Software Engineering Methods – Group 34B

Task 1: Software Architecture

Group 34B: Members:

- Maarten van der Weide
- Rithik Appachi Senthilkumar
- Stefan Creasta
- Viet Luong
- Vlad Iftimescu
- Yongcheng Huang

Scenario assigned: Rowing

Contents:

Explanation of identified bounded contexts:

In our assigned rowing system, we have identified three core contexts, and a generic context.

The core contexts are the User, Activity, and Boat contexts. We have identified them as core contexts due to the fact that they are specific and essential to the system.

It also has an Authentication context, which is deemed to be generic, since it is not specific to this system.

1. **User (core):** the context that represents the actual users of the applications. It houses the essential information for identifying individual users of the system, such as their username and certifications.
   Users, in the context of Activities, can then become:
   a) Organizers, which then becomes an Owner. Owners receive additional permissions for the Activities that they own.
   b) Attendants (participants), who participate in the activity made by the owner. They receive notifications about specific activities. Each rower then gets assigned a specific Position for a Boat in the context of a specific Activity.

2. **Activity (core):** The context that houses all the information necessary for creating, managing and removing Activities of all types. It contains attributes such as the Owner of the Activity, the participants, and a timeslot.
   There are two main types of activities:
   a) Training sessions, represented by the Training class.
   b) Competitions, housed in the Competition class. There are additional restrictions imposed on entering a Competition as opposed to a Training, such as gender, organizational and status (amateur/professional) restrictions.

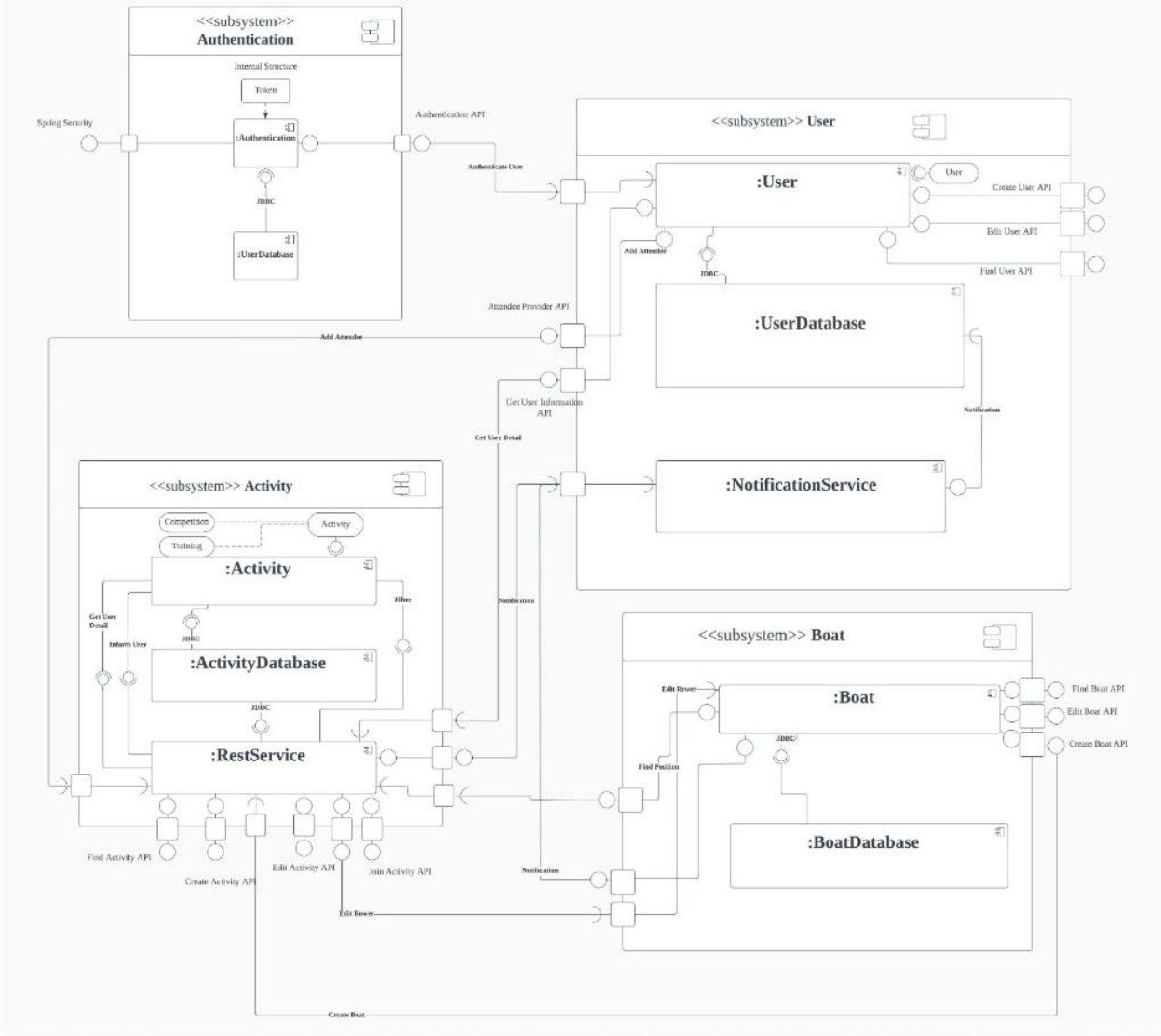
3. **Boat (core):** The context that encapsulates all the important information for the different boats that are part of certain activities.
   A boat contains
   a) A certain type.
   b) Multiple positions. A user can specify which positions they are capable of filling.
4. **Authentication (generic):** It is responsible for all the security-related aspects of the system. It takes care of functionalities such as user authentication, their permissions and the verification of the requests that are sent to the system. All of the security is handled by the Spring Security service, which means that Users will have to register and log in using a NetId and a password.

Mapping contexts to microservices:

We make use of microservices in our software architecture in order to handle the different bounded contexts.

In accordance with our contexts, we created four microservices – Activity, User, Boat, and Authentication. Each core microservice is crucial to the application, as everyone is responsible for modifying and updating a particular database. Thus, we have 3 databases: one for activities, one for boats and one for users. Because there are 3 separate databases, having fewer core microservices would be impractical, as this would mean that one microservice would manage 2 tables and scaling the whole project would become more difficult. At the same time, having additional microservices would serve no purpose, as this would lead to unnecessary dependencies between microservices. We believe that the optimal number of core microservices is 3, and with the functionality provided by the authentication microservice, which ensures that each user is registered correctly in the application, this would add up to a total of 4. The communication style between the microservices is synchronous, because waiting for a response is desirable. The architecture pattern for every microservice is hexagonal, due to the fact that there is a predefined interface for the communication between them and switching one microservice with another would pose no problem, as long as it respects the initial interface. This is valid for the databases as well. As for the security architecture, the RestService component inside the Activity microservice acts as a pseudo API Gateway, enabling the Activity microservice to communicate with the Boat and the User microservices. We implemented this way because only the Activity microservice requires communication with the rest.

UML Component Diagram for the overall architecture:



Explanation for each microservice:

a) Authentication microservice:

The authentication microservice in our project is responsible for providing users with a token that enables them to access all other microservices. This token is granted by the authentication microservice in exchange for correct login information consisting of a netID and a password. The authentication microservice exposes this functionality with 2 API-endpoints, one is called register and one is called authenticate.

Given that a user tries to authenticate themself with the authenticate API endpoint - the Authentication controller uses the AuthenticationManager interface provided by Spring

Security to authenticate the details provided by the user. This is achieved using the JwtUserDetailsService to see if the provided details match a stored record in the database. Upon successful authentication, the controller uses the JwtTokenGenerator to create a Jwt token, which is returned to the requestee.

For user registration via the register API endpoint - the controller uses the RegistrationService to store the new user record in the database, where the RegistrationService uses its UserRepository to interface with the database.

b) Activity microservice:

The activity microservice is designed to maintain and interact with all the persisted activities or persist more activities. An activity consists of the following elements: a name, a corresponding boat, a list of attendees, a starting time and the maximum number of people that can partake in the activity. Furthermore a competition extends on this and adds additional elements, these are: is the activity available for amateurs, a possible gender constraint, if only attendees of the same organization can partake and lastly the organization. The service uses a hierarchical structure to distinguish between training and competitions. This structure allows for easy scalability if the need ever arises to add more types of activities.

To facilitate actual functionality a series of endpoints are defined in the microservice. One mapping is defined for users to be able to request to join an activity, another is to create an activity, another is to accept or reject a participation request and lastly there are mappings to allow for editing and or canceling a certain activity.

Some of these mappings require interaction with other microservices. When an activity is created, the microservices makes a request to the boat microservice to persist a boat of a certain type associated with this activity. When a user requests to join a competition, the microservice requests some user data to check if the requesting user complies with the defined constraints a competition could have. The above 2 descriptions are illustrative examples, in practice there are more interactions between the activity microservice and the other services.

These requests are implemented via a RestService, this is a class that uses Spring to perform arbitrary HTTP requests, we then use this service in such a way that it interacts with endpoints defined in the other microservices, and use their response as necessary.

All the above makes it possible for the activity microservice to fulfill its responsibility of persisting and interacting with activities.

c) Boat microservice

The boat microservice is responsible for storing for each boat its type, the positions which still need to be filled, as well as the current rowers for that respective boat. There are 2 main components in this microservice which are also present in the diagram: the boat service and the boat database. Thus, this microservice has 6 API-endpoints

through which the user can modify the fields of a boat or create one. The boat communicates with the activity microservice through 5 API-endpoints, the last one being the Notification endpoint between this microservice and the user one (responsible for updating the tables).

Because the boat microservice receives a Jwt token from the RestService component, the user can insert or remove a rower from a certain boat using the Edit Rower API-endpoint. In order to find or create a boat, or edit the required positions for a boat, he can either use the Find Boat, Create Boat, or Edit Boat API-endpoints. Last but not least, one might want to find out all of the available boats which have at least one certain position available. The user can find this by using the Find Position endpoint.

The boat service is responsible for updating and retrieving the data from the database and the boat controller is responsible for handling all the requests which came through the API endpoints.

d) User microservice:

The user microservice is responsible for storing for each user, their NetId, gender, certification, status, and organization. The user microservice consists of its three main components - the UserService, NotificationService and the UserDatabase. The User microservice's API endpoints include fundamentally the endpoints for creating the user, finding the user and editing the user details. Moreover, the User Microservice communicates with the Activity microservice for adding attendees, getting user details. The user microservice's NotificationService notifies the user in the case of new events.

For a user to create a User object, edit the User object and find a user, they can use the Create User, Edit User and Find User API-endpoints. The Activity microservice interacts with the User microservice using its RestService component. Say a user wants to join an activity, they make the request via the Activity microservice. It then passes on the token to the User microservice, and the User microservice returns an object including the relevant details necessary for joining an activity.

The user microservice also has endpoints that the activity microservice uses to inform the activity owner of a new application made by another user to join their activity, and to inform the user about the result of their application. Such details are notified to the user via its Notification Service.

A crucial feature of the user microservice is its Notification Service. Any information provided to the user goes straight into their 'notifications' mailbox. This could include application results (for when a user applies for a competition/training), and application details for the activity owner (for them to see who all have applied to their activity). The user service is responsible for updating and retrieving the data from the database and the user controller is responsible for handling all the requests which came through the API endpoints.