Software Engineering Methods

Report - Scenario Homeowner's Association

Student name	Student number
Rafael Petouris	4776968
Jelt Jongsma	5496594
Vladimir Petkov	5447194
Bram Snelten	5519365
Alex Brown	5171709

Figure 1: Image of a house model titled Homeowner's Association [1].

CSE2115
Group 19a

Delft University of Technology, The Netherlands

27 November 2022

Contents

1. Bounded Contexts

1.1 Domain-Driven Design

Using domain-driven design we found the following bounded contexts:

- HOA
- User

We found these contexts by looking at our requirements and then determining the keywords. The keywords that were often used together were grouped and became the bounded contexts. In this case we realized that the functionality for security and user accounts was closely coupled and so needed to be within the same bounded context.

The HOA is our core domain and the User context is generic since the ability to have accounts and log-in is not specific to our application. The context map with these subcontexts is shown in figure 1.1. The HOA is downstream from the User context, since the HOA is influenced by how many users there are eg the HOA is dependent on the User context.

Next we'll explain how we mapped the contexts into microservices. The User context directly correspond to one User microservice. The HOA however is too big to be one microservice therefor we split the ability to create and manage activities from the HOA. The added benefit of splitting the functionality, aside from better scalability, is that the activity service can be reused for a different application or be used if additional organisations/features are added. We decided not to split the HOA further, e.g. introducing a seperate micro service for voting, because then very little functionality would be left in the HOA micro service and the voting service would be closely linked. This split we finally decided on results in the User, HOA and activities microservices. All microservices have their own database interface component. The entire UML component diagram can be seen in figure 1.2

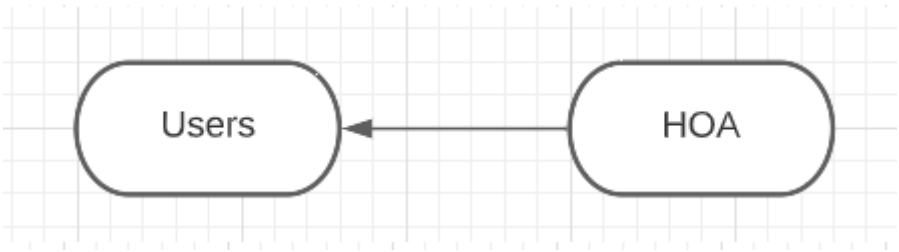

Figure 1.1: Bounded Context Map

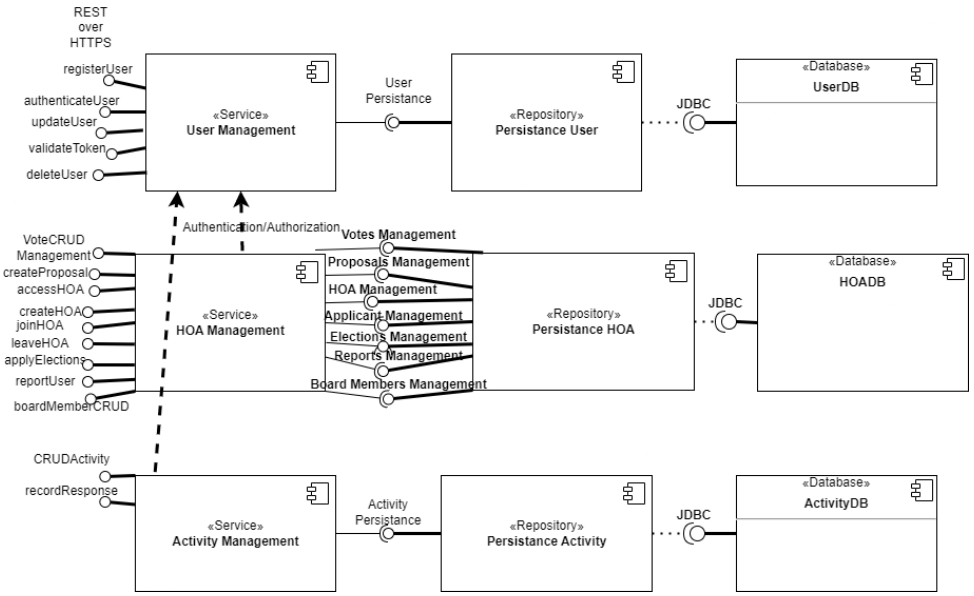

Figure 1.2: Component Diagram for the implementation of the Homeowner's Association

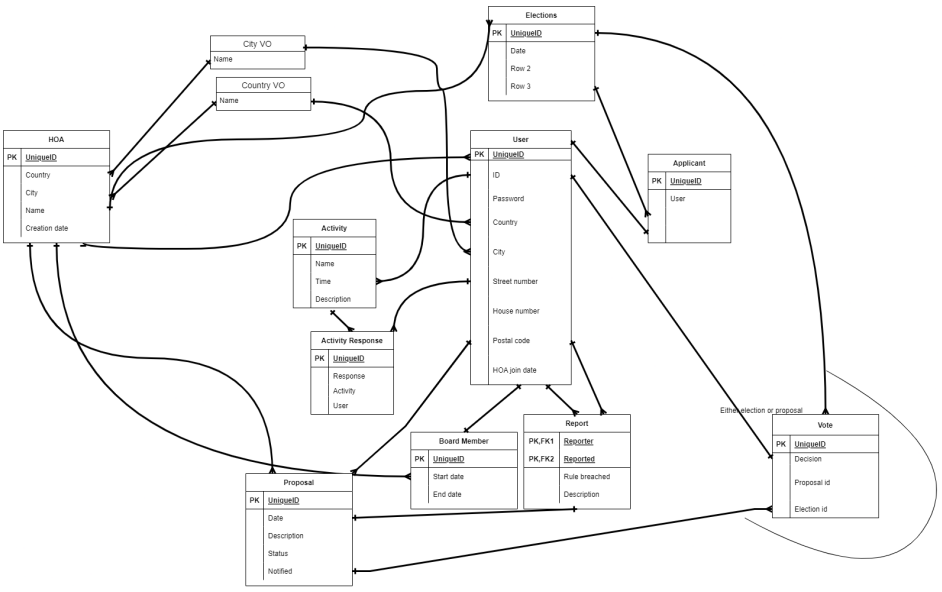

Figure 1.3: Relational schema of the Homeowner's Association

1.1.1 Users

The User microservice is responsible for authentication and security. It allows users to log-in to the system and edit/add their personal information. If a user is logged in and sends a request to any of the other microservices, Users sends the personal authentication token to the microservice and then the user is authenticated. The User database stores the accounts of all of the users including those without HOA.

1.1.2 HOA

The HOA service takes care of storing all HOA's and what members with what role are part of the associations, running elections to choose the next boardmembers and vote on rule changes, manage reports of rule violations, and store/display the entire history of the HOA.

1.1.3 Activities

Activities allows users in an HOA to create activities. While an activity is active it will be visible to all users in that HOA on the noticeboard, on the noticeboard other users can declare interest in the active activities. Once the date on the activity has passed the activity becomes inactive and can only be seen in the history of the HOA. This microservice stores the activities and are sent to the HOA service when the history needs to be accessed.

1.1.4 Architectural choices

Of course, extracting other features from the HOA is still possible, like extracting elections, reports, or even history, however, we decided that if we extract the elections component as a separate microservice the HOA itself would lose some key functionality and serve as a largely redundant microservice. Therefore, we believe that all logic surrounding the voting and rules of the HOA should be encapsulated. The activities microservice, as well as users were extracted as they could be possibly integrated with other systems that way due to their generic functionality. For example, a social network platform might want to use our users and activities microservices. Additionally, user was extracted because of its necessary authentication providing the need for separation.

2. Design Patterns

2.1 Design patterns

2.1.1 Natural language description

The decorator pattern:
Problem: we have 2 types of votes. The type of vote should change depending on the context.
The Decorator Pattern attaches additional responsibilities to a vote dynamically. The additional responsibilities being: being an election vote or being a proposal vote. This flexible form of subclassing using a decorators fits our problem well.

2.1.2 Class diagram

The decorator pattern:

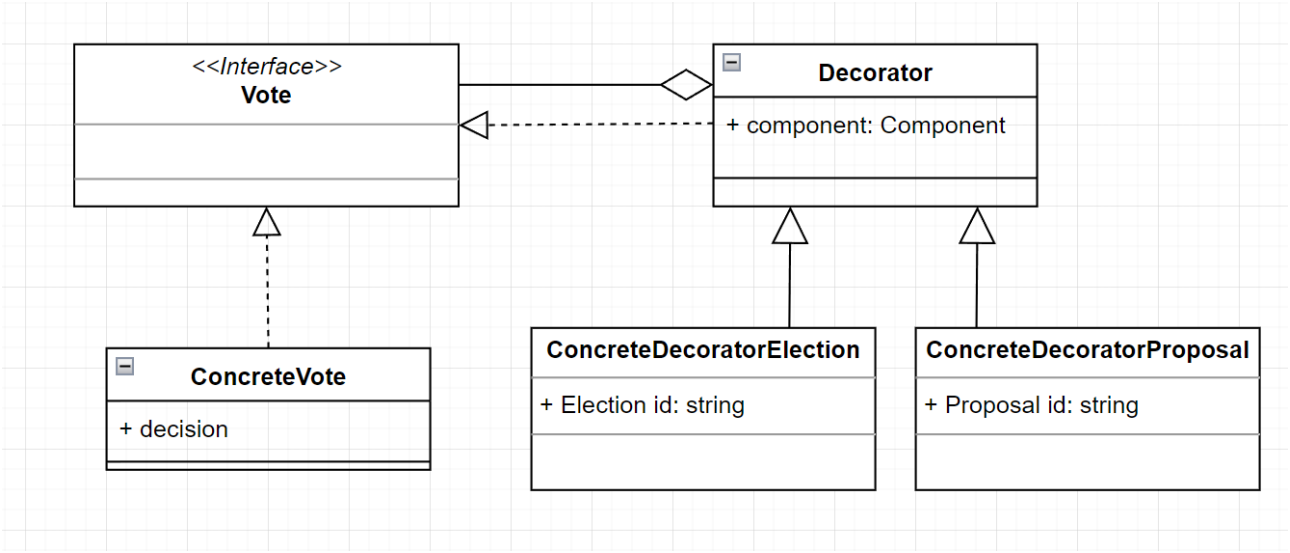

Figure 2.1: Class diagram for the votes system using a decorator pattern.

2.1.3 Design pattern implementations

The decorator pattern:

Bibliography

[1] Nick Youngson. *Homeowners association*. Retrieved: 27 November 2022. URL: https://pix4free.org/assets/library/2021-10-24/originals/homeowners-association.jpg.