DOMAIN-DRIVEN DESIGN

For the architecture, we identified 3 bounded contexts: Users, Reservations, and Authentication, which we mapped to microservices, as seen below. On the next page, a draft of the architecture is included.

<u>Users</u>

The Users microservice is designed in such a way that it takes care of both individual and team reservations at the same time. The way that is achieved is by treating ‘teams’ as **lists of users**, and so, we allow a ‘team’ to contain even one user.

The way teams are actually saved and can be accessed again is if someone decides to give a name to the specific team when booking the specific sports field. If that is not being done, that means that either the user is booking the facility individually or they don’t have the intention to book the sports field with the same people, and so those cases can be forgotten (to avoid saving every possible combination every member booked the court since the beginning of his subscription and thus saving memory).

This solution also provides a good answer to the requirement that team activities should contribute towards the total number of bookings that each individual member made since it becomes easier and cleaner in the implementation to work on a list of users every time than to work on a separate entity which then has to make a reference to and call each user.

<u>Reservations</u>

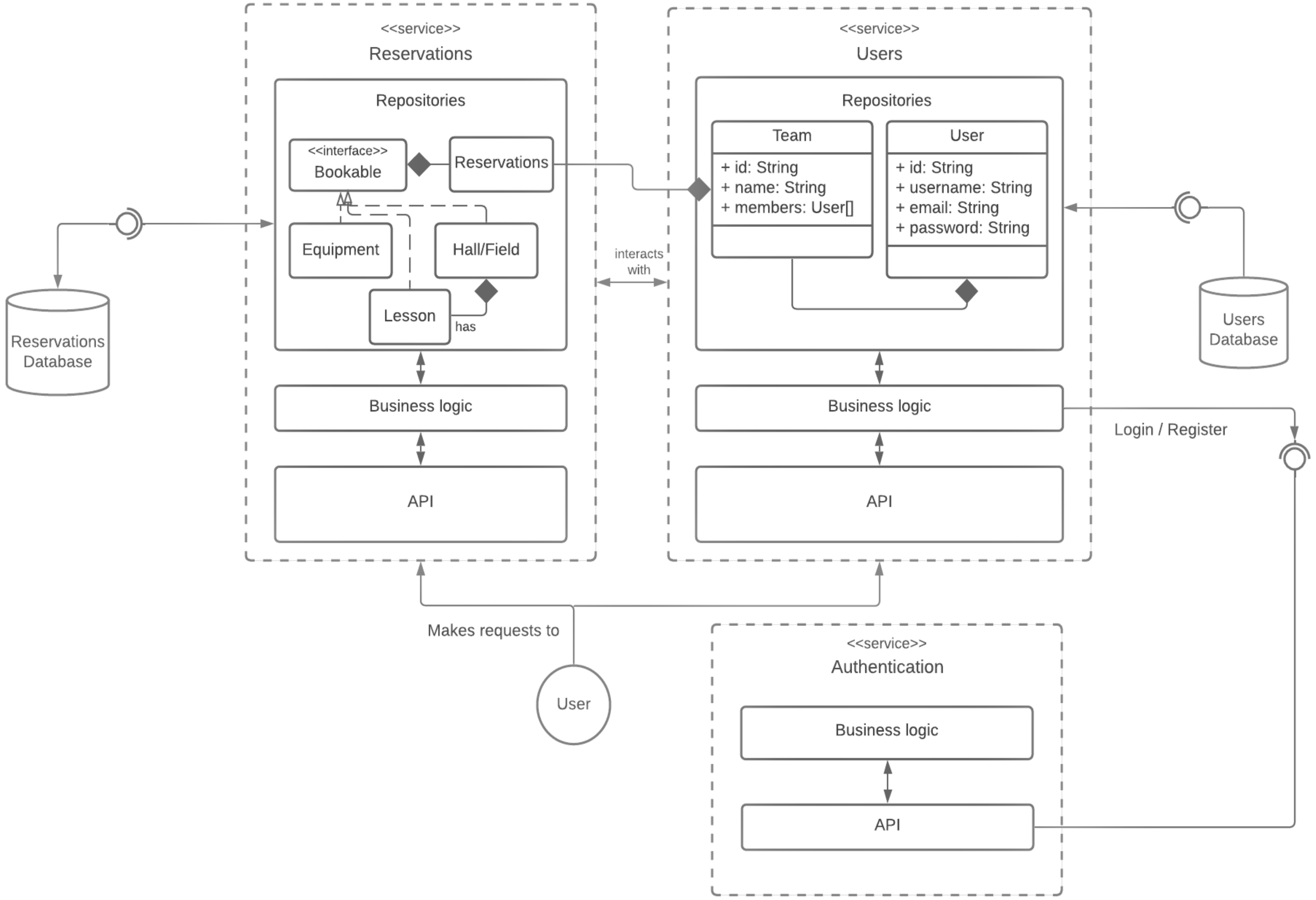
The reservations microservice is where everything related to sports fields, equipment, and booking them is dealt with. We will have a `Bookable` interface, all equipment and sports fields will inherit from which. Two separate classes will be created for Courts/Halls and Equipment respectively, as stated earlier. Lessons will also be bookable but they will have a hall attached to them, since they take place in those halls. Reservations then will be the thing linking users and equipment/halls/lessons - it will map a team (the reason behind that is explained in the previous microservice) to some bookable.
The user will then be able to make requests for booking different equipment, lessons, and halls, while passing a team as a parameter to the API.

<u>Authentication</u>

The authentication microservice is where we deal with the authentication of users who want to use our application. It will only give access to the user if the authentication is correct but it shall not keep track of what a user has ordered in the past or what subscription the user has. That is up to the class users.

Authentication will also take care of the database being safe for the users. In the sense that passwords will be protected. We plan on doing this by hashing the passwords in the registration process, and when logging in, comparing the hashes instead of plain passwords. This makes sure that if the database were to leak, the passwords would still be protected.

ARCHITECTURE - DRAFT



<<service>>
Reservations
Repositories
<<interface>>
Bookable
Reservations
Equipment
Hall/Field
Lesson
has
Business logic
API
Reservations Database
interacts with
<<service>>
Users
Repositories
Team
+ id: String
+ name: String
+ members: User[]
User
+ id: String
+ username: String
+ email: String
+ password: String
Business logic
API
Users Database
Login / Register
Makes requests to
User
<<service>>
Authentication
Business logic
API