Delft University of Technology

Assignment 1 (Draft)

Authors	*Group 31b*
Alexandra Darie	5511100
Elena Dumitrescu	5527236
Robert Vadastreanu	5508096
Sander Vermeulen	5482127
Xingyu Han	5343755
Zoya van Meel	5114470

December 2, 2022

1. Introduction

Our task is to implement a scheduling system for the rowing associations in Delft, in order to facilitate the process of finding training sessions and competitions. In order to design our architecture, we have used a Domain-Driven Design approach, following the steps presented in the Software Engineering Methods lectures.

The starting point was gathering the requirements from the given scenario. Afterward, we constructed a Context Map and a Component Diagram to showcase the architecture. In the following sections, we explain in more detail the process of designing these models.

2. Context Map

The first step in designing our architecture was identifying the bounded contexts from the domain specifications. Our current model contains four domains in total. The User, the Event and the Activity Scheduler are our core domains, while the Authentication is a generic domain. Based on these, we have constructed the Context Map which can be seen in Figure 1.

2.1 Users & Authentication

The first domain we identified was the User domain, which handles all data and functionality related to the users of our application. The data includes information like the user's rowing certificates, gender, and organization. In terms of functionality, users are able to perform two main tasks: enrolling in an event and creating an event.

For participating in an event, the user is asked to specify when they are available to participate in events and the positions they are capable of filling inside a team (out of the five possible positions). This information is used by other contexts to determine which events fit with the user's preferences and capabilities. For creating an event, an user is sending a message to the Events context. The user needs to precise exactly how many people from each position are needed in order to organise the event. Moreover, if the created event is a competition, the user have to precise the requirements (the gender, the organisation and if it is a competition only for professionals).

Since the users have accounts within the application, they will be able to authenticate using their unique IDs and passwords. This step will be fulfilled inside the Authentication Domain using the Spring Security framework. Due to the fact that logging and signing

in will take place within the aforementioned part, this domain will also keep track of the pairs of passwords and their corresponding IDs.

2.2 Events

The third domain we identified was the Event domain. This domain handles the structure of the events, which can be either training sessions or competitions. These types of events have different entry requirements. As an example, users that participate in a competition all have to be of the same gender and users competing as one team all have to be from the same organisation. The requirements are specified by the users when creating the events and are used by the Scheduler when matching users to available events.

All aspects of the event are specified by the event creator (owner). The owner specifies where the event takes place and at what date, but also which positions in the boat have already been filled. Every attribute of an event is taken into consideration by the Scheduler in order to find suitable users.

2.3 Activity Scheduler

The fourth domain we found was the Activity Scheduler. This domain's main purpose is to mediate the communication between the Event and the User domain. Firstly, the Scheduler is used to match the users with the available events, taking into consideration the user's availability and preferences and the event's date and restrictions. In addition, this domain stores the schedules, in order to know in which events the users are enrolled.

Another purpose of this domain is handling notifications, which are sent between the participating users and the event owners. After a user submits their request to join an event, the owner will be notified that they need to approve the user's request. If the owner approves, the scheduler notifies that user that they can join the event and also sends a request to the Event domain to update the available positions of that certain event.

Notifications are also used when a user wants to withdraw from an event, in which case a notification will be sent to the event owner and the available positions will be updated in the Event domain.

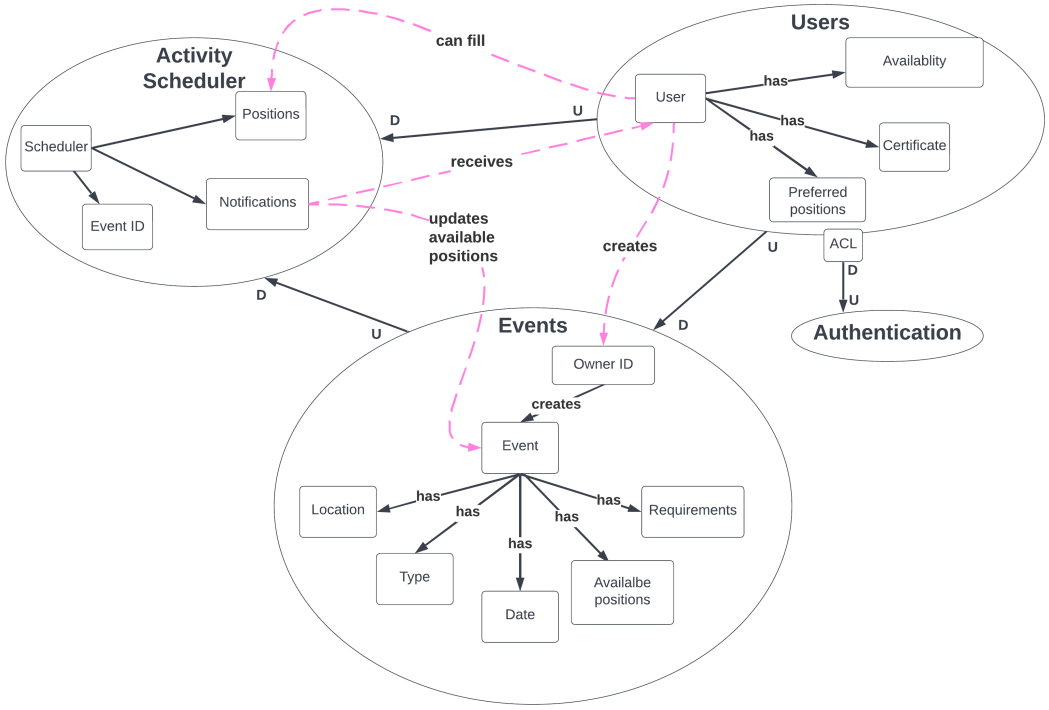

Figure 1: Context Map for the rowing application.

3. Component Diagram

To continue our design process, we have constructed an UML Component Diagram based on the previous Context Map, in which we mapped the bounded contexts into microservices and modelled in a more extensive way the relationship between them. The UML Diagram is shown in Figure 2.

The User microservice provides the interface CreateEvent, which is used by the Event microservice for retrieving the data needed to instantiate a new event. Another output of the User microservice is the AccessData interface, which will be used by the ActivityScheduler for both matching an user with an event and sending notifications. In addition, this microservice requires the Encryption and the AccesControl interfaces, which are provided by the Security microservice. These inputs are used for securely storing the credentials of the user in a database.

The ActivityScheduler microservice provides two interfaces: Notifications and Upda-

teEvents. The Notifications interface's main purpose it to collect the response of the event owner (approval/rejection) regarding users' requests to enroll in the event. In case of a positive answer, the ActivityScheduler notifies the selected user through the Notification interface. Another interface provided by this microservice is UpdateEvents. It announces the Event microservice when one position is occupied (if the owner accepted the enrollment request) or marks a position as available in case a user changes their mind regarding their participation. As input, the ActivityScheduler requires the AvailableEvents interface of the Event microservice in order to match the users with the fitted events and the UserData interface, which was explained in the User section.

The Event microservice provides the AvailableEvents interface and requires the CreateEvent and UpdateEvents interfaces, which were all presented earlier. In order for the three microservices to access the databases and cache we added a persistence layer component which facilitates the interaction between requests and component databases through repositories. This microservice provides what we called "Persistence" interface.

At the very end, we decided to use three separate databases, one for each microservice. Their sole purpose is to store the users that navigate through the application environment, events, namely training sessions or competitions, and the scheduled activities, resulting from users requests and current resources. The flow of the stated actions will be delivered using Java Database Connectivity interface.

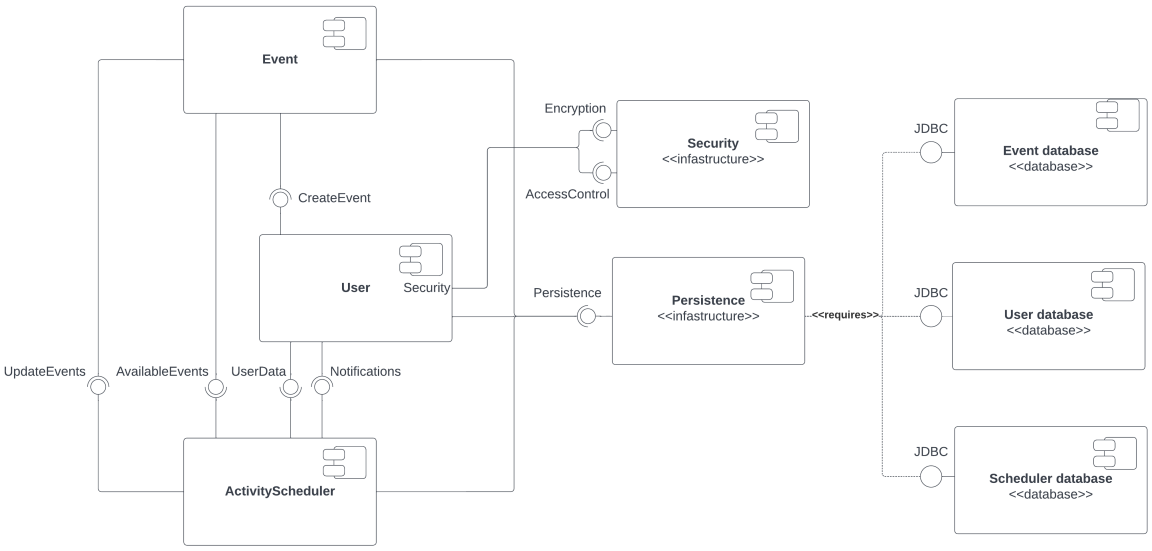

Figure 2: UML Diagram for the rowing application.