# SEM Assignment 1 - Architecture Report

Group 18B

## 1 Bounded Contexts

The task is to create a Home Owner Association (HOAs for short) application, scalable for future expansion and optimal for current use. Speaking to clients, the group has identified six bounded contexts: security, board elections, HOA management, board management, public notice board, and requirements. Each of these is independent enough to be treated differently from others. As far as an architecture model, the team has decided on microservices due to their scalability, ease of delegation of implementation to other services, and simplicity of future expansion with new features.

## 2 Mapping of Microservices

The six bounded contexts of the application translate into four microservices with the aim of achieving high cohesion and low coupling.

- The Gateway/Security Microservice was chosen to do the routing and authentication of each user, handling the security bounded context. It deserves to be separate from the others for the technical purpose of direct easy access to different parts of the application, as well as handling account creation easily. The team chose not to separate the security from the gateway since when drawing out the architecture in UML, it ended up being the only service that interacted with it. The user will interact only with the Gateway microservice directly, and it will propagate the requests to the internal microservices. It will be responsible for presenting them with all of the information they need to interact with application.
- The Voting Microservice was chosen to handle the board elections and a part of the requirements bounded context (the voting for modification of / creation of new requirements). The team decided for it to be separate from the others for the difference it has from other features and the possibility of future reuse. Elections are not the primary purpose of a Home Owner Association, but a means to an end of electing a board and adjusting requirements. They will be rare enough so that no network overhead will be felt while not losing any functionality. Additionally, since every type of vote is different, the microservice can easily be extended to host other types of elections, possibly for other apps.
- The HOA Microservice has the purpose of handling all of the internal maintenance of home-owner associations in general, addressing the HOA management and public notice board bounded contexts. This will include association creation, ensuring that a (suitable) board exists, calling elections, receiving results, verifying membership and authorization, and handling different public notice boards per association. It is a separate microservice due to the complicated conditions for an HOA's board, member requirements, and conditions for different types of elections. Their handling in a separate microservice will make the implementation of

all others far easier since they would not be concerned with eligibility when to be called, or any other homeowner association abstraction. The microservice also provides API endpoints for creating and retrieving activities on/from public notice boards.

- The Requirements Microservice is responsible for handling all the logic that the requirements bounded context needs. This includes keeping track of rules that are in place for each HOA, as their main purpose is to have a structured and well-kept neighborhood through these restrictions and requirements. These rules also need to be put in place somehow. The way that is handled is through the board of the HOA, who propose and then vote for any additions and changes to them. Members are then notified that they may not be fully aware of the things they need to do. The enforcement of the requirements is done through the actions of members: if someone is not complying, they will be reported by the others to the board, who then can decide on an appropriate action to take, whether that is kicking them out of the neighborhood or just warning them. These reports are anonymous so that no biases or conflicts between neighbors for reporting each other remain. The team decided to separate it from the other contexts because the functionality can be reused for other apps not centered around HOAs.

## 3 Considerations for other structures

In the first draft for the architecture, the Public Notice Board bounded context was handled by a separate microservice of the same name. At first, the team thought that the functionality could be separated well enough that it deserved its own microservice. It also would technically be reusable for other applications. However, when dealing with the authorization of users (i.e. is the user a member of the HOA which is associated with the Public Notice Board in question?), the team ran into difficulties. In the architecture, answering these authorization-related questions is the responsibility of the HOA microservice. That means that every request arriving at the PNB microservice would need to be checked with the HOA microservice first. This is very high coupling.

Another thing taken into consideration was the fact that like this, the PNB microservice would have just been a simple interface for accessing the database storing the activities. Since each activity is associated with exactly one HOA, remote foreign keys would need to be used. This leads to other difficulties like handling cascading for deletion, without the existence of the separate microservice adding any major benefits.

Considering the aforementioned issues, the team decided to merge what we planned to be the PNB microservice into the HOA microservice. The bounded contexts do stay separated, and PNB functionality will have its own separate repository, service, and controller, it's just that we don't have to worry about remote authorization for activities anymore. The PNB controller will only respond to requests on the subdomain "/pnb" to further separate functionality.

Additionally, at first the team had the Gateway and Security as two separate microservices, and a decision was made to merge the two. The gateway would have been used to route the user to every part of the system and tie it all together, while the security would verify through credentials provided in the gateway that the user was indeed the correct one. The reason for the change was that in the UML diagrams of the architecture, it was discovered that the gateway was the only microservice communicating with security. To avoid overcomplicating the codebase and network overhead from communication, it was decided to merge the two together.

## 4 Microservices Structure and Connections

The project makes use of an interconnected web of microservices.

- All initial connections are routed to the Security/Gateway microservice, which allows for user creation and authentication. From this microservice, the user can be routed to several other microservices. This is done to distribute logic and reduce load as much as possible. The way security is ensured is by letting all other microservice only receive requests from IPs that they know (e.g requests from the microservices discussed in the architecture). That way, the input is sanitized for all microservices, and no malicious party can interfere with them once a user authenticates in the gateway. Additionally, to allow direct connection from the gateway to the Voting and Requirement microservices, and a better user experience, the gateway will incorporate a notifications system on login that polls the HOA microservice for relevant updates. This includes public notice board activities and elections (whether for board members or requirements) based on the current user's association memberships and privileges. Those notifications will allow the user to vote in an election directly from the gateway (since the voting endpoint will be known) and directly propose a new requirement in the case the user is a board member.
- A central HOA microservice communicates with all other microservices to keep track of members, activities, and vote results. The 'HOA Management' bounded context is contained within this microservice. It handles all of the internal logic of a homeowner association, such as how it is created, maintaining that its board exists, maintaining a public notice board, calling elections, enforcing the requirements of the homeowner association as well as persistence in the database for the proper functionality of all other microservices. It communicates bi-directionally with the Gateway to connect the user to the proper HOA and its functionalities in other microservices. There is also a connection to the Votes microservice, used for sending votes and receiving the results. Communication with the requirements microservice is bidirectional so that the reports make it to the board of the HOA. The HOA will also be the service that stores most of the information of the application and provides the tables necessary for the operation of other microservices, including information like HOA membership, board membership, report history, public notice board history, etc. All of the microservices will be polling the HOA whenever they need that information, particularly the Gateway on login to retrieve the information the given user would be interested in such as PNB, reports and elections.
- Access to the Requirements microservice is granted through the HOA microservice. A board member can propose deletion rules and creation requirements, so the microservice needs to communicate with the HOA microservice to verify whether a user is a board member. This is also a necessity for them to be able to place their vote, which is routed through the Voting microservice. Member reporting is also done here, and this microservice is the last point of that request. The board members see the reports placed through the HOA microservice.
- The Voting microservice is connected to the HOA and Requirements microservices, these are the ones that will use its logic. For elections, (after the initial handling by the Gateway) all of the communication happens between the HOA and the Voting microservices. For requirement proposals, after getting a request from the Requirements microservice and conducting the voting, the Voting microservice sends the results to the HOA microservice for persistence. The implementation inside of it is purely algorithmic since there is no need for the Voting Microservice to store any data. All of its data is provided from either the Requirements or

HOA microservices. This microservice store votes only temporarily so as to allow recovery after a server failure. At the end of each vote, this temporary data is evicted after the results are stored by the HOA.

## 5 Example request-response chains for various scenarios

To conclude our architecture report, we would like to provide the reader with a couple examples of inter-microservice communication, in the context of user requests. Since we're using REST API (and thus, request-response cycles) for all of our microservices, the path of the responses will be the same as the path of the requests.

Creating an account / Logging in

| | |
|---|---|
| Gateway/Security | Everything happens in this microservice. |

Creating/joining an HOA

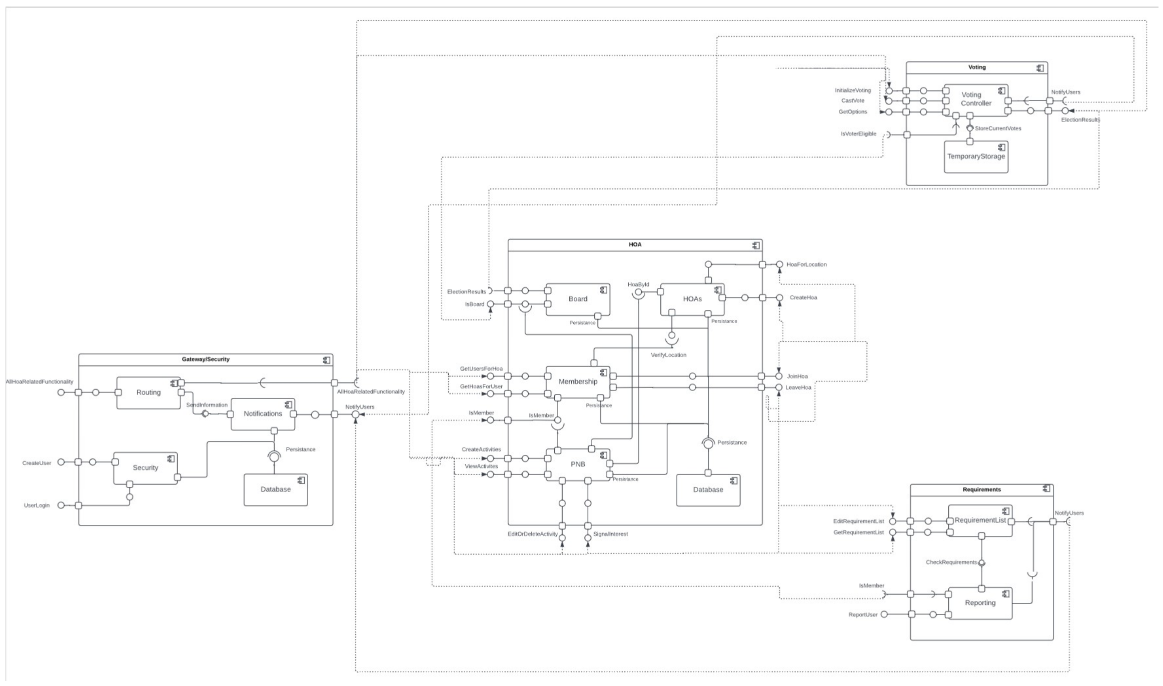
| | |
|---|---|
| Gateway/Security | Authentication and initial routing. |
| HOA | Various domain level checks, persistence. |
| Gateway/Security | Displaying success/failure. |

Starting an election for the board

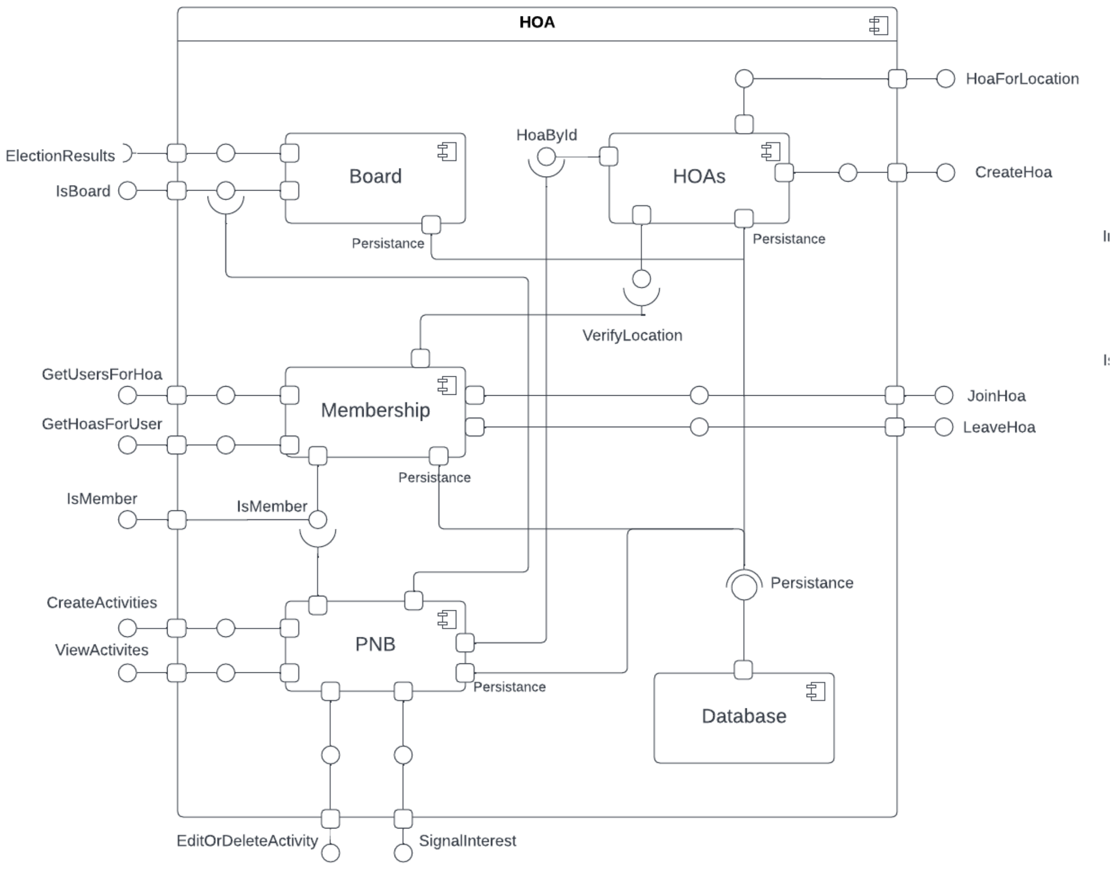
| | |
|---|---|
| Gateway/Security | User is authenticated and the request is forwarded to the Voting microservice. |
| Voting | Makes a request to the HOA microservice for authorization. |
| HOA | Responds with authorization status (checks whether have permission to start a vote). |
| Voting | If authorized, instantiate an election and make a request to HOA for a list of people to be notified. |
| HOA | Responds with a list of members to notify. |
| Voting | Forward the response back to Gateway/Security. |
| Gateway/Security | Sends final response to the user and notifies members about the election the next time they log in. |

Reporting another user

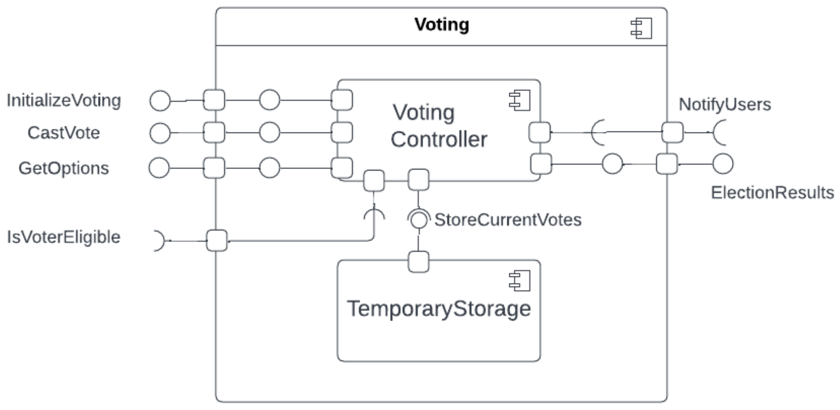
| | |
|---|---|
| Gateway/Security | Authentication and initial routing. |
| Requirements | Referencing the requirement violated by the other user. |
| HOA | Domain-level checks. |
| Requirements | Forwarding response (success/failure). |
| Gateway/Security | Display success/failure. The next time a board member of the HOA checks their notifications, they will be notified of the report. |



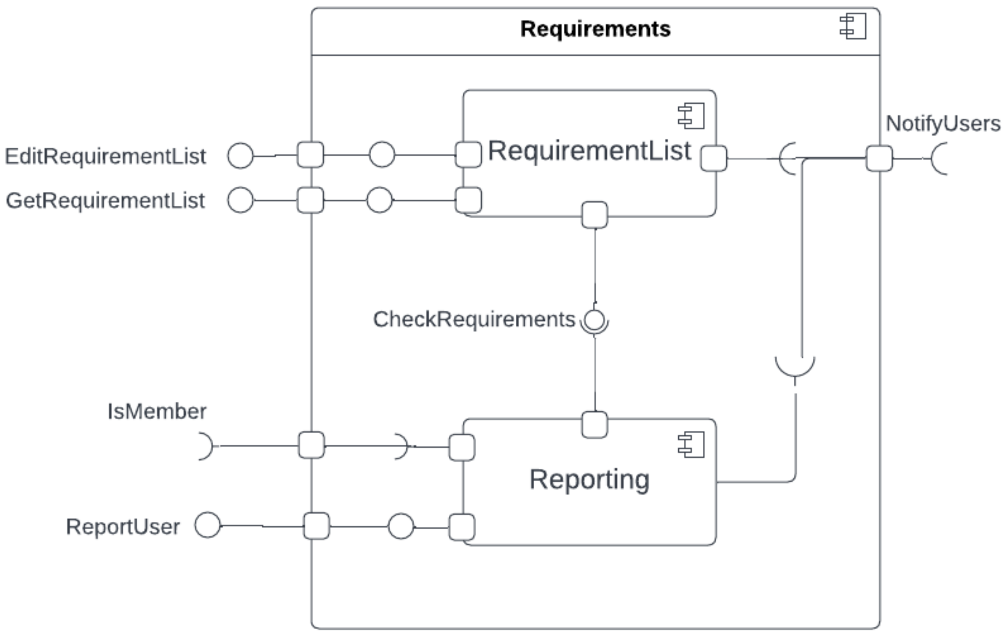
UML Diagram - Architecture



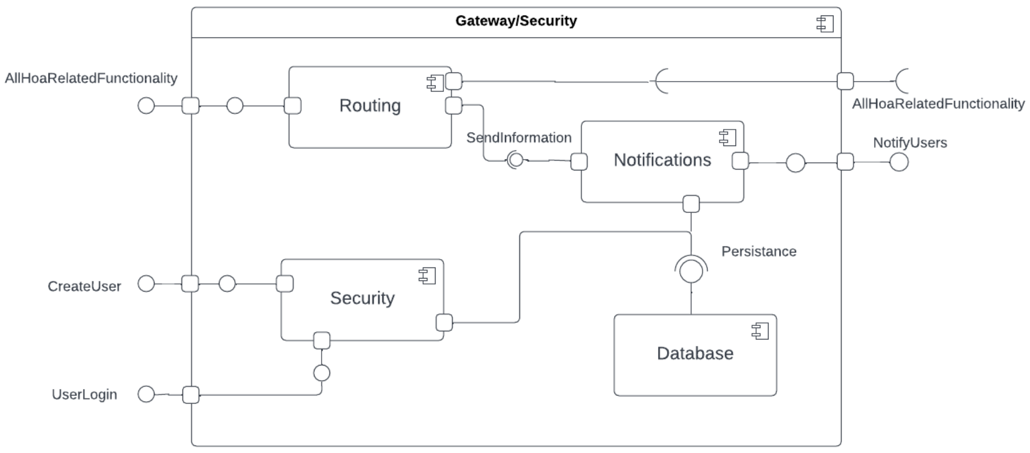
UML Diagram - HOA Microservice



UML Diagram - Voting Microservice



UML Diagram - Requirements Microservice



UML Diagram - Gateway/Security Microservice