Task 1: Software architecture

Our project addresses the Home Owners Association scenario. Users should be able to register in the system and interact with associations which have their own functionalities and logic. For our architecture, we are using microservices. First, we identified several bounded contexts based on the requirements:

Authentication (Generic domain) - This bounded context is necessary for verifying the identity of the user. It's needed for every single request to ensure both security and the proper functioning of the entire system.
Association (Core domain) - This bounded context represents members and board members of the association (which is a subset of members).
Users (Core domain) - This represents user related information such as credentials and personals.
Rules (Core domain) - This represents the rules that an association may contain. These rules can be changed and they tie into the report system too.
Notice Board (Core domain) - This holds all of the active activities in an association. Its logic is purely dependent on the activities which it displays.
History (Supporting domain) - This contains the entire history of the association including election results and rule voting results.
Report (Supporting domain) - This bounded context represents the functionality to report other users for breaking rules set by the association's council through rule-voting.
Voting (Supporting domain) - This bounded context represents all ongoing election and rule-voting events. It also contains the logic of such events, such as candidate registration, election voting, rule proposal, and rule voting.

Before we dive into how the bounded contexts are mapped into the microservices we would like to clarify the diagram, which can be found in the appendix, and the relations between the contexts. Almost every component can be in its own bounded context except for the activity component and the member components. The activity component must be within the notice board context because a notice board stores and displays only activities and the activities cannot be accessed on their own by members without being displayed by the notice board. As for the member components, an association cannot function without members therefore we must have them within its context. There is a partnership relation between the association and the voting context. There's going to be a degree of coordination between us in the development of both. That is because, functionality wise, the two of them are intertwined. Members can vote on council elections and this will then update the council when the election is over. Council members can initiate or participate in rule votes and when these votes finish they update the rules of the association. This means that the association and the voting are mutually dependent on each other for their services. Each member is a user and users can participate in the activities of their association. These relations are represented in the diagram through dotted lines.

We will now look into how the microservices are mapped. In theory you could make a microservice for each of the bounded contexts. However we decided that it would be best to combine most of the separate components with the association microservice. This is the case for the history, rules, report and notice board. We decided to take this approach because the microservices aren't that complex. The logic implemented by each of them is relatively simple. Furthermore, considering the scale and scope of the project, it's more feasible to combine them into the association microservice. That being said, they could not be combined into each other either as they're not too similar in scope or logic outside of their shared relation with the association. Therefore, in the end, we have three microservices. The association service, the authentication service, and the voting service.

Before a user can utilise the association and voting services they need to be authenticated. Once they are verified they can communicate with the association service through API calls. The voting microservice can only be accessed through the association microservice. We opted for this approach as an election or rule vote can only be held under an association and it eliminates the need for accessing the association microservice from the side of the voting service to retrieve data needed from the association. This way the association service can send the necessary data to the voting service in one go. The association microservice and the voting microservice also communicate through API calls.

Before discussing our UML component diagram we would like to show what syntax we are using. This is due to the fact that a lot of people use slightly different syntax.
In the Unified Modeling Language (UML)[1], there are multiple different types of diagrams we can make. As already specified, we are making a component diagram[2]. We are using UML version 2.5.1[3], as specified by the Object Management Group (omg). I would like to point out a few specific points from the UML specification[4].

1. Figure 10.8 and figure 10.10 from section 10.4.5
   This section shows how to model interfaces between components
2. Figure 11.11 shows how to model the implementation of an interface between components

With having discussed the syntax you can now find our UML component diagram in the appendix. On the left side of our UML component diagram we can see the public API's. The authentication interface is also a public API. Each big component is a microservice, the other nested boxes are components of those microservices.

There are three microservices: Authentication, Association and Voting.
The Authentication microservice does what its name suggests. It ensures that users only have access to their personal data and that services are accessed securely. The microservice maintains a database of all users. Users can register, authenticate or change their password. The last of these hasn't been fully implemented at the time of writing though.
The Association microservice deals with the internal logic for the HOA's operations. It keeps track of each association's members, its board (council), its notice board and its history. It also provides functionality for posting activities, joining / creating an association, registering as a

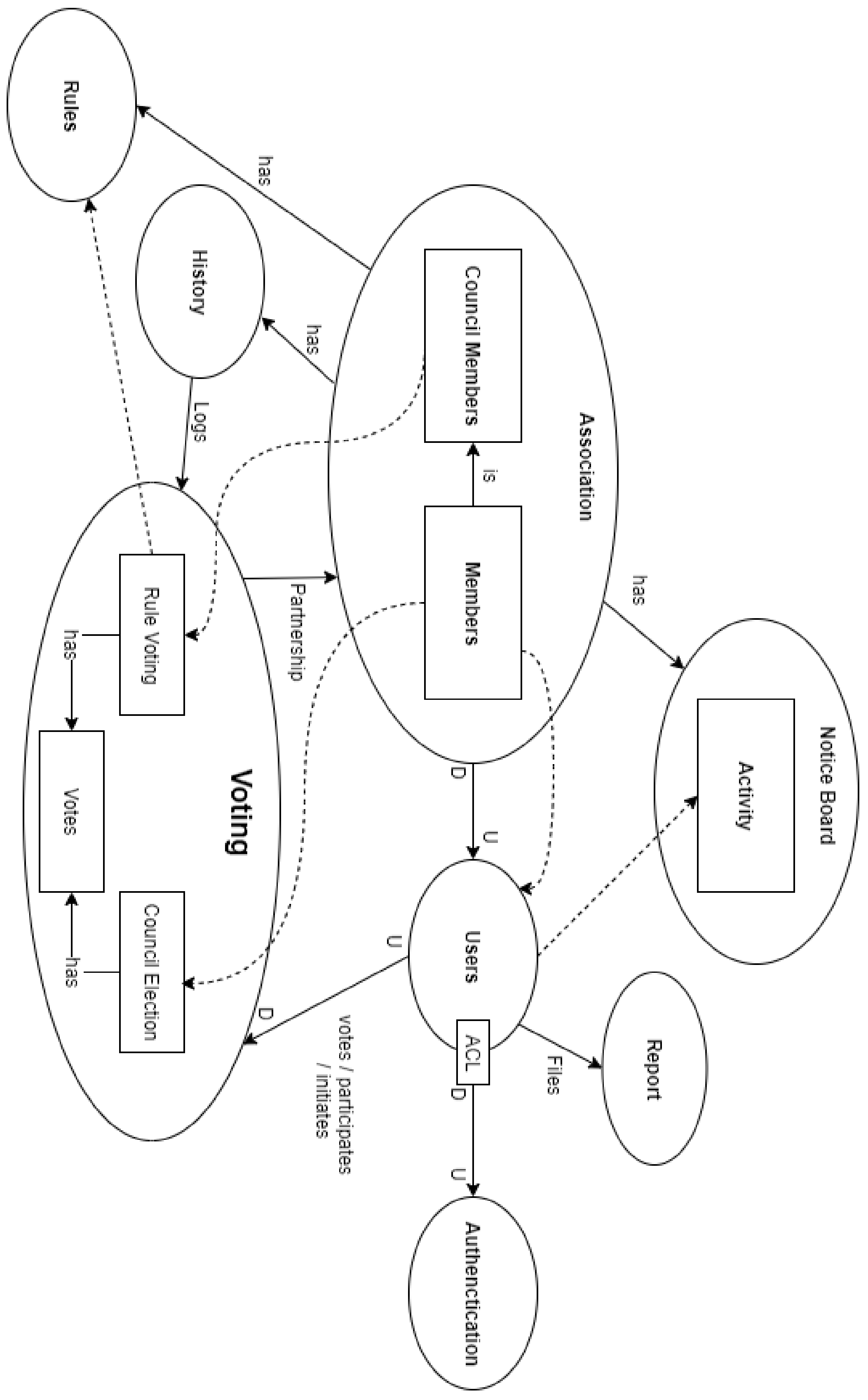
[1] "Unified Modeling Language - Wikipedia." https://en.wikipedia.org/wiki/Unified_Modeling_Language. Accessed 11 Dec. 2022.
[2] "Component diagram - Wikipedia." https://en.wikipedia.org/wiki/Component_diagram. Accessed 11 Dec. 2022.
[3] "About the Unified Modeling Language Specification Version 2.5.1." https://www.omg.org/spec/UML/2.5.1/About-UML. Accessed 11 Dec. 2022.
[4] "OMG Unified Modeling Language TM (OMG UML)." 9 Oct. 2015, https://www.omg.org/spec/UML/2.5/PDF. Accessed 11 Dec. 2022.

candidate for an election, reporting a member for breaking the rules, as well as initiating a rule vote for the council or rule changes/additions.
The Voting microservice handles council elections and rule votes. In the case of elections for the council, it keeps track of the list of candidates and how many votes each of them has. In the case of rule votes, it keeps track of the proposal (the rule) and whether it's being edited or added. At the end of an election or rule vote, all the votes are tallied and the outcome is forwarded to the Association microservice which then logs what happened.

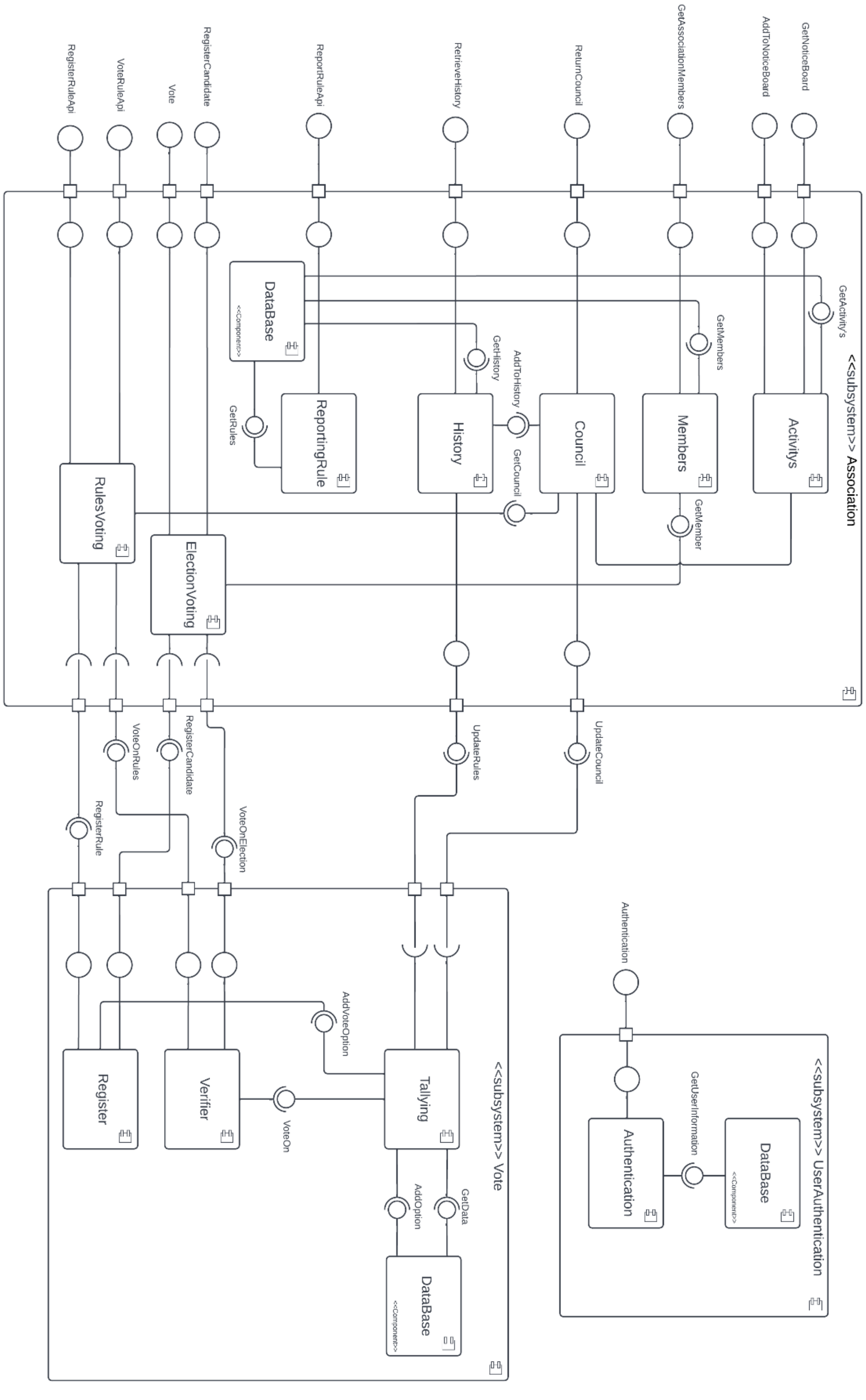
While we were required to use the microservices architecture, there are multiple ways to implement it in our given scenario. In the following lines, our decisions in the process will be explained, as well as why they were preferable to some alternatives.
The decision to make Authentication a separate microservice was pretty straight-forward. The user authenticates once when logging in and receives a token. Then, the token is used for the rest of the session. The separation also made sense from a security perspective.
Then, we decided to include the Association microservice. Since all of the actions a user performs in the application are tied to an association, this microservice will serve as a gateway to other microservices, forwarding the user's request together with all of the necessary information about the association. For that reason, this microservice is also where all of the association data, along with the history of elections, will be stored. This microservice is also where reporting will take place, since the referenced rules as well as the reports are stored here, and there is no difficult interaction present in this action, for example as opposed to elections. The activities from the notice board will also be created and stored here, since they are tied to the association.
Last but not least, we have the Voting microservice. In the board elections, as well as the rule passing process, a voting procedure takes place. We can use this similarity in voter verification, vote casting, and vote tallying, to create an interface which will be used in both voting procedures. If we decided to split these actions instead, there would be a certain level of code duplication - which would hurt the development process, if a change would have to be made in both voting procedures. Additionally, it made sense to have the voting procedures somewhere separate as they required more work than other features of a HOA such as the notice board.

Appendix

You can find the bounded context map on the following diagram:



You can find the UML component diagram in the following picture:



<<subsystem>> Association
RegisterRuleApi
VoteRuleApi
Vote
RegisterCandidate
ReportRuleApi
RetrieveHistory
ReturnCouncil
GetAssociationMembers
AddToNoticeBoard
GetNoticeBoard
DataBase
<<Component>>
GetActivitys
GetMembers
GetHistory
AddToHistory
GetRules
ReportingRule
History
Council
Members
Activitys
GetCouncil
GetMember
RulesVoting
ElectionVoting
VoteOnRules
RegisterCandidate
RegisterRule
VoteOnElection
UpdateRules
UpdateCouncil
<<subsystem>> Vote
Register
Verifier
Tallying
AddVoteOption
VoteOn
AddOption
GetData
DataBase
<<Component>>
Authentication
<<subsystem>> UserAuthentication
GetUserInformation
Authentication
DataBase
<<Component>>