Assignment 1, Part 1 - Group 22a

Task 1: Software architecture

Introduction

This project will make use of microservices as the general architecture. There are many reasons for this choice, besides the fact that it is a strong requirement for this course.
We saw many positives in this architecture that we are certain our project will flourish from.

First of all, this decision enables every one of us to become more autonomous, as each team member will work on an assigned microservice. Given how microservices will be loosely coupled, working on one microservice should not directly impact the others. This makes it much easier to split our tasks and start working on a smaller piece of the puzzle at first, rather than immediately jumping into a deep water.

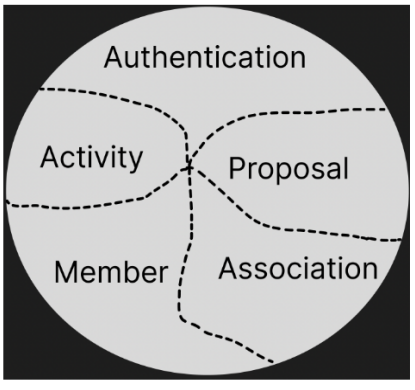
Microservices enable us to have extremely testable code, because the logic within the project can be tested without taking into account any external services. Writing highly testable code is desirable as it will ensure that our code is of high quality.

Another reason for choosing microservices is that it makes our overall project much more maintainable and allows us to easily replace services, as long as our adapters remain suitable. This will make our architecture more maintainable, scalable and isolated.

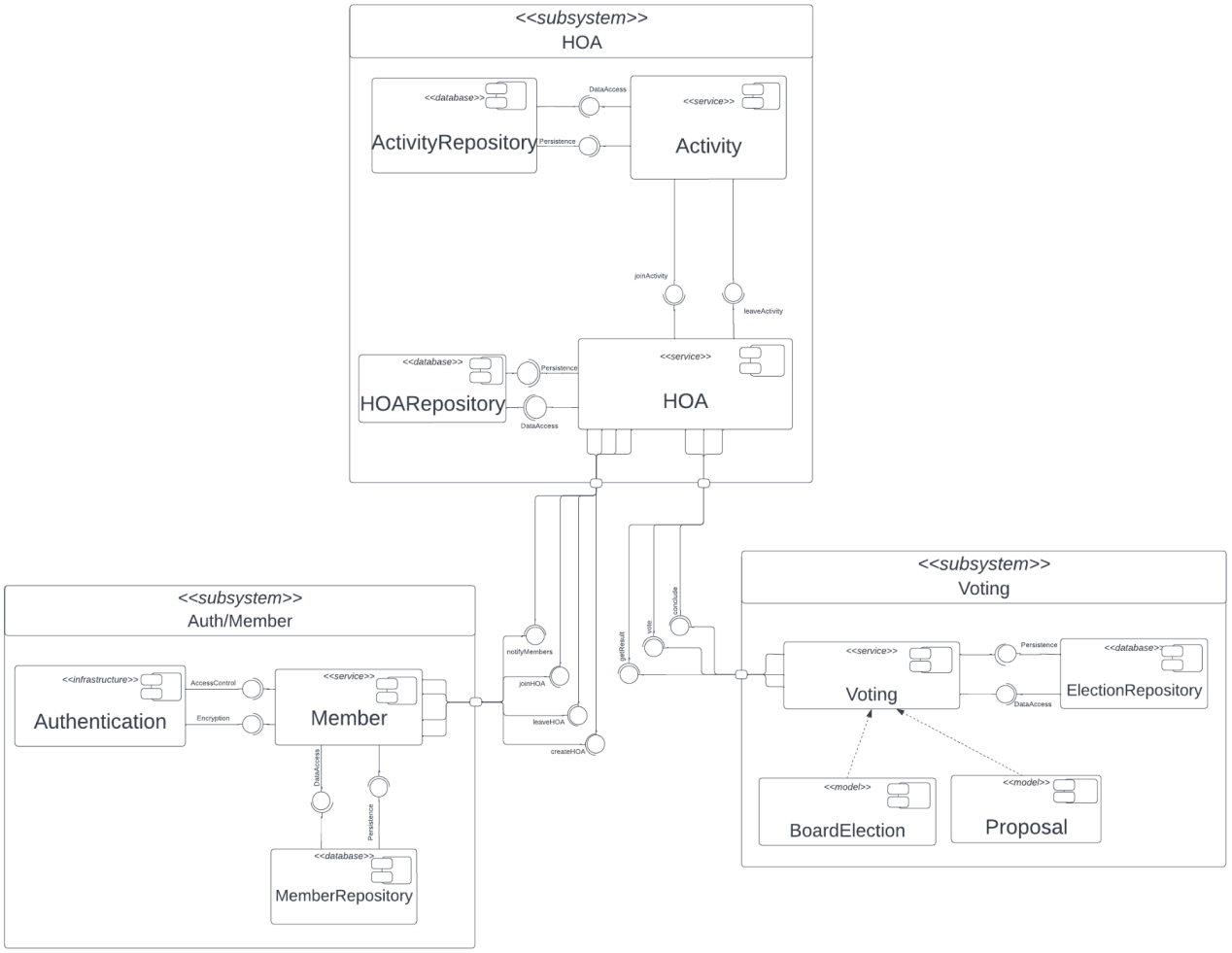
Bounded context & mapping to microservices

Our bounded context is composed of five members: member, association, authentication, proposal and activity. The first two represent the core domain, as we have identified those as the most important part of the system. The members have the possibility of creating and/or joining an association. The association is made up of members, and a few of the members make up the board. Authentication is a part of the generic domain, since it is not designed specifically for our use case. Its implementation will ensure that the members' details are securely stored and that it is impossible to tamper with them.
Lastly, we have two supporting domains. The activity domain is responsible for events that can be proposed by any member within an association, and proposals can be made by current board members in order to change the current ruleset.

- Authentication - Generic
- Member - Core
- Association - Core
- Activity - Support
- Proposal - Support



Mapping the bounded contexts to microservices was not a simple task. We have decided to group the authentication with member-related logic into one microservice, the second one manages the associations, and one microservice handles voting within associations, both the board member elections and proposal voting. A visualisation of this can be seen below, in the form of a UML component diagram. The diagram is followed by explanations of our reasoning for the design choices.



Details of microservices

The Member And Authentication Microservice

This microservice is responsible for processing all the user-related queries. It will be primarily used to:

- Process the registration and authentication of members
- Resolve membership-related queries, like joining and creating HOAs
- Store member data, including their addresses, metadata and HOA memberships

We have chosen to merge the member and authentication contexts due to their coupling, as this microservice acts as the gateway to each HOA a member is a part of, a user first needs to authenticate and access their member data before interacting with

any other part of the system. Our decision is also reinforced by the fact that the member domain is a core domain and the authentication is a generic one.

This microservice will have a public API to allow a user to access their data, including their association memberships. This service is also connected to the HOA microservice, since creating, joining and leaving an association requires persisting that information on the member side. In a typical user session, the user will first access this microservice to authenticate and get their relevant memberships, after which they will be able to directly interact with the HOA microservice for further business.

The HOA microservice

This microservice is supposed to contain the whole business logic concerning the association. It also acts as a bridge between elections and members. The following list contains the most important functionality that this microservice offers:

- It will process member requests, such as join requests or leave requests.
- It will be responsible for initialising a board election, retrieving the results and updating it accordingly. This is done with the collaboration with the Voting microservice.
- This microservice will contain the public notice board logic along with the activities associated with it.
- It will store all the association data, including their locations, members and board

The association microservice is the middle of our architecture, since the associations are the centrepiece of the system. It contains all the data of each HOA and interacts with most of the requests a user can make, apart from authentication and registration. Interacting with proposals, board elections and activities is done through this service, and since it contains all the memberships for that association any notification system will be implemented through it. In addition to the HOA data, it also contains the Activity logic. Activities have been merged with this microservice due to their relatively small size and tight relationship to the associations they belong to. They did not belong in the Voting microservice since they lack any voting mechanism, only storing the participants is required.

The Voting Microservice

This microservice is mainly responsible for dealing with various types of elections. Its duties are:

- Manage the election of a new board for a certain HOA
- Process the proposal of a new requirement by board members
- Storing ongoing and past proposals and elections

We decided to merge the proposal logic with board election, due to the commonalities between them. The voting system will be capable of creating and resolving elections and proposals depending on a set conditions, as well as persisting them. It will not have a publicly exposed API, as all the queries related to either elections or proposals will go through the relevant association. There will also be no outgoing calls from this service, as it only acts as an endpoint for the HOA service.

Design choices

In this section, we are going to list a few design choices we made along the way.

Why does a member not automatically join an HOA they create?

Starting off, when a member creates an association, it does not automatically join it. In this sense, creating and joining/leaving an HOA are completely separate. We chose this, as we thought that it should be perfectly allowable for a person to create an HOA without joining it, for instance a member from the municipality.

How will the relation between HOA and members be modelled?

The relationship between a member and HOA is many-to-many, in the sense that a member can join multiple HOAs, and HOAs host multiple members. By creating the new artificial attribute membership, we got rid of this, and exchanged it for two one-to-many relationships. This enables only the persistence of memberships, which is a lot more flexible due to the fact that now a member can join and leave any association any number of times. The HOA microservice does not have to explicitly store its members.

How do membership components model the relation of members to HOA?

Each membership models an uninterrupted period of time a person has been a part of the association. If a person leaves the HOA and then joins again, a new membership is created for them, to allow for timekeeping. Similarly, when a member joins the board, a new membership is generated for them to store the time they have served in the board. In essence, we store membership of two specific types, memberships for regular members and memberships for board members. As explained before, this allows us to do easy timekeeping to check for fulfilment of constraints. These constraints are specified in the requirements document, one example would be that a regular member can become a board member only if they have spent at least 3 years in that association.

How is time spent in an association calculated?

For each period of time a person has been a member of a given association, their join time and the duration of their membership is stored. Aggregation and summation of those components allows for calculating their total membership time as well as their board time.

How are votes stored and why?

As one of the requirements stated, a member should have the possibility of changing their vote. Using a hashmap enables the user to do just that, instead of, for instance, counting.

How does the election process work?

The HOA initialises an election. From then onwards, members can join during this time. After some time, no one is able to join the election anymore, and members can start voting. In the end, the HOA just calls for the results of the voting process.