Refugee BnB

Group 11

Christian Hougaard Pedersen (315269)

Nina Anna Wrona (315202)

Justina Ieva Bukinaitė (315199)

Karolis Sadeckas (315225)

Ignas Druskinis (315244)

Supervisors:

Joseph Chukwudi Okika
Jakob Knop Rasmussen

VIA University College

59.531

Software Engineering

3. semester

16-12-2022

Table of content

1. Abstract

On the 24th of February Russia invaded Ukraine creating a life threatening situation in many parts of Ukraine. This aggression has caused many people to lose their homes and therefore created the biggest refugee crisis in Europe in World War II ("2022 Russian invasion of Ukraine", 2022). Since the war started millions of refugees have been running to neighbouring countries looking for asylum. Although European countries have been putting a lot of effort into providing beds and housing for the refugees, the daily immigration of new people has created a lack of space. Although many people who are citizens from countries that are taking in refugees have offered their own space, the lack of a global media point, where people could post possible housing opportunities have slowed the housing process. Therefore we were compelled to create a web application, where volunteers could contribute by posting housing possibilities for people in need.
The idea has been presented to a Product Owner and potential stakeholders in order to discuss possible options for such a project and consequently 34 requirements, both functional and non-functional were created. For the project initiation 3 different actors were defined: refugee, host. Both hosts and refugees can create an account by registering and later on logic where your data is authenticated and if correctly imputed provides you access to your account. Hosts, when logged in, can create new housings with some underlying specifications, edit their personal details. Refugees, when logged in, can edit their personal details as well as few all available housing and apply, resulting in a potential agreement that is either approved or denied by the host.

The system has a layered architecture that follows an N-tier structure, that provides a limited user access for additional security as well as an opportunity for further expansion. The first tier plays as the visual representation of the system that the actors interact with and also provides the authentication implementation that it shares with the tier two. The second tier otherwise known as the logic tier contains the main business logic of the system where the data is processed and both sent to the database and retrieved from it. Also, containing the main business logic of the authentication, when users are logging in. The third tier acts as the logic for the database where it is mainly

used to structure the database and has a direct call to it. Each layer uses unique middleware to communicate with each other as well as the third and second tiers are implemented in java and first tier with C#. The communication between presentation and the business logic is done using the HTTP protocol, whereas the data transfer among business logic and data tier is done by gRPC framework. Lastly, the data tier implements a JPA repository that acts as a connector to the database that provides methods to be able to interact with it.

To conclude, in order to check the quality of the system the black box/integration and whitebox testing. According to the test results, the system communicates between the layers effortlessly and complies with requirements discussed with the Product Owner and the stakeholders.

1. Introduction

RefugeeBnB, this project was made referencing the United Nations goal number 16, it consists of a system that is designed to connect refugees with hosts offering temporary accommodation. With the ongoing war crisis, many refugees are struggling to find shelter, they do not have a platform to look for a temporary home, therefore it is hard for them to communicate with people that are willing to help.

Our platform aims to make it easier for refugees to find a place to stay, and for hosts to offer their accommodations.

RefugeeBnB, has a registration system that includes verification for both refugees and hosts, ensuring the safety and security of all parties involved.

Once registered, refugees will be able to search for available accommodations based on location and number of people. Hosts will have the option to accept or decline requests from refugees.

We believe that RefugeeBnB can make a real difference in the lives of refugees, and we are dedicated to creating a platform that is easy to use, efficient, and secure.

The report of the product will cover the creation process of RefugeeBnB, as well as the implementation and its testing. Starting off with Analysis, that gives a brief examination of what the project is about. This introduction is next followed by the Design segment, where the creation process of architecture and design will be shown.

2. Analysis

As said in the introduction, this section is going to cover in detail all the requirements given from the stakeholders. We are going to include use cases, system sequence diagram, domain model and security.

2.1 Overview of the system

The stakeholders requested a system that would provide refugees from different countries a possibility to find a place to stay. The system was supposed to equip volunteers, who are ready to share their apartments and houses with refugees, with tools to communicate their offers with the ones looking for them.

According to the stakeholders the system should have a registering option for volunteers, which we call hosts, refugees, job providers, UN representatives and an administrator. It should have a login system included. Additionally each of the actors should be able to change their personal information, stored in the system, as well as delete their account.

Hosts should be able to post accommodation they want to offer to registered refugees. Hosts should be able to overview and delete the posted accommodation from the system. When the refugee sends a request to occupy one of his posted housings, he should be able to see the details of that refugee and accept or decline that request. Additionally the host should be able to overview all the requests he has accepted and those which are still not defined as accepted or declined.

Refugees should be able to access and see some of the details of the available accommodations. Furthermore, they should be able to filter housings by country, city and amount of people that the accommodation can host. They should be able to send to the hosts of those housings information that they are interested in living there.

Whenever a host accepts that request they should be informed. They should be able to contact the host in order to discuss housing-agreement details. On top of that refugees should be provided with an opportunity to look and apply for jobs. Accordingly the system needs to have the means for displaying and managing job offers.

Job providers should be able to post job offers. They should be able to have an overview of potential applicants and those who have already shown interest in a job. They should be able to accept or decline applicants.

The UN representative should be able to check the reliability of users registering into the system before they get access to the system's tools.

The administrator should be able to access all details regarding all users. He should be able to delete or update any of the details.

2.2 Requirements

2.2.1 Functional Requirements

1. As a refugee, I would like to apply for refugee BnB, in order to be able to find accommodation.

2. As a host, I would like to apply for refugee BnB, in order to be able to receive refugees that require accommodation.

3. As a host, I would like to offer housing in order to provide a place to stay for a refugee.

4. As a refugee I would like to view all the available housing in order to choose the one that fits me most.

5. As a refugee I would like to send a request to a host in order to get an accommodation.

6. As a host I would like to see all pending requests from the refugees in order to accept or decline them.

7. As a host I would like to accept or decline the refugee candidate housing request myself in order to minimise conflicts between the hosts and refugees.

8. As a host, I want to be able to remove my listed housing(s) from the system, in case I do not want to participate in the system any longer.

9. As a refugee, i would like to be able to filter available housing by eg. space available, city or country, in order to find the accommodation best suited for my needs.

10. As a host I want to be able to add more housing, in order to widen the amount of possible housing opportunities for refugees.

11. As a refugee, I want to be able to delete my profile from the system, in case I do not need the services provided any longer.

12. As a host, I want to be able to delete my profile from the system, in case I do not want to provide the services any longer.

13. As a refugee, I would like to be able to put my lifestyle choice or/and religion on my profile, in order to minimise conflicts between me and the host.

14. As a refugee I want to be able to include my family into the housing agreement in order to let the host know how many people I want to bring with me.

15. As a host I want to know before signing the agreement how many people the refugee is bringing with him, in order to decide if I want to offer them a place to stay.

16. As a UN representative, I would like to be able to verify potential hosts and refugees, in order to minimise the chances of misuse of the system.

17. As a refugee I would like to be able to chat with the host in order to discuss the rental agreement.

18. As a job provider I would like to apply for refugee Bnb, in order to be able to offer refugees a job.

19. As a refugee I want to list my skills in order to get a job offer.

20. As a job provider I would like to see all refugees that are seeking a job and their skills in order to see if I can offer them a job.

21. As a job provider I would like to send a job offer for a refugee so he could accept it.

22. As an administrator I would like to see all of the users in the system in order to have an overview of the system.

23. As an administrator I would like to remove a user from a system in order to manage inactive users.

24. As a refugee I want to know if there are any pets in the housing I want to apply to, in order to avoid an allergic reaction.

25. As a refugee I want to see which hosts accept pets in their housing, in order to find a place where I can live with my pets.

26. As a host I want to write an agreement of housing between me and a refugee in order to ensure safe renting.

31. As a host I would like to be able to log in in order to have an account that only I have access to.

32. As a refugee I would like to be able to log in in order to have an account that only I have access to.

33. As a host I would like to be able to change my profile details in order to keep the information up to date.

34. As a refugee I would like to be able to change my profile details in order to keep the information up to date.

2.2.2 Non functional requirements

27. The system will send an automated message to a user (host/refugee) if the user has not been active in the past three months.

28. The website will have different possible languages available.

29. The hosts' housing address details are private in order to avoid catfishes finding his personal information.

30. Refugees will have personal information(phone number, email, ID copy/passport/driving licence, residence permit) private to only the people that are owners of the housing I apply to.

2.3 Use cases

After making a list of requirements, the Use Case Diagram was created, in order to visualise expected behaviour of the users. Additionally the Brief and Fully Dressed Format of Use Cases were added in the interest of detailed illustration of the actions.

2.3.1 Use Case Diagram

The Use Case Diagram represents two actors: a Refugee and a Host. Both of them can Register, Log in and Manage accounts in the system. Then their tasks divide. Refugees apply for the housing, while hosts can Manage the accommodation, including posting new accommodation, as well as managing the housing agreements.

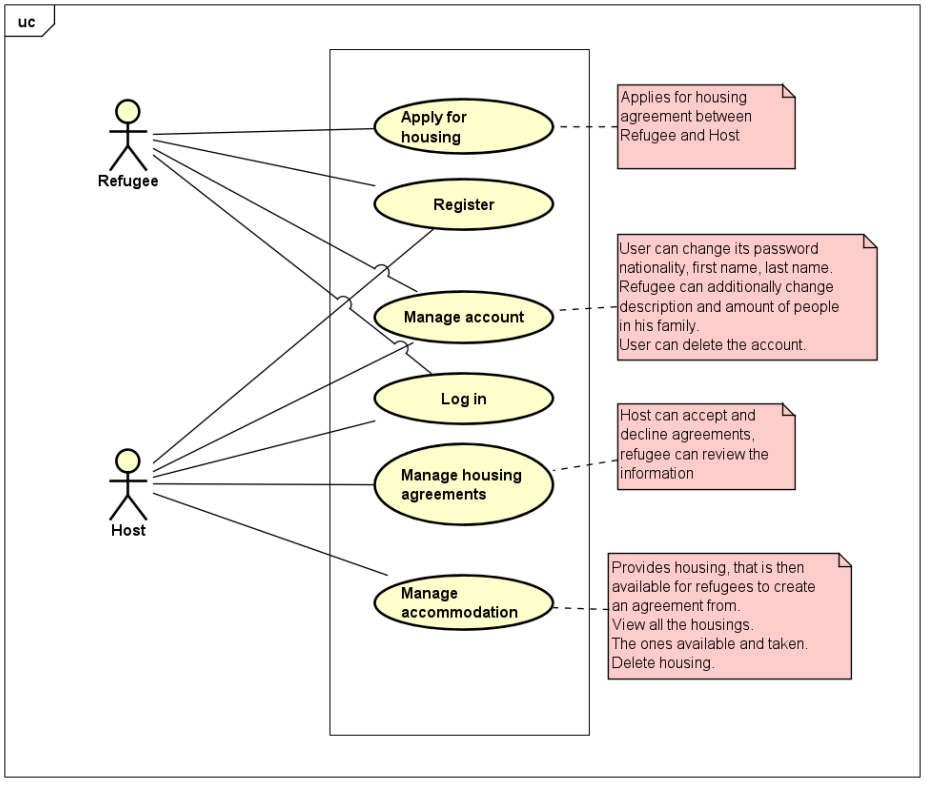


Figure 1 - Use Case Diagram

2.3.2. Use Case Descriptions

Use Case Description - Brief format

Register

The user can create an account in the system. He can choose to be either a host or a refugee.

Log in

Users can enter specific functionalities of the system by providing matching email and password.

Apply for housing

Users that are refugees can apply for living in a house that the host has published as available.

Manage account

Users can update their accounts, by changing details they have entered while registering into the system. Users can also delete their account at any moment.

Manage housing agreements

Users that are hosts can accept or decline the requests, called agreements, from the refugee that applies for their housing.

Manage accommodation

Users that are hosts can offer accommodation for refugees, free of charge. They can also delete those housings from the system any time.

Use Case Description - Fully Dressed Format

USE CASE	**Register**
Summary	A user creates a new account
Actor	Host, Refugee
Precondition	
Postcondition	Successfully creates new account
Base Sequence	1. Enter Register view 2. Enter all necessary personal details: email, password, gender, nationality, first name, last name. 3. Proceed
Alternative Sequence	2a. 1. As a host enter other, not required, personal details: middle name. 2b. 1. As a refugee enter other, not required, personal details: middle name, description of your preferences. 2. As a refugee enter other, required, personal details: number of your family members that are coming with a refugee. 3a. 1. Not all required fields are filled. 2. System displays error message
Note	

USE CASE	**Log in**
Summary	A user logs in to his account and can view booking and create new ones.
Actor	Host, Refugee
Precondition	The user must have a username and password/account.
Postcondition	Successfully logs in

USE CASE	**Log in**
Base Sequence	1. Enter Log in view for the guest 2. Enter Username and password 3. Proceed 4. User gets sent to the main page.
Alternative Sequence	3a. 1. Credentials are not accepted by the system.
Note	

USE CASE	**Apply for housing**
Summary	A refugee sends information to the owner of the chosen housing that he is interested in living there.
Actor	Refugee
Precondition	The user must have logged in as a refugee.
Postcondition	Successfully sends information.
Base Sequence	1. View all housings. 2. Choose an available housing. 3. Send a request to live there to an owner of the house.
Alternative Sequence	1a. 1. No housing was found.
Note	

USE CASE	**Manage account**
Summary	A user updates information on its account.
Actor	Refugee, Host
Precondition	The user must have logged in as a refugee or a host.

USE CASE	**Manage account**
Postcondition	Successfully sends information.
Base Sequence	1. View currently stored data. 2. Change necessary data. 3. Proceed.
Alternative Sequence	1a. 2. Delete the account.
Note	

USE CASE	**Manage housing agreements**
Summary	A host accepts or declines housing agreements.
Actor	Host
Precondition	The host must have logged in as a refugee or a host. The host must have posted a house to rent. The host must have received a request from a refugee.
Postcondition	Successfully sends a message with the response back to the refugee.
Base Sequence	1. View all pending and accepted requests. 2. Accept a request.
Alternative Sequence	2a. 1. Decline a request.
Note	

USE CASE	**Manage accommodation**
Summary	A user updates information on its account.
Actor	Host
Precondition	The user must have logged in as a host.

USE CASE	**Manage accommodation**
Postcondition	Successfully created or deleted housing.
Base Sequence	1. Add accommodation a. Fill all necessary address information: country, city, street name, house number, post code, capacity. 2. Proceed.
Alternative Sequence	1a. 1. View all housings that were already created by the host. a. i. Choose housing. ii. Delete selected housing. b. i. No housing was found.
Note	

2.4 Activity Diagram

Subsequently the activity diagrams were created. Not only does it cover the main Use Case scenarios, but also the ones that can alternatively happen. The diagram shown below presents Manage Account Use Case. The rest of the diagrams can be found in Appendix B.I.

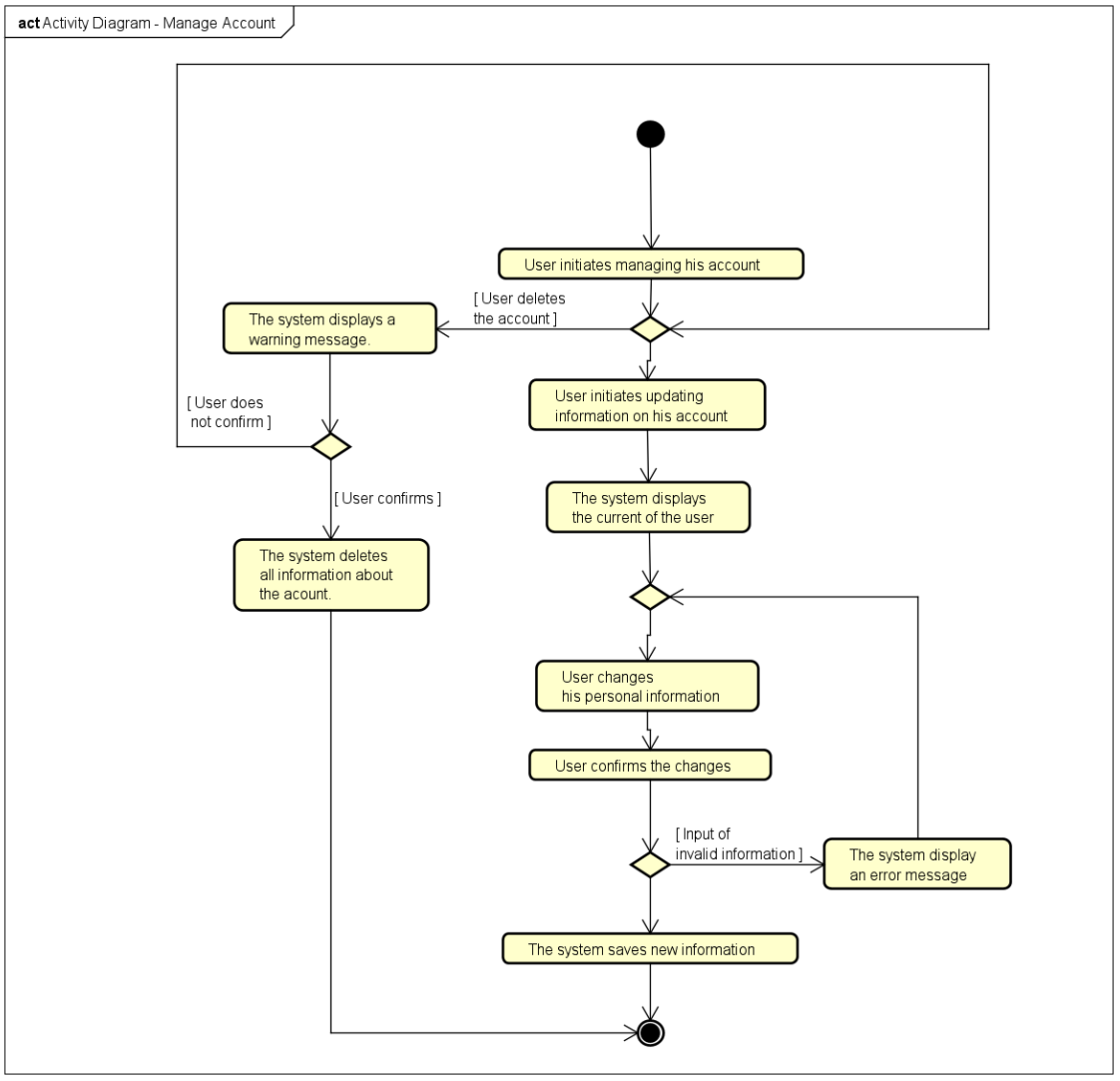


Figure 2 - Activity Diagram - Manage Account

2.5 System sequence diagram

In this System Sequence Diagram the behaviour of adding new accommodation to the system. The user, having a role of a host, requests adding a new accommodation. The System sends the form to complete in order to add the housing to the system. Host writes the address of the housing and its capacity. If the operation finishes successfully, the information of the newly added accommodation gets back to the User.

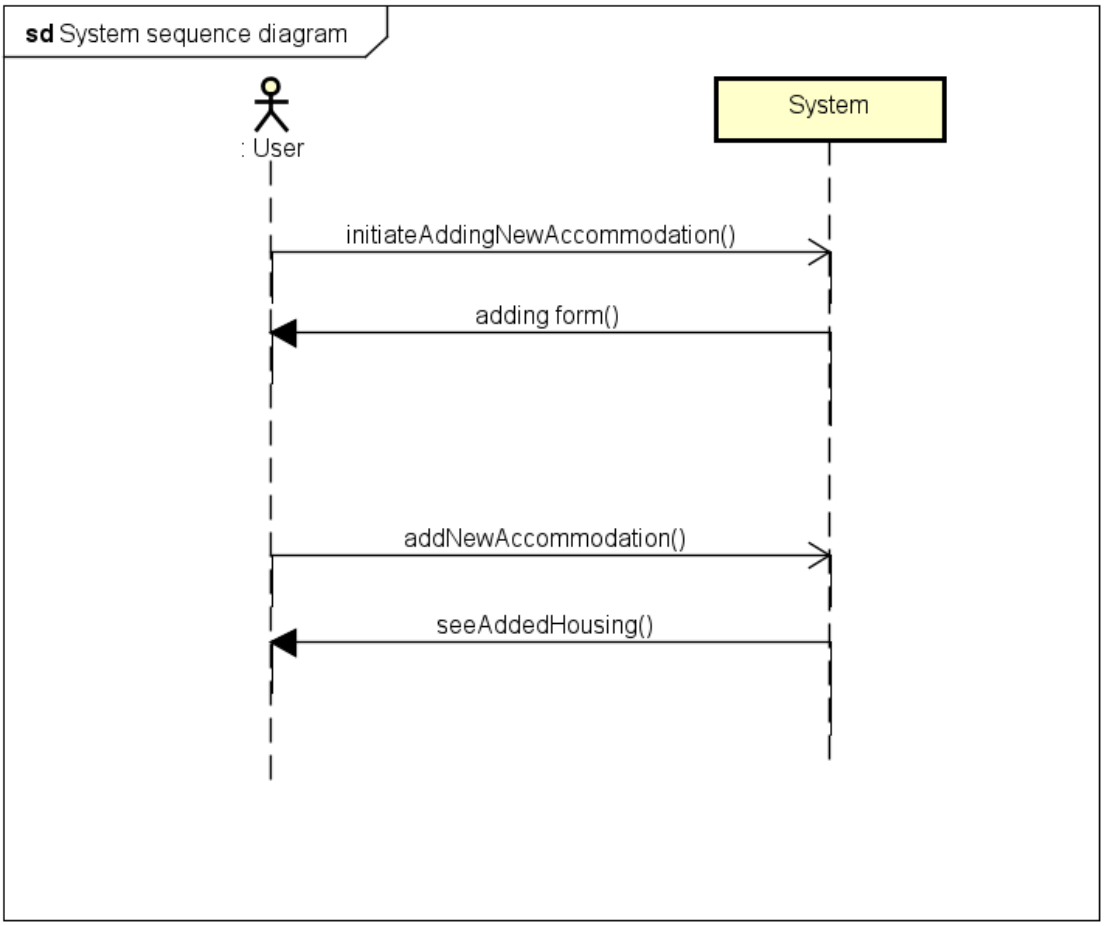


Figure 3 - System Sequence Diagram

2.5 Domain model

The next step, after finishing the System Sequence Diagram, the Domain Model was created. With regard to the Use Case Diagram the Host, UN Representative and Refugee were included. While UN Representative Host and Refugee interact differently with Agreement. Refugee applies for an Agreement, which means that he is requesting to sign a Housing Agreement with the host. On the contrary, Host can accept or decline that request. Host creates Housing with a unique Address. Agreement includes Housing that the Host is offering for the Refugee to rent.

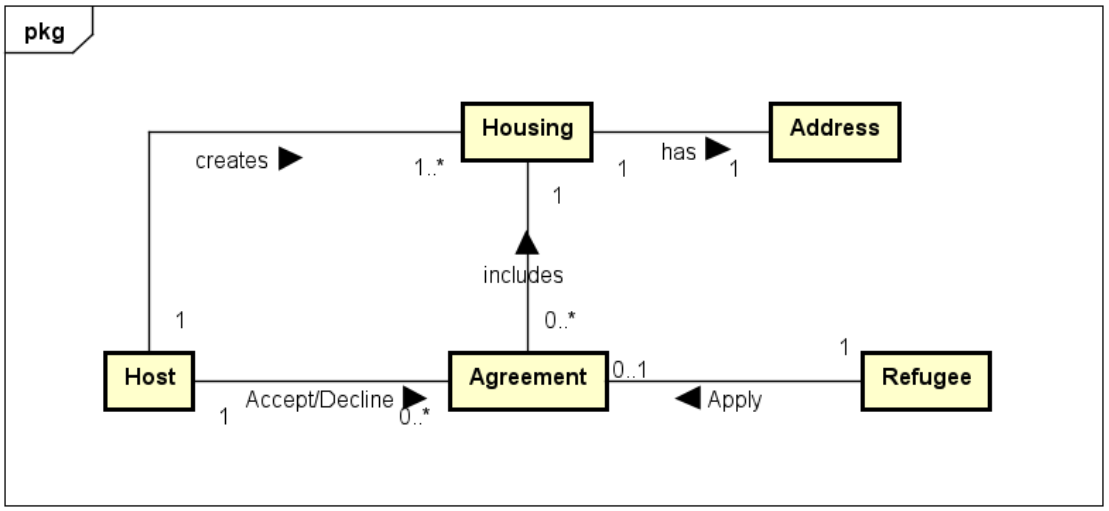


Figure 4 - Domain Model

2.6 Security

The three standard objectives that are system wants to meet are known as CIA triad: Confidentiality, Integrity and Availability.

To ensure the confidentiality and security of the vulnerable data the system has a Login Page, which restricts unwanted visitors to access sensitive information. The system stores encrypted passwords in the database, which secures them from being discovered, even if an unauthorised person accesses the database.

The UN representative as well as the administrator are responsible for checking the accuracy of the information that system possesses. Nonetheless the system secures from making unauthorised changes. One of the system functionalities is to assign each user a role, which defines the accessibility to different tools of the system. For instance only Hosts can create new accommodation to offer, however it's only Refugees that can display all available for tenancy housing. This way we ensure availability measures are met, which also means that no attacker can manipulate, access or delete any of the stored data.

Another objective of the system is to provide availability of the system at any time of the day and night. The users should not be restricted by any system downtime as well as exhaust of the resources.

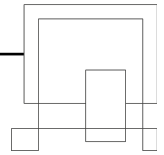
To ensure the objectives were met the threat model was established.
Consequently the vulnerability of the system was taken into consideration. The entry points, where attackers can possibly get into our system and pose a threat are the Login Page, HTTP Port, Login Function, Register Page, Register Function.

All pages of RefugeeBnB will only be accessible via TLS. The SSL certification will be provided. That will enable the website to have an encrypted connection with its users.

This will secure the action of exchanging sensitive data over the network and protect confidentiality.

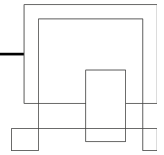
Anonymous Web Users can access both Register Page and Login Page. Any person can register into the Website, however only those who get accepted by the UN representative get the access to the inside of the system. Nevertheless it is an opportunity for the DDoS attacker to overwhelm the system with countless new accounts.

Meanwhile the Login Function accepts the users that provided credentials that already exist in the database. That’s an opportunity for the phishing attack. Hackers can use Brute Force in order to find the right passwords for emails existing in the database. What is more, attackers can use it for stealing the identity as well as taking advantage of the sensitive information they get access to. This issue could be solved by including a password manager tool to enforce users to create more complicated, more difficult to crack, passwords.



2.6.1 Threat model

Security threat	Vulnerability	Goals of the attacker	Violation	EINOO	Type	Means of the attacker	Preventive measures
Phishing attack	Weak passwords that hackers can easily crack.	Spoofing, Repudation, Information Disclosure, Eleviation of Privilage	Authentication, Non-repudation, Confidentiality, Authorization	External, Network, Online	Brute force	Active or Passive	- Include a password manager tool to enforce the use of strong passwords.
Ransomware attack	Security considerations are not checked when dealing with unreliable hosts.	Tampering (locks and encrypts users data), Denial of Service	Integrity, Availability	Internal, Network, Online	CPA	Active	- Check hosts reliability before getting into a partnership. Create a contract for them to sign while registering in the system.
DDoS attack	The website runs on a free version of the software package, which offers limited security capabilities.	Denial of Service	Availability	External, Network, Online	Brute force/DDoS	Passive	- Move the website to a more secure and trusted platform. - Get SSL certification for the website.
Ransomware attack	System does not have a firewall.	Spoofing, Repudation, Information Disclosure, Eleviation of Privilage	Authentication, Non-repudation, Confidentiality, Authorization	External, Network, Online	Ransomware	Active	- Add a firewall software. - Control the firewall's inbound traffic files.
Man-in-the-middle attack	Not having a SSL certification.	Spoofing, Repudation, Information Disclosure	Authentication, Non-repudation, Confidentiality	External, Network, Online	MITM	Active	- Get SSL certification for the website.



2.6.2 Risk assessment

Security threat	Security risk	Incident consequence	Risk level	Impact level
Phishing attack	Unauthorised users gain access to the operating system and steal data or crash the system	● Loss of productivity due to system downtime ● Leakage of certain data types can lead to loss of IP rights ● Stealing users identity	HIGH	HIGH
Ransomware attack	Data theft or loss due to security breaches from users side.	● Data loss ● Loss of users trust ● Regulatory penalties	MEDIUM	HIGH
DDoS attack	Users are not able to access the website	● Loss of clients due to the inability to create and manage housing agreements ● Advantage for competitors	MEDIUM	LOW
Ransomware attack	Unauthorised users hack into the system through the internet	● Unauthorised access to sensitive files due to system vulnerabilities. ● Stealing users identity	LOW	LOW
Man-in-the-middle attack	Users get their information stolen and their messages are modified	● Attackers get access to sensitive information.	HIGH	MEDIUM

3. Design

3.1 Architecture

As the system is intended to be used by multiple users, both hosts and refugees, at the same time, with all of them having access to the same pool of data, it became clear that the system should be designed as a multi-tiered system.
The system was designed with N-Tier architecture in mind, meaning that the functionality is split between multiple tiers.
The system is divided into the following three tiers:

- The Presentation-tier, which contains the GUI, and is the part of the system that the users will be able to interact with.
- The Logic-tier, which contains the business logic for the system.
- The Data-tier, which consists of a database used for persisting data and a program used for facilitating access to the database.

To ensure ease of access to the system, it was chosen that the Presentation-tier will be written in C#, as a Web Application using the Blazor WebAssembly framework, which makes it possible for the application to be run from a web-browser instead of the user having to download and install a program.
The Logic-tier as well as the database-connection in the Data-tier was chosen to be written in Java.
As the system will have multiple tiers that need to be able to communicate, it was necessary to decide on which middleware to use for facilitating this communication. Between the first and second tiers, it was chosen to use the HTTP-protocol for communication by designing a REST-API, and between the second and third tier it was chosen to use gRPC.

Figure 5 - System Architecture Diagram

System Architecture Diagram

The system is intended to function in such a way, that the user accesses the presentation-tier through a browser, and using the GUI, performs an action, which is then sent as a HTTP-request to the logic tier.

In the logic tier, the request sent from the user will be subject to business logic, part of which may include the logic tier transmitting one or more gRPC messages to the data tier and receiving responses for these.

The logic tier will then send a HTTP-response back to the user.

3.2 Technologies

As shown in the System Architecture Diagram, several different technologies are to be used when implementing the system. The reasoning behind this is to streamline the development process, as the chosen frameworks include predefined libraries of code, which is automatically called upon when needed, which leads to the developers being able to spend more time on the unique features of the system being developed, while leaving the low-level functionality to the framework.

For the Logic and Data tier, it was chosen to use Spring, because of its included features for creating REST-APIs, security, and persistence-handling.

For the Presentation tier, the frameworks chosen are Blazor, used for creating the WebAssembly application, and Radzen which includes pre-made components to be used in the WebAssembly to ease development.

For the communication between the Logic and Data tier, gRPC is used for its efficiency and remote procedure calling functionality.

3.2.1 Spring

As stated above, it was decided early in the design process to use the Spring framework for the logic, and data-tiers, as it was known from previous experience that this framework would allow a simple way of creating a REST-API by utilizing the Spring Web-dependency. The reasoning behind using Spring for the data tier as well, was to take advantage of the Spring Data JPA-dependency, which enables the developers to connect to a database without manually having to implement an adapter for the JDBC-driver.

3.2.2 Blazor

The Blazor framework makes it possible to create a Web Application which can run directly in a browser, eliminating the need for the user of the system to download and/or install a program to use the system.
The Blazor Web Application consists of multiple components, each of which can be defined by HTML, CSS or C# or a mix of all three.

3.2.3 REST

To ensure scalability and maintainability, it was decided to use REST (Representational State Transfer) to facilitate communication between the Presentation and Logic tier.
The REST-API uses the HTTP-protocol for communication, by exposing specific endpoints from the controllers in the logic tier, for the presentation tier to send Http Requests to.
The various endpoints exposed are to be called with different types of requests depending on the actions taken, each corresponding to a similar CRUD-function.
For instance, when at any point in the system, a new object is added, a POST-request is made with the body of the request containing a JSON representation of a domain transfer object to be created.

In a similar fashion, whenever the client from the presentation tier wants to retrieve an object, a GET-request is made to a specific endpoint, and when an object is to be removed a REMOVE-request is made.

3.2.4 gRPC

In the Refugee Bnb system, gRPC is used for the connection between the Logic and Data tiers.
gRPC is a framework for executing Remote Procedure Calls using Protocol Buffers. A Remote Procedure Call is when a program in one context can call a method to be executed in another context, i.e., on another computer.
Protocol Buffers are used to declare the contract between the entities, consisting of messages and services.
Protocol Buffers are defined using .proto files, which contains the messages and services to be used. When the proto files are compiled, the messages are generated as classes and the services are generated as an implementation base (ImplBase) with the services as methods to be overridden on the server-side. Furthermore, stub-classes are generated to be used client-side for calling the methods.

```
syntax = "proto3";
import "SharedMessages.proto";

option java_multiple_files = true;
option java_package = "via.sep3.group11.tier3.protobuf";

service Host{
  rpc CreateHost(GHost) returns (GHost);
  rpc GetHostByEmail(GEmail) returns (GHost);
  rpc getHostByHousingId(GId) returns (GHost);
  rpc deleteAccount (GEmail) returns (GEmpty);
  rpc updateInformation(GHostDetails) returns (GHost);
}
```

```
message GHostDetails{
  string firstName = 1;
  string email = 2;
  string password = 3;
  string gender = 4;
  string nationality = 5;
  string middleName = 6;
  string lastName = 7;
  GDateOfBirth dateOfBirth = 8;
}

message GHost{
  string firstName = 1;
  string email = 2;
  string password = 3;
  string gender = 4;
  string nationality = 5;
  string middleName = 6;
  string lastName = 7;
  GDateOfBirth dateOfBirth = 8;
  GHousing housing = 9;
}
```

Figure 6

The figure above shows an example of how the proto files are designed for the Refugee Bnb system. On the right, in the Host.proto file, the methods that can be called can be seen and on the right a part of the SharedMessages.proto file is shown, which is the file containing the messages that are going to be used in the system.
It was decided to let the services be Unary RPCs, meaning that the contract declares that a client sends a single request to the server, where the request is processed before a single response is returned.

3.2.5 PostgreSQL

In order for the system to function as intended, it is necessary for the data tier to actually persist the data. It was chosen to use PostgreSQL as the database due to its synergy with the already decided upon Spring Data JPA, but also because of the development team's previous experience with PostgreSQL.

3.3 Design Patterns

3.3.1 Dependency Injection

To help achieve decoupling in the system, the Dependency Injection pattern is to be used wherever possible, by using constructor injection to make sure that a single class cannot be instantiated without its dependencies.
Following the SOLID principles, this is achieved by having the client class declare an instance-variable of an interface, which is then initialised using constructor injection. These interfaces will each have a concrete implementation where all interface methods are implemented. An added benefit of this approach is that it makes it simpler if, in the future, the implementations would change, as following the Dependency Inversion principle makes it a matter of only changing the concrete implementation of the interface, instead of having to change the methods calling the interface accordingly.

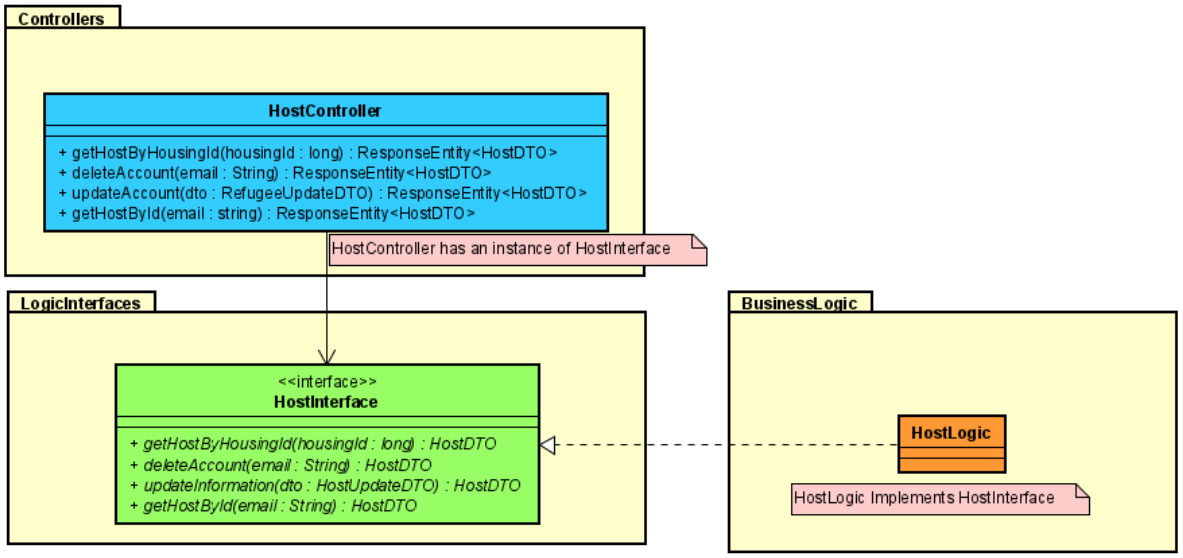


Figure 7 - Dependency Injection

As shown in the figure above, a concrete example of Dependency Injection can be found in the relations between HostController, HostInterface and HostLogic.

3.3.2 MVVM Pattern

As the presentation tier is going to be implemented using the Blazor framework, it is logical to design it to follow the Model-View-ViewModel pattern, in order to achieve separation between the GUI and the functionality behind this.

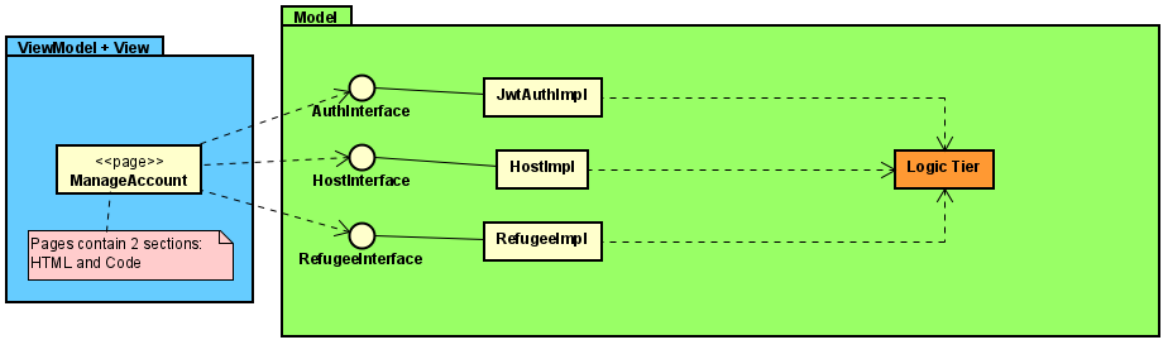


Figure 8 - MVVM

The example above shows the MVVM pattern for the Manage Account page, that is going to be accessible for both Refugees and Hosts. Blazor pages consist of a section containing the actual View, written in HTML and CSS, and a section containing code, which serves as the ViewModel. The ViewModel has access to relevant interfaces - and hereby, using dependency injection, their concrete implementations. The ViewModel requests and receives information from the Model and updates the view accordingly. The Model consists of the interfaces, their concrete implementations and the Logic and Data tiers.

3.3.3 DAO Pattern

The DAO (Data Access Object) pattern is to be used in the data tier, to achieve abstraction between the database and the java application.
The DAOs will be implemented as interfaces, with each interface pertaining to a single domain-level object such as Refugee, Host, Housing and Agreement. Each of these interfaces will have methods representing variations of CRUD operations.

Each of the DAOs will be implemented by a service-class containing an injected Spring JPA repository, which it will use to connect to and communicate with the database.

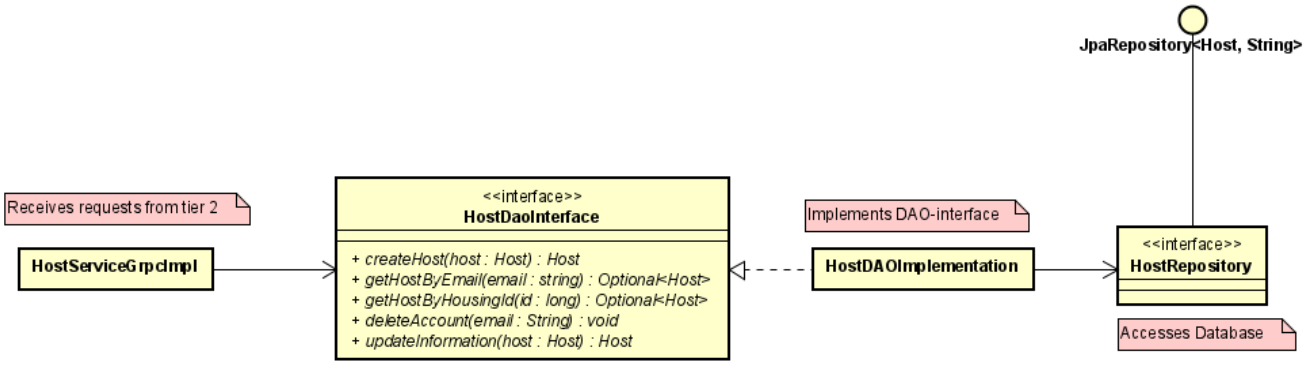


Figure 9 - DAO

Shown above, is an example of how the DAO-pattern is to be implemented in Refugee Bnb. The request from the logic tier is to be processed by the gRPC-implementation, which has an instance of the Host DAO-interface. This interface is implemented by the service class HostDAOImplementation, and calls the HostRepository that makes the call to the actual database.
If the stakeholders of Refugee Bnb would, at any time, like to change from using a database to another form of storage, this can be implemented simply by changing the DAO implementation and repository.

3.4 Security

Security is an important concern in any application, especially so when it is processing personal information about its users. Therefore, it is crucial to consider implementing various security mechanisms to reduce the risks regarding the system security.
As described earlier, the Refugee Bnb system uses the HTTP protocol for communication between the three tiers.
The communication between the first and second tier is to be implemented using a REST-API which, unless security measures are implemented, will communicate by passing messages in a data-interchange format known as JSON (JavaScript Object Notation) through the body of the requests and responses made.

This opens a vulnerability, as it will technically be possible for adversaries to intercept the messages and have access to the JSON inside of them. This vulnerability is two-fold as the adversaries will both have the possibility to read the data sent and received and to change the data passed.

3.4.1 SSL/TLS

SSL is an abbreviation of Secure Socket Layer, and TLS is an expansion of this, meaning Transport Layer Security. The aims of these protocols are to ensure secure communication by using the HTTPS protocol.
The way this is done is as follows, using Refugee Bnb as an example:
When a user accesses the system through a browser, the first thing that happens is that the browser connects to the logic tier of the system using SSL. The logic tier (server) then responds with a certificate containing its public key. The browser then verifies this certificate by checking that it has been verified by a trusted third party and uses the server's public key to create a shared session key used to encrypt messages when communicating with the server. All data being sent will now be encrypted using this session key.
This means that when a user is registering as a new user through the GUI, the personal information sent during this process will no longer be sent as plaintext, making it harder for adversaries to get access to the data.

3.4.2 Digital signatures

A digital signature can be used to provide authentication and to ensure integrity and non-repudiation. It is widely used in systems throughout the world for its efficiency. Digital Signatures are created by hashing a message with the private key of the signer before sending. The recipient then receives the hashed signature along with the public key of the sender. The recipient then generates their own hash of the message and attempts to decrypt the sender's hashed signature using the sender's public key. If the generated hash is identical to the decrypted hash, it is certain that the message has not been modified.

3.4.3 Message Authentication Codes

MACs (Message Authentication Codes) are a security mechanism used to ensure that the message is sent by the actual sender and that it has not been altered along the way.
The MAC is generated by running the message through an algorithm depending on a shared symmetric key known by both the sender and recipient before the message is sent. The MAC is then appended to the message as a signature, and when the recipient receives the message, the same operation is performed on the message, and if the newly generated MAC matches the one sent as a signature, then the receiver is assured that the message's integrity is intact. For MACs to be implemented, the following requirements must be met:

1. knowing a message and MAC, is infeasible to find another message with same MAC
2. MACs should be uniformly distributed
3. MAC should depend equally on all bits of the message

Lars Bech Sørensen, 2022, Session 6 - Confidentiality and Integrity

In Refugee Bnb, when a host is processing (accepting or declining) an agreement-request, having a MAC sent along with the messages between the first and second tier would guarantee that the message has not been tampered with, which could lead to agreements getting declined where not intended and the other way around.
To ensure faster operations than when using a regular MAC, an HMAC implementation could be made instead. The main difference between the two is that a HMAC is based on a hash-function.

3.4.3 Authentication Protocols

Another important thing to consider regarding security is the actual authentication of users accessing the system. This is achieved by implementing an Authentication

Protocol which, simply put, receives the login credentials sent by the user and tries to validate them by comparing them to the credentials that are stored by the server.
For Refugee Bnb it was decided to use Spring Security and JWT (JavaScript Web Token) for authentication purposes. Whenever a user tries to log into the system, they will send a HTTP-request to the second tier containing their credentials. These credentials are then validated against what is stored in the database, and if they match, a JWT-token is generated and returned to the user, where it is saved in a cache until the user logs out or the connection is otherwise reset.
This token, other than the verified login-status of the user, also contains the role of the user (refugee or host), a timestamp telling when it was issued, and another timestamp telling when it will expire at which point the user will need to log on again. The token is to be encrypted using a shared private key stored in the logic tier.
The different endpoints in the controllers in the logic tier can then be implemented in such a way that only certain users are allowed to send requests to them, thereby securing the endpoints from misuse from outside parties.
Every time the user wants to access one of the before-mentioned URIs, the token will be put in the Authorization header of the HTTP-request sent, and the controller will then allow access.
This will ensure that the developers control the access to the endpoints, and hereby – the system.

3.4.4 Persistence

As described earlier, to ensure persistence of data in the system, it was decided to implement a database solution for Refugee Bnb.
When starting to design the database, an ER (Entity Relation) diagram was created, going from the Domain Model created during the analysis phase.

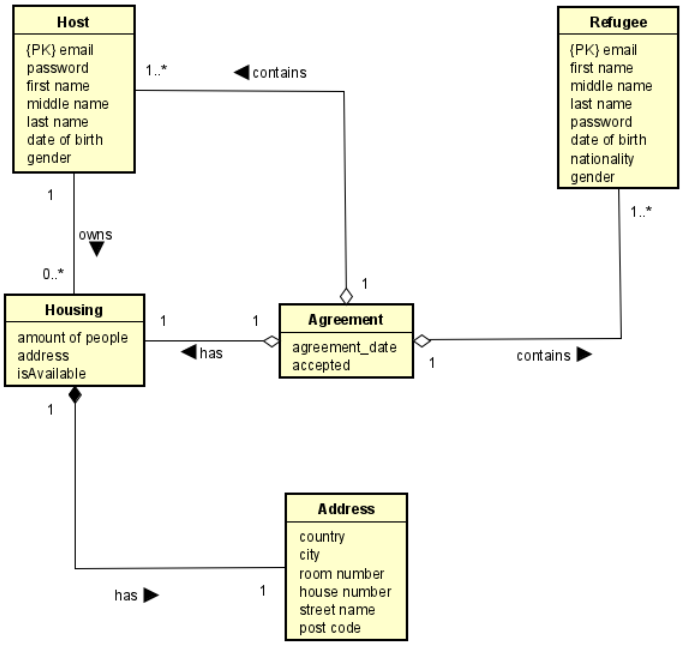


Figure 10 - ER Diagram

This entity diagram shows the basic structure of what is to become the different entities in the database, including the attributes and relations obtained from the Use Cases during the analysis phase.

After the entity diagram was created, a new diagram was made – the Class Model Diagram. This diagram is what is going to be used when implementing the database.

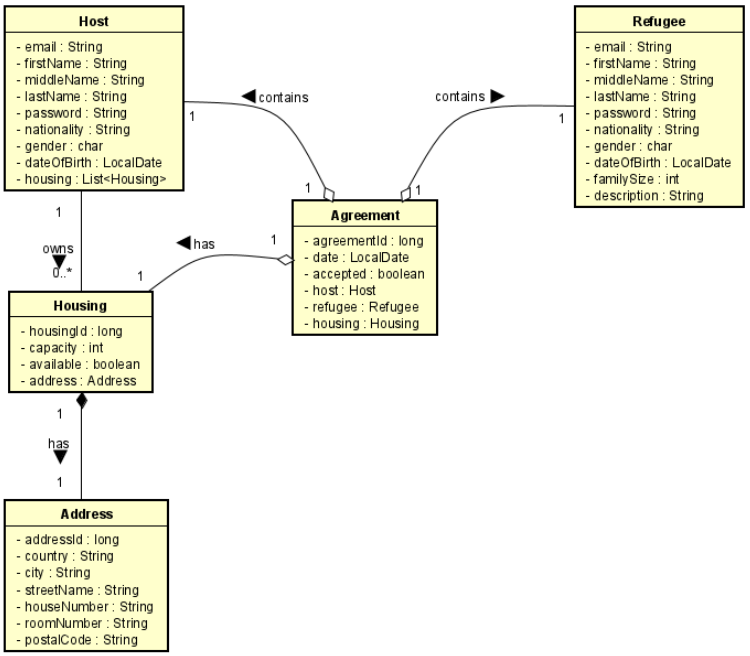


Figure 11 - Class Model Diagram

As the database is going to be implemented using Spring JPA, which is going to manage the creation of the join-table needed between host and housing, it was not deemed necessary to map the model to a logical model.
The biggest changes from the entity diagram to the class model diagram is that ids for address, housing and agreements were added, as well as navigational properties from host to housing, housing to address, and from agreement to housing, host, and refugee.

3.5 Class Diagram

After designing the Class Model Diagram, a Class Diagram for the whole system could be designed.
To ensure an easy-to-maintain structure in the code, the code across all tiers is structured into packages, with each package containing related classes.
Shown below is the class diagram for the data tier of the Refugee Bnb system.

As seen in the diagram, the code is structured in such a way that when implementing, it is easy to find the way around the system and see the relationship between the different classes.

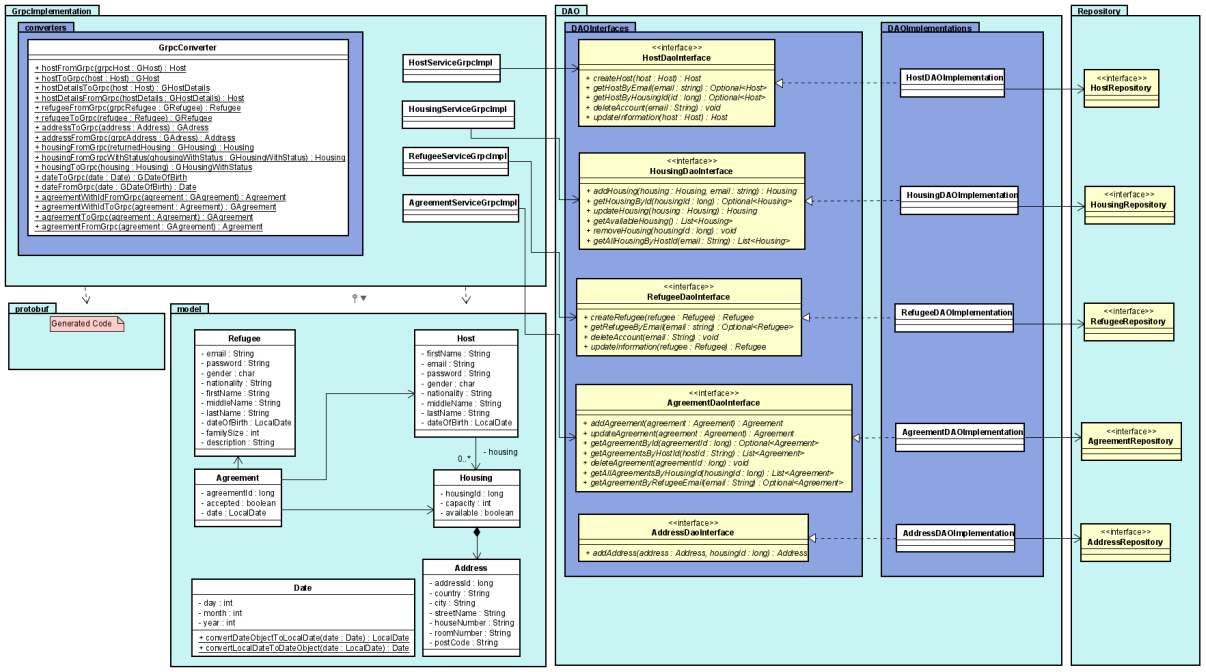


Figure 12 - Data Tier class diagram

The full class diagram can be found in Appendices B.A

3.6 Sequence Diagram

The sequence diagram shown below (can also be found in Appendices B.B) was created to illustrate how the different parts of the system will interact when a host processes an agreement, as this is one of the more complicated operations.
The Agreements boundary represents the Agreements page in the Blazor WASM, from where the user clicks either Confirm or Decline on an agreement. When the user clicks the button, a RespondAgreementDTO is created, consisting of the agreementId and a boolean value representing the choice made (true if confirming, false if declining). The page calls the RepondToAgreementAsync method in AgreementService, passing along

the domain transfer object. The service class then converts the dto to JSON, and sends it to the corresponding endpoint given by the AgreementController in the logic tier. From there, the dto is passed along to the AgreementLogic class, where the actual Agreement object is retrieved from the database.
If the user wants to confirm the agreement, the housing is retrieved from the database, its availability set to “not available” and the now updated housing is sent back to the database and stored again. Then the agreement's accepted-value is set to true, and the updateAgreement-method is passed along to the database along with the updated agreement.
If, on the other hand, the user wants to decline the agreement (or cancel it, if it was already in progress), the deleteAgreement-method is called in the data-tier, and the housings availability is set to “available”.
In both cases, after the logic has been performed, an AgreementDTO is created, consisting of the updated agreement, and is sent back to the presentation tier in JSON format, where it is then de-serialized and passed back to the Agreements page. The list of agreements is then updated to reflect the changes, and the user is shown a dialogue box confirming that the update has taken place.

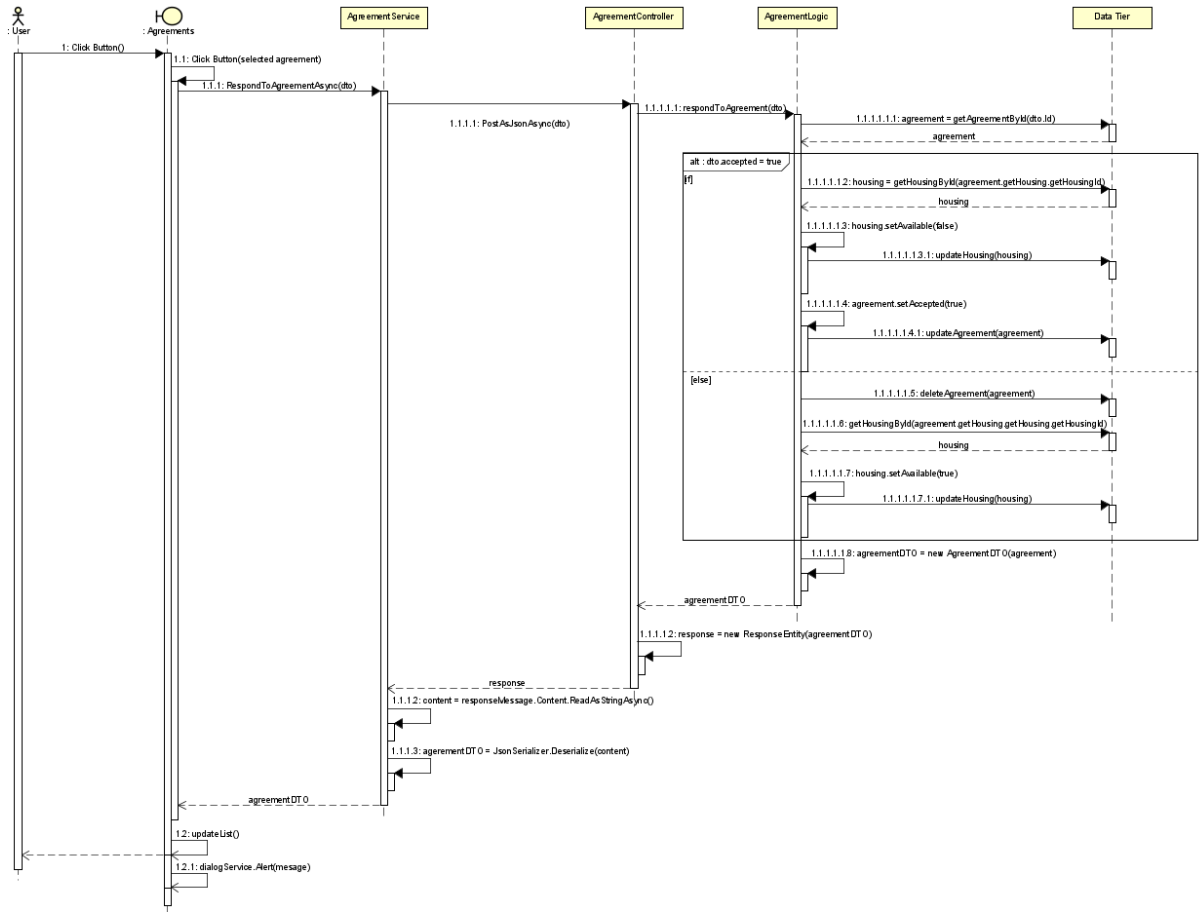


Figure 13 - Sequence Diagram

3.7 Graphical User Interface

The user interface is created using Blazor, and is designed using Radzen Components, HTML and CSS.

The GUI consists of various pages, each designed to present the needed information in a user-friendly manner.

A mock-up of how the website should end up looking was made by a member of the development team, and can be seen below.

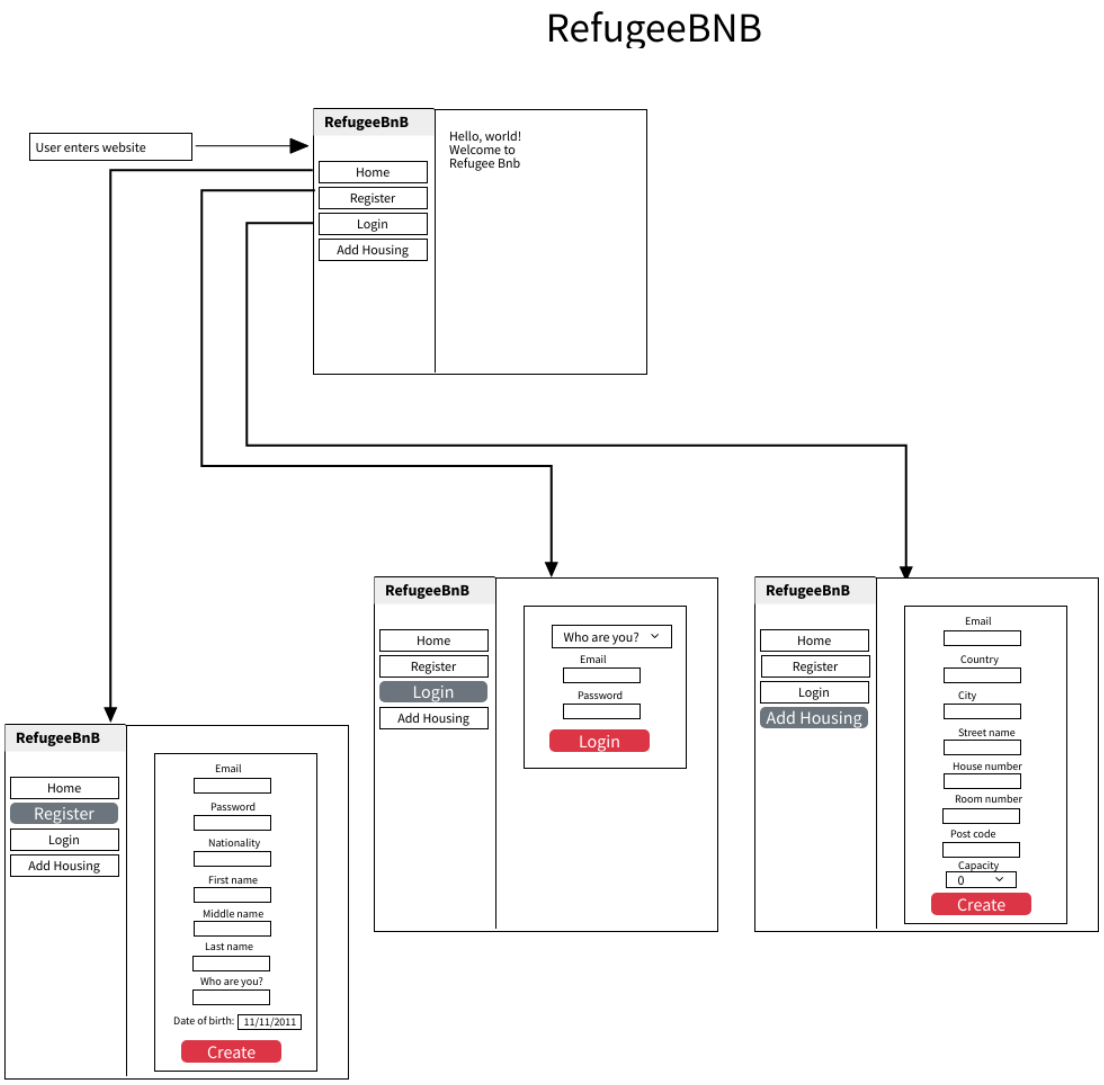
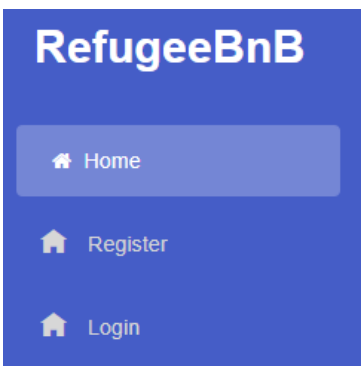
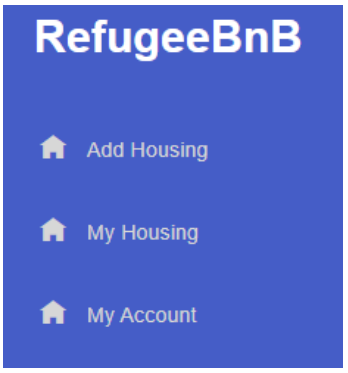


Figure 14 - Website Mockup

3.7.1 Navigation Menu

The navigation menu, present on the left side of the screen, is going to change looks and functionality depending on if a user is logged in – and the type of user being logged in to the system.





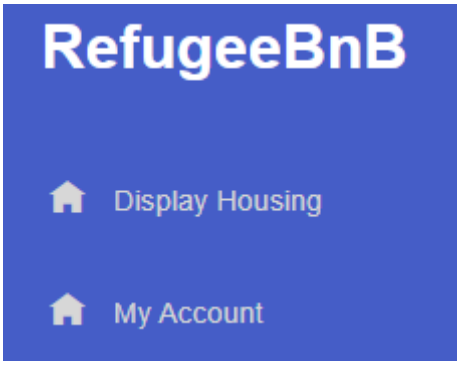


Figure 15 - Nav Menu, Not logged in *Figure 16 - Nav menu, Host* *Figure 17 - Nav menu, Refugee*

As shown in the figures above the contents of the navigation menu changes depending on the user privileges of the currently logged in user-type if any.

3.7.2 Manage Housing

When logged in as a host, it is possible for the user to see a list of their added housing. If any housing is currently part of an accepted agreement, the Delete button will not appear, and the status symbol will be a red x to indicate that the housing is not currently available.

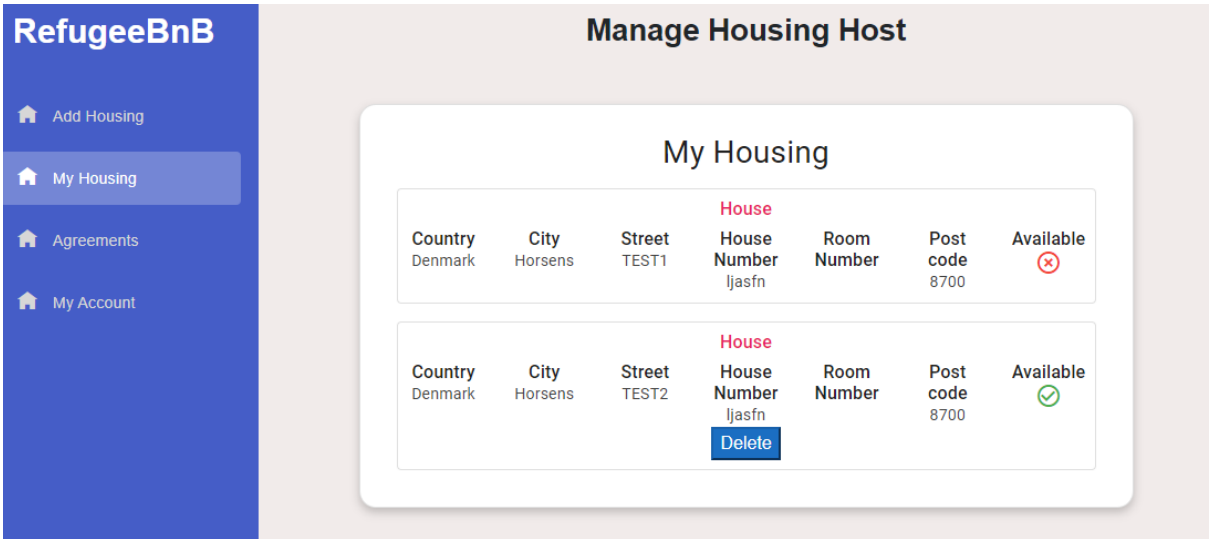


Figure 18 - Manage Housing View

3.7.3 Register

When a user wants to register for Refugee Bnb, they will be met with the following page. As shown below, depending on the type of user registering, extra fields will appear in the registration form.

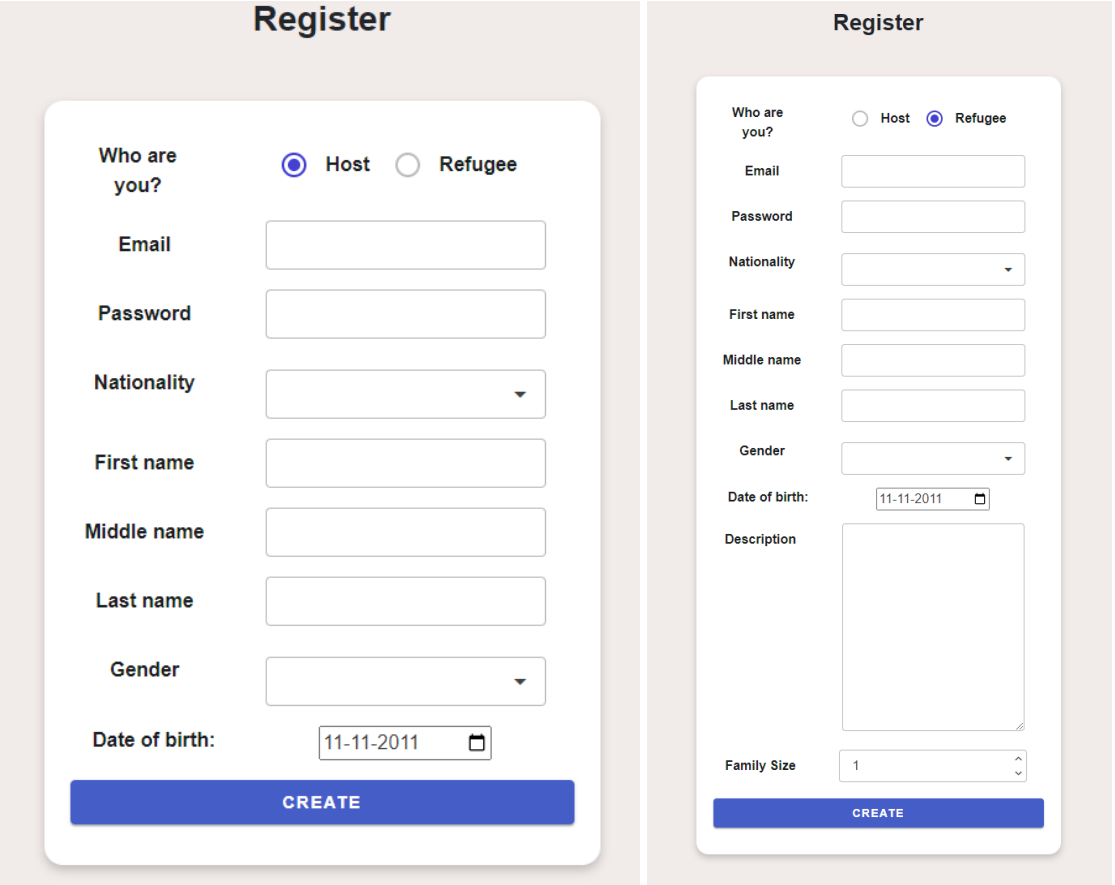


Figure 19 - Register view

4. Implementation

As outlined in the design section, the project will be implemented using the Blazor WebAssembly framework for the presentation tier, Spring for the logic as well as data tier, along with PostgreSQL for the database.

4.1 Presentation tier

4.1.1 Libraries

In addition to Blazor framework we used these libraries:

- Microsoft.AspNetCore.Authorization - provides support for user authentication and authorization
- Microsoft.AspNetCore.Components.Authorization - provides support for user authentication and authorization in client-side Blazor application.
- Radzen.Blazor - provides a collection of reusable components and tools for building interfaces with Blazor.

4.1.2 Folder structure

- BlazorWASM: Contains components and Razor pages for rendering the user interface.
- HttpClients: Contains classes and logic for making HTTP requests to the logic tier.
- Shared: Contains shared class structure with the logic and data tiers. This includes models
- and DTOs for data structure.

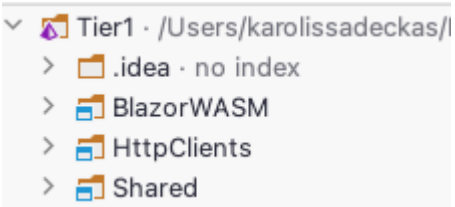


Figure 20

4.1.3 Components

In the BlazorWASM folder, we have implemented a reusable component for improved organisation, maintainability, and readability of our application. This component allows us to easily reuse code.

```
@if (ShowModal)
{
    <div class="modal-background" style="...">
        <div class="modal-box">
            @ChildContent
        </div>
    </div>
}
```

Figure 21

4.1.4 Dependency injections

We used dependency injection in Spring because it allows objects to be provided with their dependencies, improving the reusability, testability, and performance of the system. In figure 22 we are registering a class called “HousingImpl” that implements a service called “HousingInterface” with a dependency injection container. This allows us to use the “HousingImpl” class in our code by requesting it as a dependency (Figure 23).

```
builder.Services.AddScoped<HousingInterface, HousingImpl>();
```

Figure 22

```
@inject HousingInterface HousingInterface
```

Figure 23

4.1.4 Page structure

Each page has its own .razor file, which contains the HTML and C# code necessary to render the page. The .razor file also defines the URL path (Figure 24) for the page, which determines where the page will be accessible in the application.

```
@page "/Login"
```

Figure 24

The C# code for this class, including any variables, methods, and other logic needed for the page to function properly, is written inside the `@Code` block. (Figure 25)

```
@code {
    private List<Agreement> myAgreements = new List<Agreement>();
    private string resultMsg = "";
    private string color = "";
    private bool showAlert = false;
    private AlertStyle alertStyle;

    private async Task ConfirmAsync(long agreementId)
    {
        RespondAgreementDTO dto = await AgreementInterface.RespondToAgreementAsync(
            new RespondAgreementDTO(agreementId, accepted: true, errorMessage: ""));// Task<RespondAgreementDTO>
        if (dto.ErrorMessage.Equals(""))
        {
            resultMsg = "";
            color = "red";
        }

        await UpdateList();

        await dialogService.Alert(message: "Agreement confirmed");
    }

    // Unnecessary code removed for readability
}
```

Figure 25

An HTML part defines the page's appearance and layout and has interactive components such as buttons, (Figure 26) and forms. When a user interacts with these components, they trigger methods defined inside`@Code` block.

```
<button class="applyBtn" @onclick="args => ConfirmAsync(agreement.AgreementId)">Accept</button>
```

Figure 26

4.1.5 Displaying items inside the page

Inside the `@foreach` block (Figure 27), we are displaying all the housing units that were retrieved from the data tier by calling an injected service in the `OnInitializedAsync()` method (Figure 28). This method is called automatically as soon as the page is rendered, allowing the page to display the latest data from the data tier.

```
    @foreach (Housing housing in allHousing)
{
    <div class="card" style="...">
        <div class="col-lg-12 product-title">
            <RadzenText TextStyle="TextStyle.H6" TagName="TagName.H5" Class="rz-color-secondary">House</RadzenText>
        </div>
        <div >
            <div class="row d-flex">
                <div class="col-lg-2 col-lg-">
                    <RadzenText TextStyle="TextStyle.H6" TagName="TagName.H5" class="mb-0">Country</RadzenText>
                    <RadzenText TextStyle="TextStyle.Body2" class="mb-sm-2 mb-lg-0">@(housing.Address.Country)</RadzenText>
                </div>
            // Unnecessary code removed for readability
                <div class="col-lg-2">
                    <RadzenText TextStyle="TextStyle.H6" TagName="TagName.H5" class="mb-0">Post code</RadzenText>
                    <RadzenText TextStyle="TextStyle.Body2" class="mb-sm-2 mb-lg-0">@(housing.Address.PostCode)</RadzenText>
                </div>
            </div>
            <div>
                <button class="applyBtn" @onclick="args => ApplyAsync(housing)">Apply for</button>
            </div>
        </div>
    </div>
}
```

Figure 27

Additionally, we check if the response contains an error message (Figure 28) if it does, we display the error message on the page in red text.

```
protected override async Task OnInitializedAsync()
{
    HousingListDTO dto = await HousingInterface.GetAvailableHousingAsync();
    if (dto.ErrorMessage.Equals(""))
    {
        allHousing = dto.HousingList;
        resultMsg = "";
    }
    else
    {
        resultMsg = dto.ErrorMessage;
        color = "red";
        if (dto.HousingList.Count == 0)
        {
            resultMsg = "There is no available housing.";
            color = "red";
        }
    }

}
```

Figure 28

4.1.6 HttpClients

In this project, we implemented client interfaces (Figure 29) for different client implementations using dependency injection. This approach allows us to achieve loose coupling between the client interfaces and their implementations, which makes it easier to replace one implementation with another if needed. For example, if we wanted to replace an HTTP connection with a gRPC connection, we could do so without any problems because the client interfaces are not tightly coupled to their specific implementations. This makes our code more modular, flexible, and maintainable.

Figure 29

4.1.7 HTTP configuration

The HttpClient class is used to make HTTP requests to the logic tier. In order to configure the HttpClient, we added an instance of the HttpClient to the Dependency Injection container (Figure 30) with the AddScoped method. This instance is configured to use the base URL http://localhost:8081 for all requests.

```
builder.Services.AddScoped(sp :IServiceProvider => new HttpClient { BaseAddress = new Uri("http://localhost:8081") });
```

Figure 30

In all client implementations, we defined a constructor that takes an HttpClient instance as an argument. This allows the HousingImpl class to use the pre-configured HttpClient instance to make HTTP requests.

```
public class HousingImpl : HousingInterface
{
    private readonly HttpClient client;

    2 new *
    public HousingImpl(HttpClient client)
    {
        this.client = client;
    }
    // Unnecessary code removed for readability
```

Figure 31

4.1.8 DTOs

We decided to use DTOs (Figure 32) to transfer data between the presentation and logic tiers because they allowed us to easily transfer all the necessary data between the different layers of the application, and also made it easy to map the data from a DTO to other objects in the application.

```
public class RequestAgreementDTO
{
    public string RefugeeEmail { get; }
    public string HostEmail { get; }
    public Housing Housing { get; }
    public string ErrorMessage { get; }
```

Figure 32

4.1.9 HTTP Implementation

The GetAvailableHousingAsync method (Figure 33) is used to retrieve a list of available housing objects from the database. It returns a HousingListDTO object. The method first sets the authentication header using the JwtAuthImpl.Jwt token. It then sends an HTTP GET request to the /api/housing endpoint. The response is read as a string and checked to see if it has a success code. If the response is not successful, an exception is thrown with the error message. The response is then deserialized into a HousingListDTO object using JSON serialisation. Finally, the HousingListDTO object is returned.

```
public async Task<HousingListDTO> GetAvailableHousingAsync()
{
    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue( scheme: "Bearer",  parameter: JwtAuthImpl.Jwt);
    HttpResponseMessage message = await client.GetAsync( requestUri: "/api/housing");
    string result = await message.Content.ReadAsStringAsync();

    if (!message.IsSuccessStatusCode)
    {
        throw new Exception(result);
    }

    HousingListDTO housingCreated = JsonSerializer.Deserialize<HousingListDTO>(result, new JsonSerializerOptions
    {
        PropertyNameCaseInsensitive = true
    })!;

    return housingCreated;
}
```

Figure 33

4.2 Logic tier

4.2.1 Libraries

In addition to Spring framework we used these libraries:

- com.google.protobuf:protobuf-java - library for efficient serialisation and deserialization of data.
- io.github.lognet:grpc-spring-boot-starter - Spring Boot starter for using gRPC, a high-performance remote procedure call framework.
- Org.mockito:mockito-core - mocking library for unit tests in Java.
- org.projectlombok:lombok - library that provides compile-time code generation for common boilerplate code in Java.
- io.jsonwebtoken:jjwt - library for creating and verifying JSON Web Tokens.
- javax.xml.bind:jaxb-api - Java API for converting between XML and Java objects.

4.2.2 Controllers

All controller classes, including the HostController shown in Figure 34, are annotated with both @RestController and @RequestMapping. This makes the class an entry point for the logic tier and an endpoint that is mapped to specific URL patterns, allowing it to handle incoming requests for those URLs.

```
@RestController
@CrossOrigin
@RequestMapping("/api")
public class HostController {
        // class content
}
```

Figure 34

In order to follow the SOLID principle, the HostController class has a constructor that accepts a HostInterface object as an argument. This object is then stored in a field named hostLogic and can be used throughout the class. This is made possible by dependency injection, which allows an object to be passed to a class through its constructor, rather than being created inside the class. In this case, the HostController class is being injected with a HostInterface object, which it can use to access the host logic. This allows the controller to handle incoming requests and perform the appropriate actions without having to know the details of how the host logic is implemented.

```
HostInterface hostLogic;

public HostController(HostInterface hostLogic) {
    this.hostLogic = hostLogic;
}
```

Figure 35

The getHost method (Figure 36) in the HostController class is an endpoint that is mapped to the URL pattern /host/{email} using the @GetMapping and @PathVariable annotations. This method uses the value of the email path variable to call the hostLogic object and retrieve the host information for the specified email address. If the operation is successful, the method returns the host information with an HTTP OK status code. If any unexpected errors occur, the method returns an HTTP INTERNAL_SERVER_ERROR code.

```
@GetMapping("/host/{email}")
    public ResponseEntity<HostDTO> getHost(@PathVariable("email") String email){
        try {
            HostDTO host = hostLogic.getHostById(email);
            return new ResponseEntity<>(host, HttpStatus.OK);
        }

        catch (Exception e)
        {
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }


}
```

Figure 36

4.2.3 Business logic

The HostLogic class (Figure 37) implements the HostInterface interface and is called from the HostController class. To continue to follow the SOLID principle, we are using Spring-provided dependency injection in the constructor to access the required functionality without having to know the details of its implementation.

```java
@Service
public class HostLogic implements HostInterface {

    private HostCommunicationInterface hostCommunicationInterface;
    private HousingCommunicationInterface housingCommunicationInterface;
    private AgreementCommunicationInterface agreementCommunicationInterface;
    private PasswordEncoder passwordEncoder;

    public HostLogic(HostCommunicationInterface hostDAO, HousingCommunicationInterface housingDAO,
                     AgreementCommunicationInterface agreementCommunicationInterface,
                     PasswordEncoder passwordEncoder) {

        this.hostCommunicationInterface = hostDAO;
        this.housingCommunicationInterface = housingDAO;
        this.agreementCommunicationInterface = agreementCommunicationInterface;
        this.passwordEncoder = passwordEncoder;
    }
```

Figure 37

This method (Figure 38) makes several calls to the data tier in order to successfully delete a host's account. Firstly, it checks if the host exists by using the hostCommunicationInterface to retrieve the host based on their email address. If the host is not found, the method returns a HostDTO object with an error message.
Next, the method retrieves a list of agreements and housing listings associated with the host using the agreementCommunicationInterface and housingCommunicationInterface interfaces. It then checks if any of the agreements are accepted and, if so, returns a HostDTO object with an error message. If no accepted agreements are found, the method proceeds to delete all pending agreements and removes all housing listings associated with the host.
Finally, the method deletes the host entity and returns an empty HostDTO object to indicate that the host's account has been successfully deleted.

```java
@Override
public HostDTO deleteAccount(String email) {

    Optional<Host> existing = hostCommunicationInterface.getHostByEmail(email);
    if (existing.isEmpty())
    {
        return new HostDTO( host: null,  errorMessage: "Host with email: " + email
                + " not found, and therefore unable to be deleted.");
    }

    // If host is existing - get lists of agreements and housings.
    List<Agreement> agreementList = agreementCommunicationInterface.getAgreementsByHostId(email);
    List<Housing> housingList = housingCommunicationInterface.getAllHousingByHostId(email);

    // if any agreement is accepted - do not delete, instead return a dto with error message.
    for (Agreement a : agreementList) {
        if (a.isAccepted()) {
            return new HostDTO( host: null,  errorMessage: "Unable to delete. Ongoing agreement(s) found.");
        }
    }

    // remove all pending agreements
    for (Agreement a : agreementList) {
        agreementCommunicationInterface.deleteAgreement(a.getAgreementId());
    }

    // remove all housing
    for (Housing h : housingList) {
        housingCommunicationInterface.removeHousing(h.getHousingId());
    }

    // delete host entity
    hostCommunicationInterface.deleteAccount(email);

    return new HostDTO( host: null,  errorMessage: "");
}
```

Figure 38

4.2.4 GrpcClient

When a gRPC managed channel is created, we want to reuse it as much as possible for faster communication. To accomplish this, we implemented a singleton class (Figure 39) that will create the managed channel once and then use it across the logic tier.

```java
@Service
@Scope("singleton")
public class Channel {


    private ManagedChannel managedChannel;
    private HostGrpc.HostBlockingStub hostStub;
    private RefugeeGrpc.RefugeeBlockingStub refugeeStub;
    private HousingGrpc.HousingBlockingStub housingStub;
    private AgreementGrpc.AgreementBlockingStub agreementStub;

    public Channel() {
        createChannel();
    }

    public void createChannel() {
        this.managedChannel = ManagedChannelBuilder
                .forAddress( name: "localhost",  port: 8084)
                .usePlaintext()
                .build();

        resetStubs();
    }
```

Figure 39

For gRPC stubs, we create a new one the first time the stub is called. After that, we reuse the same stub (Figure 40). In case of failure, we can reset the stub, which will cause a new stub to be created the next time it is called.

```
private void resetStubs() {
    hostStub = null;
    refugeeStub = null;
    housingStub = null;
    agreementStub = null;
}

public HostGrpc.HostBlockingStub getHostStub() {
    if (hostStub == null) {
        this.hostStub = HostGrpc.newBlockingStub(managedChannel);
    }
    return hostStub;
}
```

Figure 40

4.2.5 GrpcCalls

In this method (Figure 41), we receive a host object and convert it (Figure 42) to a GHost (gRPC generated host). We then call a certain stub with the GHost as argument, and receive a response. If we don't receive a response within 1 second, we reestablish the connection and return null. Otherwise, we convert the GHost back to a regular Host class and return it.

```
@Override
public Host createHost(Host host) {
    try {
        GHost request = GrpcConverter.hostToGrpc(host);
        GHost response = channel.getHostStub().withDeadlineAfter
                ( duration: 1, TimeUnit.SECONDS).createHost(request);
        if (response == null) {
            return null;
        }
        return hostFromGrpc(response);
    } catch (StatusRuntimeException e) {
        reestablishConnection();
        return null;
    }
}
```

Figure 41

```
public static GHost hostToGrpc(Host host) {
        return GHost.newBuilder()
                .setFirstName(host.getFirstName())
                .setEmail(host.getEmail())
                .setPassword(host.getPassword())
                .setGender(String.valueOf(host.getGender()))
                .setNationality(host.getNationality())
                .setMiddleName(host.getMiddleName())
                .setLastName(host.getLastName())
                .setDateOfBirth(dateToGrpc(host.getDateOfBirth())).build();
}
```

Figure 42

4.3 Data tier

4.3.1 Libraries

- The org.postgresql - the PostgreSQL JDBC driver, which allows the application to connect to a PostgreSQL database.
- The io.github.lognet - gRPC Spring Boot starter, which allows the application to use gRPC as a communication framework.
- The com.google.protobuf - the Protocol Buffers Java library, which allows the application to use Protocol Buffers for efficient serialisation of data.
- The com.h2database - the H2 in-memory database, which allows the application to run tests that use a temporary, in-memory database.
- The org.projectlombok - the Lombok library, which allows the application to use Lombok's annotation processors to reduce boilerplate code.
- The org.springframework.boot - the Spring Data JPA module, which allows the application to use the Java Persistence API (JPA) for data persistence and access.

4.3.2 Injection

To follow the SOLID principle in the data tier we are also using dependency injection.

```
@GRpcService
public class HostServiceGrpcImpl extends HostGrpc.HostImplBase {

    @Resource
    HostDaoInterface hostDaoInterface;
```

Figure 43

4.3.3 Grpc implementation

In this class (Figure 44), we extend HostGrpc.HostImplBase which is generated through proto files. This provides methods that can be called from the logic tier (client).

```
@GRpcService
public class HostServiceGrpcImpl extends HostGrpc.HostImplBase {
```

Figure 44

In this class (Figure 44), we extend HostGrpc.HostImplBase which is generated through proto files. This provides methods that can be called from the logic tier (client). The method createHost (Figure 45) is called from the logic tier with a provided request and responseObserver. The responseObserver allows us to know who to call back to when we have processed the request.
Once the request is received, we call the DAO (Data Access Object) with the provided information. With the received information, we send a response back to the observer and let them know that we have completed processing the request.

```
@Override
public void createHost(GHost request, StreamObserver<GHost> responseObserver) {

    Host convertedRequest = GrpcConverter.hostFromGrpc(request);
    Host dataResponse = hostDaoInterface.createHost(convertedRequest);
    GHost response = GrpcConverter.hostToGrpc(dataResponse);
    responseObserver.onNext(response);
    responseObserver.onCompleted();
}
```

Figure 45

4.3.4 Database

@Entity and @Table (Figure 46) are annotations used in Java Persistence API (JPA) to specify that a class should be mapped to a database table. The @Entity annotation indicates that the class is an entity, which represents a table in a database. The @Table annotation specifies the name of the table that the entity should be mapped to. The email variable is annotated with @Id, which indicates that it is the primary key of the "host" table. This means that the email will be unique for each row in the table, and it will be used to identify and retrieve specific rows in the table.
The housing variable is annotated with @OneToMany and mappedBy = "host", which indicates that the "host" table has a one-to-many relationship with the "housing" table. This means that each host can have multiple housings, but each housing can only have one host.

```java
@Entity
@Table(name = "host")
public class Host {

    @Id
    private String email;

    @Column(name = "fName")
    private String firstName;

    @Column(name = "mName")
    private String middleName;

    @Column(name = "lName")
    private String lastName;

    @Column(name = "password")
    private String password;

    @Column(name = "nationality")
    private String nationality;

    @Column(name = "gender")
    private char gender;

    @Column(name = "dob")
    private LocalDate dateOfBirth;

    @OneToMany(mappedBy = "host")
    private List<Housing> housing;
```

Figure 46

5. Testing

5.1 Black box - Integration

For the presentation tier, we used black box testing based on use cases to ensure that the system functions as expected. This testing method allowed us to verify the integration between all three tiers, and confirm that they are working together correctly. The result overview can be seen in the table below (Figure 47), and the majority of tests were passed. Detailed testing details can be found in the appendix.

Overview:

Use case name:	Passed:	Failed:
Register(Refugee)	10	0
Register(Host)	12	0

Use case name:	Passed:	Failed:
Log in: (Refugee/Host)	6	0

Use case name:	Passed:	Failed:
Apply for housing: (Refugee)	5	1

Figure 47

5.1.1 Logic tier

For the logic tier, we used mockito to perform unit testing on individual business logic methods. This allowed us to test the functionality of each method in isolation, and verify that they are working correctly. All expected tests were passed, with a total of 43 tests. As for class coverage, 100% of classes were tested with 89% of lines covered.

Figure 48

In method (Figure 49), we are testing if it is possible to delete a host account when the host has added a few housings but none of them have agreements. Using the mockito library, we are able to simulate the response of the data tier. This allows us to test the methods in isolation.

In this example (Figure 49), we use when() to specify the behaviour of a mock object. The when() method is called on the mock object and passed to the method to be mocked, along with the expected return value. This allows us to specify the argument that must be provided to the mocked method in order for it to return the desired value using .thenReturn().

As for verify(), we use it to verify that a specific method was called on a mock object.

```
@Test
@DirtiesContext
void deleteHost_whenFewActiveAgreementsButNonAccepted_andFewHousings()
{
    // setup
    housingList.add(availableHousing1);
    housingList.add(availableHousing2);
    agreementList.add(new Agreement( agreementId: 1L,host1,availableHousing1,refugee1, isAccepted: false));
    agreementList.add(new Agreement( agreementId: 2L,host1,availableHousing2,refugee1, isAccepted: false));

    when(hostCommunication.getHostByEmail(host1.getEmail())).thenReturn(Optional.of(host1));
    when(agreementCommunicationInterface.getAgreementsByHostId(host1.getEmail())).thenReturn(agreementList);
    when(housingCommunicationInterface.getAllHousingByHostId(host1.getEmail())).thenReturn(housingList);
    // test
    HostDTO result = underTest.deleteAccount(host1.getEmail());
    // verify
    verify(hostCommunication).getHostByEmail(host1.getEmail());
    verify(agreementCommunicationInterface).deleteAgreement( agreementId: 1L);
    verify(agreementCommunicationInterface).deleteAgreement( agreementId: 2L);
    verify(housingCommunicationInterface).getAllHousingByHostId(host1.getEmail());
    verify(agreementCommunicationInterface).getAgreementsByHostId(host1.getEmail());
    verify(housingCommunicationInterface).removeHousing( housingId: 1L);
    verify(housingCommunicationInterface).removeHousing( housingId: 2L);
    verify(hostCommunication).deleteAccount(host1.getEmail());
    assertEquals( expected: "",result.getErrorMessage());
}
```

Figure 49

5.1.2 Data tier

As for the data tier, we used an H3 database to test the additional CRUD (create, read, update, delete) methods that we added to the system. This allowed us to ensure that the data tier is able to properly handle the storage and retrieval of data from the database.

6.Results and discussion:

The RefugeeBnB system works as intended, both hosts and refugees are able to interact with the system once their account has been created and added to the database. This can be seen in the testing, that the functionalities that make the system functional, work. When interacting with the system, we as a team did not see any major flaws, but one. System creates users as the first option (Danish), without giving an error that nationality has not been selected, but they have selected who they are from an option of refugee and host. This also seems to be happening to the genders. Despite that, the system functions just well and is ready for use.

7. Conclusion

The lack of the tools for refugees running from the Ukraine war to find accommodation has provided an opportunity to create a web application that would be a potential solution for the current crisis. The application would provide both, for volunteers a possibility to contribute by accommodating people in need and for refugees finding a temporary place to get back on their feet.

The main goal was to implement a system that provided secure access to both hosts and refugees as well as a possibility for future expansion of the system. Considering the possible needs to accommodate systems such as mentioned, a N-tier architecture was implemented with authorization for security, a PostgreSQL database for storing and unique middleware for tiers to communicate between each other.

In conclusion, the final product has a solid implementation of the system that works as intended and has a potential to help improve the current crisis. While not all requirements from the product owner were able to be implemented before the deadline, the critical requirements were. This makes the system usable to a satisfactory degree.

8. Project Future

In the future, it would be beneficial to implement UN representatives into the system to verify the validity of both refugees and hosts. This would help to prevent scams and ensure the safety of all parties involved. In addition, implementing a chat system would allow hosts and refugees to communicate more easily and facilitate additional assistance, such as transportation.

A major improvement to the system would be to strengthen its security by using HTTPS, so refugees and users would have a more secure connection .

Creating job opportunities for refugees, this feature would let refugees list their skills in the website, job providers could then offer work for refugees and maybe housing with it as well.

Making the website mobile friendly.

Another potential improvement would be to add the ability for hosts to upload pictures of their rooms, which would give refugees a better idea of what to expect. A language swapping tool could also be useful, as not all refugees may speak English. This would make it easier for refugees to navigate the website in their own language. Overall, these improvements would enhance the user experience and increase the effectiveness of the system.

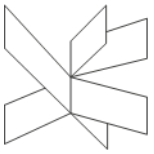
9. Sources of Information

Conklin, L. (no date) Threat Modelling process, Threat Modeling Process. Available at: https://owasp.org/www-community/Threat_Modeling_Process#complementing-code-review.

threatmodeler (2019) INFORMATION SECURITY OBJECTIVES: 8 TIPS FOR CISOS. Available at: https://threatmodeler.com/identifying-security-objectives/.

[Coulouris et al.] George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair: Distributed Systems — Concepts and Design. Fifth Edition. ISBN 978-0-13-214301-1

Kurose, J.F. and Ross, K.W. (2022) Computer networking: A top-down approach, Amazon. Pearson Education Limited. Available at: https://www.amazon.com/Computer-Networking-Global-James-Kurose/dp/1292405465.



Software Technology Engineering

Semester 3 - Class X

Refugee B&B
Process Report

Group 11
Christian Hougaard Pedersen (315269)
Nina Anna Wrona (315202)
Justina Ieva Bukinaitė (315199)
Karolis Sadeckas (315225)
Ignas Druskinis (315244)

Supervisors:

Joseph Chukwudi Okika

Jakob Knop Rasmussen

Date of completion:

15-12-22

The number of characters in the main text:

29.858

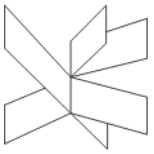
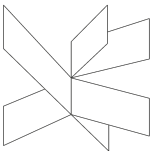


Table of Contents



Introduction

This documentation will cover how the 3rd semester project was implemented, mainly focusing on the management and dynamics of the group as well as the personal reflections that each of us experience having to work closely with four other people. The group 11 consists of Nina Anna Wrona, Karolis Sadeckas, Ignas Druskinis, Christian Hougaard Pedersen and Justina Ieva Bukinaite.

As project discussion started early in the semester, when the goal was set, we followed Unified process (UP) in order to organize our time while we are attending lectures. When the Elaboration phase was started, we had integrated SCRUM, in order to have good daily management.

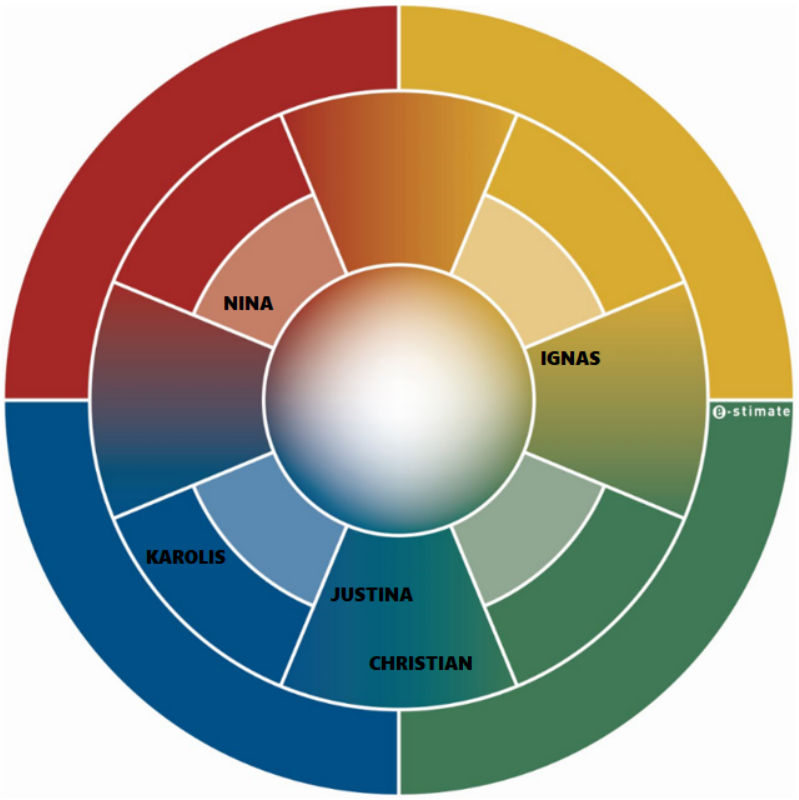
Both UP and SCRUM have been used prior to this semester, therefore the implementation of both has not been too challenging, however having a new group member as well as trying to implement the usage of Jira, software for managing SCRUM, added new challenges that will be expanded on later in the documentation.

Although we had no company to answer to, having a SCRUM master and Product owner has improved our time and decision management. Following that our supervisor Joseph Chukwudi Okika has made himself available both online and physically and we felt that we got the support that we needed.

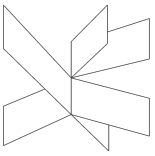
Group Description

The formation of this group has had a natural progression throughout the last few semesters where although we lost one member, we added two more that had the characteristics that we were missing and created a positive influence on the group work.

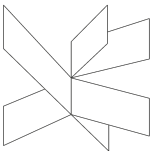
Group 11, otherwise known as “Beaver Squad” is made up of 5 people and although we have three different nationalities, we all come from the same part of the world. Therefore, we always felt that in a very general sense cultural backgrounds, or age differences, have not interfered with our work and just brought us more laughter. That being said, we were three Lithuanians among a Polish and a Danish person and although unconsciously done there may have been some conversations taken in Lithuanian instead of English, creating a massive language barrier. Despite that, since events like that happened rarely and in mostly social settings, our work process was not interrupted.



Although cultural differences were not a huge influence on our project process, having different personalities did. During the second semester we all participated in estimating personal profiles and how to interact with people who have different



profiles. Understanding each of our personalities has helped extensively to adjust the management of the project and what roles should be taken by each of us. As is shown in color team wheel, our different team personalities cover most of the colors and having Ignas joining our group this semester we have a well-rounded and colorful group. Karolis, Justina, and Christian by being blue, we always brought the precision and ensured quality, Nina having very red qualities, was always able to lead us to the right direction and do not focus on the details and lastly Ignas having both green and yellow contributed with new ideas as well as always made everyone feel comfortable and positive. However, as things are not black and white in the world, neither are people. We all tried to contribute both to achieve a best work environment as well as feeling comfortable with each other. Knowing each other's strengths and weaknesses has helped to maximize our productivity.

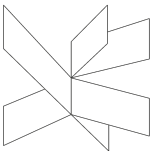


Project Initiation

When the group was formed a group contract was signed by all group members to ensure that everybody participates in SEP3 development. The contract included specifics of how to go about group work assignments and mainly the project development for the third semester as well as the consequences if the rules are not followed. As some of the group's members experienced prior problems when the contract was not taken seriously, it was essential to make sure that everybody understood the importance of it. That has helped us develop a good working ethic throughout the semester.

After the contract was signed, we received instructions about what is asked to fulfill the requirements for the SEP3 project. One of the main conditions was that the project would be in some way related to one or more of the 17 Sustainable Development Goals by the United Nations. The initial project idea was written down in order to be presented to the supervisors with a short description of what it is about and how it may help further one of the UN goals. When we were presenting it to our supervisor, the possible difficulties and risks were discussed so we would fully understand what is expected of us and so the project idea was approved.

Additionally, we were planning to use SCRUM methodology for management, so we shared the roles between each other of SCRUM master, Product Owner, and the developers. As we were required to also implement the Unified Process framework, we derived a schedule for the semester to prevent falling behind the deadline. This was a helpful tool to come back to occasionally during the lecture period and on a daily basis during the project period and has helped us to keep track of the progress. That being said, although we are familiar with both SCRUM and UP, they are still relatively new to our group and therefore both the



timeline and the product backlog have been adjusted according to the circumstances.

Project Description

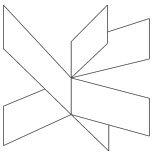
Project description phase started right after the project was approved. We focused on expanding and specifying the idea of Refugee B&B, a website that will allow refugees to gain access to free accommodation and establish an opportunity for easier integration.

While expanding on the background description we developed a well rounded idea of how the product should work. We also had to consider possible limitations such as for example "*We will focus only on European countries*". Also, possible internal risks had to be assessed in advance to be able to prioritize tasks to avoid them negatively influencing the product in the future. Following the background description, we chose to start on building a detailed product backlog, to adjust the initial timeline.

Our goal was to implement a web application where volunteers would be able to post possible housing opportunities and approve or deny their applications. Followed by refugees, who would be able to see the listings of available accommodations and apply. When the application is approved a contract is generated by the system for documentation.

As a refugee, I would like to be able to put my lifestyle choice or/and religion on my profile, in order to minimize conflicts between me and the host.	REF-14	Done	Medium	Karolis Sadeckas	Sprint 3+4
As a refugee I want to be able to include my family into the housing agreement in order to let the host know how many people I want to bring with me.	REF-15	Done	Medium	Justina Ieva Bukinaite	Sprint 3+4
As a host I want to know before signing the agreement how many people the refugee is bringing with him, in order to decide if I want to offer them a place to stay	REF-16	Done	Medium	Christian Hougaard Pedersen	Sprint 3+4
As a UN representative, I would like to be able to verify potential hosts and refugees, in order to minimise the chances of misuse of the system.	REF-17	To Do	Medium		
As a refugee I would like to be able to chat with the host in order to discuss the rental agreement.	REF-18	To Do	Medium	Karolis Sadeckas	
As job provider I would like to apply for refugee Bnb, in order to be able to offer refugees a job.	REF-19	To Do	Lowest	Ignas Druskinis	
As a refugee I want to list my skills in order to get a job offer.	REF-20	To Do	Lowest	Nina Anna Wrona	
As a job provider I would like to see all refugees that are seeking for a job and their skills in order to see if I can offer them a job	REF-21	To Do	Lowest	Karolis Sadeckas	

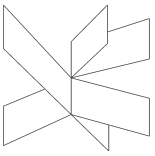
Furthermore, for the possibility of the product being valid in real life scenario, we were considering having an external person that may be a UN representative



that would validate the identification documentation when both hosts and refugees are creating a new account.

Lastly, we were considering implementing a chat system that would connect the hosts and refugees personally. Along with a possibility for a 4th actor to join the website as a job provider and put posts about available job positions.

Although some requirements were not implemented due to time constraints, the main idea for the project was achieved.



Project Execution

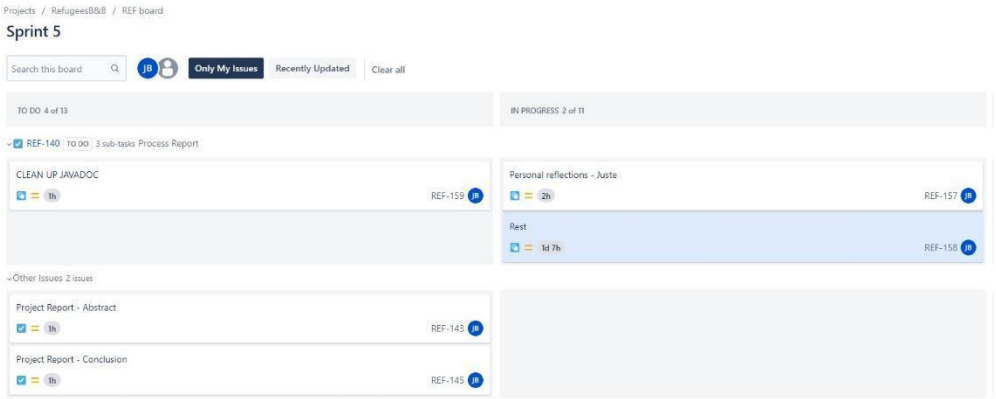
To conclude the inception phase, we went through a further approval process after the project had a more precise definition, estimated timeline and possible internal risks in the project description documentation.

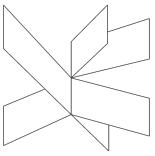
As we entered the Elaboration phase our main goal during this period became going more into detail for the infrastructure and functionality, so that when the construction phase begins, we have a clear understanding of what needs to be done. During this time, we have made use cases, so when implementing the program, we would understand a clear path the user should go through. That was followed by a building of a domain model and architectural diagram, in order to have barebone structure for further architectural development in the design section.

A bit into the Elaboration phase SCRUM framework was introduced as well as a project tracking software Jira. In a prior semester we were successful with following SCRUM, and we felt that it had improved our workflow substantially, therefore we were excited to introduce it again. However, one issue we have faced during SCRUM is using Jira. Although it may have been a very useful tool in the long run, as we started using it only when SCRUM started, for the first couple of Sprints both the SCRUM master and developers were struggling to find their way around the website. That may have been a small setback in the start causing us to reconsider using Jira, but our SCRUM master did her best to pick up the skills and so we decided to continue using it.

Furthermore, although struggling with Jira for SCRUM management, we have otherwise followed the framework guideline as best as we could depending on the circumstances. We overall had 5 Sprints that were always scheduled to take 3 days, where the sprint starts and ends at noon. Our Sprints were led by Nina, the SCRUM master, where at the start of every Sprint we had a Sprint planning

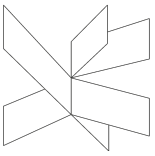
meeting, a daily SCRUM and at the end we participated in the retrospective and review meetings. During the planning meeting Karolis, our Product Owner, made a short appearance to introduce us to new backlog items that are supposed to be taken for the new Sprint. Additionally our Product Owner always made sure to come for review meetings, when he would mark the previously taken User Stories as completed or reassign them for the following Sprint. Later on, we would spend time building upon our class diagram and making a Sprint task backlog and to end the meeting we would divide the tasks equally. During daily SCRUMS we went through our progress to make sure everybody is on track. Sprint retrospective meetings were a quick sit-down to reflect on the workflow and discuss if there are any aspects of daily work that could be done better. We used the “Start/Stop/Continue” framework, in order to both comment on what is going well and what could be improved on. Lastly, in the Sprint review meeting, we present the progress to the Product Owner where the backlog items for that Sprint were either marked as done or otherwise. In order to keep track of the tasks being done, we used Jira, where each Sprint task was assigned to a person. During the Sprint, when we would start the task, we would drag the task into the “IN PROGRESS” section and when it’s done to the “DONE” section as shown in the picture. After each Sprint, the SCRUM master generated the burndown chart. Although Jira was a challenging tool to use at first, following SCRUM was a helpful framework to keep the project on track.





During the Elaboration phase we mostly focused on building a skeleton for our system, where we also implemented a proof of concept and a basic class diagram. When the analysis and design was mostly documented we were able to start on the Construction phase where the implementation became the biggest focal point and documentation was mostly used for reference and slight editing. Lastly, the last Sprint and a half became the Transition phase for our project since we have stopped taking new backlog items and concentrated on fixing the bugs in the code and wrapping up documentation.

During the time where project work was done full-time, we met almost on a daily basis or otherwise spent hours over zoom discussing the code related issues. Furthermore, the SCRUM and UP were followed thoroughly by our team, helping us develop as good of a result as we could have done in the best circumstances. And we feel that the project execution has been overall a successful and educating experience.



Personal Reflections

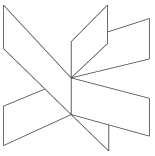
Nina Anna Wrona

This Project Period was really hard for me. I took the role of a SCRUM Master and I was doing all the documentation, task dividing and keeping everyone in a good mood. It was really demanding. We have managed to hang out once or twice, however I usually wasn't lifting up the mood in the room. I was often moody and hysterical. I reflected on that and came to the conclusion that before I come to the meeting I should do yoga, which always helps me relieve stress, anxiety and negative emotions, so that I can always greet and treat my fellows with kindness and respect.

I have put on myself a task of using a specialized SCRUM tool called Jira, which was difficult to figure out at the beginning, but helped a lot in the documentation field as well as real-life time tracking. I believe if we would be able to learn this tool before the project period started - it would be much easier to navigate. Nevertheless I am very content that we have learned this tool, I am sure we will use it in future projects.

The mistake I have made was to help others with their tasks instead of focusing on completing the ones assigned to me. This way part of my group was finished with their tasks earlier, while I had to move mine to the following Sprint. I know now that I need to learn how to plan my time better and not resign from helping others, but maybe have in mind that I can also always ask for help.

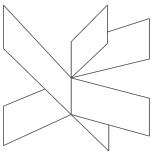
One of the biggest challenges this semester that I have set for myself was not to be late for the group meetings, in order to respect the cultural map, where most of this group leaned towards linear time in Scheduling.



I was usually just on time, 5 minutes earlier or 5 minutes late. However, compared to the previous semester, I have improved, which I am really proud of.

This semester I have relied on both external and internal motivation. There were times when curiosity and joy led to my action of willingly participating, however there were also times when the stress of the deadline and the thought of disappointing others inclined my work.

In conclusion, I see there is a room for improvement, but on the other hand I am really proud of Karolis, Christian, Justine and Ignas, because I saw how hard they have all worked and how well they have managed the stress as well as ups and downs of this project.



Karolis Sadeckas

Working on this project with a group of five people was a new experience for me. This group formed from last semester's project group with the addition of a new member, Ignas, who brought a fresh perspective and new ideas to our work.

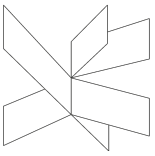
At first, I was a little unsure about how the new member would fit in with the rest of the group, but I quickly realized that he was a great fit for our team. He was able to quickly adapt to our working style and contributed valuable insights to our discussions.

One of the key factors that helped us to communicate more effectively and be more organized was the use of the Scrum framework. By following the Scrum process, we were able to break our project down into smaller, manageable tasks, and assign roles and responsibilities to each team member. This helped us to stay on track and ensure that everyone's ideas were heard and considered.

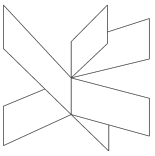
One thing that I particularly enjoyed about working with a larger group was the ability to tackle problems together and see others perspectives on them. This allowed me to learn to look at problems from different points of view, which is a valuable skill to develop.

Overall, I found that working with a larger group was both challenging and rewarding. It required us to communicate more effectively and to be more organized in order to ensure that everyone's ideas were heard and considered. But in the end, the extra effort was worth it, as we were able to produce a high-quality project that we were all proud of.

In conclusion, working with a larger group taught me the importance of effective communication and collaboration in achieving success. The use of the Scrum



framework was a key factor in our success, and I look forward to future opportunities to work with larger groups and to continue to develop my skills in these areas.



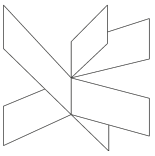
Justina Ieva Bukinaitė

Project work

In the 3rd semester the main requirement for the project was to relate the product to one of the 17 UN GOALS. As the situation in Ukraine was on everybody's minds we decided to implement something that may be useful for people running away from war. Current situation for finding accommodation for people coming from Ukraine is limited and mostly done through social media portals, therefore our idea was to create a web application where people would be able to post about possible housing opportunities and the refugees would be able to apply. We chose the name Refugee B&B as we felt that it would be catchy and easily memorable even for people who lack english skills.

To define the problem, we looked at the relevant sources that spoke about the situation for refugees right now, meaning how they were dealing with finding a place to leave and what tools they had in countries like Lithuania and Poland. Our main goal was to provide a website where refugees could register and apply for housing no matter the country they are in. That being said as we had to consider time limitations we mainly focused on Europe. However the initial expectations, in my opinion, have been too big, since we have underestimated the workload it would take to build the infrastructure we would have wanted.

For managing the project we used SCRUM and Unified Process and that has been a positive experience for the work process. Having the constant meetings in order to follow SCRUM really improved our communication and ability to stick to the deadlines that we set. However, following UP has been a more challenging experience as we were used to having more hand-ins throughout the semester that were more related to the analysis. Whereas in this semester we only had to hand in some core parts of analysis, that had led us to postponing the use case diagram. That lead to more work having to be done



when implementing the project. Also as we used SCRUM we decided to try working with Jira, which was a recommended tool to manage SCRUM. In my opinion, we should have implemented Jira into our daily work prior to starting SCRUM in order to practice more how the tool should be used.

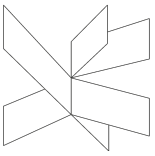
To conclude, my experience with the project process has been positive and I feel we have done an overall good job on our project and I am happy with the results that we have. We followed both SCRUM and UP to the best of our ability and the main takeaway regarding these frameworks is to do an initial planning for following them in a more detailed way.

Group work

My group was initially formed in the 1st semester, and we continued working together in both 2nd and 3rd semesters adding two more people. We always had a good dynamic and adding another two members in my opinion has only improved our relationships. As in the 1st semester we were unsuccessful in enforcing the contract, we really wanted to focus on following the rules this semester. In my opinion the group members have done a good job at generally participating and working on the project, according to the contract that was signed.

Moreover, in the contract we have a clause mentioning that if anybody needs help with the subjects they should come to one of us for help. Unfortunately, I felt that some group members didn‘t use that opportunity enough and it interfered with other group members' load.

Since I didn‘t have roles of either SCRUM master or Product Owner, my responsibilities were mainly to contribute as a developer meaning, I would take more coding and focus more on writing the documentation rather than leading and organizing. For example, I have taken on the task of writing the process report, which has been a big challenge for me. That being said, I still feel that

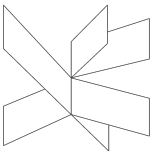


my coding skill could be improved and in the future I could put in more work when working on the lecture material as well as asking for more help.

Furthermore, our group has implemented the E-stimate personal profiles into our daily routines meaning people who were more detail oriented were focusing on fixing or testing the code and those who had leadership qualities took upon a more of a management role. That has helped us to be motivated as we were undertaking the tasks that fit each of us the best. Also, since our group is multicultural as well as multigenerational, it had a positive influence in our workflow. Having people come from different generations and countries gave us a broader understanding of how the product should work. For example some of us came from Easter Europe where the current situation in Ukraine is more applicable as both Lithuania and Poland have been very closely working with the refugees. Whereas Denmark although, also taking in refugees has been having more of a back seat due to geographical differences. Therefore, being from Easter Europe, we had a more personal perspective of what the refugees would expect, whereas Denmark focused more on the general influence on the world of refugees.

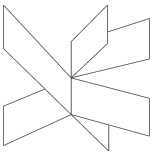
Lastly, I feel that we had beneficial support from our supervisor, as Joseph has always been available when we contacted him and have given us a constructive response to our questions. However, in my opinion we should have put more effort into organizing regular meetings, because sometimes, we weren‘t sure about certain topics. For example, as the system had been tested, we were not sure what it required and even though we had contacted the supervisor via email, to give us some direction, I felt we could have had a full meeting for a more detailed answer.

I think my main takeaway from working with my group has been the importance of communication, meaning if there are any issues between one another we



should speak about it. The lack of communication could cause a detrimental impact on the project and although it may have not interfered with it this semester in the future we should speak out if there are subjects that we are struggling with.

In conclusion, I am happy to be part of our team since it has been a very educational experience for both when working with the implementation of the code and participating in a 5 people group work and using frameworks like SCRUM and UP. However, for future teamwork improvement we should speak out more about teammates struggling with certain subjects, so that it would not create a bigger dent into our final results than it already did.

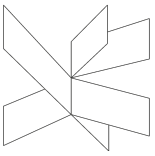


Christian Haugaard Pedersen

Just as in the previous two semesters, working on the semester project has been a really nice experience from which I have learned a lot.

Before starting this semester, the three international Software Engineering classes in our semester were merged into two, meaning that we all got new classmates. Luckily, Nina, Karolis, Justina and I made a request to the administration to be put in the same class as we would like to work together on the semester project - and that request was fulfilled.
One day, Ignas approached us with a proposition for him to join our group, and even though none of us knew him we accepted, making us a group of five people.

According to the E-stimate Personal Profiles we made on our second semester, Ignas is primarily of the yellow type, which has been really beneficial for the group, as the rest of the group consists of three people whose types are primarily blue, so having a group member of another personality type has filled in a gap because of the different ways of working and approaching a problem. As stated above, Justina, Karolis, and I are all primarily blue, which can sometimes lead to us spending a lot of time discussing the problem at hand in order to make sure we get it right. This can lead to some time being wasted, that could be spent better on actually working on the project. Luckily, Nina, whose primary personality type is red, has again this semester, been really great at breaking up these discussions in order for us to continue working in as efficient a manner as possible.

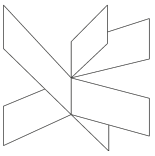


Of course, this had the effect of changing the combination of nationalities in the group, as Ignas, like Justina and Karolis, is Lithuanian. Even though we now had 3 Lithuanians in the group, all communication was still done in english.

We have not experienced any major cultural conflicts, probably because we are now in our third semester, which means that all of us have some previous experience working with people of other cultures.
One issue, which could have potentially caused problems is the age difference present in the group. The youngest person in the group, Nina, is 19 and the oldest, me, is 30 which could easily cause conflicts along the way, coming from both directions. Luckily, it has never escalated further than light-hearted teasing.

As none of us are native english speakers, some small issues have occurred regarding communication, as it can sometimes be hard to get your point across when speaking a second language. But seeing as, at least the four original members of the group, also worked together last semester, and therefore know each other quite well by now, this has not caused any major conflicts.

Like last semester, we have used SCRUM and Unified Process as the general frameworks for managing the work-flow of the project. To me, this has been really helpful, as I thrive the most when all tasks are structured and split evenly between the group members, so everybody knows exactly what to do themselves, and what the rest of the group is doing.
As a new thing this semester, we tried using Jira to manage the work-flow, to varying degrees of success. As we have never used the system before, it has caused some frustrations to figure out how it works.

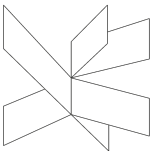


Ignas Druskinis

This semester has been quite interesting for me. I ended up in a different class, and my previous semester team mates were in a different class. I knew there was an option to write an email and be moved to a different class, so I would have a guaranteed spot in a good group. But I found out about that option quite late, and felt like exploring new things, as I always am.. That's how I ended up in a new class where I only knew a couple of people. When the day came to choose a group, I asked some interesting looking people if I could join them, and to my surprise they were quite easy about it and said sure.

The first couple of meetings I felt like an alien and didn't want to express myself as much as I usually would, but the more meetings we had the better things started to look, which I am very happy for. For the project, I worked on many different things, and that helped me take a grasp of what we are doing in our current semester. Teammates helped me notoriously this semester. If there was something I did not understand they were always happy to help, or at least I think they were . With that being said there were a couple of things I was struggling with, even following the tutorials I felt stuck, so it took more time to do tasks for me as it would usually take for them. During the penultimate sprint, I was already supposed to have one of my features implemented and working, but I failed, and then I pushed myself hard to make it work as I understood that something like that is unacceptable, as everyone else has finished their tasks on time.

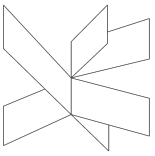
For the meetings I always tried to be on time, there were days when I came 30 minutes earlier, there were days when I was a couple of minutes late as well. There was also a day when I could not attend a meeting, because I had a



concert going on. I informed my group a couple of weeks prior so they would have a heads up on why I am absent during one of the meetings.

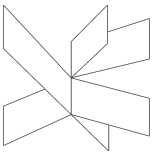
I am really happy that I ended up with this group, my last semester group mates were older of age, and sometimes they had bigger responsibilities than studies, so we would not meet as constantly, and there were no talks about going out to get something to eat or drink at evenings, as it is with this group, and that makes me very happy and more connected with them.

To end my brag on what I did and what I did not, I would like to once again appreciate my new teammates, and push myself more, because I feel like there were things that I could have helped them with more.



Supervision

During the 3rd semester we were assigned Joseph Chukwudi Okika as our supervisor and overall, we had a good experience. During the lecture period we had several hand-ins that were reviewed by Joseph, and we always received constructive feedback that was explained clearly. While working on the project full-time, we only met with the supervisor once, where we wanted approval for the general structure for the project since it was updated. Also, Joseph was contacted by us by email several times where he made himself available to answer questions and give suggestions for improving our project. Therefore, we felt that we had enough support.

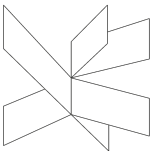


Conclusion

In conclusion, the process to fulfill the essential project requirements has been relatively smooth. Over time, our group has built a great working relationship where even including new people has only improved overall working experience. Although, coming from different countries and even different generations, we acknowledged that we have different personalities and implemented the knowledge of unique personal profiles to fit our group dynamic.

Moreover, implementing SCRUM and Unified Process frameworks into our daily work routines has heavily improved our productivity. Also, the frameworks had a positive influence on the experience of making this project, since using both SCRUM and UP have prevented us from cramming everything into the last minute where stress may have reduced the quality of the project and put us in a difficult mental state. Therefore, future recommendation would be to use SCRUM and UP to the best of our ability.

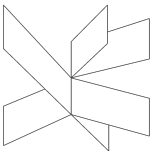
Lastly, the main negative takeaway has been to start using Jira when the SCRUM has already started. As we aim to be detailed in our documentation, Jira has interfered with it in the beginning. Therefore, the recommendation is if we choose to use tools like Jira, it has to be learned in advance.



Resources

https://via.itslearning.com

https://studienet.via.dk/projects/Engineering__project_methodology/SitePages/Home.aspx



Appendices