ChadList

NoBrainNoPain

Group Members:

Tomás Gres (315185),

Dragos-Daniel Bonaparte (315261),

Dan-Sebastian Ceapa (315162),

Chiril Luncasu (315171),

Matas Armonaitis (315263)

Supervisors:

Jakob Knop Rasmussen (JKNR)

Joseph Chukwudi Okika (JOOK)

VIA University College

54270 Characters

Software Engineering

3rd Semester

17 / 12 / 2022

Table of content

Abstract

A small startup business called ChadList entered the market with an ambitious vision towards the 12th United Nations Goals (Anon., 2022n), to be precise, Responsible consumption and production. The startup offered research in terms of the waste generated worldwide and got to an alarming conclusion that there is about 53.6Mt of e-waste in 2019 alone (Anon., 2022m), around 17Mt of textile waste only in 2018 (Anon., 2022e) and in 2017 the furniture waste generated by Americans was 12.2Mt (Anon., 2022l). Their approach to the problem was to offer people an easy and efficient way to sell or offer their unused items to other people that can make use of them. The startup's ideas and concerns were thereafter discussed with the Product Owner and the stakeholders to agree on a common and realistic vision of how the service will function. After which 22 functional requirements were selected carefully.
From the requirements given, the system will be composed of a simple set of actors, authenticated users, and unauthenticated users. Those requirements represent the main functionalities that can be performed by the aforementioned actors: creating a user which after will require the user to log in, once logged in, a user can create an item post and add images to it, and the marketplace can be accessed by either of the actors, an authenticated user is required to start a message with the seller, while the unauthorized user must rely on the phone number or email address, a user can edit his/her account, a user can edit an item and or change its status to sold or available.
The distributed system is composed of a 3-tier architecture which will offer faster development, improved scalability, improved reliability, and improved security making it clear, secure, and easy to maintain (Anon., 2022f). This layered architecture will allow the service to function everywhere regardless of location, making it more accessible to the end user. The system is presented as follows: The first tier, also known as the presentation tier is represented by a C# implementation of Blazor representing the user interface and it will be the tier that the end-user will interact directly with. The second tier will be the Business Logic Tier representing our middleware tier implemented in C# that will make sure that the third tier will be able to communicate back and forth with the first tier, but they must not communicate directly with each other creating a security flaw in the system. The third tier also known as the Data Access Tier is written in Java

and will serve as the persistence which is connected to a database created in PostgreSQL. The first and second tiers communicate through Representational State Transfer also known as REST, while the second and the third tier will communicate through Remote Procedure Call open sourced by Google also known as gRPC.

To assess the quality of the product, the team used both Black boxing and White boxing testing. The tests written conclude that the product proves to meet the requirements and the main ideas of the stakeholders and the Product Owner.

1. Introduction

Our customer is a non-profit organization that focuses on waste management. It has decided to enroll in the competition between recycling and wasting, hoping that its approach will motivate people to waste less and make use of items for longer. We were contacted by the company with the offer to develop a system that would help its users to sell and exchange their belongings that they no longer need.
The issue with most secondhand websites like eBay and Craigslist is that it is not available worldwide, and in consequence, the end users who would benefit from the service are excluded(Anon., 2022q). Another aspect that the startup wanted to address is the need for an authenticated user when buying an item. While Facebook Marketplace addresses the problem with worldwide coverage, it lacks anonymity for its users, requiring the end user to have an account before using the service which is something inconvenient for the user, especially if the user uses the service to only buy items.

The business owner decided to take on the opportunity and combine both worlds, where there is access to the service worldwide and anonymity for the end user that only uses the website for purchases. Having all those requirements laid down, the Product Owner, together with the stakeholders decided to include the following delimitations for the system:

- No map for the area where the product is located
- No shipping support whatsoever (Anon., 2022v)
- No monetary transactions through the website (Anon., 2022s)
- Users require an account only for selling items
- No complex recognition of illegal items (Anon., 2022t, Anon., 2022k)
- The website will limit its domain only to “.com” (The change of region will be done on the website)

-

To accommodate the requirements, the project decided to adhere to the Unified Process methodology. Having the Agile manifesto in mind, the project went sequentially through 4 phases:

I. Inception
II. Elaboration
III. Construction
IV. Transition

2. Analysis

The purpose of ChadList is to reduce wasting of reusable items throughout the world by giving everyone access to a platform where they can sell and buy those items quickly and reliably. The Product Owner put in a lot of effort in understanding the client as much as possible.
The system will have two types of users: unauthorized users and authorized users. Authorized users will be able to sell and buy items while unauthorized users will only be able to buy items.

2.1 Functional Requirements

2.1.1 Critical Priority

1. As a user I want to register, by specifying my first name, last name, phone number, email, date of birth and gender and log in specifying the email and password I used for the registration, in order to access my account.
2. As a user I want to be able to post an item for sale specifying its name, description of the item, price, currency and status of the item, in order to sell an item I don't need anymore.
3. As a user I want to message the user over an item they are selling in order to negotiate and ask details about the item.
4. As a user I want to be able to search for a specific item specifying the part of name, part of the description and min and max price in order to find an item that I am looking for.

5. As a user I want to be able to view details of a specific item in order to decide about the purchase.
6. As a user I want to be able to see all my active and previous chats with the sellers, to be able to verify information between what the seller said and what the item is about.
7. As a user I want to be able to report an item in case the item does not respect the rule about forbidden items.

2.1.2 High Priority

8. As a user I want to be able to edit my profile in order to modify any information that became outdated.
9. As a user I want to be able to delete or edit an item, in case I did any mistake during posting or I decided not to sell the item, or I sold the item.
10. As a user I want to be able to see all my notifications in order to keep track of messages and item updates.

2.1.3 Medium Priority

11. As a user I want to be able to have a list of items that might interest me in order for me to find better items.
12. As a user I want to be able to add an item to my wish list in order to save items that I am not sure yet that I want.
13. As a user I want to get a notification whenever there is an update on an item in my wish list in order to be up to date about my saved items.
14. As a user I want to be able to see the sellers rating in order to see if he is a trustable person.
15. As a user I want to be able to comment/rate a seller to help other users trust the seller or not.
16. As a user I want to be able to see a seller’s profile in order to see any details that interest me such as address, rating or the amount of sold products.
17. As a user I want to be able to delete items from my wish list, in order to remove items that do not interest me anymore.
18. As a user I want to be able to change the password so that I don’t get locked out of my account.
19. As a user I want to be able to delete my account in case I do not want to use the service anymore.

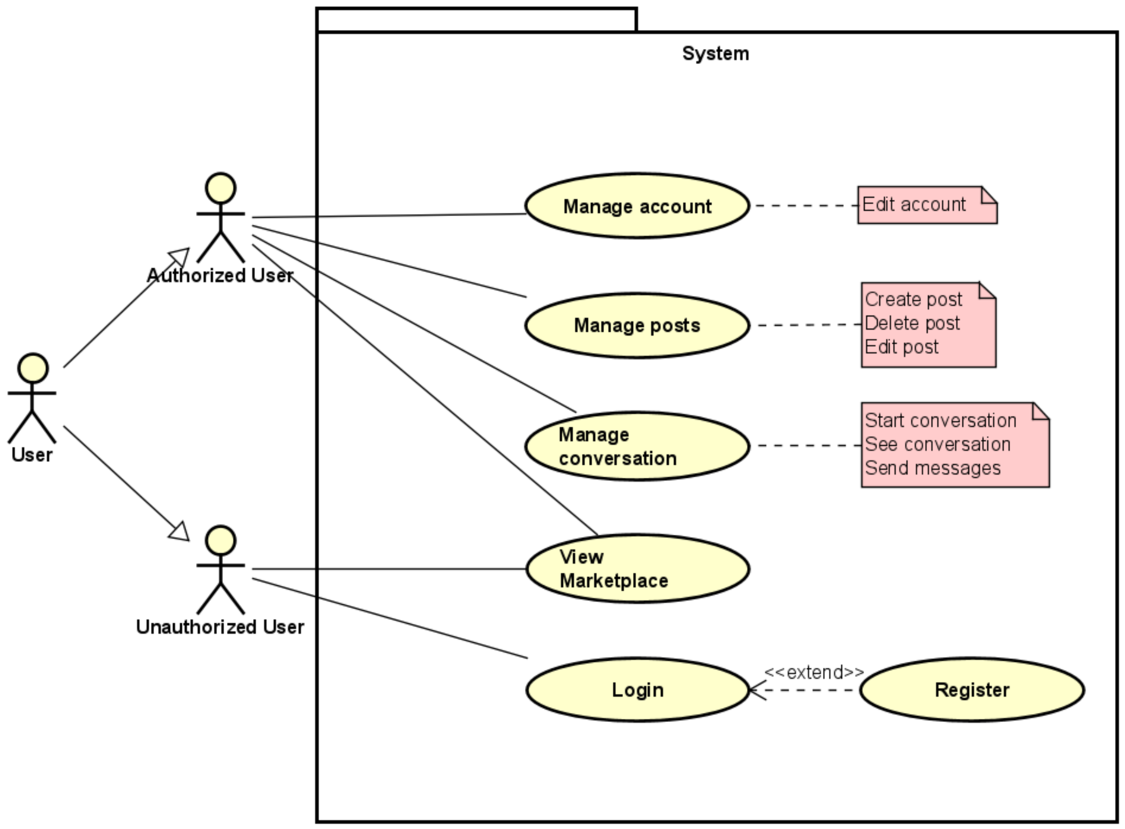
2.1.4 Low Priority

20. As a user I want to be able to sort the items by categories, in order to find what interests me.
21. As a user I want to be able to track how many people added a specific item to their wish list for me to take a faster decision about the item.
22. As a user I want to be able to see related items when I look at a specific post, in order to have a better chance of getting a better deal.

2.1.5 Non-Functional Requirements

1. The system must be able to work on Mozilla Firefox v106.0 and previous, Google Chrome v107.0 and previous and Microsoft Edge 107.0 and previous

2.2 Use Case Diagram



Use case explanations:

- Manage account
    - The user can choose to edit his/her profile at any time by pressing the person icon on the right top corner
- Manage posts
    - An authenticated user can create a post
    - An authenticated user can edit a post
    - An authenticated user can delete a post
- Manage conversation
    - An authenticated user can create a conversation with the seller
    - An authenticated user can see a conversation with a specific seller
    - An authenticated user can send messages in a conversation with the seller
- View Marketplace
    - Any user can view the items and press on them for a detailed view
    - An authenticated user can start a conversation with the seller from the detailed view
- Login
    - An unauthorized user can log into the system to gain access to more features
- Register
    - An unauthorized user can register into the system

2.3 Use Case – Fully Dressed Description (Manage Posts)

The following use case will represent the main page of the website whereas the user will spend most of the time. The use case shows the abilities of a user with an account.

Use case	Manage posts
Summary	The user can create/edit/delete a post.
Actor	User
Precondition	The user must be logged into his account.
Postcondition	The details and information for the post will be saved into a database.
Basic Scenario	Create Posts 1. The actor logins to his account. 2. The actor presses the marketplace button. 3. A window will be shown with all items that are for sale and their description. 4. If the actor wants to sell an item, he needs to press the create item post button. 5. A window will be shown, where the actor can add all the necessary information about the item. Edit Posts 1. The actor presses my item button. 2. The actor chooses the item, and presses the edit item button

	3. A window will be shown with the item where the actor can change the details about the post. 4. To save the changes, the actor should press the submit button below, and the changes will be saved. Delete Posts 1. Users can delete the post by pressing the delete item button. 2. The item will be deleted from the marketplace.
Alternative Scenario	
Exception	If one of the fields or more are empty, an error will be shown.

Table 1 – Manage posts – Fully dressed description

2.4 Activity Diagram (Manage Posts)

The following diagram represents the way the system manages posts. The diagram covers both main scenarios and alternate scenarios.

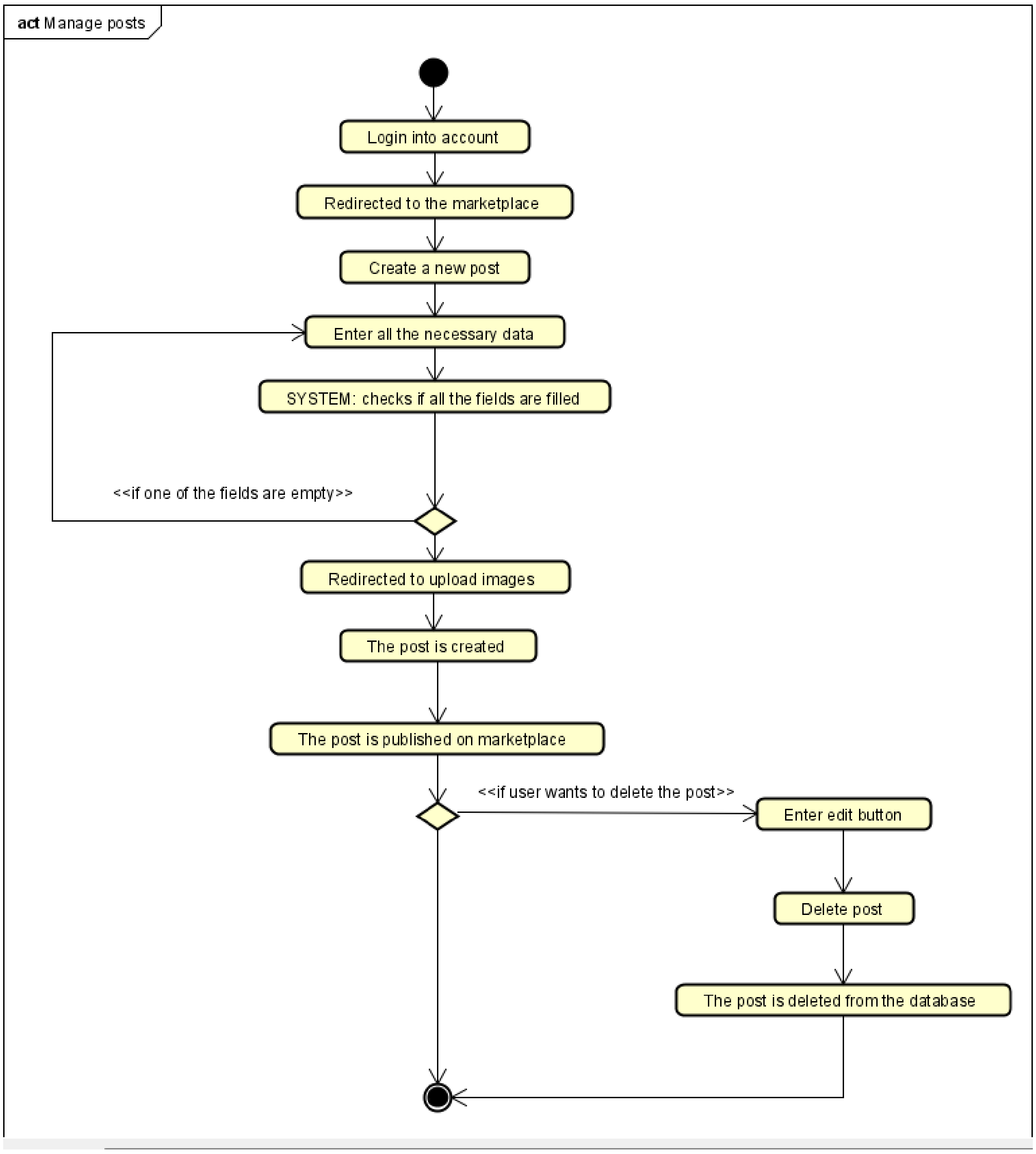

act Manage posts
Login into account
Redirected to the marketplace
Create a new post
Enter all the necessary data
SYSTEM: checks if all the fields are filled
<<if one of the fields are empty>>
Redirected to upload images
The post is created
The post is published on marketplace
<<if user wants to delete the post>>
Enter edit button
Delete post
The post is deleted from the database

Figure 2 – Activity diagram – Manage posts

2.5 System Sequence Diagram

A system sequence diagram is used to represent how the system will respond to the user's input and every step that the system will perform towards completing its task. The following sequence diagram represents the steps that the user and the system must take in order to create a post.

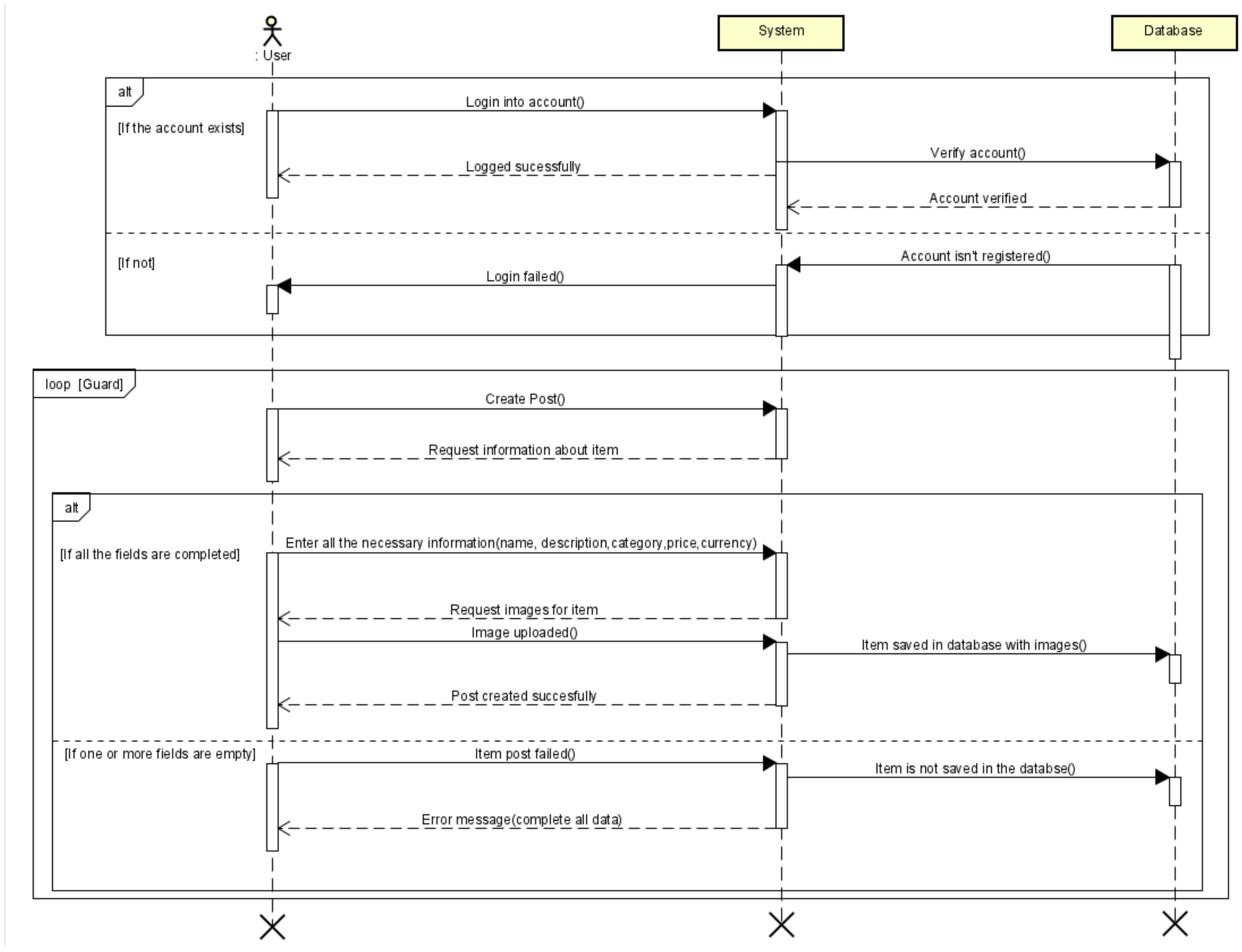

: User
System
Database
alt
[If the account exists]
Login into account()
Verify account()
Logged sucessfully
Account verified
[If not]
Account isn't registered()
Login failed()
loop [Guard]
Create Post()
Request information about item
alt
[If all the fields are completed]
Enter all the necessary information(name, description,category,price,currency)
Request images for item
Image uploaded()
Item saved in database with images()
Post created succesfully
[If one or more fields are empty]
Item post failed()
Item is not saved in the databse()
Error message(complete all data)

Figure 3 – System Sequence Diagram – Manage posts

2.6 Domain Model

The diagram shown below represents the relationships between the actor and the components. A **User**, after registering and creating an account, can decide to sell their **Item** (things they own and do not need anymore). If they wish to buy that item, they can start a **Conversation** with them to negotiate the price and delivery by sending a **Message**. After a message is sent, the seller will get a **Notification**.
Another **User**, decides to enter the marketplace to search for an item, the user not being authorized, can only see the item and will rely only on the **phone number** to purchase the item.

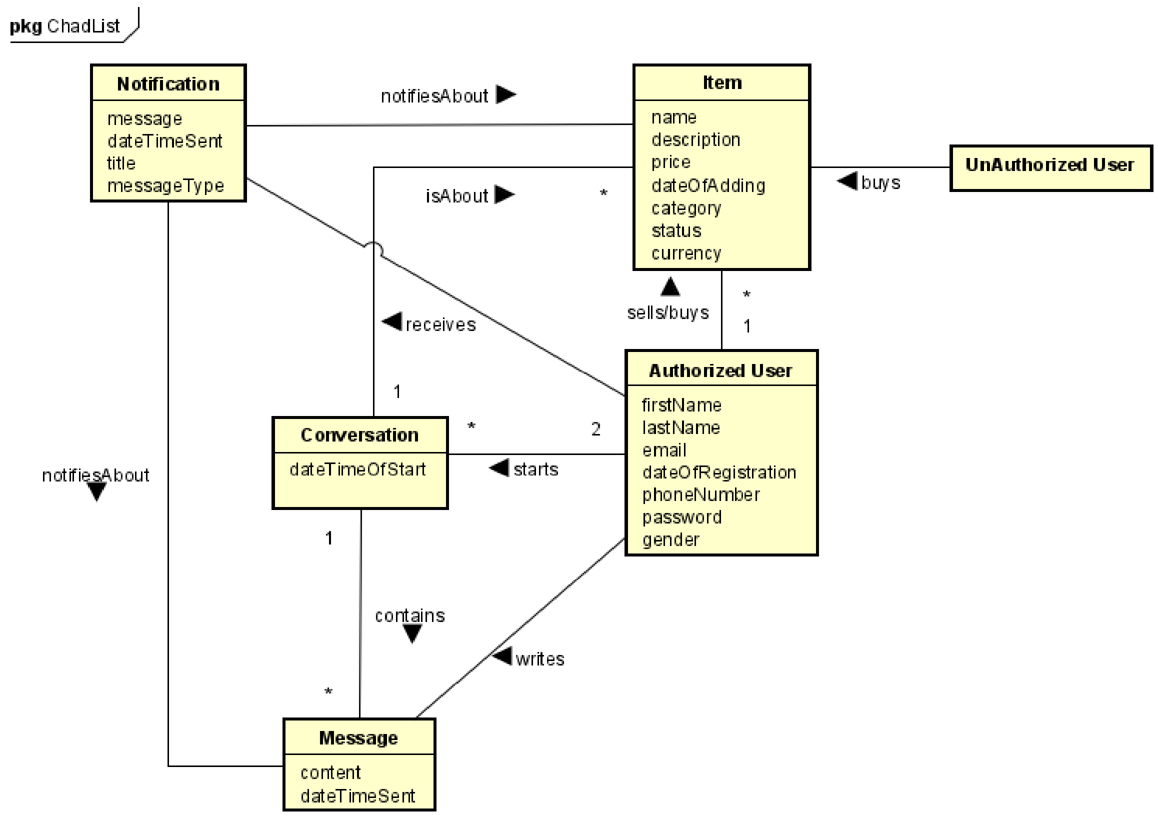


Figure 4 – Domain Model

2.7 Security

2.7.1 **Threat model** Click or tap here to enter text.(Anon., 2022v)

Threat case	TPM	Objective violation	STRIDE	X.800	EINOO	Consequence
XSS (Cross-site scripting)(Wang and Zhang, n.d.)	Data not being validated	Confidentiality, Integrity, Authorization	Spoofing identity, Tampering with data, Information disclosure	Passive	External Network Online	Privacy is disclosed, User's data is stolen
MitM (Man in the middle)	Privacy and confidentiality breach	Integrity, Confidentiality, Authorization	Spoofing identity, tampering with data, information disclosure	Active and Passive	Insiders Network Online Offline	Privacy disclosed, The attacker can intercept information between them and the service.
Brute-force attack	User authentication, data authorization and validation is not handled	Authentication, Confidentiality	Repudiation, Spoofing identity, information disclosure	Active	External Network Online	Personal information disclosure, data modification, customers' identity stolen
DoS (Denial-of-service attack)	Host not having enough protection for such recurring attacks	Availability	Denial of service	Active	External Insiders Network Online	Server's unavailability to the users
The Heart Bleed Bug	The length of the message is not checked	Confidentiality, privacy	Tampering Information disclosure	Active: Modification	External Network Online	The system might return confidential data, such as the server's private decryption or signing key

Table 2 – Threat Model

2.7.2 Objectives

The stakeholders requested that the website be secure. The Product owner communicated back to the team that confidentiality, and the integrity of the website should be of high standards.
First of all a website that does not provide confidentiality is not a trustable one, and in turn makes the website less probable to have any users.
Data confidentiality security mechanisms need to be implemented in such way such as unauthorized individuals will not be able to access it.
This will ensure that the system remains trustable, and the user will have its privacy when using the system.
As for integrity, the authorization was created in an ingenious way such as data integrity is assured to be changed only by authorized individual. Any sensitive information will not be edited without the user being authorized. The system integrity is maintained when even if a user accesses the URL of a page illegally the authorization will not allow access to anything on the page.
Lastly, the availability will be guaranteed by a strong and secure host. The system will have to be deployed on a host with good reviews and great security management for providing availability and prevent DoS attacks from occurring.

2.7.3 Threat prevention

The Product Owner presented the threats that the system will be prone to, to all the stakeholders following small changes in the requirements.
The prevention of the threats began with:

- **XSS prevention** (Anon., 2022u)**:**
    - Escape user input. Escaping means to convert the key characters in the data that a web page receives to prevent the data from being interpreted in any malicious way. It doesn't allow the special characters to be rendered.
    - Validate user input. Treat anything that originates data from outside the system as untrusted. Validate all the input data. Use an allowlist of known, acceptable, good input.

- Sanitize data. Examine and remove unwanted data, such as HTML tags that are deemed to be unsafe. Keep the safe data and remove any unsafe characters from the data.

- **MITM prevention** (Anon., 2022r)**:**
    - Strong WEP/WAP Encryption on Access Points
    - Strong Router Login Credentials
    - Virtual Private Network
    - Force HTTPS
    - Public Key Pair Based Authentication
- **Brute-force prevention** (Anon., 2022ac)**:**
    - Longer passwords
    - Complex passwords
    - Limit login attempts
    - Using captcha
    - Two factor authentication
    - Cloudflare
- **DoS prevention** (Anon., 2022j)**:**
    - Good Cloud Mitigation Provider.
    - Strong Firewall
    - Internet Service Provider (ISP)
- **The Heart Bleed Bug prevention:**
    - Using a new and updated version of OpenSSL

2.7.4 Risks

Threat mitigation is important, but the system must also take in consideration what is the risk of a threat and how much will it impact the service. This analysis will show which of those threats will have priority to be solved. By having a priority the system can be developed with the most important threat prevention, allowing the others to be addressed later in the lifetime of the system.

Threat case	**Occurrence risk**	**Impact on the system**	**Priority**
XSS (Cross-site scripting)(Wang and Zhang, n.d.)	Medium	Medium	Medium
MitM (Man in the middle)	High	High	High
Brute-force attack	Low	Low	Low
DoS (Denial-of-service attack)	Medium	High	Medium
The Heart Bleed Bug	Low	Medium	Low

Table 3 – Threat risk assessment

3. Design

3.1 System Architecture

In Figure 5, the system architecture and the communication protocols/middleware is shown visually. The presentation tier is implemented using Blazor Web Assembly (C#). From the presentation tier, the C# objects are converted to JSON objects and sent via HTTP protocol to the business tier. The business tier is implemented using .NET 6.0 (C#). The HTTP Controllers in the business tier handling the communication with the presentation tier follow Representational State Transfer (REST). The business logic then communicates with the persistence/database tier using Remote Procedure Calls (gRPC) and data sent through is serialized using Protocol Buffers. The database tier is implemented in Java and Spring Boot Framework. The database itself is implemented using PostgreSQL and Object Relational Mapping tool - Hibernate which is the implementation of the JPA specification.

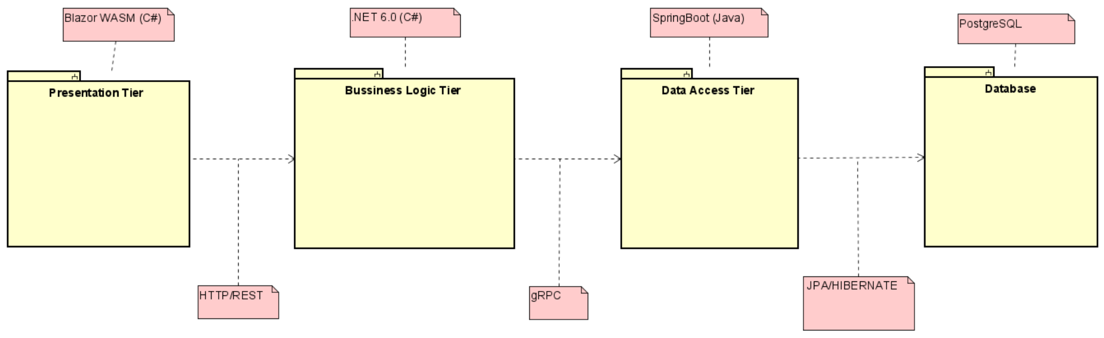


Figure 5 – Tier architecture

3.2 Technologies

3.2.1 Blazor Web Assembly

Blazor Web Assembly (Blazor WASM) was used for the design of the interactive user interface. The visual part was implemented using HTML, CSS, Bootstrap and the functional part was written in C#. The graphical design was adjusted using Bootstrap library and CSS.

Authentication and authorization were implemented using this technology as well. If a user was successfully logged in, the system would grant him privileges to create new item posts, contact other users using our messaging system as well as editing their profile and the items they posted.

3.2.2 HTTP/REST

Hypertext Transfer Protocol and Representational State Transfer were used for communication between presentation and business tier. The HTTP protocol accesses resources identified by an endpoint and returns the relevant information about the requested resources.

On the presentation tier, the HTTP Requests: POST, GET, PUT, PATCH and DELETE were used. Each of the requests is sent to the business tier and there the http endpoint controllers (which were implemented following REST principles) handle the requests.

The POST requests are used to create new entities (for example new item post). GET requests are used to retrieve one or multiple entities from the endpoint. PUT and PATCH requests are used to update entities either by replacing the whole entity or just updating a part of the attributes of the entity respectively. The DELETE request is used to remove entities from the endpoint.

3.2.3 gRPC

Google Remote Procedure Call with Protocol Buffers are used between the business tier and the database tier. The gRPC uses Protocol Buffers by default as Interface Definition Language (IDL) for describing both the service interface and the structure of the payload messages, which allows it to communicate between tiers implemented using different technologies.

GRPC provides four types of calls: Unary (where client sends single request and gets a single response), Server streaming RPCs (where client sends single request and gets a sequence of messages back), Client streaming RPCs (where client sends a stream of messages to the server, and after all messages are sent, the server sends a single response back to the client) and the Bidirectional streaming (where both sides send messages using a read-write stream, they can read and send messages independently and the message order in both streams is preserved)

In the system, the unary call was used for example for creating an item post, where a Create DTO was sent through, and a new Item message was sent back to the client. The Server streaming RPCs were for example used for retrieving all items the user posted. The last two methods were not used in our system. (Anon., 2022h)

3.2.4 Spring Boot Framework

Spring Boot was used in the database tier. Spring Boot is a Java Framework that is used to create micro services. It allows to create standalone Java applications that do not rely on an external server because it embeds a web server such as Tomcat or Netty. (Anon., 2022x)

3.2.5 JPA/Hibernate

Java Persistence API is Java specification focused on ORM (Object Relational Mapping). Hibernate is a Java Framework that implements JPA specifications. JPA represents how to define POJO (Plain Old Java Object) as an entity and manage it with relations using some meta configurations. They are defined either by annotations or by XML files. (Anon., 2022p)

3.2.6 PostgreSQL

PostgreSQL is a relational database system which supports both SQL (relational) and JSON (non-representational) object representations.
PostgreSQL database is created using JPA and Hibernate and used to store entities like users, items, messages etc. In the system. (Anon., 2022ad)

3.3 Database

3.3.1 Entity Relations Diagram

As a starting point of the database design, an entity relations model was created. The ER diagram follows the Domain model to a great extent, with some changes. For the user, a decision was made to use a special id attribute instead of using the email or phone number for simplicity.

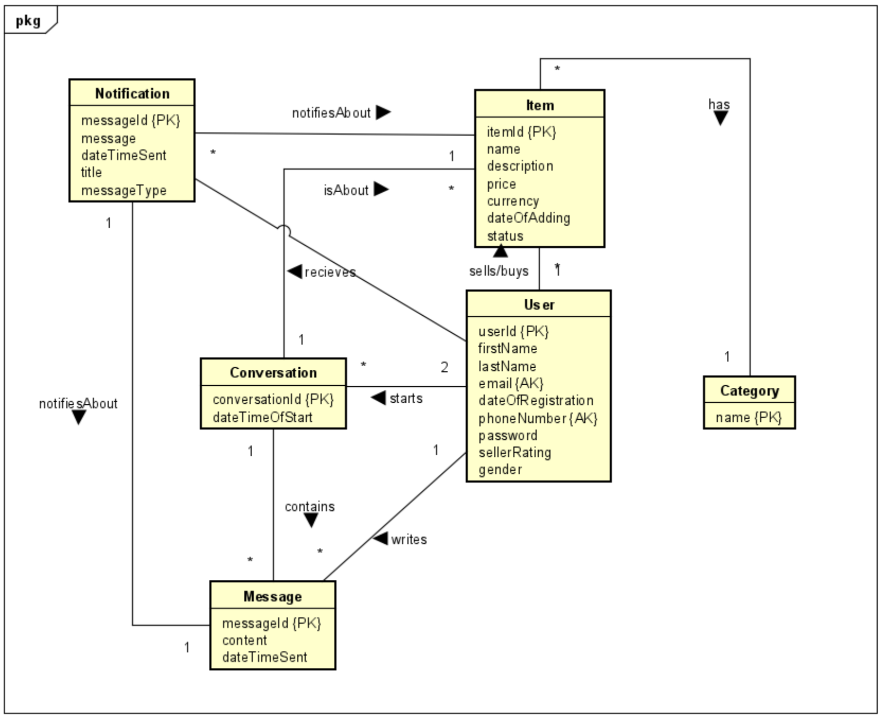


Figure 6 – Entity Relations Diagram

3.4 Design Patterns

3.4.1 Adapter Pattern

Adapter Patterns are used in the system for the JPA Repositories to be used by GRPC Services, in this example, the save method is mapped to createItem and updateItem methods, the findAll method is mapped to getItems method, findById is mapped to getItemById and deleteById is mapped to deleteItemById.

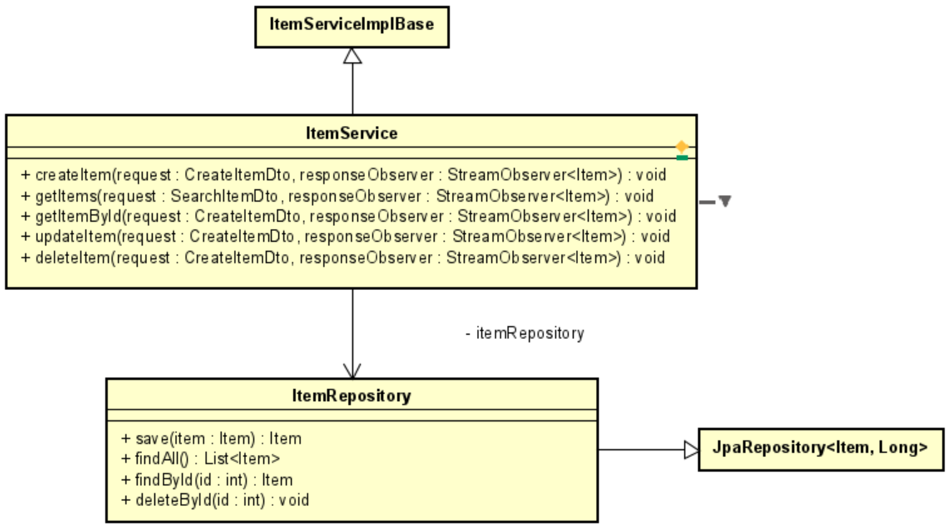

ItemServiceImplBase
ItemService
+ createItem(request : CreateItemDto, responseObserver : StreamObserver<Item>) : void
+ getItems(request : SearchItemDto, responseObserver : StreamObserver<Item>) : void
+ getItemById(request : CreateItemDto, responseObserver : StreamObserver<Item>) : void
+ updateItem(request : CreateItemDto, responseObserver : StreamObserver<Item>) : void
+ deleteItem(request : CreateItemDto, responseObserver : StreamObserver<Item>) : void
- itemRepository
ItemRepository
+ save(item : Item) : Item
+ findAll() : List<Item>
+ findById(id : int) : Item
+ deleteById(id : int) : void
JpaRepository<Item, Long>

Figure 7 – Adapter Pattern

3.4.2 Object Relational Mapping

Object Relational Mapping is a design pattern that simplifies converting OOP objects to database entities and vice versa. In this system JPA and Hibernate are used and follow this design pattern.

3.4.3 Database Access Objects

DAO pattern is used to encapsulate the database tier access in a series of interfaces and thus allowing for the database tier to be replaced without having to extensively change methods and classes in the business tier.

Database Access Object contain all the basic CRUD methods (create, read/get, update and delete). Each database entity (user, item, notification, conversation etc.) has its own DAO.

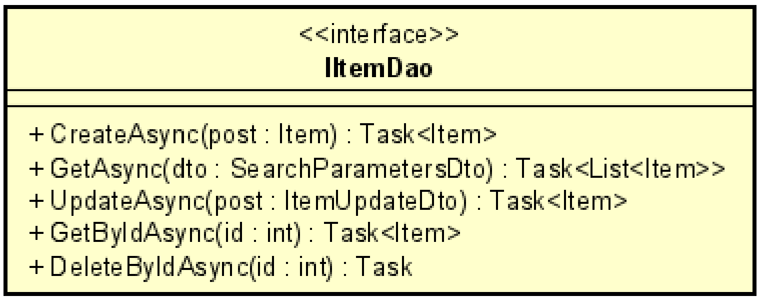

<<interface>>
IItemDao
+ CreateAsync(post : Item) : Task<Item>
+ GetAsync(dto : SearchParametersDto) : Task<List<Item>>
+ UpdateAsync(post : ItemUpdateDto) : Task<Item>
+ GetByIdAsync(id : int) : Task<Item>
+ DeleteByIdAsync(id : int) : Task

Figure 8 – Data Access Object

3.4.4 Data Transfer Objects

DTOs or Data Transfer Objects are objects that carry data between processes in order to reduce the number of methods calls.

In the system, the DTOs are used to specify necessary data for creating, updating or searching an entity, but leaving out information that is unnecessary.

For example, as the item's Id attribute is generated by the database, it is not necessary to specify it when creating the entity. Another unnecessary field is dateTimeCreated, as it will take the current time when it is saved to the database. The isSold attribute is also unnecessary, as in the beginning the item is not sold, so it is always false in the beginning. (Anon., 2022y)

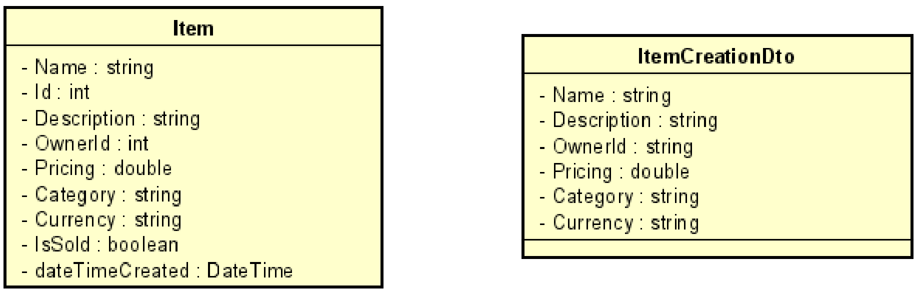


Figure 9 – Data Transfer Object

3.5 Solid Principles

The system was designed and implemented with great focus on the SOLID design principles, however, not all of the principles were useful for this specific project, so not all of them were used. (Anon., 2022w)

3.5.1 Single Responsibility Principle

In the business logic, there are controller classes, which are responsible for retrieving and posting data from and to the presentation tier. The logic classes are then responsible for validating the data sent from the presentation layer. Lastly there are Dao classes which are responsible for handling transfer of data to and from database tier. Each of those classes have only one responsibility.

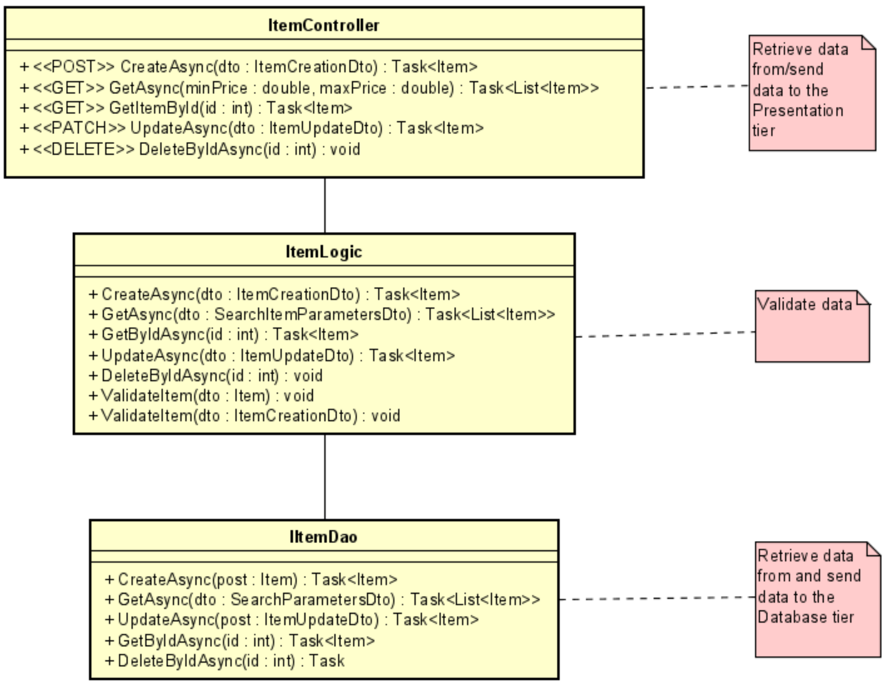

ItemController
+ <<POST>> CreateAsync(dto : ItemCreationDto) : Task<Item>
+ <<GET>> GetAsync(minPrice : double, maxPrice : double) : Task<List<Item>>
+ <<GET>> GetItemById(id : int) : Task<Item>
+ <<PATCH>> UpdateAsync(dto : ItemUpdateDto) : Task<Item>
+ <<DELETE>> DeleteByIdAsync(id : int) : void
Retrieve data from/send data to the Presentation tier
ItemLogic
+ CreateAsync(dto : ItemCreationDto) : Task<Item>
+ GetAsync(dto : SearchItemParametersDto) : Task<List<Item>>
+ GetByIdAsync(id : int) : Task<Item>
+ UpdateAsync(dto : ItemUpdateDto) : Task<Item>
+ DeleteByIdAsync(id : int) : void
+ ValidateItem(dto : Item) : void
+ ValidateItem(dto : ItemCreationDto) : void
Validate data
IItemDao
+ CreateAsync(post : Item) : Task<Item>
+ GetAsync(dto : SearchParametersDto) : Task<List<Item>>
+ UpdateAsync(post : ItemUpdateDto) : Task<Item>
+ GetByIdAsync(id : int) : Task<Item>
+ DeleteByIdAsync(id : int) : Task
Retrieve data from and send data to the Database tier

Figure 10 – Single responsibility principle

3.5.2 Dependency Inversion Principle

The Dependency inversion principle states that Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.

In the system, this is used for example in the logic and DAO classes. The controllers rely on logic interfaces, instead of the implementation of the logic itself. Same for the logic classes, which rely on DAO interfaces instead of the DAO classes themselves.

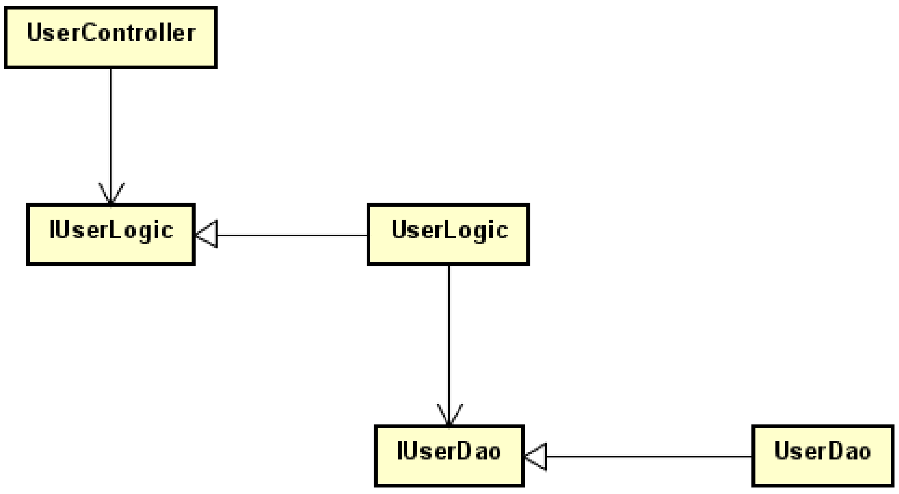


Figure 11 – Dependency Inversion Principle

3.6 Class Diagram

For a full resolution versions check in the appendixes

3.6.1 Presentation Tier:

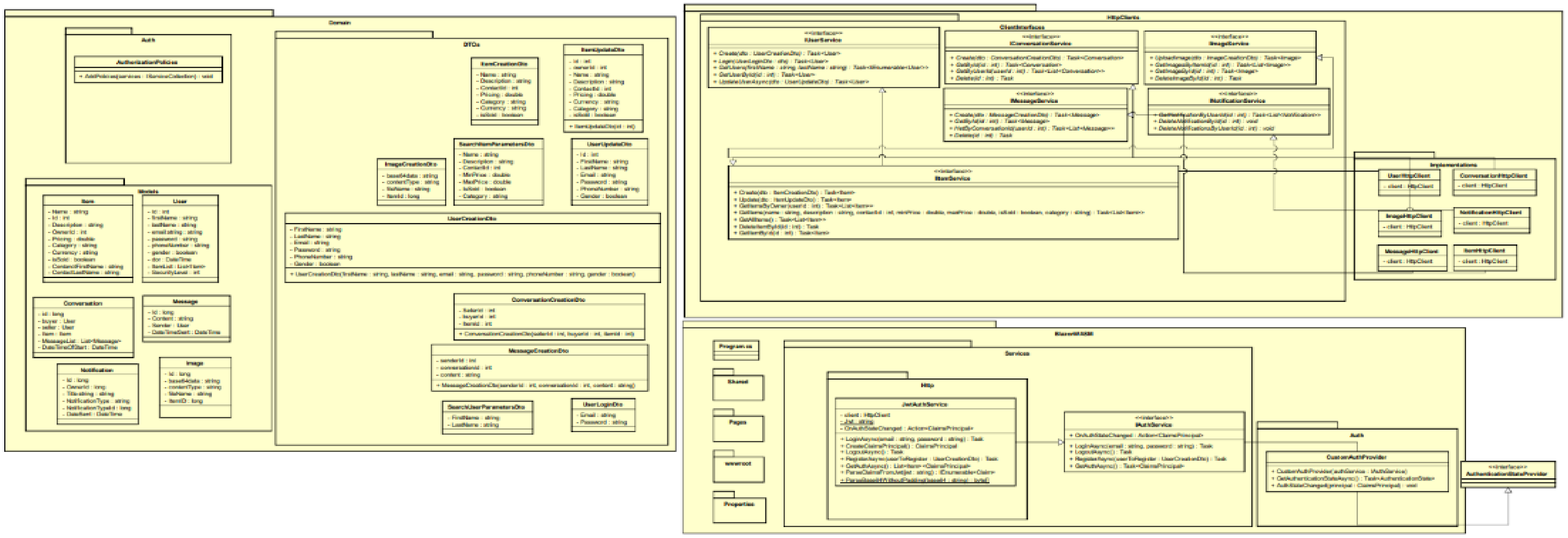


Figure 12 – Presentation Tier diagram

3.6.2 Business Logic Tier:

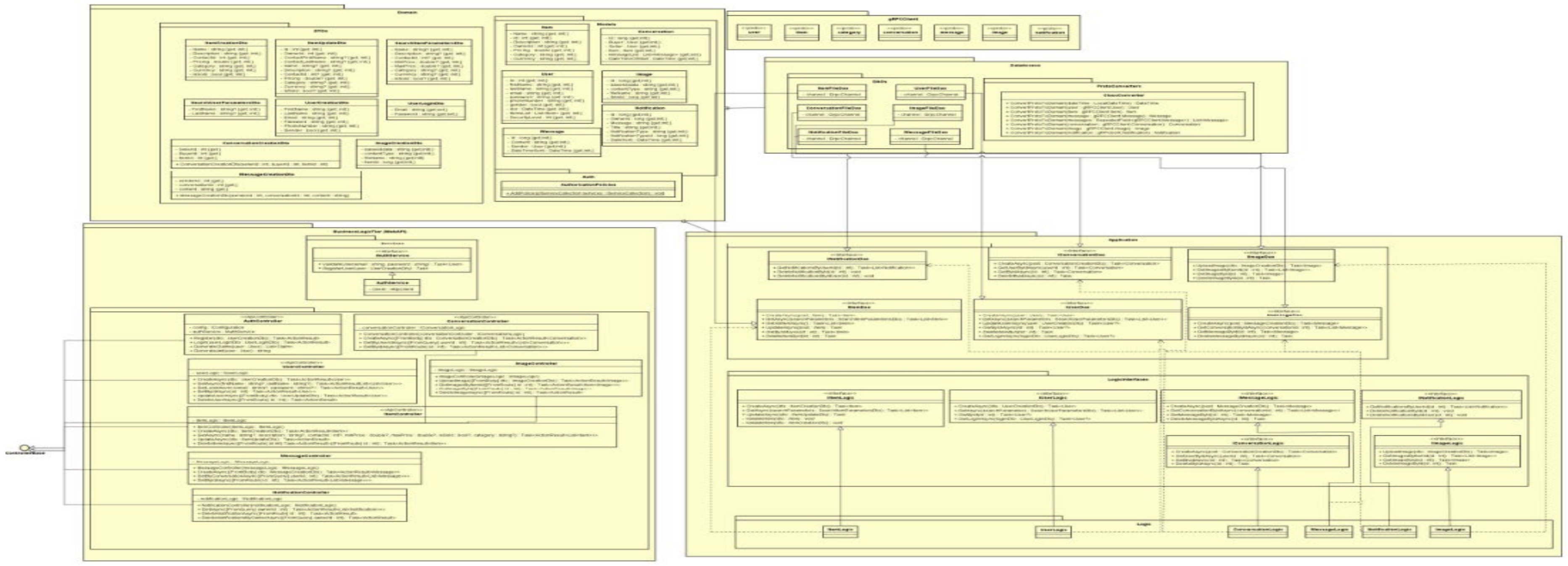
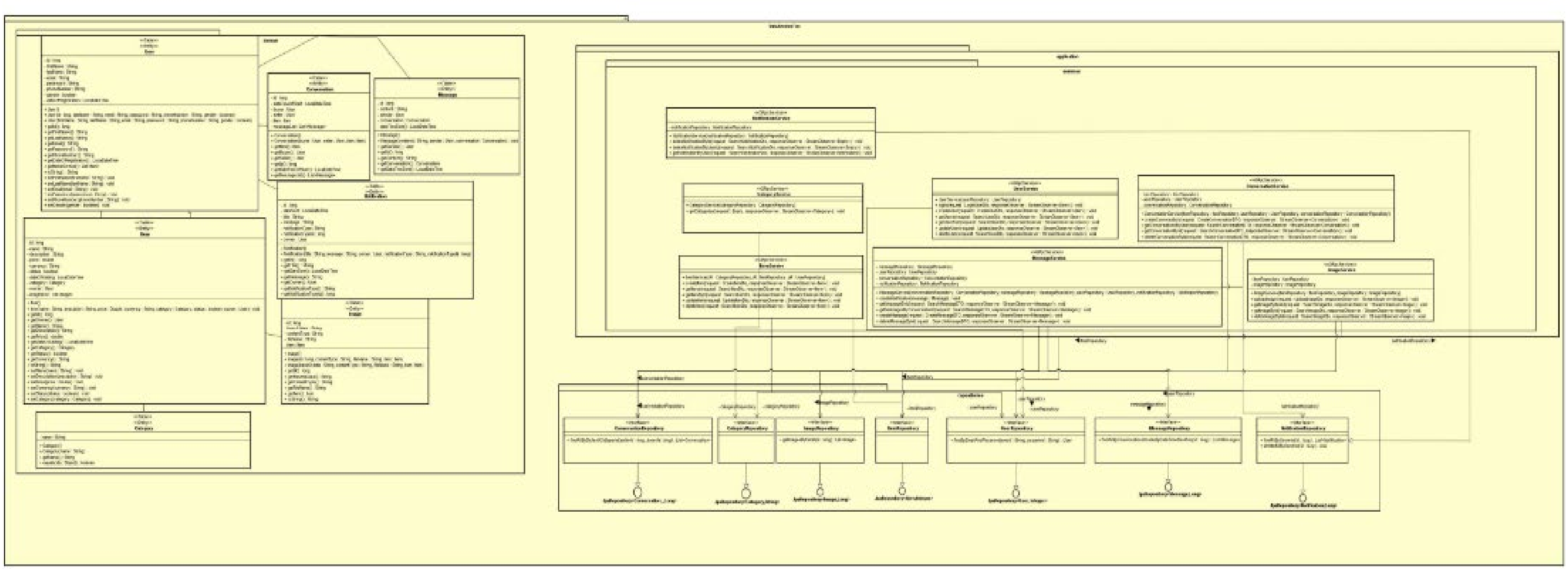


Figure 13 – Business Logic Tier diagram

3.6.3 Data Access Tier:



For the full Class Diagram in better quality, refer to Appendix L – Class Diagram

Figure 14 – Data Access Tier diagram

3.7 Sequence Diagram

The following sequence diagram shows the parts of the system that interact with each other in order to create a new item. When a user creates a new item by entering the information about it, that information is then sent through the view to the ItemHttpClient, where it is sent throguh HTTP protocol to the business tier. There the information is validated and afterwards sent to the database tier via gRPC. On the database tier it is sent to the ItemRepository and saved to the database.

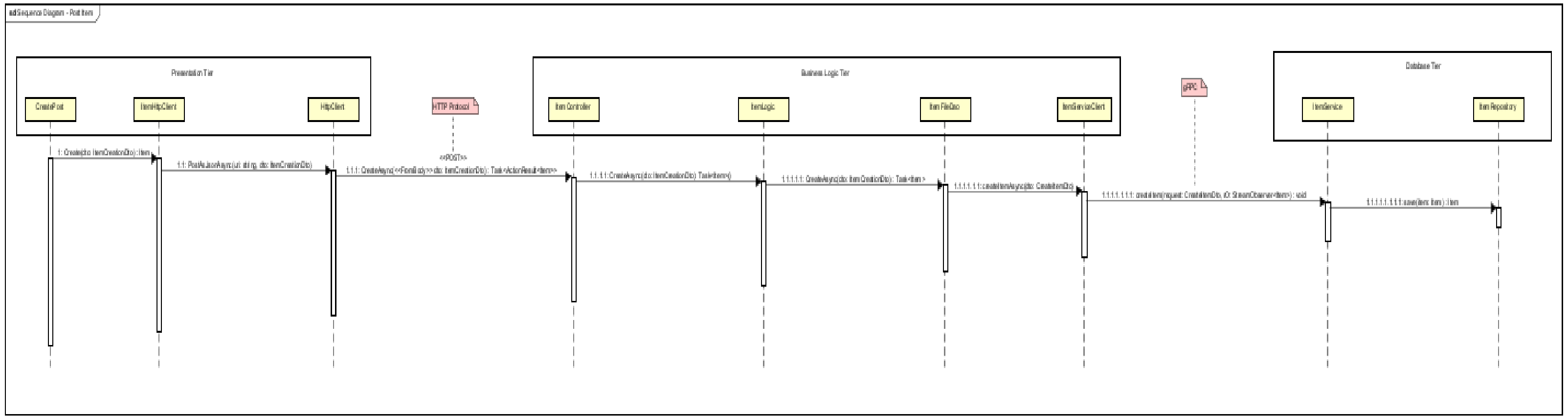

sd Sequence Diagram - Post Item
Presentation Tier
CreatePost
ItemHttpClient
HttpClient
HTTP Protocol
<<POST>>
Business Logic Tier
ItemController
ItemLogic
ItemFileDao
ItemServiceClient
gRPC
Database Tier
ItemService
ItemRepository
1: Create(dto: ItemCreationDto): Item
1.1: PostAsJsonAsync(uri: string, dto: ItemCreationDto)
1.1.1: CreateAsync(<<FromBody>> dto: ItemCreationDto): Task<ActionResult<Item>>
1.1.1.1: CreateAsync(dto: ItemCreationDto): Task<Item>
1.1.1.1.1: CreateAsync(dto: ItemCreationDto): Task<Item>
1.1.1.1.1.1: createItemAsync(dto: CreateItemDto)
1.1.1.1.1.1.1: createItem(request: CreateItemDto, iO: StreamObserver<Item>): void
1.1.1.1.1.1.1.1: save(item: Item): Item

Figure 15 – Sequence Diagram

For a better-quality image see the Appendix J – Sequence Diagram

3.8 Security Mechanisms

3.8.1 Authentication

Authentications in systems is very important to differentiate between specific users as well as making sure that each user can only access their own account. There are three main types of authentication factors: something you know, something you have and something you are. Something you know is for example a password (it is something no one else except for you knows), Something you have is for example your smartphone and something you can be for example biometrics (face, fingerprints, voice). During Authentication process a user provides one of these factors but more usually a combination of those factors. For example, to access your bank account you need your smartphone (something you have), your face/fingerprint (to unlock your phone) and a service code or a PIN (to access the bank app itself).

3.8.2 Password Hashing

Password hashing is a way for the user to log into their account without their password ever being stored in the database. Instead of that, the password is changed using multiple mathematical functions which result in a string of a specific length of 64 characters (SHA256). The hashing algorithm is made in a way that whenever you enter a certain password, it will always result in the same hash. Therefore, when the user enters the password to log in, the password is hashed and then compared to the hash in the database. Another important ability of the hashing algorithm is that there is no way to get the password back from the hash. However, there are websites like crackstation.net which store hashes to many different passwords and are therefore able to crack the hash anyways. It is still important for the user to choose a strong password. (Anon., 2022i)

3.9 Graphical User Interface Design

The Graphical User Interface is in a form of web application made with Blazor Web Assembly which contains features for both authenticated and unauthenticated users. The view itself is made using HTML and designed using Bootstrap and CSS. The aim for the design was to be simple and easy to learn for new users.

3.9.1 Main View

The main page contains a navigation bar which enables the user to enter the marketplace and about us page and messages and a categories list as well as a search bar to look for items. After the user logs in, they can see their messages and their items listed for sale as well as their wish list.

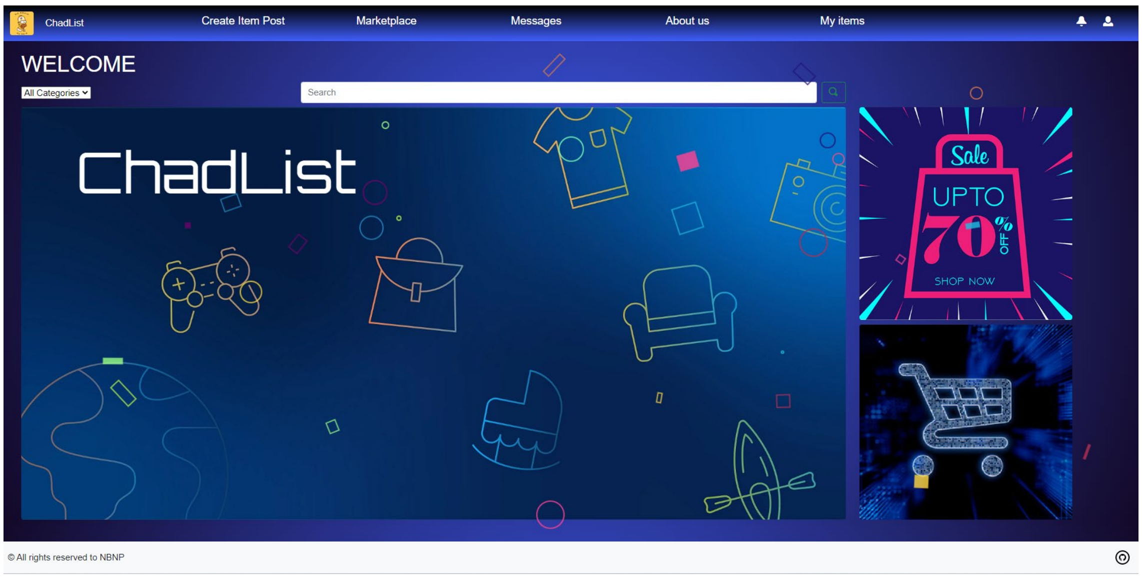


Figure 16 – Main page view (Anon., 2022a)

3.9.2 Registration View

Registration view is used for the users to register a new account. They enter their credentials and information about themselves for other users to recognize them.

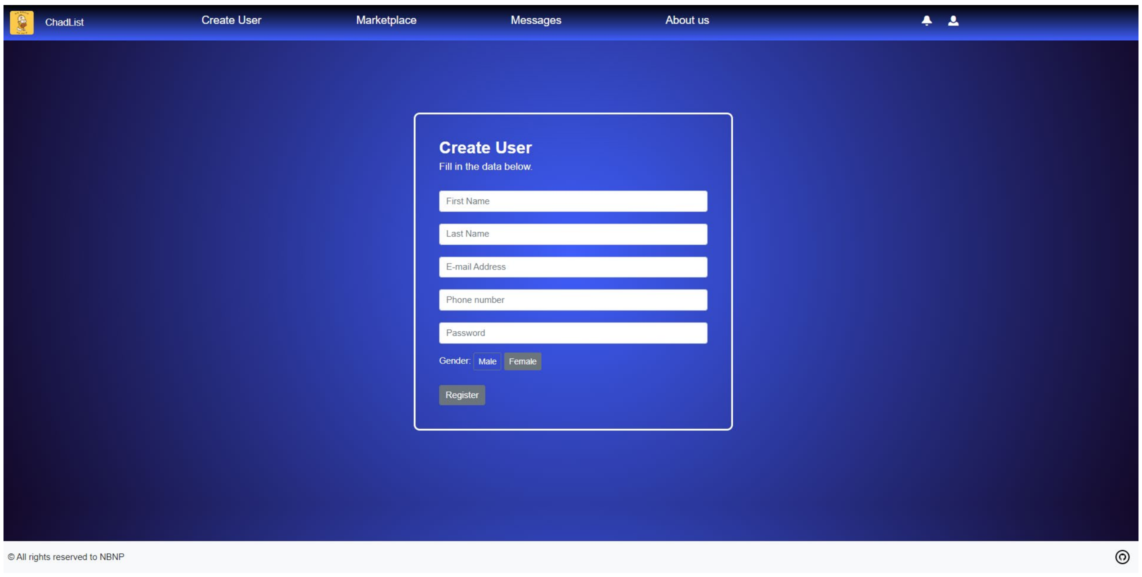


Figure 17 -Registration view

3.9.3 Marketplace View

In the Marketplace view the user can see all the items listed for sale, as well as to filter between them and selecting a specific category or a price range. Every item shows a main picture, a price, name and description.

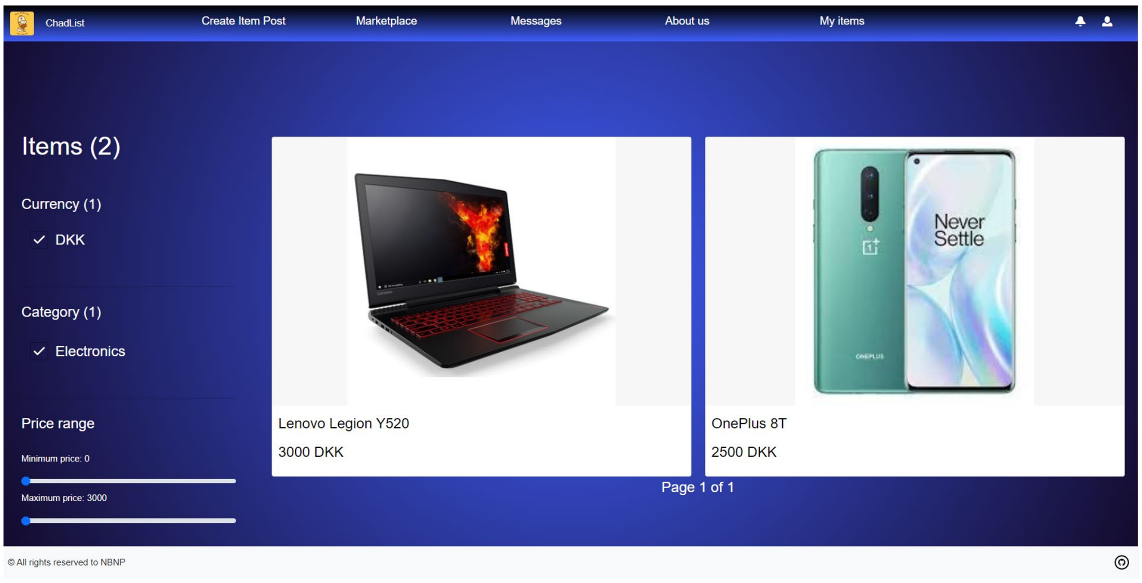
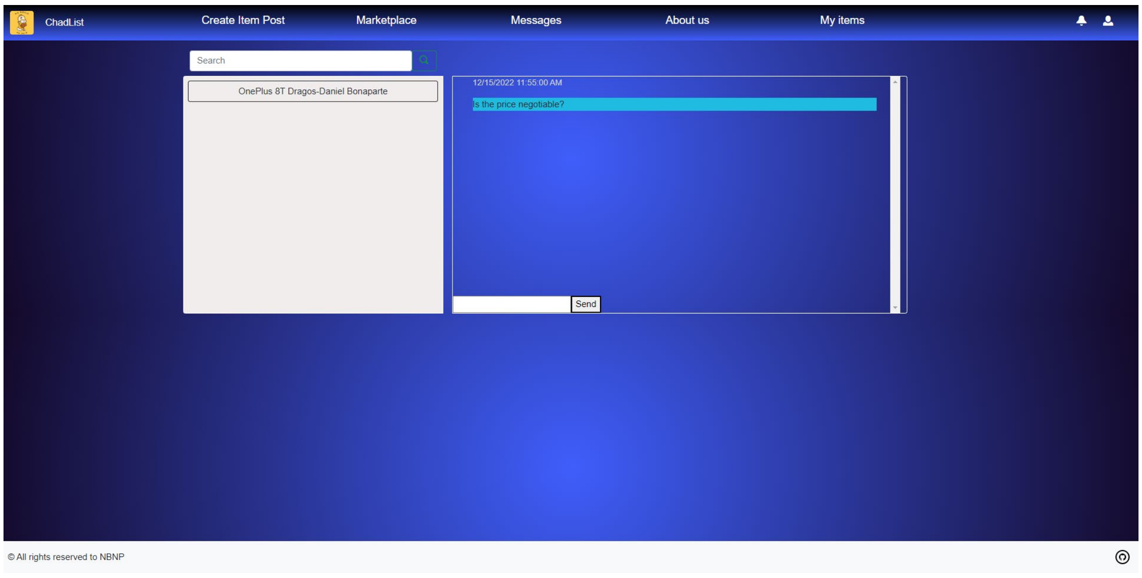


Figure 18 – Marketplace (Anon., 2022b)

3.9.4 Message view

In the messages view, the user can see all their conversations with other users. They are also able to read and send messages to other users.

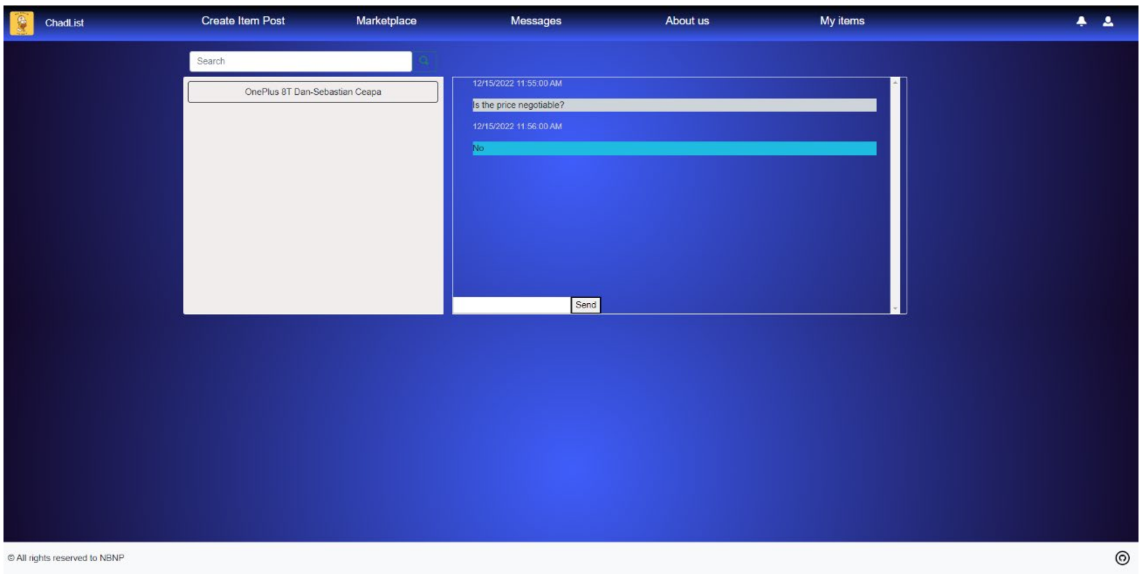


OnePlus 8T Dragos-Daniel Bonaparte

12/15/2022 11:55:00 AM

Is the price negotiable?

Figure 19 – Buyer messages view



OnePlus 8T Dan-Sebastian Ceapa

12/15/2022 11:55:00 AM

Is the price negotiable?

12/15/2022 11:56:00 AM

No

Figure 20 – Seller messages view

4. Implementation

4.1 Data Access Tier

4.1.1 Implementation in Java

After elaborating certain factors, the service's 3rd tier was chosen to be implemented in Java. It is responsible for handling and accessing the database and handle the gRPC requests.

The application is a Springboot implementation with dependencies to spring framework, Google's protobuf and Junit.

Spring framework provided the application with the necessary annotations, features and most importantly JPA/Hibernate, which is going to be the main way of communicating with the Database.

Google's protobuf is used to add in the project the .proto file type and provide the system with instructions on how to generate the protobuf.

Junit provide the necessary annotations for conducting tests on the code.

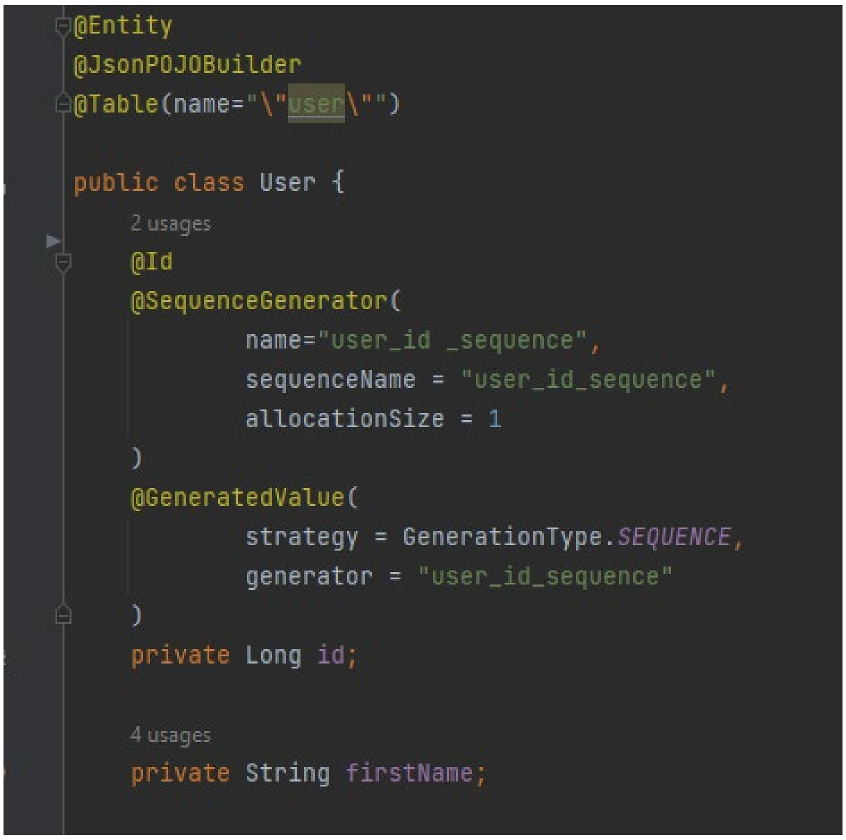
As the choice for PostgreSQL is only motivated by the fact that the developers worked with the technology previously and had a good understanding of it's language features.

More information about gRPC, Springboot and JPA/Hibernate can be found at chapter 3.2

4.1.2 Code snippets

This chapter will contain explanations and images of actual code written in this tier.

Everything in the tier begins from declaring the domain classes as entities for the JPA's database generation feature



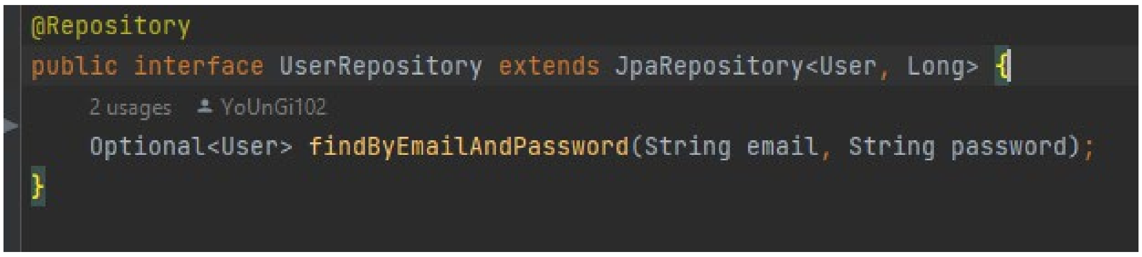
```
@Entity
@JsonPOJOBuilder
@Table(name="\"user\"")

public class User {
    2 usages
    @Id
    @SequenceGenerator(
            name="user_id _sequence",
            sequenceName = "user_id_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "user_id_sequence"
    )
    private Long id;

    4 usages
    private String firstName;
```

Figure 21 – Declaration of a domain class as entity

As seen above JPA/Hibernate requires the @Entity annotation for the class to be recognised as entity and be generated in the database. By using the annotation @Table, the programmer can choose to name the table, otherwise JPA/Hibernate will assign the name of the class as the name of the table. Even tough declaring the id is not necessary, by declaring the id manually using the @Id annotation and declaring the "id" as a sequenced value, the database will be able to ensure that every object inside the entity will have a unique id.

If the @Entity annotation is placed, JPA/Hibernate will take all the variables from the class and consider them as column in the future table. To exclude any variable, the @Transient annotation can be used.

After declaring all the domain classes as Entities, the system will require a way to communicate with the generated database. For this JPA/Hibernate comes with basic methods such as save(), delete(), get() and so on. The user also has the opportunity to declare his/her own methods.



```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    2 usages   YoUnGi102
    Optional<User> findByEmailAndPassword(String email, String password);
}
```

Figure 22 – UserRepository interface

The annotation @Repository will indicate JPA/Hibernate that the following interface is going to extend the JpaRepository<User,Long> interface and use it to perform interactions with the database. The Repository is not required to contain anything else, but in some cases the user might declare other methods for ease of use.
The method “findByEmailAndPassword” is a user created method that will return the user that contains the matching pair of email and password.
In terms of JPA/Hibernate the process is to be repeated for all the domain classes.
As of now, the system can communicate with the database efficiently and easy, so now the implementation of gRPC begins.

Implementation of gRPC starts by creating the .proto files.

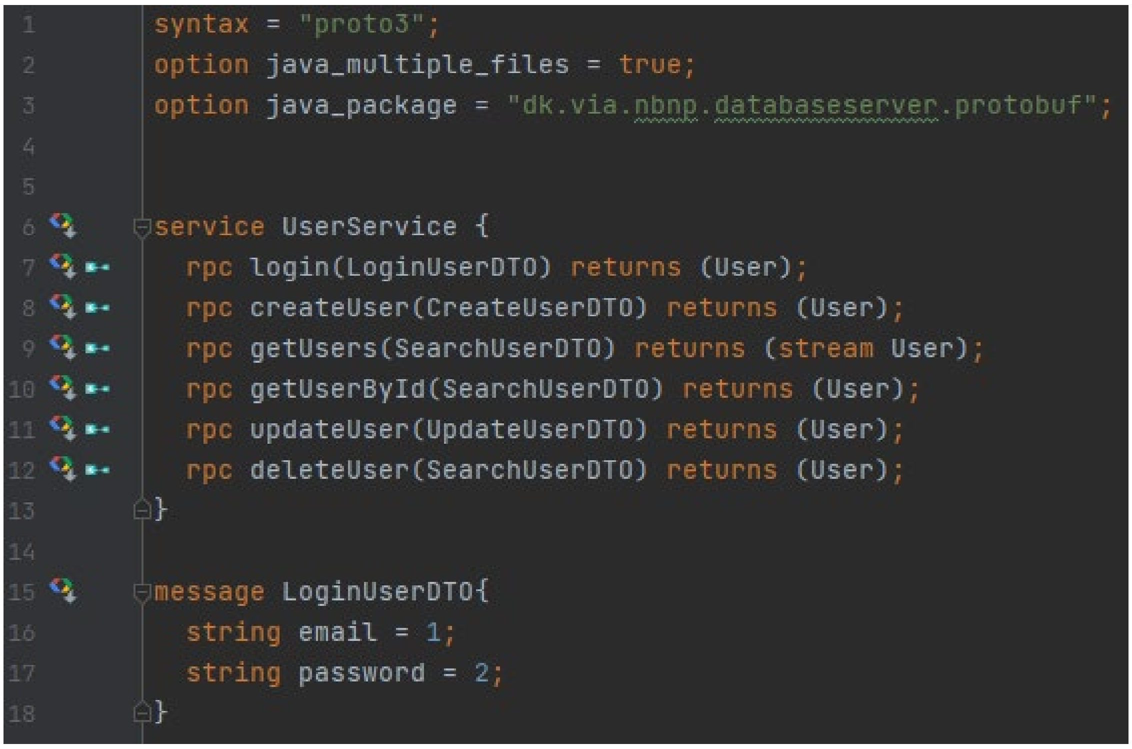
A proto file must specify the version of proto used and can include optional generation statements.

An important note is that the .proto file is going to be used in both server side (3rd tier) as well as in the client side (2nd tier).

In this file the programmer can define one or more services that will be used as methods later on in the code.

The “rpc” key word at the beginning of all the methods indicates that those methods are used for communication between the server and the client. The “returns” key word is used to define what will the method return after execution

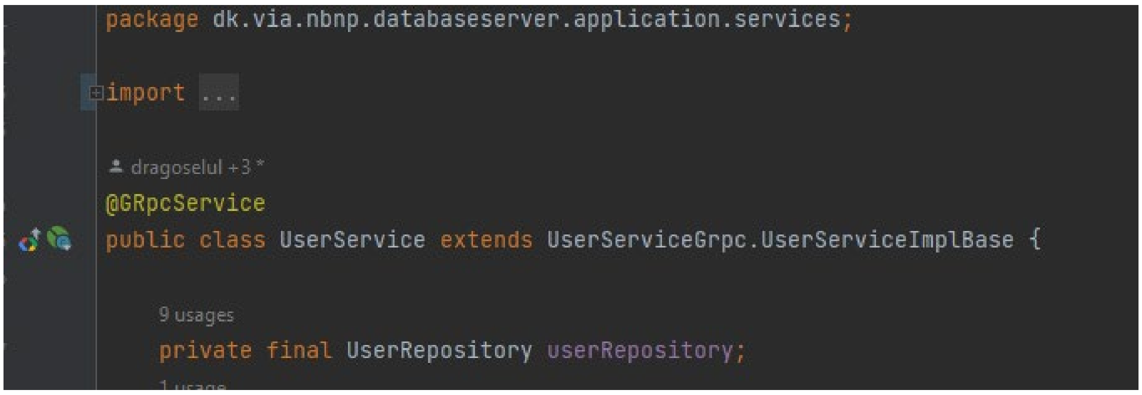
All the parameters inside the methods and all the objects returned are declared inside the proto file as messages. In this example “LoginUserDTO” is defined as a message and contains two string variables that are going to serve as the email and password for the user.Important note about the defined messages is that the User used inside the roto file is not the same as the one inside the domain package despite the fact that they have the same variables.



```
syntax = "proto3";
option java_multiple_files = true;
option java_package = "dk.via.nbnp.databaseserver.protobuf";


service UserService {
  rpc login(LoginUserDTO) returns (User);
  rpc createUser(CreateUserDTO) returns (User);
  rpc getUsers(SearchUserDTO) returns (stream User);
  rpc getUserById(SearchUserDTO) returns (User);
  rpc updateUser(UpdateUserDTO) returns (User);
  rpc deleteUser(SearchUserDTO) returns (User);
}

message LoginUserDTO{
  string email = 1;
  string password = 2;
}
```

Figure 23 – User proto file – Java View

After the proto file has been written, the system will have to generate the protobuf. By using maven the protobuf can be generated easily and reliable, here all the messages and all the services declared in the proto file will be generated as java classes. Furthermore, the gRPC's server interface is now available for the developer to implement.



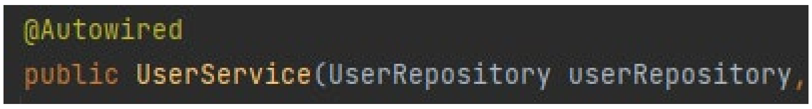
```
package dk.via.nbnp.databaseserver.application.services;

import ...

dragoselul +3 *
@GRpcService
public class UserService extends UserServiceGrpc.UserServiceImplBase {

    9 usages
    private final UserRepository userRepository;
```

Figure 24 – gRPC Service implementation

To start with, the implementation class of the gRPC server implementation is required to be annotated as a @GRpcService, otherwise the Springboot constructor will not initialize the class as a gRPC implementation, thus the server not being recognized. Furthermore the class must extend the service's implementation base such as all the "rpc" methods created in the proto file will be inherited and overridden by the developers necessity.

The instance variable UserRepository will be used to create queries and contact the database for any information requested through the server.

The following annotation @Autowired is used for constructor injection. Constructor injection is the method used by Springboot in order to initialize the class once and be available throughout the tier.



```
@Autowired
public UserService(UserRepository userRepository,
```

Figure 25 – Injection constructor

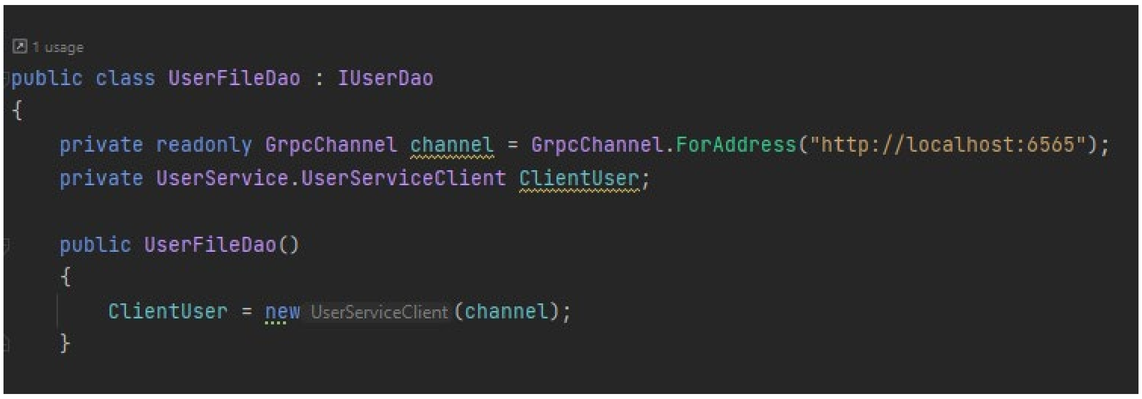
4.2 Business Logic Tier

4.2.1 Implementation in C#

The system is implemented in C# due to the easy of use with the presentation tier. The choice was also affected when choosing the way of communicating between the Data Access Tier and this tier, since the system was meant to be heterogenous, gRPC was chosen as it is available in both Java and C#.
As the stakeholders presented a wish for the system to be available for deployment on a tight schedule, the product owner chose Blazor on the Presentation Tier, thus in turn making the choice indirectly for the Business Logic Tier as well.

4.2.2 Code Snippets

For the sake of continuity, the following subchapter will continue with the gRPC implementation from the last chapter.
As of for the client side of gRPC, other than copying the proto files, the client must contain information about the IP address and port of the server implementation side. In turn, if the information is not provided or incorrect the client will display a “Not implemented” error.

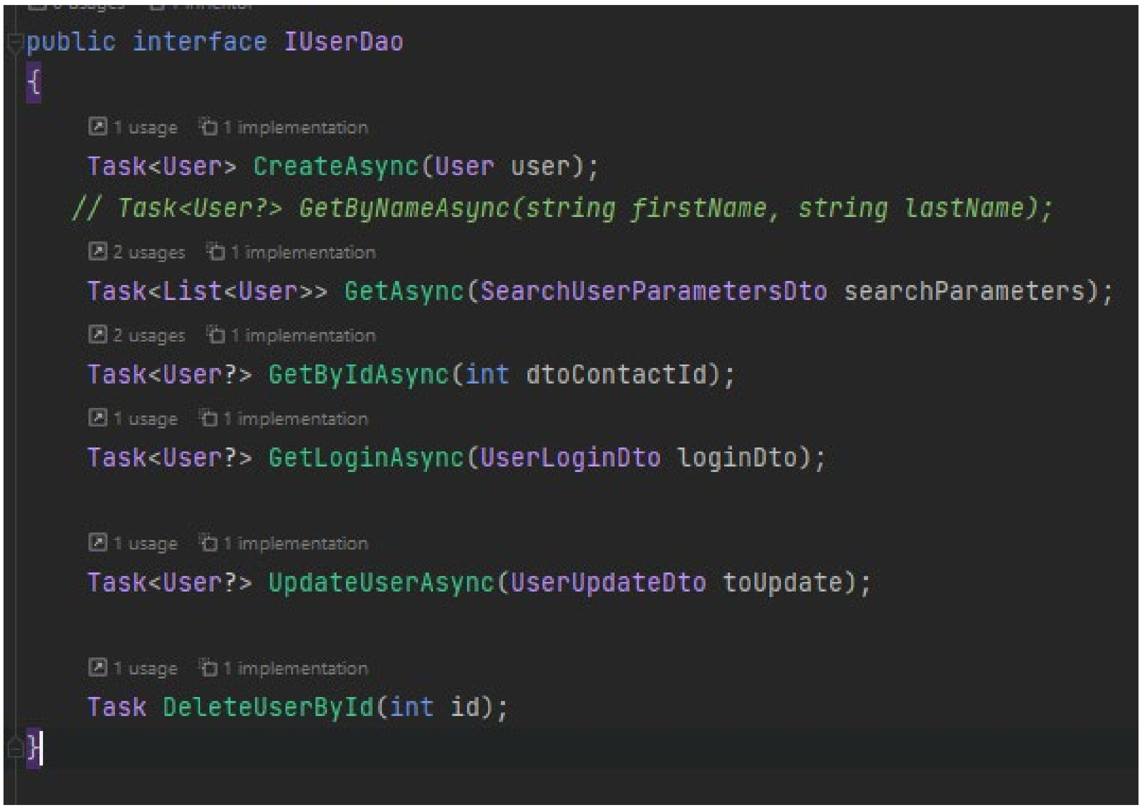


```
1 usage
public class UserFileDao : IUserDao
{
    private readonly GrpcChannel channel = GrpcChannel.ForAddress("http://localhost:6565");
    private UserService.UserServiceClient ClientUser;

    public UserFileDao()
    {
        ClientUser = new UserServiceClient (channel);
    }
```

Figure 26 – gRPC client connection channel

The above image shows how the GrpcChannel is created, and how it is initialized.

Here instead of extending the ImplBase of the gRPC and making the class a server as well, the programmer will create a variable of UserServiceClient and initialize it with the gRPC channel that has been created just above it.
Notice that the class implements the IUserDao which in fact contains the methods needed by the Presentation Tier.

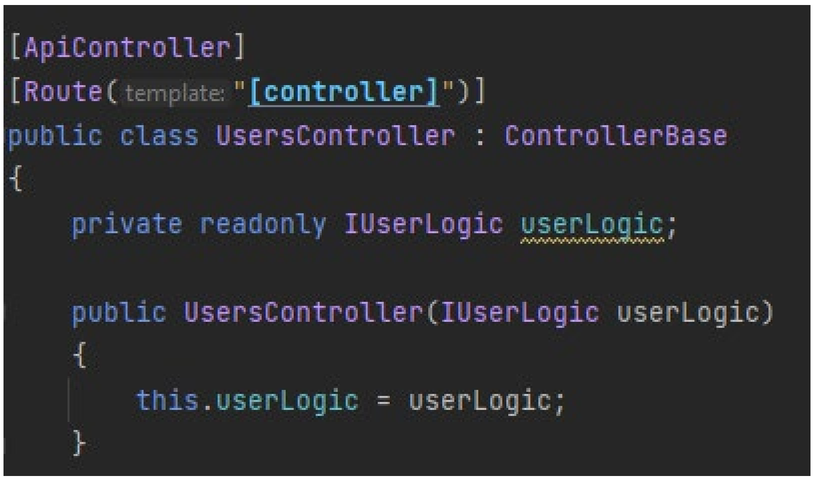


```
public interface IUserDao
{
    1 usage   1 implementation
    Task<User> CreateAsync(User user);
   // Task<User?> GetByNameAsync(string firstName, string lastName);
    2 usages   1 implementation
    Task<List<User>> GetAsync(SearchUserParametersDto searchParameters);
    2 usages   1 implementation
    Task<User?> GetByIdAsync(int dtoContactId);
    1 usage   1 implementation
    Task<User?> GetLoginAsync(UserLoginDto loginDto);

    1 usage   1 implementation
    Task<User?> UpdateUserAsync(UserUpdateDto toUpdate);

    1 usage   1 implementation
    Task DeleteUserById(int id);
}
```

Figure 27 – IUserDao interface

The job of this class is to make sure that the information received from the Data Access Tier is valid, contains what is need and also make sure that the gRPC objects received are mapped back to the Domain classes.
After this part of the code, there comes the part where REST API must be implemented. It is a very crucial part of the code since this is the way of the Presentation Tier to communicate with the Business Logic Tier.

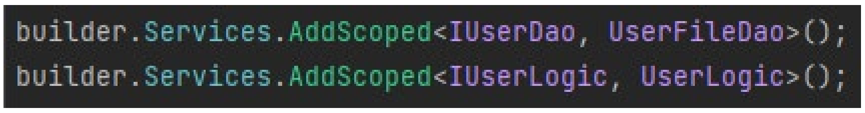
There are annotations also in C# which are inserted inside [ApiController].



```
[ApiController]
[Route( template: "[controller]")]
public class UsersController : ControllerBase
{
    private readonly IUserLogic userLogic;

    public UsersController(IUserLogic userLogic)
    {
        this.userLogic = userLogic;
    }
```

Figure 28 – REST controller

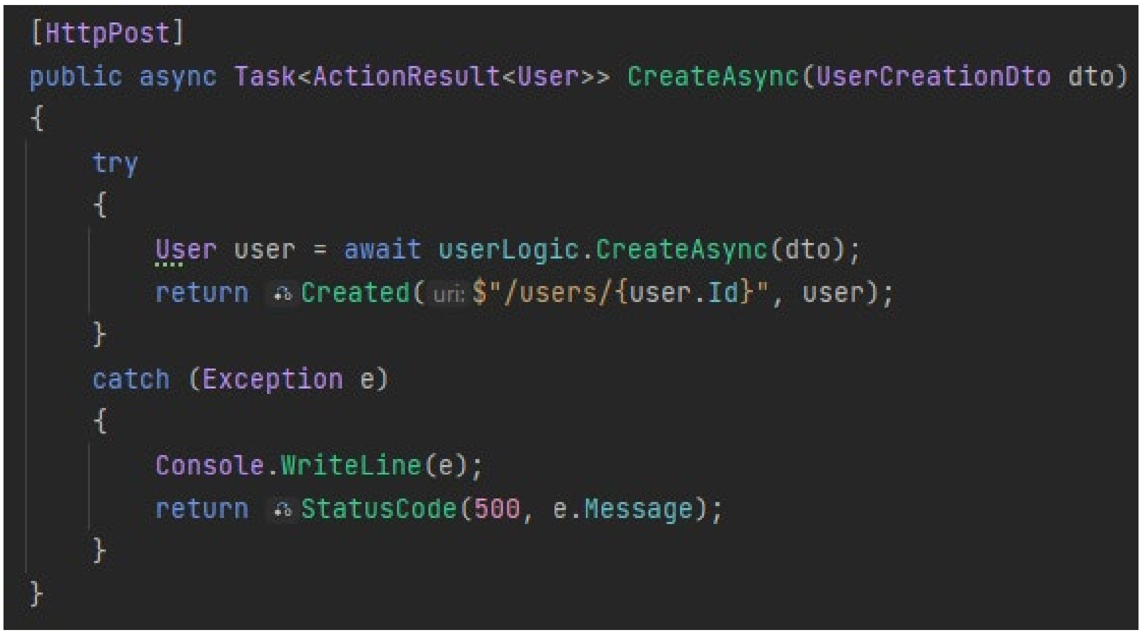
The [ApiController] annotation is used to instruct C#'s compiler that this class will perform CRUD operations inside it. The [Route("[controller]")] is used to create a URL such that everything is consistent. For example, the web URL https://localhost:7171 applied in the above image. For the system to differentiate which REST controller is accessing, the URL must change from the previous https://localhost:7171 to https://localhost:7171 + / + The name of the class without the "Controller" part which in this case will give https://localhost:7171/Users
The IUserLogic variable will make sure the information received arrives the gRPC client and will get back the response from the Data Access Tier.
C# uses injection on a similar level as Java for the Controller, as of IUserLogic, the interface will be initialized in the main class where the builder will take it as a service inside Program.cs.



```
builder.Services.AddScoped<IUserDao, UserFileDao>();
builder.Services.AddScoped<IUserLogic, UserLogic>();
```

Figure 29 – Injection in C#

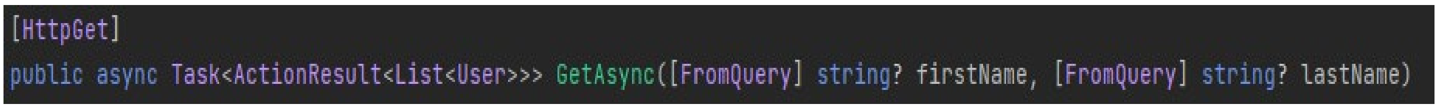
Information gets through REST by providing either JSON strings, or querrys



```
[HttpPost]
public async Task<ActionResult<User>> CreateAsync(UserCreationDto dto)
{
    try
    {
        User user = await userLogic.CreateAsync(dto);
        return Created(uri: $"/users/{user.Id}", user);
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        return StatusCode(500, e.Message);
    }
}
```

Figure 30 – Creating a user in REST

The above screenshot is used to create a User. REST uses CRUD as the template for the annotations, for example the [HttpPost] annotation above will be used to create always. As seen above the method also contains a parameter, a Data Transfer Object to be precise, which will be sent from the Presentation Tier to the Business Logic Tier as a JSON string. On the other hand, the bellow image shows that it is not the only way it can receive information, the use of queries is also very common among REST.



```
[HttpGet]
public async Task<ActionResult<List<User>>> GetAsync([FromQuery] string? firstName, [FromQuery] string? lastName)
```

Figure 31 – Having queries inside a REST method

Data Transfer Objects as name can imply, their main purpose is to make transferring data easier. For example, there may be cases like a Login when the system does not require all the information from a user just to perform an authorization. It is logical also that it is way easier to handle only the email and the password rather than filtering through the data to get everything needed for a simple login. Thereafter the system is cleaner and easier to change in case of any design flaw noticed or design rechoice.

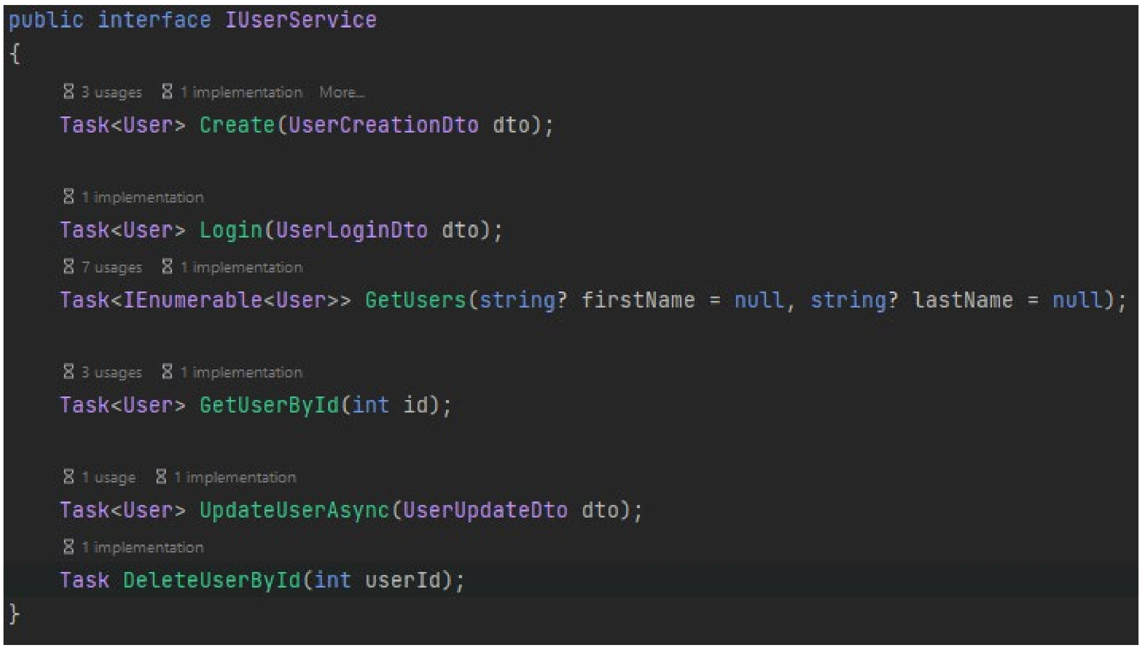
4.3 Presentation Tier

4.3.1 Implementation in C# Blazor

As stated in the previous subchapter, the Product Owner together with the stakeholders decided that for a fast and reliable implementation Blazor should be used because of its high support from Microsoft, which also provides extremely important documentation for it. Even tough REST supports heterogenous systems, implementing everything in the same language was a very easy task and made the development go smoother.

4.3.2 Code Snippets

To keep continuity furthermore the code snippets will start where the REST request are made.

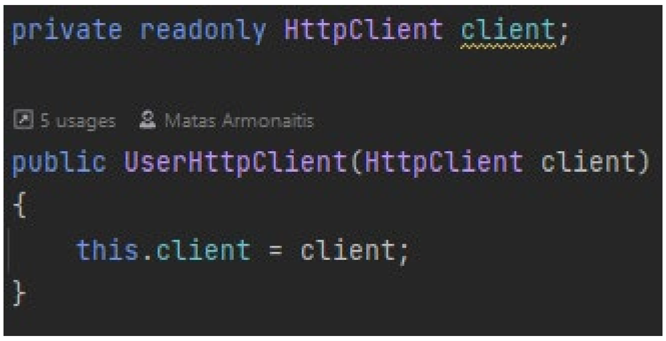


```
public interface IUserService
{
    3 usages   1 implementation   More...
    Task<User> Create(UserCreationDto dto);

    1 implementation
    Task<User> Login(UserLoginDto dto);
    7 usages   1 implementation
    Task<IEnumerable<User>> GetUsers(string? firstName = null, string? lastName = null);

    3 usages   1 implementation
    Task<User> GetUserById(int id);

    1 usage   1 implementation
    Task<User> UpdateUserAsync(UserUpdateDto dto);
    1 implementation
    Task DeleteUserById(int userId);
}
```

Figure 32 – IUserService REST methods

This is more or less the same interface as the IUserDao (Figure TODO), those methods are used for all the tasks needed throughout the website.

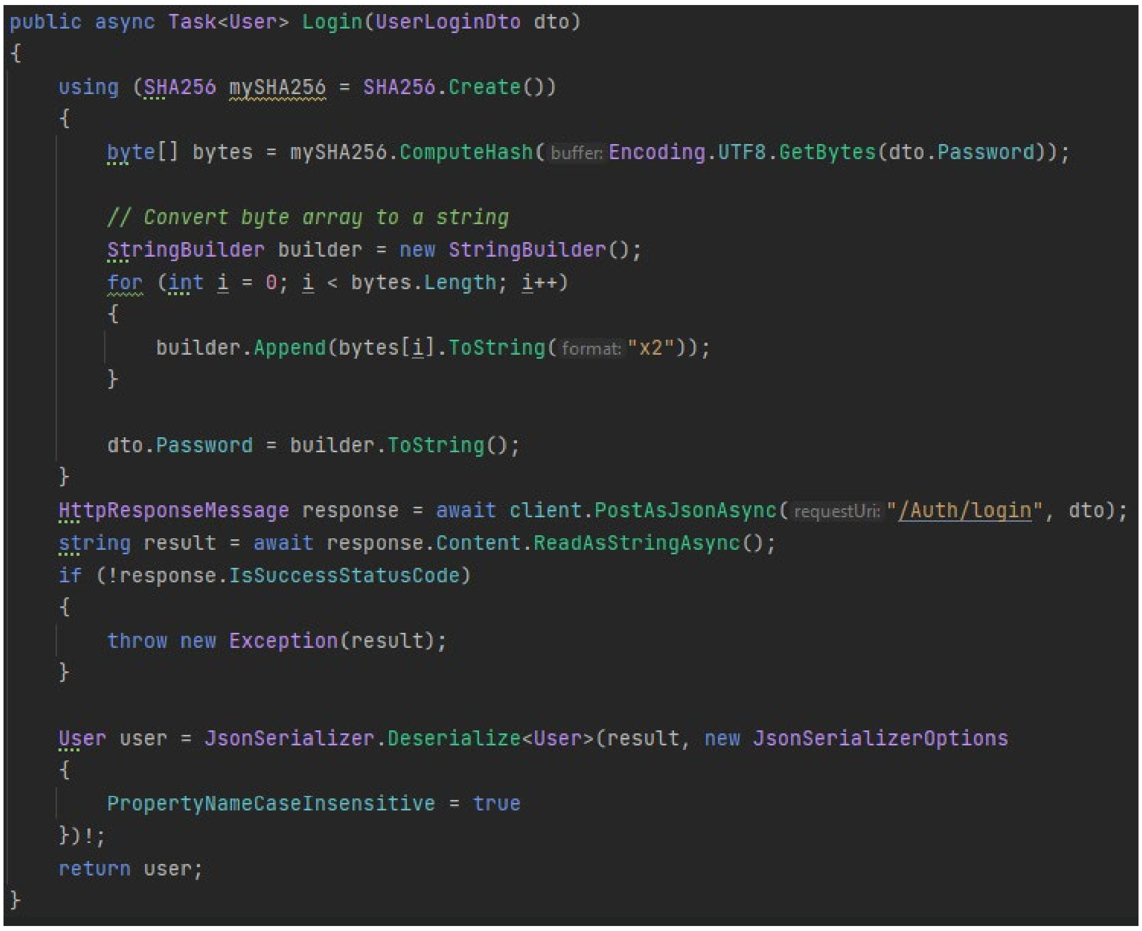
Furthermore, the implementation will imply a HttpClient, which will need URL’s to make requests for the REST controllers.



```
private readonly HttpClient client;

5 usages   Matas Armonaitis
public UserHttpClient(HttpClient client)
{
    this.client = client;
}
```

Figure 33 – HttpClient and Injection

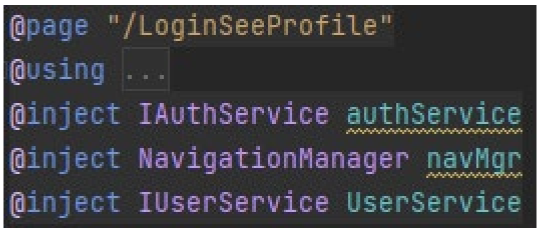
In the above picture there is the HttpClient and the injection that takes place in the class implementing the IUserService interface.



```
public async Task<User> Login(UserLoginDto dto)
{
    using (SHA256 mySHA256 = SHA256.Create())
    {
        byte[] bytes = mySHA256.ComputeHash( buffer: Encoding.UTF8.GetBytes(dto.Password));

        // Convert byte array to a string
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < bytes.Length; i++)
        {
            builder.Append(bytes[i].ToString( format: "x2"));
        }

        dto.Password = builder.ToString();
    }
    HttpResponseMessage response = await client.PostAsJsonAsync( requestUri: "/Auth/login", dto);
    string result = await response.Content.ReadAsStringAsync();
    if (!response.IsSuccessStatusCode)
    {
        throw new Exception(result);
    }

    User user = JsonSerializer.Deserialize<User>(result, new JsonSerializerOptions
    {
        PropertyNameCaseInsensitive = true
    })!;
    return user;
}
```

Figure 34 – Login method

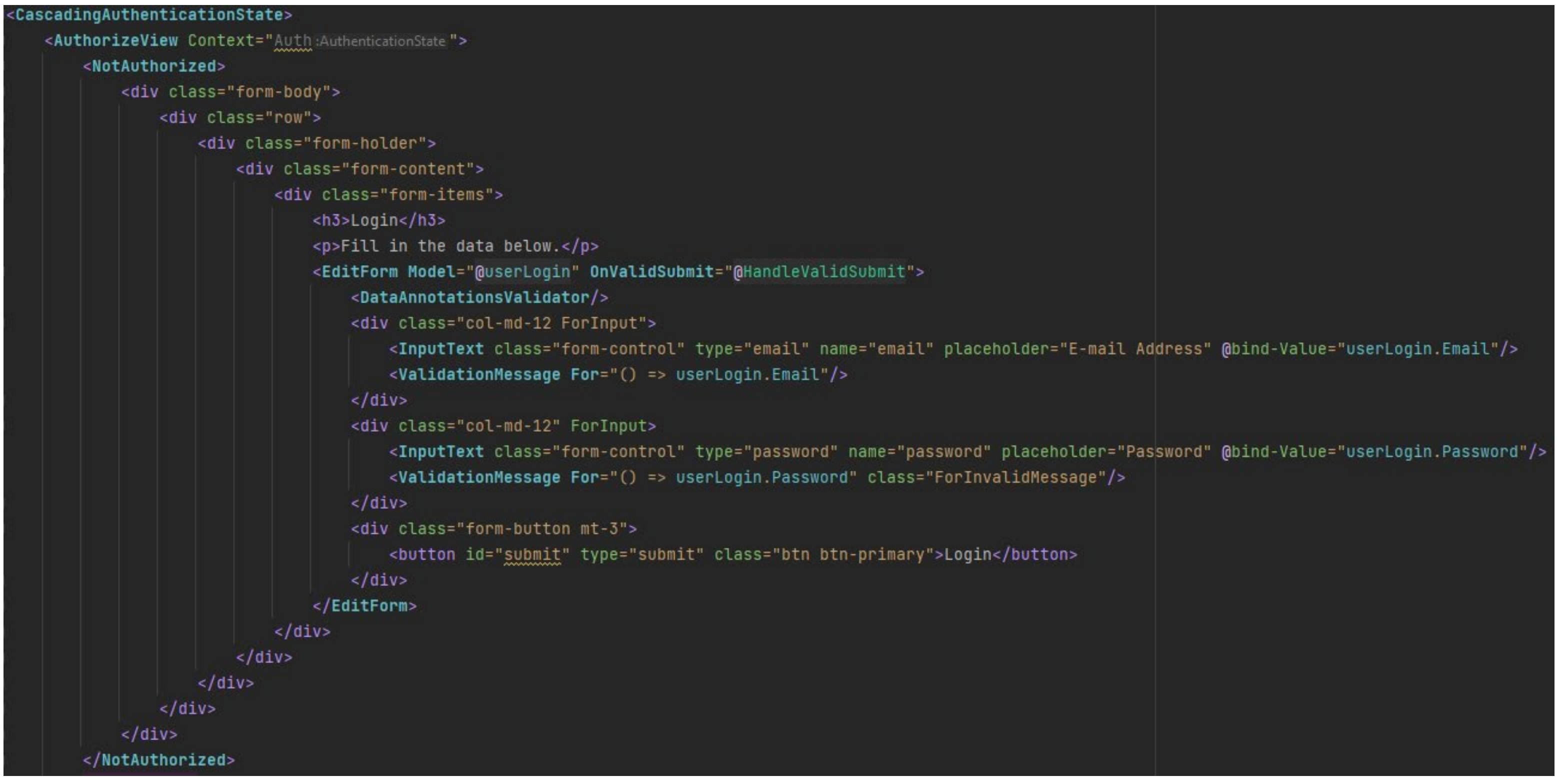
The above picture represents the Login method which hashes the password in SHA256(Anon., 2022g) after which the client creates a PostAsJsonAsync message, where the programmer provides the URL for the controller and the method that needs to be accessed, and the provided object that will be parsed to JSON, in the above case is the UserLoginDto. After a reply message is returned the JSON is then deserialized to the object declared by the programmer and then also returned to the website. If any error occurs or the authorization fails due to non-existing user, password, or email mismatch the method will throw an exception.



```
@page "/LoginSeeProfile"
@using ...
@inject IAuthService authService
@inject NavigationManager navMgr
@inject IUserService UserService
```

Figure 35 – Login page with injection for the services

The above screenshot contains the page URL and injects the IUserService and other services to be used for Loging in.



```
<CascadingAuthenticationState>
    <AuthorizeView Context="Auth:AuthenticationState">
        <NotAuthorized>
            <div class="form-body">
                <div class="row">
                    <div class="form-holder">
                        <div class="form-content">
                            <div class="form-items">
                                <h3>Login</h3>
                                <p>Fill in the data below.</p>
                                <EditForm Model="@userLogin" OnValidSubmit="@HandleValidSubmit">
                                    <DataAnnotationsValidator/>
                                    <div class="col-md-12 ForInput">
                                        <InputText class="form-control" type="email" name="email" placeholder="E-mail Address" @bind-Value="userLogin.Email"/>
                                        <ValidationMessage For="() => userLogin.Email"/>
                                    </div>
                                    <div class="col-md-12" ForInput>
                                        <InputText class="form-control" type="password" name="password" placeholder="Password" @bind-Value="userLogin.Password"/>
                                        <ValidationMessage For="() => userLogin.Password" class="ForInvalidMessage"/>
                                    </div>
                                    <div class="form-button mt-3">
                                        <button id="submit" type="submit" class="btn btn-primary">Login</button>
                                    </div>
                                </EditForm>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </NotAuthorized>
```

Figure 36 – NotAuthorized view for Login

The above image represents the not authorized view of the page, this allows the programmer to create two pages in one, in this case if the user is not authorized, the page will display the login.

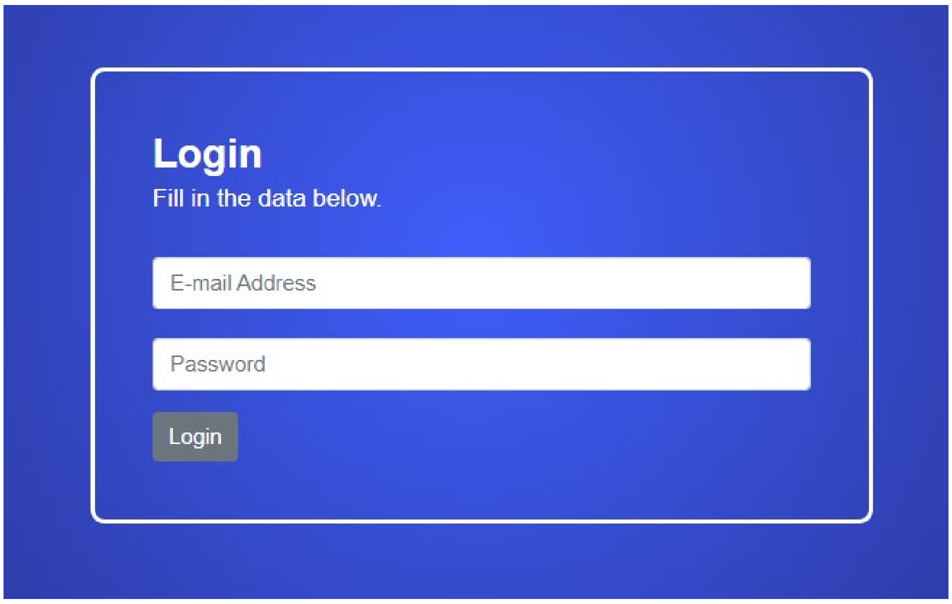


Figure 37 – Login view for not authorized

The page contains an EditForm which validates the data, and will disable the button to login if the input data is incorrect or empty.

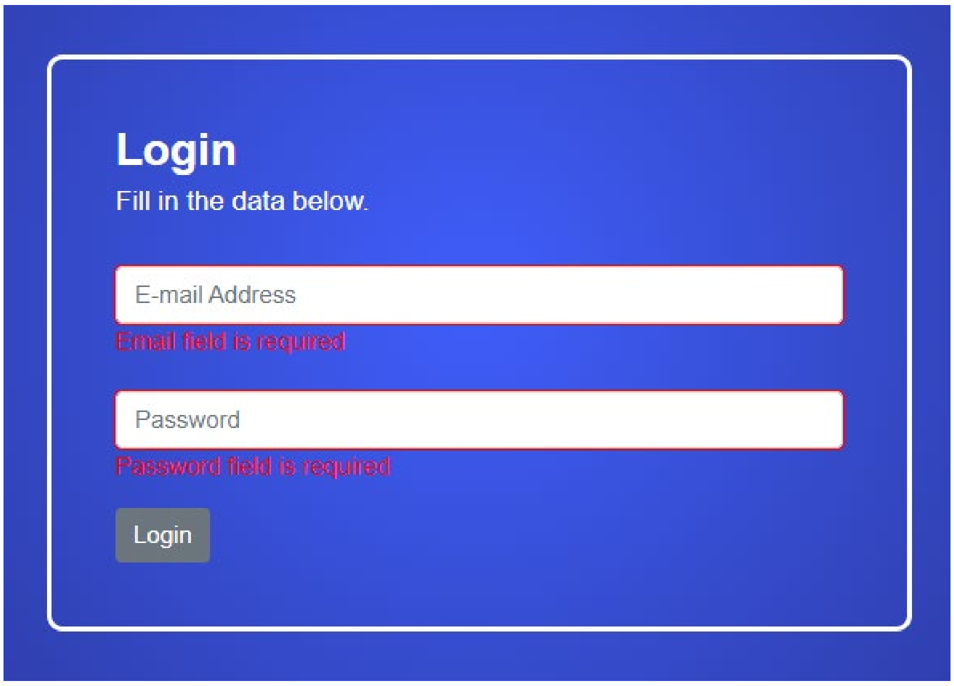
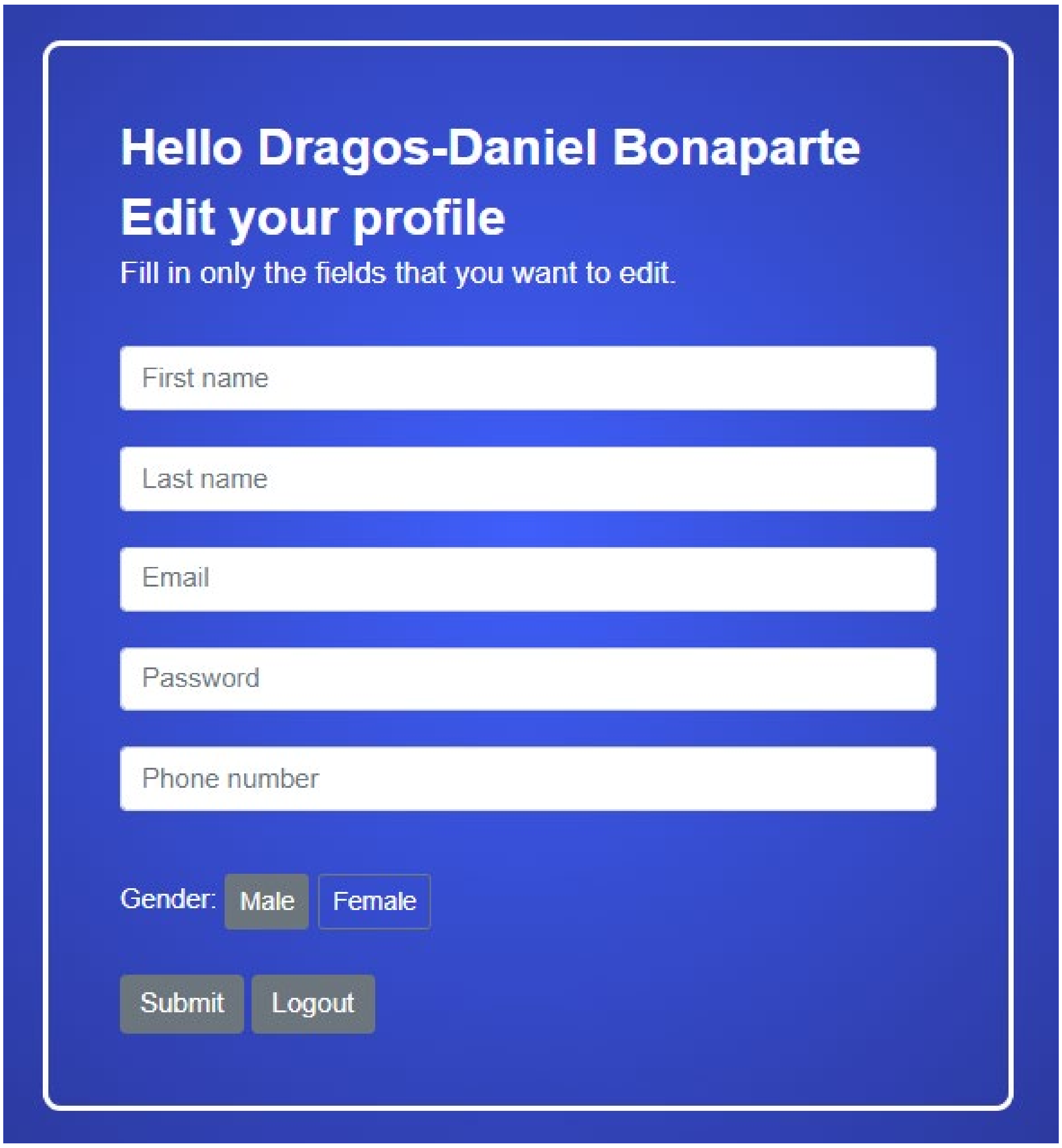


Figure 38 – Login form invalid

Following up is the Authorized view of the page


Hello Dragos-Daniel Bonaparte
Edit your profile
Fill in only the fields that you want to edit.

First name

Last name

Email

Password

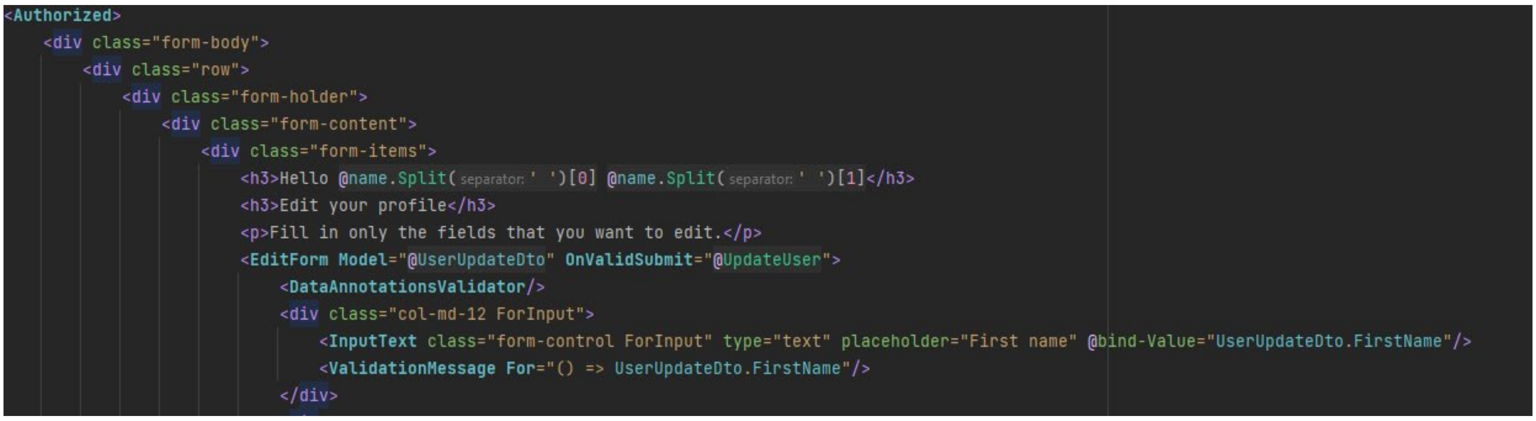
Phone number

Gender: Male Female

Submit Logout

Figure 39 – Edit profile inside Login page

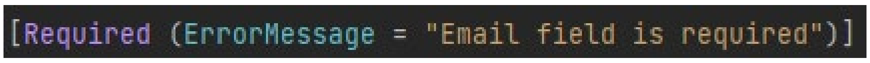
The above screenshot contains as well an EditForm and also has validation for incorrect information or empty.



```
<Authorized>
    <div class="form-body">
        <div class="row">
            <div class="form-holder">
                <div class="form-content">
                    <div class="form-items">
                        <h3>Hello @name.Split( separator: ' ')[0] @name.Split( separator: ' ')[1]</h3>
                        <h3>Edit your profile</h3>
                        <p>Fill in only the fields that you want to edit.</p>
                        <EditForm Model="@UserUpdateDto" OnValidSubmit="@UpdateUser">
                            <DataAnnotationsValidator/>
                            <div class="col-md-12 ForInput">
                                <InputText class="form-control ForInput" type="text" placeholder="First name" @bind-Value="UserUpdateDto.FirstName"/>
                                <ValidationMessage For="() => UserUpdateDto.FirstName"/>
                            </div>
```

Figure 38 – Authorized view of the login page

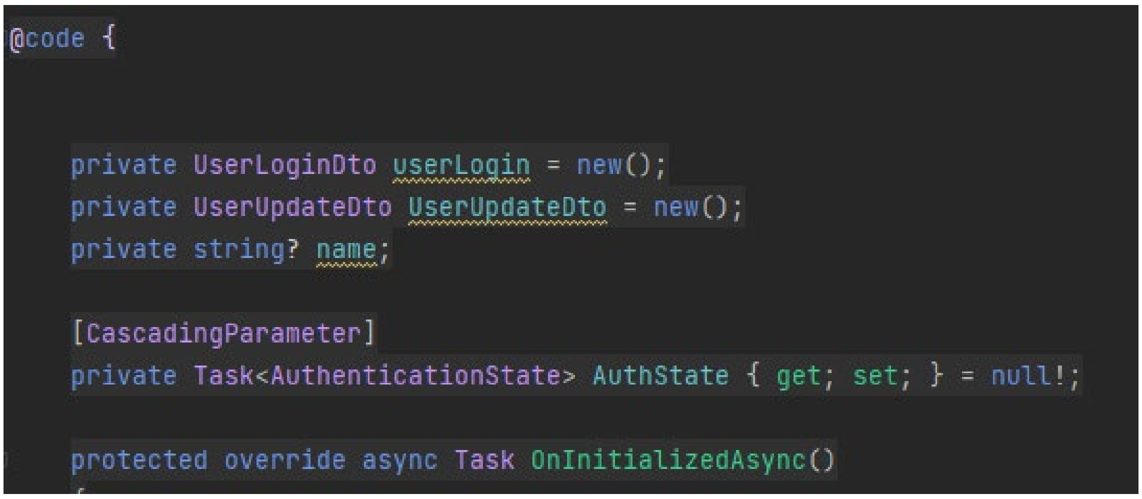
In the above snippet the Authorized tag shows the beginning of the authorized view, and then the EditForm (Anon., 2022f, Anon., 2022o, Anon., 2022d, Anon., 2022c) tag makes sure that the data is validated. If the data is invalid the ValidationMessage (Anon., 2022e) will appear displaying the error message defined in the model of the editForm, in this case the UserUpdateDto does not have any error message so the next screenshot is taken from UserLoginDto.



```
[Required (ErrorMessage = "Email field is required")]
```

Figure 39 – Error message

Because this is a Blazor implementation the page can contain C# code next to the HTML. The OnInitializedAsync() is a overridden method that is executed before the display of the page such as when the page requests the objects to display them, the page has already loaded them in the memory.



```
@code {


    private UserLoginDto userLogin = new();
    private UserUpdateDto UserUpdateDto = new();
    private string? name;

    [CascadingParameter]
    private Task<AuthenticationState> AuthState { get; set; } = null!;

    protected override async Task OnInitializedAsync()
```

Figure 40 – Code inside login page

A very important note is that because the system is going to be new in the market, the Product Owner decided that there will be some processing involved on the client's side such as the filter for the marketplace.(Anon., 2022ab)

5. Test

To make sure that software the Product Owner expects is working, the software was tested every sprint. The test included some unit tests and some manual testing.

5.1 Test Cases

The manual testing was a crucial tool to make sure that everything in the system works as intended and the tiers cooperated with each other the way they were intended. The tests were created based on use case descriptions. For each implemented description a test was done to make sure it is implemented correctly. For these tests we used black box testing.

Scenario	Test scenario	Test steps	Test data	Expected results	Actual results	Pass /Fail
1	Login into the account	1. The user enters his email. 2. The user enters his password		The user is logged in	As expected	Pass
2	Creating a user	1. The user enters his credentials (First name, Last name, e-mail address, phone		The user is registered in the system	As expected	Pass

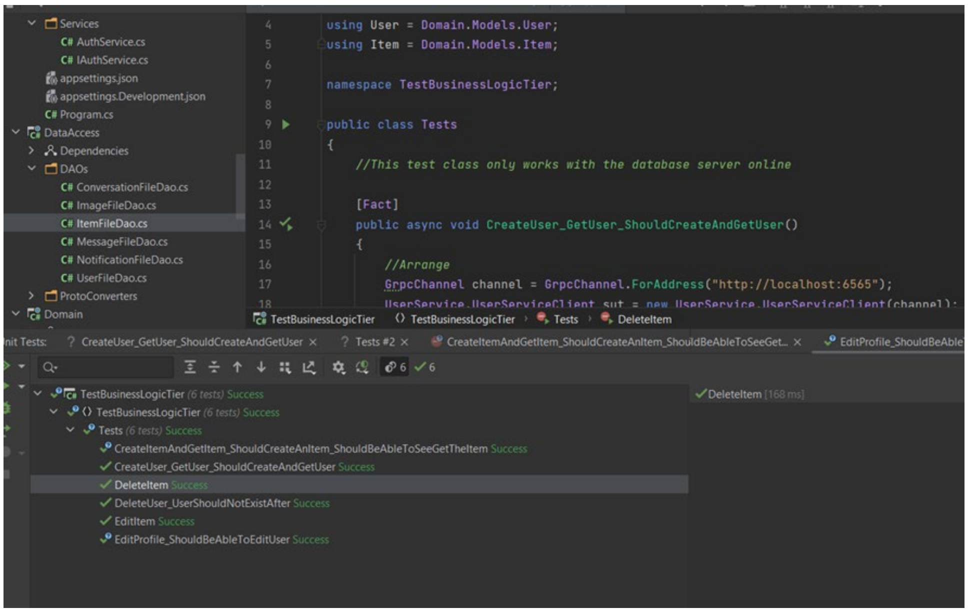
		number, password, and gender) 2. User presses the register button				
3	View Marketplace	1. The user presses the marketplace button. 2. The user will be redirected to the marketplace		The marketplace will show a list with all listened items	As expected	Pass
4	Filtering items	1. The user can filter items by choosing the category, currency, and price range		The marketplace will show the items with filter	As expected	Pass
5	See post details	1. The user presses the marketplace button 2. The user chooses the item he wants to buy		The program will show all the information about that item	As expected	Pass
6	Message seller	1. The user presses the marketplace button 2. The user chooses the item he wants to buy 3. The user presses the message seller button 4. The user will be redirected to messages		The user can see the conversation of the item he wants to buy and to message the seller	As expected	Pass
7	Create post	1. The user must login into his account 2. The user presses the create post button 3. The user enters all the necessary data 4. The user will be redirected to another window to insert a picture		The item will be placed on the marketplace	As expected	Pass

8	Edit post details	1. The user must login into his account 2. The user goes to his items 3. The user chooses the item he wants to edit 4. The user presses the edit button 5. The user edits the fields he wants 6. The user submits the changes		The item will be edited with the new changes both on users' profile and marketplace	As expected	Pass
9	Delete item	1. The user must login into his account 2. The user goes to his items 3. The user chooses the item he wants to edit 4. The user presses the delete button		The item will be deleted from users profile and marketplace	As expected	Pass
10	Edit user	1. The user must login into his account 2. The user presses the profile icon 3. The user chooses the information he wants to change 4. The user submits the changes		The new information about the user will be updated	As expected	Pass
11	Log out	1. The user must login into his account 2. The user presses the profile icon 3. The user presses the log out button		The user will be logged out from his account	As expected	Pass

Table 4 – Test cases

5.2 Business logic tier/Data Access tier tests

In order to test both tiers it was decided to make the tests in the business tier, but to also use the database server from data access tier for testing some of the use cases. Once again use case descriptions were used to create the test cases. This allowed to see when adding additional implementation if there were any bugs that influenced some of already completed use cases without the need for interacting with the presentation tier. Xunit library was used for these tests.



```
using User = Domain.Models.User;
using Item = Domain.Models.Item;

namespace TestBusinessLogicTier;

public class Tests
{
    //This test class only works with the database server online

    [Fact]
    public async void CreateUser_GetUser_ShouldCreateAndGetUser()
    {
        //Arrange
        GrpcChannel channel = GrpcChannel.ForAddress("http://localhost:6565");
        UserService.UserServiceClient sut = new UserService.UserServiceClient(channel);
```

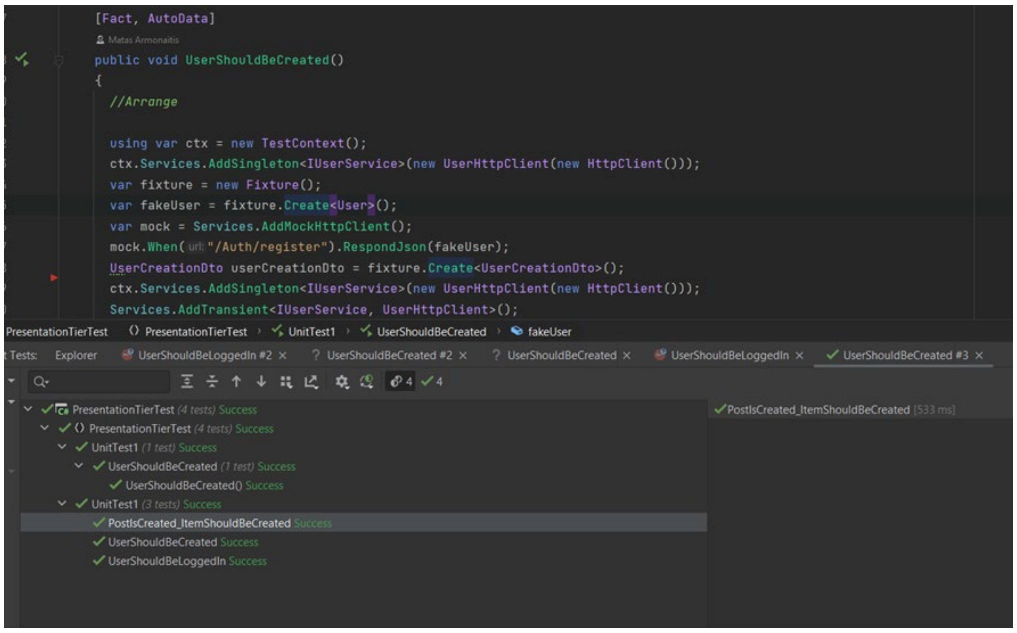
```
TestBusinessLogicTier (6 tests) Success
  TestBusinessLogicTier (6 tests) Success
    Tests (6 tests) Success
      CreateItemAndGetItem_ShouldCreateAnItem_ShouldBeAbleToSeeGetTheItem Success
      CreateUser_GetUser_ShouldCreateAndGetUser Success
      DeleteItem Success
      DeleteUser_UserShouldNotExistAfter Success
      EditItem Success
      EditProfile_ShouldBeAbleToEditUser Success

DeleteItem [168 ms]
```

Figure 41 – Business Logic Tier/Data Access Tier tests

5.3 Unit tests

Because of the amounts of manual testing, testing in other tiers and the difficulty of automated tests, the development team decided to only implement necessary tests for the presentation tier. The tests are made to work without being dependant on other tiers, meaning it required mocking and quite a bit of research in order to make them work. Only necessary tests for the presentation tier to work were implemented in order for the development team to fit in to the time schedule.

Bunit and Nunit.autofixture frameworks and white box together with aaa(Arrange, Act, Assert) strategy was used to implement the unit tests. Bunit allowed to make the tests independent from the other tiers and autofixture allowed us to input random data without the need of manual input. In total 3 Tests for the presentation tier were implemented, which all passed in the end. In order to keep other more simple units like setters and getters working properly, manual testing was used throughout all of the sprints.



```
[Fact, AutoData]
Matas Armonaitis
public void UserShouldBeCreated()
{
  //Arrange

  using var ctx = new TestContext();
  ctx.Services.AddSingleton<IUserService>(new UserHttpClient(new HttpClient()));
  var fixture = new Fixture();
  var fakeUser = fixture.Create<User>();
  var mock = Services.AddMockHttpClient();
  mock.When( url: "/Auth/register").RespondJson(fakeUser);
  UserCreationDto userCreationDto = fixture.Create<UserCreationDto>();
  ctx.Services.AddSingleton<IUserService>(new UserHttpClient(new HttpClient()));
  Services.AddTransient<IUserService, UserHttpClient>();
```

```
PresentationTierTest  PresentationTierTest › UnitTest1 › UserShouldBeCreated › fakeUser
Tests:  Explorer  UserShouldBeLoggedIn #2 ×  ? UserShouldBeCreated #2 ×  ? UserShouldBeCreated ×  UserShouldBeLoggedIn ×  ✓ UserShouldBeCreated #3 ×

✓ PresentationTierTest (4 tests) Success                          ✓PostIsCreated_ItemShouldBeCreated [533 ms]
  ✓ PresentationTierTest (4 tests) Success
    ✓ UnitTest1 (1 test) Success
      ✓ UserShouldBeCreated (1 test) Success
        ✓ UserShouldBeCreated() Success
    ✓ UnitTest1 (3 tests) Success
      ✓ PostIsCreated_ItemShouldBeCreated Success
      ✓ UserShouldBeCreated Success
      ✓ UserShouldBeLoggedIn Success
```

Figure 42 – Presentation Tier tests

6. Results and Discussion

22 functional requirements total, including 7 critical, 3 high, 9 medium, and 3 low priority requirements, are listed in the Product Backlog. At first, the main objective of the development team was to satisfy the Product Owner by at least implementing the essential priority needs. The needs that were chosen for each sprint of the development process did not always begin with the most important and work their way down in priority; instead, if some of the lower priorities could be implemented in conjunction with the higher priorities, they were also added to the sprint.

Out of a total of 22, 11 conditions have been totally met. Due to a lack of time, the team opted to concentrate on the most crucial components of the program and make sure they function as intended and have through all necessary testing. The most crucial components of the software must be functional, and the other features can be added later, it was concluded after discussing this with the product owner.

The most challenging aspects of the project throughout development were encountered when implementing the critical requirements. Examples would include adding images to the Items, which needed extensive research to ensure it looked and worked as good as possible. The following example would be the time-consuming and labor-intensive process of integrating the data access and business logic tiers. Making the marketplace filtering function was the third challenging aspect because the team lacked much blazor and css experience. Fortunately, the team dedicated a lot of time to these implementations during the sprints, but it still took longer to implement than expected. The last issue was estimating how much work could be accomplished in each sprint. The team almost always fell behind schedule because they assigned too many tasks during each sprint, believing they could finish them on time.

At the conclusion of the project, the development team tested each developed feature to ensure that it is functional, and the product owner approved it after ensuring that it functions as intended. The program was black box and white box tested through manual testing and unit testing. In the end, the product owner approved it based on the requirements of the stakeholders.

7. Conclusions

The request of the non-profit organization was to help towards achievement of one of the United Nations goals towards sustainable future, specifically goal number 12 – responsible production and consumption. A decision was made to develop a system where people could easily give away their unused things or even sell them for some extra money instead of throwing them away in the garbage.

The solution was based on the need to have a simple system that can be used by anyone, so simplicity was a priority while developing the system. Scrum and Unified process were used as a development framework.

The system consists of 23 requirements (22 functional and 1 nonfunctional), which are covered in 6 use cases. The use cases were centered mainly around users, items, conversations and messages. In order to differentiate between users a login system was implemented, and each user is logged in using an email address and a password. If a user chooses not to register an account, they can still browse the items for sale. However, they will not be able sell items themselves or to contact the seller using the messaging system provided by the application. They will therefore have to contact the seller using a different means (for example phone number or email address).

Following these requirements, a decision was made to split the implementation into three parts: presentation (containing the user interface), business logic (responsible mainly for validation of the data) and database (responsible for storing data). The system was afterwards designed in a heterogeneous way. In order to achieve that HTTP protocol with RESTful API was used between presentation and business tier, and gRPC was used between business and database tier. The presentation tier was developed using Blazor Web Assembly, business logic was developed using .NET 6.0 and database tier using Java Spring Boot.

Implementation and testing were done hand in hand, meaning that after each finished requirement a thorough testing was concluded in order to make sure everything is working properly in both happy and unhappy scenarios. Black box testing was concluded using Swagger (for business tier), BloomRpc (for database tier). The user interface was tested manually by entering different values and trying different scenarios. The white box is done using BUnit, XUnit and NUnit.autowire.

The end of the project resulted in all critical and most of high priority requirements were met, which resulted in a working system that can be used by the end users.

8. Project future

A product's goal is to please its customers and give them high-caliber services. Because of this, a major priority during the entire development process was to admit the system's flaws and concentrate on fixing them in the future to give users a more positive experience.

Implementing the security elements that were specified and covered in the design part would thus be the first step in a future upgrade plan. The system is currently susceptible to the vast majority of assaults that could compromise the system and its data, which is the cause of this. Therefore, it's crucial to make sure the system offers a safe environment for users to communicate in.

The delimitations mentioned in the Project Description were also among the initial features that the development team thought about including in a later version of the system. Implementing financial transactions within the system, for instance, would be a useful feature. This will enable system users to use their chosen means of making financial transactions, whether it be through the system's built-in features or a third-party API like MobilePay.

The management of shipping is a crucial feature that will need to be included in the future in order to provide system users with a positive user experience. The technology now only allows consumers to make purchases; however, shipping is taken care of off-camera.

The features that were not used throughout the development phase are additional features that might be considered during future developments. The "report" feature wasn't essential, but it would undoubtedly enhance the user experience. The same would apply to features that are specific to the products of the business, including having time-limited events, signing up for an item presale and automatically buying the item when it goes on sale.

When adding new features and upgrading the system, there are a few factors related to the code implementation that could also be considered. One of them is deeper class separation to concentrate on managing resources for just one functionality.

9. Sources of information

Banger, D., 2014. A Basic Non-Functional Requirements Checklist « Thoughts from the Systems front line.... Available at: https://dalbanger.wordpress.com/2014/01/08/a-basic-non-functional-requirements-checklist/ [Accessed January 31, 2017].

Business Analyst Learnings, 2013. MoSCoW : Requirements Prioritization Technique — Business Analyst Learnings. , pp.1–5. Available at: https://businessanalystlearnings.com/ba-techniques/2013/3/5/moscow-technique-requirements-prioritization [Accessed January 31, 2017].

Dawson, C.W., 2009. *Projects in Computing and Information Systems*, Available at: http://www.sentimentaltoday.net/National_Academy_Press/0321263553.Addison.Wesley.Publishing.Company.Projects.in.Computing.and.Information.Systems.A.Students.Guide.Jun.2005.pdf.

Gamma, E. et al., 2002. *Design Patterns – Elements of Reusable Object-Oriented Software*, Available at: http://books.google.com/books?id=JPOaP7cyk6wC&pg=PA78&dq=intitle:Design+Patterns+Elements+of+Reusable+Object+Oriented+Software&hl=&cd=3&source=gbs_api%5Cnpapers2://publication/uuid/944613AA-7124-44A4-B86F-C7B2123344F3.

IEEE Computer Society, 2008. *IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation*,

Larman, C., 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*,

Mendeley.com, 2016. Homepage | Mendeley. Available at: https://www.mendeley.com/ [Accessed February 2, 2017].

YourCoach, S.M.A.R.T. goal setting | SMART | Coaching tools | YourCoach Gent. Available at: http://www.yourcoach.be/en/coaching-tools/smart-goal-setting.php [Accessed August 19, 2017].

Anon. 2022a. *14 CSS Particle Backgrounds*. [online] Available at: <https://freefrontend.com/css-particle-backgrounds/> [Accessed 28 November 2022].

Anon. 2022b. *35 CSS Grid Examples*. [online] Available at: <https://freefrontend.com/css-grid-examples/> [Accessed 28 November 2022].

Anon. 2022c. *An Overview On InputRadioGroup Component In Blazor WebAssembly*. [online] Available at: <https://www.learmoreseekmore.com/2020/12/blazorwasm-inputradiogroup-overview.html> [Accessed 5 December 2022].

Anon. 2022d. *Blazor: How to use the onchange event in <select> when using @bind also? - Stack Overflow*. [online] Available at: <https://stackoverflow.com/questions/58125929/blazor-how-to-use-the-onchange-event-in-select-when-using-bind-also> [Accessed 5 December 2022].

Anon. 2022e. *Blazor University - Validation*. [online] Available at: <https://blazor-university.com/forms/validation/> [Accessed 5 December 2022].

Anon. 2022f. *c# - Binding Type double values in InputText of Blazor app Razor page - Stack Overflow*. [online] Available at: <https://stackoverflow.com/questions/68274549/binding-type-double-values-in-inputtext-of-blazor-app-razor-page> [Accessed 5 December 2022].

Anon. 2022g. *Compute SHA256 Hash In C#*. [online] Available at: <https://www.c-sharpcorner.com/article/compute-sha256-hash-in-c-sharp/> [Accessed 18 November 2022].

Anon. 2022h. *Core concepts, architecture and lifecycle* | *gRPC*. [online] Available at: <https://grpc.io/docs/what-is-grpc/core-concepts/> [Accessed 15 December 2022].

Anon. 2022i. *CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc.* [online] Available at: <https://crackstation.net/> [Accessed 15 December 2022].

Anon. 2022j. *Denial of Service and Prevention - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/deniel-service-prevention/> [Accessed 13 December 2022].

Anon. 2022k. *Facebook Marketplace Rules for you to know before putting up anything on sale* | *Apps*. [online] Available at: <https://www.republicworld.com/technology-news/apps/facebook-marketplace-rules-for-you-to-know-before-putting-up-anything.html> [Accessed 24 September 2022].

Anon. 2022l. *Furniture waste – The forgotten waste stream - Recycle Track Systems*. [online] Available at: <https://www.rts.com/blog/furniture-waste-a-growing-issue/> [Accessed 12 December 2022].
Anon. 2022m. *Global e-waste generation 2010-2019 | Statista*. [online] Available at: <https://www.statista.com/statistics/499891/projection-ewaste-generation-worldwide/> [Accessed 12 December 2022].
Anon. 2022n. *Home - United Nations Sustainable Development*. [online] Available at: <https://www.un.org/sustainabledevelopment/> [Accessed 12 December 2022].
Anon. 2022o. *How do I get the checkbox value if it is checked? | Blazor FAQ | Syncfusion*. [online] Available at: <https://www.syncfusion.com/faq/how-do-i-get-the-checkbox-value-if-it-is-checked> [Accessed 5 December 2022].
Anon. 2022p. *JPA vs Hibernate: Know The Difference - InterviewBit*. [online] Available at: <https://www.interviewbit.com/blog/jpa-vs-hibernate/> [Accessed 15 December 2022].
Anon. 2022q. *List on International eBay Sites*. [online] Available at: <https://galaxy.maropost.com/s/article/list-on-international-ebay-sites> [Accessed 24 September 2022].
Anon. 2022r. *Man in the Middle (MITM) Attacks | Types, Techniques, and Prevention*. [online] Available at: <https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/> [Accessed 13 December 2022].
Anon. 2022s. *Payment methods policy | eBay*. [online] Available at: <https://www.ebay.com/help/policies/payment-policies/payment-methods-policy?id=4269> [Accessed 24 September 2022].
Anon. 2022t. *Policies on restricted or prohibited items – overview | eBay*. [online] Available at: <https://www.ebay.com/help/policies/prohibited-restricted-items/prohibited-restricted-items?id=4207> [Accessed 24 September 2022].
Anon. 2022u. *Protect from cross-site scripting attacks - IBM Garage Practices*. [online] Available at: <https://www.ibm.com/garage/method/practices/code/protect-from-cross-site-scripting/> [Accessed 13 December 2022].
Anon. 2022v. *Shipping and delivery for buyers | eBay*. [online] Available at: <https://www.ebay.com/help/buying/shipping-delivery/shipping-delivery-buyers?id=4005> [Accessed 24 September 2022].

Anon. 2022w. *SOLID: The First 5 Principles of Object Oriented Design | DigitalOcean*. [online] Available at: <https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design> [Accessed 15 December 2022].
Anon. 2022x. *Spring Boot Tutorial | DigitalOcean*. [online] Available at: <https://www.digitalocean.com/community/tutorials/spring-boot-tutorial> [Accessed 15 December 2022].
Anon. 2022y. *The DTO Pattern (Data Transfer Object) | Baeldung*. [online] Available at: <https://www.baeldung.com/java-dto-pattern> [Accessed 15 December 2022].
Anon. 2022z. *The Environmental Crisis Caused by Textile Waste*. [online] Available at: <https://www.roadrunnerwm.com/blog/textile-waste-environmental-crisis> [Accessed 12 December 2022].
Anon. 2022aa. *Threats and Pitfalls.pdf*. [online] Available at: <https://via.itslearning.com/LearningToolElement/ViewLearningToolElement.aspx?LearningToolElementId=2613431> [Accessed 13 December 2022].
Anon. 2022ab. *Web application client vs server side processing*. [online] Available at: <https://www.pragimtech.com/blog/blazor-webAssembly/web-application-client-vs-server-side-processing/> [Accessed 5 December 2022].
Anon. 2022ac. *What Is A Brute Force Attack? How To Prevent It?* [online] Available at: <https://www.cloudways.com/blog/what-is-brute-force-attack/> [Accessed 13 December 2022].
Anon. 2022ad. *What is PostgreSQL*. [online] Available at: <https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/> [Accessed 15 December 2022].
Anon. 2022ae. *What is Three-Tier Architecture | IBM*. [online] Available at: <https://www.ibm.com/cloud/learn/three-tier-architecture> [Accessed 12 December 2022].
Wang, X. and Zhang, W., n.d. Cross-site scripting attacks procedure and Prevention Strategies. [online] https://doi.org/10.1051/03001.

10. Appendices

1. Appendix A – User Interface Design
2. Appendix B – Group Contract
3. Appendix C – Project Description
4. Appendix D – Test Cases
5. Appendix E – User Cases
6. Appendix F – Activity Diagrams
7. Appendix G – EER Diagram
8. Appendix H – System Sequence Diagram
9. Appendix I – Code
10. Appendix J – Sequence Diagram
11. Appendix L – Class Diagram
12. Appendix M – Sprint Burndown Charts
13. Appendix N – Domain Model
14. Appendix O – Astah Files