter einer erzeugten Markdown-Datei mit Kontext aus der Quellkonfiguration.

7.3.3 PDF-Extraktion

Ein Teil der erfassten Quellen stellt Veranstaltungen nicht als HTML-Detailseiten bereit, sondern als PDF-Dokumente – etwa Terminkalender einzelner Gemeinden. Um diese nicht auszuschliessen, sucht der Crawler nach jeder HTML-Konvertierung die Seite nach PDF-Verweisen ab. Erkannt werden dabei nicht nur direkte `.pdf`-Links, sondern auch PDF-URLs mit Query-Parametern sowie eingebettete Quellen in `iframe`-, `embed`- und `object`-Elementen (Code 7).

The image shows a code editor window with a green header bar. The header bar contains a file icon, the filename 'backend/utils/pdf_extractor.py', and a 'PYTHON' language indicator. The code area has a light green background and contains the following Python code:

```
soup = BeautifulSoup(html or "", "html.parser")
for tag in soup.find_all(["a", "iframe", "embed", "object"]):
    add(tag.get("href"))
    add(tag.get("src"))
    add(tag.get("data"))
```

Code 7: Erkennung verlinkter und eingebetteter PDF-Quellen im HTML.

Gefundene PDFs werden heruntergeladen und mit PyMuPDF in Text extrahiert, der anschliessend an die Markdown-Datei angehängt wird. Damit durchläuft PDF-Inhalt dieselbe Pipeline wie HTML-Inhalt – inklusive Splitter und LLM-Extraktion – ohne gesonderte Behandlung. Die Gemeinde Niederbüren etwa stellt ihren Veranstaltungskalender als PDF bereit; der Crawler erkennt den Link, extrahiert den Text und übergibt die vollständige Markdown-Datei an die nächste Stufe.

Die Extraktion funktioniert zuverlässig nur bei PDF-Dokumenten mit maschinenlesbarem Text. Gescannte PDFs liefern wenig oder keinen verwertbaren Text; sehr umfangreiche Dokumente können das Kontextlimit des LLM überschreiten, was jedoch nicht die Extraktion selbst, sondern erst die nachgelagerte LLM-Stufe betrifft.

7.3.4 Fazit

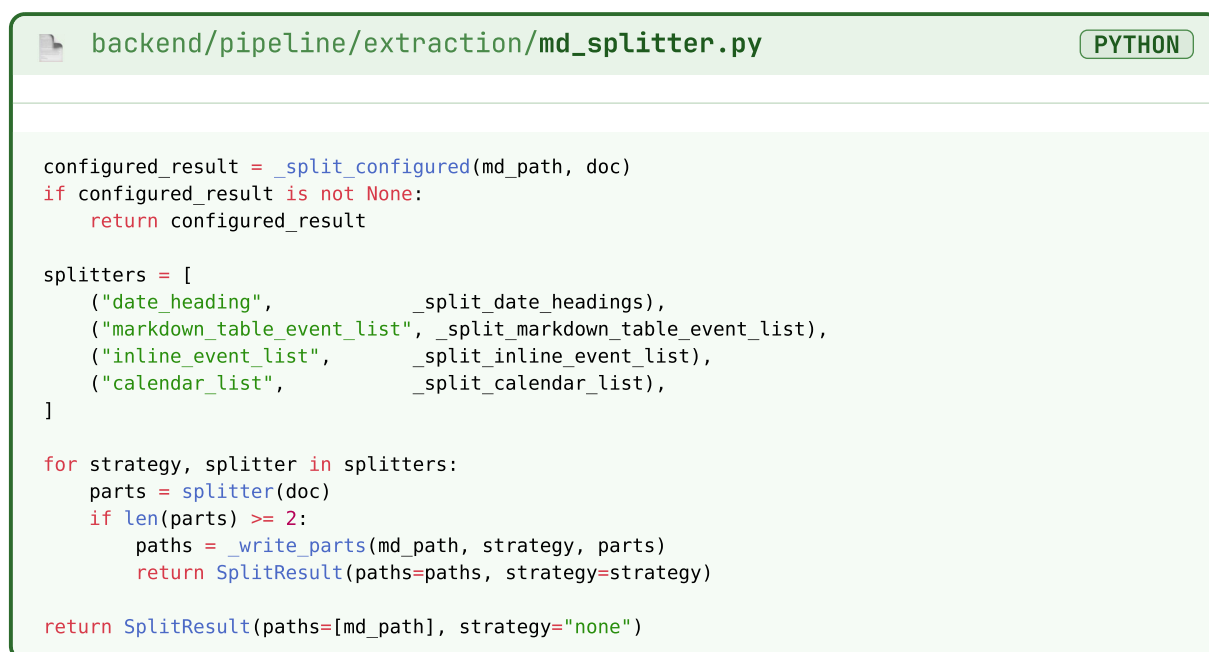
Der Markdown-Crawler schafft ein einheitliches Zwischenformat für heterogene Quellen: Playwright-Rendering und statischer Fallback decken die Bandbreite moderner und einfacher Webseiten ab, das Frontmatter transportiert Quellkontext verlustfrei durch die Pipeline, und die PDF-Integration schliesst eine relevante Lücke ohne zusätzliche Sonderpipeline. Die erzeugte Markdown-Datei wird im nächsten Schritt vom Splitter ausgewertet, der erkennt, ob eine Datei mehrere Veranstaltungen enthält, und diese vor der LLM-Extraktion in separate Einheiten aufteilt.

7.4 Markdown-Splitter

Nicht alle Quellen verfügen über eigene Detailseiten pro Veranstaltung. Gemeinde- Websites und PDF-Kalender liefern häufig eine einzige Seite mit zahlreichen Veranstaltungen, aus der der Markdown-Crawler zwangsläufig ein einzelnes, grosses Dokument erzeugt. Der Splitter steht zwischen dem Crawler und der LLM-Extraktion und zerteilt solche Dokumente in separate Markdown-Dateien – je eine pro Veranstaltung. 7 der 82 Quellen sind als Split-Quellen konfiguriert; hinzu kommen PDF-Kalender, die über Kapitel 7.3.3 in den gleichen Ablauf eingespiessen werden.

7.4.1 Strategieauswahl

Die Kernfrage ist, ob und wie ein Dokument aufgeteilt werden soll. `split_markdown_file()` beantwortet sie in zwei Schritten (Code 8): Zuerst wird geprüft, ob im Frontmatter ein expliziter Splitter konfiguriert ist (`splitter_enabled: true`). Ist das der Fall, wird ausschliesslich dieser verwendet. Greift kein konfigurierter Splitter, werden vier automatische Erkennungsstrategien in fester Reihenfolge versucht. Erzeugt keine davon mindestens zwei Teile, wird die Datei unverändert weitergereicht.



```

backend/pipeline/extraction/md_splitter.py PYTHON

configured_result = _split_configured(md_path, doc)
if configured_result is not None:
    return configured_result

splitters = [
    ("date_heading", _split_date_headings),
    ("markdown_table_event_list", _split_markdown_table_event_list),
    ("inline_event_list", _split_inline_event_list),
    ("calendar_list", _split_calendar_list),
]

for strategy, splitter in splitters:
    parts = splitter(doc)
    if len(parts) >= 2:
        paths = _write_parts(md_path, strategy, parts)
        return SplitResult(paths=paths, strategy=strategy)

return SplitResult(paths=[md_path], strategy="none")

```

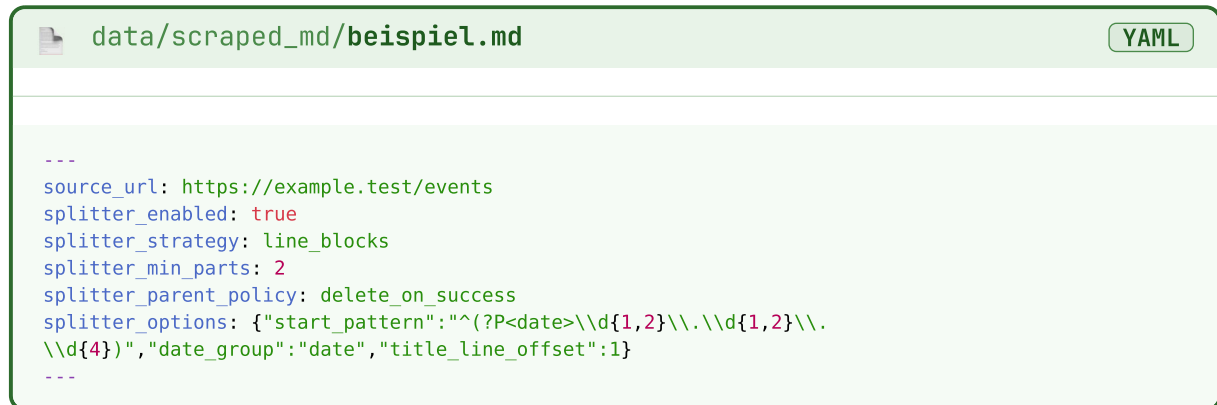
Code 8: Priorisierung: konfigurierter Splitter zuerst, automatische Strategien als Fallback.

7.4.2 Automatische Strategien

Die vier automatischen Strategien decken die häufigsten Dokumentstrukturen ab, ohne dass eine Quellkonfiguration nötig ist. `date_heading` erkennt Markdown-Überschriften der Form `## 13.06.2026 Dorffest` und erzeugt pro Überschrift eine eigene Datei. `markdown_table_event_list` verarbeitet Tabellen, in denen die erste Zelle Datum und Uhrzeit und die zweite Zelle den Eventtitel enthält. `inline_event_list` behandelt Accordion- oder Contao-artige Eventblöcke, bei denen aufeinanderfolgende strukturierte Textabschnitte je eine Veranstaltung beschreiben. `calendar_list` ist auf kalenderartige Texte aus PDF-Dokumenten ausgelegt – sie erfordert mindestens drei erkannte Veranstaltungen und leitet die Jahreszahl aus der Quell-URL oder dem Dokumentinhalt ab.

7.4.3 Konfigurierte Strategien

Für Quellen mit bekannter, aber unregelmässiger Struktur greift die konfigurierte Variante. Das Frontmatter der Markdown-Datei trägt `splitter_enabled: true` sowie eine Strategieangabe und optional `splitter_options` mit quellspezifischen Parametern (Code 9). Diese Werte stammen direkt aus der Quellkonfiguration und werden vom Markdown-Crawler ins Frontmatter übertragen.



```

---
source_url: https://example.test/events
splitter_enabled: true
splitter_strategy: line_blocks
splitter_min_parts: 2
splitter_parent_policy: delete_on_success
splitter_options: {"start_pattern": "^(?P<date>\\d{1,2}\\..\\d{1,2}\\..\\d{4})", "date_group": "date", "title_line_offset": 1}
---

```

Code 9: Frontmatter mit konfiguriertem Splitter und quellspezifischen Optionen.

Die flexibelste konfigurierte Strategie ist `line_blocks`: Sie erkennt Startzeilen anhand eines konfigurierbaren regulären Ausdrucks – typischerweise eine Datumszeile wie `13.06.2026` – und fasst alles bis zur nächsten Startzeile zu einem Eventblock zusammen. Neben generischen Strategien wie `line_blocks` und `calendar_list` enthält die Strategie-Map auch quellspezifische Aliasse wie `degersheim_agenda_box` oder `niederhelfenschwil_event_box`, die sprechend benannt sind, aber auf dieselben Implementierungen zeigen – für Nachvollziehbarkeit ohne Codeduplizierung.

Jede erzeugte Split-Datei erbt das Frontmatter der Parent-Datei (Quelle, Kanton, Fallback-Adresse) und ergänzt die Felder `split_from`, `split_index` und `split_strategy`. Nach erfolgreicher Aufteilung löscht der Worker die Parent-Datei, sofern `splitter_parent_policy: delete_on_success` gesetzt ist.

7.4.4 Fazit

Der Splitter löst das strukturelle Problem, dass Listing-Seiten und PDF-Kalender mehrere Veranstaltungen in einem Dokument bündeln. Die Kombination aus automatischer Erkennung für unbekannte Quellen und expliziter Konfiguration für schwierige Sonderfälle hält den Wartungsaufwand gering: neue Quellen mit Standardstruktur funktionieren ohne Konfiguration, Problemquellen werden einmalig im Config-File eingetragen. Das Ergebnis sind Markdown-Dateien, die idealerweise genau eine Veranstaltung enthalten und damit die optimale Eingabe für die LLM-Extraktion bilden.

7.5 LLM-Extraktion

In der Semesterarbeit [1, p. 34] unterstützte ein regelbasierter Markdown-Parser die LLM-Extraktion: Er versuchte, Felder wie Datum, Uhrzeit und Ort deterministisch aus dem Markdown zu identifizieren und dem LLM als Vorarbeit zu übergeben. Der Ansatz erschien effizient – bei wenigen, gut bekannten Quellen funktionierte er. Mit wachsender Quellenvielfalt zeigte sich jedoch ein Muster, das auf Overfitting hindeutet: Für neue Quellen produzierte der Parser zunehmend fehlerhafte Vorschläge, die das LLM übernahm statt zu ignorieren. Um die Unsicherheit des Parsers sichtbar zu machen, wurden Confidence Scores eingeführt, die gemeinsam mit den Kandidaten an das LLM übergeben wurden. Damit wurde die Payload jedoch so gross, dass einzelne Events bis zu fünf Minuten Verarbeitungszeit benötigten. Der MD-Parser wurde daraufhin vollständig entfernt.

Stattdessen wurde die LLM-Pipeline von Grund auf neu aufgebaut: mit saubererem Prompt Engineering, deterministischen Guardrails und einem schnelleren Modell. Der Ablauf ist in Kapitel 4.2.2 dargestellt.

7.5.1 Markdown-Vorverarbeitung

Bevor das Markdown das LLM erreicht, durchläuft es zwei Bereinigungsschritte in `backend/llm/event_context.py`.

`clean_markdown()` entfernt YAML-Frontmatter, bereinigt Mojibake-Zeichenfolgen, normalisiert Leerzeichen und filtert bekannte UI-Rauschzeilen wie «Jetzt teilen», «Abbrechen» oder Countdown-Fragmente («Tage», «Std», «Min»). Das Ergebnis ist lesbares Deutsch ohne Navigations- und Schnittstellenartefakte.

`compact_markdown_for_llm()` begrenzt den an das LLM gesendeten Text auf 3000 Zeichen (konfigurierbar über `LLM_MD_MAX_CHARS`). Die Strategie ist nicht simples Abschneiden: Zunächst wird ein Head-Budget von bis zu 1500 Zeichen reserviert, da der Seitenanfang bei Detail-Seiten typischerweise Titel, Datum und Ort enthält. Anschliessend werden Anker-Zeilen – Überschriften, Zeilen mit expliziten Labels wie «Datum:» oder «Ort:» sowie Zeilen mit Datums- und Zeitmustern – identifiziert und mit einem Kontext-Fenster von je zwei Zeilen bevorzugt einbezogen. Übersprungene Abschnitte werden mit `...[gekürzter Abschnitt]...` markiert, sodass das LLM erkennt, dass Inhalt fehlt. Die vollständigen Markdown-Dateien bleiben in `data/scraped_md/` erhalten; die Komprimierung betrifft ausschliesslich den LLM-Input.

7.5.2 Locked Facts

Bevor das LLM aufgerufen wird, extrahiert `build_locked_facts()` deterministisch drei Felder direkt aus dem Markdown: `source_url` aus dem Frontmatter, `source_type` durch URL-Matching gegen bekannte Portale sowie `coordinates` durch Regex-Suche nach Koordinatenformaten in Links, Map-Parametern und dekadischen Koordinatenfeldern (Code 10).

```
backend/llm/event_context.py PYTHON

def build_locked_facts(md: str, *, source_url_hint: str | None = None) -> LockedFacts:
    source_url = source_url_hint or extract_source_url(md)
    return LockedFacts(
        source_url=source_url,
        source_type=derive_source_type(source_url),
        coordinates=extract_coordinates(md),
    )
```

Code 10: Deterministische Extraktion der gesperrten Felder vor dem LLM-Aufruf.

Diese Felder werden dem LLM als «Locked Facts» im Prompt übergeben – das Modell wird explizit angewiesen, sie unverändert zu übernehmen – und durch die Guardrails nach dem LLM-Aufruf erneut erzwungen. Damit wird verhindert, dass das LLM source URLs kürzt, Koordinaten erfindet oder den Source-Type verändert.

7.5.3 Prompt Engineering

Der aktive System-Prompt in `backend/llm/llmhub.py` definiert die Rolle des Modells, den Output-Kontrakt und die Hard-Regeln (Code 11).

```
backend/llm/llmhub.py PYTHON

SYSTEM_PROMPT_FULL = (
    "You are a strict event extractor for German Markdown event pages.\n"
    "Extract semantically from the Markdown.\n"
    "Locked facts are truth and must be copied unchanged.\n"
    "Return only real calendar events. Do not invent data.\n"
    "If a value is not explicit or strongly supported, return null, "
    "except documented defaults.\n"
    "Return ONLY JSON in this shape: {\"events\": [{...}]}.\n"
    "Each event object may contain ONLY these keys: "
    "\"title, description, date, full_location, country, country_short_id, "
    "\"city_name, city_zip, street, street_number, start_time, end_time, "
    "\"source_url, source_type, coordinates, category, event_status.\n"
    "Do not output database id; the database creates it.\n"
    "Use null instead of placeholders such as '-', '?', 'n/a', "
    "or empty strings.\n"
)
```

Code 11: System-Prompt für die LLM-Extraktion.

Der User-Prompt enthält die Locked Facts, optionale Evidence Hints, Extraktionsprioritäten und das komprimierte Markdown. Besonders relevant sind die Extraktionsprioritäten: Das Modell wird angewiesen, Fakten in räumlicher Nähe zum Eventtitel zu bevorzugen, explizite Labels wie «Termin:», «Datum:» oder «Wo?» gegenüber unspezifischen Datumsangaben zu priorisieren und Navigations- und Footer-Inhalte zu ignorieren. Die Adresszerlegung wird explizit beschrieben (Venue, Strasse, Hausnummer, PLZ, Stadt), da viele Gemeinde-Websites

Adressen als freie Textfelder ohne strukturierte Auszeichnung enthalten. Die `category` wird auf `null` gesetzt – sie wird in Kapitel 7.7 separat zugewiesen.

7.5.4 Deterministische Guardrails

Nach dem LLM-Aufruf durchläuft jedes extrahierte Event `apply_event_guardrails()` in `backend/llm/guardrails.py`. Diese Funktion ist vollständig deterministisch und unabhängig vom Modell. Sie deckt unter anderem ab:

Datum, Zeit und PLZ werden mit Regex normalisiert: `dd.mm.yyyy` und `HH:mm` als einzige akzeptierte Formate, ISO-Daten werden umgewandelt, ungültige Zeiten verworfen. Platzhalter wie `<->`, `<?>`, `<n/a>`, `<...>` werden auf `null` gesetzt. Bekannte schlechte Titel wie «Veranstaltungskalender», «Kopfzeile» oder URL-Fragmente werden erkannt und wenn möglich durch den ersten validen Heading aus dem Evidence Pack ersetzt. Die `event_status`-Guardrail (Code 12) scannt nicht nur das LLM-Ausgabefeld, sondern das gesamte Markdown auf Schlüsselwörter für Absagen, Verschiebungen und Verzögerungen – das Modell kann einen Status übersehen, die Guardrail nicht.


backend/llm/guardrails.py
PYTHON

```

_STATUS_KEYWORDS = {
    "cancelled": ("abgesagt", "annulliert", "storniert", "cancelled", "canceled"),
    "postponed": ("verschoben", "postponed", "rescheduled", "neu angesetzt"),
    "delayed": ("verspätet", "verspaetet", "delayed", "verzögert"),
}

def _normalize_status(value: Any, clean_md: str) -> str:
    low_md = (clean_md or "").lower()
    for status, words in _STATUS_KEYWORDS.items():
        if any(word in low_md for word in words):
            return status
    # Bekannter Alias oder Standardwert «scheduled»
    text = _clean_text(value)
    if text:
        low = text.lower().strip()
        if low in _STATUS_ALIASES:
            return _STATUS_ALIASES[low]
    return _DEFAULT_EVENT_STATUS

```

Code 12: Status-Guardrail: Markdown-Scan überschreibt das LLM-Ausgabefeld.

7.5.5 Evidence Hints und Modellwahl

Parallel zur Prompt-Arbeit wurde ein Evidence-System entwickelt: `build_evidence_pack()` in `event_context.py` extrahiert aus dem Markdown strukturierte Hinweise – Überschriften, Zeilen mit expliziten Labels («Datum:», «Ort:») und Zeilen mit Datums- und Zeitmustern, jeweils mit umliegenden Kontext-Zeilen. Diese wurden als zusätzlicher JSON-Block im User-Prompt als «Evidence Hints» eingebunden.

Das Problem war dasselbe wie beim MD-Parser: Die Payload wurde zu gross. Mit dem ursprünglichen Modell stiess die Verarbeitungszeit in inakzeptable Bereiche; ein Wechsel auf das schnellere `minimax-m2.7` brachte deutlich kürzere Laufzeiten, führte aber zum nächsten Problem – das Modell hat ein grösseres Kontextfenster von 196'608 Token (Tabelle 38), ist dabei aber empfindlicher gegenüber grossen Prompts in Bezug auf Laufzeit. Damit wurden die Evidence Hints als aktiv konfigurierbare Option beibehalten (`LLM_USE_EVIDENCE_HINTS`), aber standardmässig deaktiviert.

Eigenschaft	Wert
Modell	<code>minimax-m2.7</code>
Typ	Chat
Kontextfenster	196'608 Token
Durchsatz	45.3 tok/s
Laufzeitumgebung	Mac Studio / LM Studio
Temperatur (konfiguriert)	0

Tabelle 38: Eigenschaften des eingesetzten LLM-Modells.

Die Temperatur wird auf `0` gesetzt, damit die Extraktion deterministisch und reproduzierbar bleibt – eine Voraussetzung für aussagekräftige Evaluationsläufe (Kapitel 8.1.2). Das Modell wird über den intern bereitgestellten LLMHub-Endpoint (`https://api.llmhub.infs.ai/v1/`) angebunden.

Das Evidence-System bleibt im Code intakt, weil die Modellentwicklung in diesem Bereich sehr schnell voranschreitet [14] – was heute an Kontextlimits scheitert, könnte mit dem nächsten Modell-Update problemlos funktionieren.

7.5.6 Fazit

Was letztlich funktionierte, war die Kombination aus schnellem Modell, komprimiertem aber strukturiertem Prompt und konsequenten deterministischen Guardrails. Der Evaluationslauf (Kapitel 8.1.2) bestätigt das: F1 von 0.862 mit perfekter Genauigkeit bei Datum, Adresse und Status. Die Kategorieklassifikation bleibt die einzige Schwachstelle und wird separat in der Category-Worker-Stufe behandelt. Der nächste Schritt nach der LLM-Extraktion ist das Geo-Enrichment, das extrahierte Ortsangaben in Koordinaten überführt.

7.6 Geo-Anreicherung

Die Geo-Anreicherung überführt die vom LLM extrahierten Ortsangaben in kartografisch nutzbare Koordinaten. Die grundlegende Architektur ist gegenüber der Semesterarbeit unverändert [1, p. 32-33]: Als Referenzbasis dient die offizielle Adressdatenbank von swisstopo, die lokal in der Tabelle `swiss_addresses` vorgehalten wird. Externe Geokodierungs-APIs werden bewusst vermieden – aus Gründen der Datensicherheit, der Reproduzierbarkeit und der Offline-Fähigkeit. Die Matching-Strategie folgt dem gleichen gestuften Prinzip: von exaktem Treffer bis zu sukzessiv weniger restriktiven Kriterien, wobei mehrdeutige Treffer konsequent verworfen werden.

7.6.1 Erweiterungen gegenüber der Semesterarbeit

Neu hinzugekommen sind drei Punkte.

Reverse Lookup. Enthält ein Event bereits gültige Koordinaten – etwa weil sie von Guide oder aus einem eingebetteten Kartenlink stammen – wird kein Forward-Lookup durchgeführt. Stattdessen ermittelt der Geo-Worker die nächstgelegene Adresse in `swiss_addresses` per Bounding-Box-Abfrage mit Euklidischer Distanz-Sortierung und füllt damit fehlende Adressfelder (PLZ, Ort, Strasse). Strasseninformationen werden aus dem Reverse Lookup nur übernommen, wenn die ursprüngliche Extraktion bereits einen Strassenhinweis enthielt – andernfalls würde die nächstgelegene Adresse willkürlich wirken.

Mehrstufige generische Suche. Schlägt der exakte strukturierte Match fehl, durchläuft der Worker bis zu fünf Tierstufen: Normalisierung von Bindestrichen und Leerzeichen in Strassenamen, tokenbasierte `LIKE`-Suche, alternative Stadtname-Schreibweisen (z.B. «Rapperswil-Jona» → «Rapperswil») und als letzte Stufe eine stadtname-unabhängige Suche – die jedoch nur akzeptiert wird, wenn dabei exakt ein eindeutiger Koordinatentreffer resultiert (Code 13).


backend/pipeline/workers/geo_worker.py
PYTHON

```

# Tier 5: ignore city (VERY conservative: must be unique)
if hno:
    sql = """
    SELECT lat, lon FROM swiss_addresses
    WHERE zip = %(zip)s
       AND LOWER(REGEXP_REPLACE(stn_label, '[-\\s]+', '', 'g')) =
       LOWER(REGEXP_REPLACE(%(street)s, '[-\\s]+', '', 'g'))
       AND UPPER(REGEXP_REPLACE(adr_number, '\\s+', '', 'g')) =
       UPPER(REGEXP_REPLACE(%(hno)s, '\\s+', '', 'g'))
    LIMIT 20;
    """
    out = _fetch_one_or_none(sql, {"zip": zip_i, "street": st, "hno": hno})
    if out:
        return out
  
```

Code 13: Tier 5: stadtname-unabhängige Suche, nur bei eindeutigen Treffern.

PLZ+Ort als Koordinatenquelle explizit ausgeschlossen. Eine PLZ-Ort-Kombination allein ist für eine kartografische Punktdarstellung zu ungenau und wird bewusst übersprungen. Für Gemeinde-Quellen ohne spezifische Adresse greift stattdessen die konfigurierte Fallback-Adresse aus der Source-Konfiguration (Kapitel 7.1).

7.6.2 Fazit

Die gestufte Matching-Strategie mit lokalem Swisstopo-Datensatz bewährt sich auch in dieser Iteration: Veranstaltungen mit vollständiger Adresse werden zuverlässig georeferenziert, mehrdeutige Treffer werden konservativ verworfen statt zu raten. Erkennbaren Verlust gibt es primär bei Events, deren Adresse vom LLM nur unvollständig extrahiert wurde – ein Problem, das auf der Extraktionsebene liegt, nicht auf der Geokodierungsebene. Für Gemeinde-Quellen ohne spezifische Adresse sichert die konfigurierte Fallback-Location Kapitel 7.1.4 einen kartografisch verwertbaren Eintrag, und der neue Reverse-Lookup schliesst die Lücke bei Quellen wie Guidle, die bereits Koordinaten liefern aber keine strukturierte Adresse.

Das georeferenzierte Event-Objekt wird anschliessend der Kategorie-Klassifikation übergeben.

7.7 Event Kategorisierung

Die Kategorisierung ordnet jedem Event automatisch genau eine von 23 vordefinierten Kategorien zu und setzt damit das Feld `category`. Sie läuft nach der Geo-Anreicherung und unmittelbar vor der Persistierung. Ziel der Stufe ist eine thematische Ordnung der Veranstaltungen. Diese ermöglicht im Frontend eine gezielte Filterung sowie eine kategorienabhängige Darstellung und unterstützt damit unmittelbar die Recherche- und Lagebeurteilungsfunktion des Systems. In der Studienarbeit wurden alle Events noch einheitlich behandelt. Eine kategorienbasierte Unterscheidung nach Event-Typen war dort ausdrücklich als Erweiterungsempfehlung formuliert [1, p. 46]. Die vorliegende Arbeit setzt diese Empfehlung als eigenständige Pipeline-Stufe um und ersetzt das zuvor frei durch das Sprachmodell befüllbare Kategoriefeld durch ein embeddingbasiertes Verfahren. Das Kapitel beschreibt zunächst die Einordnung in die Pipeline und die Abgrenzung zur Studienarbeit, anschliessend das Kategorienschema, die Initialisierung der Kategorie-Embeddings, die Zuweisung zur Laufzeit, die Fallback-Logik sowie abschliessend die Anbindung an Persistierung und Frontend.

7.7.1 Einordnung und Abgrenzung zur Studienarbeit

In der Studienarbeit existierte das Feld `category` zwar bereits im Datenmodell, wurde jedoch als freier Texteintrag durch das Sprachmodell befüllt, etwa mit Werten wie Konzert oder Konferenz [1, p. 30]. Eine systematische, auf ein festes Kategorienschema gestützte Zuweisung fand nicht statt. In der vorliegenden Arbeit bestimmt das Sprachmodell die Kategorie bewusst nicht mehr.

Die Zuweisung der endgültigen Kategorie übernimmt stattdessen eine eigene Stufe, der Category-Worker. Dies verhindert, dass das Modell frei formulierte und damit uneinheitliche Kategorien erzeugt, und stellt die Zuordnung auf ein definiertes Schema. Im aktiven Verarbeitungsfluss erhält der Geo-Worker die Events und gibt sie an die Kategorie-Stufe weiter, deren Worker das Ergebnis schliesslich an die Persistierung übergeben. Aktuell laufen vier Category-Worker parallel.

7.7.2 Kategorienschema

Die Kategorien sind in der Konstante `CATEGORY_DESCRIPTIONS` definiert und orientieren sich an den Event-Subtypen von Schema.org. Sie decken das Spektrum typischer Veranstaltungen ab und werden in der Tabelle `categories` gespeichert. Jede Kategorie besitzt einen internen englischen Namen, eine deutsche Beschreibung sowie einen vorberechneten Embedding-Vektor. Die Beschreibungen sind dabei nicht reiner Oberflächentext, sondern die semantische Grundlage der Kategorie-Embeddings. Je trennschärfer eine Kategorie beschrieben ist, desto zuverlässiger lässt sie sich in der späteren Ähnlichkeitssuche treffen.

Im Backend werden ausschliesslich die englischen Kategorienamen gespeichert. Das Frontend bildet diese auf deutsche Labels ab, die für Filterung und Darstellung verwendet werden. Tabelle 39 zeigt diese Zuordnung.

Interner Name (EN)	Frontend-Label (DE)
<code>BusinessEvent</code>	Geschäftsereignis
<code>ChildrensEvent</code>	Kinderveranstaltung
<code>ComedyEvent</code>	Comedyveranstaltung
<code>ConferenceEvent</code>	Konferenz
<code>CourseInstance</code>	Kursveranstaltung
<code>DanceEvent</code>	Tanzveranstaltung
<code>DeliveryEvent</code>	Lieferereignis
<code>EducationEvent</code>	Bildungsveranstaltung
<code>EventSeries</code>	Veranstaltungsreihe
<code>ExhibitionEvent</code>	Ausstellung
<code>Festival</code>	Festival
<code>FoodEvent</code>	Kulinarik
<code>Hackathon</code>	Hackathon
<code>LiteraryEvent</code>	Literaturveranstaltung
<code>MusicEvent</code>	Musikveranstaltung
<code>PerformingArtsEvent</code>	Darstellende Kunst
<code>PublicationEvent</code>	Publikationsereignis
<code>SaleEvent</code>	Verkaufsereignis
<code>ScreeningEvent</code>	Filmvorführung
<code>SocialEvent</code>	Gesellschaftliche Veranstaltung
<code>SportsEvent</code>	Sportveranstaltung
<code>TheaterEvent</code>	Theater
<code>VisualArtsEvent</code>	Bildkunst

Tabelle 39: Zuordnung der internen Kategorienamen zu den deutschen Frontend-Labels.

Die Datenbanktabelle hält den Embedding-Vektor als Spalte vom Typ `vector(1024)` vor. Die Dimension von 1024 muss mit dem eingesetzten Embedding-Modell übereinstimmen. Code 14 zeigt die für die Kategorisierung relevante Embedding-Spalte der Tabelle.

```
db/init/02_tables.sql SQL

CREATE TABLE categories (
  id          SERIAL PRIMARY KEY,
  name        VARCHAR(255) NOT NULL UNIQUE, -- interner EN-Name (Schema.org-Typ)
  description TEXT,                        -- deutsche Beschreibung als Embedding-Grundlage
  embedding   vector(1024)                -- vorberechneter Vektor
);
```

Code 14: Für die Kategorisierung relevante Embedding-Spalte der Tabelle `categories`

7.7.3 Initialisierung der Kategorie-Embeddings

Die Kategorie-Embeddings werden nicht bei jeder Zuweisung neu berechnet, sondern einmalig nach dem Einspielen oder Ändern der Beschreibungen erzeugt und persistent gespeichert. Die Funktion `init_category_embeddings()` lädt alle Kategorien ohne Embedding, bündelt ihre Beschreibungen im Format `Name: Beschreibung` und erzeugt die Vektoren über LLMHub in einem einzigen Batch-Aufruf mit dem Modell `best-embedding`. Die Reihenfolge der zurückgegebenen Vektoren wird dabei anhand des von der Schnittstelle gelieferten Index stabilisiert. Code 15 zeigt den entsprechenden Ablauf.

```
backend/database/categorysation.py PYTHON

async def init_category_embeddings() -> None:
    cur.execute("SELECT id, name, description FROM categories WHERE embedding IS NULL")
    rows = cur.fetchall()

    descriptions = [f"{r[1]}: {r[2]}" for r in rows]

    vectors = await embed_batch(descriptions)

    for cat_id, vector in zip(ids, vectors):
        cur.execute(
            "UPDATE categories SET embedding = %s::vector WHERE id = %s",
            ([" " + ", ".join(map(str, vector)) + "], cat_id),
        )
    conn.commit()
```

Code 15: Einmalige Berechnung und Speicherung der Kategorie-Embeddings

Für die Verwaltung der Kategorien stehen mehrere Befehle bereit, die sich auch gebündelt über einen entsprechenden Task ausführen lassen (Tabelle 40).

Befehl	Beschreibung
<code>reset:category:descriptions</code>	Kategorie-Beschreibungen in der Datenbank einspielen (UPSERT)
<code>clear:category:embeddings</code>	Alle Embeddings löschen, etwa nach einer Beschreibungsänderung
<code>init:category:embeddings</code>	Embeddings für alle Kategorien ohne Embedding berechnen

Tabelle 40: Befehle zur Verwaltung der Kategorie-Beschreibungen und -Embeddings.

7.7.4 Zuweisung zur Laufzeit

Im laufenden Betrieb liest ein Category-Worker ein Event aus der Queue und setzt dessen Kategorie zunächst sicherheitshalber zurück. Anschliessend bildet er den Anfragetext ausschliesslich aus Titel und Beschreibung im Format `Titel: Beschreibung`. Ort, Datum, Uhrzeit oder Quelle fliessen bewusst nicht in die Kategorisierung ein, da sie für die thematische Einordnung nicht aussagekräftig sind. Der Anfragetext wird in einen Vektor eingebettet und über den pgvector-Operator `<=>` mit den gespeicherten Kategorie-Embeddings verglichen. PostgreSQL gibt die Kategorie mit der kleinsten Vektordistanz und damit der grössten semantischen Ähnlichkeit zurück, wie Code 16 zeigt.

backend/database/categorysation.py
PYTHON

```

async def find_closest_category(text: str) -> str:
    vector = await embed(text)
    vec_str = "[" + ",".join(map(str, vector)) + "]"

    with get_db_connection() as conn, conn.cursor() as cur:
        cur.execute("""
            SELECT name
            FROM categories
            WHERE embedding IS NOT NULL
            ORDER BY embedding <=> %s::vector
            LIMIT 1
        """, (vec_str,))
        row = cur.fetchone()
    if row is None:
        raise RuntimeError("Keine Kategorien mit Embeddings in der Datenbank.")
    return row[0]

```

Code 16: Semantische Ähnlichkeitssuche der nächstgelegenen Kategorie

Da der Anfragetext im selben Format wie die Kategoriebeschreibungen aufgebaut wird, vergleicht die Suche semantisch vergleichbare Repräsentationen. Sind keine Kategorie-Embeddings vorhanden, löst die Funktion einen Fehler aus, der in der nachfolgenden Fallback-Logik aufgefangen wird.

7.7.5 Fallback-Kategorie

Schlägt die Ähnlichkeitssuche fehl, etwa wegen eines fehlenden API-Schlüssels, eines Netzwerkproblems, einer leeren Kategorientabelle oder eines Datenbankfehlers, weist der Worker die Kategorie `SocialEvent` zu, statt das Event zu blockieren (Code 17).



```

backend/pipeline/workers/category_worker.py PYTHON

try:
    category = await find_closest_category(query_text)
    ev.category = category
except Exception as e:
    logger.warning(
        "[CAT_WORKER %d] Kategorie-Lookup fehlgeschlagen für %r: %s – Fallback: %r",
        worker_id,
        title,
        e,
        FALLBACK_CATEGORY,
    )
    ev.category = FALLBACK_CATEGORY
  
```

Code 17: Robuste Zuweisung mit Fallback-Kategorie bei fehlgeschlagenem Lookup

Dieser Fallback sichert die Robustheit der Pipeline, ist fachlich jedoch eine Vereinfachung: Bei einem Ausfall der Embedding-Infrastruktur werden Events nicht verworfen, aber pauschal unter `SocialEvent` eingeordnet. Der Fallback ist zudem mit der Vollständigkeitsprüfung der Persistierung verknüpft. Ein Event ohne Kategorie würde dort als unvollständig markiert. Durch die garantierte Zuweisung einer Kategorie wird dieser Zustand vermieden.

7.7.6 Anbindung an Persistierung

Nach der Zuweisung schreibt der Worker das Event in die Stage-Datei der kategorisierten Events und reicht es anschliessend an die Persistierungsstufe weiter. Wie die übrigen Stufen protokolliert auch die Kategorisierung Beginn, Ergebnis und Schreibvorgang strukturiert, sodass die Zuordnung im Nachhinein nachvollziehbar bleibt. Die Datenbank gibt das Feld `category` an die API weiter, die eine kategorienbasierte, gross- und kleinschreibungsunabhängige Abfrage sowie eine Filterung über mehrere Kategorien hinweg ermöglicht.

7.7.7 Fazit

Mit der Kategorisierung wird die in der Studienarbeit lediglich empfohlene Event-Typisierung [1, p. 46] in eine eigenständige, kontrollierte Pipeline-Stufe überführt und das zuvor frei durch das Sprachmodell befüllte Kategoriefeld durch ein deterministisches Verfahren auf einem festen Schema ersetzt. Im praktischen Einsatz zeigt sich allerdings, dass sich einzelne Kategorien des Schema.org-Inventars inhaltlich stark überschneiden, etwa `SocialEvent` und `Festival` oder `PerformingArtsEvent` und `TheaterEvent`. Dadurch fallen Ereignisse mitunter in eine semantisch plausible, aus Nutzersicht jedoch nicht eindeutig erwartete Kategorie. Eine schärfere Abgrenzung der Kategoriebeschreibungen oder eine bewusst reduzierte, an den Anwendungsfall angepasste Auswahl dürfte die Zuordnungsqualität nach unserer Einschätzung weiter steigern und stellt damit den naheliegendsten Ansatzpunkt für zukünftige Verbesserungen dieser Stufe dar. Die so verbesserte Kategorisierung sollte zudem auf einem grösseren, ausgewogeneren Testdatensatz validiert werden, da der bisherige Gold-Datensatz die Kategorieverteilung noch nicht in allen realen Varianten gleichmässig abdeckt.

7.8 Deduplizierung und Persistenz

In der Semesterarbeit wurde die Deduplizierung ausschliesslich über zwei Datenbankconstraints gehandhabt [1, p. 20]: Für Events mit Quell-URL galt `(source_url, date, start_time)` als eindeutiger Schlüssel; ohne URL kam `(date, start_time, coordinates, title)` zum Einsatz. Diese Lösung hatte zwei strukturelle Schwächen. Erstens liess sich ein Event nicht aktualisieren, wenn sich der Inhalt hinter einer gleichbleibenden URL änderte – der Constraint verhinderte jeden neuen Eintrag, ohne den alten zu überschreiben. Zweitens konnten gleichartige Felder bei unabhängigen Veranstaltungen am gleichen Ort und Datum – etwa mehrere Events an einer Messe oder Hochschule – fälschlicherweise als Duplikat erkannt werden [1, p. 46].

Die neue Lösung verlagert die Deduplizierungslogik vollständig in die Anwendungsschicht (`database/db.py`, Funktion `persist_event()`). Damit kann differenzierter entschieden werden: nicht nur ob ein Event doppelt vorliegt, sondern auch *woher* das Duplikat stammt und ob eine Aktualisierung angebracht ist.

Jedes Event erhält vor dem Datenbankvergleich einen **Fingerprint** – ein normalisiertes Tupel aus elf stabilen Feldern (Code 18) – sowie eine **kanonische URL**, bei der Tracking-Parameter entfernt und Gross-/Kleinschreibung vereinheitlicht werden, sodass zwei URLs derselben Seite als gleich erkannt werden.

```
backend/database/db.py PYTHON

_FINGERPRINT_FIELDS = [
    "title", "date", "start_time", "end_time",
    "full_location", "city_name", "city_zip",
    "street", "street_number",
    "coordinates", "event_status",
]
```

Code 18: Felder, die den Event-Fingerprint bilden.

7.8.1 Persistenzentscheidung

Die Persistenzfunktion lädt alle bestehenden Events desselben Datums und vergleicht deren Fingerprint mit dem neuen Event. Stimmt kein Fingerprint überein, folgt ein URL-basierter Abgleich. Aus dem Ergebnis leiten sich die in Tabelle 41 dargestellten Aktionen ab.

Aktion	Bedingung	Datenbankwirkung
deduped	Gleicher Fingerprint und gleiche kanonische Quell-URL	Keine Änderung
merged	Gleicher Fingerprint, aber andere Quell-URL	Keine Änderung
unchanged	Gleiche Quell-URL, stabile Felder unverändert	Keine Änderung
updated	Gleiche Quell-URL, mindestens ein stabiles Feld verändert	Bestehende Zeile überschrieben
inserted	Keine Übereinstimmung	Neuer Datensatz eingefügt

Tabelle 41: Mögliche Ergebnisse der Persistenzentscheidung und ihre jeweilige Datenbankwirkung.

7.8.2 Anwendungsbeispiele

Merge. Ein gutes Beispiel für erfolgreiche Zusammenführung zeigt die Veranstaltung «Mütterberatung mit Familientreff» in Andwil. Dieselbe Veranstaltung wurde auf der Gemeinde-Website zweimal angelegt – mit leicht unterschiedlichem Beschreibungstext, aber identischem Fingerprint (Abbildung 10). Das System erkennt den gleichen Fingerprint, stellt fest, dass die zweite URL eine andere ist als die erste, und verwirft den Eintrag als **merged** (Abbildung 11). Die Grundidee dahinter: **deduped**-Einträge entstehen typischerweise bei erneuten Crawls derselben Quelle, **merged**-Einträge dagegen während desselben Crawl-Laufs, wenn verschiedene Quellen das gleiche Event listen.

Mütterberatung mit Familientreff

Othmarzentrum
Arneggerstrasse 3
9204 Andwil
29. Juni 2026, 14.30 Uhr - 16.30 Uhr, 2 Stunden

Beratung durch eine Fachperson der Mütter- und Väterberatung, parallel findet der Familientreff statt.

Kontakt

Mütterberatung mit Familientreff

Othmarzentrum
Arneggerstrasse 3
9204 Andwil
29. Juni 2026, 14.30 Uhr - 16.30 Uhr, 2 Stunden

Beratung durch eine Fachperson der Mütterberatung, parallel findet der Familientreff statt.

Jeden 1. Montag im Monat
ohne Voranmeldung: 14.30-16.30 Uhr
mit Voranmeldung: 13.30-14.30 oder 16.30-17.00 Uhr

Abbildung 10: Die Veranstaltung «Mütterberatung mit Familientreff» erscheint auf zwei verschiedenen Seiten der Gemeinde-Website mit leicht abweichendem Beschreibungstext, aber identischem Fingerprint.

29.06.2026	Mütterberatung mit Familientreff	Othmarzentrum	Verein Mütter- und Väterberatung Untertoggenburg-Wil-Gossau
29.06.2026	Mütterberatung mit Familientreff	Othmarzentrum	Verein Mütter- und Väterberatung Untertoggenburg-Wil-Gossau

Abbildung 11: Nur ein Eintrag in der Datenbank: Der zweite Fund wurde als **merged** verworfen.

Aktualisierung. Der `updated`-Pfad greift nur bei Events, deren Quell-URL eindeutig einer einzelnen Veranstaltung entspricht. Quellen, die den MD-Splitter verwenden oder aus denen das LLM mehrere Events in einem Durchgang extrahiert, werden von der Aktualisierungslogik ausgenommen (Code 19). Der Grund: Diese Quellen teilen sich eine URL über mehrere unterschiedliche Events hinweg. Das Lesecafé der Bibliothek Ludothek Degersheim ist ein typisches Beispiel – gleicher Titel, gleiche Uhrzeit, gleiche Quell-URL, aber unterschiedliche Termine (Abbildung 12). Ohne die Ausnahme würde der spätere Termin den früheren schlicht überschreiben, statt als eigenständiger Eintrag gespeichert zu werden.

```

backend/database/db.py PYTHON

def _source_url_allows_update(event_data: dict) -> bool:
    return not (
        _is_split_source_event(event_data)
        or _is_multi_event_source_event(event_data)
    )

```

Code 19: Update-Logik wird für Split- und Multi-Event-Quellen deaktiviert.

29.06.2026 Lesecafé ›

Ort: Bibliothek Ludothek Degersheim
Zeit: 09.00 Uhr
Organisator: Bibliothek Ludothek Degersheim
Kontakt: Marianne Hälg

24.08.2026 Lesecafé ›

Ort: Bibliothek Ludothek Degersheim
Zeit: 09.00 Uhr
Organisator: Bibliothek Ludothek Degersheim
Kontakt: Marianne Hälg

Abbildung 12: Zwei Lesecafé-Einträge mit gleicher URL, gleichem Titel und gleicher Uhrzeit, aber unterschiedlichem Datum – korrekt als separate Events gespeichert.

7.8.3 Verbleibende Herausforderungen

Auch nach der URL-Ausnahme gibt es Fälle, die der Fingerprint-Logik entgehen. Variiert der Titel minimal – sei es durch einen Tippfehler auf der Quell-Website oder durch leicht abweichende Formulierung – entstehen zwei separate Einträge. Das Beispiel «Grillabend bei Zeifels» und «Grillabend bei Zweifels» (Abbildung 13) zeigt, dass ohne manuellen Eingriff nicht determiniert werden kann, ob es sich um denselben Event mit Tippfehler oder um zwei tatsächlich verschiedene Veranstaltungen handelt.

26.06.2026 Grillabend bei Zeifels ›

26.06.2026 Grillabend bei Zweifels ›

Ort: Familie Zweifel
Zeit: 18.00 Uhr
Organisator: Natur und Vogelschutz
Kontakt: E.Weinmann

✉ [E-Mail schreiben](#)
 🌐 nuvsdegersheim.ch
 Anmeldung erforderlich

Abbildung 13: «Grillabend bei Zeifels» und «Grillabend bei Zweifels» – minimale Titelabweichung verhindert Zusammenführung, obwohl es sich vermutlich um dasselbe Event handelt.

Dasselbe Problem tritt bei quellübergreifenden Duplikaten auf. Die «KI-Entdeckungsreise für ältere Einsteigerinnen und Einsteiger» erscheint auf Guidle mit diesem vollständigen Titel, auf Eventfrog hingegen als «KI-Entdeckungsreise für ältere Einsteigende» (Abbildung 14). Die unterschiedlichen Fingerprints verhindern eine Zusammenführung. Eine rein deterministische Deduplizierung kann diesen Fall nicht sicher lösen, da gleiche Zeit und Ort bei unterschiedlichem Titel ebenso auf zwei separate Veranstaltungen im selben Gebäude hinweisen könnte.



Abbildung 14: Gleiche Veranstaltung auf Guidle und Eventfrog mit leicht abweichendem Titel – beide Einträge verbleiben in der Datenbank.

7.8.4 Fazit

Die Verlagerung der Deduplizierung in die Anwendungsschicht ermöglicht eine differenzierte Persistenzentscheidung, die über das reine Verhindern von Duplikaten hinausgeht: Events können aktualisiert, quellübergreifend zusammengeführt oder als unverändert erkannt werden. Der fachliche Fingerprint kombiniert mit der kanonischen URL deckt den Grossteil der Fälle zuverlässig ab; die Ausnahme für Split- und Multi-Event-Quellen verhindert, dass Folge-Crawls eigenständige Termine überschreiben. Wo Titel minimal abweichen oder Quellplattformen unterschiedliche Formulierungen verwenden, stösst der deterministische Ansatz jedoch an seine Grenzen. Eine weiterführende Arbeit könnte hier einen semantischen, vektorbasierten Ansatz evaluieren: Die Kombination von Fingerprint-Informationen mit einem Ähnlichkeitsvergleich der Eventbeschreibung via Embedding liesse sich potenzielle Duplikate identifizieren, ohne tatsächlich verschiedene Veranstaltungen am gleichen Ort fälschlicherweise zu unterdrücken. Darüber hinaus sollte die Deduplizierung um eine explizite Quellenverwaltung erweitert werden, damit alle Quellen eines konsolidierten Events dauerhaft am Event selbst nachvollziehbar bleiben. Ebenso wäre ein eigener Gold-Datensatz für die Deduplizierung sinnvoll, um den Anteil korrekt zusammengeführter und korrekt getrennt gehaltener Events quantitativ zu messen; im Rahmen dieser Arbeit wurde die Deduplizierung anhand konkreter Testfälle, jedoch noch nicht als Precision-/Recall-Metrik ausgewertet.

Das folgende Kapitel beschreibt das Datenbankschema, in das die so aufbereiteten Events persistiert werden.

7.9 Datenbank

Die PostgreSQL-Datenbank bildet die zentrale, strukturierte Ablage der aufbereiteten Veranstaltungen und zugleich die Referenzbasis für Kategorisierung und Geokodierung. Gegenüber der Studienarbeit, deren Datenmodell den Ort in einem einzelnen Freitextfeld führte und die Deduplizierung über zwei Unique-Constraints auf Tabellenebene löste [1, p. 20], wurde das Schema in mehreren Punkten erweitert und die Eindeutigkeitslogik in die Anwendungsschicht verlagert Kapitel 7.8. Neu sind insbesondere strukturierte Adressfelder, ein Quelltyp- sowie ein Datenqualitätsfeld, eine zusätzliche Gemeinde-Referenztable und die Embedding-Spalte für die automatische Kategorisierung. Dieses Kapitel beschränkt sich auf die implementierungsrelevanten Aspekte, namentlich den Aufbau und die Initialisierung der Datenbank, die Begründung der Schemaerweiterungen gegenüber der Studienarbeit sowie die geografischen Integritäts-Constraints zur Durchsetzung der Kantonsbeschränkung. Die vollständige, feldweise Schemadefinition der Tabellen ist Bestandteil der Architekturdokumentation.

7.9.1 Aufbau und Initialisierung

Die PostgreSQL-Datenbank läuft als Docker-Container `postgres_db` (Image `pgvector/pgvector:pg18-trixie`, erweitert um `pg_textsearch`, siehe `db/dockerfile`). Beim allerersten Start eines neuen Datenvolumes führt Postgres automatisch alle SQL-Skripte unter `db/init/` in alphanumerischer Reihenfolge aus. Tabelle 42 gibt einen Überblick über diese Skripte und ihren jeweiligen Zweck.

Datei	Zweck
<code>01_extensions.sql</code>	Aktiviert die benötigten Postgres-Extensions
<code>02_tables.sql</code>	Erstellt die Tabellen <code>categories</code> , <code>gemeinde_geocoding</code> , <code>events</code>
<code>03_constraints.sql</code>	Fügt den ZIP-Allowlist-Constraint auf <code>events.city_zip</code> hinzu
<code>04_indices.sql</code>	Erstellt die Performance- und Such-Indizes
<code>05_comments.sql</code>	Hinterlegt Tabellen-Kommentare
<code>06_seed_data.sql</code>	Lädt Test-Events und die Gemeinde-Geokoordinaten

Tabelle 42: Initialisierungsskripte der Datenbank unter `db/init/` und ihr jeweiliger Zweck.

Aktiviert werden drei Erweiterungen, die in Code 20 dargestellt sind.


db/init/01_extensions.sql
SQL

```

CREATE EXTENSION IF NOT EXISTS vector;
CREATE EXTENSION IF NOT EXISTS pg_textsearch;
CREATE EXTENSION IF NOT EXISTS pg_trgm;

```

Code 20: Aktivierte PostgreSQL-Erweiterungen

Die Erweiterung `vector` wird für die Embedding-Spalte der Kategorien und die Ähnlichkeitssuche benötigt, `pg_textsearch` für die BM25-basierte Titelsuche; `pg_trgm` ist für eine trigramm-basierte Textsuche vorbereitet, im aktuellen Stand jedoch nicht zentral im Einsatz. Über die Initialisierungsskripte werden drei Tabellen direkt angelegt: `events` als finale Zieltabelle, `categories` für die Kategorisierung und `gemeinde_geocoding` als grobe Gemeinde-Referenz. Die präzise Adresstabelle `swiss_addresses` wird hingegen nicht über `db/init` erstellt, sondern separat über dedizierte Backend-Tasks vorbereitet und geladen. Der Grund dafür ist, dass die

`swiss_addresses` -Tabelle mit rund 3.5 Millionen Einträgen deutlich umfangreicher ist als die anderen Tabellen und deshalb eine eigenständige Initialisierung erfordert.

7.9.2 Schemaerweiterungen gegenüber der Studienarbeit

Im folgenden werden die wichtigsten Schemaerweiterungen der `events` -Tabelle gegenüber der Studienarbeit beschrieben und begründet. Das vollständige Schema ist im Kapitel 4.5.1 dokumentiert.

Das ursprüngliche `location` -Feld als einzelnes Freitextfeld wurde durch eine Kombination aus `full_location` und strukturierten Adressfeldern ersetzt. `full_location` bewahrt den Originaltext der Quelle und dient als Fallback. Die Einzelfelder ermöglichen präzises Filtern nach Region oder Stadt im Cockpit sowie eine verbesserte Trefferquote bei der Geokodierung gegen die Swiss-Addresses-Tabelle.

Der ISO-Ländercode `country_short_id` erlaubt eine einfache, sprachunabhängige Filterung nach Land und ist kompatibel mit gängigen Bibliotheken und Kartendiensten.

Das Feld `description` wurde ergänzt, damit im Cockpit neben dem Titel auch eine kurze inhaltliche Vorschau angezeigt werden kann, ohne dass die Quelle erneut abgerufen werden muss. Diese Description wird auch für die semantische Kategorisierung verwendet.

`event_status` ermöglicht es, abgesagte oder verschobene Veranstaltungen gezielt zu kennzeichnen und im Cockpit anzuzeigen.

`completion_status` unterscheidet vollständig erfasste Events von Datensätzen, bei denen einzelne Felder fehlen..

`updated_on` hält fest, wann ein Datensatz zuletzt aktualisiert wurde. Dies erleichtert inkrementelle Scraping-Läufe, bei denen nur veränderte oder neu hinzugekommene Events verarbeitet werden sollen.

`category` speichert die Schema.org-Eventkategorie als Freitextfeld.

`source_type` hält fest, von welcher Crawling-Quelle ein Event stammt. Das Feld erlaubt es, Events im Cockpit nach Quelle zu filtern.

In der SA [1, p. 20] sicherten zwei Unique-Constraints die Eindeutigkeit von Events ab. In der BA wird darauf verzichtet. Die Duplikaterkennung übernimmt deshalb die Crawler-Logik, die bei jedem Lauf die vorhandenen Events mit den neu gesammelten Daten abgleicht und nur tatsächlich neue oder geänderte Events in die DB schreibt.

7.9.3 Geografische Einschränkung auf den Kanton St.Gallen

Während die Deduplizierung bewusst in die Anwendungsschicht verlagert wurde, eignen sich Datenbank-Constraints für die Durchsetzung einer harten, nicht-funktionalen Anforderung: Das System soll ausschliesslich Veranstaltungen aus dem Kanton St.Gallen aufnehmen (Kapitel 6.1.4). Diese Einschränkung wird über zwei `CHECK` -Constraints auf der Tabelle `events` erzwungen.

Der erste Constraint lässt ausschliesslich Postleitzahlen zu, die dem Kanton St.Gallen zugeordnet sind. Events ohne Postleitzahl (`NULL`) bleiben zulässig, da deren Zuordnung gegebenenfalls erst zu einem späteren Zeitpunkt erfolgt.

Der zweite Constraint stellt sicher, dass Events mit Koordinaten zwingend eine Postleitzahl besitzen (Code 21). Die zugrunde liegende Überlegung ist, dass die Geo-Anreicherung für gültige Schweizer Koordinaten vor der Persistierung eine Schweizer Postleitzahl auflösen oder erhalten kann. Ein Event mit Koordinaten, aber ohne Postleitzahl, stammt daher mit hoher Wahrscheinlichkeit aus dem Ausland, ohne diesen Constraint würde es dennoch auf der Karte erscheinen, da es über Koordinaten verfügt.

```
db/init/03_constraints.sql SQL

ALTER TABLE events
ADD CONSTRAINT events_coordinates_require_city_zip CHECK (
  coordinates IS NULL
  OR city_zip IS NOT NULL
);
```

Code 21: Constraint, der Koordinaten ohne zugehörige Postleitzahl ausschliesst

Auch diese Constraints arbeiten nicht vollständig fehlerfrei; in seltenen Fällen passieren einzelne Events die Prüfung, obwohl sie geografisch ausserhalb des Kantons liegen. Abbildung 15 zeigt ein solches Beispiel.

22.08.2026 – Oberriet – Religion



Rheintaler Wallfahrt

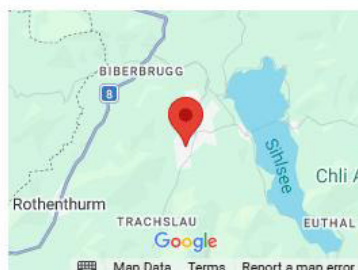
(Töffliwallfahrt) nach Einsiedeln

Wann

22.08.2026 08:00 bis 17:00

Wo

Einsiedeln
Klosterplatz Einsiedeln
9463 Oberriet



Kontakt

Sekretariat Seelsorgeeinheit
Blattenberg
0041 71 761 11 38
sekretariat@se-blattenberg.ch
<https://www.se-blattenberg.ch/de/seelsorgeeinheit-blattenberg/veranstaltungen>

Abbildung 15: Gesamtansicht des Cockpits.

Im dargestellten Fall der Veranstaltung „Töffliwallfahrt nach Einsiedeln“, hinterlegte der Veranstalter auf der Eventseite die Koordinaten des Zielorts Einsiedeln, behielt in der Adresse jedoch die Postleitzahl des Startorts bei 9463 Oberriet. Da diese Postleitzahl gültig und dem Kanton St.Gallen zugeordnet ist, wird sie durch die Geokodierung nicht verändert, sodass das Event den Constraint passiert, obwohl die Koordinaten nach Einsiedeln im Kanton Schwyz verweisen. Eine mögliche Gegenmassnahme bestünde darin, systematisch die Koordinaten als massgebliche Quelle zu betrachten; im vorliegenden Fall würde dies die Sachlage jedoch ebenfalls nicht vollständig abbilden, da der Startort der Veranstaltung tatsächlich im Kanton St.Gallen liegt.

7.9.4 Fazit

Das Datenbankschema wurde gegenüber der Studienarbeit gezielt erweitert, um den gestiegenen Anforderungen des Cockpits gerecht zu werden: Strukturierte Adressfelder, Quell- und Statusinformationen sowie die Embedding-Spalte schaffen die Grundlage für präzises Filtern, semantische Kategorisierung und eine verlässliche Geokodierung. Die Verlagerung der Deduplizierung in die Anwendungsschicht erhöht die Flexibilität, während die `CHECK`-Constraints die harte Beschränkung auf den Kanton St.Gallen weitgehend zuverlässig durchsetzen. Einzelne Grenzfälle wie die Töffliwallfahrt zeigen jedoch, dass eine rein datenbankseitige Validierung an inhaltlich widersprüchlichen Quelldaten ihre Grenzen findet.

7.10 API und Datenbereitstellung

Der API-Service bildet das Bindeglied zwischen der PostgreSQL-Datenbank und dem Svelt-eKit-Frontend und stellt die persistierten Veranstaltungsdaten über RESTful-Endpunkte bereit. Neben dem Cockpit-Frontend können darüber auch externe Dienste der Kantonspolizei St.Gallen strukturiert auf die Daten zugreifen. Eine solche Lese-Schnittstelle zwischen Datenbank und Frontend bestand bereits in der Studienarbeit [1, p. 21]. In der vorliegenden Arbeit wurde sie um eine serverseitige Filterung nach Kategorie und Quellentyp, eine Volltextsuche sowie einen Exportendpunkt erweitert.

7.10.1 Rolle und Technologie-Stack

Der Datenfluss verläuft von der Pipeline über die Datenbank und den API-Service bis zum Frontend beziehungsweise zu externen Diensten. Der API-Service liest dabei ausschliesslich aus der Datenbank und nimmt selbst keine Schreibvorgänge an Eventdaten vor.

7.10.2 Datenbankankbindung

Statt für jede Anfrage eine neue Verbindung zu öffnen, verwendet der Dienst einen Connection Pool, der beim ersten Zugriff initialisiert und anschliessend wiederverwendet wird. Dadurch sinkt der Verbindungsaufwand und die API bleibt auch unter Last stabil (Code 22).


api_service/db.py
PYTHON

```

_pool: psycopg_pool.ConnectionPool | None = None

def _get_pool() -> psycopg_pool.ConnectionPool:
    global _pool
    if _pool is None:
        url = os.getenv("DATABASE_URL")
        _pool = psycopg_pool.ConnectionPool(url, min_size=1, max_size=10, open=True)
    return _pool

```

Code 22: Lazy initialisierter Connection Pool für den Datenbankzugriff

Für alle Frontend-Abfragen werden nur die tatsächlich benötigten Felder zurückgegeben, um die übertragene Datenmenge zu reduzieren. Diese Feldliste ist zentral definiert (Code 23).


api_service/db.py
PYTHON

```

FIELDS_FOR_FRONTEND = "id, description, city_name, title, date, start_time, end_time, full_location, category, source_type, event_status, completion_status, updated_on, coordinates, source_url"

```

Code 23: Zentrale Feldliste der an das Frontend zurückgegebenen Spalten

Das vollständige Datenbankschema der Events-Tabelle ist im Kapitel 4.5.1 beschrieben.

7.10.3 Gefilterter Hauptendpunkt: GET /events/

Der wichtigste Endpunkt für unser Frontend ist `GET /events/`. Er lädt alle Events eines Zeitraums und unterstützt optional die Filterung nach Stadt, Kategorie und Quellentyp sowie eine Volltextsuche in einer einzigen Abfrage. Tabelle 43 listet die unterstützten Abfrageparameter. Wobei die Stadtfilterung im Moment nicht eingesetzt ist doch als mögliche Erweiterung in der API schon umgesetzt ist.

Parameter	Typ	Pflicht	Beschreibung
<code>start</code>	YYYY-MM-DD	Ja	Startdatum des Zeitraums
<code>end</code>	YYYY-MM-DD	Ja	Enddatum des Zeitraums
<code>city</code>	string[]	Nein	Ortsname(n), case-insensitive
<code>category</code>	string[]	Nein	Kategorie(n), z.B. MusicEvent
<code>source_type</code>	string[]	Nein	Quellentyp(en): eventseiten, gemeindeseiten
<code>q</code>	string	Nein	Suchbegriff für die Titelsuche

Tabelle 43: Abfrageparameter des Hauptendpunkts `GET /events/`.

Die Antwort folgt stets derselben Struktur und kapselt die Eventliste samt Metadaten zum abgefragten Zeitraum (Code 24).

`api_service/main.py`
PYTHON

```

return {
    "success": True,
    "count": len(events),
    "period": {
        "start": start.isoformat(),
        "end": end.isoformat()
    },
    "events": events
}
```

Code 24: Einheitliche Antwortstruktur des Hauptendpunkts

Das Frontend baut die Anfrage in einer zentralen Ladefunktion auf und übernimmt aus der Antwort ausschliesslich das Feld `events` in den globalen Store. Die SQL-Abfrage wird dynamisch aus Bedingungen zusammengesetzt, wobei jeder optionale Parameter die `WHERE`-Klausel erweitert. Mehrfachauswahlen bei Kategorie und Quellentyp werden über `ANY` abgebildet (Code 25), sodass mehrere `category`- oder `source_type`-Parameter zulässig sind.

```

api_service/db.py PYTHON

if category:
    conditions.append("LOWER(category) = ANY(%s)")
    params.append([c.strip().lower() for c in category])

if source_type:
    conditions.append("LOWER(source_type) = ANY(%s)")
    params.append([s.strip().lower() for s in source_type])

```

Code 25: Serverseitige Mehrfachfilterung nach Kategorie und Quellentyp

Eine vom Frontend erzeugte Anfrage kann damit beispielsweise mehrere Kategorien und Quellentypen für einen Zeitraum kombinieren.

7.10.4 Volltextsuche

Die Suche im Frontend läuft nicht über einen separaten Suchendpunkt, sondern über denselben Hauptendpunkt mit dem Parameter `q`. Die Eingabe wird entprellt und löst anschliessend eine erneute Abfrage aus. Liegt ein Suchbegriff vor, ergänzt das Backend die Bedingung um einen `title`-Filter und sortiert die Treffer über den BM25-Index `idx_title_search` nach Relevanz; die Ergebnismenge ist dabei begrenzt (Code 26).

```

api_service/db.py PYTHON

if q:
    q_clean = q.strip()
    conditions.append("title ILIKE %s")
    params.append(f"%{q_clean}%")
    where = " AND ".join(conditions)
    sql = f"""
        SELECT {FIELDS_FOR_FRONTEND}
        FROM events
        WHERE {where}
        ORDER BY title <@> to_bm25query(%s, 'idx_title_search') ASC
        LIMIT 15
    """

```

Code 26: Titelsuche mit BM25-Relevanzsortierung über den `pg_textsearch`-Index

7.10.5 Exportfunktion

Ergänzend stellt der Dienst einen Exportendpunkt `GET /events/export` bereit, der dieselben Filterparameter wie `GET /events/` akzeptiert und die gefilterten Events über den Parameter `format` als CSV- oder JSON-Datei zurückgibt. Die Antwort ist eine `StreamingResponse` mit einem `Content-Disposition`-Header; im Frontend wird der Export nicht über eine reguläre Anfrage, sondern über einen dynamisch erzeugten Download-Link ausgelöst.

7.10.6 Fazit

Der API-Service stellt mit dem gefilterten Hauptendpunkt, der Volltextsuche und der Exportfunktion alle Schnittstellen bereit, die das Cockpit-Frontend für die Anzeige, Suche und Weiterverarbeitung der Eventdaten benötigt. Die Erweiterungen gegenüber der Studienarbeit umfassen die serverseitige Mehrfachfilterung, die BM25-basierte Titelsuche sowie den CSV- und JSON-Export und wurden in diesem Kapitel im Detail beschrieben.

Darüber hinaus existieren in der FastAPI-Anwendung weitere Endpunkte (z.B. für Hilfsabfragen), die im aktuellen Frontend nicht aktiv eingesetzt werden. Diese werden hier nicht weiter beschrieben, ihre vollständige Definition inklusive Parameter und Antwortschema ist über die automatisch generierte Swagger-Dokumentation des Dienstes einsehbar.

7.11 Frontend-Umsetzungen

Das Frontend bildet die Präsentationsschicht des Cockpits und stellt die aufbereiteten Veranstaltungsdaten interaktiv dar. Es ist als Single-Page-Application mit SvelteKit umgesetzt und ermöglicht die räumliche und zeitliche Exploration der Events auf einer Leaflet-Karte. Damit führt es den bereits in der Studienarbeit gewählten Ansatz einer kartenbasierten, von der Datenverarbeitung entkoppelten Darstellung fort [1, p. 37], erweitert ihn jedoch um ein Filterpanel für Kategorie, Quellentyp und Uhrzeit, eine Titelsuche, eine kategorieabhängige Markerdarstellung mit Clustering sowie eine Gebietsfilterung anhand polizeilicher Zonen. Das Kapitel beschreibt zunächst den Aufbau und den Datenfluss der Anwendung, anschliessend die Karten- und Markerdarstellung, die Suche sowie die einzelnen Filter und den Export. Die Gesamtansicht des Cockpits ist in Abbildung 16 dargestellt.

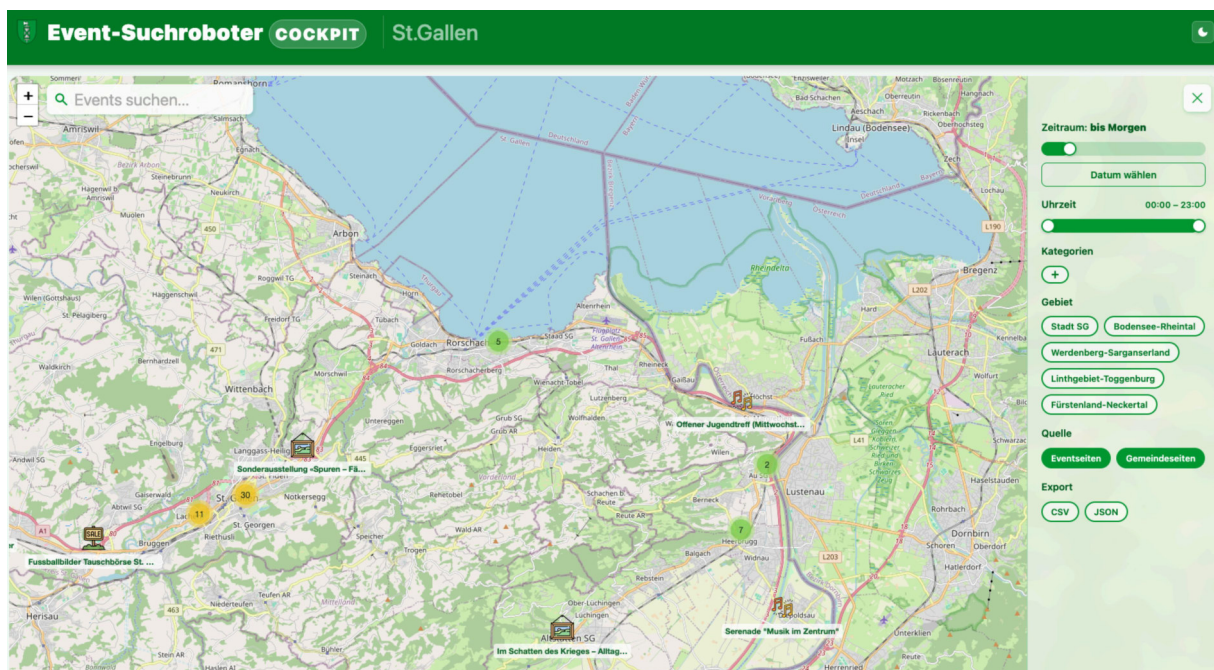


Abbildung 16: Gesamtansicht des Cockpits.

7.11.1 Aufbau der Komponentenhierarchie

Die Hauptkomponente `App.svelte` rendert die statische Navigation mit Logo und Titel sowie die eigentliche Kartenkomponente `Map.svelte`. Das Filterpanel ist als eigenständige Komponente `FilterComponent.svelte` implementiert und wird innerhalb von `Map.svelte` eingebettet, um direkten Zugriff auf die Filterzustände zu ermöglichen. Die Daten werden über zentrale Svelte-Stores verwaltet, die von beiden Komponenten abonniert werden können.

7.11.1.1 Karte und Marker-Clustering

Die Karte wird mit Leaflet initialisiert und verwendet OpenStreetMap als Kartengrundlage. Die Event-Marker werden nicht einzeln auf die Karte gesetzt, sondern einer `MarkerClusterGroup` hinzugefügt, die nahe beieinanderliegende Marker zusammenfasst, zu sehen in Abbildung 17. Beim Hineinzoomen lösen sich die Cluster auf. Liegen mehrere Events nahezu am selben Ort, werden die Marker beim Öffnen eines Clusters kreisförmig aufgefächert. Dieses Verhalten stammt aus dem Standardumfang von `leaflet.markercluster` und adressiert unmittelbar die in der SA als Erweiterungsempfehlung benannte unübersichtliche Darstellung mehrerer Events an identischen Koordinaten [1, p. 46].

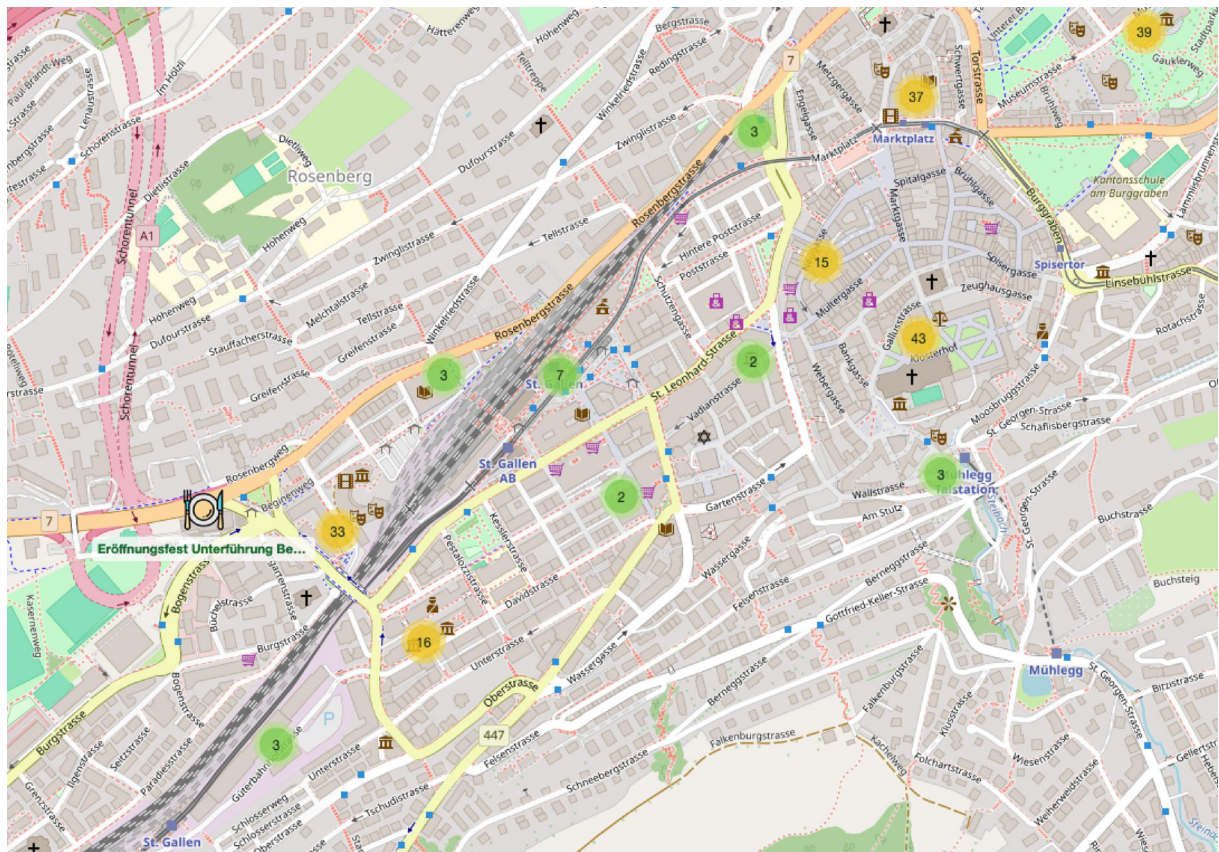


Abbildung 17: Cluster-Darstellung der Events auf der Karte.

Jede Kategorie besitzt ein eigenes, an Schema.org orientiertes Marker-Icon im Format 36×36 Pixel. Beim Rendern dient `event.category` als Schlüssel für die Zuordnung. Existiert ein passendes Icon, wird es als Leaflet-Icon gesetzt, andernfalls greift der Standardmarker. Der Ankerpunkt platziert dabei die Spitze des Icons auf den exakten Koordinaten des Events (Code 27).

```
frontend/src/lib/Map.svelte TYPESCRIPT  
  
const iconUrl = categoryIconMap[event.category];  
const markerOptions = iconUrl  
  ? {  
    icon: L.icon({ iconUrl, iconSize: [36, 36], iconAnchor: [18, 36], popupAnchor: [0, -36] }),  
    riseOnHover: true,  
  }  
  : { riseOnHover: true };
```

Code 27: Kategorieabhängige Marker-Icons mit definiertem Ankerpunkt

Jeder Marker zeigt ein dauerhaftes Tooltip mit dem Titel sowie ein Popup, das Beschreibung, Datum, Start- und Endzeit, Ort, Kategorie, Status, Quelle und den Zeitpunkt der letzten Aktualisierung enthält.

7.11.1.2 Suche

Das Suchfeld liegt über der Karte. Eingaben werden mit einer Verzögerung von 300 Millisekunden entprellt und lösen anschliessend einen erneuten Aufruf von `loadEvents()` mit dem Suchbegriff aus. Die Suche ist damit serverseitig: Das Backend filtert nach dem Titel und sortiert die Treffer nach BM25-Relevanz (Code 28).

```
frontend/src/lib/Map.svelte TYPESCRIPT  
  
function onSearchInput(e: Event) {  
  const q = (e.target as HTMLInputElement).value;  
  searchQuery = q;  
  dropdownOpen = true;  
  if (searchDebounce) clearTimeout(searchDebounce);  
  searchDebounce = setTimeout(() => runSearch(q), 300);  
}
```

Code 28: Entprellte Sucheingabe vor dem serverseitigen Suchaufruf

Die Treffer werden als Dropdown angezeigt. Ein Klick auf ein Resultat bewegt die Karte zum gewählten Event, sofern dessen Koordinaten einem gültigen Format entsprechen und somit eine gültige Position auf der Karte darstellen.

7.11.1.3 Filterpanel

Das Filterpanel, in Abbildung 18 zu sehen, ist als schmale, halbtransparente Seitenleiste mit Glassmorphism-Effekt umgesetzt und gliedert sich in die Bereiche Zeitraum, Uhrzeit, Kategorie, Quelle sowie Export.

Abbildung 18: Filterbereich

Der Zeitraumfilter bietet zwei umschaltbare Modi. Im Schnellmodus wählt ein einstufiger Slider zwischen acht vordefinierten Zeiträumen von „Heute“ bis „1 Jahr“, im Datumsmodus stehen zwei native Datumsfelder für eine freie Eingabe von Start- und Enddatum zur Verfügung. Die aktive Auswahl wird über den `filterStore` bereitgestellt. Der Uhrzeitfilter ist ein Zwei-Punkt-Schieberegler von 0 bis 23 Uhr. Im Unterschied zu Zeitraum, Kategorie und Quelle wird er nicht an die API übergeben, sondern filtert ausschliesslich die auf der Karte dargestellten Marker, indem die Stunde der Startzeit gegen den gewählten Bereich geprüft wird.

Die Kategoriauswahl erfolgt über eine Mehrfachauswahl mit Chips. Aktive Kategorien werden als gefüllte Chips dargestellt und lassen sich per Klick entfernen, ein zusätzliches Dropdown bietet die übrigen Kategorien an. Intern werden die englischen, an Schema.org angelehnten Schlüssel gespeichert, passend zum Wert aus der API, während die Anzeige in der Benutzeroberfläche auf Deutsch erfolgt. Die Quellenauswahl unterscheidet zwischen `Eventseiten` und `Gemeindeseiten`, die als umschaltbare Chips dargestellt und standardmässig beide aktiv sind. Die angezeigten Bezeichnungen werden vor der Übergabe an die API in die internen Datenbankwerte übersetzt.

7.11.1.4 Gebietsfilter anhand polizeilicher Zonen

Eine fachlich für die polizeiliche Nutzung relevante Erweiterung ist der Gebietsfilter, wie in Abbildung 19 zu sehen. Er beruht auf KML-Dateien, die fünf Zonen abbilden: die Stadt St.Gallen, Bodensee-Rheintal, Werdenberg-Sarganserland, Linthgebiet-Toggenburg und Fürstenland-Neckertal. Diese Dateien werden aus dem statischen Verzeichnis des Frontends geladen, in Polygon-Geometrien umgewandelt und lokal gehalten.

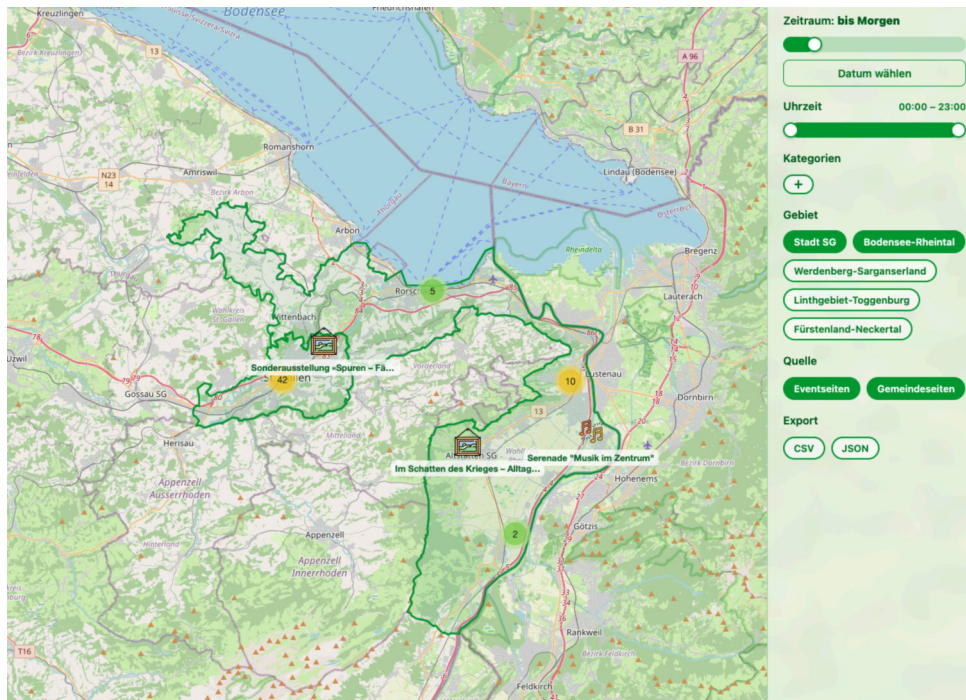


Abbildung 19: Gebietsfilter mit den fünf polizeilichen Zonen

Beim Rendern wird für jeden Eventpunkt geprüft, ob er innerhalb eines der aktivierten Gebiete liegt, andernfalls wird kein Marker gesetzt. Diese Filterung wurde aus Zeitgründen nur client-seitig umgesetzt.

7.11.1.5 Export

Das Cockpit unterstützt den Export der aktuell gefilterten Eventliste als CSV oder JSON. Der Export übernimmt Zeitraum, Kategorie und Quelle als Parameter und löst den Download über einen dynamisch erzeugten Link auf den Endpunkt `/events/export` aus (Code 29).

```
frontend/src/lib/FilterComponent.svelte TYPESCRIPT

function triggerExport(format: 'csv' | 'json') {
  const parts: string[] = [
    `start=${$activeDateRange.start}`,
    `end=${$activeDateRange.end}`,
    `format=${format}`,
    ...[...$selectedCategories].map(c => `category=${encodeURIComponent(c)}`),
    ...[...$selectedSources].map(s => `source_type=${encodeURIComponent(SOURCE_LABEL_TO_DB[s] ?? s)}`),
  ];
  const a = document.createElement('a');
  a.href = `${PUBLIC_API_URL}/events/export?${parts.join('&')}`;
}
```

Code 29: Aufbau des Download-Links für den gefilterten Export

7.11.2 Store-Schichten und Datenfluss

Die Stores bilden eine entkoppelte Vermittlungsschicht zwischen UI (Filter) und Darstellung (Karte). Filterkomponente und Karte kennen sich nicht direkt, sie kommunizieren ausschliesslich über den geteilten Store-Zustand. Reaktivität wird deklarativ (`$effect`, `$store`, `derived`) eingesetzt, wo Svelte den Datenfluss verwaltet, und imperativ (`.subscribe()`, `onMount/onDestroy`), wo eine Fremdbibliothek (Leaflet) ausserhalb des Svelte-Reaktivitätssystems gesteuert werden muss.

Das State-Management ist auf zwei Dateien aufgeteilt, ein externer State-Container wie Redux wird nicht verwendet, die eingebauten Stores genügen.

7.11.2.1 Event-Store

Der `event_store.ts` in Code 30 kapselt den API-Zugriff in der Funktion `loadEvents()` und legt das Ergebnis in `eventStore` ab. Diese Kapselung bündelt den einzigen Schreibzugriff auf die Eventliste an einer Stelle, sodass die Map-Komponente und die Filter-Komponente die Daten ausschliesslich lesen können.

```
frontend/src/lib/stores/event_store.ts  TYPESCRIPT

export async function loadEvents(startDate: string, endDate: string, q?: string) {
  isLoadingEvents.set(true);

  try {
    const params = new URLSearchParams({ start: startDate, end: endDate });
    if (q) params.set('q', q);
    for (const cat of get(selectedCategories)) params.append('category', cat);
    for (const src of get(selectedSources)) params.append('source_type', SOURCE_LABEL_TO_DB[src] ??
src);
    const response = await fetch(`${PUBLIC_API_URL}/events/?${params}`);
    if (!response.ok) {
      throw new Error("HTTP Error: " + response.status);
    }

    const data : EventResponse = await response.json();

    eventStore.set(data.events);

  } catch (error) {
    console.error("Fehler beim Laden der Events:", error);
  } finally {
    isLoadingEvents.set(false);
  }
}
```

Code 30: loadEvents() als einziger Schreibzugriff auf die Eventliste

7.11.2.2 Filter-Zustand

Der `filter_store.ts` bündelt den gesamten Filter-Zustand und die zugehörigen Mapping-Tabellen an einer Stelle. Wichtigster Bestandteil ist der abgeleitete Store `activeDateRange`, der je nach Modus aus mehreren Eingabe-Stores ein konsolidiertes `{ start, end }` erzeugt. Konsumenten müssen die Modus-Logik dadurch nicht kennen.

Die Kapselung im Store-Modul macht zudem zwei wiederkehrende Aufgaben unsichtbar für UI und API. Erstens trennen die Mapping-Tabellen den Anzeigewert in deutscher Sprache vom technischen Key, der an die API übergeben wird. Zweitens kapseln Toggle-Funktionen wie

`toggleCategory()` die immutable Aktualisierung über `new Set(...)`, die nötig ist, damit Svelte die Änderung erkennt, die Komponenten rufen die Funktion auf, ohne dieses Detail kennen zu müssen.

7.11.3 Fazit

Das Frontend baut die in der SA begonnene Kartenvisualisierung zu einem vollständigen Lage-Cockpit aus. Die Komponentenhierarchie bleibt mit `App.svelte`, `Map.svelte` und `FilterComponent.svelte` bewusst flach, den Zusammenhalt stellen zentrale Svelte-Stores her, die als entkoppelte Vermittlungsschicht zwischen Filter und Karte dienen und Domänenwissen wie Mapping-Tabellen sowie immutable Aktualisierungen an einer Stelle kapseln.

Auf dieser Basis kombiniert das Cockpit serverseitige Filter für Zeitraum, Kategorie, Quellentyp und entprellte Titelsuche mit clientseitigen Filtern für Uhrzeit und polizeiliche Gebietszonen aus KML-Polygonen. Die Ergebnisse werden über kategorieabhängige, an Schema.org orientierte Marker mit Clustering dargestellt, das die in der SA als Erweiterungsempfehlung benannte Überlagerung an identischen Koordinaten direkt adressiert, und sind über CSV- und JSON-Export weiterverwendbar. Damit ist die Verarbeitungskette von der Datenerfassung bis zur Darstellung vollständig beschrieben, sodass das folgende Kapitel die Evaluation des Gesamtsystems behandelt.

Der mit dem Auftraggeber durchgeführte Akzeptanztest (Kapitel 8.2.2) bestätigte den aktuellen Stand des Frontends als grundsätzlich tauglich und positiv aufgenommen.

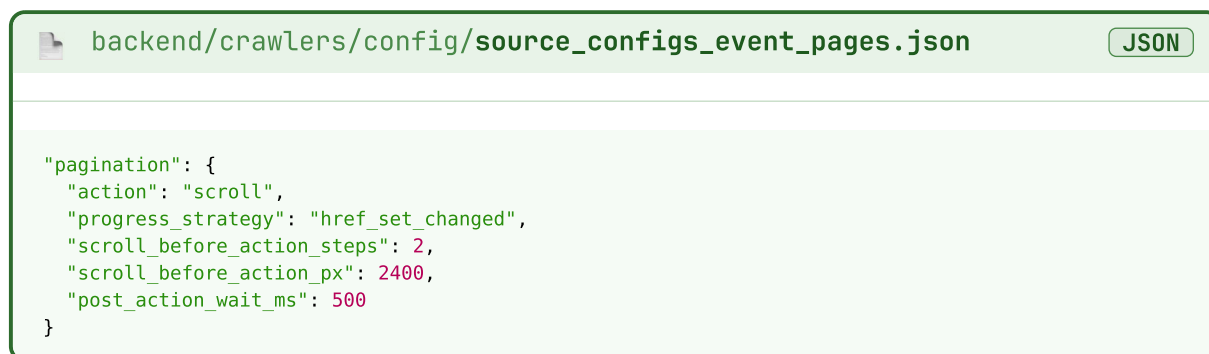
7.12 Zusätzliche Quellen

Neben den Gemeinde-Websites wurden, wie in den Zielen festgehalten (Kapitel 1.4.1), gezielt weitere Quellen eingebunden. Diese unterscheiden sich in Struktur und Zugänglichkeit erheblich von den Gemeinde-Seiten und erforderten jeweils quellspezifische Anpassungen an der Crawler-Konfiguration.

7.12.1 Guidle

Guidle ist ein Schweizer Eventportal, das Veranstaltungen aus verschiedenen Regionen aggregiert. Für die Abdeckung des Kantons St. Gallen wird die Region direkt über einen Hash-Parameter in der Listing-URL gefiltert (`#where=-131053491296198`).

Der markante Unterschied zu den Gemeinde-Quellen liegt in der Pagination: Guidle verwendet Infinite Scroll statt klassischer Seitennavigation. Der Crawler muss die Seite wiederholt nach unten scrollen und warten, bis neue Links erscheinen. Als Fortschrittsstrategie dient `href_set_changed` – der Crawler scrollt, bis sich die Menge der gesammelten Links in einem Durchgang nicht mehr verändert. Code 31 zeigt die entsprechende Konfiguration.

The image shows a code editor window with a green border. The title bar at the top reads 'backend/crawlers/config/source_configs_event_pages.json' and has a 'JSON' icon on the right. The editor contains a JSON object for pagination configuration. The code is as follows:

```
"pagination": {  
  "action": "scroll",  
  "progress_strategy": "href_set_changed",  
  "scroll_before_action_steps": 2,  
  "scroll_before_action_px": 2400,  
  "post_action_wait_ms": 500  
}
```

Code 31: Pagination-Konfiguration für Guidle (Infinite Scroll).

Zusätzlich war ein expliziter `user_agent`-String nötig sowie ein erhöhtes `timeout_ms` von 120 Sekunden, da Guidle deutlich mehr Events lädt als eine typische Gemeinde-Seite.

Eine besondere Herausforderung ergibt sich aus der Art, wie Guidle mehrtägige Veranstaltungen darstellt. Jedes Datum einer mehrtägigen Veranstaltung erscheint als separater Eintrag in der Eventliste – jedoch unter derselben URL. Der Crawler markiert eine URL beim ersten Fund als gesehen und überspringt sie bei allen weiteren Vorkommen. Im frühen Teil eines Crawls ist das unproblematisch; nach langem Scrollen enthält jede neue Bildschirmfüllung vorwiegend spätere Termine bereits gesehener mehrtägiger Events, sodass kaum neue URLs hinzukommen. Mit der alten Logik – einmal kein Fortschritt bedeutet Abbruch – endete der Crawl hier verfrüht. Der Screen Shake (beschrieben in Kapitel 7.2.5) gibt dem Lazy-Loading-Mechanismus von Guidle einen erneuten Impuls und erlaubt bis zu drei Scroll-Versuche bevor aufgegeben wird. Diese Strategie ist nicht perfekt: Auch nach drei Versuchen kann der Crawler am Ende der Liste stecken bleiben und vereinzelte zukünftige Events verpassen. Da es sich dabei stets um weit in der Zukunft liegende Termine handelt, werden diese beim nächsten regulären Crawl-Lauf zuverlässig erfasst.

Guidle bietet auch einen API-Zugang an, der in Kapitel 3.6 evaluiert wurde. Die Entscheidung fiel gegen die API-Nutzung; ein Wechsel bleibt jedoch als Option für zukünftige Iterationen offen.

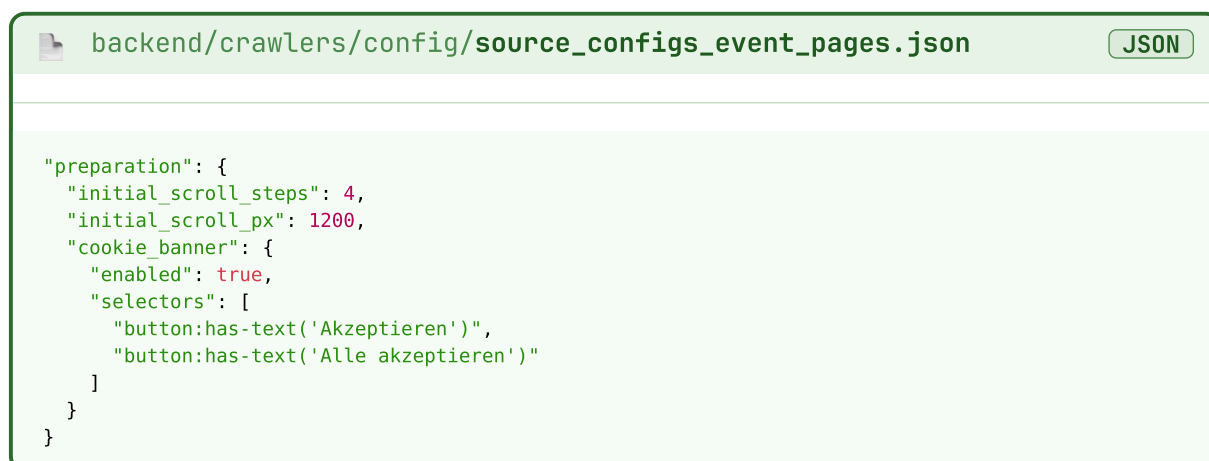
7.12.2 Rheintal Kalender

Als weitere Eventportal-Quelle wurde der Rheintal Kalender (kalender.rheintal.com) eingebunden. Technisch verhält er sich analog zu Guidle: Infinite Scroll mit `document_height_changed` als Fortschrittsstrategie, ein langer initialer Wartezeit- und Scroll-Vorlauf sowie ein expliziter User-Agent. Die Quelle ist relevant, weil einzelne Gemeinden im Rheintal ihre Veranstaltungen dort zusätzlich zu oder anstelle ihrer eigenen Websites publizieren [15], [16]. Der Kalender erfasst damit Events, die über die Gemeinde-Quellen allein nicht erreichbar wären.

7.12.3 Mobilizon

Mobilizon ist eine dezentrale Event-Plattform. Weltweit existieren zahlreiche unabhängige Mobilizon-Instanzen; in der Schweiz sind es derzeit sechs. Die Konfiguration zielt auf die Schweizer Instanz mobilisons.ch mit geographischem Filter auf die Region Appenzell, die dem Kanton SG am nächsten liegt.

Zwei Besonderheiten erforderten spezifische Anpassungen: Erstens werden die Event-Karten erst nach einem initialen Scroll geladen – ohne `initial_scroll_steps: 4` bleibt die Seite leer. Zweitens zeigt mobilisons.ch einen Cookie-Banner, dessen Bestätigung explizit konfiguriert werden muss (Code 32).



```
backend/crawlers/config/source_configs_event_pages.json JSON
{
  "preparation": {
    "initial_scroll_steps": 4,
    "initial_scroll_px": 1200,
    "cookie_banner": {
      "enabled": true,
      "selectors": [
        "button:has-text('Akzeptieren')",
        "button:has-text('Alle akzeptieren')"
      ]
    }
  }
}
```

Code 32: Cookie-Banner- und Scroll-Konfiguration für mobilisons.ch.

Die Pagination erfolgt per Klick auf den Weiter-Button, wobei der Selektor aufgrund der verschachtelten Komponentenstruktur von mobilisons.ch verhältnismässig komplex ausfiel.

Obwohl das Crawling technisch einwandfrei funktioniert, liefert keine der sechs Schweizer Instanzen Events mit Bezug zum Kanton St. Gallen [17]. Alle extrahierten Veranstaltungen werden daher durch die geografische DB-Constraint abgelehnt. mobilisons.ch bleibt in der Konfiguration als Quelle erhalten – sollte sich die Veranstaltungsdichte in der Region erhöhen, wäre die Integration ohne weitere Anpassungen sofort nutzbar.

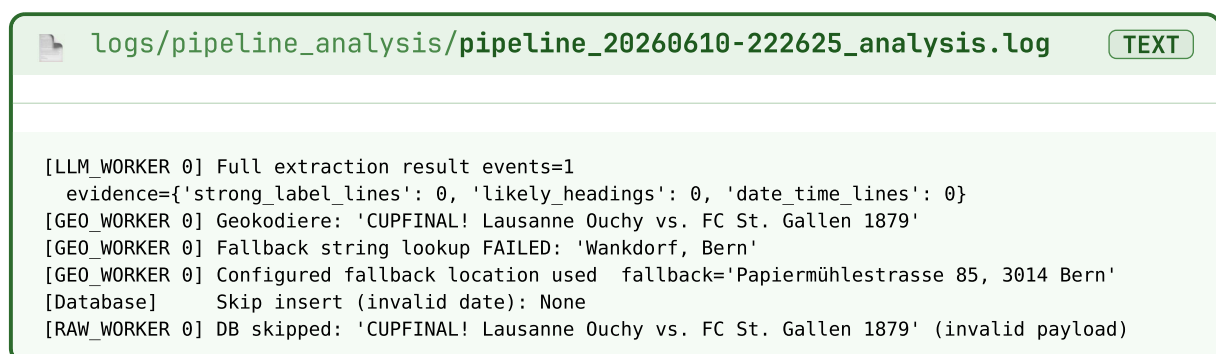
7.12.4 FCSG-Forum

Das FCSG-Forum wurde als Proof-of-Concept für Forum-Quellen gewählt, da es ohne Login zugänglich ist und potenziell relevante Fan-Veranstaltungen und informelle Treffen enthält. Da der Thread keine separaten Event-URLs besitzt, wird er mit `use_listing_url_only: true` direkt als einzige Quelle an die Pipeline übergeben.



Abbildung 20: FCSG-Forum-Thread mit Titel «CUPFINAL! Lausanne Ouchy vs. FC St. Gallen 1879 / Sonntag, 14:00 Uhr / Wankdorf Bern».

Technisch funktionierte das Crawling einwandfrei: Die Seite wurde geladen, das Markdown gespeichert, und das LLM extrahierte ein Event. Der Titel enthält auf den ersten Blick alle relevanten Angaben – Gegner, Wochentag, Uhrzeit und Ort. Dennoch scheiterte die Datums-extraktion: «Sonntag» ist ein relativer Wochentag ohne absolutes Datum, und der LLM-Worker gab `date: None` zurück. Code 33 zeigt die relevanten Log-Zeilen.



Code 33: Pipeline-Log: FCSG-Forum-Event extrahiert aber wegen fehlendem Datum abgelehnt.

Der Evidence-Block des LLM bestätigt das: `date_time_lines: 0` – keine einzige Zeile im extrahierten Markdown enthielt ein strukturiertes Datum. Der Grund ist im Markdown ersichtlich: Der eigentliche Beitragstext lautet nur «Mached eu unsterblich». Datum und Ort stehen ausschliesslich im Thread-Titel, in dem der Wochentag ohne Jahres- oder Monatsangabe verwendet wird. Selbst wenn der LLM den Posting-Timestamp (17.05.2026) hinzuzieht, bleibt die Ableitung «nächsten Sonntag = 18.05.2026» eine Inferenz, die das Modell nicht mit ausreichender Sicherheit vornahm.

Dies verdeutlicht das grundlegende Problem von Foren als Crawl-Quelle: Die Informationsstruktur ist vollständig autorenpflichtig und inkonsistent. Ein allgemeiner Crawler für das gesamte Forum würde auf noch heterogenere Beitragsformate treffen und wäre kaum zuverlässig umsetzbar.

7.12.5 Facebook

Die offizielle FCSG-Facebook-Seite wurde aus demselben Grund wie das Forum evaluiert: Fanveranstaltungen und Spielhinweise werden dort regelmässig geteilt. Der Crawler akzeptierte den Cookie-Banner erfolgreich und extrahierte den ersten sichtbaren Beitrag.

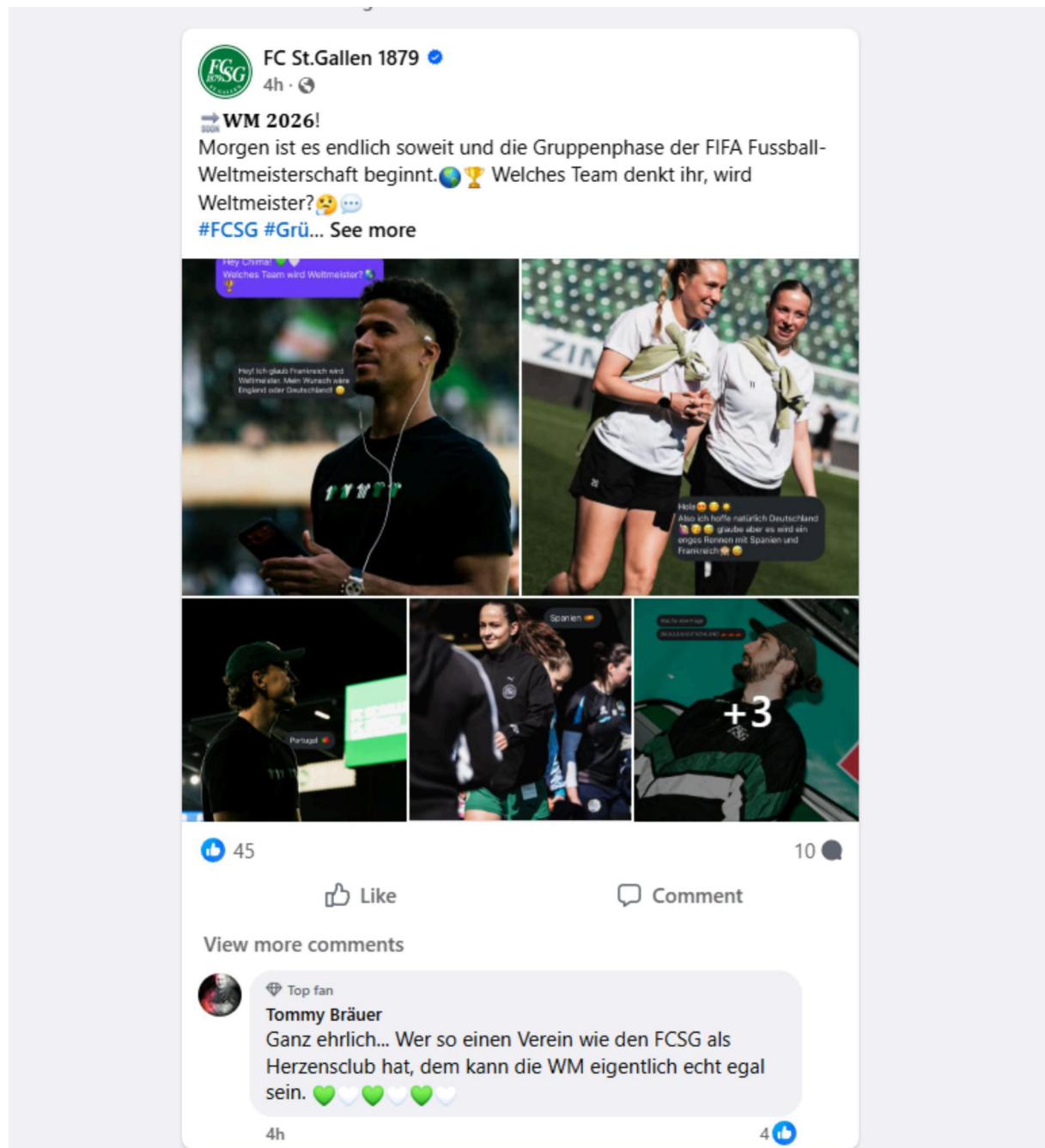


Abbildung 21: Erster sichtbarer Beitrag auf der FCSG-Facebook-Seite: Fan-Engagement-Post zur FIFA Weltmeisterschaft 2026, kein Event.

Der extrahierte Beitrag – «Welches Team denkt ihr, wird Weltmeister?» – ist reines Fan-Engagement und kein Event. Das LLM lieferte folgerichtig `events=0` bei ebenfalls leerem Evidence-Block. Direkt im Anschluss erscheint im Markdown der Login-Wall (Abbildung 22): Jeder weitere Scroll wird durch ein Login-Popup unterbrochen, was eine Weiterverarbeitung verhindert.

```
data/scraped_md/www.facebook.com_FCSG1879_[...] .md MARKDOWN

?? ????!

Morgen ist es endlich soweit und die Gruppenphase der FIFA Fussball-Weltmeisterschaft beginnt. Welches Team denkt ihr, wird Weltmeister?

#FCSG #Grü... Mehr anzeigen

...

Du würdest gern mehr sehen von FC St.Gallen 1879 | Saint Gallen ?

E-Mail-Adresse oder Telefonnummer
Passwort
Anmelden
```

Code 34: Gescraptes Markdown der FCSG-Facebook-Seite: Nach dem ersten Beitrag folgt unmittelbar der Login-Prompt.

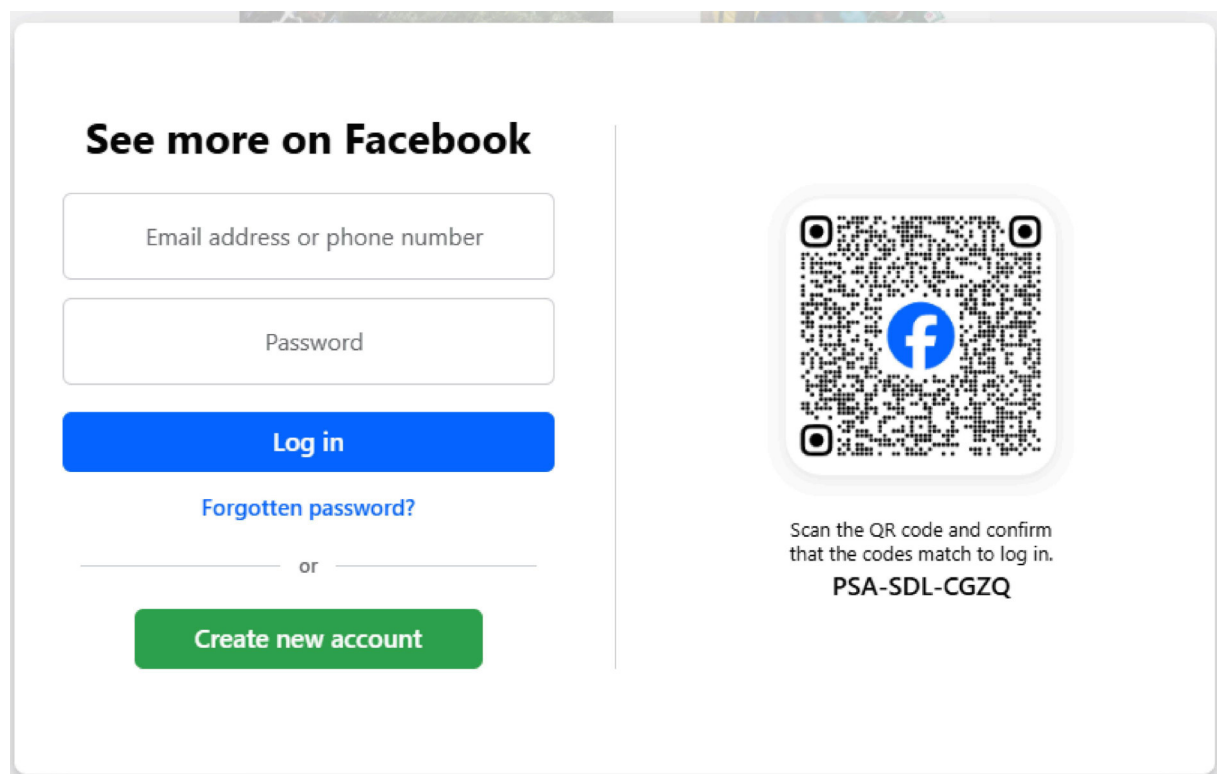


Abbildung 22: FCSG-Facebook-Profilseite: Nach dem ersten Beitrag folgt unmittelbar der Login-Prompt.

Erschwerend kommt hinzu, dass Facebook stark obfusziertes HTML verwendet – CSS-Klassen und Attributnamen werden serverseitig neu generiert, was stabile Selektoren dauerhaft unmöglich macht. Beide Social-Media-Quellen bleiben als Proof-of-Concept in der Konfiguration erhalten, eignen sich aber nicht für den produktiven Einsatz.

8 Qualitätssicherung

Die Qualitätssicherung stellt sicher, dass die entwickelte Software den festgelegten Anforderungen entspricht. Dieses Kapitel umfasst das Testkonzept, die Validierung der nicht-funktionalen Anforderungen sowie das Logging und Monitoring im Betrieb.

8.1 Testkonzept

Mit der vollständigen Neuarchitektur der Pipeline, also konfigurationsgesteuerter Crawler, Markdown-Splitting, LLM-Extraktion, Geo-Enrichment und Datenbankpersistierung, war eine entsprechende Testbasis notwendig. Die bestehenden Tests aus der Semesterarbeit [1] passten nicht mehr zur neuen Struktur und wurden neu aufgebaut. Das Testkonzept verfolgt zwei übergeordnete Ziele: Erstens sollen zentrale Pipeline-Schritte nach Änderungen stabil bleiben und Regressionen vermieden werden. Zweitens soll die Qualität der LLM-Extraktion, als nicht-deterministischer Kernschritt, messbar und reproduzierbar bewertet werden können.

Das Testkonzept umfasst drei Bereiche: automatisierte Unit Tests für die deterministische Systemlogik, einen LLM-Evaluationstest mit einem manuell kuratierten Gold-Datensatz sowie statisches Linting mit Ruff. Die CI/CD-Pipeline führt alle Tests bei Merge Requests automatisch aus.

8.1.1 Unit Tests

Die Unit Tests sind mit `pytest` umgesetzt und in `backend/tests/` nach Fachbereichen strukturiert: Crawler-Konfiguration, Link-Extraktion, Markdown-Splitting, Pipeline-Handoff, LLM-Preprocessing und Guardrails, Datenbank-Hilfsfunktionen sowie JSONL-Utilities. Zum Zeitpunkt der Dokumentation umfasst die Testbasis 63 automatisierte Unit Tests.

Ein zentrales Entwurfsprinzip ist die vollständige Isolation von externen Systemen. Das gemeinsame Test-Setup in `backend/tests/conftest.py` stubbt Playwright, psycopg und pyproj bereits beim Import, sodass kein Browser, keine Datenbank und keine native Geodatenbibliothek gestartet werden muss. LLM-Aufrufe werden nicht gemockt, sondern umgangen: Getestet werden ausschliesslich die deterministischen Vor- und Nachverarbeitungsschritte rund um die LLM-Extraktion.

Im Frontend wird mit Vitest die Store-Schicht abgedeckt, also jene Stelle, an der die Filter- und Eventabfrage-Logik unabhängig von Leaflet und DOM getestet werden kann. Die Tests liegen unter `frontend/src/lib/stores/` und umfassen 28 Tests in zwei Dateien: `filter_store.spec.ts` prüft die Datumsbereich-Ableitung für alle Slider-Steps der Schnellauswahl, die Konsistenz des deutsch/englischen Kategorie-Mappings sowie das Toggle-Verhalten für Kategorien, Quellen und Gebiete. `event_store.spec.ts` testet mit gemocktem `fetch` den Aufbau der Backend-Query (Datumsbereich, optionaler Suchbegriff, Kategorien, Quellen-Label auf DB-Wert), die Store-Aktualisierung bei Erfolg sowie die Fehlerbehandlung, bei der bestehende Events erhalten bleiben.

8.1.2 LLM-Evaluationstest

Da sich der zentrale Qualitätsanspruch des Systems auf die LLM-Extraktion konzentriert, wurde ergänzend zu den Unit Tests ein eigenständiger Evaluationstest entwickelt. Dieser Test befindet sich in `backend/tests/test_evaluation/` und folgt dem Prinzip eines Gold-Datensatzes: Für eine repräsentative Auswahl echter Markdown-Dateien aus verschiedenen Quellen werden die erwarteten Events manuell annotiert und eingefroren. Der Evaluationslauf ruft das LLM live auf, vergleicht die Ausgaben gegen die Goldstandard-Annotationen und berechnet Metriken für Event-Erkennung und Feldgenauigkeit.

8.1.2.1 Gold-Datensatz

Die Markdown-Dateien liegen in `backend/tests/test_evaluation/fixtures/gold_md/` und decken bewusst verschiedene Quellentypen und Schwierigkeitsgrade ab: einfache Einzelevents von Gemeinde-Seiten (Altstätten), mehrteilige Splitter-Ergebnisse (Degersheim), Events von Portalen (Guidle, Mobilizon, Rheintal Kalender, Eventfrog) sowie fünf explizite Nicht-Event-Fälle (Wildhaus-Newsmeldungen), bei denen das LLM kein Event extrahieren sollte. Das Format der Goldstandard-Annotationen ist in `backend/tests/test_evaluation/fixtures/README.md` dokumentiert.

Der aktuelle Gold-Datensatz umfasst 35 Testfälle (30 Event-Fälle, 5 Nicht-Event-Fälle) mit 30 erwarteten Events.

8.1.2.2 Ergebnisse

Tabelle 44 zeigt die Event-Erkennungsmetriken des letzten Evaluationslaufs.

Metrik	Wert
Erwartete Events	30
Erkannte Events (Predicted)	35
Übereinstimmende Events (Matched)	28
False Positives	7
False Negatives	2
Precision	0.800
Recall	0.933
F1-Score	0.862

Tabelle 44: Event-Erkennungsmetriken des LLM-Evaluationslaufs (09.06.2026).

Das Modell zeigt einen hohen Recall: Von 30 erwarteten Events wurden 28 korrekt erkannt. Die 7 False Positives sind aussagekräftiger als die rohe Zahl vermuten lässt: zwei davon stammen aus den Wildhaus-Newsmeldungen, bei denen das LLM Datumsangaben in redaktionellen Texten fälschlicherweise als Eventdaten interpretierte (Tabelle 45).

Feld	Genauigkeit	Mean Score
date	1.000	1.000
city_name	1.000	1.000
city_zip	1.000	1.000
street	1.000	1.000
street_number	1.000	1.000
country	1.000	1.000
event_status	1.000	1.000
title	0.964	0.979
full_location	0.957	0.977
start_time	0.926	0.926
end_time	0.917	0.917
category	0.429	0.429

Tabelle 45: Feldgenauigkeit der LLM-Extraktion über 28 gematchte Events.

Die Feldgenauigkeit zeigt ein klares Muster: Strukturierte Daten wie Datum, Adresse und Status werden nahezu fehlerfrei extrahiert. Die einzige signifikante Schwachstelle ist die Kategorie-klassifikation mit 42.9%. Die Konfusionsmatrix zeigt, dass ein Grossteil der Fehlklassifikationen auf `SocialEvent` entfällt, einer Residualkategorie, in die das Modell tendiert, wenn kein spezifischerer Typ passt. Diese Schwäche ist bekannt und wird in Kapitel 5.1.1 eingeordnet.

Die Evaluation basiert auf 35 Testfällen, was einen guten ersten Eindruck der Extraktionsqualität vermittelt; für statistisch belastbarere Aussagen wäre ein grösseres Gold-Dataset empfehlenswert.

8.1.3 Linting und CI/CD

Für Codequalität wird Ruff eingesetzt. Es prüft Backend-Module und Testdateien auf Stilprobleme und typische Fehlerquellen. Die Tests werden lokal mit `uv run pytest` ausgeführt, in der CI/CD-Pipeline laufen sie zusätzlich automatisch bei jedem Merge Request. Damit wird verhindert, dass Änderungen an Crawling, Splitting oder LLM-Verarbeitung unbemerkt bestehende Funktionalität beschädigen.

8.2 Test Plan mit Endanwendern

Zur Evaluation des Event-Suchroboter-Cockpits wurden sowohl ein Usability-Test mit einem Endanwender als auch ein Akzeptanztest mit dem Auftraggeber durchgeführt. Ziel war es, die Benutzerfreundlichkeit der Anwendung sowie die Erfüllung der fachlichen Anforderungen zu überprüfen.

8.2.1 Usability-Test

Dieser Abschnitt beschreibt den Usability-Test mit einem Endanwender. Ziel war die Überprüfung, ob die zentralen Funktionen des Event-Suchroboter-Cockpits verständlich bedient werden können und ob während der Nutzung erkennbare Hürden auftreten.

- Datum: 12.05.2026
- Ort: Lokal
- Teilnehmer: Marcel Braun
- Moderator: Martyn Tarquin Foreman

Vorbereitung

- Funktionsfähige Event-Cockpit-Umgebung bereitgestellt.
- Aktuelle Eventdaten über die Crawler-Pipeline eingespielt.
- Sichergestellt, dass ausreichend Events für die Testaufgaben vorhanden sind.
- Kartenansicht, Suchfunktion und Filter vorbereitet.

Session-Überblick

- Ziele der Sitzung erläutert und den Fokus auf die Auffindung von Usability-Problemen gelegt.
- Offenes und ehrliches Feedback ausdrücklich erwünscht.
- Darauf hingewiesen, dass auch kleine Probleme oder Unsicherheiten relevant sind.
- Die Bedienung wurde beobachtet, ohne die einzelnen Schritte vorzugeben.

Aufgaben

- Ein beliebiges Event finden.
- Nach einer Eventkategorie filtern.
- Nach einem bestimmten Datum filtern.
- Nach einer bestimmten Uhrzeit oder Zeitspanne filtern.
- Die Suchfunktion verwenden.
- In die Karte hinein- und herauszoomen.
- Die Karte verschieben.
- Ein Event in St.Gallen finden.
- Ein Event ausserhalb von St.Gallen finden.
- Eventdetails öffnen und prüfen.
- Zwischen Dark Mode und Light Mode wechseln.
- Zusammenfassung eines Events prüfen.

Beobachtungen

- Der Teilnehmer konnte die wichtigsten Funktionen erfolgreich nutzen.
- Die Navigation auf der Karte funktionierte intuitiv.
- Die Anwendung reagierte auch auf unterschiedlichen Bildschirmgrößen responsiv.
- Die Event-Popups wurden als hilfreich und gut umgesetzt wahrgenommen.
- Die erweiterte Zeitbereichsfilterung wurde als nützlich für die Planung bewertet.

- Die Event-Zusammenfassungen wurden positiv bewertet, da sie nicht lediglich den Originaltext der Quelle wiederholen.
- Die vorhandenen Eventinformationen wurden als vollständig wahrgenommen.
- Die Kategorien waren nicht alphabetisch sortiert.
- Eine Gefährdungs- oder Risikoeinstufung der Events war nicht vorhanden.

Teilnehmerkommentare

- Dark Mode und Light Mode wurden sehr positiv bewertet.
- Die Zeitbereichsfilterung wurde als sehr hilfreich für die Planung beschrieben.
- Die Zusammenfassungsfunktion wurde positiv hervorgehoben.
- Die Event-Popups wurden als sehr gelungen bewertet.
- Der visuelle Rahmen und die Gestaltung der Anwendung wurden positiv wahrgenommen.
- Alle relevanten Informationen seien vorhanden.
- Die Anwendung sei responsiv.
- Die fehlende Gefährdungseinstufung wurde angemerkt.
- Die nicht alphabetische Sortierung der Kategorien wurde als störend wahrgenommen.
- Als zusätzliche geografische Abdeckung wurde Winterthur erwähnt.

Fazit und nächste Schritte

Der Teilnehmer bewertete das Event-Cockpit insgesamt positiv. Besonders hervorgehoben wurden Dark Mode und Light Mode, die responsiven Ansichten, die Event-Popups, die erweiterte Zeitbereichsfilterung sowie die automatisch erzeugten Event-Zusammenfassungen.

Als konkrete Verbesserung innerhalb des Projektumfangs wurde die Sortierung der Kategorien identifiziert. Diese sollten alphabetisch geordnet werden, damit die gewünschte Kategorie schneller gefunden werden kann.

Die fehlende Gefährdungs- oder Risikoeinstufung wurde angesprochen, ist jedoch nicht Bestandteil der vorliegenden Arbeit. Ebenso liegt eine geografische Erweiterung, beispielsweise um Winterthur, ausserhalb des definierten Projektumfangs, da sich die Anforderungen auf Events im Kanton St.Gallen beschränken.

8.2.2 Akzeptanztest

Dieser Abschnitt beschreibt den formalen Akzeptanztest mit dem Auftraggeber. Ziel war die Überprüfung, ob die entwickelte Lösung die definierten Anforderungen erfüllt und für den vorgesehenen Einsatzzweck geeignet ist.

- Datum: 27.05.2026
- Ort: Online
- Teilnehmer: Patrick Steinmann
- Moderator: Martyn Tarquin Foreman

Vorbereitung

- Funktionsfähige Event-Cockpit-Umgebung bereitgestellt.
- Aktuelle Eventdaten über die Crawler-Pipeline eingespielt.
- Kartenansicht, Suchfunktion und Filter vorbereitet.
- Sicherstellung, dass ausreichend Events aus verschiedenen Quellen vorhanden sind.

Session-Überblick

- Ziele der Sitzung erläutert und den Fokus auf die fachliche Abnahme des Systems gelegt.
- Zentrale Funktionen des Cockpits vorgestellt.
- Feedback zu Funktionalität, Vollständigkeit und Praxistauglichkeit eingeholt.

Aufgaben

- Übersicht über die auf der Karte dargestellten Events gewinnen.
- Nach Events suchen.
- Nach Kategorien filtern.
- Nach Datum filtern.
- Eventdetails anzeigen und prüfen.
- Die automatische Kategorisierung mehrerer Events beurteilen.
- Die Darstellung von Event-Clustern auf der Karte bewerten.
- Performance und Reaktionsverhalten der Anwendung beurteilen.
- Eignung des Cockpits für den polizeilichen Arbeitskontext beurteilen.

Beobachtungen

- Die Benutzeroberfläche reagierte schnell und flüssig.
- Die Karteninteraktion funktionierte zuverlässig.
- Die Kategorisierung der Events wurde als nachvollziehbar bewertet.
- Die Suchfunktion lieferte die erwarteten Ergebnisse.
- Die Filterfunktionen funktionierten grundsätzlich wie erwartet.
- Die Cluster-Darstellung erschwerte teilweise die Auswahl einzelner Events bei hoher Dichte.

Teilnehmerkommentare

- Die Kartenansicht vermittele einen guten Überblick über die Veranstaltungslage.
- Die Suchfunktion sollte klarer kommunizieren, ab wann Ergebnisse angezeigt werden.
- Die Bezeichnungen einiger Filteroptionen sollten vereinheitlicht werden.
- Die Datumsfilterung sollte technisch überprüft werden.
- Die Möglichkeit eines Datenexports wurde als sinnvoll bewertet.

Fazit und nächste Schritte

Der Akzeptanztest verlief erfolgreich. Die Anwendung erfüllte die definierten Kernanforderungen hinsichtlich automatisierter Datenerfassung, Visualisierung und Analyse von Veranstaltungen. Besonders positiv bewertet wurden die Übersichtlichkeit der Kartenansicht sowie die Exportfunktion der gefilterten Eventdaten.

Die identifizierten Verbesserungsvorschläge betreffen hauptsächlich die Benutzerfreundlichkeit und stellen keine grundlegenden funktionalen Mängel dar.

Folgende Verbesserungsmaßnahmen wurden identifiziert:

- Verbesserung der Benutzerführung bei der Suchfunktion.
- Optimierung der Darstellung stark verdichteter Eventcluster.

Die Ergebnisse bestätigen die Eignung des Proof-of-Concepts als Grundlage für eine mögliche Weiterentwicklung in Richtung eines produktiven Lage- und Informationscockpits.

8.3 Validierung von NFRs

Die nicht funktionalen Anforderungen sind in Kapitel 6.1.4 beschrieben. Folgende Tabellen beschreiben den abschliessenden Status der NFRs.

ID	NFR-01 Geografische Einschränkung auf den Kanton St. Gallen
Anforderung	Das System soll Ereignisse geografisch auf den Kanton St. Gallen einschränken können.
Priorität	Mittel
Status	erfüllt
Begründung	Die Quellenkonfiguration listet ausschliesslich Gemeinden und Anbieter aus dem Kanton St. Gallen, und die Geo-Anreicherung gleicht Adressen gegen die Tabelle <code>swiss_addresses</code> ab.

Tabelle 46: Validierung NFR-01 Das System soll Ereignisse geografisch auf den Kanton St. Gallen einschränken können.

ID	NFR-02 Konsistente Sprache und Terminologie
Anforderung	Die Benutzeroberfläche und die Dokumentation sollen konsistent in Deutsch (CH) verfasst sein.
Priorität	Mittel
Status	erfüllt
Begründung	Die Cockpit-Oberfläche ist durchgängig in Deutsch (CH). Ausnahme ist der Titel der Hauptseite, der bewusst als «Event-Suchroboter-Cockpit» formuliert ist, um die Fachlichkeit und den Zweck der Anwendung klar zu kommunizieren.

Tabelle 47: Validierung NFR-02 Die Benutzeroberfläche und die Dokumentation sollen konsistent in Deutsch (CH) verfasst sein.

ID	NFR-03 Fehlertoleranz bei externen Quellen
Anforderung	Fehler einzelner Datenquellen sollen die Verarbeitung anderer Quellen nicht verhindern.
Priorität	Hoch
Status	erfüllt
Begründung	Die Pipeline isoliert Fehler pro Quelle durch eine queue-basierte Worker-Architektur, sodass ein nicht erreichbares Listing oder ein einzelnes fehlerhaftes Markdown den Gesamtlauf nicht abbricht. Erfolgreiche, leere und fehlerhafte Quellen werden in Stage-Dateien und Logs separat festgehalten.

Tabelle 48: Validierung NFR-03 Fehler einzelner Datenquellen sollen die Verarbeitung anderer Quellen nicht verhindern.

ID	NFR-04 Nachvollziehbarkeit durch Logging und Monitoring
Anforderung	Die Verarbeitungsschritte der Datenpipeline sollen nachvollziehbar protokolliert werden.
Priorität	Hoch
Status	erfüllt
Begründung	Jeder Pipeline-Schritt schreibt strukturierte JSONL-Stage-Dateien sowie Logs mit Zeitstempel, Quellenbezug und Fehlerkontext, sodass technische Fehler, leere Ergebnisse und ungültige Eventdaten unterscheidbar sind. Zum Abschluss eines Pipeline-Laufs wird eine Zusammenfassung der verarbeiteten Quellen und Ereignisse erstellt, womit die Akzeptanzkriterien abgedeckt sind.

Tabelle 49: Validierung NFR-04 Die Verarbeitungsschritte der Datenpipeline sollen nachvollziehbar protokolliert werden.

ID	NFR-05 Wiederherstellbarkeit nach Fehlern
Anforderung	Das System soll nach einem technischen Fehler mit geringem Aufwand wiederhergestellt werden können.
Priorität	Hoch
Status	erfüllt
Begründung	Alle Komponenten in Docker-Containern laufen mit Restart-Policy, und die persistierten Eventdaten liegen in einem Volume der PostgreSQL-Datenbank, womit sie einen Container-Absturz unbeschadet überstehen. Da die Pipeline ihre Zwischenstände in JSONL-Stage-Dateien hält, kann ein abgebrochener Lauf nach dem Wiederanlauf reproduzierbar fortgesetzt oder neu gestartet werden.

Tabelle 50: Validierung NFR-05 Das System soll nach einem technischen Fehler mit geringem Aufwand wiederhergestellt werden können.

ID	NFR-06 Einfache Installation und lokale Ausführbarkeit
Anforderung	Das System soll auf einem neuen Host-System mit geringem Aufwand lokal gestartet werden können.
Priorität	Mittel
Status	erfüllt
Begründung	Frontend, Datenbank und API werden über eine zentrale `docker-compose.yml` orchestriert, und die notwendigen Umgebungsvariablen und Installationsschritte für die Crawler-Pipeline sind in der Projekt-README dokumentiert.

Tabelle 51: Validierung NFR-06 Das System soll auf einem neuen Host-System mit geringem Aufwand lokal gestartet werden können.

ID	NFR-07 Erweiterbarkeit der Datenquellen
Anforderung	Neue Datenquellen sollen mit begrenztem Aufwand in die bestehende Pipeline integriert werden können.
Priorität	Hoch
Status	erfüllt
Begründung	Neue Quellen werden über die deklarative JSON-Crawler-Konfiguration angebunden, ohne dass der Datenbank-, API- oder Frontend-Code angepasst werden muss. Quellenspezifische Eigenheiten wie Splitter-Strategien oder Listing-Fallbacks sind als optionale Felder gekapselt, sodass eine neue Gemeindeseite oder Plattform mit überschaubarem Aufwand integrierbar ist.

Tabelle 52: Validierung NFR-07 Neue Datenquellen sollen mit begrenztem Aufwand in die bestehende Pipeline integriert werden können.

ID	NFR-08 Datenqualität bei extrahierten Events
Anforderung	Extrahierte Events sollen vor der Speicherung auf minimale Vollständigkeit und Plausibilität geprüft werden.
Priorität	Hoch
Status	erfüllt
Begründung	Vor der Persistierung werden Events normalisiert und gegen Guardrails geprüft, und ein deterministisch berechneter Vollständigkeitsstatus markiert fehlende Pflichtfelder wie Titel, Datum oder Ort. Unvollständige oder ungültige Events werden nicht stillschweigend als vollständig behandelt, sondern explizit markiert, und die Quell-URL bleibt jederzeit erhalten, womit die Akzeptanzkriterien strukturell abgedeckt sind.

Tabelle 53: Validierung NFR-08 Extrahierte Events sollen vor der Speicherung auf minimale Vollständigkeit und Plausibilität geprüft werden.

ID	NFR-09 Vermeidung unnötiger Duplikate
Anforderung	Das System soll mehrfach gefundene Events erkennen und die Anzeige unnötiger Duplikate reduzieren.
Priorität	Hoch
Status	erfüllt
Begründung	Die NFR wird als erfüllt eingeschätzt. Die Persistenzschicht entscheidet über einen fachlichen Fingerprint aus Titel, Datum, Ort und Quelle, ob ein Event neu eingefügt, mit einem bestehenden zusammengeführt oder als Duplikat verworfen wird. Die Quell-URL bleibt erhalten und Ähnlichkeitstreffer werden bewusst mit Vorsicht behandelt, sodass unterschiedliche Events mit ähnlichem Titel nicht stillschweigend zusammengelegt werden.

Tabelle 54: Validierung NFR-09 Das System soll mehrfach gefundene Events erkennen und die Anzeige unnötiger Duplikate reduzieren.

ID	NFR-010 Benutzbarkeit der Event-Übersicht
Anforderung	Relevante Events sollen im Frontend schnell auffindbar und verständlich dargestellt werden.
Priorität	Hoch
Status	erfüllt
Begründung	Durch Usability-Tests mit Testpersonen wurde bestätigt, dass die Karte und Filter intuitiv bedienbar sind, und die Detailansicht alle geforderten Informationen klar darstellt, wird die NFR als erfüllt eingeschätzt.

Tabelle 55: Validierung NFR-010 Relevante Events sollen im Frontend schnell auffindbar und verständlich dargestellt werden.

ID	NFR-11 Performance bei typischer Nutzung
Anforderung	Das Frontend soll bei typischen Datenmengen ohne spürbare Verzögerungen bedienbar bleiben.
Priorität	Mittel
Status	erfüllt
Begründung	Die zeit-, kategorie- und quellenbasierten Filter werden serverseitig auf einer Datenbank ausgeführt, während die Karte mit Marker-Clustering auch bei mehreren hundert Events flüssig bedienbar bleibt. Konkrete Reaktionszeitmessungen unter 2 Sekunden für Filter und 1 Sekunde für die Detailansicht stehen noch aus, in der praktischen Nutzung wirken die Reaktionen aber im Zielbereich.

Tabelle 56: Validierung NFR-11 Das Frontend soll bei typischen Datenmengen ohne spürbare Verzögerungen bedienbar bleiben.

ID	NFR-12 Schutz sensibler Konfigurationsdaten
Anforderung	Sensible Konfigurationsdaten wie API-Schlüssel oder Zugangsdaten dürfen nicht im Quellcode gespeichert werden.
Priorität	Hoch
Status	erfüllt
Begründung	API-Schlüssel und Datenbankzugangsdaten werden ausschliesslich über Umgebungsvariablen aus einer lokalen <code>.env</code> -Datei geladen, die per <code>.gitignore</code> aus dem Repository ausgeschlossen ist.

Tabelle 57: Validierung NFR-12 Sensible Konfigurationsdaten wie API-Schlüssel oder Zugangsdaten dürfen nicht im Quellcode gespeichert werden.

ID	NFR-13 Respektvoller Umgang mit öffentlichen Quellen
Anforderung	Das System soll öffentliche Quellen technisch und organisatorisch schonend nutzen.
Priorität	Hoch
Status	erfüllt
Begründung	Der URL-Crawler arbeitet mit kontrollierter Parallelität und limitierter Abfragefrequenz pro Quelle und greift ausschliesslich auf öffentlich zugängliche Inhalte aus der Crawler-Konfiguration zu. Personenbezogene Daten werden weder als eigenständiges Ziel gesammelt noch ausgewertet, sondern fliessen nur indirekt als Teil öffentlich publizierter Eventbeschreibungen in das System ein.

Tabelle 58: Validierung NFR-13 Das System soll öffentliche Quellen technisch und organisatorisch schonend nutzen.

8.4 Deployment Test

Um die Lauffähigkeit des Systems auf externen Infrastrukturen zu validieren, wurde die Applikation auf einem Linux-Server deployt. Damit konnte nachgewiesen werden, dass das System unabhängig von der lokalen Entwicklungsumgebung funktioniert und die einzelnen Komponenten auch in einer realen Serverumgebung korrekt zusammenspielen.

Für das Deployment wurde eine durch die OST bereitgestellte virtuelle Maschine im internen Netzwerk verwendet. Die wichtigsten technischen Eckdaten der VM sind in Tabelle 59 zusammengefasst.

Attribut	Wert
Hostname	srbsci-124.ost.ch
Netzwerk	intern (OST)
Betriebssystem	Ubuntu Linux
CPU	2 vCPU
RAM	4096 MB
Disk	50 GB

Tabelle 59: Technische Eckdaten der Deployment-VM.

Das Deployment erfolgte mittels Docker Compose, wie in der README Anhang Abschnitt F.2.5 dokumentiert. Alle Systemkomponenten wurden auf dem Server betrieben: Backend, Frontend, Datenbank sowie die Crawler-Pipeline. Im Vergleich zur lokalen Entwicklungsumgebung musste die Concurrency der Crawler-Pipeline aufgrund des begrenzten Arbeitsspeichers der VM (4 GB RAM) reduziert werden, um eine stabile Ausführung zu gewährleisten. Abgesehen von dieser Anpassung verhielt sich das System auf dem Server identisch zur lokalen Umgebung, was die Portierbarkeit des Docker-basierten Deployments bestätigt.

Das Frontend ist bis zum End-of-Life des Servers über folgende URL im OST-internen Netzwerk erreichbar: <http://srbsci-124.ost.ch:5173/>

8.5 Prozess-Logging

Für den Betrieb und die Fehlersuche wurden die Logging-Mechanismen gegenüber der Studienarbeit deutlich erweitert. Während die ursprüngliche Lösung hauptsächlich auf einer zentralen Logdatei basierte [1, p. 31], verfolgt das Event-Suchroboter-Cockpit heute mehrere komplementäre Logging-Ansätze für unterschiedliche Zielgruppen.

Die Logging-Architektur dient dabei nicht nur der Fehlersuche, sondern auch der Nachvollziehbarkeit einzelner Verarbeitungsschritte, der Qualitätskontrolle sowie dem produktiven Monitoring des Systems.

8.5.1 Pipeline Runtime Log

Das zentrale Pipeline-Log protokolliert weiterhin den gesamten Ablauf eines Crawlvorgangs. Im Gegensatz zur ursprünglichen Lösung umfasst es heute sämtliche Verarbeitungsschritte von der Quellerschliessung bis zur Persistierung in der Datenbank.

Typische Logeinträge dokumentieren:

- Crawling einzelner URLs
- Queue- und Worker-Aktivitäten
- Parallelitäts- und Rate-Limit-Kontrolle
- Markdown-Aufbereitung
- LLM-Extraktion
- Geo-Enrichment
- Kategorisierung
- Persistierung und Deduplizierung
- Fehler- und Warnmeldungen

Dadurch kann jederzeit nachvollzogen werden, welchen Weg eine Quelle durch die Pipeline genommen hat und in welcher Stufe Probleme aufgetreten sind.

Ein typischer Pipeline-Lauf zeigt die Verarbeitung einer Quelle über mehrere Worker hinweg. Dabei werden Crawling, LLM-Extraktion, Geokodierung, Kategorisierung sowie die anschließende Persistierung in die Datenbank protokolliert. Zusätzlich werden Laufzeiten, Token-Verbrauch, Retry-Vorgänge und Warnmeldungen erfasst. Dadurch kann sowohl die technische Verarbeitung als auch das Laufzeitverhalten einzelner Pipeline-Stufen analysiert werden.

8.5.2 Crawl-Zusammenfassung

Nach jedem vollständigen Pipeline-Lauf wird zusätzlich eine automatische Loganalyse durchgeführt. Diese fasst die wichtigsten Kennzahlen des Crawls zusammen und ermöglicht eine schnelle Beurteilung des Gesamtergebnisses.

Beispielsweise werden folgende Werte ausgewertet:

- Anzahl analysierter Logzeilen
- Fehler und Warnungen
- Neue Events
- Aktualisierte Events
- Zusammengeführte Duplikate
- Unveränderte Events
- Mehrdeutige Aktualisierungen

Die Zusammenfassung erlaubt es, den Erfolg eines Crawls auf einen Blick zu beurteilen, ohne das vollständige Runtime-Log analysieren zu müssen. Abbildung 23 zeigt ein Beispiel einer automatisch erzeugten Crawl-Zusammenfassung.

```
Log file: D:\GitProjects\School\BA - FS2026\event-suchroboter-cockpit\logs\pipeline_20260607-093443.log
Lines parsed: 1331
Summary:
  error: 0
  warning: 27
  dedup: 30 (details hidden)
  merge: 0
  update: 0
  unchanged: 0 (details hidden)
  ambiguous_update: 0
  insert: 19
```

Abbildung 23: Automatisch generierte Zusammenfassung eines Crawlvorgangs

8.5.3 LLM-Prozess-Logging

Während das Runtime-Log den technischen Ablauf dokumentiert, fokussiert das LLM-Prozess-Log auf die fachliche Verarbeitung einzelner Events.

Jedes Event erhält bereits während der Extraktion eine eindeutige **trace_id**. Diese Kennung wird über sämtliche Pipeline-Stufen hinweg mitgeführt und erlaubt die lückenlose Nachverfolgung eines einzelnen Events.

Für jede Verarbeitung werden unter anderem folgende Informationen protokolliert:

- Eingehende Markdown-Datei
- Quell-URL
- Verwendete Locked Facts
- LLM-Antworten
- Guardrail-Anpassungen
- Geo-Ergebnisse
- Kategorisierung
- Persistierungsentscheidungen
- Finale Event-Payload

Ein typischer Logeintrag dokumentiert die vollständige Eventstruktur inklusive Titel, Datum, Ort, Koordinaten, Kategorie und Status. Zusätzlich wird über die `trace_id` eindeutig festgehalten, welches Event verarbeitet wurde und in welche Datei das Ergebnis geschrieben wurde.

Dadurch kann nachvollzogen werden, ob Fehler oder Inkonsistenzen bereits während der Extraktion entstanden sind oder erst in späteren Verarbeitungsschritten auftreten. Die `trace_id` ermöglicht dabei die Verknüpfung sämtlicher Verarbeitungsschritte eines Events über die gesamte Pipeline hinweg.

8.5.4 Prompt- und Hint-Logging

Zusätzlich kann die Pipeline die tatsächlich an das Sprachmodell gesendeten Prompts protokollieren. Diese Funktion dient primär der Entwicklung, Qualitätssicherung und Optimierung der Extraktionslogik.

Wird die Konfiguration

`LLMHUB_LOG_FULL_PROMPT=true`

aktiviert, speichert die Pipeline den vollständigen Request an das LLM. Dies umfasst:

- System Prompt
- User Prompt
- Markdown-Inhalt
- Locked Facts
- Source Hints
- Geo-Hints
- Modellinformationen

Dadurch kann exakt nachvollzogen werden, welche Informationen dem Modell zur Verfügung standen.

Neben dem eigentlichen Markdown werden zusätzliche Hinweise an das Modell übergeben. Dazu gehören die Quell-URL, bekannte Koordinaten sowie unveränderliche Locked Facts wie `source_url`, `source_type` oder `coordinates`. Diese Hinweise dienen dazu, die Extraktionsqualität zu verbessern und dem Modell konsistente Informationen bereitzustellen.

Gleichzeitig erleichtert das Prompt-Logging die Analyse von Extraktionsfehlern erheblich, da exakt nachvollzogen werden kann, welche Eingaben dem Modell vorlagen. Eine detaillierte Beschreibung der Hint-Mechanismen erfolgt in Kapitel 7.5.

Da vollständige Prompt-Logs sehr gross werden können und potenziell sensible Daten enthalten, ist diese Funktion standardmässig deaktiviert.

8.5.5 Monitoring

Für den produktiven Betrieb wird zusätzlich die Plattform Sentry eingesetzt.

Sentry sammelt Fehler und Warnungen zentral und stellt diese in Echtzeit über ein Web-Dashboard dar. Kritische Fehler werden automatisch per E-Mail an die Entwickler weitergeleitet, wodurch Probleme frühzeitig erkannt werden können.

Die zentrale Überwachung der Anwendung ist in Abbildung 24 dargestellt. Über das Dashboard können Fehlerereignisse analysiert, Trends erkannt und Ursachen systematisch untersucht werden. Dadurch wird die Reaktionszeit bei Problemen deutlich reduziert.

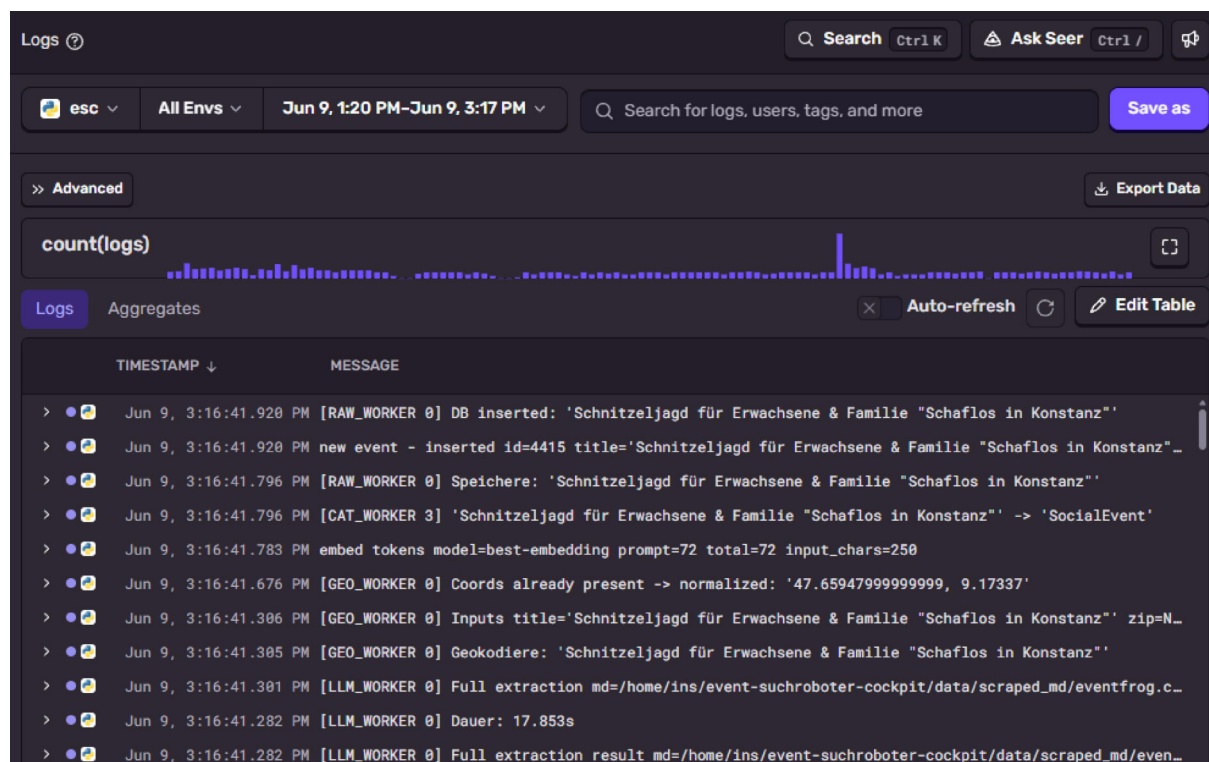


Abbildung 24: Sentry-Dashboard zur Überwachung von Fehlern und Warnungen

Durch die Kombination aus Runtime-Logging, Event-Traceability, automatischer Loganalyse und Sentry-Monitoring entsteht eine umfassende Beobachtbarkeit des Systems. Dies erleichtert sowohl die Entwicklung als auch den späteren produktiven Betrieb erheblich.

9 Projektmanagement

Die erfolgreiche Umsetzung des Projekts basiert auf einer strukturierten organisatorischen Basis und einem kontinuierlichen Controlling. Dazu beleuchtet dieses Kapitel zunächst die Rollenverteilung und die Kollaborationsprozesse der Teammitglieder. Anschliessend werden das systematische Risikomanagement sowie der tatsächliche zeitliche Aufwand der Entwicklungsphase analysiert, bevor die Projektarbeit mit einer kritischen Reflexion im Fazit abschliesst.

9.1 Planung

Die organisatorischen Rahmenbedingungen der Arbeit konzentrieren sich primär auf die klare Aufgabenverteilung im Projektteam sowie die eingesetzten Sitzungsformate, mit denen die Zusammenarbeit über die gesamte Projektlaufzeit koordiniert wurde. Da die vorliegende Arbeit nahtlos eine bestehende Studienarbeit fortführt [1], konnte das Team von Beginn an auf bereits eingespielte Abläufe zurückgreifen. Dies erübrigte eine lange Orientierungsphase und erlaubte es, den Fokus sofort auf die inhaltliche Weiterentwicklung zu richten.

9.1.1 Rollenverteilung

Die Verantwortlichkeiten wurden so verteilt, dass beide Teammitglieder einen klar abgegrenzten Zuständigkeitsbereich erhielten und sich fachlich vertiefen konnten, anstatt die Arbeit breit aufzuteilen. Die Aufteilung ist in Tabelle 60 zusammengefasst.

Wer	Rolle	Verantwortlichkeiten
Martyn Foreman	Projektleitung, Backend-Entwicklung	Projektadministration. Entwicklung der Crawling-Pipeline, Integration neuer Datenquellen, LLM-basierte Event-Extraktion sowie Event-Deduplizierung und -Aktualisierung.
Nicola Aebi	Backend-, Frontend- und CI/CD-Entwicklung	Entwicklung von Frontend, CI/CD-Pipeline, APIs und Datenbank inklusive der Event-Kategorisierung.

Tabelle 60: Rollenverteilung im Projektteam

9.1.2 Sitzungsformate

Die Abstimmung im Projekt erfolgte über drei feste Formate, die sich in Zielsetzung und Teilnehmerkreis unterscheiden und in Tabelle 61 zusammengefasst sind. Das wöchentliche Teammeeting fand jeden Montag ab 14:00 Uhr statt und diente der internen Koordination. Das bi-wöchentliche Stakeholder-Meeting wurde jeden zweiten Dienstag um 13:00 Uhr durchgeführt, beginnend am 03.03.2026 und abschliessend am 09.06.2026. An ihm nahmen der Industriepartner Patrick Steinmann, der Betreuer der Bachelorarbeit Joël Schwab sowie das Projektteam teil, fallweise ergänzt durch Prof. Stefan Keller. Die genaue Anwesenheit pro Sitzung ist in den Meeting-Protokollen dokumentiert Anhang Abschnitt F.3.1. Ergänzend wurden bei Bedarf weitere Termine für technische Abstimmungen oder die Behebung konkreter Probleme vereinbart.

Format	Rhythmus	Teilnehmende	Zweck
Teammeeting	Wöchentlich, Montag ab 14:00	Projektteam	Vorbereitung des Stakeholder- Meetings, Rückblick auf den aktuellen Stand, kurze Retrospektive, Besprechung neuer Aufgaben, Sprint-Planung und Priorisierung
Stakeholder- Meeting	Bi-wöchentlich, Dienstag 13:00	Industriepartner, Betreuer, Projektteam, fallweise Prof. Keller	Vorstellung des Fortschritts, fachliche Abstimmung und Rückmeldung der Stakeholder
Nach Bedarf	Flexibel	Projektteam	Technische Abstimmungen und Problembehebung

Tabelle 61: Sitzungsformate im Projekt

9.2 Vorgehen

Dieses Unterkapitel beschreibt das gewählte Vorgehensmodell sowie die eingesetzten Werkzeuge zur Aufgaben- und Zeitverwaltung. Das Projekt folgte einem iterativen, sprintbasierten Ansatz, der die kontinuierliche Weiterentwicklung des Proof-of-Concepts mit den regelmässigen Abstimmungen aus Kapitel 9.1 verband.

9.2.1 Sprintstruktur

Die Projektlaufzeit wurde in acht Sprints zu je zwei Wochen unterteilt, einschliesslich eines einleitenden Kickoff-Sprints und eines abschliessenden Sprints. Der zweiwöchige Rhythmus war auf das bi-wöchentliche Stakeholder-Meeting abgestimmt, sodass am Ende jedes Sprints ein vorzeigbarer Fortschritt präsentiert werden konnte.

9.2.2 Aufgabenverwaltung

Sämtliche Arbeitsschritte wurden als Issues erfasst und einem Agile Board zugeordnet. Das Board gliederte die Issues in die vier Zustände *Uncategorisiert*, *In Progress*, *Zur Validierung* und *Abgeschlossen*, wodurch der Bearbeitungsstand jederzeit nachvollziehbar war. Alle Issues wurden in den wöchentlichen Teammeetings besprochen, bei denen neue Aufgaben erfasst, priorisiert und den Sprints zugeordnet wurden.

9.2.3 Zeiterfassung

Die Erfassung des Arbeitsaufwands erfolgte direkt auf den Issues in YouTrack. Jeder Zeiteintrag wurde einer von vier Kategorien zugeordnet: *Admin*, *Development*, *Documentation* und *Research*. Diese Kategorisierung ermöglichte eine durchgängige Nachvollziehbarkeit des Aufwands und bildet die Grundlage für die Auswertung in Kapitel 9.4.

9.3 Risikomanagement

Dieses Unterkapitel identifiziert die für das Projekt relevanten Risiken und beschreibt die Massnahmen zu deren Eindämmung. Die Risiken wurden zu Beginn des Projekts erfasst und bewertet und während der gesamten Projektlaufzeit fortlaufend überprüft und bei Bedarf angepasst. Tabelle 62 ordnet die Risiken nach Eintrittswahrscheinlichkeit und Auswirkung in einer Matrix ein; Tabelle 63 beschreibt sie anschliessend im Detail samt der jeweiligen Gegenmassnahme.

	Niedrig	Mittel	Hoch	Sehr hoch	Kritisch
Unmöglich					
Unwahrscheinlich		R4	R3	R8	
Möglich		R7	R6	R2	
Wahrscheinlich			R1		
Sehr wahrscheinlich			R5		

Tabelle 62: Risikomatrix zu Beginn des Projekts

Nr.	Risiko	Wahrscheinlichkeit	Gefahr	Massnahme
R1	Änderungen an Event-Websites	wahrscheinlich	hoch	Konfigurationsgetriebene Quellen; Fehler werden geloggt.
R2	Ungenaue LLM-Extraktion	möglich	sehr hoch	Guardrails und Validierung reduzieren Extraktionsfehler.
R3	Fehlende oder falsche Geocodierung	unwahrscheinlich	hoch	Schweizer Adressdatenbank zur Ortsvalidierung.
R4	Duplikate oder veraltete Events in der Datenbank	unwahrscheinlich	mittel	Deduplizierungs- und Merge-Logik verhindert Mehrfacheinträge.
R5	Uneinheitliche Datenqualität der Quellen	sehr wahrscheinlich	hoch	Semantische Extraktion mittels LLM.
R6	Performance-Probleme bei der Verarbeitung	möglich	hoch	Asynchrone Pipeline mit separaten Worker-Stufen.
R7	Frontend zeigt nicht alle relevanten Events	möglich	mittel	Filter- und Suchfunktionen verbessern die Auffindbarkeit.
R8	API oder Datenbank nicht erreichbar	unwahrscheinlich	sehr hoch	Docker-basierte Bereitstellung und Monitoring.

Tabelle 63: Identifizierte Projektrisiken mit Bewertung und Massnahmen

9.4 Zeitmanagement

Dieses Unterkapitel wertet den über YouTrack erfassten Arbeitsaufwand aus. Über die gesamte Projektlaufzeit wurden insgesamt 729 Stunden und 10 Minuten aufgewendet. Tabelle 64 schlüsselt diesen Aufwand nach Teammitglied und Tätigkeitskategorie auf.

Person	Total	Admin	Development	Documentation	Research
Martyn Foreman	375h30m	48h30m	221h00m	72h00m	34h00m
Nicola Aebi	353h40m	30h40m	202h00m	81h00m	40h00m
Total	729h10m	79h10m	423h00m	153h00m	74h00m

Tabelle 64: Aufgewendete Zeit nach Person und Tätigkeitskategorie

Die Verteilung über die Tätigkeitskategorien entspricht den Erwartungen an einen Proof-of-Concept: Mit rund 58 Prozent entfällt der grösste Anteil auf die Entwicklung, gefolgt von der Dokumentation mit etwa 21 Prozent. Administration und Recherche machen mit rund 11 beziehungsweise 10 Prozent vergleichbare Anteile aus. Der Gesamtaufwand ist mit 375 Stunden und 30 Minuten gegenüber 353 Stunden und 40 Minuten zwischen den beiden Teammitgliedern weitgehend ausgeglichen.

Die Unterschiede in der Verteilung lassen sich auf die Rollenaufteilung zurückführen. Auf Martyn Foreman entfällt aufgrund der Projektleitung ein höherer Administrationsaufwand sowie infolge des stärker fokussierten Aufgabenbereichs ein höherer Entwicklungsanteil. Nicola Aebi verzeichnet einen höheren Anteil an Recherche und Dokumentation, was dem inhaltlich breiter gefächerten Aufgabenbereich entspricht.

9.5 Fazit

Das gewählte Projektmanagement hat sich im Verlauf der Arbeit bewährt. Der zweiwöchige Sprintrhythmus und die feste Sitzungskadenz aus wöchentlichem Teammeeting und bi-wöchentlichem Stakeholder-Meeting wurden über die gesamte Laufzeit eingehalten. Lediglich ein bis zwei Termine mussten verschoben werden oder fanden ohne einzelne Teilnehmende statt, was im Rahmen des Üblichen liegt und in den Meeting-Protokollen nachvollzogen werden kann (siehe Anhang Abschnitt F.3.1).

Die klare Rollenaufteilung erwies sich als wesentlicher Erfolgsfaktor. Die Trennung in einen Schwerpunkt Backend und einen Schwerpunkt Frontend, CI/CD und Datenbank ermöglichte es beiden Teammitgliedern, sich im jeweiligen Bereich zu vertiefen, statt die Arbeit breit zu streuen. Die in Kapitel 9.4 ausgewertete Aufwandverteilung bestätigt diese Aufteilung: Der für einen Proof-of-Concept zu erwartende Entwicklungsschwerpunkt bildet sich ebenso ab wie die rollenbedingten Unterschiede zwischen den Teammitgliedern. Da die Arbeit auf der vorangegangenen Studienarbeit aufbaut [1], traten insgesamt nur wenige Überraschungen auf und die Planung musste kaum korrigiert werden.

Auch das Risikomanagement profitierte von dieser Vorerfahrung. Die zu Projektbeginn identifizierten Risiken entsprachen weitgehend den tatsächlich eingetretenen, sodass die Bewertung nur geringfügig angepasst werden musste. Eine Ausnahme bildete die unerwartet hohe Zahl an Quellwebsites, die ihre Struktur innerhalb des Bearbeitungszeitraums von 16 Wochen veränderten und mehrfache Anpassungen der Quellkonfiguration erforderten.

Als besonders wertvoll erwiesen sich die bi-wöchentlichen Sitzungen mit dem Industriepartner. Die dort eingebrachten Rückmeldungen, insbesondere zum Frontend, prägten die Ausgestaltung des Produkts massgeblich. Zugleich erzeugte die Verpflichtung, alle zwei Wochen einen Fortschritt vorzuweisen, einen konstanten und produktiven Arbeitsdruck, der zu einem kontinuierlichen Projektfortschritt beitrug. Verbesserungspotenzial besteht im Umgang mit der Dokumentation: Diese konzentrierte sich zu stark auf die Schlussphase des Projekts und hätte in den bi-wöchentlichen Sitzungen früher und regelmässiger thematisiert werden sollen.

Insgesamt boten die etablierten Strukturen einen verlässlichen Rahmen für die Umsetzung des Proof-of-Concepts. Die folgende Softwaredokumentation beschreibt die in diesem Rahmen entstandene Lösung im Detail.

10 Softwaredokumentation

Die Softwaredokumentation richtet sich an Betreiber, Anwender und Entwickler des Systems. Sie umfasst Installationsanleitung, Benutzerhandbuch und ein Kapitel zur Weiterentwicklung.

10.1 Installation

Setup des wird detailliert im README der GitHub-Repository beschrieben. Es umfasst die Installation von Python, die Einrichtung einer virtuellen Umgebung, die Installation der benötigten Pakete über uv und die Konfiguration der Datenbankverbindung. Alle Schritte sind mit entsprechenden Befehlen und Hinweisen versehen, um eine reibungslose Inbetriebnahme zu gewährleisten. Anhang Abschnitt F.2.5

10.2 Benutzerhandbuch

Das Frontend des Event-Suchroboter-Cockpits ist als kartenzentrierte Single-Page-Anwendung umgesetzt. Beim Aufruf erscheint eine interaktive Karte des Kantons St.Gallen, auf der die extrahierten Events als gruppierte Marker dargestellt werden. Alle Interaktionselemente sind als Overlay über der Karte angeordnet.

10.2.1 Kopfzeile und Darstellung

Die Kopfzeile zeigt links das Kantonswappen und den Titel der Anwendung. Rechts befindet sich ein Symbol-Button zum Wechsel zwischen hellem und dunklem Modus. Die Wahl bleibt für künftige Sitzungen erhalten. Ohne explizite Wahl folgt die Anwendung der Systemeinstellung.

10.2.2 Karte und Event-Marker

Die Marker sind nach Kategorie eingefärbt und tragen den Eventtitel als Beschriftung. Bei geringem Zoom werden nahe beieinanderliegende Marker zu Clustern zusammengefasst. Ein Klick auf einen Cluster zoomt in den Bereich hinein. Ein Klick auf einen einzelnen Marker öffnet ein Popup mit Titel, Beschreibung, Datum, Start- und Endzeit, Adresse, Kategorie, Status, Quellenlink und Zeitpunkt der letzten Aktualisierung.

10.2.3 Suchfunktion

Oben links befindet sich die Suchleiste. Während der Eingabe erscheinen die ersten fünf Treffer mit Titel, Datum und Ort. Ein Klick auf einen Treffer zoomt die Karte direkt zum entsprechenden Event. Über die Schaltfläche mit dem Kreuzsymbol wird die Suche zurückgesetzt und es werden wieder alle Events innerhalb des aktiven Filters angezeigt.

10.2.4 Burgermenü und Filter

Oben rechts öffnet das Burger-Symbol die Filter-Sidebar. Die Sidebar enthält folgende Filter:

- **Zeitraum:** Per Schieberegler kann zwischen vordefinierten Bereichen gewählt werden (heute, diese Woche, dieser Monat usw.). Über die Schaltfläche *Datum wählen* lassen sich Start- und Enddatum auch frei festlegen.
- **Uhrzeit:** Ein zweiseitiger Schieberegler grenzt die Anzeige auf einen Stundenbereich von 0 bis 23 Uhr ein.
- **Kategorien:** Ausgewählte Kategorien erscheinen als Chips. Über die -Schaltfläche öffnet sich eine Liste aller verfügbaren Kategorien. Ein Klick auf einen aktiven Chip entfernt diesen wieder.
- **Gebiet:** Wählbare Teilregionen des Kantons (z.B. Rheintal, Linthgebiet). Die Auswahl blendet die zugehörige Polygonfläche auf der Karte ein und schränkt die Marker auf das gewählte Gebiet ein.
- **Quelle:** Einschränkung auf *Eventseiten* oder *Gemeindeseiten*.
- **Export:** Die aktuell gefilterte Eventliste kann als CSV- oder JSON-Datei heruntergeladen werden.

Änderungen an den Filtern wirken sofort. Die Karte wird neu gezeichnet und die Marker werden entsprechend aktualisiert. Ein erneuter Klick auf das X-Symbol schliesst die Sidebar wieder.

10.3 Weiterentwicklung

Dieses Kapitel beschreibt mögliche Weiterentwicklungen des Event-Suchroboter-Cockpits aus Sicht der Softwarearchitektur und Implementierung. Während der technische Bericht die Weiterentwicklung allgemein einordnet, richtet sich dieses Kapitel an zukünftige Entwicklerinnen, Entwickler und Architektinnen. Im Vordergrund stehen konkrete funktionale und nicht-funktionale Erweiterungen, die auf dem bestehenden Proof-of-Concept aufbauen und für eine spätere Produktivsetzung relevant sind.

Die folgenden Punkte sind als technische Roadmap zu verstehen. Sie priorisieren keine konkrete Umsetzung, zeigen jedoch auf, an welchen Stellen das System erweitert, stabilisiert oder fachlich ausgebaut werden kann.

10.3.1 Datenquellen

- Integration offizieller APIs von Eventfrog und Guidle
- Erweiterung auf weitere Kantone
- Unterstützung zusätzlicher Social-Media- und Community-Plattformen
- Automatische Qualitätsbewertung einzelner Quellen
- Regelmässige Prüfung, ob bestehende Quellen noch erreichbar und aktuell sind

10.3.2 Extraktion und Klassifizierung

- Batching von LLM-Anfragen zur Reduktion von Latenz und Kosten
- Selektive Aktivierung des LLMs nur bei unsicheren oder komplexen Quellen
- Einsatz kleinerer spezialisierter Modelle für wiederkehrende Extraktionsaufgaben
- Verfeinerung des Kategoriensystems
- Mehrstufige Klassifikation mit Konfidenzbewertung
- Klar getrennte Kategorien: Schema.org hat zum Teil sehr ähnliche Kategorien, z.B. `MusicEvent` vs. `Festival`
- Manuelle Bestätigung bei niedriger Klassifikationssicherheit

10.3.3 Deduplizierung und Aktualisierung

- Verbesserung der Event-Matching-Verfahren über mehrere Quellen hinweg
- Trennung zwischen echten Duplikaten und legitimen Eventaktualisierungen
- Historisierung von Eventänderungen
- Quellenübergreifende Konsolidierung von Metadaten
- Sichtbare Darstellung der Merge- und Duplikatbeziehungen im Frontend

10.3.4 Geo-Komponente

- Integration zusätzlicher Geokodierungsdienste für Sonderfälle
- Unterstützung flächiger Geometrien für Festivals oder grössere Veranstaltungsareale
- Unterstützung routenbasierter Veranstaltungen wie Umzüge, Läufe oder Touren
- Erweiterung der zeitlich-räumlichen Koinzidenzanalyse
- Verbesserung der Validierung bei unvollständigen oder mehrdeutigen Ortsangaben

10.3.5 Frontend

- Verlagerung der Gebietsfilterung in den API-Dienst
- Reduktion der übertragenen Datenmenge durch serverseitige Filterung
- Erweiterte Such- und Exportfunktionen
- Rollen- und benutzerspezifische Ansichten

- Visualisierung der Datenqualität, Quellenlage und Klassifikationssicherheit pro Event

10.3.6 Betrieb und Wartung

- Komplette Containerisierung und automatisiertes Deployment
- Geregelte Update-Zyklen für Crawler, Quellenkonfigurationen und Datenmodelle
- Strukturierter Umgang mit Datenbankmigrationen
- Erweiterte Monitoring- und Alerting-Funktionen
- Qualitätsmetriken für Quellen-Aktualität, Extraktionsgüte und Klassifikationskonfidenz
- Betriebskonzept für Logs, Fehleranalyse und Wiederverarbeitung fehlerhafter Pipeline-Stufen

10.3.7 Fachliche Erweiterungen

- Automatische Risikoeinstufung von Veranstaltungen
- Unterstützung polizeilicher Gefährdungsbeurteilungen
- Priorisierung von Events anhand definierter fachlicher Kriterien
- Integration externer Lage- und Kontextdaten
- Prüfung, welche Datenmerkmale für eine belastbare Gefährdungsanalyse erforderlich sind

10.3.8 Produktivsetzung

- Durchführung einer Machbarkeitsstudie für den produktiven Einsatz
- Evaluation rechtlicher und organisatorischer Anforderungen
- Betriebs- und Supportkonzept
- Benutzerstudien mit Endanwenderinnen und Endanwendern
- Pilotbetrieb bei der Kantonspolizei St.Gallen
- Bewertung, ob ein produktives Lage- und Informationscockpit auf Basis des Proof-of-Concepts realistisch umgesetzt werden kann

Hilfsmittelverzeichnis

Aufgabenbereich	Tool
Literatur-Recherche	Google, ChatGPT, Claude
Übersetzung	DeepL, Google Translate
Coding	Visual Studio Code, PyCharm, Claude Code, Codex
Textoptimierung, Rechtschreibe- und Grammatikprüfung	DeepL, Claude, Codex
Zusammenarbeit und Projektmanagement	Outlook, Teams, GitLab, YouTrack,