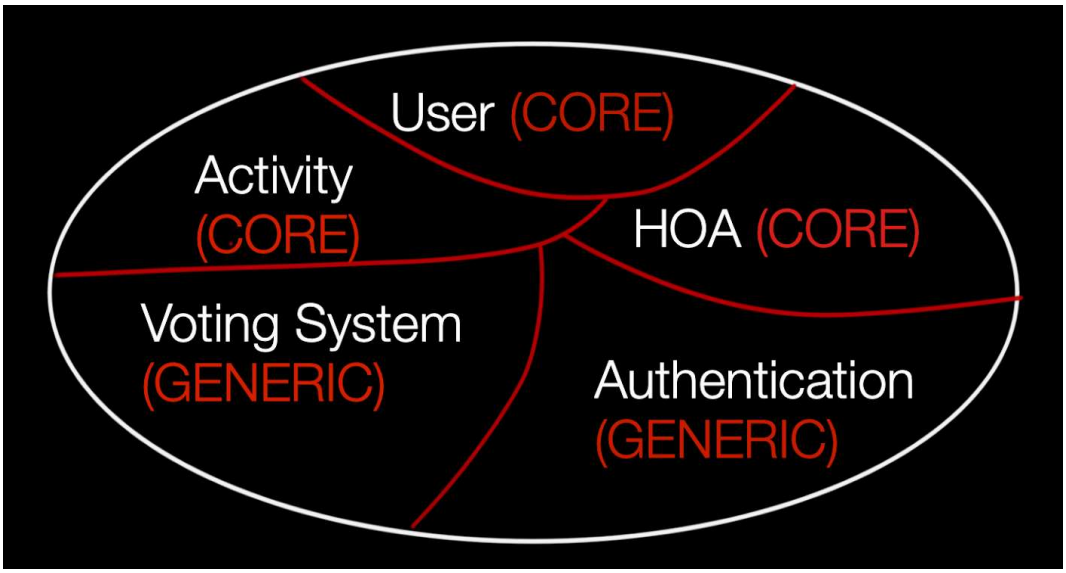
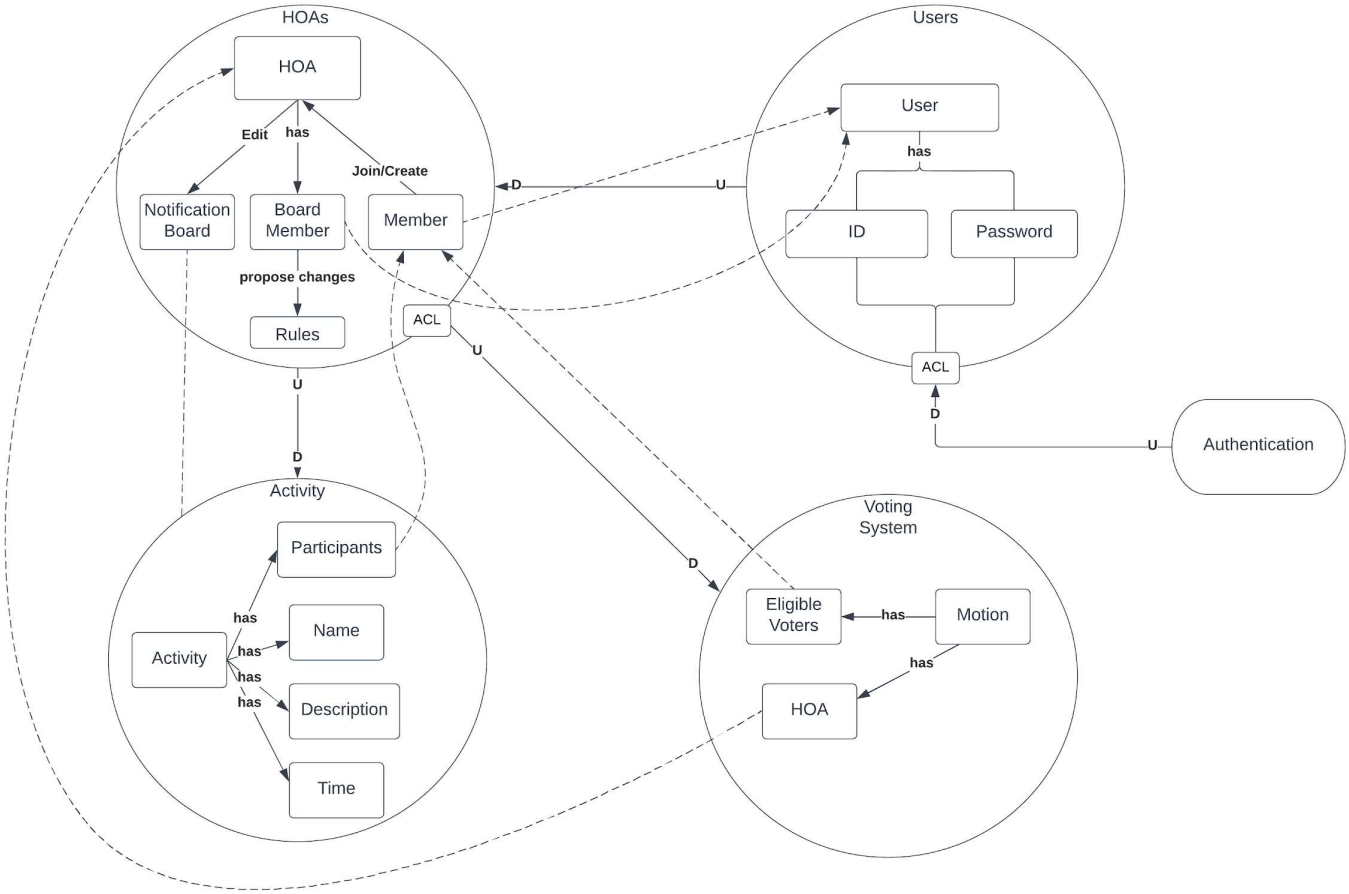
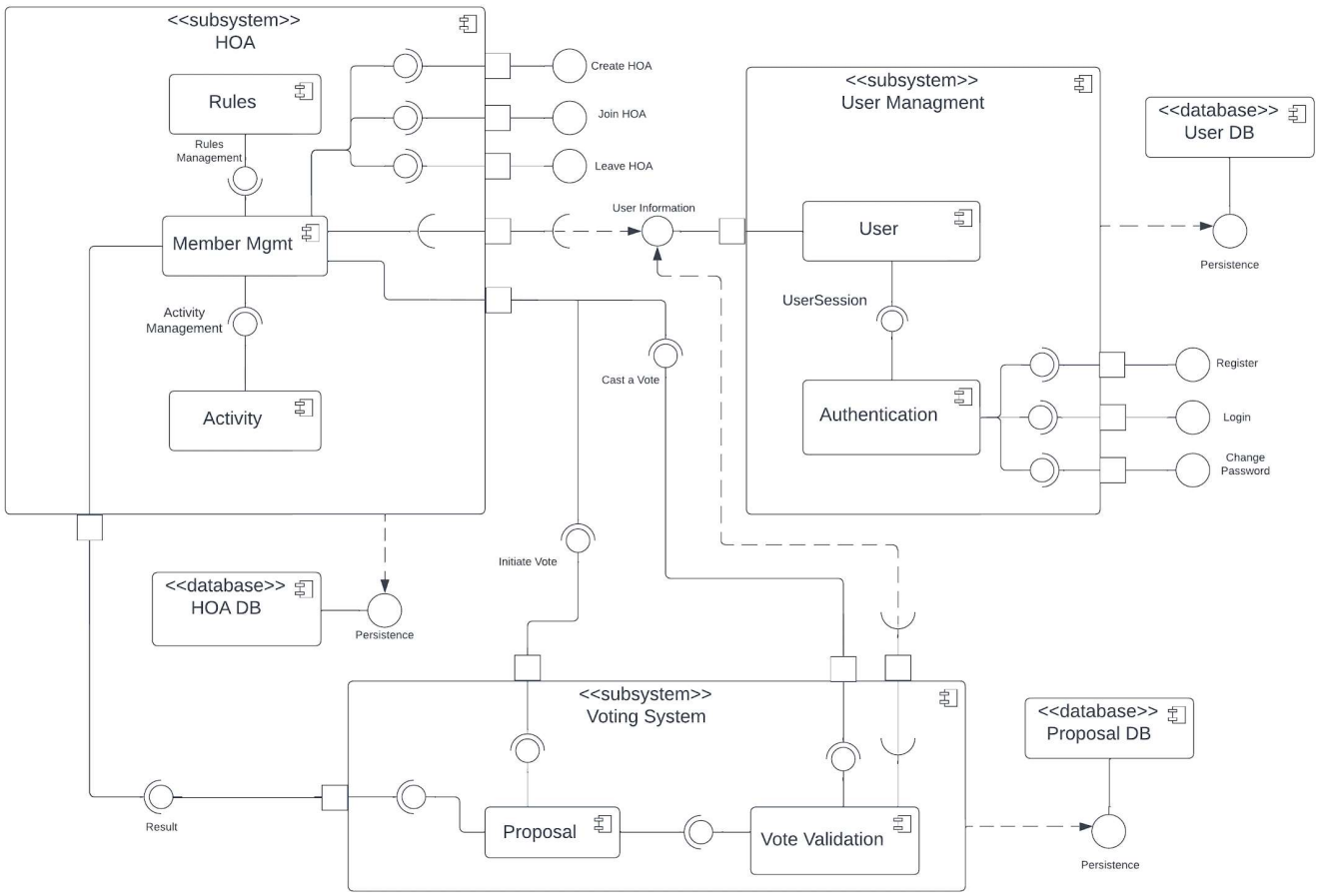
Group 22B - Assignment 1





Context Map



UML component diagram

Task 1

1. Bounded Context

We have identified 5 different bounded contexts from the given scenario using DDD (Domain-Driven Design). These domains are described as following

a) Users - The user is the central part of the whole project (and hence a *core domain*) and is the one who can Create or Join an HOA, Update its rules, Join its board and Create Activities. A user can have one of the following two subscriptions to a HOA: a Member or a Board Member.

i) Member - a member of an HOA is someone who follows the joined HOAs' rules and has the option to create and participate in Activities on the Notification Board posted by the HOAs joined by the given user. When it comes to a HOA board a member of an HOA can either vote for board members or if the users meets some requirements can apply for that board, go through a Voting process and if successful will become a

ii) Board Member - board members are the core part of an HOA. They are responsible for creating, updating and voting on rules

concerning the HOA they are a part of, which apply to all members of that HOA.

b) **Home Owners Associations (HOAs) -** Our whole application revolves around HOAs. That is why this is a *core domain*. A HOA is identified with a country, a city and the name of the HOAs. Users can join a HOA by providing their address. Every HOA has a board, which guides its requirements, and a notice board.

   To become a board member, each user can apply as long as they satisfy certain requirements. Board members can propose changes to the rules. Then a voting process takes place, in order to decide whether the changes are accepted or not.

   The notice board is used to publish activities that members of the HOA can attend or not, or to notify the members if there were any changes to the rules of the HOA.

c) **Activity -** The activities are an essential part of the HOAs. That is why this is a *core domain*. We decided to make them a separated entity since they require an independent set of operations. This includes that any members of a HOA can publish activities which other members can show interests. The activities are managed by the Notification Board. Each activity includes a name, a description, a time and participants (which are members of that HOA)

d) **Voting System -** We made this a separate bounded context that acts as a service to the HOA. That is why this is a *generic domain*. All that is needed to handle votes. Votes can be of various kinds: each member of an association can vote for the board candidates and each board member can vote on new rules for the association. We decided to separate the voting from the other contexts because it can be performed independently by a single component of the system, to avoid duplications and to improve security.
   Having a voting system separated from the rest also allows for further scalability: the system can be used to implement functionalities such as joining the activities or reporting members that don't follow the association's rules.
   Each vote will have a description of the proposal and a list of people that have the right to vote. The system will take care of applying the voting

rules (such as not counting abstainers).

e) **Authentication** - We will need a service to authenticate users (as it is a service, it is a *generic domain*) and verify user credentials. This will be done by a distinct microservice. We will be using spring security for this purpose, it will check the validity of the username and password the user gives to it, and

2. Microservices

We decided to implement the system as a collection of microservices since this architecture offers several major advantages over others. Firstly, using microservices greatly improves scalability. Each service is relatively small and easily maintained, therefore they can be developed, tested and deployed independently of the other services in the system. This allows developers to scale specific components of the system separately rather than scaling the entire system at once. Furthermore, using microservices also improves team collaboration. Each group of developers can focus on a specific component, which creates higher efficiency and better communication within a development team.

For the low-level architectural design of each microservice, we decided to use Ports and Adapters architecture. Firstly, it helps isolate core business from the services it uses. These services can be changed or replaced without affecting the core behavior of the system. Secondly, using Ports and Adapters improves the system's testability. Using loosely coupled dependencies and Inversion of Control allows the core business to be tested independently of the external services.

At the beginning, we mapped each bounded context to one microservice, However, this approach creates unnecessary overheads for communication between different services. Therefore, we merged Activity into HOA microservice since the two would have been highly coupled and an Activity microservice wouldn't contain much functionality. We also merged Authentication into the User microservice since only the users need authentication and all the actions in the system are performed by users that need to have their credentials verified. This leaves us with 3 microservices, which are described as follows:

a) **User Microservice (also takes care of Authentication)**
The User Microservice is responsible for managing user credentials, which includes registration and authentication. The user service has a database that stores the user's full name, username and hashed password, the membership information was agreed to be stored in HOA instead due to the fact that one member can have multiple addresses for different HOAs. All other

microservices will be fetching jwt tokens from here to authenticate sessions. There are endpoints to register, authenticate and fetch the full name of a user.

b) **HOA Microservice**

The HOA Microservice is responsible for HOA management and for member management within the HOA. It has 4 repositories - one that stores the HOAs and their details (id, name, country, city, etc.), one that manages the activities within an HOA, one that stores the Participation of users in activities (activityId and userId) and one for managing the user-HOA relationship (we call it Membership). The user database consists of an id of the user and every unique user is mapped to an id of the HOA that they are part of.

The Activity entity stores the activityId, hoaId, description and the date the activity takes place. We also have another entity, Participation, that stores the details about which user participates in which activities. This part of the microservice provides API endpoints to add activities, to allow users to participate in activities, and to let users pull out of activities if they want to.

c) **Voting Microservice**

The Voting Microservice manages all the voting processes, both for the board member elections and for the rule changes. The service allows to initiate a voting procedure and to set a deadline for closing the voting process. It allows to edit the proposal, adding new options to vote on and to begin the voting procedure, as well as allowing the users to add their vote on a proposal. It is also responsible to validate the votes, checking that all requirements are satisfied. After the deadline passes, the result of the vote is stored in the database. Each proposal stores all the votes each proposal received, so that it is possible to retrieve the whole history of elections and rule changes.