

# Documentation

Studienarbeit  
Documentation

## What's up in RJ

Semester: Fall 2023



Version: 1.0  
Date: 2023-12-22

**Project Team:** MichaelENZler  
Fabio Stocker

**Project Advisor:** Prof. Frank Koch  
**Industry Partner:** Michael Güntensperger, AdaptIT GmbH



School of Computer Science  
OST Eastern Switzerland University of Applied Sciences

# Abstract

This project is about creating a simple-to-use web application that centralizes a list of events and organizers in the vicinity of Rapperswil-Jona. Its purpose is to present all kinds of fun, interesting and exciting events to the regional residents.

The application is built from scratch, there is no existing platform used as a base. With a centralized platform it is possible for organizers to easily publish their own events. Anyone can sign up as an organizer, making it appealing to all people and organizations looking to advertise their events. Users of the platform benefit from this by being able to find events advertised by regional, national, and even global organizers. After creating a user account, they can simply subscribe to the newsletter, delivered twice a month, informing them about new and upcoming events that match their preferences.

The approach is to create a web application consisting of a frontend, backend, and database layer. Deployed with a cloud hosting provider, the application is widely accessible to all users. Following the initial inception phase, the application underwent design and development during the elaboration phase. The implementation of the entire application was then done during the construction phase. Additionally, various tests were conducted, including usability tests with real end users to evaluate the user experience.

This first iteration is a good starting point to create a seamless platform where users and organizers are able to interact with each other. In future steps, it is recommended to implement additional features to increase the application's appeal. This project has considerable potential to become a successful platform for event organizers and attendees.

# Management Summary

## Initial Situation

Rapperswil-Jona is an active city with a diverse cultural program. There are a lot of activities and events people can visit or participate in. However, if you want to be up-to-date with the latest and greatest events, you currently have to visit many different websites or read through the daily newspaper. There is no single source of information. This gives off a strange vibe that there isn't really a lot going on in the region. That's where 'What's up in RJ' comes in. The goal of this project is to centralize this information and show the residents that there is actually a lot to do in Rapperswil-Jona. With a central platform, it is possible for organizers to publish their own events. Anyone can sign up as an organizer, making it very easy to create your first events. Normal users of the platform benefit from this by being able to find interesting events advertised by many different organizers. After creating a user account, you can simply subscribe to the newsletter, delivered twice a month, informing you about new and upcoming events that match your preferences. This project is built from scratch, there is no existing platform used as a base.

## Procedure and Technologies

The approach is to create a web application consisting of a frontend, backend, and database layer. The frontend is implemented with the Angular framework and its single-page application feature. To style all pages easily and consistently, the Tailwind CSS library is used. Furthermore, the frontend depicts an elegant user interface and communicates with the backend through a simple REST API. The backend in turn implements the business logic and communicates with the database and handles all data flows. It is based on Node.js, utilizing the Express web application framework to expose its API. The database layer, based on PostgreSQL, stores the data persistently. All layers together build the web application, deployed automatically to DigitalOcean using its App Platform. The application is accessible to all users through a domain, just like any other website. SendGrid handles all personalized newsletter deliveries, and is accessed through their easy-to-use API. Thanks to the different layers, separation of concerns is achieved and any layer can easily be replaced by another technology stack with minimal effort.

## Results

The final result is a Minimum Viable Product. Everyone can access the application through their browser and is able to create an account. Organizers are allowed to create, update and delete events. All other normal users are able to view, search, and filter these events. Checking out various organizers is also possible. User accounts additionally can subscribe to the newsletter and set their event preferences. They then receive a personalized newsletter twice a month with upcoming events that

match their chosen preferences. With these core features, the application is on a basic level ready for production. However, there are many things that can still be improved, and a ton of new features that can be implemented in future iterations. They may even be necessary to improve the attractiveness of the application. For example, an image upload feature to set an avatar for an organization, like their logo, or have a primary image for an event, using a picture of a past iteration of the event. Other ideas to implement are: a map view to geographically see where events are located, a calendar view just like your calendar at home, and a more sophisticated search functionality. In conclusion, this project is a good starting point to create a platform to share events with the residents of Rapperswil-Jona. The project can be further developed to become a successful platform for event organizers and attendees.

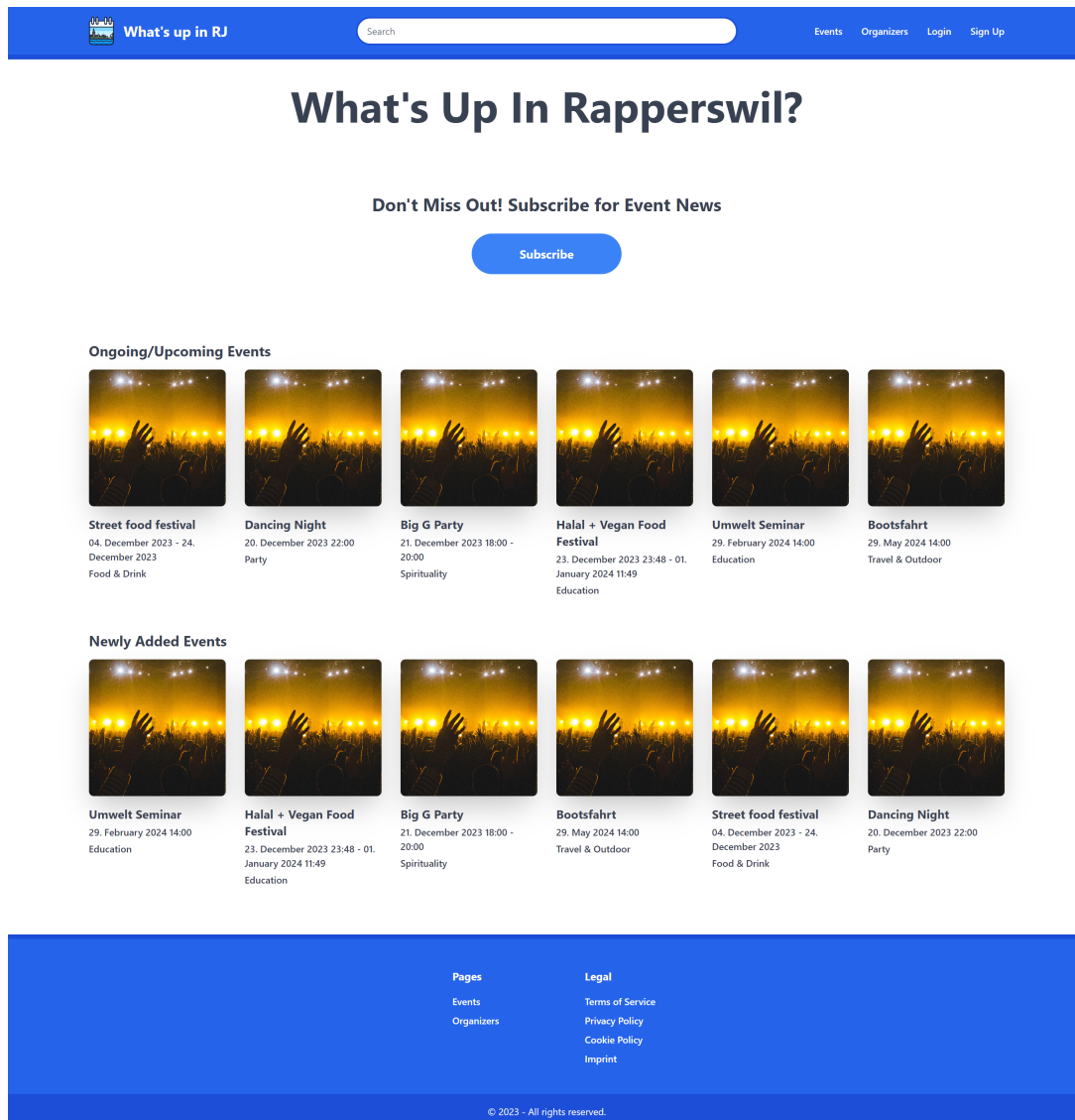


Figure 1: Home page for visitors

# Contents

<b>I</b>	<b>Project Overview</b>	<b>1</b>
<b>1</b>	<b>Overview</b>	<b>2</b>
1.1	Initial Situation . . . . .	2
1.2	Task . . . . .	2
1.3	Conditions . . . . .	2
1.4	System Context . . . . .	3
<b>II</b>	<b>Product Documentation</b>	<b>4</b>
<b>2</b>	<b>Requirements</b>	<b>5</b>
2.1	Functional Requirements . . . . .	5
2.2	Non-Functional Requirements . . . . .	8
2.3	Minimum Viable Product . . . . .	11
<b>3</b>	<b>Domain Analysis</b>	<b>12</b>
3.1	Domain Model . . . . .	12
<b>4</b>	<b>Architecture</b>	<b>14</b>
4.1	Overall Architecture . . . . .	14
4.2	Backend Architecture . . . . .	15
4.2.1	Component . . . . .	15
4.2.2	Structure . . . . .	16
4.3	Frontend Architecture . . . . .	17
4.3.1	Frontend Structure . . . . .	18
4.3.2	Request Flow . . . . .	19
4.4	UI Mockups . . . . .	21
4.5	Deployment Architecture . . . . .	27
4.6	Technologies . . . . .	28
<b>5</b>	<b>Quality Measures</b>	<b>30</b>
5.1	Quality Assurance . . . . .	30
<b>6</b>	<b>Test Plan</b>	<b>32</b>
6.1	Introduction . . . . .	32
6.2	Test Objectives . . . . .	32
6.3	Test Strategy . . . . .	32
6.4	Test Environment . . . . .	32
6.5	Test Specifications Functional Requirements . . . . .	33

6.6	Test Specifications Non-Functional Requirements . . . . .	34
6.7	End User Tests . . . . .	35
6.8	Test Schedule . . . . .	36
6.8.1	CI/CD Tests . . . . .	36
6.8.2	Integration Tests . . . . .	36
6.8.3	End User Tests . . . . .	36
<b>7</b>	<b>Implementation</b>	<b>37</b>
7.1	Visual Design . . . . .	37
7.2	Backend API Endpoints . . . . .	37
7.2.1	Special API Endpoints . . . . .	37
7.2.2	Rate Limiting . . . . .	38
7.3	Database . . . . .	38
7.4	Features . . . . .	40
7.4.1	Event Handling . . . . .	40
7.4.2	Newsletter . . . . .	43
7.4.3	Search . . . . .	44
7.4.4	Filtering . . . . .	44
7.4.5	Home . . . . .	45
7.4.6	Organizers . . . . .	46
7.4.7	Account . . . . .	46
7.4.8	Message . . . . .	46
7.5	Error Handling . . . . .	47
7.5.1	Backend . . . . .	47
7.5.2	Frontend . . . . .	48
7.6	Security . . . . .	48
7.6.1	Authentication . . . . .	48
7.6.2	Password Hashing . . . . .	49
7.6.3	Input Validation . . . . .	49
7.7	Code Documentation . . . . .	50
7.8	Deployment . . . . .	50
7.8.1	Scaling . . . . .	50
7.8.2	Frontend Environment Script . . . . .	52
7.9	Testing . . . . .	52
7.9.1	Automated Testing . . . . .	52
7.9.2	Manual Testing . . . . .	53
<b>8</b>	<b>Results</b>	<b>54</b>
8.1	Functional Requirements . . . . .	54
8.2	Non-Functional Requirements . . . . .	54
8.3	Minimum Viable Product . . . . .	55
<b>9</b>	<b>Conclusion</b>	<b>56</b>
9.1	Result Reflection . . . . .	56
9.2	Goal Achievement . . . . .	56
9.3	Future Vision . . . . .	56

<b>III Project Documentation</b>	<b>59</b>
<b>10 Project Plan</b>	<b>60</b>
10.1 Resources . . . . .	60
10.2 Processes and Meetings . . . . .	60
10.3 Schedule . . . . .	61
10.4 Organization and Roles . . . . .	62
10.5 Risk Management . . . . .	63
10.6 Planning Tools . . . . .	65
10.7 Git . . . . .	66
<b>11 Time Tracking Report</b>	<b>67</b>
<b>Glossary</b>	<b>70</b>
<b>List of Figures</b>	<b>72</b>
<b>List of Tables</b>	<b>73</b>
<b>Listings</b>	<b>74</b>
<b>IV Appendix</b>	<b>75</b>
<b>12 Test Plans</b>	<b>76</b>
12.1 Test Plan #1 . . . . .	76
12.1.1 Introduction . . . . .	76
12.1.2 Test Objectives . . . . .	76
12.1.3 Test Strategy . . . . .	76
12.1.4 Test Environment . . . . .	77
12.1.5 Test Schedule . . . . .	77
<b>13 Test Reports</b>	<b>78</b>
13.1 Functional Test Protocol 29.11.2023 . . . . .	78
13.2 Non-Functional Test Protocol 29.11.2023 . . . . .	79
13.2.1 Reports from Test Users . . . . .	82
<b>14 Application Screenshots</b>	<b>88</b>
<b>15 Backend API Endpoints</b>	<b>100</b>
<b>16 Task</b>	<b>121</b>

Part I

**Project Overview**



# Chapter 1

## Overview

This document includes all the information about the project ‘What’s up in RJ’. It is divided into four parts: Project Overview, Product Documentation, Project Documentation and Appendix. The first part should give the reader the context and a short introduction to the project. In the Product Documentation, information about requirements, architecture, implementation and results of the product itself are given. The Project Documentation part contains information about the project itself, like the project plan and time tracking. The appendix holds a lot of additional information about the project and product, such as reports and protocols of meetings.

### 1.1 Initial Situation

The product ‘What’s up in RJ’ is an idea of the industry partner ‘AdaptIT GmbH’. As there are a lot of events in the region of Rapperswil-Jona, and there is no central place to view all these events, the idea of ‘What’s up in RJ’ was born. The idea is to create a platform where all those events are listed. The focus of this project is the implementation of a platform.

### 1.2 Task

The task is to create a web application, which lists all the events in the region of Rapperswil-Jona. For an organizer, it should be possible to create an event. For a user, it should be possible to search and filter for events. Additionally, as a user, there should be a possibility to subscribe to a newsletter. The final goal is give the user the possibility to find an event with few clicks for a specific date or a specific category.

### 1.3 Conditions

This project was done in the context of a ‘Studienarbeit’ at the Eastern Switzerland University of Applied Sciences (OST). The team consists of two students of the Computer Science department. The project was done in the time from 18.09.2023 to 22.12.2023, which is a total of 14 weeks, or one semester. As it is a ‘Studienarbeit’, one student should work 240 hours on the project, meaning a student should work roughly 17 hours per week. For every 30 hours, one credit point is granted. A student will receive 8 credit points for this project.

## 1.4 System Context

No existing infrastructure exists for this project. Therefore, the web application has to be implemented from scratch. There will be three layers: frontend, backend and database. The frontend is the part visible to the user. It depicts the application. The backend implements the business logic, while the database stores the data.

**Part II**

**Product Documentation**

# Chapter 2

# Requirements

## 2.1 Functional Requirements

In the requirements, a distinction is made between non-optional and optional requirements. Non-optional requirements need to be implemented, while optional ones do not. The optional requirements are marked with a \*. All functional requirements are listed as use cases in table 2.1 and figures 2.1 and 2.2. They show the connection between actors and use cases in use case diagrams.

### Actors

Following actors are defined for the system:

#### Organizer

Organizers are responsible for planning and executing events within the system. They have the authority to create, manage, and oversee their own event details after successful login.

#### User

Users are individuals interested in discovering and participating in events. They can search for specific events and apply filters to refine their event choices. They are also able to create an account to become a registered user.

#### Registered User

Registered users are users that have created an account. In addition to the functionality of a normal user, they can also subscribe to newsletters.

#### Administrator\*

The administrator plays a pivotal role in managing the platform. They are responsible for overseeing organizer accounts and the events they create. Additionally, administrators are in charge of approving events for public visibility.

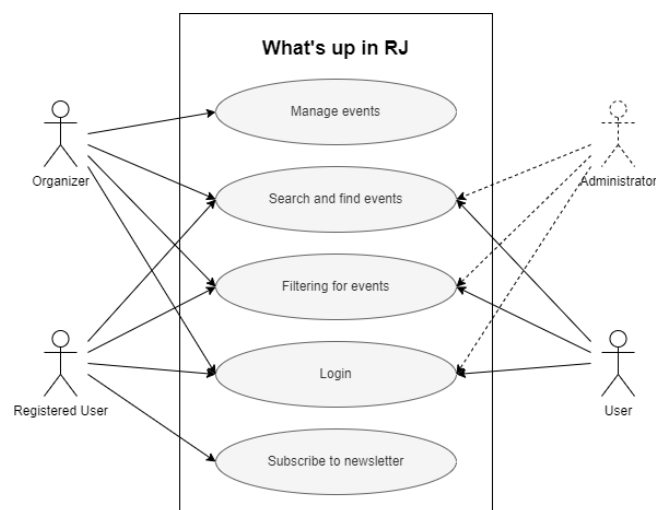
### Use Cases

#	Actor	Goal	Description
UC1	Organizer	Manage events	The organizer can create, delete and edit their own events.
UC2	User Registered User Organizer Administrator	Search and find events	All system actors have the capability to view events and locate them through the utilization of the search functionality.
UC3	User Registered User Organizer Administrator	Filtering for events	All actors possess the functionality to apply event filters, such as filtering events by category, e.g. sports.
UC4	User Registered User Organizer	Login	Users and organizers of the platform have the capability to register and create an account, enabling them to become registered users with persistent access through login credentials.
UC5	Registered User	Subscribe to newsletter	Registered users possess the ability to select categories aligned with their interests, enabling them to subscribe to newsletters accordingly.
UC6*	Administrator	Manage organizers	The administrator oversees organizer accounts and can perform standard CRUD operations on them.
UC7*	Administrator	Approve events	The administrator can approve events for public visibility.
UC8*	Organizer	View dashboard and statistics	Organizers are provided with a dedicated dashboard where they can access statistics, including metrics i.e. event clicks.
UC9*	Organizer	Create organizer page	Organizers have the capability to oversee and customize their individual pages, allowing them to present themselves as desired.
UC10*	Registered User	Get AI based newsletter	Users receive newsletters generated by an AI system, personalized according to their interests on events.
UC11*	User Registered User	Use mobile app	Users are provided with the opportunity to install a mobile application on their smartphones, enabling them to receive push notifications.
UC12*	User Registered User Organizer Administrator	View news ticker	The web application displays a news ticker on the homepage.
UC13*	Organizer	Upload pictures	Organizers can add pictures to various entities (events, organizers page, etc.).

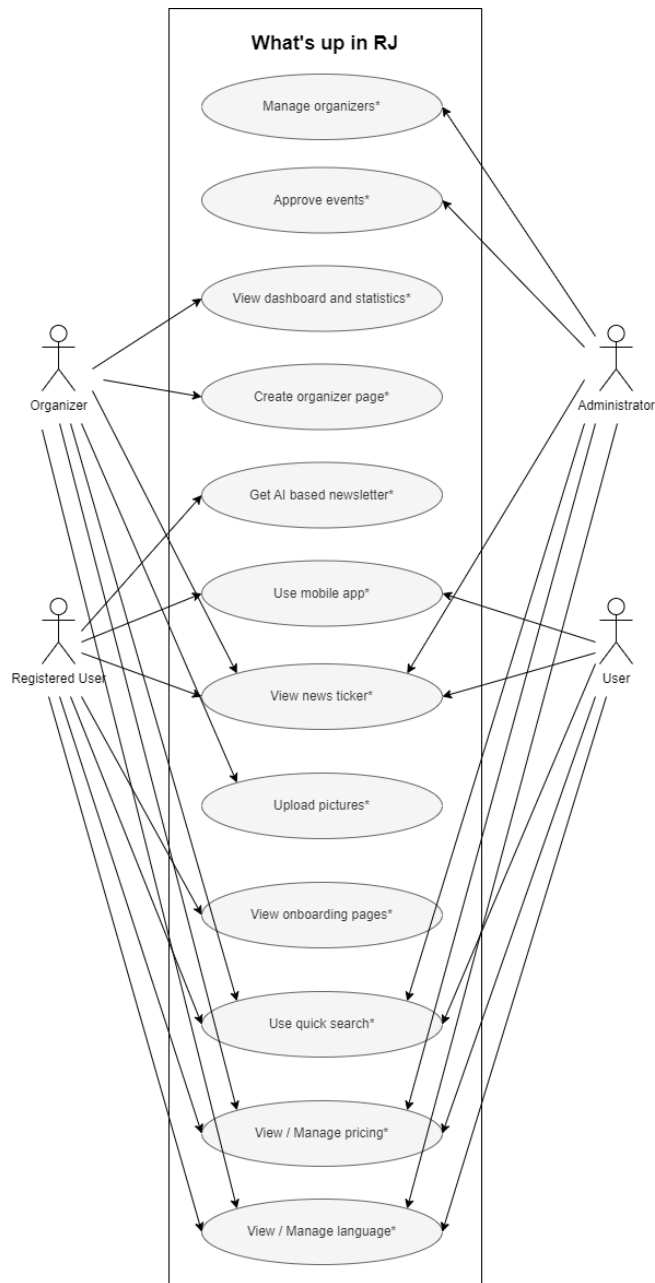
Continued on next page

#	Actor	Goal	Description
UC14*	Registered User	View onboarding pages	After signing up, the registered user can add their preferences and sign up for the newsletter via onboarding pages.
UC15*	User Registered User Organizer Administrator	Use quick search	The system actors can search for events via quick search buttons on the home page (e.g. 'today', 'tomorrow', or 'this week')
UC16*	User Registered User Organizer Administrator	View / Manage Pricing	The organizer can add a price to their event and all actors can see that price. Additionally, the organizer can supply e.g. a 'ticketmaster' URL and the price will then be displayed automatically by fetching it from the respective API.
UC17*	User Registered User Organizer Administrator	View / Manage Language	To see what languages are supported at an event, the organizer can add a language to their event and all actors can see that language.

**Table 2.1:** All use cases



**Figure 2.1:** Use case diagram with required use cases with optional actor 'Administrator' included



**Figure 2.2:** Use case diagram showing optional use cases

## 2.2 Non-Functional Requirements

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

### Collaborative

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

Acceptance Criteria: All required features are implemented.

### **Performance**

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

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

### **Response Time**

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

Acceptance Criteria: The pages load in under 200ms.

### **Responsiveness**

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

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

### **Browser Compatibility**

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

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

### **Accessibility**

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

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

### **User Satisfaction**

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

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

### **Scalability**

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

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



## **Error Handling**

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

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

## **Security**

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

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

## **Privacy**

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

Acceptance Criteria: Privacy regulations are implemented and tested.

## **Password Security**

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

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

## **User Data Isolation**

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

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

## **Modularity**

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

Acceptance Criteria: Business logic is modular and extensible.

## **API Testing**

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

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

## Deployment

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

Acceptance Criteria: The implemented functionality is deployed on DigitalOcean.

## 2.3 Minimum Viable Product

The Minimum Viable Product, short MVP, consists of:

- Events can be created, deleted, and edited by organizers.
- Search for events is possible.
- Events can be filtered by category.
- Users can register and login.
- Users can subscribe to a newsletter.

# Chapter 3

## Domain Analysis

### 3.1 Domain Model

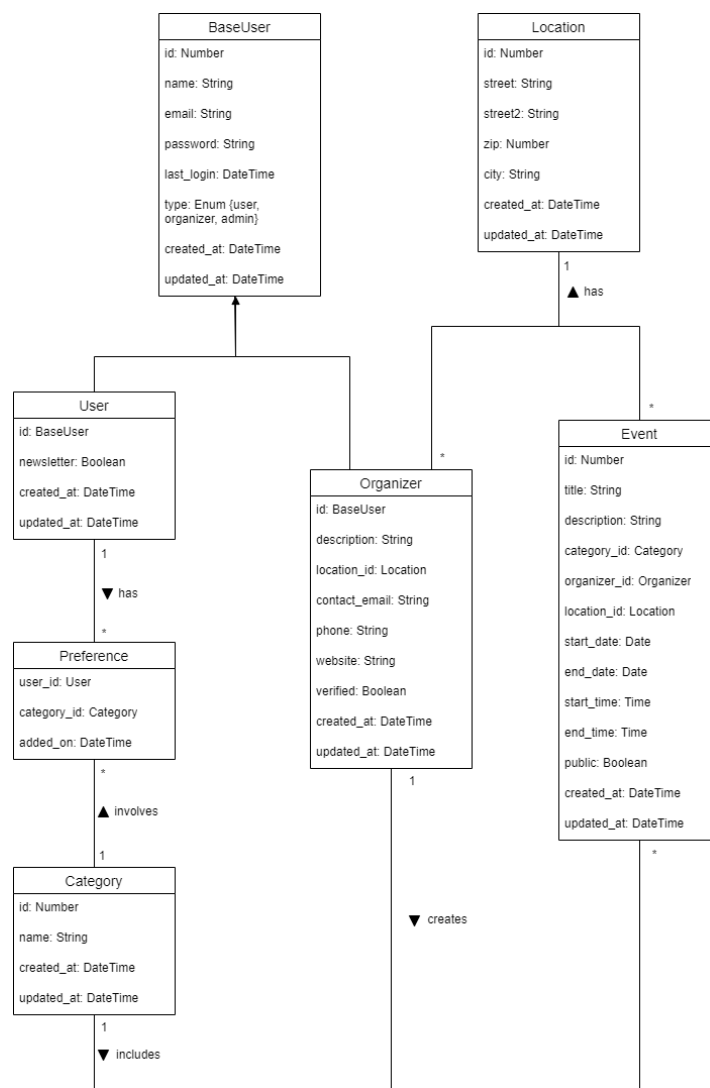


Figure 3.1: Domain model for the MVP

The main actors of the application, user and organizer, are both extending the base user entity. This is done to avoid code duplication and to allow the application to be easily extended with more user types in the future. As seen in the diagram, an organizer and a user have different attributes and different relations to other entities. The user has the additional attribute of newsletter and has a relation to preference, as a user can have multiple preferences. These preferences belong to a category, which is also its own entity. Each event is able to have a single category. Thanks to the relation between user and preference, it is possible to create a relation between user and category. Thus, it is possible to create a newsletter for a user based on the user's preferences. The organizer on the other hand has several more attributes: a description, phone, website, etc. Those attributes are a preparation for the optional requirement of the organizer page and are not needed for the MVP. The organizer has a relation to the event as the organizer is the one creating the event. The central element of the domain model is the event, which is the main entity of the application. An event has several attributes: a title, description, start date, end date, start time, end time, etc. Finally, the location is an entity that has a relation to the event and the organizer. This is done to avoid code duplication as both events and organizers can have the same type of location. However, each entity has its own location object and relation.

## Notes

This domain model (see figure 3.1) contains all elements from the MVP and the functional requirements. Optional requirements are, except for some attributes, not included.

# Chapter 4

## Architecture

### 4.1 Overall Architecture

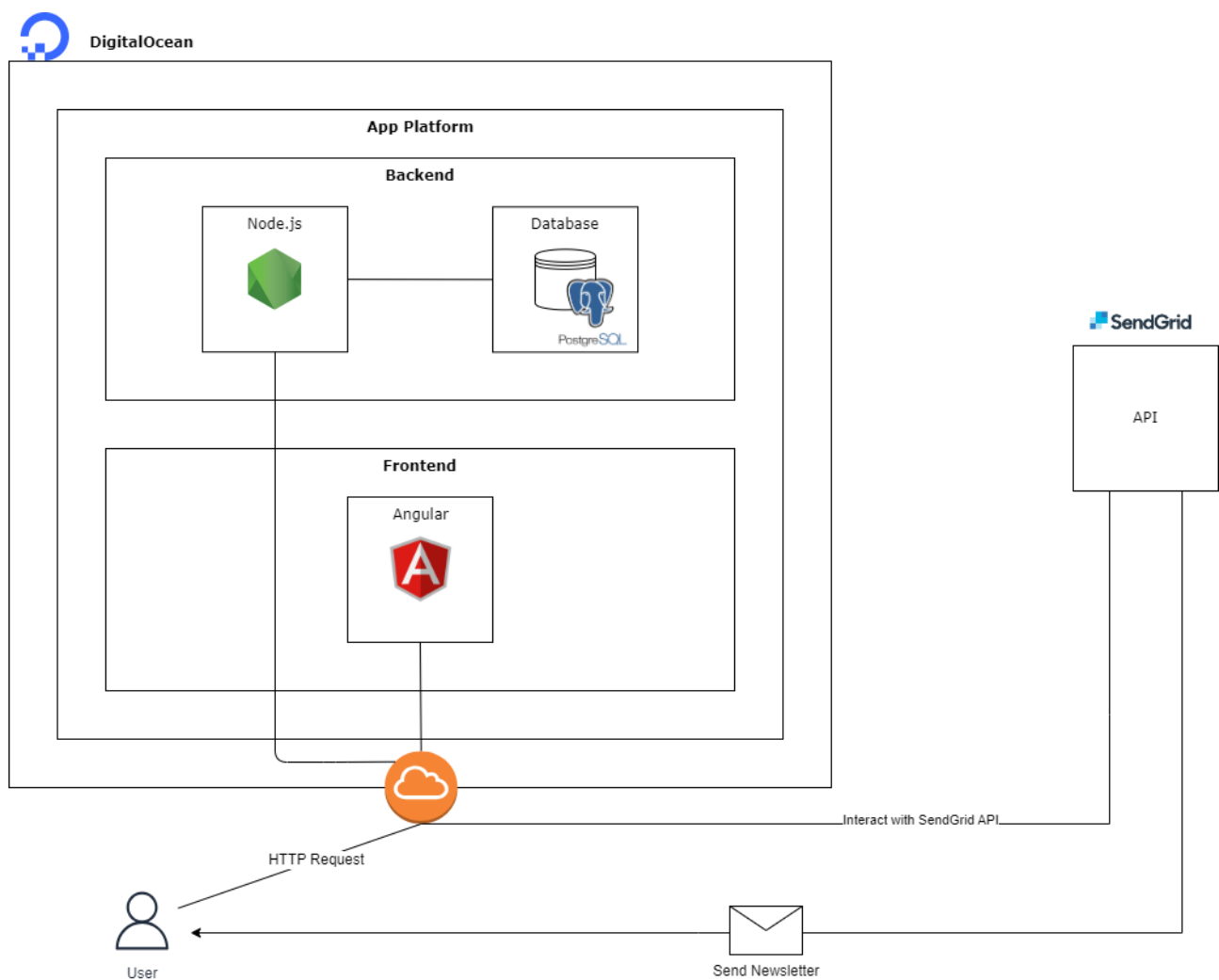


Figure 4.1: Architecture of the project

The architecture shown in figure 4.1 displays the architecture of the project on a high level. The project is deployed on DigitalOcean using the App Platform feature. Both frontend and backend are running in separate containers. The mail service used for the newsletter is not part of DigitalOcean but rather a third-party service. The frontend is a web application built with Angular. The backend is a REST API built with Express running on Node.js. It communicates directly with the MySQL database. Furthermore, the backend can access the internet to update newsletter subscriptions and send emails via the mail service. The user can access the website via a web browser. First a request is sent to the frontend to get the page content. After a successful request, the browser sends another request to the backend to get the data for the page, if applicable. Finally, the page will then be displayed to the user, optionally with the data provided by the backend. The user will receive a newsletter by email if they have subscribed to it directly from the mail service.

## 4.2 Backend Architecture

Node.js and TypeScript as well as the Express framework are used to build the backend. The decision for this is described in the technologies section 4.6. In the following sections, the architecture and the structure of the backend is described in more detail.

### 4.2.1 Component

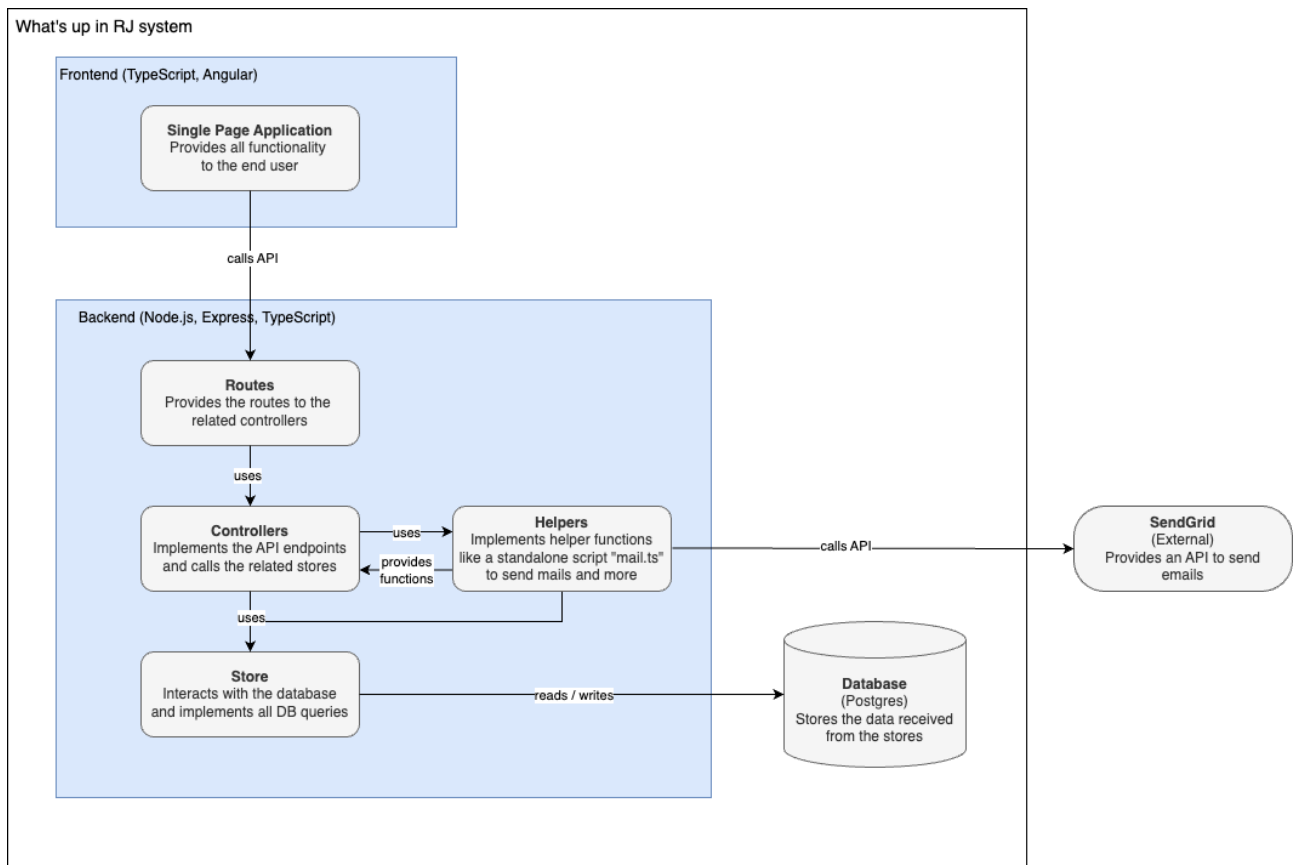


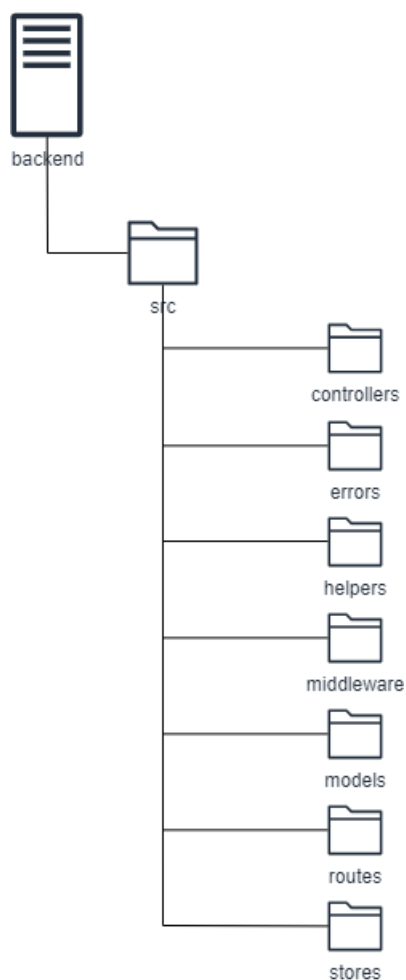
Figure 4.2: Component Diagram

The component diagram in figure 4.2 shows the components from a backend perspective. The frontend is, from this perspective, a single component: A web application. It provides the user interface, where all the functionality is accessible. The backend is accessed through API calls, where the route subcomponent handles the incoming requests. The route subcomponent then forwards the traffic to the correct controller subcomponent, which handles the logic of the request. The controller subcomponent also calls the related store subcomponent, which handles all database queries. The store subcomponent is the only part of the code that directly reads from and writes to the database. The helper subcomponent contains standalone scripts such as the ‘mail.ts’ script, which calls the SendGrid API to send out all newsletter emails. Finally, the database is a component on its own, which stores all the data.

## Notes

The diagram is a generalization of the backend architecture and does not include all components. There are also components for models, middlewares, helpers and errors. They will be described in more detail during the following section.

### 4.2.2 Structure



**Figure 4.3:** Backend Structure

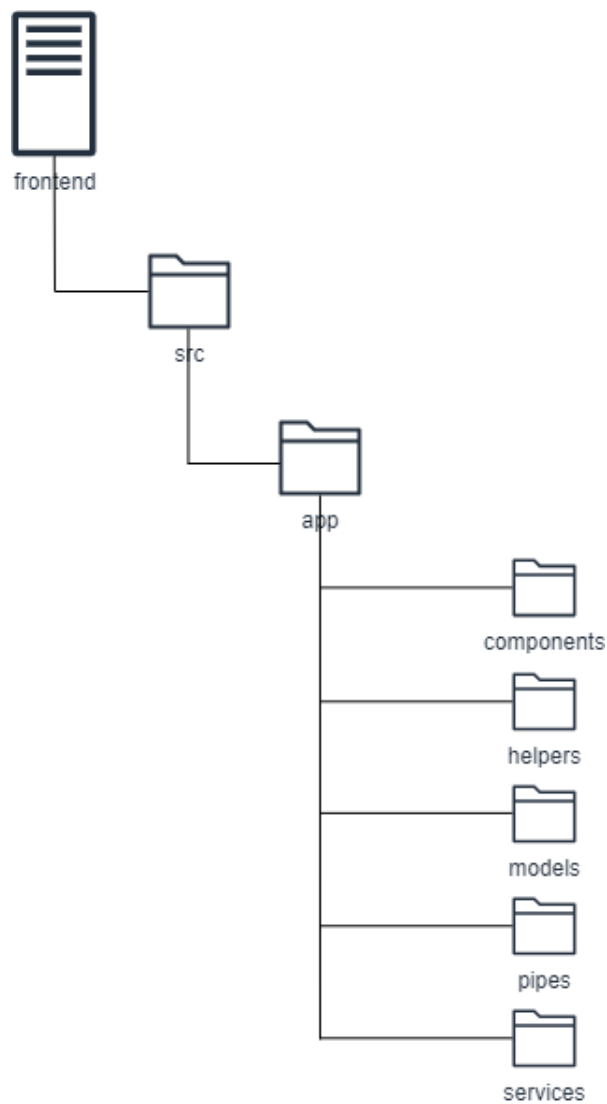
The figure 4.3 shows the structure of the backend. This is a common structure for a Node.js application using the Express framework, with some additional folders needed for this project. The 'src' directory contains all the source code of the backend. The directories for routes, controllers and stores contain the files for the components described in the component diagram, see 4.2. The 'errors' directory contains all the error classes used to handle and throw specific errors of the API. The directory named 'helpers' includes all relevant helper functions. With those helper functions it is possible to reduce code duplication and to make the code more readable. The 'middleware' directory contains all middleware functions, which usually get called in between the route and controller subcomponents. An example would be authentication, where a route is locked behind authentication, the request gets checked for a valid token before being handled in the controller. The models are used to define the structure of the data stored in the database. All database models and associations are stored in the 'models' directory, where they will be set up by Sequelize on startup. Finally, the entry point of the backend is the 'main.ts' file, which is stored in the 'src' directory. This file initializes the backend and starts the server.

### 4.3 Frontend Architecture

The Angular framework together with the TypeScript and SCSS languages are used to build the frontend. The decision for this is described in the technologies section 4.6. Angular implements a component-based architecture. This means that the application is built out of components that are reusable and can be combined to create more complex components. All components are then combined to create the application. A bootstrap component is defined that acts as the root component of the single page application. The next section describes the structure of the frontend in more detail.



### 4.3.1 Frontend Structure



**Figure 4.4:** Structure of the frontend

In the figure 4.4 the structure of the frontend is shown. The 'src' directory not only contains the visible folders but also some core files:

- **index.html:** The main HTML file of the application. It contains the root component of the application.
- **main.ts:** The main TypeScript file of the application. It contains the bootstrap function that starts the application.
- **styles.scss:** The main SCSS file of the application. It contains the global styles of the application.

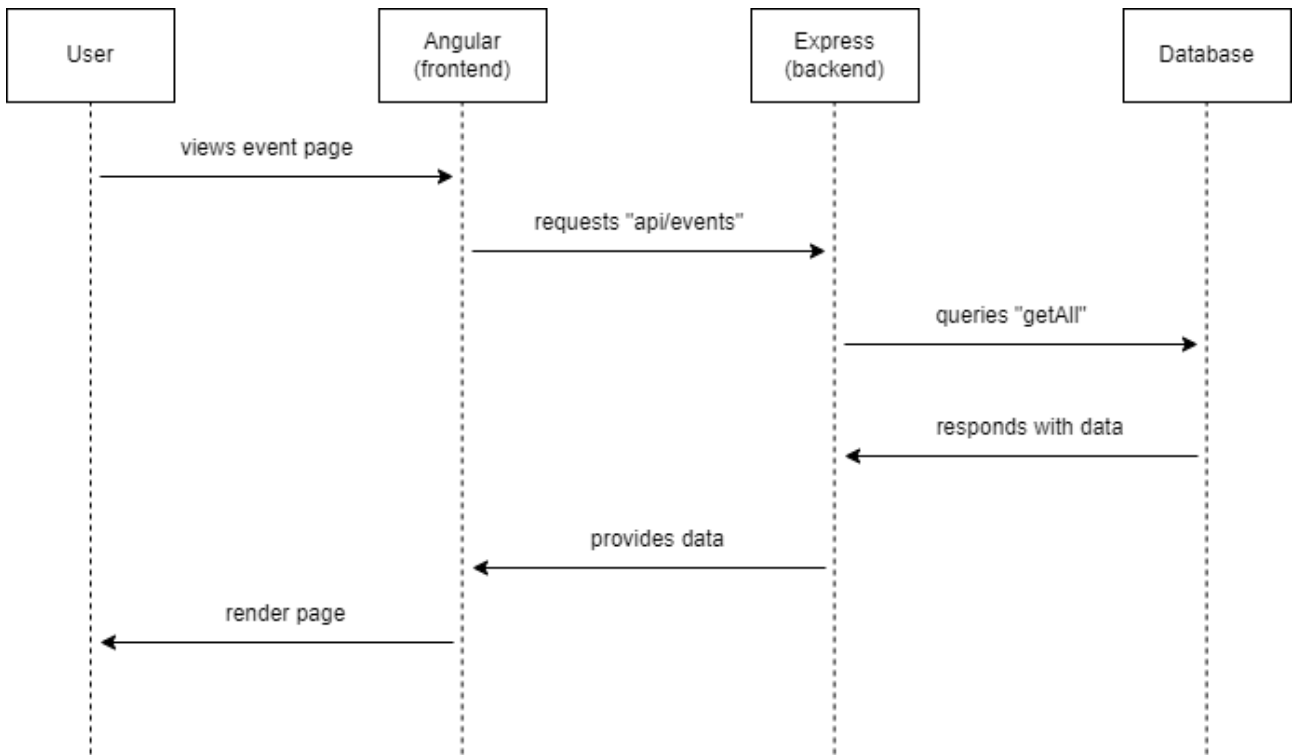
The actual components are located in the 'app' directory. Inside the 'app' directory are additional core files of the application:

- **app-routing.module.ts**: The routing module of the application. It contains all the routes of the application including guards that specify when a user can activate or deactivate a route, and additional metadata.
- **app.component.html**: The HTML template of the root component.
- **app.component.scss**: The SCSS styles of the root component.
- **app.component.spec.ts**: The unit tests of the root component.
- **app.component.ts**: The TypeScript file of the root component handling the logic.
- **app.module.ts**: The main module of the application. It contains all the components, services, etc. that are used in the application.

The subdirectory ‘components’ itself contains all the reusable components of the application. This means that there are several subdirectories for every single component e.g. home, message, etc. Each of those directories contain a HTML template, a SCSS styles, a spec file for unit tests, and a TypeScript file for the logic of the component. All helper functions that are used in multiple components are located in the ‘helpers’ directory. They are used to avoid code duplication of functions that are being used in multiple components, e.g. a function to format a date. The ‘models’ directory contains all custom TypeScript typings. The typings contain the structure of each object received from the backend, and also the object that gets sent to the backend (to create or update). Additionally, a few other interfaces are defined for various components and guards. With the ‘pipes’ directory it is possible to create custom pipes that can be used directly in the HTML templates, making it easier to use helper functions. The ‘services’ directory is used to define the functionality of guards and communicate with external resources (as seen from the frontend’s perspective), such as the backend API. Each service consists of a TypeScript file for the logic, and a spec file to test the service. At least one service exists for each backend API endpoint, e.g. ‘/api/me’ uses the MeService, ‘/api/events’ uses the EventService, etc.

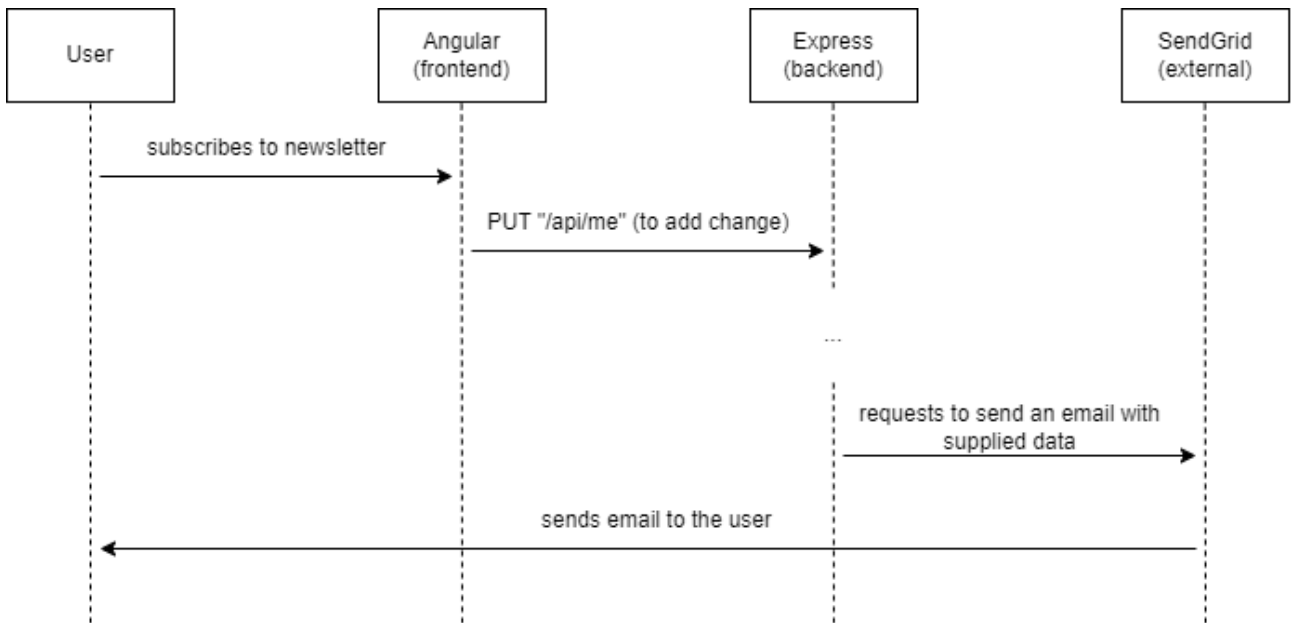
### 4.3.2 Request Flow

The following graphic shows the flow of a request from the user when accessing the events page. The flow for the other pages is similar, but sometimes additional data is supplied by the user, e.g. when creating a new event.



**Figure 4.5:** Flow of a request for an event

The next graphic illustrates what happens when a user subscribes to the newsletter.



**Figure 4.6:** Flow of a request for subscribing to the newsletter

## 4.4 UI Mockups

A series of mockups were created to visualize the user interface of the web application.

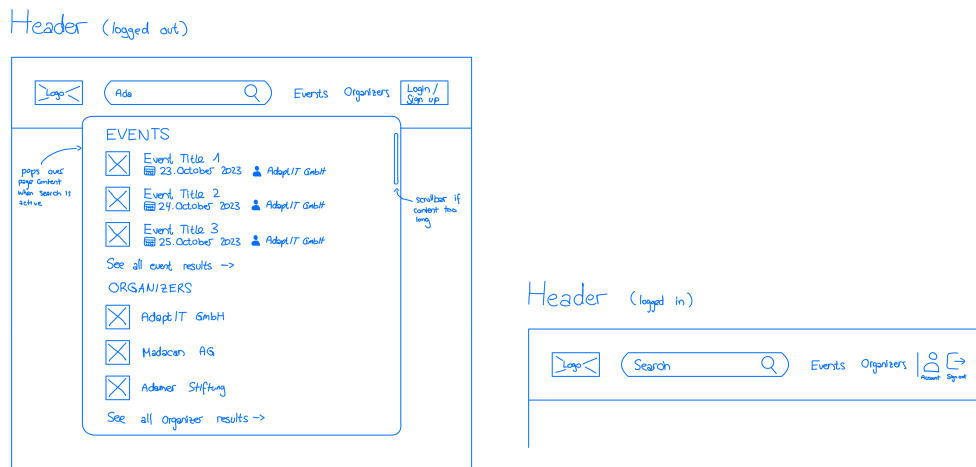


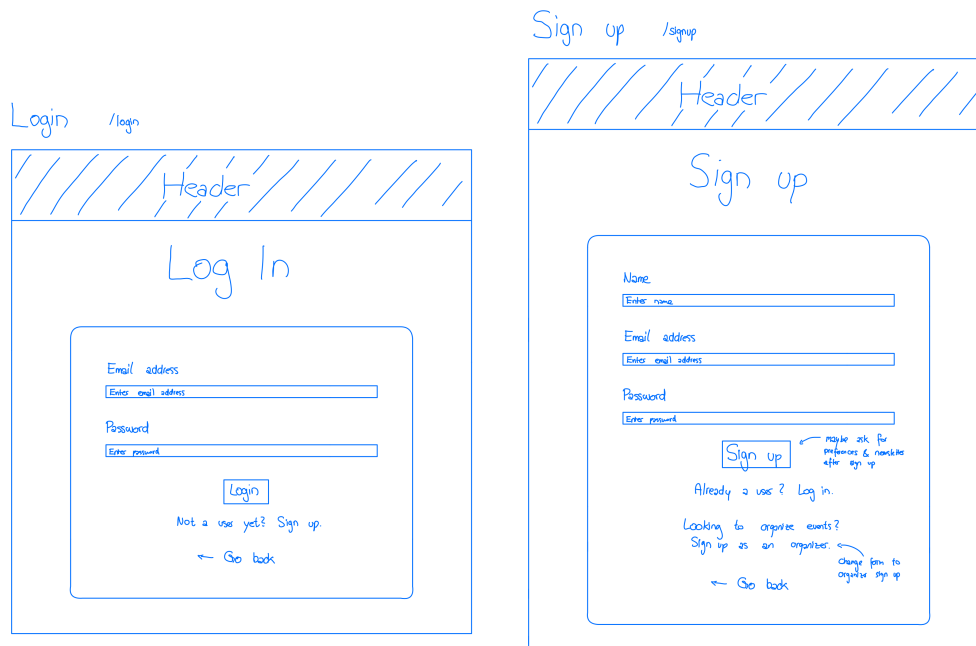
Figure 4.7: Header

In figure 4.7 the header of the web application is shown, with focus on the extended search bar. After a keyword for the search is entered, there will be a list of suggestions of events and organizers that match the keyword. Moreover, the header features buttons for both logging in and signing up, as well as for navigating to the events or organizers pages.



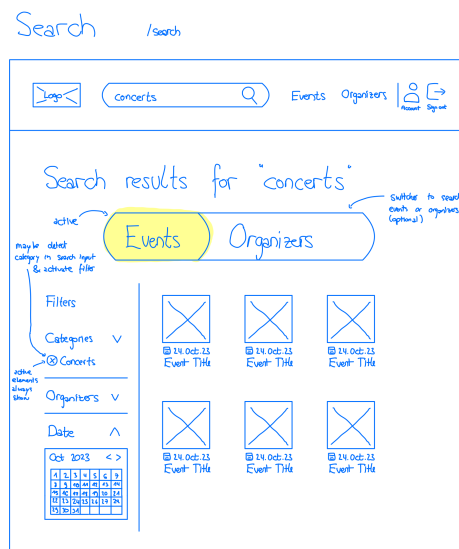
Figure 4.8: Home Page

In figure 4.8 the home page including the footer is shown. This page shows some different suggestions for events. In a first section there will be a list of featured popular events. The section below shows preferred events based on the user's interests. This will only be shown if a user is logged in and has preferences set. In the third section there will be a list of new events and new organizers. The last section shows recently used events if the user is logged in.



**Figure 4.9:** Login and Sign-up Pages

The figure 4.9 visualizes the login and sign-up pages. From the login page the user can navigate to the sign-up page and vice versa. On the sign-up page one can also sign up as a new organizer.



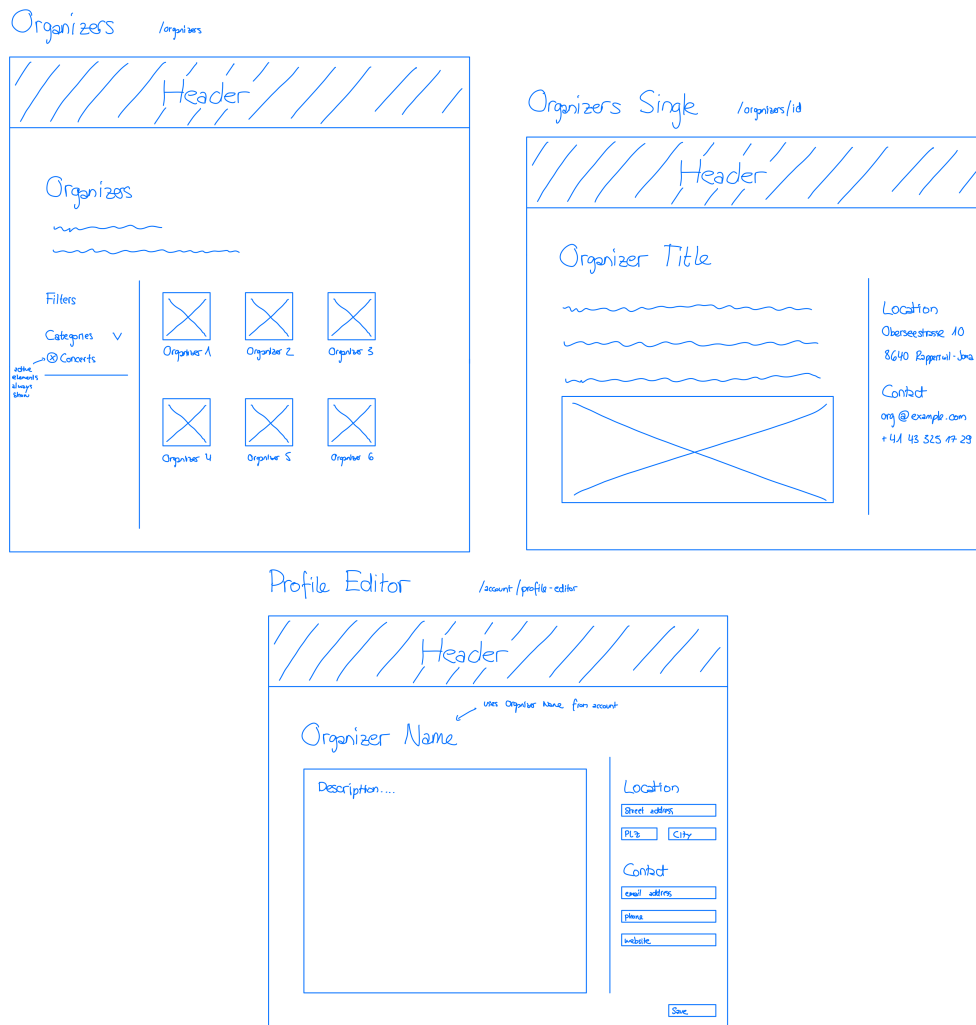
**Figure 4.10:** Search Page

This figure 4.10 shows the search page. The user is able to change between events and organizers, depending on what they are looking for. On the left of the search results there will be a filter bar to filter the results by different criteria. If a filter is active, it will always be shown regardless of the state of the filter bar. An active filter can be removed by clicking on the cross next to it.



**Figure 4.11:** Event Pages

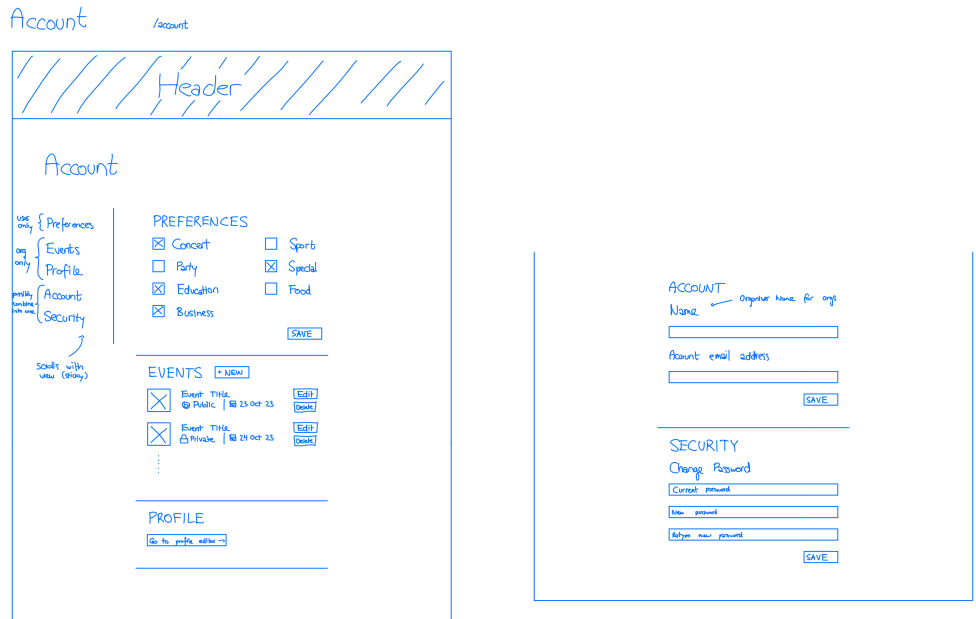
This figure 4.11 shows all four pages related to events. The first page is the overview page, which shows all upcoming events and provides filter options. The second page is the view of a single event with all its information. Depending on the visibility set by the organizer, the page might not be accessible for everyone. The third page is the page for creating a new event with all the necessary information. It is only accessible for organizers. With the last page the user can edit an existing event. This page is also only accessible for the organizer of the event.



**Figure 4.12:** Organizer Pages

The figure 4.12 shows all three pages related to organizers. Using the first page, one gets an overview of all organizers and can filter them. The second page shows the profile of an organizer. The last image shows the page for editing an organizer and its information displayed in the profile.

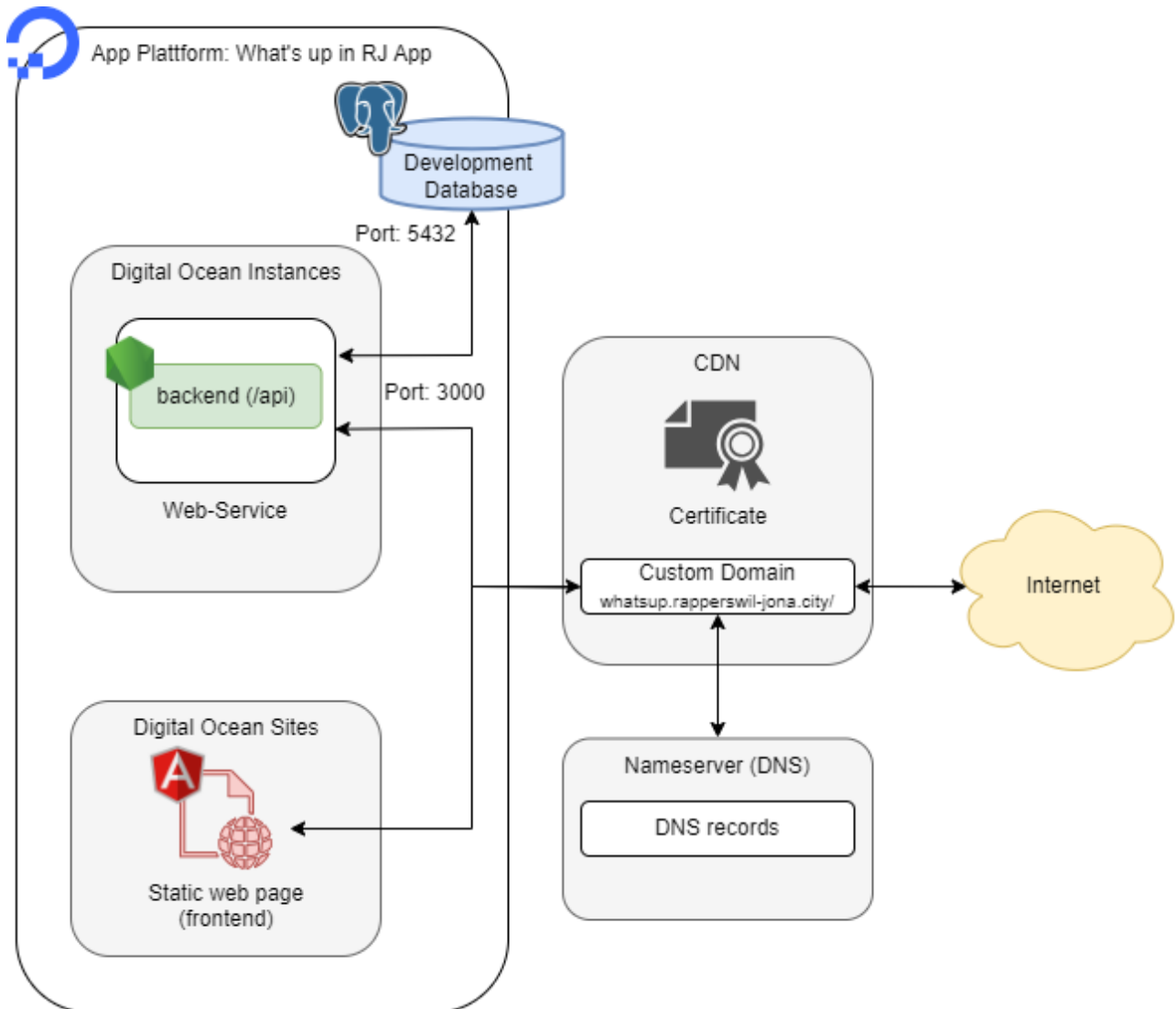




**Figure 4.13:** Account Page

The figure 4.13 shows the account page from a logged-in user or organizer. It is possible to change name, email and password. Only a user is also able to set preferences for events. Additionally, only an organizer can access the profile editor and can view (private), edit, delete or create their events.

## 4.5 Deployment Architecture



**Figure 4.14:** Deployment architecture of the project

The deployment architecture shown in figure 4.14 displays the architecture of the project from the perspective of DigitalOcean. For the deployment of the project, the App Platform feature by DigitalOcean is used. Therefore, the following three components are created in the context of an app on the App Platform:

- **Frontend:** The frontend is built as a static site. A static site gets pre-rendered and can then be easily distributed across content delivery networks (CDN), which is done by DigitalOcean. With this, a user can easily access the frontend without having to wait for the server to render the page first, decreasing latency.
- **Backend:** As the backend has to provide the API functionality, it is built as a web-service, which is a container running Node.js. Port '3000' and the path `/api`, routed to the backend

with the ‘HTTP Request Routes’ feature, are exposed to the internet. This allows the user to access the backend API with the `/api` path.

- **Database** The database is a Postgres development database provided by DigitalOcean. As it is a development database, it is unmanaged and not backed up. A development database is used because it delivers enough performance for the project and the costs can be kept low. For a production database, a managed database should be used.

The custom domain needs to be configured to point to the project. All nameserver and DNS records of this domain will be managed by the industry partner. Therefore, this will not be part of this documentation.

## 4.6 Technologies

Table 4.1 shows the chosen technologies for the project and the reasoning behind the selection. It is important that the chosen technologies are easily accessible and have a large community. This is to ensure that the project is maintainable in the future. Additionally, the project shall be JavaScript based.

Technology	Objective	Statement
Angular	Frontend	Angular is a popular frontend framework with a large community and a lot of documentation to go on. Another suitable option would be React, but Angular was chosen as we have used similar frameworks e.g. Nuxt in past projects. React has a steeper learning curve and is more difficult to get started with. Additionally, it comes with JSX, which is not as easy to use as HTML. On the other hand, Angular has many features out of the box that React does not have (e.g. routing, forms, etc.).
SCSS	Frontend	SCSS is a CSS preprocessor. It is a good choice for this project as it is easy to use and has a lot of features that make it easier to write CSS.
Tailwind	Frontend	Tailwind is a CSS library. We chose Tailwind over other design libraries because it is utility-first, therefore very flexible, easy to use, and you do not have to explicitly write CSS, making it easier to understand the code.

Continued on next page

Technology	Objective	Statement
Node.js / Express	Backend	Using Node.js for the backend is required for this project as the project needs to be JavaScript based. Additionally, Node.js is one of the most popular backend frameworks. As the frontend is Angular, it makes sense to use Node.js for the backend as they both depend on JavaScript. Therefore, no other backend framework was considered.
PostgreSQL	Database	The first decision was to use MySQL for the database as it is a popular choice for web applications. However, as the project is hosted on DigitalOcean and they provide Postgres-only development databases for low costs, the decision was made to use Postgres instead.
DigitalOcean	Hosting	DigitalOcean is a cloud hosting provider. This is a condition set by the industry partner for this project. Nevertheless, DigitalOcean is a suitable hosting provider as it is easy to use and has many useful features.
DigitalOcean App Platform	Deployment	The DigitalOcean App Platform is a service provided by DigitalOcean. It allows for easy deployment of applications. It is a great choice, as deploying the application is done fast and easy, and can additionally be automated.
SendGrid	Mail	SendGrid is a third-party service for sending and creating emails. Other options like Mailchimp or Mailgun were considered, but SendGrid was chosen because of the free plan and the easy-to-use API, which has a good documentation. Therefore, with its full range of features and a developer-friendly free plan, SendGrid has been chosen.

**Table 4.1:** Technologies used in the project

# Chapter 5

## Quality Measures

### 5.1 Quality Assurance

#### Definition of Done

**Code review:** All code changes are put in a merge request and must be reviewed by the other team member to ensure code quality.

**Unit testing:** All code changes must be accompanied by automated tests that pass without error.

**Manual testing:** All test cases defined in chapter 6 have passed successfully, and the software meets the acceptance criteria.

**Test documentation:** All test cases are reported in a test summary containing the test result.

**Documentation:** The documentation is always up-to-date. In the event of changes, the documentation is adjusted.

#### Continuous Integration Pipeline

The CI/CD pipeline on GitLab is used to optimize the development process and improve quality of the software. The test process is integrated in the pipeline to ensure that the software is tested consistently. This includes unit and linter tests and lets the pipeline fail if a unit tests fails. When the linter fails, the pipeline issues a warning, and merging remains possible. However, an exception is made for merges into the main branch, where both lint and unit tests must pass, to ensure the quality of the code in the main branch.

#### Code Metrics

Code quality metrics are measured with Code Quality, which is integrated into the CI/CD pipeline. The following metrics are measured:

- **Complexity:** The complexity of the code is measured with the Cognitive Complexity metric. The higher the complexity, the more difficult it is to understand the code. The metric is calculated based on the number of conditions and the nesting level of the code.
- **Bug Risk:** The bug risk is measured with maintainability and keywords that indicate a high bug risk.

## Quality Measurement Tools

Table 5.1 shows the used technologies to ensure code quality for the project.

Technology	Objective	Statement
Code Quality	Code analysis	Code Climate (Code Quality) is directly implemented on the GitLab CI/CD and analyses the source code's quality and complexity.
ESLint	Project code quality	The linter is used to analyse the code syntax of the front- and backend and is also integrated in the CI/CD pipeline to ensure code quality.

**Table 5.1:** Technologies for quality measurement

Table 5.2 shows the technologies that were planned to be used to ensure code quality for the project, but were not implemented due to time constraints.

Technology	Objective	Statement
Jasmine	Backend unit testing	Jasmine is used for backend unit tests due to its clean syntax and built-in test runner, ensuring the reliability and quality of the backend code.
Jasmine + Karma	Frontend unit testing	Jasmine and Karma handle frontend unit testing. Karma, acting as a test runner, simplifies testing frontend code in different browsers, guaranteeing code reliability and quality. It's also the default testing suite of Angular.

**Table 5.2:** Additional technologies for quality measurement

# Chapter 6

## Test Plan

### 6.1 Introduction

This test plan aims to verify the functionalities of this project based on the requirements provided in the document. Depending on project progress, the test plan will be updated accordingly. Only the latest version of the test plan will be included in this chapter. Previous iterations of the test plan will be archived in the appendix in chapter 12. The archive only contains sections that have been changed.

### 6.2 Test Objectives

The objectives of this test plan are as follows:

- To verify that the system and its relevant actors can add, modify and delete events.
- To verify that the web application is able to find events by search.
- To verify that the system can filter events by category.
- To verify that the system is able to let users register and login.
- To verify that the system is capable of sending newsletter emails to subscribed users.
- To verify that the system is capable of unsubscribing users from the newsletter.

### 6.3 Test Strategy

The testing strategy will be a combination of manual and automated testing techniques. Manual testing is used to verify the functionalities of the system, while automated testing is used to lint the code and successfully build the project. The automated tests will be executed on every push to a branch and on every merge request. Optional requirements are excluded from this test plan as they are not required for the project to be successful.

### 6.4 Test Environment

The following environment will be used for testing:

- Operating System: Apple macOS

- Browser: Firefox, Google Chrome, Safari
- Testing Tool: Postman

## 6.5 Test Specifications Functional Requirements

The following test cases, as shown in table 6.1, will be executed manually for the project:

No.	Description of test case	Precondition	Input	Expected Output
1	Organizer creates event	Organizer must be logged in and approved	Click on 'New Event' and enter details	Event is added and visible on the events overview page
2	Search for event	User is on the page and focuses search field	Use search field to type the name or category of an event one is looking for	Events that are related to this search will be displayed
3	Filter events	User has searched for an event or is on the event overview	Apply filters in the menu	Only related events will be visible
4	Users and organizers can sign up or login	User or organizer is on the login/sign up page	User/organizer enters login details	User/organizer is logged in and sees user/organizer related data
5	Subscribe to newsletter	User is logged in	User selects subscribe button	User will receive newsletters per email
6	Unsubscribe from newsletter	User is logged in and subscribed to newsletter	User selects unsubscribe button	User will not receive newsletters per email

**Table 6.1:** Test specifications for the functional requirements



## 6.6 Test Specifications Non-Functional Requirements

The test cases shown in table 6.2 will be executed manually for the project:

No.	Description of test case	Precondition	Input	Expected Output
1	Collaborative: Desired features are implemented	—	User tries to do all use cases from the MVP	All use cases can be accomplished
2	Performance: Backend is able to handle 1000 requests per minute	—	Backend requests	The whole system keeps running without errors
3	Response Time: The pages are loaded in under 200ms	The user has a reliable internet connection and is using an up-to-date web browser	Load pages	Each page is loaded in under 200ms
4	Responsiveness: Web-application is responsive on mobile, tablet, and desktop	—	Pages are loaded on the mentioned devices	The web application is responsive and visually appealing on all devices
5	Browser Compatibility: Web-application runs on Firefox, Chrome, and Safari	Browsers are installed	Web-application is loaded on the mentioned browsers	Pages are shown and fully functional on all browsers
6	Accessibility: Access over the customer-provided domain is available	—	Load page using the provided domain name	Pages are loaded
7	User Satisfaction: Users rate the UI	Test users are selected and can provide ratings	Test users navigate the page and provide ratings	3/4 users rate the UI minimum 8/10
8	Scalability: Database handles up to 10,000 events and 1,000 users	A database capable of handling those numbers of events and users is running	—	Database can handle those numbers without errors

Continued on next page

No.	Description of test case	Precondition	Input	Expected Output
9	Error Handling: Errors do not cause failures but display error messages	—	Errors are triggered	Error messages are displayed and logged
10	Security: Communication encrypted	—	Communication between frontend and backend is triggered	Communication is SSL encrypted
11	Security: Input validation	—	SQL injection is attempted	SQL injection is prevented
12	Data Privacy: Application is implemented with data protection regulations in mind	—	User data is stored	User data is stored in a way that complies with data protection regulations
13	Password Security: Passwords are stored securely	—	User registers and enters a password	Passwords are securely hashed and not stored in plain text in the database
14	User Data Isolation: Users can only access their own data	User is logged in	User tries to access data of another user	User is not able to access data of another user
15	Modularity: Business logic is modular and extensible	—	New features are added	New features can be added without changing existing code
16	Testing: API testing	The API testing tool has been configured	Tests have been executed	All tests successfully pass
17	Deployment: Implemented functionality is deployed	Developer is authenticated on DigitalOcean	Deployment is triggered	Implemented functionality is deployed

**Table 6.2:** Test specifications for the non-functional requirements

## 6.7 End User Tests

The end user tests will be conducted by 4 different test users. Each test user will be given a questionnaire. The same test cases will be utilized as outlined in the functional requirements section. These tests verify the use cases and ensure coverage across all layers of the application. Additionally, the end users have the opportunity to provide feedback on the user interface and the overall user experience. They are asked to rate the user interface on a scale from 1 to 10, with 1 being the lowest and 10 being

the highest possible rating.

## **6.8 Test Schedule**

### **6.8.1 CI/CD Tests**

The CI/CD tests are executed on every push to a branch. All tests have to pass before a merge request can be merged.

### **6.8.2 Integration Tests**

The mentioned integration tests have to be done once the corresponding features are implemented. Those functional tests should be done before 01.12.2023. All non-functional tests are done before 01.12.2023.

### **6.8.3 End User Tests**

The end user tests will take place before 15.12.2023.

# Chapter 7

## Implementation

This chapter describes the most important aspects and details regarding the implementation of the project.

### 7.1 Visual Design

The design of the project is based on the wireframes that were created in the design phase. The wireframes were used as a guideline for the design, but were not strictly followed. The current design uses blue and white as its main colours. The blue colour is used for almost all elements, while the white colour is mostly used as a background colour. Red is applied as an accent colour for error messages and certain buttons like delete and reset. The colour green is used for success messages, while grey is an additional colour for buttons. The design is kept simple and minimalistic, with the goal of making it easy to use and understand. It's also important to note that the design is aimed to be consistent across all pages, making it easier for the user to use different pages with the same design. One example of this are the overview pages and the search page, which all have the same design and layout.

### 7.2 Backend API Endpoints

All backend API endpoints have been described in the 'README.md' file of the 'backend' directory in the project itself. Due to the length of the document, it is not included in this chapter. However, a copy has been added to the appendix chapter 15. The document contains a list of all endpoints, including their URL, method, access, parameters, and return values. Additionally, it contains a list of all enums and models that are used in the project. The enums and models are described in detail, including all properties and their types.

#### 7.2.1 Special API Endpoints

Some endpoints seem to be duplicated, however, they serve different purposes. Good examples are the 'GET /events' and 'GET /me/events' endpoints. The first endpoint returns all public events, while the second endpoint returns all events of the currently logged-in user only, including private events. The same applies to the 'GET /events/:id' and 'GET /me/events/:id' endpoints. The first endpoint returns the event with the given ID, while the second endpoint returns the event with the given ID if the currently logged-in user is the organizer of the event.

## 7.2.2 Rate Limiting

The backend API is rate limited to prevent abuse. The rate limiting is done with the help of the `express-rate-limit` npm package and is applied to all routes on the backend component. Rate limiting is currently configured to 100 requests per 60 seconds per IP address. If a user exceeds the limit, they will receive a 429 error code with the message ‘Too many requests, please try again later’.

## 7.3 Database

The database has already been described and designed in previous chapters. Therefore, this section concentrates on the implementation of the database. To implement the domain model, the ‘sequelize’ npm package was used as an Object Relational Mapper (ORM). This makes it possible to easily define the models as objects. The generated models can then be used to interact with the database’s entities. Each model corresponds to a table in the database. Sequelize makes sure that all entities and their associations exist when the application boots up.

In a first step, Sequelize authenticates with the database and makes sure the credentials are correct by creating a test connection. Next, the entities and associations of the database get set up by defining their respective models. Lastly, a model synchronization gets executed, syncing states of the local JavaScript model with the database. The following code is simplified to showcase the different actions.

```
1 // Create the Sequelize object
2 const sequelize = new Sequelize(
3   process.env.POSTGRES_DATABASE,
4   process.env.POSTGRES_USER,
5   process.env.POSTGRES_PASSWORD,
6   {
7     host: process.env.POSTGRES_HOST || 'localhost',
8     port: port || 5432,
9     dialect: 'postgres',
10    dialectOptions: {
11      ssl: {
12        require: true,
13        rejectUnauthorized: false
14      }
15    },
16    logging: false,
17    pool: {
18      max: 10,
19      min: 0,
20      acquire: 30000,
21      idle: 10000
22    }
23  }
24 );
25
26 // Test the connection to the database
27 await sequelize.authenticate();
28
29 // Set up all models and associations
30 setupAssociations(sequelize);
31
```

```
32 // Model Synchronization
33 await sequelize.sync();
```

**Listing 7.1:** Sequelize Sync

The following example shows the ‘id’ and ‘name’ fields of the base user model. The ‘id’ field is an integer, set as a primary key, and will auto increment with each new entry. The ‘name’ field is of type ‘string’, cannot be ‘null’ or empty, and is defined to have between 3 and 255 characters.

```
1 export const initializeBaseUser = (sequelize: Sequelize) => {
2   return sequelize.define(name, {
3     id: {
4       type: DataTypes.INTEGER,
5       primaryKey: true,
6       autoIncrement: true
7     },
8     name: {
9       type: DataTypes.STRING(255),
10      allowNull: false,
11      validate: {
12        notEmpty: {
13          msg: 'Name cannot be empty'
14        },
15        len: {
16          args: [3, 255],
17          msg: 'Name must be between 3 and 255 characters'
18        }
19      }
20    },
21    // additional properties as defined in the domain model
22    // ...
23  }, {
24    // additional options for the model
25  });
26 }
```

**Listing 7.2:** Base User Model

Additionally, the associations need to be defined, so Sequelize knows how the tables are connected with each other. The following code snippet shows the associations of the base user model with the organizer and user models. The organizer and user model each have the foreign key of the base user on their ‘id’ field.

```
1 User.belongsTo(BaseUser, {
2   foreignKey: 'id',
3   as: 'base_user'
4 });
5
6 Organizer.belongsTo(BaseUser, {
7   foreignKey: 'id',
8   as: 'base_user'
9 });
```

**Listing 7.3:** Base User Associations

The outcome of the start-up procedure, whether it was successful or not, gets logged to the console. The backend is ready to interact with the database if no error messages are shown.

## 7.4 Features

This section describes the implementation of the major features of the project.

### 7.4.1 Event Handling

Each of the following features is a page of the application and is defined as a component.

#### Create Event

Several attributes are required to create an event. It is differed between public and private events. Public events are visible to all users, while private events are only visible to the organizer who created the event. For public events it is required to set a title, a description, a start date and a category. This ensures that an event has a minimum amount of information for a user to decide on whether they want to attend the event. On the other hand, a private event only requires setting a title. This makes an event more flexible for an organizer as they can add more information at a later date and do not have to fill out all the information at once. All events currently use a placeholder image, which is identical for all events.

The create event page consists of a simple form. All required fields are marked with an asterisk to let the user know, which fields need to be filled out. Per default the event visibility is set to private. If the user changes the visibility to public (or vice versa), the required fields and their corresponding asterisks get updated. After the organizer submits the create event form, several checks are done to ensure that the event is valid. Those checks are done through the service 'EventValidatorService'. This makes it possible to reuse the validation logic for other components, e.g. the edit event component. There are many checks and therefore only the most important ones are described in the following list:

- If an event is public, check if the event has the required fields title, description, start date and category.
- The end date of an event must be exactly the same as or after the start date, it cannot be before the start date.
- If the event is a one-day event, the end time must be after the start time.
- The zip code must be between 1000 and 9999.
- A location requires a street, a zip code and a city.

If one of the aforementioned checks fails, the field is marked as invalid and the user receives a message that describes the error near the field. This makes it easy for the organizer to find the error and correct it. The same behaviour is implemented in all other forms on the website.



**Figure 7.1:** Time Field Error

Additionally, when the organizer reloads or leaves the page without saving the event, a confirmation dialogue pops up to prevent the user from losing the unsaved data. This is done with the help of a

host listener that listens to the 'beforeunload' event of the browser. It shows the dialogue, if there are any unsaved changes in the form. The following code snippet shows the host listener.

```
1 @HostListener('window:beforeunload', ['$event'])
2 public unloadNotification($event: BeforeUnloadEvent): void {
3     if (this.showBeforeUnloadNotification() && this.hasUnsavedChanges()) {
4         $event.preventDefault();
5         $event.returnValue = true;
6     }
7 }
```

**Listing 7.4:** Host Listener

Since Angular acts as a single page application in this project, another check is also needed in order to prevent the user from losing any data when switching pages. The current host listener only covers the 'beforeunload' event, for example when the user navigates away from the page by entering a new address in the browser address bar, or decides to reload the page entirely through the browser function. However, normal navigation on the website itself does not emit this event. The additional check uses Angular's guards feature in coordination with 'canDeactivate' that defines if a route can be deactivated. When registering the event create route, the 'canDeactivate' property is passed with the 'UnsavedChangesGuard' guard. The 'UnsavedChangesGuard' checks if the message should be shown and if there are unsaved changes in the component that is about to be deactivated. There are some edge cases where you do not want the message to pop up, hence the additional check with the 'showBeforeUnloadNotification()' function.

```
1 export class UnsavedChangesGuard {
2
3     public canDeactivate(component: CanComponentDeactivate): boolean {
4         if (component.showBeforeUnloadNotification() &&
5             component.hasUnsavedChanges()) {
6             return window.confirm('You have unsaved changes! Do you really want
7                 to leave?');
8         }
9         return true;
10    }
11 }
```

**Listing 7.5:** UnsavedChangesGuard

This applies to all other routes using a form, except for the login and register forms as there is no benefit in using this feature there.

Once the organizer creates the event, they will get forwarded to the edit page of the freshly created event.

## Edit Event

The edit event page is similar to the create event page. It shares the same form and the same validation logic, with the sole difference that the form is pre-filled with the data of the current event. More difficult for an edit page is to check if there are any changes on the form. This is relevant if the organizer leaves the page while having pending changes. This is done by comparing the form data with the data of the event that is edited. If there are any changes, the confirmation dialogue pops up and asks the organizer if they want to leave the page without saving the changes. Additionally, the



organizer can delete the event, which leads them to the account page, displaying a success message. The page remains the same if an event is saved, and a success message is shown.

## Event Overview

The event overview page consists of a list of events that are directly loaded via the backend API. By default, only 50 events get fetched from the backend API to reduce the amount of data that is loaded and prevent the page from slowing down with huge data sets. At the end of the list exists a load more button, which loads the next 50 events from the backend API. This pagination system ensures better stability for the client.

## Event Single

The event single page can be found whenever an event gets displayed in an overview. Various information is visible on this page: All the event details as well as the organizer with their contact information. The data gets fetched from the backend API, using the URL path as an indicator, which event the user is viewing. Minor styling decisions improve the readability of the event details. For example, when an event has the same start and end date, only the start date is shown, to improve readability. The description field also allows Markdown formatting. If an organizer writes their description in Markdown, it will be rendered with HTML tags on the event single page, thus making the whole description design customizable. The Markdown is rendered directly in the backend before a response to an API request is returned. It is also only done on endpoints that require the rendered content, implying there is a different endpoint to receive the raw Markdown content, e.g. when editing an event you typically do not want the HTML output inside the form. The backend uses the custom renderer shown below to render the Markdown into HTML and set the heading levels correctly. As an underlying Markdown library handling all rendering, marked is used. Setting the headings is done because in Markdown the '#', or hashtag, is used as a heading. It starts at the 'h1' HTML tag with a single hashtag. However, the document's 'h1' tag is already set to the event title and would cause semantic issues. Therefore, the headings get shifted accordingly: 'h1' to 'h2', 'h2' to 'h3', etc. This also means that there are only five headings available instead of the original six.

```
1  const customRenderer = new marked.Renderer();
2
3  customRenderer.heading = function (text, level) {
4    if (level < 6) {
5      level += 1;
6    }
7
8    return `
```

**Listing 7.6:** Markdown Rendering

The rendered HTML is then set as the inner HTML of a 'div' HTML element with class 'markdown', which is styled with the aforementioned class. This is done in the frontend component.

```
1 <div class="mt-1">
2   <div class="markdown" [innerHTML]="event.description"></div>
3 </div>
```

**Listing 7.7:** Markdown HTML

As the description should have slightly smaller font sizes and stylings since it's a subcomponent of the page, it was necessary to style each HTML element that the Markdown parser supports. This gives an organizer the possibility to have almost complete control over their description style and make it more unique and appealing for the user.

## 7.4.2 Newsletter

The newsletter is a feature that is only available to registered users. It is not available to organizers as they are the ones publishing the events. The 'mail.ts' file in the 'helpers' directory contains all the logic for the newsletter. Sending out the newsletter is scheduled with the help of a cron job. It is scheduled to be sent out twice a month: On the 1st and 15th of each month at midnight server time. Certain criteria need to be met for a newsletter to be sent out, as shown in the following list.

- The user is subscribed to the newsletter.
- The user has at least one preference set.
- The selected preference has at least one event. This means that at least one event needs to have the same category as the preference.
- The event is public.
- The event has been created in between the last newsletter date and the current newsletter date (today), e.g. in the range of 1st to 14th of the month if today's date is the 15th.
- The event start date is at the earliest the day after the current newsletter date (today).

If all criteria are met, the event is added to the newsletter. This is repeated for all matching events. A maximum of 10 events are then added to the newsletter. The content is prepared, namely the date format of all date and time fields is changed to a more readable format. Finally, the full event object is then passed to the newsletter template with the SendGrid API and the email is sent out. This is done for each user that meets the criteria. Interacting with the SendGrid API is done through an API key.

The newsletter template is a predefined HTML template that is used for every newsletter. It uses SendGrid's dynamic template feature, which allows passing in your own data to the template, replacing the predefined placeholders. This also means that a dynamic template needs to be created in the SendGrid dashboard and the template ID needs to be set in the environment variables. The template can be designed in the SendGrid dashboard using a drag and drop editor or simple HTML code. For the sending to work, a sender name, sender email address and a reply-to email address have to be specified. The sender email address has to first be verified on SendGrid's dashboard as an approved sender address. This can be done through a simple verification email to the specified email. An alternative approach would be to verify ownership of the whole domain through a DNS record, however, this has not been done for this project as it doesn't use the whole domain and just a single address is needed.

### 7.4.3 Search

The search is split into two components, the search page and the search popover. The search bar, including the search popover, is accessible on every page, as long as the header is visible. The search algorithm is a simple string matching algorithm, looking for the search query in multiple fields of the event and organizer models. It even includes lookups in the event's organizer, including the base user, category and location. For organizers it includes lookups in the organizer's base user and location. Values that are not strings, e.g. dates, have a separate comparison. Dates have a predefined list of formats the user can enter in the search bar, which then automatically get converted into dates to compare to the actual values saved inside the database. For zip codes, it checks if the user entered a number. If valid, the zip code can be compared and found. It's a relatively expensive search approach compared to alternatives as multiple queries need to be done, one for each entity. Creating a more sophisticated search algorithm was not a priority, as the current algorithm is sufficient for the project's scope. It also opens up the possibility to implement a more sophisticated search algorithm like Elasticsearch in the future, as the current search algorithm is easily replaceable.

#### Search Page

The search page shows a list of events and organizers that match the search query. The search query is set as a parameter in the URL, similar to event IDs on single pages, which then returns the specified amount of events and organizers that match the query. The search page content itself is an Angular component. It shows two tabs 'Events' and 'Organizers', which are both filled with the corresponding search results. The separation had to be done because of the different filters that are available for events and organizers.

#### Search Popover

The search popover is a separate component that is used inside the header component. It is only visible if the user enters at least three characters in the search bar. The popover hides itself if the user clicks outside of the popover or if the user enters less than three characters in the search bar. It includes a short overview of the search, suggesting events and organizers that match the search query. The popover is fully interactive, meaning the user can click on the search results and gets redirected to the corresponding page. If the user presses 'Enter', the view switches to the search page, where additional functionality is available.

### 7.4.4 Filtering

The filter component is used on the event and organizer overview pages, and the search page. It is a separate component designed for reusability in other components. It accepts multiple parameters as shown in the list below.

- 'filters': The filters that are available for the component as an array. The following strings can be passed in an array to enable the corresponding filter: 'order', 'category\_id', and 'organizer\_id'.
- 'defaultOpen': Whether the filter's collapsible elements should be open by default.
- 'categories': An array of all available categories. This is used to display all available categories in the category filter.
- 'organizers': An array of all available organizers. This is used to display all available organizers in the organizer filter.

- ‘filterChanged’: An event emitter that emits an event whenever a filter has been changed. This is used to notify the parent component that the filters have changed.

By default, the filters are collapsed. The user can expand the filters by clicking on the filter title. If a filter has active items, they are shown even if the filter is collapsed. This makes it easier for the user to see which filters are active, and the user can easily remove them again by clicking on the ‘x’ icon. Filters are added to the URL as query parameters, thus making it possible to share the current filter state with other users or save the current filter state for later use.

The backend handles the filtering through query parameters. The following default filters can be used with all entities.

- ‘limit’: The amount of entities to return. Defaults to unlimited.
- ‘offset’: The offset of the entities to return. Defaults to 0.
- ‘order’: The order of the entities to return. The default order is different for each entity.

Depending on the entity, additional filters are available.

- ‘category\_id’: A list of category IDs to filter by. Only entities that match at least one of the category IDs are returned.
- ‘organizer\_id’: A list of organizer IDs to filter by. Only entities that match at least one of the organizer IDs are returned.

This makes the filtering highly customizable and flexible.

### 7.4.5 Home

The home page is the first page a user sees when visiting the website without any additional path. Its purpose is to give the user a quick overview of interesting events, and encouraging them to subscribe to the newsletter. Multiple views of different events are shown. The ongoing/upcoming events section, including six events, whose start date and time is closest to the current date/time. The new events section showcasing the six newest events created. They are queried with the ‘created\_at’ field, which holds the date the event has been created. Additional sections on the home page are only visible if different conditions are met. Following cases will show the described sections:

- The user is not logged in: The user sees a button to subscribe to the newsletter. This will lead them directly to the sign-up page, as they need to be registered and logged in to subscribe to the newsletter.
- The user is logged in but is not subscribed to the newsletter: The user sees the same button, which should convince them to subscribe to the newsletter. The button leads to the account page instead, where the user can easily subscribe to the newsletter.
- The user is logged in and subscribed but has no preferences set: The subscribe button now tries to get the user to set or change their interests. This button also leads to the account page, where the user can set their interests.

- The user is logged in, subscribed and has preferences set: The user additionally sees a section with a list of events that are recommended for them. It uses the user's preferences to determine, which event categories are suitable for the user. If there are no events matching the user's preferences, the 'preferred events' section is not shown.
- An organizer is logged in: As an organizer can not subscribe to the newsletter and has no preferences, they only see the default sections.

Additionally, to make the home page more appealing and interactive, the main title and the prompt to subscribe to the newsletter are changing on every reload of the page. This is done with a predefined list of sentences, from which a random entry is chosen each time.

#### 7.4.6 Organizers

Organizers are one type of user in the system. They are able to create, edit and delete their own events. Everyone is able to register as an organizer. This makes it possible for organizers to create events even before the admin has verified their account. The events cannot be set to public and therefore are not visible to other users until the organizer's account has been verified. The goal of this procedure is to make it easier for organizers to start using the platform while still ensuring that spam events are not published and fake organizers do not have any influence. Due to the non-existent admin portal, the verification process is currently disabled and every organizer is automatically verified. The organizer can only edit and delete their own events and not events from other organizers. This is done by checking if the current user is the organizer of the event that is about to be edited or deleted. If the user is not the organizer, the backend API returns an error message with the corresponding error code. The frontend then shows a message to the user, informing them that they are not allowed to edit or delete the event. This ensures user data isolation and prevents unauthorized access to other organizers' data.

#### 7.4.7 Account

Any registered user can access the account page. The account page is split into multiple sections, each section containing different information and functionality. It shows the user's information, including their name and email address. Additionally, all users can change their password in the security section of the account page. Users of type 'user' can subscribe to the newsletter and set their preferences. Organizers can access and edit their events. Events are split into two sections: 'Ongoing/Future' and 'Past'. The ongoing/future section shows all events that are currently ongoing, will start in the future, or have no date set (private events). The past section allows the user to view, edit, and delete old events. The split has been done intentionally to better organize events on the account page. By default, the ongoing/future tab is active. The last section is the 'Delete Account' section named 'Danger Zone', where all users can delete their account. Deleting an account is a permanent action and cannot be undone. The user is asked to confirm the deletion of their account before proceeding. If the user confirms the deletion, the account is deleted and the user gets logged out. All data associated with the user gets deleted as well, including events, preferences, etc.

#### 7.4.8 Message

A message has the properties 'type' and 'text'. The type can either be 'success' or 'error', styling the message accordingly. The text contains the message content that is to be shown to the user. Setting and showing the message is mostly done by the message component. It generates the needed HTML markup and handles the styling. A success message has a title 'Success' with a green background and

the message text. An error message has a title ‘Error’ with a red background and the error message text. The component is used on most pages, usually whenever an error message is warranted.

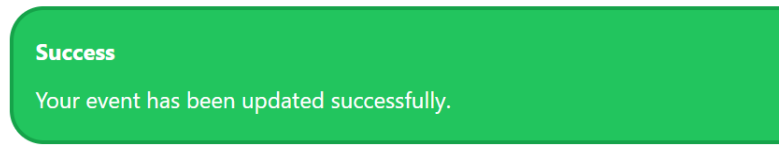


Figure 7.2: Success Message

## 7.5 Error Handling

### 7.5.1 Backend

When an error occurs on the backend side during a request, the system will not crash. Instead, the error will be caught and a response with the corresponding error code and an error message is sent. The following error codes can currently be sent out by the backend API.

- 400: Bad Request
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 409: Conflict
- 500: Internal Server Error
- 502: Bad Gateway
- 503: Service Unavailable
- 504: Gateway Timeout
- Default: Internal Server Error

These error codes are used throughout the whole backend. For example, in the following code snippet, if the logout fails, the backend would send a 500 error response with the message ‘Internal Server Error’.

```
1 static logout = (req: Request, res: Response, next: NextFunction): void => {
2   try {
3     req.session = null;
4     res.status(200).json({ success: true });
5   } catch (err) {
6     return next(new InternalServerError(undefined, err));
7   }
8 }
```

Listing 7.8: Logout Error Handling

The error responses are handled through a global error handler at a central location in the code. This makes it easy to change the error handling logic or error messages in the future. To create errors, new instances of the class ‘ApiError’ are created. Multiple helper classes exist, named accordingly to the error, to ease the process of generating error instances, which inherit from the ‘ApiError’ class. These error instances are then used to generate the error message and status code of the response. The error classes use basic information like status code, error message, the original error object, the transaction that was used at the time of failure, and whether the error should be logged. The original error object is used to log the original error, if log equals true. The supplied transaction will be rolled back automatically. This way, all transaction rollbacks happen at one place. The error ‘Expired JWT’ is not logged, as it happens too frequently and is not relevant for the operation of the project. It can be safely ignored. The following code snippet shows the constructor of the ‘ApiError’ class.

```
1 constructor(statusCode?: number, message?: string, originalError?: unknown,
2   transaction?: Transaction, log?: boolean) {
3   // ApiError extends the default Error JavaScript class, setting a default
4   // message with the help of the status code if none has been defined yet.
5   super(message || ApiError.getDefaultMessage(statusCode));
6   // Sets the status code, defaults to Internal Server Error
7   this.statusCode = statusCode || 500;
8   // Sets the original error for further use
9   this.originalError = originalError;
10  // Sets the transaction that was used while the error occurred
11  this.transaction = transaction;
12  // Sets the (default) log value
13  if (log === undefined) {
14    log = true;
15  }
16  this.log = log;
17  // Adds additional basic information to the error
18  this.name = this.constructor.name;
19  Error.captureStackTrace(this, this.constructor);
20 }
```

**Listing 7.9:** ApiError Class

The constructors of the subclasses are similar to the ‘ApiError’ constructor, minus the status code as each class itself already represents a status code. They solely pass the information to their parent class, in this case ‘ApiError’, directly defining the status code.

## 7.5.2 Frontend

The frontend catches errors every time it interacts with the backend API as these are the most likely spots where errors occur. A lot of errors can happen with network requests, reaching from a DNS issue to an unreachable component due to an outage or incorrectly configured network configuration. Catching these errors is done by simple ‘try catch’ blocks that wrap around the API call.

## 7.6 Security

### 7.6.1 Authentication

The current authentication system is based on JSON Web Tokens (JWT). The JWT is stored in the browser as an HTTP-only cookie. This makes it impossible for JavaScript to access the cookie. The

cookie is set to expire after one day. The JWT contains the user ID and the user type. The user ID is used to identify the user and the user type is used to determine, which features are available to the user. The JWT is signed with a secret key, which is stored in the environment variables. The secret key is also used to verify the JWT. The JWT is verified each time a protected route is accessed, e.g. the account page. In the backend, the token is received with the ‘express-session’ middleware and passed to other custom middlewares for verification and further use. If the JWT is invalid, the user is not authenticated and cannot access the protected route. Each request from the frontend to the backend API is intercepted by an HTTP interceptor. The interceptor checks if the request failed with a 401 error code, logs out the user, and redirects them to the login page if this is the case. This ensures that the user is not able to access any protected routes if they are not authenticated. Even if the user tries to access a protected route by entering the URL in the browser address bar, they will be redirected to the login page. Should the case arise that the user accesses a protected route with an expired JWT and not get redirected, the frontend will show an error message, and no data will be loaded. Once the user logs out, the cookie gets deleted and the user is no longer authenticated. If a new user registers, they are automatically logged in and redirected to the home page. The same applies to users who log in. The user is redirected to the page they tried to access before logging in.

When a user logs in, the first check is if the user exists. If the user does not exist, an error is immediately returned. This can lead to timing attacks, as the response time is different if the user exists or not. However, we decided to not implement a time attack prevention as it is not a critical part of the application and the attacker can not do much with this information. If the user exists, the password is compared with the hashed password. If the passwords match, the user is logged in, the last login date is updated, and the JWT is generated and sent to the user.

All changes to the current user and the user’s data are done through the ‘/me’ endpoints. It checks the JWT for validity and authorization. This allows the data to be isolated from other users, ensuring data can only be edited by its owner.

## 7.6.2 Password Hashing

The passwords are hashed with the help of the bcrypt npm package. The package uses the ‘bcrypt’ hashing algorithm to hash the passwords. The passwords are hashed with a salt of 15 rounds, greatly increasing the security of the passwords. The salt is automatically generated by the package and is stored together with the hashed password. Hashing and comparing passwords takes a lot of time, thus the process isn’t the fastest. However, this is not a problem as the hashing and comparing is only done when a user registers, logs in, or changes their password. It also makes it harder for an attacker to brute force the passwords.

## 7.6.3 Input Validation

Every input field is validated, sanitized and escaped. The frontend validation is done with the help of Angular’s built-in form validators and custom validators. Validating the input fields is done by checking if the input field is empty, if the input field is required, and if the input field has a minimum and maximum length. The validators are used in the corresponding HTML template of the component. On the backend side, the inputs are validated by checking each field individually. Sanitizing and escaping is only done for the description fields in an event and an organizer, as they are the only fields that also allow HTML to render, though this is only done after the select query has been executed, and the Markdown has been rendered, e.g. for the event single page. The sanitizing and escaping is done with the help of the DOMPurify npm package. All inputs are automatically escaped by the Sequelize



ORM, as it uses prepared statements to prevent SQL injection attacks. Angular also automatically treats all values as untrusted and sanitizes and escapes all inputs by default.

## 7.7 Code Documentation

The backend API is documented in a single ‘README.md’ file in the backend repository. The README contains a list of all enums, models, and endpoints as well as other important information, e.g. rate limiting. Each enum, model and endpoint is described thoroughly, including all needed information to use the API. Additionally, core parts of the backend, like the store subcomponents, have a more detailed description of the methods and their parameters directly in the code. Those comments contain parameters, return values and possible errors that can occur as shown in the following example.

```
1 /**
2  * Gets a base user by email
3  *
4  * @param email The email of the base user to get
5  * @param includePassword Whether to include the base user password field
6  * @param transaction The transaction to use
7  * @returns The base user with the given email or null if the base user was
8  *           not found
9  * @throws {Error} if an error occurred
10 */
```

**Listing 7.10:** BaseUserStore Comments

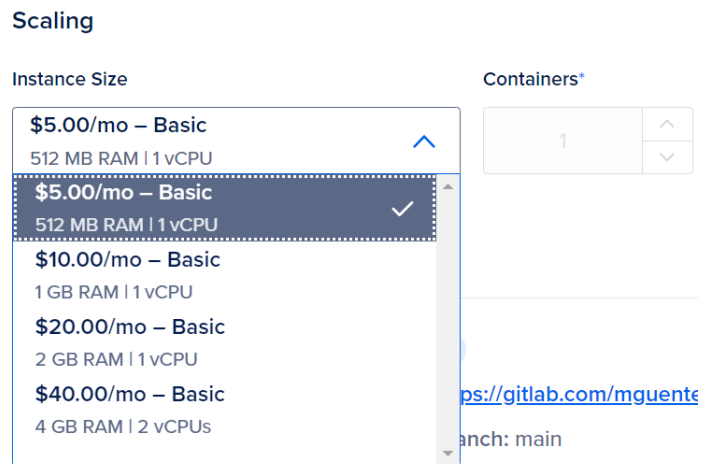
## 7.8 Deployment

The deployment is done with the DigitalOcean App Platform feature. First, it is necessary to connect DigitalOcean with the GitLab repository. An app can then be created where the paths of the frontend and the backend can be selected. The deployment is fully automated, meaning every push to the main branch triggers a new deployment. The backend is defined as web service and has a running instance with the cheapest plan. The specs of the machine are 1 vCPU and 512 MB of memory. The frontend component is hosted as a static website. It does not require any additional resources to run. The database is a Postgres development database, which has 512 MB of memory, a shared CPU, and 1 GB of disk space available. The ‘What’s up in RJ’ app is added as a trusted source to the database. Only trusted sources are able to connect to the database, hardening security. In addition, all environment variables are set directly in the DigitalOcean App Platform. For local development, ‘dotenv’ files are used that need to be defined every time the project is set up on a new machine. However, since environment variables holding sensitive information such as API keys, etc. should not be uploaded to the Git repository, the environment variables are defined through DigitalOcean’s environment variables feature. An additional benefit of defining the environment variables in DigitalOcean is that the variables can be selectively encrypted. As such, secrets, database credentials, and the SendGrid API key are encrypted and not visible in the user interface. The rest of the variables do not need to be handled confidentially and are therefore not encrypted. App wide variables like the application URL or the environment ‘production’ are set for both frontend and backend.

### 7.8.1 Scaling

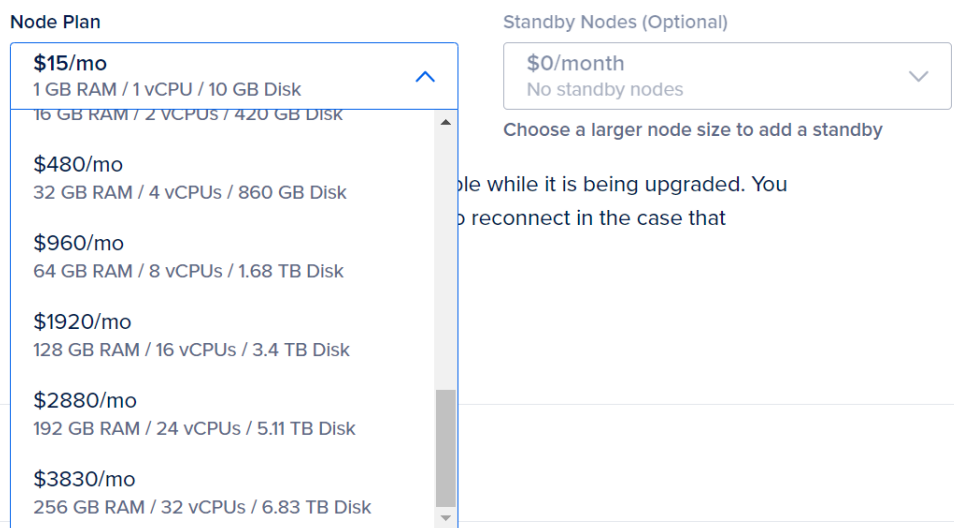
As scaling may be important for the application in the future, it is possible to further scale up the application. The backend can easily be scaled horizontally by having multiple instances users can

use. The following screenshot shows possible scaling options, however, even more powerful options are available.



**Figure 7.3:** Scaling Backend

On the other hand, the database can and needs to be scaled up for a production environment. The database has a lot of headroom to scale up. The following screenshot shows the most powerful options generally available.



**Figure 7.4:** Scaling Database

For a production environment we recommend to start with the smallest options: 1 GB RAM / 1 vCPU for the backend, and 1 GB RAM / 1 vCPU / 10 GB disk for the database. The logs can then be analysed over time, and if the performance is deemed to be insufficient, the performance can be increased together with the growing number of users accessing the application.

## 7.8.2 Frontend Environment Script

The frontend environment script ‘setenv.mjs’ is a simple Node.js script that is executed right before the Angular build process. The file ends with ‘.mjs’ because it is written in ECMAScript modules. It is used to set the environment variables for the frontend in the ‘environment.ts’ file. The script is executed in the ‘build’ script in the ‘package.json’ file, together with the Angular build process. It is needed because the ‘APP\_URL’ environment variable is not the same in a local development environment as it is in the production environment. The ‘APP\_URL’ environment variable is set by DigitalOcean and defines the application URL. It is not used locally as the frontend and the backend are simply hosted on different ports on the same machine. The variable is dynamic, meaning it changes depending on the configuration of the DigitalOcean App Platform. The script creates the ‘environment.ts’ file with the correct environment variables at build time. The following code snippet shows the script.

```
1 import { writeFileSync } from 'fs';
2
3 const targetPath = './src/environments/environment.ts';
4 const envConfigFile = `
5 export const environment = {
6   production: true,
7   apiUrl:
8     '${process.env.APP_URL}${process.env.APP_URL?.includes('localhost') ?
9     '' : '/api'}'
10 };
11 writeFileSync(targetPath, envConfigFile, { encoding: 'utf-8' });
```

Listing 7.11: Frontend Environment Script

It defines the target path and the contents of the ‘environment.ts’ file. Should the ‘APP\_URL’ environment variable not contain ‘localhost’, ‘/api’ is appended to the URL because the backend API is hosted on the same domain as the frontend but on a different path. The script is only executed in the production environment, as the ‘APP\_URL’ environment variable is normally not set in a local development environment. If the script is executed in a local development environment, the ‘APP\_URL’ environment variable is undefined and the application would define an invalid URL. However, if the ‘APP\_URL’ environment variable is set in a local development environment before the script is executed, the script would work as intended. The ‘build:prod’ script in the ‘package.json’ file does just that to make it easier to test the build for the production environment locally.

## 7.9 Testing

The testing is divided into automated and manual testing. They are both described in the following sections.

### 7.9.1 Automated Testing

For automated testing it was planned to use popular unit testing frameworks. Due to time constraints all parties involved agreed to not implement any automated tests. Therefore, the following section describes how the automated tests would have been implemented. The tests would have been run on every push to a Git branch with GitLab’s CI/CD feature, ensuring that the tests are always up-to-date.

## Backend Testing

The backend would have been tested with the help of Jasmine, a JavaScript testing framework. The frontend's test suite encouraged the use of Jasmine as it is the default testing framework for Angular. The tests would have been written in TypeScript, as the backend is written in TypeScript as well. They would have been split into multiple files, each file testing a specific component. API testing would have been done by mocking the API calls and testing the responses.

## Frontend Testing

The frontend would have been tested with the help of the built-in Angular testing framework 'Jasmine' and the test runner 'Karma'. The tests would have been written in TypeScript, as the frontend is written in TypeScript as well. They would have been split into multiple files, each file testing a specific component. The tests would have been run in a headless browser, e.g. Chrome, to ensure that the tests are not affected by the browser. Additional tests could have been done, including interaction testing, e.g. testing if a button click leads to the correct page, or if a form submission leads to the correct API call. However, this would have required a lot of additional time and was therefore not implemented.

### 7.9.2 Manual Testing

The manual testing was done with a checklist of the functional requirements 6.5 and the non-functional requirements 6.6. The functional requirements test all required features from an end user perspective. The non-functional requirements test the performance and the security of the application. The test reports with comments can be found in the appendix chapter 13. Additionally, end user testing has been conducted with the help of four different users who were not involved in the project. Therefore, a questionnaire was created to guide the users through the testing process. The idea of the questionnaire is to let the user test all the functional requirements without a detailed description on how to achieve the various goals. This ensures that the users test the application as they would use it in a real-life scenario. Finally, the users were asked to write feedback if there were any issues, and how they rate the user interface design of the application. The reports of the end user tests can be found in the appendix chapter 13.2.1.

# Chapter 8

## Results

This chapter describes the results of the project. It defines, which of the functional and non-functional requirements are fulfilled and which are not. Additionally, it describes if the minimum viable product is achieved.

### 8.1 Functional Requirements

The final product implements the following functional requirements defined as use cases in chapter 2.1.

- **UC1: Manage events**
- **UC2: Search and find events**
- **UC3: Filtering for events**
- **UC4: Login**
- **UC5: Subscribe to newsletter**

All the required use cases are achieved and working. The table 13.1 shows the testing of the functional requirements and their results.

Use cases **UC6 to UC17** are defined as optional requirements. None of them were implemented. The focus lied on the core functionality as well as the required use cases. Therefore, the time was not sufficient to implement any optional requirements.

### 8.2 Non-Functional Requirements

The final product fulfills the most non-functional requirements. The following list contains the names of the fulfilled non-functional requirements, which were defined in chapter 2.2.

- **NFR1 Collaborative**
- **NFR2 Performance**
- **NFR5 Browser Compatibility**
- **NFR6 Accessibility**
- **NFR7 User Satisfaction**

- **NFR8 Scalability**
- **NFR9 Error Handling**
- **NFR10 Security**
- **NFR11 Data Privacy**
- **NFR12 Password Security**
- **NFR13 User Data Isolation**
- **NFR14 Modularity**
- **NFR15 API Testing**
- **NFR16 Deployment**

The following non-functional requirement is partly fulfilled.

- **NFR3 Response Time**

The ‘Response Time’ NFR defines that pages have to load in less than 200ms. This is not possible for pages that handle passwords, e.g. the login, sign up, and account pages. As the requirement **NFR12 Password Security** is more important, this requirement can not be fulfilled with today’s hardware and is therefore set to ‘Partial Pass’.

One non-functional requirement has not been fulfilled.

- **NFR4 Responsiveness**

Due to time constraints the responsive design was not implemented. This was accepted by the industry partner as the focus of this project is set on the core functionality and the required requirements. All non-functional requirements are tested, and the results are shown in the table 13.2.

## 8.3 Minimum Viable Product

This project created a working web application with the following features:

- Everyone can access the application through a web browser (optimized for laptop and desktop computers using Chrome, Firefox or Safari).
- Everyone can view events and their details.
- Everyone can search and filter for events.
- Organizers and users can create an account, log in and change their account details.
- Organizers can create, edit and delete events.
- Users can define preferences by selecting categories they are interested in.
- Users can subscribe to a newsletter that contains events matching their preferences.
- Users can unsubscribe from the newsletter.

With these features, the application implements all points of the minimum viable product defined in 2.3.

# Chapter 9

## Conclusion

This chapter contains the conclusion of the project and sets the results in relation to the goals. It also contains the vision for the future of the project.

### 9.1 Result Reflection

With the result chapter 8, the achieved functional and non-functional requirements are described. The project was able to achieve all required functional requirements and most of the non-functional requirements. Two non-functional requirements were not completely achieved. The 'Response Time' NFR only partially passed, though with security in mind, making the application more secure thanks to longer loading times. The password uses multiple rounds to hash, making it more difficult for attackers to crack any password hash. The 'Responsiveness' NFR failed due to time constraints. This is one of the requirements, which needs a lot of effort to complete and also has a high impact, as the application would be usable on mobile devices and smaller screens in general. Throughout the project the decision was made to not implement the responsive design and instead focus on functionality. As this has been accepted by the industry partner, the project can be considered a success.

### 9.2 Goal Achievement

The goal was to create a responsive web application, which is able to showcase events of a city. From an end user perspective, the goal was to create a web page, where they can find events they like with just a few clicks. Furthermore, an end user should be able to receive a newsletter with their preferred events.

This is a basic summary of the main goal. Except the responsive design, which was described in previous sections, all goals were achieved. A web application is available, where users and organizers can log in. Organizers can create events and users can find events they like, through the website and the newsletter. The main goal of this project has been achieved.

### 9.3 Future Vision

The project still has a lot of potential to grow. First, some changes and improvements need to be done. The following list contains the most important items.

- Implement the responsive design of the website.

- An event cannot be updated once it has started due to the event start date and time being in the past. This should be changed to allow updates until the event has ended.
- The code quality report still contains some minor issues e.g. simplify if statements, which should be fixed.
- Implement lazy loading of Angular components, as any restricted component is not needed on the first load unless the user is logged in.
- The search term in the search input field should be deleted when the search context is left.
- Preferences should be deleted, and optionally created, in bulk to reduce requests to the database. Currently, each preference is created/deleted individually.
- Add a new property 'hasMore' to the response of any list (events, organizers, search) that holds a boolean and lets the requester know if more items can be fetched. This only applies to paginated lists and is used for the load more button on overview pages and the search page.
- Send a default newsletter even if there are no events in the users preferences. Currently, no newsletter gets sent if there are no new events available in the user's selected categories.
- Add JWT refresh tokens to the authentication process for better security.
- Use UUID or similar for all IDs of entities. This mostly prevents guessing of IDs and the ability to see how many items in total there are per entity.
- Add extensive testing to front- and backend and include it in the CI/CD pipeline.
- All optional requirements defined in chapter 2.1 can be implemented. Some preparations are already done, e.g. organizer attributes for the profile editor.

The following list shows optional improvements, which need further discussion before implementation.

- Newsletter should only include new events in the upcoming X weeks instead of all events in the future.
- Add 'RETURNING' to SQL queries, so they only return the needed rows, e.g. the 'create' function only returns the ID, no other information is needed.
- The title on the home page could be changed to a catching slogan or something similar instead of a rotating title.

Additionally, there are many features and improvements, which could also be implemented. The following list contains some ideas, which could be implemented in the future.

- Authentication through passkeys, making the application more future-proof.
- Better notifications for the user, e.g. toast notifications.
- A sitemap for search engine optimization.
- A simple Markdown preview for all description fields.
- Use Elasticsearch (or similar) for better search results.



- A dashboard for organizers with statistics and dedicated event management.
- Implement tags for events. Organizers can add existing tags to events or create new ones. Users can then search and filter by tags in addition to categories.
- A calendar and map view for events.
- The ability to bookmark events. Logged-in users will be able to check their bookmark list to easily see the events they're interested in.
- Event registration, including payment (e.g. through Stripe).
- Friends list for users, letting them see events friends attend, etc.
- Event reviews, discussions and ratings.

These ideas could be implemented in future iterations of the project and would greatly raise usability of the project.

Part III

**Project Documentation**

# Chapter 10

## Project Plan

### 10.1 Resources

#### Time

The project is expected to last for 14 weeks, with a planned effort of 240 hours per person. This means that each person has an average of 17 hours per week available to contribute to the project.

#### Cost

The only costs that will be incurred during the project are the cloud service costs. To keep the costs as low as possible, the cheapest machines and database will be used on DigitalOcean. The machine for the backend has a monthly cost of \$5, while the database costs \$7 per month as it is only a development database with low performance. The total cost for the cloud services is \$12 per month. Furthermore, the mail service for the newsletter will be provided by SendGrid, which is free for up to 100 emails per day. For the duration of the project, no costs will be incurred for the mail service. Additionally, the domain name will also charge a small fee, which is not known, as the industry partner will provide it. In conclusion, the industry partner has the costs for the domain name and the cloud services.

### 10.2 Processes and Meetings

#### Processes

Scrum+ is utilized to plan the project, which enables the combination between long-term planning with RUP and short-term planning with Scrum. The four project phases of RUP are used as a rough plan in conjunction with milestones, as shown below. The sprint duration is two weeks, with the start and end on Fridays. Each team member is assigned a specific role to distribute responsibilities.

#### Meetings

Most of the meetings are scheduled on Fridays, including the daily scrum, sprint planning and sprint review. In these meetings, the backlog is updated and tasks are assigned for the following sprint. Additionally, the progress is verified. All meetings between project advisor and industry partner are scheduled on Friday afternoons. Further meetings such as refinement meetings will take place as needed.

## 10.3 Schedule

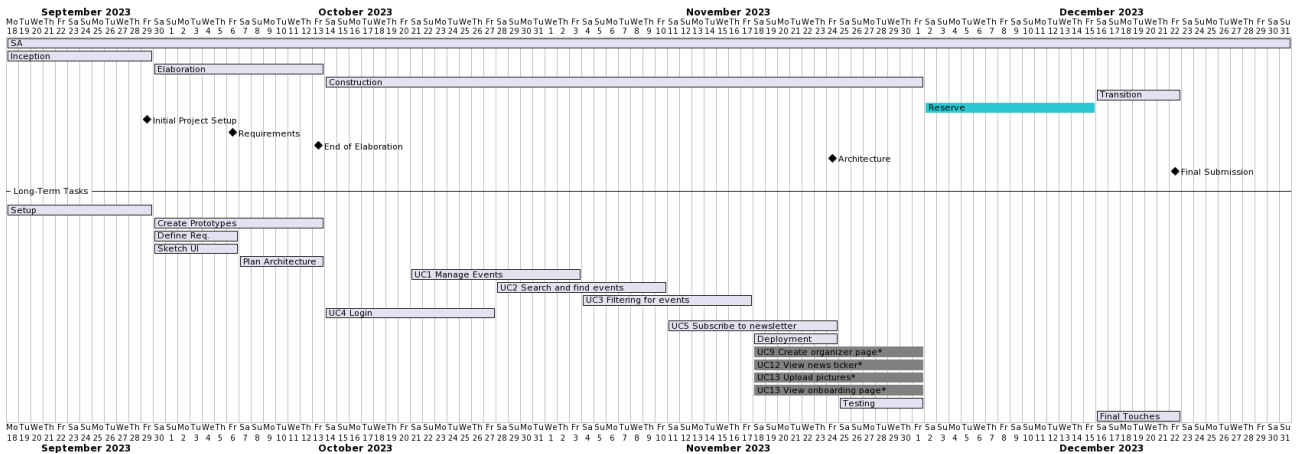


Figure 10.1: Project schedule

The optional tasks, marked dark grey, were omitted to prioritize refining the remaining parts of the application.

The reserve time, marked blue, is added due to the absence of one team member for two weeks because of military service.

### Phases

As Scrum+ is used for this project, the four-phase framework included in RUP will be adopted.

- **Inception:** The first phase of a project involves setting up the necessary environment, tools, and resources for the team to begin working. This includes the setup of Jira, the creation of both GitLab projects for the documentation and source code, and assigning roles to the team members. This phase is completed once the 'Initial Project Setup' milestone has been reached.
- **Elaboration:** In this phase, the project's requirements are identified and more documentation such as the domain model and use cases are created. First prototypes are created to ensure the compatibility of the technologies used and GUIs are sketched. The elaboration phase is completed with the reach of milestone 'End of Elaboration'.
- **Construction:** The Construction phase is the primary stage where the development work takes place. During this phase, the team writes the business logic and implements the UI. Comprehensive testing is also performed to ensure the web API and the frontend work together as expected. The goal of this phase is to produce a fully functional and deployable software system. This phase is completed if either all optional requirements are implemented or week 11 is reached.
- **Transition:** The final phase will be used to finish the documentation. This project and phase will be finished with the milestone 'Final Submission'.

### Milestones

The project progress will be mostly measured using the following milestones. The milestones should be reached before the meetings between the team members, the project advisor and the industry partner.

- **Initial Project Setup:** Required tools and documentation is set up.
- **Requirements:** Domain model, requirements and use cases are defined.
- **End of Elaboration:** Software architecture plan exists.
- **Architecture:** All required functionality is implemented. The MVP is working. Only optional requirements and integration testing remains to be done.
- **Final Submission:** The project is finished.

## 10.4 Organization and Roles

### Roles

To better divide the project workload, tasks are split into multiple roles between the project members. The roles are defined as shown in table 10.1.

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

**Table 10.1:** All roles with the assigned members

#### Project Owner

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

#### Scrum Master

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

#### Backend Developer

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

## **Frontend Developer**

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

## **Deployment Manager**

The Deployment Manager is responsible for deploying the software to the production environment. They ensure that the software is deployed without any issues and that it is available to users. They also monitor the performance of the software and ensure that it is scalable and reliable. Additionally, the Deployment Manager takes care of the CI/CD pipeline during development and release.

## **Architect**

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

## **10.5 Risk Management**

Table 10.2 shows the risks that were identified during the inception phase.

No.	Risk	Mitigation	Probability	Severity	Exposure
1	Cloud service charges high costs.	Switch to a lower-end machine.	Possible	Marginal	Medium
2	Cloud service provider is down.	A reputable cloud service provider with strong uptime should be used. Implement redundancy by deploying across multiple availability zones if possible.	Unlikely	Critical	Medium
3	Newsletter emails get sent to spam folder.	Use services like MailChimp that mark emails trustworthy.	Possible	Marginal	Medium
4	Third party service for newsletter service is down.	As the newsletter emails will be sent out after the provider outage, which possibly lasts only a few hours, this risk can be accepted.	Unlikely	Marginal	Medium

**Table 10.2:** Initially identified risks

Table 10.3 shows the current status of the risks.

No.	Risk	Mitigation	Probability	Severity	Exposure
1	Cloud service charges high costs.	The costs are fixed to \$12 per month for the deployed application. As this has been accepted by the industry partner, this risk is mitigated for this project.	Possible	Marginal	Mitigated
2	Cloud service provider is down.	A reputable cloud service provider with strong uptime should be used. Implement redundancy by deploying across multiple availability zones if possible. With that the risk can be accepted.	Unlikely	Minor	Low
3	Newsletter emails get sent to spam folder.	With the use of SendGrid, this risk is mitigated.	Possible	Marginal	Mitigated
4	Third party service for newsletter service is down.	As the newsletter emails will be sent out after the provider outage, which possibly lasts only a few hours, this risk can be accepted. Given SendGrid's high reliability with minimal downtimes, the probability is categorized as rare.	Rare	Marginal	Low

**Table 10.3:** Current status of the risks

This table can get extended, if more risks are identified during the project. The risk matrix was utilized to evaluate the identified risks.

## 10.6 Planning Tools

For issue and time tracking, Jira is used. This tool gives a good overview over all current issues and their status, and it also supports Scrum. The additional plugin 'Clockwork Free' is used to summarize the recorded times per issues. The meetings and documentation work are also tracked by separate issues.



## 10.7 Git

### Gitflow

The Gitflow branching model is used for this project. This means that there are at least two branches: `main` and `develop`.

The `main` branch is used for releases into production.

The `develop` branch is used for development and is the main/default branch for the project. It is created from the `main` branch.

Feature branches are created from the `develop` branch and are not prefixed. They are merged back into the `develop` branch once a feature has been completed.

Additionally, branches prefixed with `hotfix/` are for critical bugfixes that need to be deployed to production immediately (e.g. last minute fixes). The `hotfix` branches are created from the `main` branch and are merged back into both `main` and `develop`.

### GitLab Repositories

To share and separate the source code and documentation, two different GitLab repositories are used. The branches described in the Gitflow section are being used. A merge request is created for every feature branch, which will be merged into the `develop` branch. After the merge request has been approved by another team member, the feature branch will get merged by the assignee and is deleted afterwards.

The GitLab repository for the documentation is located at <https://gitlab.ost.ch/wuir/documentation/>.

The GitLab repository for the source code is located at <https://gitlab.com/mguenten/whats-up-in-rj/>.

# Chapter 11

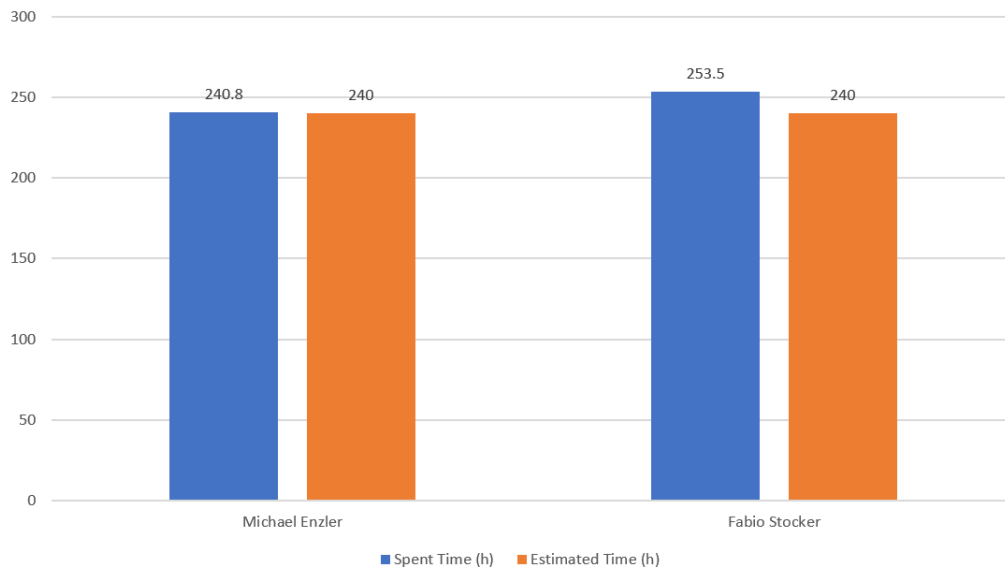
## Time Tracking Report

### Procedure

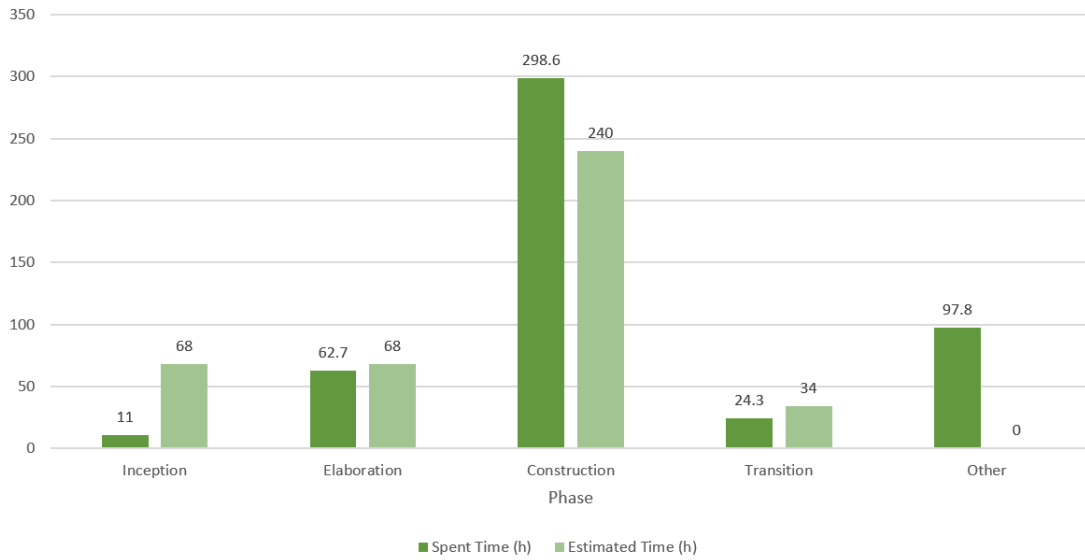
All issues are created on the Jira backlog. On sprint planning meetings, issues are assigned to the ongoing sprint. For an overview, the current issues of a sprint are allocated to a status: TO DO, IN PROGRESS, AWAITING REVIEW, DONE, TIME TRACKING. The last category is only used for tracking time of meetings, general work on the documentation, etc. The spent time is noted on each task and summarized with the 'Clockwork Free' plugin. Using this data, it is possible to estimate if the final project submission is still on schedule.

### Statistics

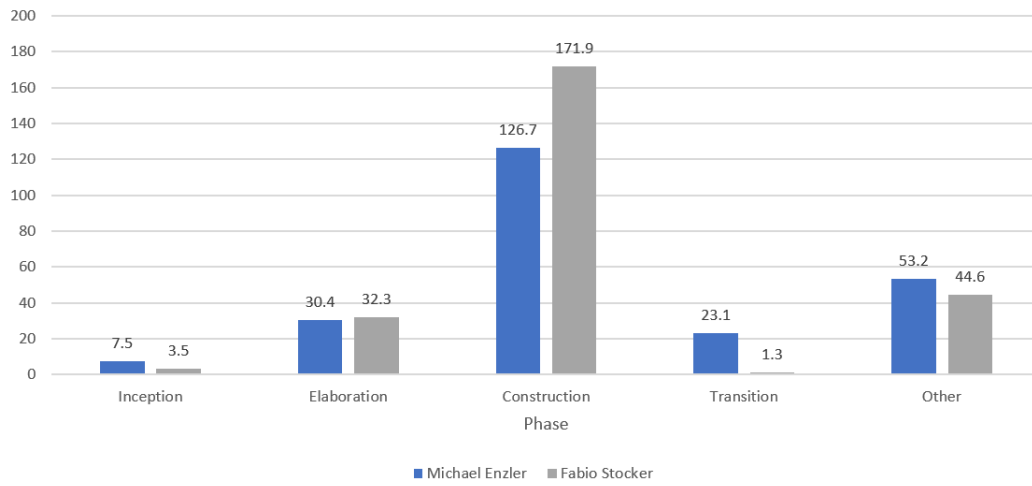
The following graphs illustrate the current status of the time spent on different sub-areas. All numbers in the graphs are in hours, and are rounded to the nearest single digit decimal number. The phase 'Other' includes tasks outside of a specific phase, e.g. meetings, overall documentation, etc.



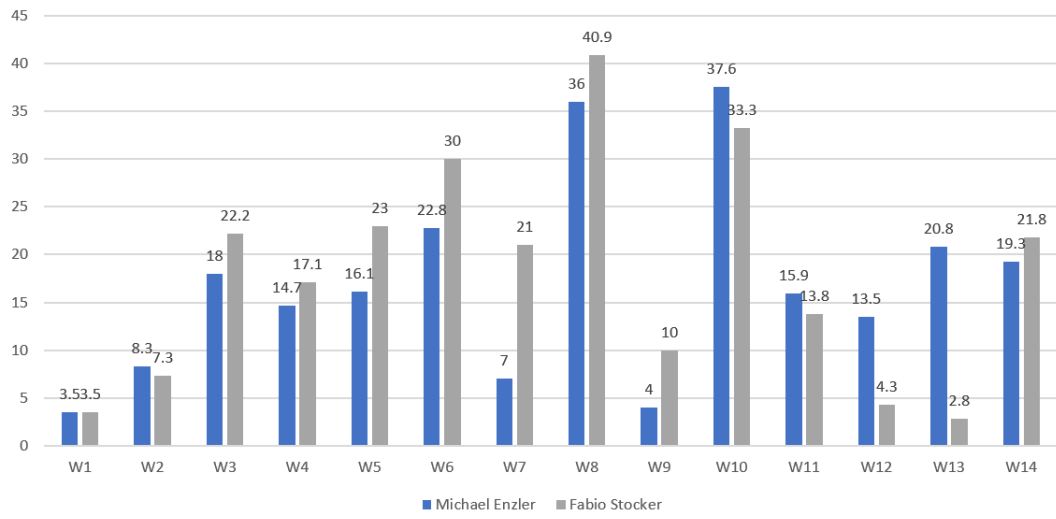
**Figure 11.1:** Time spent per person



**Figure 11.2:** Time spent per phase



**Figure 11.3:** Time spent per phase and person



**Figure 11.4:** Time spent per week

# Glossary

- API** Application Programming Interface: A set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.
- CI** Continuous Integration: The practice of merging all developers' working copies to a shared main-line several times a day.
- CD** Continuous Delivery: A software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.
- CI/CD** Continuous Integration/Continuous Delivery: A set of practices that combines the processes of continuous integration and continuous delivery.
- CLI** Command-Line Interface: A means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).
- CPU** Central Processing Unit: The electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.
- CRUD** Create, Read, Update, Delete: The four basic functions of persistent storage.
- CSS** Cascading Style Sheets: A style sheet language used for describing the presentation of a document written in a markup language like HTML.
- DOM** Document Object Model: A cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document.
- FR** Functional Requirement: A requirement that specifies a function that a system or system component must be able to perform.
- HTML** Hypertext Markup Language: The standard markup language for documents designed to be displayed in a web browser.
- HTTP** Hypertext Transfer Protocol: An application-layer protocol for transmitting hypermedia documents, such as HTML.
- HTTPS** Hypertext Transfer Protocol Secure: An extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network.
- JSON** JavaScript Object Notation: An open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types.

- JWT** JSON Web Token: An open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- MVP** Minimum Viable Product: A product with just enough features to satisfy early customers, and to provide feedback for future product development.
- NFR** Non-Functional Requirement: A requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.
- npm** Node Package Manager: A package manager for the JavaScript programming language.
- ORM** Object-Relational Mapping: A programming technique for converting data between incompatible type systems using object-oriented programming languages.
- RAM** Random-Access Memory: A form of computer memory that can be read and changed in any order, typically used to store working data and machine code.
- REST** Representational State Transfer: A software architectural style that defines a set of constraints to be used for creating web services.
- RJ** Rapperswil-Jona: A municipality in the Wahlkreis (constituency) of See-Gaster in the canton of St. Gallen in Switzerland.
- SCSS** Sassy CSS: A superset of CSS3's syntax that's uses the power of JavaScript to create more maintainable style sheets.
- SPA** Single-Page Application: A web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages.
- SSL** Secure Sockets Layer: A standard security technology for establishing an encrypted link between a server and a client.
- SQL** Structured Query Language: A domain-specific language used in programming and designed for managing data held in a relational database management system.
- UI** User Interface: The space where interactions between humans and machines occur.
- URL** Uniform Resource Locator: A reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.
- UUID** Universally Unique Identifier: A 128-bit number used to identify information in computer systems.
- UX** User Experience: A person's emotions and attitudes about using a particular product, system or service.
- vCPU** Virtual Central Processing Unit: A unit of measurement provided to a virtual machine (VM), commonly known as a virtual private server (VPS).

# List of Figures

1	Home page for visitors . . . . .	iii
2.1	Use Case Diagram . . . . .	7
2.2	Use Case Diagram . . . . .	8
3.1	Domain Model . . . . .	12
4.1	Architecture Overview . . . . .	14
4.2	Component Diagram . . . . .	15
4.3	Backend Structure . . . . .	16
4.4	Frontend Structure . . . . .	18
4.5	Request Flow Event . . . . .	20
4.6	Request Flow Mail . . . . .	20
4.7	Header . . . . .	21
4.8	Home Page . . . . .	22
4.9	Login and Sign-up Pages . . . . .	23
4.10	Search Page . . . . .	23
4.11	Event Pages . . . . .	24
4.12	Organizer Pages . . . . .	25
4.13	Account Page . . . . .	26
4.14	Deployment Architecture . . . . .	27
7.1	Time Field Error . . . . .	40
7.2	Success Message . . . . .	47
7.3	Scaling Backend . . . . .	51
7.4	Scaling Database . . . . .	51
10.1	Project Schedule . . . . .	61
11.1	Time spent per person . . . . .	67
11.2	Time spent per phase . . . . .	68
11.3	Time spent per phase and person . . . . .	68
11.4	Time spent per week . . . . .	69

# List of Tables

2.1	Use Cases . . . . .	7
4.1	Technologies . . . . .	29
5.1	Quality Measurement Technologies . . . . .	31
5.2	Additional Quality Measurement Technologies . . . . .	31
6.1	Test Specifications Functional Requirements . . . . .	33
6.2	Test Specifications Non-Functional Requirements . . . . .	35
10.1	Role Assignments . . . . .	62
10.2	Initial Risks . . . . .	64
10.3	Current Risks . . . . .	65



# Listings

7.1	Sequelize Sync . . . . .	38
7.2	Base User Model . . . . .	39
7.3	Base User Associations . . . . .	39
7.4	Host Listener . . . . .	41
7.5	UnsavedChangesGuard . . . . .	41
7.6	Markdown Rendering . . . . .	42
7.7	Markdown HTML . . . . .	43
7.8	Logout Error Handling . . . . .	47
7.9	ApiError Class . . . . .	48
7.10	BaseUserStore Comments . . . . .	50
7.11	Frontend Environment Script . . . . .	52

Part IV  
Appendix

# Chapter 12

## Test Plans

### 12.1 Test Plan #1

#### 12.1.1 Introduction

This test plan aims to verify the functionalities of this project based on the requirements provided in the document.

#### 12.1.2 Test Objectives

The objectives of this test plan are as follows:

- To verify that the system and its relevant actors can add, modify and delete events.
- To verify that the web application is able to find events by search.
- To verify that the system can filter events by category.
- To verify that the system is able to let users register and login.
- To verify that the system is capable of sending newsletter emails to subscribed users.
- To verify that the system is capable of unsubscribing users from the newsletter.
- To achieve 80% test coverage of the business logic.

#### 12.1.3 Test Strategy

The testing strategy will be a combination of manual and automated testing techniques. Manual testing is used to verify the functionalities of the system, while automated testing is used to verify the business logic of the system. The automated tests will be executed on every push to a branch. Goal of the testing strategy is to achieve 80% test coverage of the business logic.

Optional requirements are excluded from this test plan, since it is unknown whether they will be implemented.

#### **12.1.4 Test Environment**

The following environment will be used for testing:

- Operating System: Apple macOS
- Browser: Firefox, Google Chrome, Safari
- Unit Testing Tool: Jasmine

#### **12.1.5 Test Schedule**

##### **Unit Tests**

The unit tests must be created and executed, before a merge request with new or modified code is accepted.

# Chapter 13

## Test Reports

### 13.1 Functional Test Protocol 29.11.2023

No.	Description of test case	Precondition	Input	Expected Output	Actual Output	Pass/Fail
1	Organizer creates event	Organizer must be logged in and approved	Click on 'New Event' and enter details	Event is added and visible on the events overview page	Event is added and visible	Pass
2	Search for event	User is on the page and focuses search field	Use search field to type the name or category of an event one is looking for	Events that are related to this search will be displayed	Related events are displayed	Pass
3	Filter events	User has searched for an event or is on the event overview	Apply filters in the menu	Only related events will be visible	Only related events are displayed	Pass
4	Users and organizers can sign up or login	User or organizer is on the login/sign up page	User/organizer enters login details	User/organizer is logged in and sees user/organizer related data	Sign up is successful and user/organizer is logged in	Pass
5	Subscribe to newsletter	User is logged in	User selects subscribe button	User will receive newsletters per email	Newsletter has been received on the 01. December 2023	Pass

Continued on next page

No.	Description of test case	Precondition	Input	Expected Output	Actual Output	Pass/Fail
6	Unsubscribe from newsletter	User is logged in and subscribed to newsletter	User selects unsubscribe button	User will not receive newsletters per email	No newsletter has been received	Pass

Test report for the functional requirements

### Notes

The test case #5 and #6 were tested with two different users (email addresses) to ensure that the newsletter is only sent to the subscribed user. The two users were created at 29.11.2023 and one of them subscribed to the newsletter. As the newsletter is sent out at the first of every month, only one of the users should receive the newsletter, which was the case and therefore the test was successful.

## 13.2 Non-Functional Test Protocol 29.11.2023

No.	Description of test case	Precondition	Input	Expected Output	Actual Output	Pass/Fail
1	Collaborative: Desired features are implemented	—	User tries to do all use cases from the MVP	All use cases can be accomplished	All use cases can be accomplished via the user interface	Pass
2	Performance: Backend is able to handle 1000 requests per minute	—	Backend requests	The whole system keeps running without errors	The systems keeps running, no error occurs	Pass
3	Response Time: The pages are loaded in under 200ms	The user has a reliable internet connection and is using an up-to-date web browser	Load pages	Each page is loaded in under 200ms	All pages that are related to the authentication take longer than 200ms. All other pages load under 200ms	Partial Pass

Continued on next page

No.	Description of test case	Precondition	Input	Expected Output	Actual Output	Pass/Fail
4	Responsiveness: The web application is responsive on mobile, tablet, and desktop	—	Pages are loaded on the mentioned devices	The web application is responsive and visually appealing on all devices	The web application is not responsive.	Fail
5	Browser Compatibility: Web-application runs on Firefox, Chrome, and Safari	Browsers are installed	Web-application is loaded on the mentioned browsers	Pages are shown and fully functional on all browsers	Pages are shown correctly on all mentioned browsers	Pass
6	Accessibility: Access over the customer-provided domain is available	—	Load page using the provided domain name	Pages are loaded	Web-application is loaded via the provided domain	Pass
7	User Satisfaction: Users rate the UI	Test users are selected and can provide ratings	Test users navigate the page and provide ratings	3/4 users rate the UI minimum 8/10	All 4 users gave a minimum rating of 8.	Pass
8	Scalability: Database handles up to 10,000 events and 1,000 users	A database capable of handling those numbers of events and users is running	—	Database can handle those numbers without errors	No errors occurred with 10,000 public events and 1,000 users	Pass
9	Error Handling: Errors do not cause failures but display error messages	—	Errors are triggered	Error messages are displayed	Errors either from the front-end or the back-end are displayed and do not cause failures	Pass

Continued on next page

No.	Description of test case	Precondition	Input	Expected Output	Actual Output	Pass/Fail
10	Security: Communication encrypted	—	Communication between frontend and backend is triggered	Communication is SSL encrypted	Communication to frontend and backend is SSL encrypted	Pass
11	Security: Input validation	—	SQL injection is attempted	SQL injection is prevented	SQL injection is prevented	Pass
12	Data Privacy: Application is implemented with data protection regulations in mind	—	User data is stored	User data is stored in a way that complies with data protection regulations	User data is securely stored	Pass
13	Password Security: Passwords are stored securely	—	User registers and enters a password	Passwords are securely hashed and not stored in plain text in the database	Password is hashed	Pass
14	User Data Isolation: Users can only access their own data	User is logged in	User tries to access data of another user	User is not able to access data of another user	User can't access any other data than his own	Pass
15	Modularity: Business logic is modular and extensible	—	New features are added	New features can be added without changing existing code	Features are added without changing existing code.	Pass
16	Testing: API testing	The API testing tool has been configured	Tests have been executed	All tests successfully pass	Manual tests are successful.	Pass
17	Deployment: Implemented functionality is deployed	Developer is authenticated on DigitalOcean	Deployment is triggered	Implemented functionality is deployed	Deployment is triggered after a push into the main branch is done	Pass

Test specifications for the non-functional requirements



## Notes

Test case #3 was set to 'Partial Pass' even though the authentication pages take longer than 200ms to load. This is because when a password is entered, set or changed, the password is hashed, and the hash is stored in the database. The hashing process takes longer than 200ms as it is configured to do multiple rounds of hashing. As security is more important than performance when handling passwords, the test case is set to passed. Test case #4 was set to 'Fail' as the web application does not incorporate a responsive design. Due to time constraints, and the focus being set on functionality by the industry partner, this has not been implemented. The decision was discussed in meeting 10 (see ??) and approved to fail by the industry partner. Test case #7 was tested with four different users and the results of their testing and rating are documented in the section 13.2.1. For privacy reasons, the names of the test users have been redacted. Test case #8 was tested locally due to various constraints with DigitalOcean. Test case #16 was tested manually instead of automatically. This was discussed in the meeting 6 (see ??), where it was decided to concentrate on the functionality and not on automated tests. Therefore, only manual tests were executed.

### 13.2.1 Reports from Test Users

# What's up in RJ Testing / Questionnaire

Name: [REDACTED] \_\_\_\_\_

Date: 12.12.2023 \_\_\_\_\_

Use the following link to access the Web-Application: <https://whatsup.rapperswil-jona.city/>

Feel free to look around and check out everything. If you have any concerns, write them into the comment section at the end of the questionnaire.

Perform the following actions as described in the following points. If you encounter any problems, use the lines underneath to describe your problem. **Please use your own email addresses, as you may receive a newsletter.**

1. Create an **organizer** account via the sign up. You can also try to edit that information via the **account** page later on.

---

2. CRUD some events as you like via the **account** page. Also create at minimum one **public** event.

---

3. Create a **user** account via the sign up.

---

4. Look / search for your created **public** events on the different pages (Home, Search, Events).

---

5. Try out the **filters** on the different pages.

---

6. Subscribe to the **newsletter** and set your **preferences**.

---

With the next points you can leave us some feedback. Please use them and help us to improve the application.

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

8  
\_\_\_\_\_

8. Have you encountered any other problems or bugs throughout your exploration process?

---

9. Comments:

The save button on the account page is a bit misleading, as it is in password section.

It feels like you don't have to click it when doing other changes.

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

# What's up in RJ Testing / Questionnaire

Name: [REDACTED]

Date: 12.12.2023

Use the following link to access the Web-Application: <https://whatsup.rapperswil-jona.city/>

Feel free to look around and check out everything. If you have any concerns, write them into the comment section at the end of the questionnaire.

Perform the following actions as described in the following points. If you encounter any problems, use the lines underneath to describe your problem. **Please use your own email addresses, as you may receive a newsletter.**

1. Create an **organizer** account via the sign up. You can also try to edit that information via the **account** page later on.

---

2. CRUD some events as you like via the **account** page. Also create at minimum one **public** event.

Problems: Automatically filling out address in the event creation form, puts the street number in the street 2 textfield. I wasn't able to change the "thumbnail" of the event.  
Suggestion: A textfield/autofill/search function for the Category dropdown would be nice, since it contains a lot of elements. Maybe highlight your own events on the event overview page  
Maybe allow editing of event (if you are the owner) when navigating to the event through the event overview page.

---

3. Create a **user** account via the sign up.

---

4. Look / search for your created **public** events on the different pages (Home, Search, Events).

---

5. Try out the **filters** on the different pages.

Problems: Subscribing to the newsletter forced me to change my password since it's in the same form.  
Suggestion: Have a different submit button for the newsletter and account details. It's not clear that the save button also subscribes you to the newsletter.

---

6. Subscribe to the **newsletter** and set your **preferences**.

---

With the next points you can leave us some feedback. Please use them and help us to improve the application.

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

9

---

8. Have you encountered any other problems or bugs throughout your exploration process?

The label "Filter", looks the same as all the filter options (tried clicking it)

---

9. Comments:

The look and feel is very nice and intuitive. It is instantly clear what the page is for, and what the different functions are, so

usability in my opinion is very very good. The design is very clean, minimalistic and clear. I love it <3

---

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

# What's up in RJ Testing / Questionnaire

Name: [REDACTED]

Date: 01.12.2023

Use the following link to access the Web-Application: <https://whatsup.rapperswil-jona.city/>

Feel free to look around and check out everything. If you have any concerns, write them into the comment section at the end of the questionnaire.

Perform the following actions as described in the following points. If you encounter any problems, use the lines underneath to describe your problem. **Please use your own email addresses, as you may receive a newsletter.**

1. Create an **organizer** account via the sign up. You can also try to edit that information via the **account** page later on.

---

2. CRUD some events as you like via the **account** page. Also create at minimum one **public** event.

---

3. Create a **user** account via the sign up.

---

4. Look / search for your created **public** events on the different pages (Home, Search, Events).

---

5. Try out the **filters** on the different pages.

---

6. Subscribe to the **newsletter** and set your **preferences**.

---

With the next points you can leave us some feedback. Please use them and help us to improve the application.

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

9

---

8. Have you encountered any other problems or bugs throughout your exploration process?

I cannot contact support

---

9. Comments:

---

---

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

# What's up in RJ Testing / Questionnaire

Name: [REDACTED]

Date: 05.12.2023

Use the following link to access the Web-Application: <https://whatsup.rapperswil-jona.city/>

Feel free to look around and check out everything. If you have any concerns, write them into the comment section at the end of the questionnaire.

Perform the following actions as described in the following points. If you encounter any problems, use the lines underneath to describe your problem. **Please use your own email addresses, as you may receive a newsletter.**

1. Create an **organizer** account via the sign up. You can also try to edit that information via the **account** page later on.

---

2. CRUD some events as you like via the **account** page. Also create at minimum one **public** event.

---

3. Create a **user** account via the sign up.

---

4. Look / search for your created **public** events on the different pages (Home, Search, Events).

---

5. Try out the **filters** on the different pages.

---

6. Subscribe to the **newsletter** and set your **preferences**.

---

With the next points you can leave us some feedback. Please use them and help us to improve the application.

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

10

---

8. Have you encountered any other problems or bugs throughout your exploration process?

---

9. Comments:

Very easy to use, especially if you want to create an event!

---

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

## Notes

Test user #1 gave the suggestion to reposition the save button on the account page. This is understandable as it isn't obvious to the normal user that this button is not just for the password change. A new position for the button can be considered in further development of the application.

Test user #2 had some suggestions and issues. The described suggestions don't need further explanation here, they can be considered for future projects. The problem with the automatic filling of the location form is a difficult problem, as there are several tools that can be used to fill the form automatically, and each browser handles autocomplete attributes differently. To make it fit all tools is not possible to achieve in this project, but it should be considered in further development. The issue regarding the inability to change the thumbnail is known, as that feature will not be implemented during this project. For question 5, they had the same problem as already mentioned by test user #1. The save button should be repositioned. Additionally, they mentioned that the filter title looks too similar compared to the filter options. Therefore, a restyling of the filter title should be considered in the future.

Test user #3 wasn't able to contact the support. They may get this message by generating an error. In the case of an error, the user gets a message that advises them to try the action again or contact the support if the issue persists. As there isn't a support set up yet, the user is not able to contact them. The error message should be changed in further development, or a support contact should be set up.

Test user #4 has not encountered any issues.

# Chapter 14

## Application Screenshots

The screenshot shows the home page of the 'What's Up In Rapperswil?' website. The page has a blue header with the logo 'What's up in RJ' on the left, a search bar in the center, and navigation links for 'Events', 'Organizers', 'Login', and 'Sign Up' on the right. The main heading is 'What's Up In Rapperswil?'. Below this is a call to action: 'Don't Miss Out! Subscribe for Event News' with a blue 'Subscribe' button. The page is divided into two main sections: 'Ongoing/Upcoming Events' and 'Newly Added Events'. Each section contains six event cards, each with a placeholder image of hands raised in a field at night. The 'Ongoing/Upcoming Events' section lists: 'Street food festival' (04. December 2023 - 24. December 2023, Food & Drink), 'Dancing Night' (20. December 2023 22:00, Party), 'Big G Party' (21. December 2023 18:00 - 20:00, Spirituality), 'Halal + Vegan Food Festival' (23. December 2023 23:48 - 01. January 2024 11:49, Education), 'Umwelt Seminar' (29. February 2024 14:00, Education), and 'Bootsfahrt' (29. May 2024 14:00, Travel & Outdoor). The 'Newly Added Events' section lists: 'Umwelt Seminar' (29. February 2024 14:00, Education), 'Halal + Vegan Food Festival' (23. December 2023 23:48 - 01. January 2024 11:49, Education), 'Big G Party' (21. December 2023 18:00 - 20:00, Spirituality), 'Bootsfahrt' (29. May 2024 14:00, Travel & Outdoor), 'Street food festival' (04. December 2023 - 24. December 2023, Food & Drink), and 'Dancing Night' (20. December 2023 22:00, Party). The footer is blue and contains links for 'Pages' (Events, Organizers) and 'Legal' (Terms of Service, Privacy Policy, Cookie Policy, Imprint). A copyright notice '© 2023 - All rights reserved.' is at the bottom.

Home page for visitors

# Find Your Next Event

Edit Your Subscription Interests

My Interests

### Preferred Events



**Street food festival**  
04. December 2023 - 24. December 2023  
Food & Drink

### Ongoing/Upcoming Events



Home page for users

# Ready To Go Out?

### Ongoing/Upcoming Events



<b>Street food festival</b> 04. December 2023 - 24. December 2023 Food & Drink	<b>Dancing Night</b> 20. December 2023 22:00 Party	<b>Big G Party</b> 21. December 2023 18:00 - 20:00 Spirituality	<b>Halal + Vegan Food Festival</b> 23. December 2023 23:48 - 01. January 2024 11:49 Education	<b>Umwelt Seminar</b> 29. February 2024 14:00 Education	<b>Bootsfahrt</b> 29. May 2024 14:00 Travel & Outdoor
--	--	---	---	---	---

### Newly Added Events



Home page for organizers



## Login

**Email Address**

**Password**

[Login](#)

Not a user yet? [Sign up](#)

Login page

### Sign Up

**Name**

**Email Address**

**Password**

**Confirm Password**

[Sign Up](#)

Already a user? [Log in](#)

Looking to organize events? [Sign up as an organizer](#)

### Sign Up

**Organization Name**

**Account Email Address**

**Password**

**Confirm Password**

[Sign Up](#)

Already a user? [Log in](#)

Looking to attend events? [Sign up as a user](#)

Signup page for users and organizers

## Good afternoon, Michael

Edit your account details below.

- Preferences
General
Security

### Preferences

Change your newsletter and event preferences.

#### Newsletter

Do you want to receive our newsletter?

Yes, I would like to receive the newsletter.

#### Event Preferences

Choose the categories you are interested in.

- Art, Business, Charity, Community, Concerts, Culture, Education, Entertainment, Family, Fashion, Food & Drink, Health, Hobbies, Holiday, Home & Lifestyle, Music, Networking, Other, Party, Pets, Politics, Religion, Science & Tech, Seasonal, Spirituality, Sports & Fitness, Travel & Outdoor

- Preferences
General
Security

### General

Change your account details.

#### Name

Michael

#### Email Address

michael.enzler@ost.ch

### Security

Change your password.

#### Current Password

Current Password

#### New Password

New Password

#### Confirm New Password

Confirm New Password

Save

### Danger Zone

If you delete your account, there is no going back.

Delete Account

User account page

### Good afternoon, Organisatoren AG





Edit your account details below.

- Events
- Profile
- General
- Security

#### Events [+ New](#)

View and manage your events.

Ongoing/Future Past

-  **Street food festival**  
04. December 2023  
Public [View](#) [Edit](#) [Delete](#)
-  **Umwelt Seminar**  
29. February 2024  
Public [View](#) [Edit](#) [Delete](#)
-  **Bootsfahrt**  
29. May 2024  
Public [View](#) [Edit](#) [Delete](#)
-  **This is a test event**  
Private [Edit](#) [Delete](#)

### Good afternoon, Organisatoren AG


Edit your account details below.

- Events
- Profile
- General
- Security

#### Events [+ New](#)

View and manage your events.

Ongoing/Future Past

-  **Dancing Night**  
20. December 2023  
Public [View](#) [Edit](#) [Delete](#)

#### Profile

Edit your profile.

[Go To Profile Editor](#)

- Events
- Profile
- General
- Security

#### General

Change your account details.

**Organization Name**

**Account Email Address**

#### Security

Change your password.

**Current Password**

**New Password**

**Confirm New Password**

[Save](#)

#### Danger Zone

If you delete your account, there is no going back.

[Delete Account](#)

## Events

**Filters**

Sort by

Category

Organizer



**Street food festival**  
04. December 2023 - 24. December 2023  
Food & Drink



**Dancing Night**  
20. December 2023 22:00  
Party



**Big G Party**  
21. December 2023 18:00 - 20:00  
Spirituality



**Halal + Vegan Food Festival**  
23. December 2023 23:48 - 01. January 2024  
11:49  
Education



**Umwelt Seminar**  
29. February 2024 14:00  
Education



**Bootsfahrt**  
29. May 2024 14:00  
Travel & Outdoor



**Halal + Vegan Food Festival**  
23. December 2023 23:48 - 01. January 2024  
11:49  
Education



**Umwelt Seminar**  
29. February 2024 14:00  
Education



**Bootsfahrt**  
29. May 2024 14:00  
Travel & Outdoor

[Load More](#)

## Events

**Filters** [Reset](#)

Sort by

x Created (ascending)

Category

x Travel & Outdoor

Organizer

- Bestest Organization
- Organisatoren AG
- OST
- Ticketmaster AG
- Timon Enterprize



**Bootsfahrt**  
29. May 2024 14:00  
Travel & Outdoor

[Load More](#)

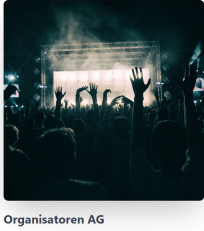
Organizers

Filters Category

Grid of 5 organizer cards: Bestest Organization, Organisatoren AG, OST, Ticketmaster AG, Timon Enterprize

Organizers

Filters Reset Category x Travel & Outdoor



Organisatoren AG

Load More

Organizers Overview page

## New Event

**Title \***

**Description**

# About My Super Cool Event

\*\*I also support markdown\*\*

## Subtitle

- Try out some lists
- They are cool
- How about nested lists?

1. And some numbered lists
2. They are cool too

**Visibility**

**Start Date**

**End Date**

**Start Time**

**End Time**

**Street**

**Street 2**

**Zip**  **City**

**Category**

## Edit Event

**Title \***

**Description \***

Hello and welcome to the street food festival in rapperswil!

We have food from a lot of different countries like:

- india
- japan
- thailand
- turkey
- and a lot more

## come and enjoy food and drinks!!

**Visibility**

**Start Date \***

**End Date**

**Start Time**

**End Time**

**Street**

**Street 2**

**Zip**  **City**

**Category \***



04. December 2023 - 24. December 2023

Hello and welcome to the street food festival in rapperswil! We have food from a lot of different countries like:

- india
- japan
- thailand
- turkey
- and a lot more

come and enjoy food and drinks!!

**Organizer**  
 Organisatoren AG

**Category**  
 Food & Drink

Search Results

Events



**Dancing Night**  
20. December 2023 22:00  
Party



**Bootsfahrt**  
29. May 2024 14:00  
Travel & Outdoor



**Big G Party**  
21. December 2023 18:00 - 20:00  
Spirituality

December 2023 Party 20:00 Spirituality 23. December 2023 23:48 - 01. January 2024 11:49 Education Travel & Outdoor  
Food & Drink

Newly Added Events

Search Results

Events



**Dancing Night**  
20. December 2023 22:00  
Party



**Street food festival**  
04. December 2023 - 24. December 2023  
Food & Drink



**Bootsfahrt**  
29. May 2024 14:00  
Travel & Outdoor

Organizers



**Organisatoren AG**



**Bestest Organization**

Dancing Night 20. December 2023 22:00 Party

Street food festival 04. December 2023 - 24. December 2023 Food & Drink

Bootsfahrt 29. May 2024 14:00 Travel & Outdoor

Search popover



### Search Results for "part"

- Events
- Organizers

- Filters
- Sort by
- Category
- Organizer



**Dancing Night**  
20. December 2023 22:00  
Party

**Bootsfahrt**  
29. May 2024 14:00  
Travel & Outdoor

**Big G Party**  
21. December 2023 18:00 - 20:00  
Spirituality

Load More

### Search Results for "org"

- Events
- Organizers

- Filters
- Category



**Organisatoren AG**

**Bestest Organization**

Load More

Search page



## New Events just dropped in Rapperswil!

### Christmas Dance

by [Let's Dance](#)

20 December 2023 18:00 - 20:00

Music

### Eminem Concert

by [Ticketmaster AG](#)

22 May 2024 20:00 - 23:00

Concerts

---

If you'd like to unsubscribe from this newsletter or edit your newsletter preferences, you can do so by [clicking here](#).

Newsletter

## Chapter 15

# Backend API Endpoints

# What's up in RJ - Backend

---

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

## Rate Limiting

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

## Enums

### UserType

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

- **user**: The user is a normal user.
- **organizer**: The user is an organizer.
- **admin**: The user is an administrator.

## Objects

### Error

The error object is used to represent an error in the API. It has the following structure:

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

### Success

The success object is used to represent a successful operation in the API. It has the following structure:

```
{
  "success": true
}
```

### BaseUser

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

- **id**: (number) The ID of the user.

- **name**: (string) The name of the user.
- **email**: (string) The email address of the user.
- **password**: (string) The password of the user in hashed form.
- **type**: ([UserType](#)) The type of the user.

## User

The user object is used to represent a user in the database. It extends the [BaseUser](#) object and has the following additional properties:

- **id**: (number) The ID of the user.
- **newsletter**: (boolean) Whether the user is subscribed to the newsletter or not.
- **base\_user**: ([BaseUser](#)) The base user object.

## Organizer

The organizer object is used to represent an organizer in the database. It extends the [BaseUser](#) object and has the following additional properties:

- **id**: (number) The ID of the organizer.
- **description**: (string | null) The description of the organizer.
- **contact\_email**: (string | null) The contact email address of the organizer.
- **phone**: (string | null) The phone number of the organizer.
- **website**: (string | null) The website of the organizer.
- **verified**: (boolean) Whether the organizer is verified or not.
- **base\_user**: ([BaseUser](#)) The base user object.
- **location**: ([Location](#)) The location of the organizer.

## Location

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

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

## Category

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

- **id**: (number) The ID of the category.
- **name**: (string) The name of the category.

## Event

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

- **id**: (number) The ID of the event.
- **title**: (string) The title of the event.

- **description**: (string | null) The description of the event.
- **start\_date**: (string | null) The start date of the event.
- **end\_date**: (string | null) The end date of the event.
- **start\_time**: (string | null) The start time of the event.
- **end\_time**: (string | null) The end time of the event.
- **public**: (boolean) Whether the event is public or not.
- **organizer**: ([Organizer](#)) The organizer of the event.
- **category**: ([Category](#)) The category of the event.
- **location**: ([Location](#)) The location of the event.

## Preference

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

- **user**: ([User](#)) The user of the preference.
- **category**: ([Category](#)) The category of the preference.

## Endpoints

/auth

Handles authentication.

### POST /auth/register

Registers a new user.

#### Access

- Everyone

#### Parameters

##### Body Parameters

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

#### Response

- **201 Created**
  - ([User](#) | [Organizer](#)) The user object if the user was created successfully.
- **400 Bad Request**
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an [Error](#) object.
- **409 Conflict**

- If the email address is already in use.
- Returns an **Error** object.
- **500 Internal Server Error**
  - If the user could not be created.
  - Returns an **Error** object.

## POST /auth/login

Logs in a user.

### Access

- Everyone

### Parameters

#### Body Parameters

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

### Response

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

## POST /auth/logout

Logs out a user.

### Access

- Everyone

### Parameters

This endpoint does not accept any parameters.

### Response

- **200 OK**
  - (**Success**) The success object if the user was logged out successfully.
- **500 Internal Server Error**
  - If the user could not be logged out.
  - Returns an **Error** object.

## /me

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

### GET /me

Gets the current user.

#### Access

- Authenticated users

#### Parameters

This endpoint does not accept any parameters.

#### Response

- **200 OK**
  - (**User | Organizer**) The user object if the user was retrieved successfully.
- **400 Bad Request**
  - If the token is invalid.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user is not authenticated.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the user could not be retrieved.
  - Returns an **Error** object.

### PUT /me

Updates the current user.

#### Access

- Authenticated users

#### Parameters

#### Body Parameters



- **name**: (string) Optional. The name of the user.
- **email**: (string) Optional. The email address of the user.
- **password**: (string) Optional. The password of the user in plain text.
- **currentPassword**: (string) Optional. The current password of the user in plain text. Required if **password** is specified, otherwise it will not update the password.
- **User** only:
  - **newsletter**: (boolean) Optional. Whether the user is subscribed to the newsletter or not.
  - **preferences**: (`{ id: number; value: boolean }[]`) Optional. The preferences of the user. The **id** is the ID of the category and the **value** is whether the user wants to enable the category or not.
- **Organizer** only:
  - **description**: (string) Optional. The description of the organizer.
  - **contact\_email**: (string) Optional. The contact email address of the organizer.
  - **phone**: (string) Optional. The phone number of the organizer.
  - **website**: (string) Optional. The website of the organizer.
  - **location**: ([Location](#)) Optional. The location of the organizer.

### Response

- **200 OK**
  - (**User | Organizer**) The user object if the user was updated successfully.
- **400 Bad Request**
  - If the token is invalid.
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - If the specified user does not exist.
  - If the password is incorrect.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user is not authenticated.
  - Returns an **Error** object.
- **404 Not Found**
  - If the user does not exist.
  - Returns an **Error** object.
- **409 Conflict**
  - If the email address is already in use.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the user could not be updated.
  - Returns an **Error** object.

### DELETE /me

Deletes the current user.

### Access

- Authenticated users

#### Parameters

This endpoint does not accept any parameters.

#### Response

- **200 OK**
  - (**Success**) The success object if the user was deleted successfully.
- **400 Bad Request**
  - If the token is invalid.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user is not authenticated.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the user could not be deleted.
  - Returns an **Error** object.

#### GET /me/preferences

Gets the preferences of the current user.

#### Access

- Authenticated **User** users

#### Parameters

This endpoint does not accept any parameters.

#### Response

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

#### GET /me/events

Gets all events, including private events, of the current user.

#### Access

- Authenticated **Organizer** users

#### Parameters

##### Query Parameters

- **limit**: (number) Optional. The maximum number of events to return. Defaults to unlimited.
- **offset**: (number) Optional. The number of events to skip. Defaults to **0**.
- **order**: (Order) Optional. The order of the events. Can be of type **Order** from **sequelize**.
- **category\_id**: (number[]) Optional. The IDs of the categories to filter by.
- **organizer\_id**: (number[]) Optional. The IDs of the organizers to filter by.

#### Response

- **200 OK**
  - (**Event[]**) An array of event objects if the events were retrieved successfully.
- **400 Bad Request**
  - If the token is invalid.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user is not authenticated.
  - If the user type is missing from the token.
  - Returns an **Error** object.
- **403 Forbidden**
  - If the user is not an **Organizer**.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the events could not be retrieved.
  - Returns an **Error** object.

#### GET /me/events/:id

Gets an event of the current user.

#### Access

- Authenticated **Organizer** users

#### Parameters

##### Path Parameters

- **id**: (number) Required. The ID of the event.

**Response**

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

**/categories**

Handles categories.

**GET /categories**

Gets all categories.

**Access**

- Everyone

**Parameters****Query Parameters**

- **limit**: (number) Optional. The maximum number of categories to return. Defaults to unlimited.
- **offset**: (number) Optional. The number of categories to skip. Defaults to **0**.
- **order**: (Order) Optional. The order of the categories. Can be of type **Order** from **sequelize**.

**Response**

- **200 OK**
  - (**Category[]**) An array of category objects if the categories were retrieved successfully.
- **500 Internal Server Error**

- If the categories could not be retrieved.
- Returns an **Error** object.

### **GET /categories/:id**

Gets a category.

#### **Access**

- Everyone

#### **Parameters**

##### **Path Parameters**

- **id**: (number) Required. The ID of the category.

#### **Response**

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

### **POST /categories**

Creates a new category.

#### **Access**

- Admin users

#### **Parameters**

##### **Body Parameters**

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

#### **Response**

- **201 Created**

- (number) The ID of the category if the category was created successfully.
- **400 Bad Request**
  - If the token is invalid.
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user type is missing from the token.
  - Returns an **Error** object.
- **403 Forbidden**
  - If the user is not an administrator.
  - Returns an **Error** object.
- **409 Conflict**
  - If the category already exists.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the category could not be created.
  - Returns an **Error** object.

### **PUT /categories/:id**

Updates a category.

#### **Access**

- Admin users

#### **Parameters**

##### **Path Parameters**

- **id**: (number) Required. The ID of the category.

##### **Body Parameters**

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

#### **Response**

- **200 OK**
  - (**Category**) The category object if the category was updated successfully.
- **400 Bad Request**
  - If the token is invalid.
  - If the ID is not supplied or not a number.
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an **Error** object.

- **401 Unauthorized**
  - If the user type is missing from the token.
  - Returns an **Error** object.
- **403 Forbidden**
  - If the user is not an administrator.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the category could not be updated.
  - Returns an **Error** object.

### **DELETE /categories/:id**

Deletes a category.

#### **Access**

- Admin users

#### **Parameters**

##### **Path Parameters**

- **id**: (number) Required. The ID of the category.

#### **Response**

- **200 OK**
  - (**Success**) The success object if the category was deleted successfully.
- **400 Bad Request**
  - If the token is invalid.
  - If the ID is not supplied or not a number.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user type is missing from the token.
  - Returns an **Error** object.
- **403 Forbidden**
  - If the user is not an administrator.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the category could not be deleted.
  - Returns an **Error** object.

### **/events**

Handles events.

### **GET /events**

Gets all public events, which are not in the past.

#### Access

- Everyone

#### Parameters

##### Query Parameters

- **limit**: (number) Optional. The maximum number of events to return. Defaults to unlimited.
- **offset**: (number) Optional. The number of events to skip. Defaults to 0.
- **order**: (Order) Optional. The order of the events. Can be of type **Order** from **sequelize**.
- **category\_id**: (number[]) Optional. The IDs of the categories to filter by.
- **organizer\_id**: (number[]) Optional. The IDs of the organizers to filter by.

#### Response

- **200 OK**
  - (**Event[]**) An array of event objects if the events were retrieved successfully.
- **500 Internal Server Error**
  - If the events could not be retrieved.
  - Returns an **Error** object.

#### GET /events/:id

Gets an event.

#### Access

- Everyone

#### Parameters

##### Path Parameters

- **id**: (number) Required. The ID of the event.

#### Response

- **200 OK**
  - (**Event**) The event object if the event was retrieved successfully.
- **400 Bad Request**
  - If the ID is not supplied or not a number.
  - Returns an **Error** object.
- **404 Not Found**
  - If the event does not exist, is private or the organizer is not verified.
  - Returns an **Error** object.



- **500 Internal Server Error**
  - If the event could not be retrieved.
  - Returns an **Error** object.

## POST /events

Creates a new event.

### Access

- Authenticated **Organizer** users

### Parameters

#### Body Parameters

- **title**: (string) Required. The title of the event.
- **description**: (string) Optional. The description of the event.
- **start\_date**: (string) Optional. The start date of the event.
- **end\_date**: (string) Optional. The end date of the event.
- **start\_time**: (string) Optional. The start time of the event.
- **end\_time**: (string) Optional. The end time of the event.
- **public**: (boolean) Optional. Whether the event is public or not. If set to **true**, **description**, **start\_date**, and **category\_id** are required.
- **category\_id**: (number) Optional. The ID of the category of the event.
- **location**: (**Location**) Optional. The location of the event.

### Response

- **201 Created**
  - (number) The ID of the event if the event was created successfully.
- **400 Bad Request**
  - If the token is invalid.
  - If the user type is missing from the token.
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user is not authenticated.
  - If the user ID is missing from the token.
  - Returns an **Error** object.
- **403 Forbidden**
  - If the user is not an **Organizer**.
  - If the organizer is not verified and the event is set to public.
  - Returns an **Error** object.
- **404 Not Found**
  - If the organizer does not exist.

- Returns an **Error** object.
- **500 Internal Server Error**
  - If the event could not be created.
  - Returns an **Error** object.

## PUT /events/:id

Updates an event.

### Access

- Authenticated **Organizer** users

### Parameters

#### Path Parameters

- **id**: (number) Required. The ID of the event.

#### Body Parameters

- **title**: (string) Optional. The title of the event.
- **description**: (string) Optional. The description of the event.
- **start\_date**: (string) Optional. The start date of the event.
- **end\_date**: (string) Optional. The end date of the event.
- **start\_time**: (string) Optional. The start time of the event.
- **end\_time**: (string) Optional. The end time of the event.
- **public**: (boolean) Optional. Whether the event is public or not. If set to **true**, **description**, **start\_date**, and **category\_id** are required.
- **category\_id**: (number) Optional. The ID of the category of the event.
- **location**: (**Location**) Optional. The location of the event.

### Response

- **200 OK**
  - (**Event**) The event object if the event was updated successfully.
- **400 Bad Request**
  - If the token is invalid.
  - If the ID is not supplied or not a number.
  - If the user type is missing from the token.
  - If the request body is missing any of the required parameters.
  - If the request body contains invalid parameters.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user is not authenticated.
  - If the user ID is missing from the token.
  - Returns an **Error** object.

- **403 Forbidden**
  - If the user is not an **Organizer**.
  - If the owner of the event is not the current user.
  - If the organizer is not verified and the event is set to public.
  - Returns an **Error** object.
- **404 Not Found**
  - If the event does not exist.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the event could not be updated.
  - Returns an **Error** object.

## **DELETE /events/:id**

Deletes an event.

### **Access**

- Authenticated **Organizer** users

### **Parameters**

#### **Path Parameters**

- **id**: (number) Required. The ID of the event.

### **Response**

- **200 OK**
  - (**Success**) The success object if the event was deleted successfully.
- **400 Bad Request**
  - If the token is invalid.
  - If the ID is not supplied or not a number.
  - If the user type is missing from the token.
  - Returns an **Error** object.
- **401 Unauthorized**
  - If the user is not authenticated.
  - If the user ID is missing from the token.
  - Returns an **Error** object.
- **403 Forbidden**
  - If the user is not an **Organizer**.
  - If the owner of the event is not the current user.
  - Returns an **Error** object.
- **404 Not Found**
  - If the event does not exist.
  - Returns an **Error** object.
- **500 Internal Server Error**

- If the event could not be deleted.
- Returns an **Error** object.

## /organizers

Handles organizers.

### GET /organizers

Gets all verified organizers.

#### Access

- Everyone

#### Parameters

##### Query Parameters

- **limit**: (number) Optional. The maximum number of organizers to return. Defaults to unlimited.
- **offset**: (number) Optional. The number of organizers to skip. Defaults to **0**.
- **order**: (Order) Optional. The order of the organizers. Can be of type **Order** from **sequelize**.
- **category\_id**: (number[]) Optional. The IDs of the categories to filter by.

#### Response

- **200 OK**
  - (**Organizer[]**) An array of organizer objects if the organizers were retrieved successfully.
- **500 Internal Server Error**
  - If the organizers could not be retrieved.
  - Returns an **Error** object.

### GET /organizers/:id

Gets a verified organizer.

#### Access

- Everyone

#### Parameters

##### Path Parameters

- **id**: (number) Required. The ID of the organizer.

#### Response

- **200 OK**

- (**Organizer**) The organizer object if the organizer was retrieved successfully.
- **400 Bad Request**
  - If the ID is not supplied or not a number.
  - Returns an **Error** object.
- **404 Not Found**
  - If the organizer does not exist or is not verified.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the organizer could not be retrieved.
  - Returns an **Error** object.

## /search

Handles search.

### GET /search/:query

Searches for events and organizers.

#### Access

- Everyone

#### Parameters

##### Path Parameters

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

##### Query Parameters

- **limit**: (number) Optional. The maximum number of results to return. Defaults to unlimited.
- **offset**: (number) Optional. The number of results to skip. Defaults to **0**.
- **order**: (Order) Optional. The order of the results. Can be of type **Order** from **sequelize**.
- **category\_id**: (number[]) Optional. The IDs of the categories to filter by.
- **organizer\_id**: (number[]) Optional. The IDs of the organizers to filter by.

#### Response

- **200 OK**
  - (object) An object containing the results. It has the following properties:
    - **events**: (**Event**[]) An array of event objects.
    - **organizers**: (**Organizer**[]) An array of organizer objects.
- **400 Bad Request**
  - If the query is not supplied or not a string.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the results could not be retrieved.

- Returns an **Error** object.

## /search/events/:query

Searches for events.

### Access

- Everyone

### Parameters

#### Path Parameters

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

#### Query Parameters

- **limit**: (number) Optional. The maximum number of results to return. Defaults to unlimited.
- **offset**: (number) Optional. The number of results to skip. Defaults to **0**.
- **order**: (Order) Optional. The order of the results. Can be of type **Order** from **sequelize**.
- **category\_id**: (number[]) Optional. The IDs of the categories to filter by.
- **organizer\_id**: (number[]) Optional. The IDs of the organizers to filter by.

### Response

- **200 OK**
  - (**Event**[]) An array of event objects if the events were retrieved successfully.
- **400 Bad Request**
  - If the query is not supplied or not a string.
  - Returns an **Error** object.
- **500 Internal Server Error**
  - If the events could not be retrieved.
  - Returns an **Error** object.

## /search/organizers/:query

Searches for organizers.

### Access

- Everyone

### Parameters

#### Path Parameters

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

### Query Parameters

- **limit**: (number) Optional. The maximum number of results to return. Defaults to unlimited.
- **offset**: (number) Optional. The number of results to skip. Defaults to `0`.
- **order**: (Order) Optional. The order of the results. Can be of type `Order` from `sequelize`.
- **category\_id**: (number[]) Optional. The IDs of the categories to filter by.

### Response

- **200 OK**
  - (`Organizer[]`) An array of organizer objects if the organizers were retrieved successfully.
- **400 Bad Request**
  - If the query is not supplied or not a string.
  - Returns an `Error` object.
- **500 Internal Server Error**
  - If the organizers could not be retrieved.
  - Returns an `Error` object.

# Chapter 16

## Task



# **Aufgabenstellung Semesterarbeit „What's up in RJ“**

**Autor: Frank Koch  
Version: 1.0**

**Anpasst am: 07.09.23**

---

## 1. Beteiligte Personen

- Studierende: Michael Enzler, Fabio Stocker
- Industriepartner: AdaptIT GmbH, Michael Güntensperger
- Betreuer: Frank Koch

## 2. Problembeschrieb

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

Ziel wäre deshalb die Entwicklung einer Website, ähnlich aufgebaut wie eine Newsseite mit dem Programm des heutigen / der nächsten Tage. Organisatoren sollen die Möglichkeit haben ihre Events aufzuschalten, Vereine sollen sich vorstellen können, usw.

Durch einen Login und die Auswahl persönlicher Interessen soll eine Seite inklusive Newsletter erstellt werden, welche auf entsprechende Events in den nächsten 2-4 Wochen aufmerksam machen.

Natürlich könnte die angestrebte Lösung auch in anderen Regionen eingesetzt werden.

## 3. Aufgabenstellung

Um die Zielgruppe anzusprechen soll eine responsive Website erstellt werden, die sowohl auf Desktop wie auch auf dem Handy übersichtlich dargestellt wird. Für die Eventorganisatoren gibt es einen Bereich, welcher Auswertungen zu den Events generiert, z.B. Buttonklicks, etc. Die Webseite soll nach aktuellen UI-Standards entwickelt werden. Ziel ist das Finden eines passenden Abendprogramms mit möglichst wenigen Klicks.

Gerne werden die Interessen und Ideen der Studierenden berücksichtigt.

### Technische Umgebung

Für die Umsetzung wird mit Web-Technologien gearbeitet.

- Frontend: React / Angular
- Backend: Node.js
- Datenbank: MySQL
- Host: Z.B. DigitalOcean

### Funktionale Anforderungen

- Ausgewählte Veranstaltende können Events erfassen
- Suchfunktion auf der Hauptseite, um entsprechende Events zu finden
- Filterfunktion nach Themengebieten z.B. Sport
- Login für Seitenbesuchende
- Seitenbesuchende können für Sie interessante Kategorien auswählen und erhalten entsprechende Newsletter

### Optionale Anforderungen

- Admin-Bereich um die Accounts von Veranstaltenden zu verwalten
  - Freigabeprozess von Events durch Admin
  - Dashboard für Veranstaltende mit Auswertungen z.B. Buttonklicks
  - Erstellung von Einzelseiten für Vereine
  - AI um auf Basis von gewählten Interessen, Newsletter mit Events zusammenzustellen
  - Mobile-App mit Push-Nachrichten
-

---

## Nicht-Funktionale Anforderungen

- Das Entwicklerteam implementiert die Features gemäss der abgesprochenen Priorität mit dem Kunden
- Das Backend sollte 1000 Requests pro Minute handeln können
- Jede Seite sollte nicht länger als 200ms für das Laden benötigen
- Die Seite soll sowohl auf dem Mobile, Tablet und Desktop gut aussehen (Responsive)
- Die Web-Applikation sollte auf Firefox, Chrome und Safari laufen.
- Via Internet sollte auf eine vom Kunden zur Verfügung gestellte Domain zugegriffen werden können.
- Drei von vier Test-user sollten das UI (Kategorien: layout, responsiveness, colour, content) der Applikation mit einem Tablet mit einer Note von mindestens 8 von 10 bewerten, wobei 10 das beste ist.
- Die Datenbank soll bis zu 10'000 Events und 1'000 Benutzende managen können.
- Errors sollen keine Systemfehler erzeugen, aber eine Error Nachricht Zeigen und das System auf den vorherigen Zustand zurücksetzen.
- Jeder Error soll im System geloggt werden
- Jede Kommunikation zwischen Fron- und Backend soll mit einem SSL-Zertifikat verschlüsselt werden.
- Daten welche in Eingabefelder abgefüllt werden, sollen zuerst validiert werden, bevor diese durch das System verarbeitet werden. SQL Injection test der Eingabefelder sollte keine Verletzlichkeiten zeigen.
- Die Webapplikation soll datenschutzkonform umgesetzt werden.
- User-Passwörter werden nicht in plain-text in der Datenbank gespeichert.
- Wenn sich ein User in die Web-Applikation einloggt, werden ihm auch nur seine Daten / auf Daten die er Zugriff haben soll, angezeigt.
- Businesslogik im Backend soll modular aufgebaut werden, so dass sie erweitert werden kann.
- Die Backend-API soll durch API-testing Tools getestet werden.
- Implementierte Funktionalität (Datenbank, Backend, Frontend,...) sollen deployed werden.

## 4. Zur Durchführung

Mit dem Betreuer finden Besprechungen gemäss Absprache statt. Die Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsergebnisse in vorläufigen Versionen abzugeben.

## 5. Dokumentation und Abgabe

Siehe Leitfaden Abschnitt 5.5 "Umfang und Form der Abgabe".

## 6. Termine

Siehe veröffentlichte «Termine SA HS23».

## 7. Bewertung

Siehe veröffentlichter «Leitfaden BA SA» Abschnitt 6 "Bewertung", insbesondere 6.4.

Rapperswil, den 28.08.23

Frank Koch

---