Software Architecture

Bounded contexts

In order to correctly identify the bounded contexts of our domain we performed the following procedure:

1. We firstly performed a simple word cloud analysis on our scenario description in order to easily visualise the most important parts of our project.
   The words with the most relevance are Training, Competition, Certificate, Account, User, Security. (See Fig 1 in the Appendix)

2. We used these previously found words and some more we decided to add to construct logical groups of the items. (See Fig 2 in the Appendix)

3. We used the previous logical groups to build our bounded contexts and created a bounded context map. (See Fig 3 in the Appendix)

Several bounded contexts were identified as the result of our procedure. They are the following:

- Users
- Activities
- Activity Finder
- Notifications
- Authentication

Users

The users are the main focus point of our system as they are the ones who will be interacting with it. A user is a complex entity that holds many attributes. This bounded context also shares space with Certificates which is closely related since a user will be able to specify which certificates they own.

Activities

Activities are events that users can choose to participate in, such as a training or a competition. These events have certain characteristics that the user will need to know and take advantage of. A closely related topic to activities are boats which will also take part of this context since an activity will need boats.

Activity finder

This bounded context handles finding a user an activity to join. It will take advantage of the user's attributes and the available activities to determine the best action to recommend the user.

Notifications

Users need to receive notifications when relevant events take place. For example a user might need to get notified if his password changed or that his activity reached the maximum number of people.

Authentication

This bounded context represents the need for security and will manage which users have access to what resources and the registration of a user.

Microservices

We decided to perform a one-to-one mapping from our bounded contexts to the microservices since their logical separation was sufficiently decoupled to use them directly. We instead decided to add one more microservice, the Gateway, which will be a facade for the end-user to interact with, greatly simplifying the system.

A comprehensive UML drawing depicting the workings and relations of the microservices can be found in Fig 4 in the Appendix.

Users Microservice

This microservice will handle the storage and interaction of the user entities and their certificates. It will expose as an API the following class of endpoints:

- Getting data about user or users
- Managing the details of a user
- Adding new users
- Getting data about certificates
- Managing the certificates
- Adding new certificates

In order to perform these actions the microservice must store a good representation of the data. The user entity is complex and contains many attributes. A simplified view on the user entity and its relation with certificates can be described by the following UML class diagram.

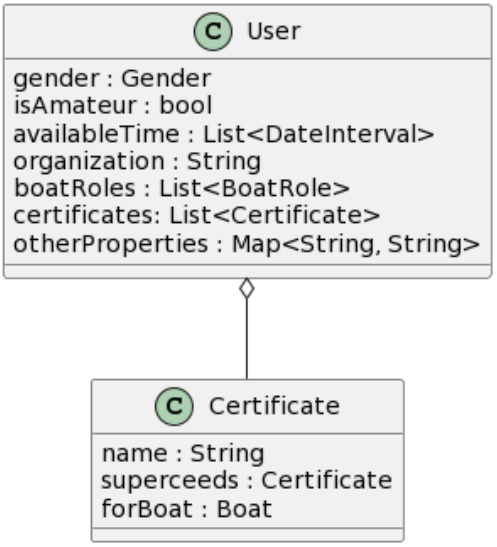

Fig 3 - User class diagram

The relation between Users and certificates constitutes a simple aggregation.
On the other hand both objects have cross service relations. More specifically the **User** has references to **BoatRoles** and **Certificate** has a reference to **Boat** which are both located inside the **Activities Microservice**. These relations are handled by storing a proxy to the real object in the other microservice. Once a service requires this information the proxy will automatically request it from the Activities microservice.

Activities Microservice

This microservice will handle the storage and interaction of the activity entities and of the boats. It will expose as an API the following class of endpoints:

- Getting data about activities
- Managing activities
- Getting data about boats
- Managing boats

In order to perform these actions it will need to represent the general concepts of activities and boats as entities. The activity entity is one of the more interesting as it actually encompasses two different types of activities, the training and the competition. The competition is a special type of training which allows the Activity Owner to impose restrictions on the users that may join(only allowing a specific gender, only allowing professional players or only allowing members of his organisation to take part).
A simplified view on the entities can be viewed in the following UML diagram.

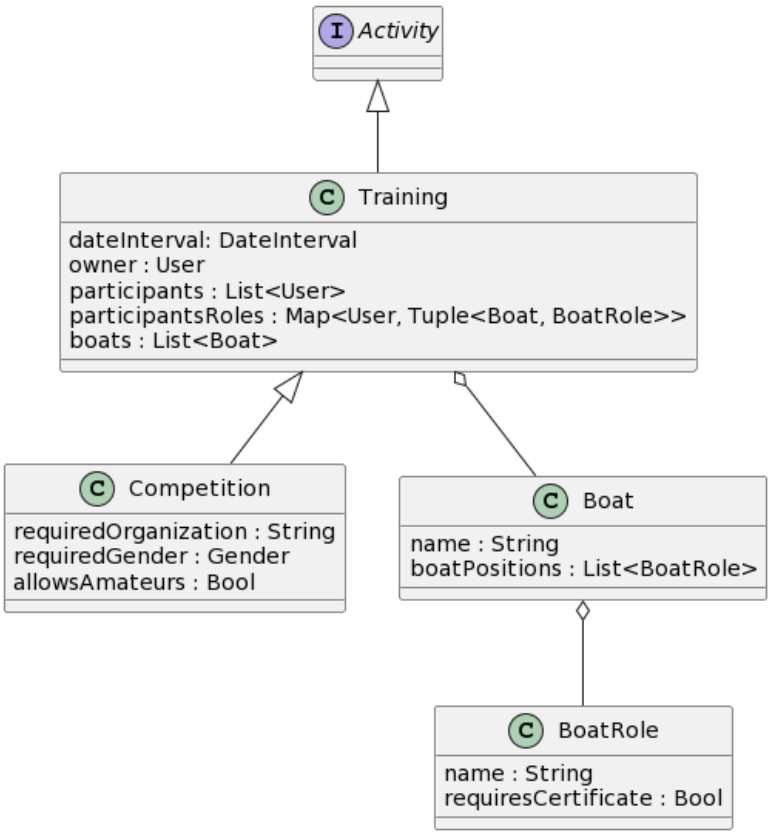

Fig 4 - Activity and boat entity relations

This diagram depicts the relations the different types of activities have and their members.
This diagram also showcases some relations to objects outside its scope(the User object). This object will be retrieved from the Users Microservice.

Activity finder Microservice

This microservice constitutes the main functionality of our system. It handles the searching and aggregation of the activities a user is able to join. The activity finder has close relations to the Users and to the Activities Microservice.
It will use the users in order to query and check information about certificates and users. At the same time it will use the activities microservice in order to get a batch of trainings or competitions and look for the ones the requesting user fulfils the requirements.

Using a separate microservice for matchmaking will allow the users and activities to be more loosely coupled. If the internal logic of either the users or the activities service changes, the matchmaking system will not need to take this into account. Also, by making this functionality a separate microservice the matchmaking system can be extended more easily to allow multiple roles to be specified by a user.

Notifications Microservice

The notifications microservice is the most decoupled of all of them. It has one purpose and that is to notify a user of an event by some predefined way. The current method to notify the user of an event involves sending a personalised email to their email inbox. The microservice exposes only one endpoint and that is the SendNotification endpoint which given a user, a type of event and the model with details relevant for the event will produce an email and send it to the user. The way this process is handled is by storing templates for every type of event there can be and also providing data models for them. Once a request enters the microservice it will fill in the template with the provided model and send the email using Spring Email.

Authentication Microservice

The authentication provides the security of the system.
The user is able to register using a userID, password and email. After registration, the password is hashed using a hashing service, this improves the security of someone's account. The microservice will also ensure that a user is indeed who they say they are and only allow the owner of the account to change their availability times, add certificates, change their password, etc.
Lastly the authentication will make a clear distinction between admins and users. For example, only the admins can create the competitions and training sessions.

Gateway Microservice

This is the microservice which did not have a direct mapping from one of the context maps which sets it apart from the rest. The purpose of this service is to simplify some of the more complex patterns of our system. The service will act as a facade to the user as it will constitute the only entrypoint into the system. Since it constitutes as the only entrypoint we can handle all authorization inside this microservice and later trust the requests coming from this microservice thus greatly reducing the amount of checks and tokens we have to pass around our system. On the other hand being the only entrypoint into our system could constitute a bottleneck as such if the number of users grows at an alarming rate, duplicating this service and using a load balancer would be recommended.

Design Patterns

2.1 Write a natural language description of why and how the pattern is implemented in your system (6 pts per each design pattern)

We decided on two types of design patterns in our system.

- The chain of responsibility
- The strategy pattern

We implemented a chain of responsibility in order to perform a sequence of verifications when a user performs a register request. We have multiple stages and they all check a different characteristic. For example one could check that the username is not already taken and another might check that the email has the right format. This pattern is implemented in the authentication microservice.

We implemented the strategy pattern in our activity finder microservice in order to be able to interchange different types of searching strategies. We need this functionality in order to allow different strategies to be applied at different times of the week. For example if there are less activities on monday we might choose to employ the Random strategy which will easily find our desired activity since there are not as many to consider. On the other hand if there are a lot of activities we might decide to employ the complex or the greedy strategy which employs some sort of heuristic in order to find the activity faster and use the least amount of resources.

2.2 Make a class diagram of how the pattern is structured in your code (3 pts per each design pattern)

2.3 Implement the design pattern (6pts per each pattern)

Appendix

Fig 1 - The word cloud analysis

Fig 2 - The logical groups of ideas

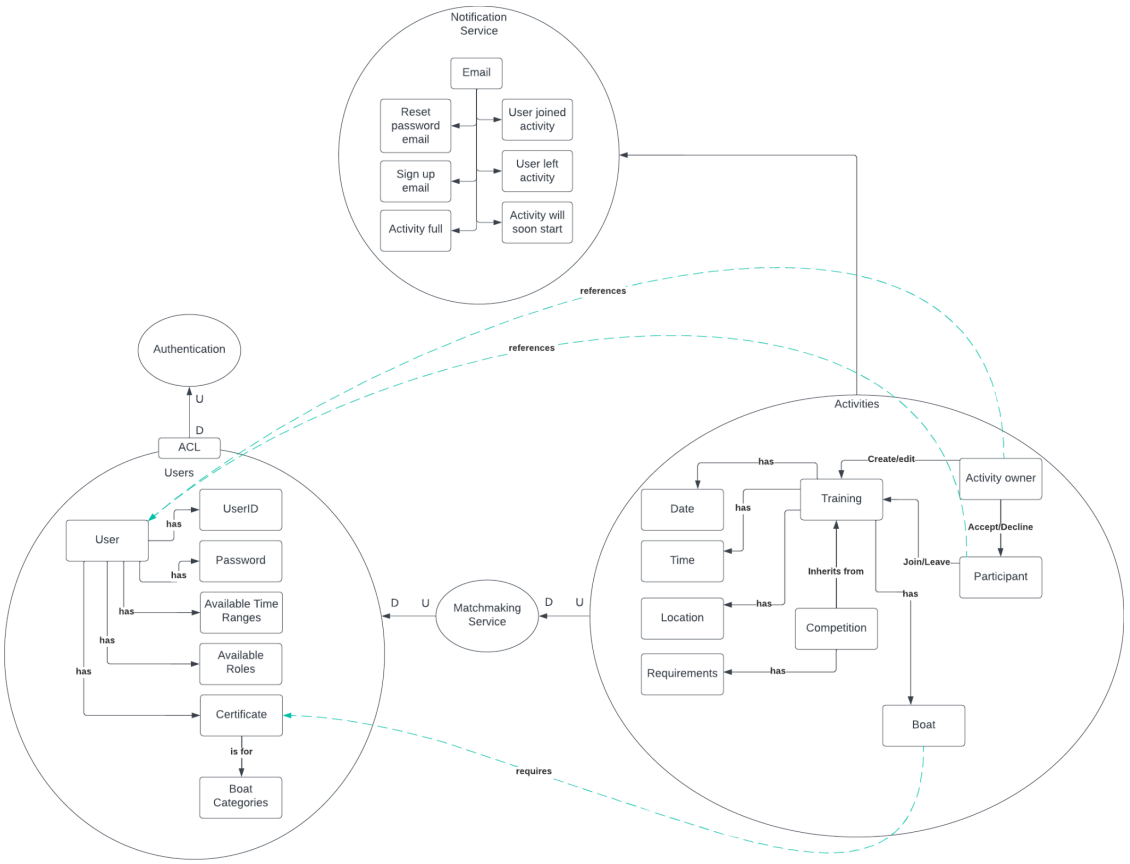

Fig 3 - The context map of the bounded contexts

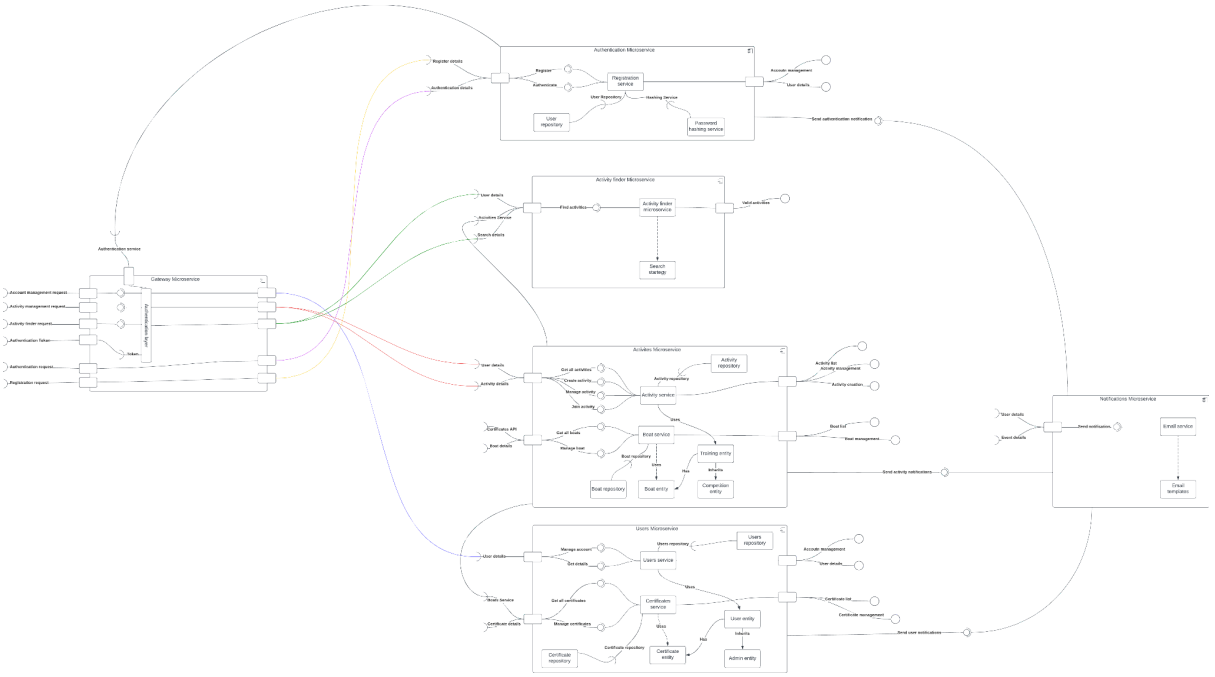

Fig 4 - The UML drawing of the microservices