Documentation ÜK 223 - Team 4

Aryan Bisen, Karol Krawiec & Neslihan Avsar

Table of Contents

Important Links

Padlet

GitHub Repository Frontend

GitHub Repository Backend

Figma Design

Introduction

This document serves to understand the project expectations and the technical implementation and testing of it.

Project Description

OurSpace is a **multi-user web application** designed for participation and management of **events**. Our application uses a **user-role-authority** system to check the authorization of a person when performing certain actions, while still using the same API. Accounts are categorized into two groups: “User” and “Admin”. Users can create new events, edit and delete their own events as well as add other users to their events. Admins can manage the events of all users, but not participate in the events themselves.

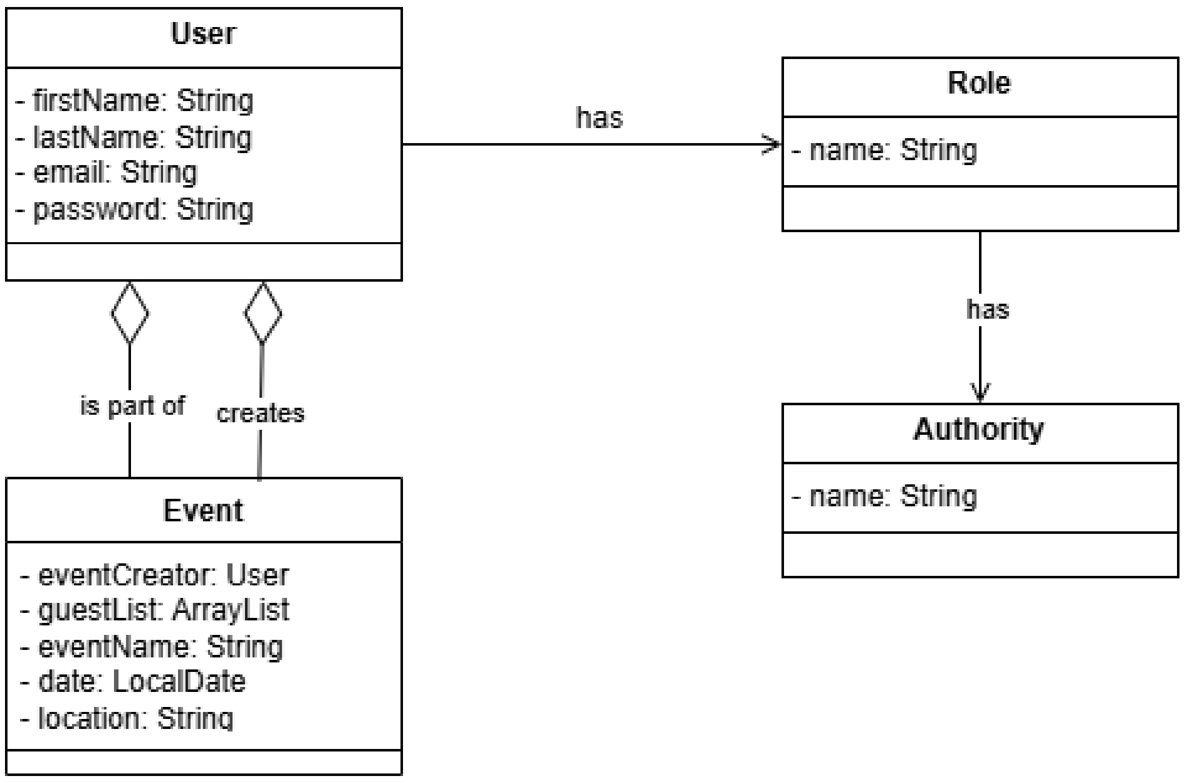
Project Technologies

The platform uses a **Spring Boot** backend for robust, scalable, and secure server-side logic while interacting with a **React** (Typescript) frontend. We use a **PostgreSQL Docker** container for our database and test our application with **Cypress** and **Postman**.

Planning

Domain-Model

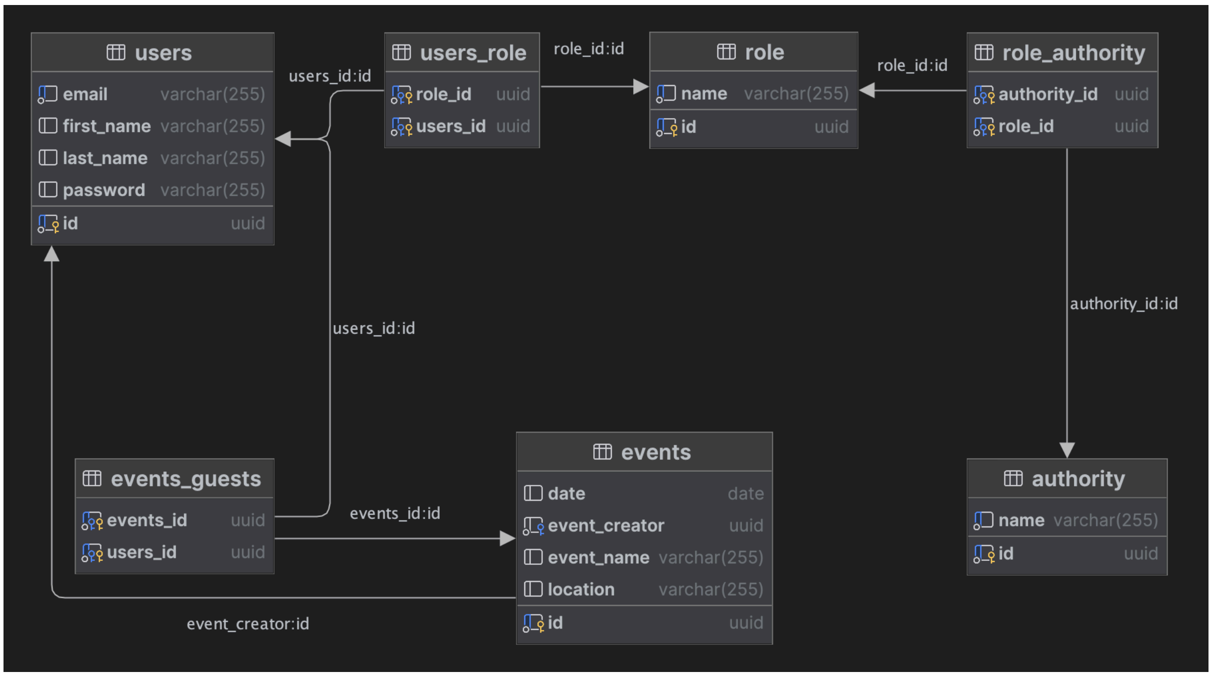
This domain model describes the various models we use for our application.



Here, you can see that the **User** model has a first name, last name, email and a password. The **User** model has a **Role**, which contains a name field. Every **Role** has one or more **Authorities**, which also contains a name field.

The **Event** model has a list of guests (which are **Users**), an event name, a date and a location. Every **Event** is created by one **User**.

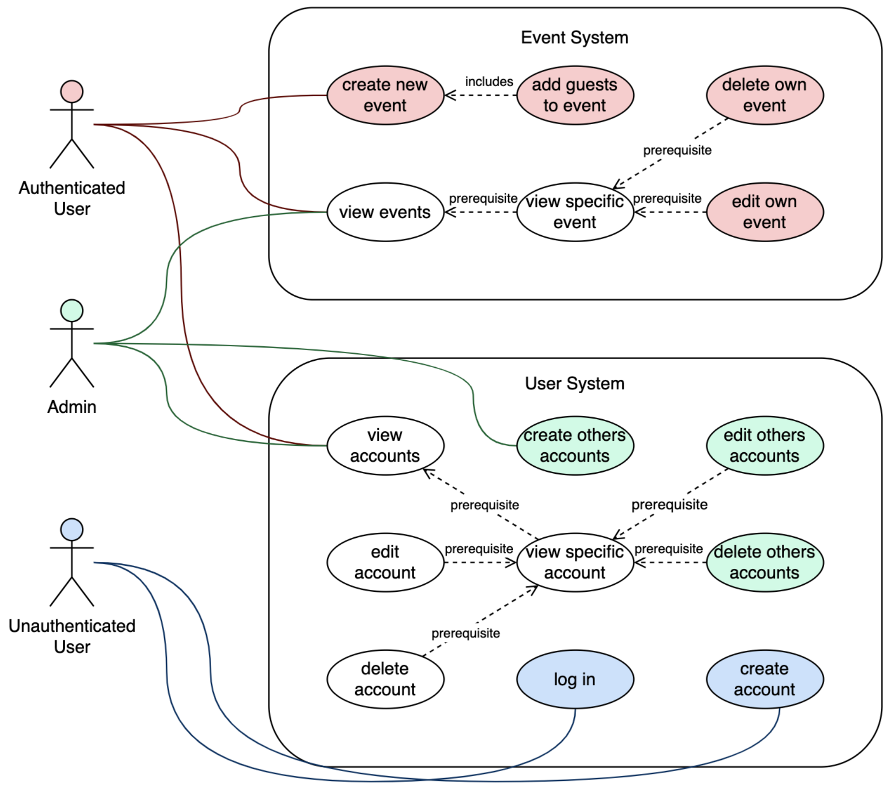
Entity-Relationship-Diagram



This is our ERD. We can see here that most relationships are a many-to-many relationship, which is why we have multiple tables for this purpose. Everything else on it has already been described in the previous section, hence, we will leave that out.

Use-Case-Diagram

This is a use-case diagram for the event system and the user system.



Color Code

red - authenticated user

green - admin

blue - unauthenticated user

white - authenticated user & admin

With the diagram, we describe all the actions regarding events and users / account management, which users can perform them and which actions include or require other actions.

Here is a table to summarize which actions can be performed by a specific group.

Authenticated User	**Admin**	**Authenticated User & Admin**	**Unauthenticated User**
create new event	create others accounts	view events	log in
add guests to event	edit others accounts	view specific event	create account
delete own event	delete others accounts	view accounts	
edit own event		view specific account	
		edit account	
		delete account	

Use-Case-Definitions

The Use Case Definition describes how a user interacts with a system to achieve a specific goals

Create Event	
Actor	User & Admin
Description	Create a new event with relevant details. URL: http://localhost:8080/create-event
Preconditions	➔ The user is registered and logged into the app.
Postconditions	➔ The event is successfully created and stored in the system.
Normal Course	➔ User creates a new event by providing event details (event creator, name, date, location, guest list). ➔ The app validates the input data. ➔ The user clicks on the save button. ➔ The app confirms the successful creation of the event.
Alternative Courses	➔ If mandatory details are missing or invalid, the app returns an error message.
Exceptions	➔ If the DB is unavailable or it didn't get saved in DB, an error is returned.

Retrieve all Events	
Actor	User & Admin
Description	View a list of all events. URL: http://localhost:8080/home
Preconditions	➔ At least one event exists.
Postconditions	➔ A list of events is displayed
Normal Course	➔ User requests a list of events. ➔ The app retrieves events from the database.
Alternative Courses	➔ If no events exist, the app returns an empty list.
Exceptions	➔ If the DB is unavailable, an error is returned.

Retrieve Event Details	
Actor	User & Admin
Description	View details of a specific event. URL: http://localhost:8080/event/:id
Preconditions	➔ Event exists in the DB.
Postconditions	➔ Event details are displayed.
Normal Course	➔ User requests event details using the event ID. ➔ The app fetches event information from the database and displays them..
Alternative Courses	➔ If the event ID is invalid, the app returns an error message.
Exceptions	➔ If the DB is unavailable, an error is returned.

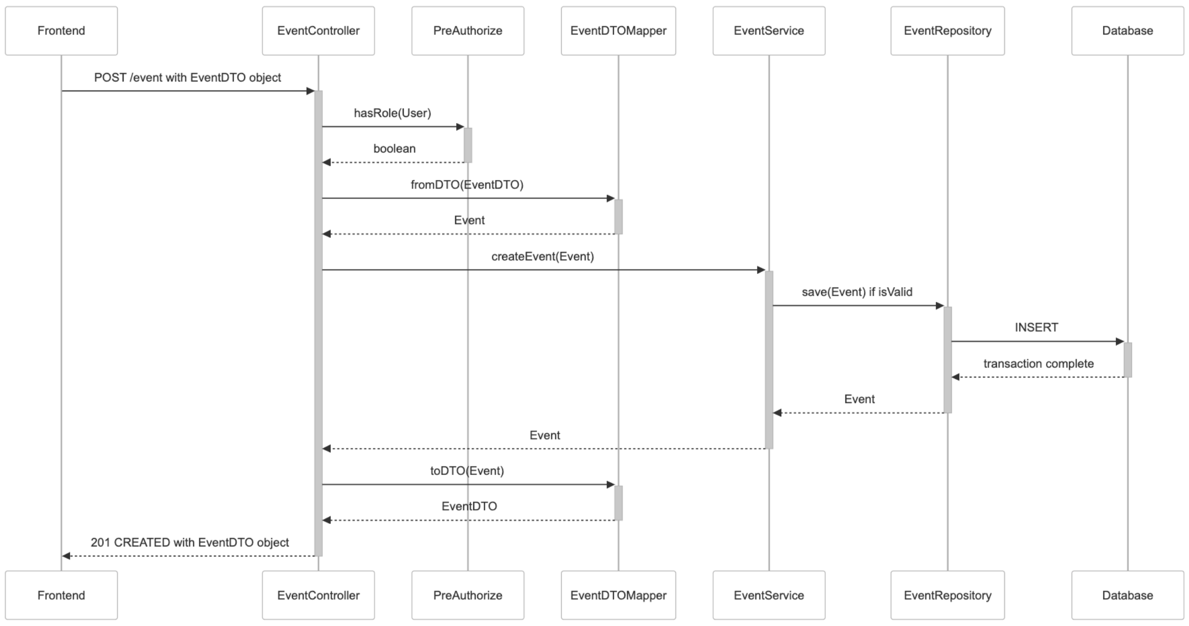
Update Event	
Actor	Event creator
Description	Modify event details. URL: http://localhost:8080/edit-event
Preconditions	➔ The user is the creator of the event.
Postconditions	➔ The event details are updated successfully.
Normal Course	➔ Event creator provides updated event details. ➔ The app verifies the user’s permission. ➔ The app updates the event details in the database and confirms the update on the web page.
Alternative Courses	➔ If the user is not the creator, an error is returned.
Exceptions	➔ If the DB is unavailable or it didn’t get updated in DB, an error is returned.

Delete Event	
Actor	Event creator
Description	Delete event details. URL: http://localhost:8080/event/:id
Preconditions	➔ The user is the creator of the event.
Postconditions	➔ The event gets deleted from the app successfully.
Normal Course	➔ User clicks on the delete button, which only shows up if he is the creator of the event. ➔ The system verifies if the user has the right to delete this event. ➔ App shows a confirmation after deletion is done.
Alternative Courses	➔ If the user is not the creator, an error is returned.
Exceptions	➔ If the DB is unavailable or it didn't get updated in DB, an error is returned.

List Guests of an Event (Pagination)	
Actor	User & Admin
Description	View a paginated list of all guests of an event URL: http://localhost:8080/event/:id
Preconditions	➔ The event exists and has guests.
Postconditions	➔ A paginated list of guests is displayed.
Normal Course	➔ User requests the guest list for a specific event. ➔ The app retrieves guest list data with pagination ➔ The app displays the paginated guest list.
Alternative Courses	➔ If no guests exist, the app returns an empty list.
Exceptions	None.

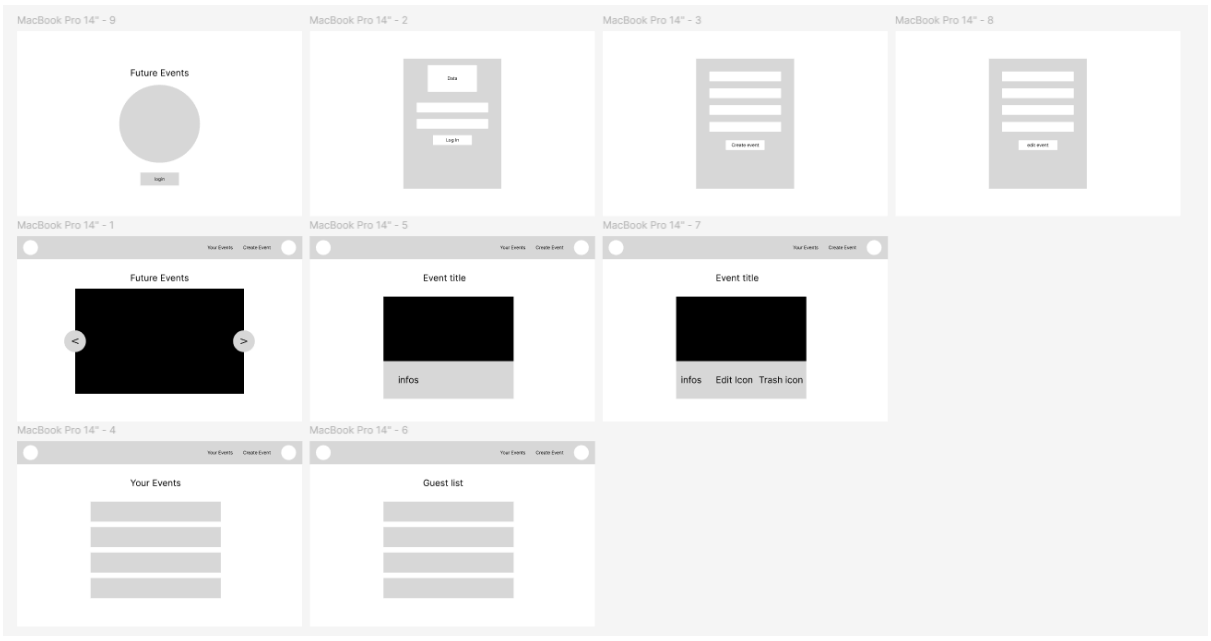
Sequence-Diagram

This sequence represents a typical sequence for when a user creates a new event, including authorization, DTO mapping, validation and database operations.



1. The **Frontend** sends a POST /event request with an EventDTO object to the **EventController**.
2. The EventController checks if the user has the required role by calling **PreAuthorize**, which returns a boolean indicating authorization status.
3. If authorized, the EventController converts the EventDTO into an Event object using **EventDTOMapper**.
4. The mapped Event object is then passed to **EventService**, which handles business logic. If the event is valid, EventService calls **EventRepository** to save it.
5. EventRepository performs an INSERT operation into the **Database**, and once the transaction is complete, the Database confirms it.
6. The saved Event object is then returned back up the chain from EventRepository to EventService and then to EventController.
7. The EventController converts the saved Event back into an EventDTO using EventDTOMapper.
8. The EventController responds to the Frontend with a 201 CREATED status, returning the EventDTO object.

Figma LoFi Prototype



We created a little LoFi prototype of the website. It includes where the different items and data should be. We also defined all of the sites that we need:

- Landing Page
- Home Page
- Login Page
- Create Event
- Edit Event
- Your Events
- Guest List
- Event Detail Page
- Event Creator Detail Page

Documentation and Testing

Swagger UI API Documentation

In order to properly document our API, we use Swagger UI. There, we can describe each of our endpoints with the @Operation annotation and our controllers with the @Tag annotation.

It can be accessed with this URL:

http://localhost:8080/swagger-ui/index.html

Testing Strategy

With **Postman**, we can test individual endpoints of our backend. We use this for component (and integration) tests.

For end-to-end tests, we use **Cypress**. This Cypress test describes the process of creating an event:

- Navigate to landing page (http://localhost:3000/)
- Click on LOGIN button (navigates to http://localhost:3000/login)
- Fill out login form
- Click on LOGIN button (navigates to http://localhost:3000/home)
- Click on create event (navigates to http://localhost:3000/create-event)
- Fill out create event form
- Click on CREATE EVENT button (creates the event in the database)

Endpoints Test Cases

GET /event (Get all events)

Test case	role	Request	Expected Status code	Expected response
Get all events successfully	Admin	GET /events with valid JWT	200 OK	An array of events with correct fields.
Unauthorized request	-	GET /events with no JWT	401 Unauthorized	Unauthentication error message is returned.

GET /event/{id} (Get event by id)

Test case	Role	Request	Expected Status code	Expected response
Get event successfully	User	GET /event with valid JWT	200 OK	Response contains event objects with correct fields.
Unauthorized request	-	GET /event with valid JWT	401 Unauthorized	Unauthentication error message is returned.

POST /event (Create a event)

Test case	Role	Request	Expected Status code	Expected response
Create a event successfully	User	POST /event with valid JWT	201 Created	Created event
Unauthorized request	-	POST /event with no JWT	401 Unauthorized	Unauthentication error message is returned.
Guest list in request contains a admin	User	POST /event with valid JWT	400 Bad request	Bad request error message with string: Cannot add admin to guest list.

PUT /event (Update a event by id)

Test case	Role	Request	Expected Status code	Expected response
Update a event successfully	User (Creator)	PUT /event with valid JWT	201 Created	Updated event
Unauthorized request	User (Not the creator)	PUT /event with invalid JWT	403 Forbidden	Forbidden error message with string: Can't update events you don't own.
Guest list in request contains a admin	User	PUT /event with valid JWT	400 Bad request	Bad request error message with string: Cannot add admin to guest list.

DELETE /event (Delete a event by id)

Test case	Role	Request	Expected Status code	Expected response
Delete a event successfully	User (Creator)	DELETE /event with valid JWT	200 OK	
Unauthorized request	User (Not the creator)	DELETE /event with invalid JWT	403 Forbidden	Forbidden error message with string: Can't delete events you don't own.