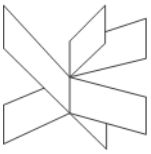


Project Report - GodEats

Group 8

Daniel Railean, 294241

Satish Gurung, 299123

Dimitrian Cebotaru, 299115

Mark Vincze, 224478

Supervisors:

Troels Mortensen

Joseph Chukwudi Okika

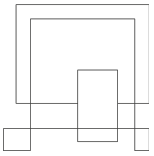
VIA UC

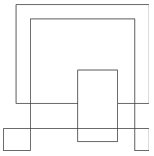
40995 Characters

Software Engineering

3rd Semester

04/06-2021





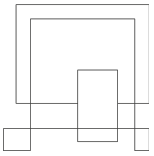
[Daniel]

Abstract

There is a need for a healthy food delivery service on the market. The “GodEats” project was developed to try to address this issue, a project which would supposedly deliver ingredients for healthy food to be cooked at home. The three tier architecture was used with a presentation tier for which C# and the Blazor Framework was utilized, a business tier, for which C# was used and its WebApi functionality, and a data tier, for which Java and the Spring framework was used, together with Hibernate ORM and SQLite database.

The networking technologies used were REST over HTTPS for inter tier communication and SignalR for the live-chat functionality.

The resulting product is a web-application that is able to handle user orders and with small improvement would be able to work in production.



[Mark, Dimitrian, Satish, Daniel]

1 Introduction

Nowadays everyone is fond of travelling, exploring new things, meeting other people and experiencing new cuisine. People like eating and they enjoy trying out new and varied dishes. But whenever the term “healthy eating” comes to mind, every individual thinks of cooking their own food. However it is not possible for everyone to cook food at home. As a result, they prefer food from restaurants as it is easily available and less time consuming for them.

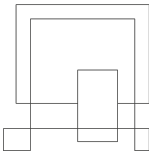
There is a dramatic increase in the restaurant industry. (Figure 1 shows the revenue of the gastronomy industry in Denmark from 2014 to 2018. Over the shown period, the gastronomy industry's revenue increased constantly. It reached its peak in 2018, amounting to a revenue of nearly 45.9 billion Danish kroner). (Denmark: revenue of gastronomy industry 2014-2018 | Statista, 2021)

In order to facilitate the consumption of home-cooked food, the creation of a platform is proposed that would help people with their cooking journey. The platform would also be able to do the calories and macro-nutrients count for an even more conscious consumption of the cooked food.

The aim for this project is to create a platform able to manage a food delivery service. The delimitation for this project is that GodEats will not be able to physically deliver the ordered food. Even though it is intended for use in the Danish market, the user interface will be offered in English Only as most of the Danish know English. (guides and Denmark, 2021)

This project report will guide the reader through the development process of the GodEats web platform. The following chapter will cover the Analysis part.

To read more about the project description go to Appendix A.



[Daniel, Mark]

2 Analysis

In this chapter, the requirements for the system are defined along with diagrams which outline the system structure. A short overview for all the use cases will be presented to better understand their purpose. Two of the use case descriptions will be discussed in greater detail in this report.

[Mark, Dimitrian, Satish, Daniel]

2.1 Requirements

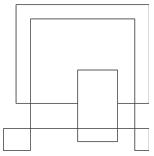
When elaborating the requirements, attention was paid to the SMART criteria. The requirements were then grouped by actors in descending order by importance using the MoSCoW method [Dysart, M., 2020] [Haughey, D.]

[Daniel, Dimitrian]

2.2 Actor descriptions

The actor descriptions will describe which of the features are available to each user, and how the functionality is divided between the actors.

Guest - Guest is an actor that uses the platform. At the start of using the platform every user is a guest.He has access to basic features as viewing the available recipes , he can also register and/or log in. After the guest registers or logs in, he becomes a user.



User - The user is an actor that can search for recipes, order the ingredients, see the order's history. He can view the current status of the order and edit his profile as well.

Administrator - The Administrator , later also referred as “admin” is an actor on the platform that can add and edit recipes and ingredients. He can also cancel the orders.

[Mark,Dimitrian,Satish,Daniel]

2.3 Functional Requirements

Guest requirements

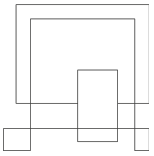
1. As a guest, I would like to register into the platform.
2. As a guest, I would like to log into the platform.
3. As a guest, I would like to search for recipes using search filters.

User requirements

1. As a user, I would like to order the ingredients for a recipe.
2. As a user, I would like to search for recipes using search filters.
3. As a user, I would like to view my order history.
4. As a user, I would like to view my order summary.
5. As a user, I would like to edit my profile.
6. As a user, I would like to view the current status of the order.
7. As a user, I would like to chat on the platform.

Administrator requirements

1. As an administrator, I would like to add recipes and ingredients.
2. As an administrator, I would like to edit recipes and ingredients.
3. As an administrator, I would like to cancel orders.
4. As an administrator, I would like to add new administrators.
5. As an administrator, I would like to reply to the messages in the chat system.



[Mark, Dimitrian, Satish, Daniel]

2.4 Non-Functional Requirements

1. The system should implement Three-Tier Architecture. Data should be stored in a DataBase.
2. The system has to use Java and C#

[Dimitrian, Daniel]

2.5 Use Cases

The figure 1 shows the use case diagram.

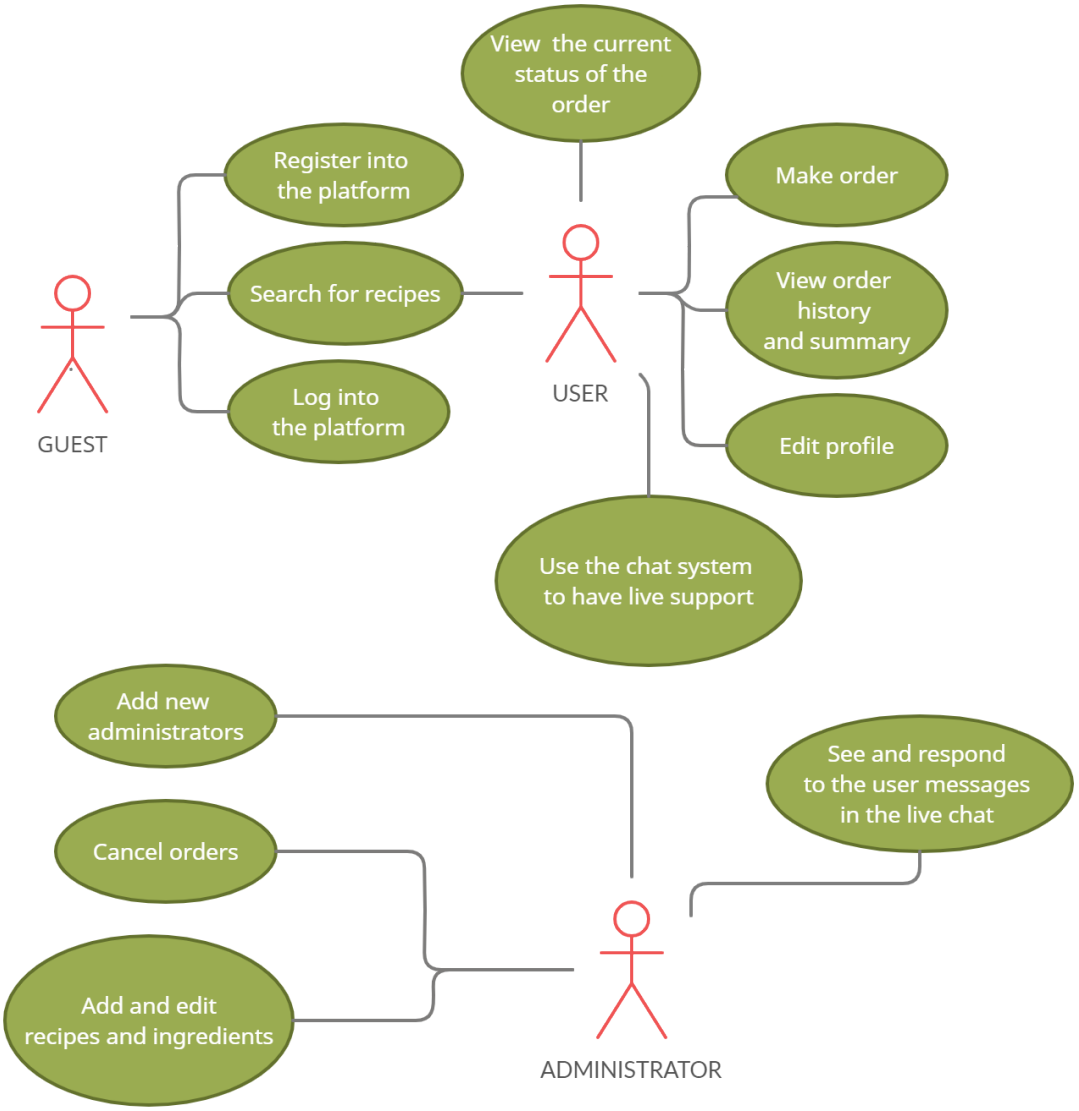
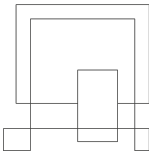


Figure 1. Use case diagram



A brief summary of the use cases can be read below.

Use cases summary:

1. Register into the platform

The guests can register on the platform. In order to register, the actor has to input email and password.

2. Search for recipes

The user and the guest can search for recipes on the platform. The search can be done by name, calories , allergens and other existing criteria.

3. Log into the platform

To log in, the user must input the email and the password he used to register on the platform.

4. Make order

A user has the possibility to make an order, using the buy option of a recipe, choosing the delivery options with the delivery prices, selecting his address and paying for the successful order.

5. View order history, status and summary

A user can view his order history in the event he wants to reorder something, but forgot what it was. He can also view an order`s summary, where he can find the status of the order, invoice amount, order date and other order related information such as shipment number.

6. Edit profile

A user has the possibility to edit his profile, specifically edit his email and update his password. To do so he must be logged in and know his current password.

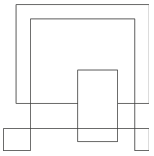
7. Cancel orders

System administrator is in power to cancel orders by the request of a user made by any means of communication (phone, email …).

8. Add and edit recipes

When there is a change in stock or a new recipe appears, the administrator can add information about the new recipe to the system, therefore keeping the inventory updated.

9. Cancel orders



Administrators can cancel an order if the user requests that. The order can also be changed if there is a problem with the supply. To cancel an order the administrator needs to know either the user`s email, or user`s id.

10. Add new Administrators

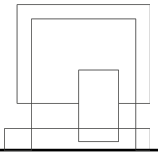
When one administrator is not enough, another one can be added. To do so, an existing administrator specifies the email and password of the new administrator. After that, the new administrator can log in.

11. Chat with Administrator

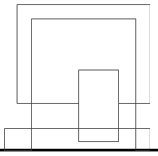
Users have the option to chat with the administrator. To do so they need to access the respective page , connect and ask a question. After that, an available administrator will have the option to start the live communication.

The use cases are further elaborated in the use case descriptions below. Only the use case descriptions for “Register a new user and login” and “Order the ingredients for a recipe” are provided.

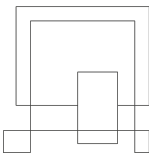
The interested reader can find the rest of the use case descriptions in Appendix B.



Use case	Register a new user and login
Summary	Register on the platform and become a user. The registered user is logged in to platform, Fulfills Requirement : Guest -1, Guest -2
Actor	Guest
Precondition	The actor has opened the application and is connected to the server.
Postcondition	The actor is logged in.
Base sequence	[S1] Register new user 1. The actor requests to sign up. 2. The system requests the actor to input required information (email, password, address, phone number) 3. The actor inputs the requested info and requests to sign up. [Ex. 3a] 4. The system has registered the actor as a user.
	Precondition: The actor is already registered [S2] Login 1. The actor requests to log in. 2. The system requests the actor to input email and password. 3. The actor inputs the login information. [Ex. 3a] 4. The system logs the actor in.
Exception sequence	[S1] 3a - Invalid information - The actor inputs invalid information or an already taken the login name. 1. The system shows an error message, specifying the problem. 2. [go to step 2 of base sequence]
	[S2] 3a - Incorrect login - The user inputs login information that does not match any information in the system. 1. The system shows an error message, specifying the problem. 2. [go to step 2 of base sequence]
Notes	● A valid email should follow the email pattern, in regex “/^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$/” ● A valid password is a password consisting of at least 5 characters and a maximum of 30 characters .



Use case	Order the ingredients for a recipe.
Summary	The user orders the ingredients for a wanted recipe. Fulfills Requirement : User 2.
Actor	User
Precondition	The actor is logged in and found the suitable recipe
Postcondition	System registers a new order.
Base sequence	[S1] Order the ingredients 1.Actor selected the buy option of a recipe. [EX 1a] 2.System shows available delivery options along with the delivery prices 3.Actor can select one of his addresses. [EX 3a] 4.System shows available payment methods. 5.Actor pays the order and sees order confirmation. [EX 5a] 6.System registers new orders with the status “Pending shipment”.
Exception sequence	[S1] 1a. Actor tries to pay for an existing order from the orders page. 1. System shows available payment methods. 2. Actor pays the order and sees order confirmation. (ex 5.a) 3a - Actor has no addresses- The actor has never ordered and therefore has no addresses registered into the system 1. The system shows input fields for Name, Address 1, Address 2, City, Postal Code. 2. The system validates the input fields and registers new addresses. 5a - Actor fails to pay / payment not processed . 1. System shows the warning message. 2. The system registers new order with the status : “Pending payment” 3. User is redirected to his orders page where he can try to pay it again.
Notes	● A valid Name can only consist of letters min length 4 max length 20. ● A valid Address 1, 2 can only consist of letters and numbers min length 4 max length 20. ● A valid City can only consist of letters min length 4 max length 20. ● A valid Postal Code consists only of numbers , length 4.



[Mark, Dimitrian, Daniel]

2.6 Domain Model

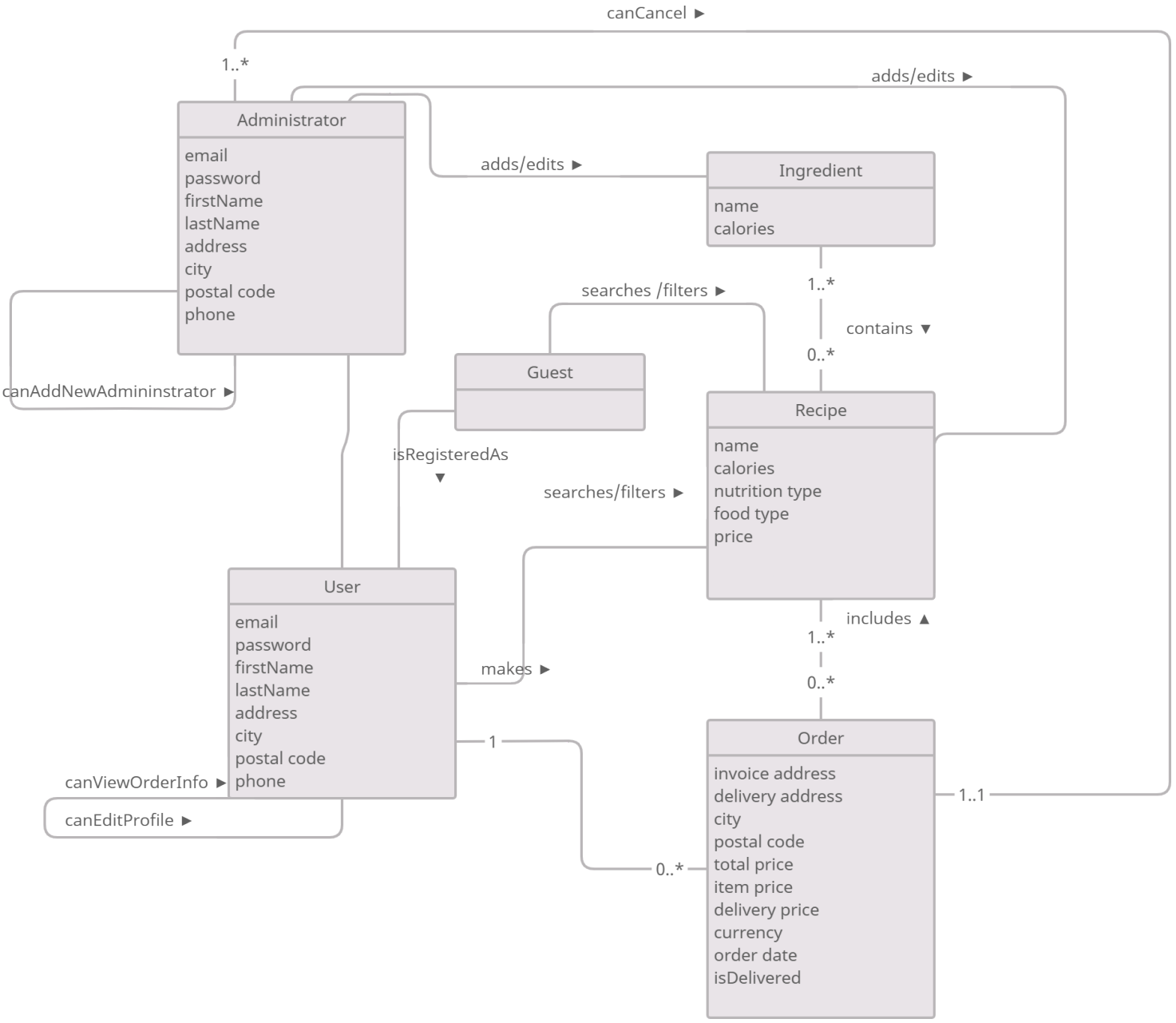


Figure 2. Domain model

In figure 2 the domain model can be seen, which presents all entities in the system and interactions between them. The model describes a total of six entities. Starting entity is the guest, it is the person that is new to the system, it can interact with the Recipe entity specifically, search for recipes. It can also become a user by registering or logging in. User is the main entity, it can also search for recipes but also order their delivery. To help out users, there is the Administrator entity. It can cancel orders, add new ingredients and recipes, but also disable the user profile in the event of misbehaviour.

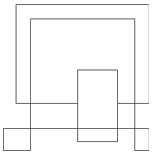
[Daniel, Satish]

Security

This subsection of the analysis will provide the reader with the threat model, state the objectives for security as well as discuss risk associated with those threats, and look into possible risk assessment models.

A threat model is basically knowledge about the adversary. All the mechanisms and motivation for the attackers trying to compromise the data. There are **three main objectives**: **Confidentiality**, this includes preventing the attackers from obtaining customer data, including passwords and profile information. **Integrity**, this includes preventing attackers from unauthorized modifying the application data, **Availability,** this includes providing the required services even while under an attack, and **Authorisation,** which ensures that only privileged accounts have access to certain functions.

In the table below you can see some of the possible attacks performed on a system, where Anything likely to cause damage to the system during an attack is counted as Threat.

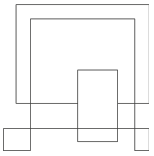


Starting with **STRIDE** and following the **Brute force, COA, KPA, CPA** and others such as “**Man in the middle"**, together with the Risk involved and possible prevention methods.

Threat	Property violation	Attacker means	Threat risk
Spoofing	Authenticity	active	**LOW**
Tampering	Integrity	active	**MEDIUM**
Repudiation	Non-repudiability	passive	**LOW**
Information disclosure	Confidentiality	active	**HIGH**
Denial of Service	Availability	active	**MEDIUM**
Elevation of Privilege	Authorization	active	**HIGH**
Man in the middle attack (kind of spoofing)	Authenticity	active	**LOW**
Brute Force	Confidentiality	active	**MEDIUM**
A chosen-plaintext attack (CPA)	Confidentiality	passive	**LOW**
A ciphertext-only attack (COA)	confidentiality	passive	**LOW**
A known-plaintext attack (KPA)	confidentiality	passive	**LOW**

Table 1: The threats associated together with Property violated, means of attack and Threat risks

The security concerts will be addressed again in the Design part. Together with the possible preventive measures.



3 Design

This chapter describes the architecture of the project via an architecture and package diagram. Choice of REST over SOAP as well as choice of C# over JAVA in the Business tier will also be discussed. Class diagram will be presented and a few classes related to the discussion above RegisterUser and OrderRecipe use cases will be presented.

3.1 Architecture

[Daniel, Dimitrian, Mark]

The architectural pattern used for project development was tiered architecture, used to organize the functionality of a given layer and place its functionality on the appropriate server. Functionally decomposition of the application was done into a three-tier architecture:

1. Presentation Tier (.NET Blazor)
2. Application/business logic tier (later referred as Business tier) (.NET)
3. Data tier (JAVA SpringBoot)

Tiers communicate with each other using RESTful APIs , it's also worth noting that in a three-tier application, all communication goes through the application tier. The presentation tier and the data tier cannot communicate directly with one another.

Below you can see the architecture and deployment diagram as well as tiers discussed in greeter detail.

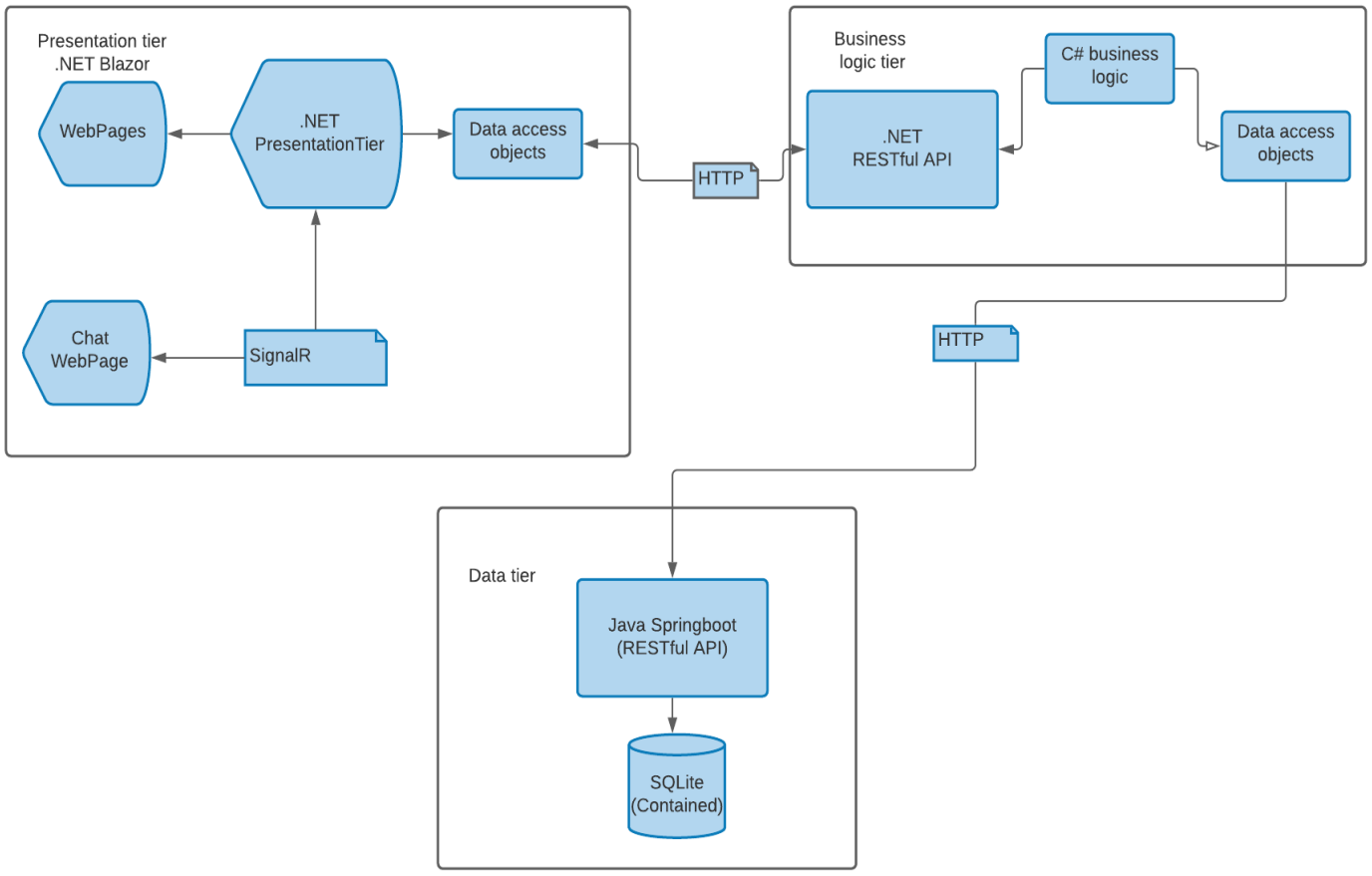


Figure 3. Architecture Diagram

- Presentation tier

The presentation tier is the user interface. This is what the user sees and interacts with. Its main role is to display information and collect information from the user. In this case it displays information about the services in a web browser as a web page.

The Presentation tier was developed using HTML, CSS, JavaScript and at the core .NET Blazor, a framework that allows the building of Interactive web UI with C#.

Below you can see the Starting page of the presentation tier that is available for an user after registration

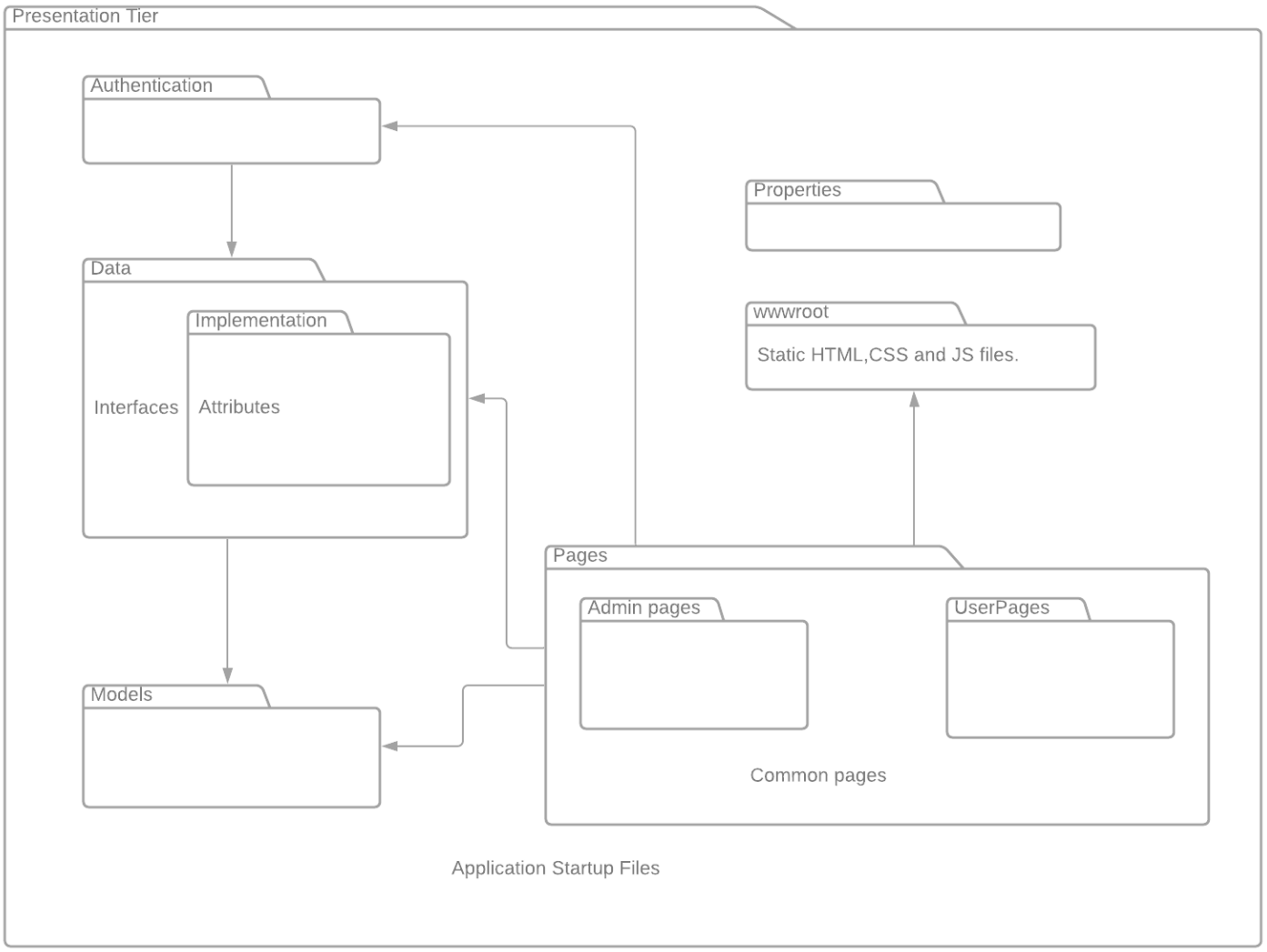


Figure 4. Presentation tier package diagram

- Business tier

The Business tier also known as the logic tier or middle tier is the main part of the application. The Business tier is the part that connects the Data tier to the Presentation tier. It is also the part where multiple web-services come together to be transformed and served to the final user. In this case, the data tier was written is C#, more about the technological choice later, and has the role to accept HTTP requests from the Presentation tier, do its magic, and write the data to the Data tier also using HTTP requests. The role of the Business tier

was decided by the architecture, had the choice between writing the business tier using C# .NET WebApi and JAVA SpringBoot.

The Business tier communicates with the data tier using API calls.

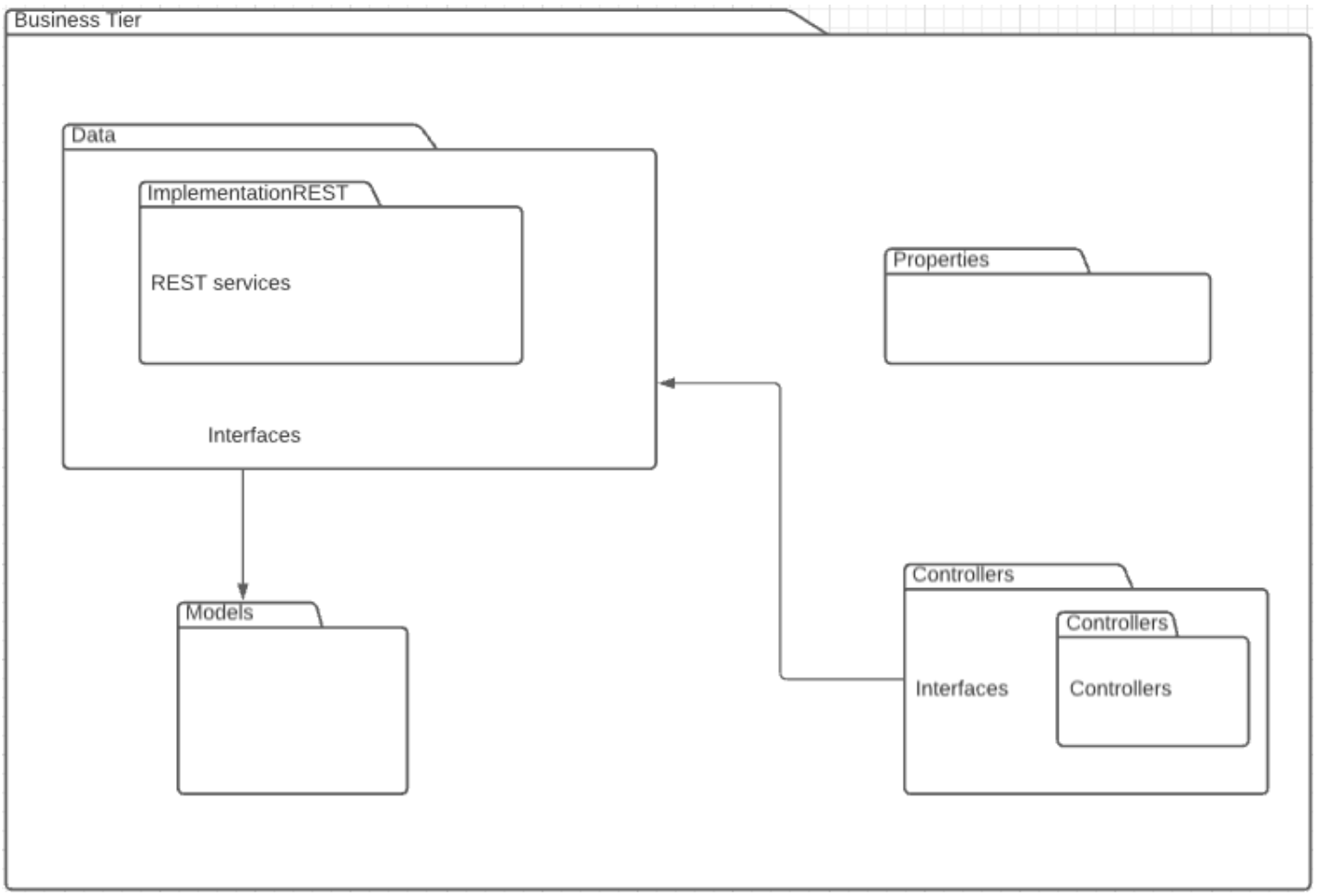


Figure 5. Business tier package diagram

- Data tier

Data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed. In this case Data tier was written using a JAVA Springboot ,which hosts endpoints for all CRUD operations with the models. Hibernate was used, which is an ORM (Object Relational Mapping) tool available in SpringFramework, it is a method of

abstraction over a database which provides the developer with an easy to access repository, just like it would be a simple list of items.

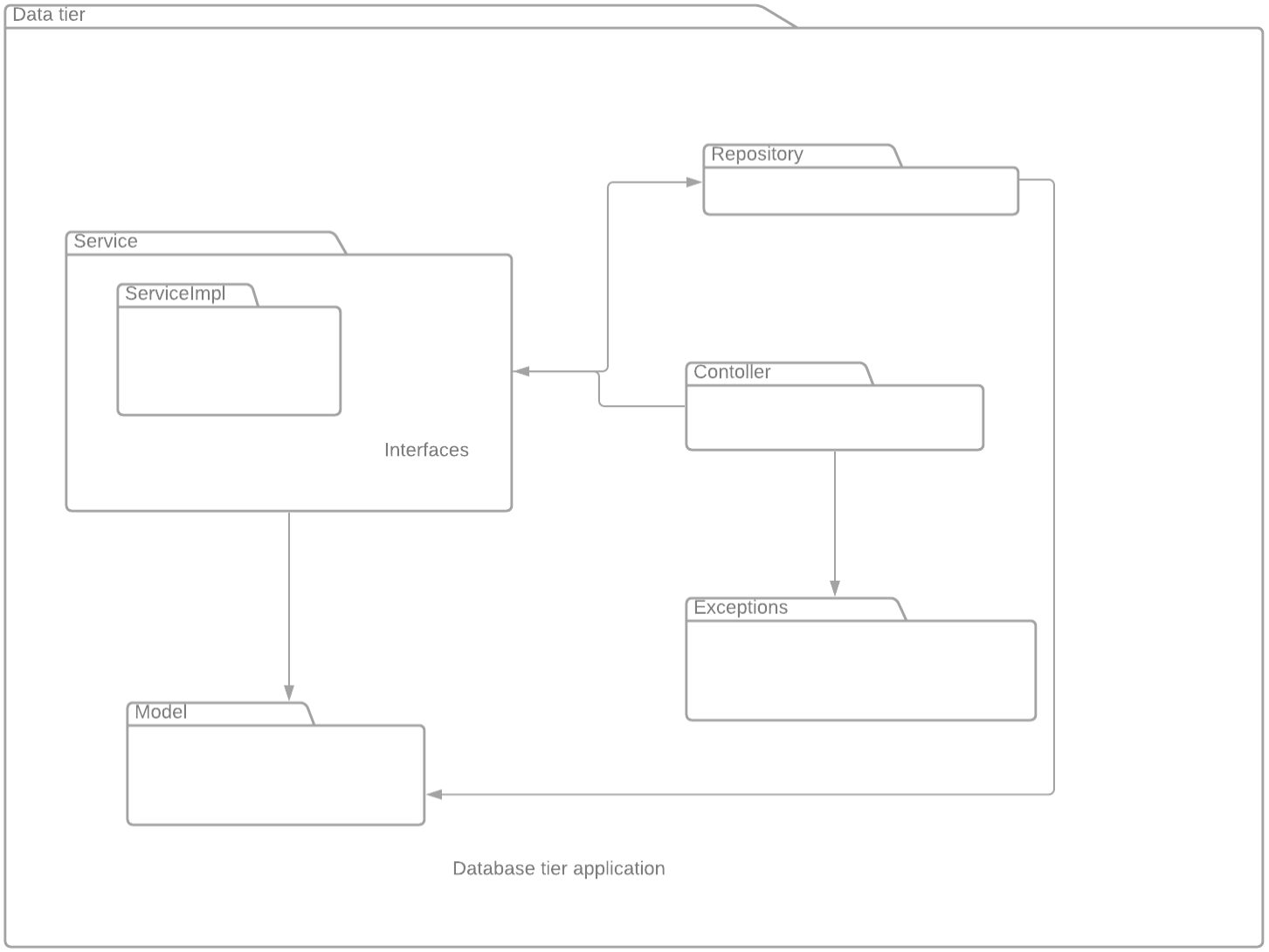


Figure 6. Data tier package diagram

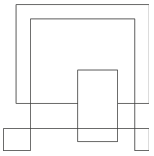


Figure 4, 5 and 6 shows the package diagram of the system. Looking closely the system actually follows three tier architecture as it is perfectly runnable on three different servers. And the communication between them occurs using messages. RESTful APIs were to handle the data transfer between layers. There is a “Models” package which contains wrapper classes used in the system.

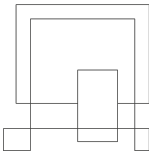
The Appendix C will contain the architecture and package diagrams and In the Appendix D there are class diagrams for all of the tiers.Together with all other diagrams used during development. The approach to designing the class diagram was to use SOLID (SOLID Principles in Java Application Development | JRebel by Perforce, 2021).

In order to achieve this, SOLID principles avoided the dependency of high level modules on the low level models by using abstractions (interfaces). The interfaces are open for extension and closed for modification. Which was experienced during development as endpoints for REST APIs were developed as there was a need for them. Also, there was a great effort for making classes have a single responsibility. This is why there is not a single controller for the entire application, nor a single data access object.

[Daniel]

3.2 Technologies

During the development, there was need for the communication between the tiers so by analysing the SEP3 course description where one of the requirements was the use of at least two networking technologies so a Web-Service was built and a chat system. The team had at least two options to choose from for each part of the networking.



For the Web-Service decision to choose between:

- SOAP based web service
- REST based web service

and for the real-time chat part two choices as well::

- WebSockets or
- SignalR

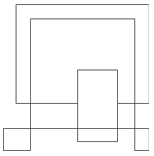
Each of the above mentioned technologies had their advantages and disadvantages.

SOAP relies exclusively on XML to provide its messaging services, with the main idea behind being that programs built on different platforms could exchange data in an easy manner. SOAP is easily extensible; this is why the XML used to send and receive messages can become extremely complex. SOAP also requires more bandwidth and the SOAP request and response messages are usually more heavy.

REST on the other hand provides a lighter-weight alternative, as instead of using XML to make a request, REST usually relies on a simple URL and unlike SOAP you are not forced into using XML. The fact that it is easier to use and more flexible, while also having a smaller learning curve, made us choose REST.

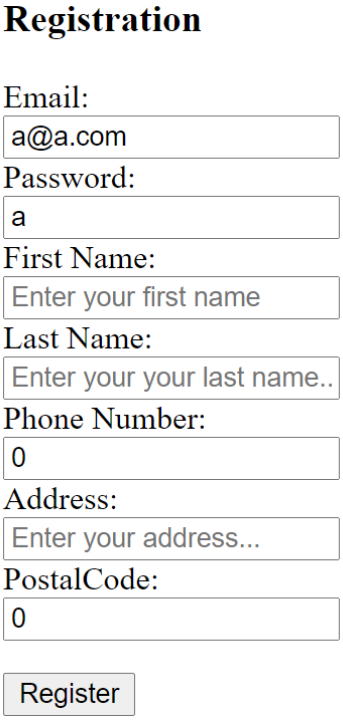
The decision between WebSockets and SignalR was pretty much straight forward as SignalR is basically an abstraction over WebSockets ,in the cases where WebSockets connection is possible, otherwise it will fall back to other protocols. The fact that SignalR is backwards compatible with older browsers and saves time while not forcing you into writing low level code, as in the case of WebSockets made us choose SignalR over WebSockets .

For the database management system, SQLite was chosen over others like PostgreSQL and MySQL. The decision was based on the fact that SQLite is self contained, the whole process of setting it up required only the import of its Java driver library. And while a special “Dialect” had to be written to be able to use Hibernate ORM.



[Daniel]

3.3 User interface Sketches



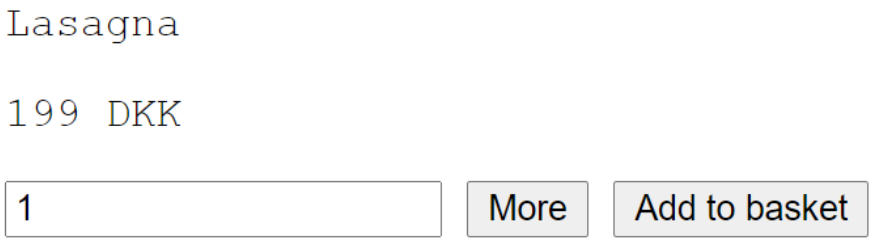
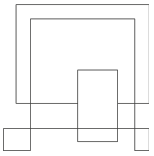


Figure 7 (left) and **Figure 8**, showing the Sketch of User registration process and the UI the user sees when adding an item to the basket.

User interface sketches were done in plain HTML, just to put everything in place, this was partly dictated by the frontend framework (Blazor) and its default styled pages.

The final design was not sketched and was done by styling the existing HTML ”bone”, with the help of icons and a couple of CSS frameworks.

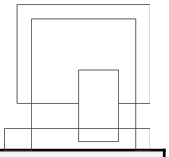


[Daniel, Satish]

3.4 Security

In the design chapter there was the threat model. This part of the design will talk about the possible security mechanisms that could be implemented in order to reduce the overall risk and prevent the attacks discussed earlier. Below you can see the threats presented in the Analysis part together with prevention measures.

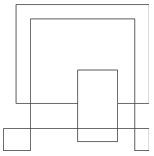
Threat	Property violation	Attacker means	Threat risk	Preventive measures
Spoofing	Authenticity	active	**LOW**	**Use of HTTPS instead of HTTP.** Using encryption(**symmetric or asymmetric**). Use of **VPN**
Tampering	Integrity	active	**MEDIUM**	Requiring a **digital signature** when trying to modify a publicly available information.
Repudiation	Non-repudiability	passive	**LOW**	**Enforce** client to **send his signature** along with HTTPS request
Information disclosure	Confidentiality	active	**HIGH**	Enforce encryption on server-side, **not storing any unencrypted data.**
Denial of Service	Availability	active	**MEDIUM**	Using a **DDoS mitigation** company (ex CloudFlare). Enforce a delay/limit for client requests



Elevation of Privilege	Authorization	active	**HIGH**	**Minimizing the number** of privileged accounts, **keeping a log** of these account activities, **regularly updating** the password.
Man in the middle attack (kind of spoofing)	Authenticity	active	**LOW**	Use of **SSL**, together with **HTTPS** protocol. **End to end** encryption (**asymmetric**)
Brute Force	Confidentiality	active	**MEDIUM**	**Enforcing a limit** for password trials, Banning by IP for some time when limit is exceeded
A chosen-plaintext attack (CPA)	Confidentiality	passive	**LOW**	**"Salting"** the user input before encryption the Make encryption mechanism unavailable for user
A ciphertext-only attack (COA)	confidentiality	passive	**LOW**	**"Salting"** the data before encryption. Increase encryption
A known-plaintext attack (KPA)	confidentiality	passive	**LOW**	Do not generate encryption on the client side. **Make** encryption mechanism **unclear**

Table 2: threat model together with prevention strategies.

There is not a perfectly secure system, because the world is not impeccable either, but at least awareness can be spread of the possible vulnerabilities and attempt to address them.



3.5 Class Diagram

[Mark, Dimitrian, Daniel]

To see the high-detail class diagram for each of the tiers you can go to Appendix C. In the following subsection, the path will be discussed that the data travels from the Presentation tier through Business tier to the Data tier, for the use case: Register User.

Each Tier has a user model, which similar to the class below, with some difference due to the naming conventions different in C# and Java.

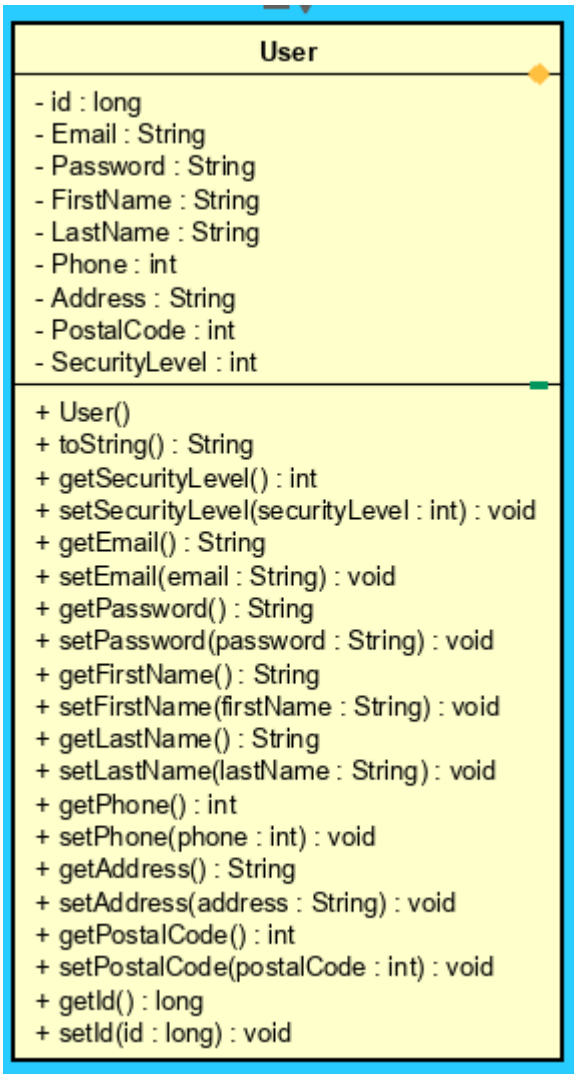


Figure 9. User Model from the Presentation tier

The presentation tier has a user model which it uses to validate the user inputs during the registration. After the registration, data travels to the user service data access object:

The UserModel automatically propagates changes to all interested listeners, making it easy for the GUI to stay in sync with the data.

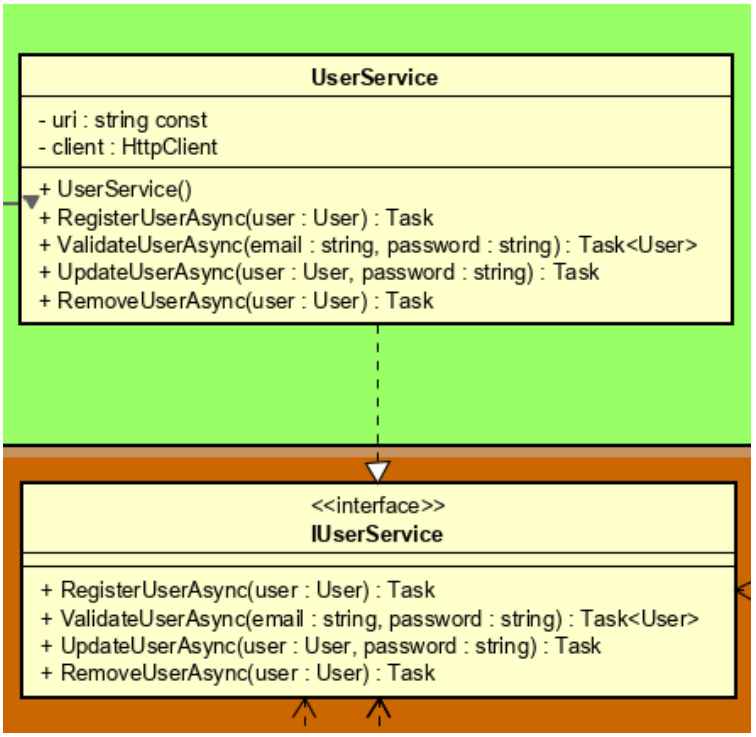


Figure 10. IUserService and UserService from the Presentation Tier

The user service serializes the Object into JSON and sends it to the Business tier REST API endpoint.

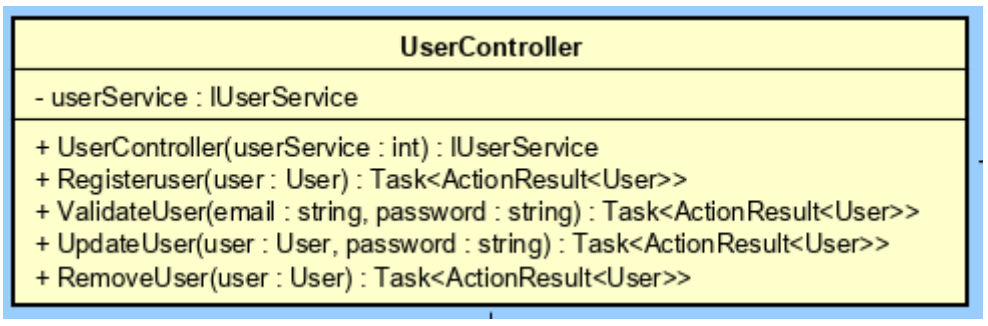


Figure 11. UserController from Business Tier

In the Business tier part. The JSON is Deserialized back to the User Model, the necessary fields are updated, and then sent to the Business tier user service which is another data access object.

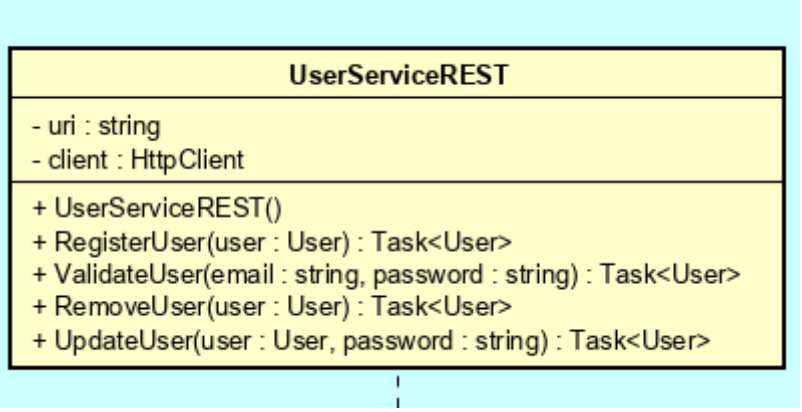


Figure 12. UserServiceRest from Business Tier

Business tier user service Serializes the now updated object into JSON and then sends to the Data tier REST API endpoint where the data is received, deserialized back to an Object and then sent to UserService in the data tier, where it is Saved using User Repository class, which extends the CRUD Repository class/interface. Which uses an ORM system to save the data to the SQLite database, using the “Dialect” provided.

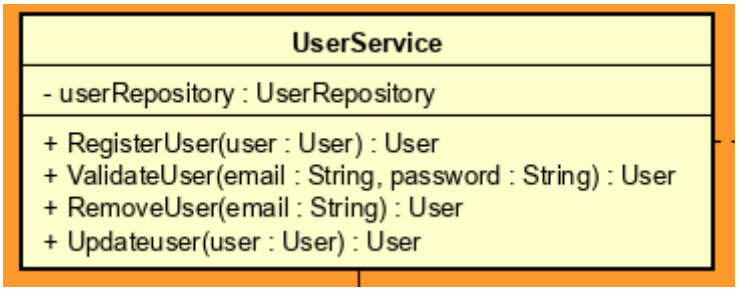


Figure 13. User Service class from the data tier.

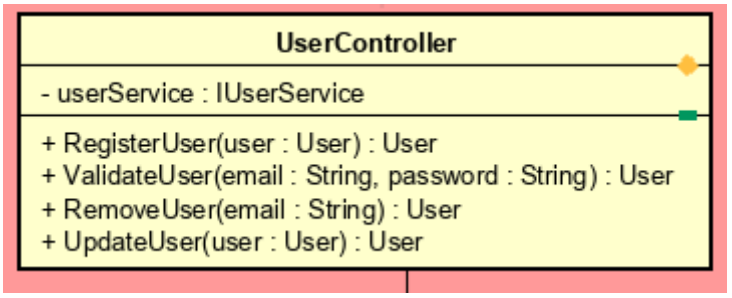


Figure 14. UserController from Data Access Tier

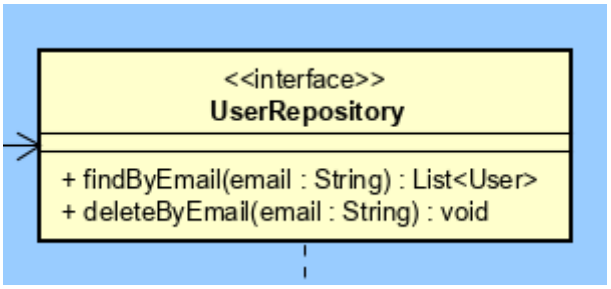


Figure 15. UserRepository interface from Data Access Tier

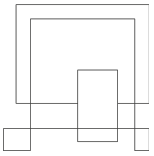
Physical database Model

As discussed earlier ORM was used to transform the Objects to relations. This is why a Physical database Model is not relevant for the application, but from the way ORM works, it should be assumed that it is similar to the domain Model.

4 Implementation

[Dimitrian, Daniel]

In this section, the libraries and frameworks will be discussed used to implement the design. And will once again look at the **Register A new user** use case. And follow the path from the ui to the Database.



4.1 Libraries & Frameworks used :

- **SpringBoot -** a java framework for building **APIs** together with necessary dependencies (more in the Data tier/Java/pom.xml)
- **Blazor** - a free and open-source web framework that enables developers to create web apps using C# and HTML
- **SQLite** - a database management system. SQLite allows you to have a database in the project, with no authentication required. The key feature of SQLite is that it is **self-contained.**
- **Bulma** - a CSS framework on the Flexbox layout. With Bulma, the extensive range of built-in features means faster turnaround and less CSS code writing.
- **Bootstrap** - another CSS framework used as default by **Blazor**.
- **SignalR** is a set of libraries that are small and focused on building real-time web applications as well as client applications cross-platform.
- **Swagger** is a language-agnostic specification for describing REST APIs. It allows both computers and humans to understand the capabilities of a REST API without direct access to the source code.

4.2 Design patterns used

- **Flyweight** design pattern is used by default when building application with Blazor.(Flyweight, 2021) The required classes are set up as services in the “startup” class and can be later “injected” into the necessary classes.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSignalR();
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddMudServices();
    services.AddScoped<IUserService, UserService>();
    services.AddScoped<IOrderService, OrderService>();
    services.AddScoped<IRecipeService, RecipeService>();
    services.AddScoped<IIngredientService, IngredientService>();
    services.AddScoped<IBasketService, BasketService>();
    services.AddSingleton<IChatService, ChatServiceInMemory>();
    services.AddBlazoredLocalStorage();
    services.AddBlazoredSessionStorage();
    services.AddBlazoredLocalStorage(config:LocalStorageOptions =>
```

Figure 16. The Flyweight pattern used by .NET

- **Singleton** is the design pattern that was used together with flyweight , it is a pattern that restricts the instantiation of a class to one "single" instance (Singleton, 2021), basically it ensures that all the classes requesting another class , are having the exact same object.

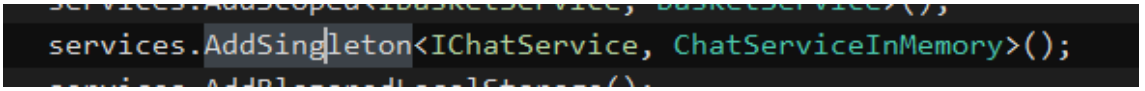


Figure 17. The singleton pattern used in the Startup Class in the presentation tier.

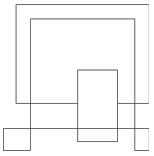
- **Repository pattern** The repository pattern is a strategy for abstracting data access. (Petrakovich, 2021) In this case **ORM** was utilized together with the Repository pattern for the data tier, in order to make the process of **CRUD** operations with data as painless as possible. Basically it allows the interaction with a database as if it would be a list object.

```
public interface UserRepository extends CrudRepository<User, Long> {
    List<User> findByEmail(String Email);
    @Transactional
    void deleteByEmail(String email);
}
```

Figure 18. The repository pattern used in the Data tier

4.3 Data travel from the Presentation to the Business tier

When the user inputs all the necessary fields required for registration, and clicks on “Register”, the “ AddNewUser” method is called



```
private void AddNewUser()
{
    UserService.RegisterUserAsync(newUserItem);
    NavigationManager.NavigateTo( uri: "/");
}
```

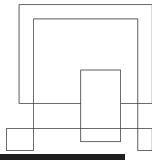
Figure 19. The method is called when pressing register button.

In the method call the use of the **"RegisterUserAsync"** can be seen, method of the **UserService** object. This method accept the User Model as an object, Serializes it and then uses the **HTTP POST method** to send it to the Business tier endpoint

```
public async Task RegisterUserAsync(User user)
{
    var userAsJson string = JsonSerializer.Serialize(user,serializeOptions);
    HttpContent content = new StringContent(userAsJson,
        Encoding.UTF8,
        mediaType: "application/json");
    await client.PostAsync(uri, content);
}
```

Figure 20. The RegisterUserAsync method in the UserService class from Presentation tier

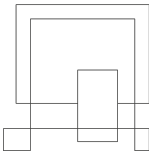
In the business tier User controller the presence of **HTTP POST** endpoint used earlier can be seen, which accepts the User Model object from the request body, deserializes it and then uses the **User Service RegisterUser** method.



```
[HttpPost]
Daniel Railean
public async Task<ActionResult<User>> RegisterUser([FromBody] User user)
{
    try
    {
        User returned = await userService.RegisterUser(user);
        return Ok(returned);
    }
    catch (Exception e)
    {
        return StatusCode(500, e.Message);
    }
}
```

Figure 21. The RegisterUser method in the UserController from Business Tier

After the RegisterUser method call is called , the user security level is set to 1, thus providing some Authorisation control, then serialized and sent to the **"/RegisterUser"** endpoint of the Data tier. The result was awaited and then returned back.



```
0+1 usages  dimitrian +2
public async Task<User> RegisterUser(User user)
{
    user.SecurityLevel = 1;
    string UserAsJson = JsonSerializer.Serialize(user);
    Console.WriteLine(UserAsJson);
    HttpContent content = new StringContent(
        UserAsJson,
        Encoding.UTF8,
        mediaType: "application/json");
    HttpResponseMessage response =  await client.PostAsync(requestUri: uri+"/RegisterUser", content);
    if (!response.IsSuccessStatusCode)
    {
        APIError apiError = JsonSerializer.Deserialize<APIError>(await response.Content.ReadAsStringAsync());
        throw new Exception(message: $@"Error: {apiError.message}");
    }
    string result = await response.Content.ReadAsStringAsync();
    User userReceived = JsonSerializer.Deserialize<User>(result, new JsonSerializerOptions{ PropertyNamingPolicy
        = JsonNamingPolicy.CamelCase});
    return userReceived;
}
```

Figure 22.. RegisterUser Method in the User Service from the Business tier

The User controller has the above mentioned RegisterUser endpoint, which accepts the User in the Request Body then uses the RegisterUser endpoint of the UserService, and throws a new error if unsuccessful.

```
@PostMapping("/RegisterUser")
public User RegisterUser(@RequestBody User user){
    try {
        User returnUser = userService.RegisterUser(user);
        System.out.println(returnUser);
        return returnUser;
    }catch (Exception e){
        throw new UserExistsException();
    }
}
```

Figure 23. Register User endpoint from the Data tier

The RegisterUser method tries to find a user with the exact same email as the one trying to register. If it does it throws a new Exception specifying the problem, if a user with such an email was not found, then it uses the userRepository object to save the user.

```
@Override
public User RegisterUser(@Validated User user) throws Exception {
    List<User> users = userRepository.findByEmail(user.getEmail());
    if(users.size()>0){throw new Exception("User with this email already exists");}
    return userRepository.save(user);
}
```

Figure 24. RegisterUser method from the Data tier user Service

The user repository is just an interface that extends the Crud repository which allows the ORM mapping and the abstraction over the SQL tables. Here the data path ends, and it is safely saved in the SQLite database

```
public interface UserRepository extends CrudRepository<User, Long> {
    List<User> findByEmail(String Email);
    @Transactional
    void deleteByEmail(String email);
}
```

Figure 25. User repository interface from the data tier

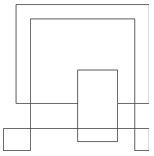
The hibernate reads the application.properties file to know the database access details and the used Dialect, which was kindly implemented to be able to use SQLite, which is not supported by default.

Figure 26.

[Dimitrian]

Test

There are different methods that can be used for software testing. The one that was used in detail below is a Black Box Integration testing, utilizing test cases derived from the use case descriptions. While performing a black-box test, the interaction is made with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.



To avoid confusion, it may be worth pointing out that the sunny scenario test cases are marked with green color and the exception test cases are marked with red color.

[Dimitrian]

5.1 Test Specifications

Below, the test cases and exception test cases for the **Register a new user and LogIn** and **Order the ingredients for a recipe** use cases can be found. The interested reader is referred to Appendix H, which contains all of the test cases specification.

5.1.1 Test cases

Table 3: Test case: **Register a new user** - Passed.

Step	Action	Reaction
1.	The actor prompts the system to register new user	The system prompts the actor to input his email, password, first name, last name, phone number, address and the postal code.
2.	The actor inputs the information, severally: “ Thomas@gmail.com ” “ ********** ” “ Tom ” “ Gandon ” 45347655 “ Silkeborgvej 15 “ 8700 and clicks on the Register button.	The system successfully creates a new user

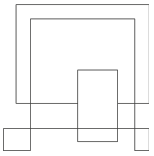


Table 4: Test case: **Register a new user** - Invalid email address- Passed.

Step	Action	Reaction
1.	The actor prompts the system to register new user	The system prompts the actor to input his email, password, first name, last name, phone number, address and the postal code.
2.	The actor inputs the information, severally: “ 33333333 ” “ ********** ” “ Tom ” “ Gandon ” 45347655 “ Silkeborgvej 15 “ 8700 and clicks on the Register button.	The system shows an error. “ The Email field is not a valid e-mail address. ”

Table 5: Test case: **LogIn** - Passed.

Step	Action	Reaction
1.	The actor prompts the system to LogIn.	The system prompts the actor to input his email and password.
2.	The actor inputs the information, severally: “ Thomas@gmail.com ” “ ********** ” and clicks on the LogIn button.	The system successfully log in the user

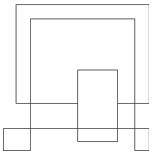


Table 6: Exception Test case: **LogIn** - No password filled - Passed.

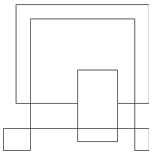
Step	Action	Reaction
1.	The actor prompts the system to LogIn.	The system prompts the actor to input his email and password.
2.	The actor inputs the email: “ Thomas@gmail.com ”,but not the password and clicks on the LogIn button.	The system shows an error. “ Enter Password ”

Table 7: Test case: **Order the ingredients for a recipe** - Passed.

Step	Action	Reaction
1.	The actor selects the amount of recipes and clicks on the Add to Basket button	The system saves the amount of recipes in the User Basket.

Table 8: Test case: **Order the ingredients for a recipe** - Invalid quantity- Passed.

Step	Action	Reaction
1.	The actor inputs a negative amount of recipes and clicks on the Add to Basket button	The system shows an error. “ Please select at least one piece to add to the cart.”



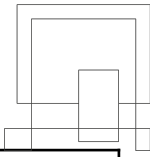
[Daniel]

Results and Discussion

The reader is welcome to try the application out. The source code for all of the three tiers can be found in Appendix E along with the installation guide. Instructions for use are described in the User guide in the Appendix F. In the table below the reader can find which of the use-cases were implemented and whether the functionality works as intended. Below the table discussed the reasons for why some of the user stories were not implemented or do not work as intended.

6.1 Implementation status of requirements

Use Case	Implementation	Functionality
As a guest, I would like to register into the platform.	Implemented	Fully functional
As a guest, I would like to log into the platform.	Implemented	Fully functional
As a guest, I would like to search for recipes using search filters.	Implemented	Functional partially (no search filters)
As a user, I would like to order the ingredients for a recipe.	Implemented	Fully functional
As a user, I would like to search for recipes using search filters.	Implemented	Functional partially (no search filters)
As a user, I would like to view my order history.	Implemented	Fully functional
As a user, I would like to view my order summary.	Implemented	Fully functional
As a user, I would like to edit my profile.	Implemented	Fully functional
As a user, I would like to view the current status of the order.	Implemented	Fully functional
As a user, I would like to chat on the platform.	Implemented	Fully functional



As an administrator, I would like to add recipes and ingredients.	Implemented	Not functional, Presentation tier problem
As an administrator, I would like to edit recipes and ingredients.	Implemented	Not functional, Presentation tier problem
As an administrator, I would like to cancel orders.	Implemented	Not functional, Presentation tier problem
As an administrator, I would like to add new administrators.	Not implemented	Not implemented
As an administrator, I would like to reply to the messages in the chat system.	Implemented	Fully functional

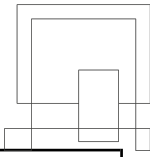
Table 3: Implementation status of requirements

The non functional requirements have been fully satisfied as the system adheres to three tier architecture, Uses C# for the Presentation and Business tier, use Java for the Data tier and have a database.

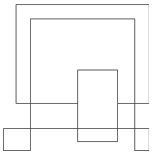
6.2 Discussion

All the basic functional requirements were implemented successfully, except for the CRUD operations on orders recipes and ingredients, which broke during the last scrum sprint due to a model change, although some of the features are not fully implemented, the system fulfills its main purpose and is usable, as the new Orders can be added by sending the REST request body through a message broker such as Insomnia or Postman. And while this is a shame of a solution, the time for fixing the presentation tier was not enough.

From the security standpoint, the system is not yet ready to be used, as it has some critical security flaws, which set the system unusable until they are to be fixed. Some of the major flaws are presented in the table below:



Nr	Vulnerability	Possible Security Objective Violated	Threat risk	Possible fix
1	As is , the system uses HTTP instead of HTTPS	Integrity, Confidentiality Authenticity	HIGH	Signing SSL certificate and sending them for validation / The use of a cloud service provider which usually provides SSL certificates
2	System saves the data on client side in an unencrypted manner	Confidentiality Authenticity	HIGH	Saving the user information in device memory and using JWT for authentication and reauthentication.
3	Current user is stored in LocalStorage of the browser in the form of JSON (unencrypted)	Confidentiality	HIGH	Saving the user information in device memory and using JWT for authentication and reauthentication.
4	Passwords sent and are saved as plain string in the data tier	Availability	HIGH	salting and encrypting the passwords before saving
5	System has no protection against brute forcing a password	Confidentiality	MEDIUM	Enforcing a exponentially increasing delay between unsuccessfully password retries
6	System has no protection against DDoS attack	Availability	LOW	Enforcing a limit on user requests after which impose a delay on each request.



[Daniel, Mark]

Conclusions

It might be difficult for people to start cooking, and make this process easier, by providing a platform able to deliver fresh ingredients for healthy products right to the customer.

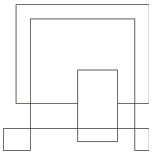
In the analysis chapter the actor descriptions described the way users interact with the system following the SMART principles and MoSCoW method. Use case diagram was presented along with two use case descriptions, which are further inspected in the design and implementation chapters. Also discussed the security threats involved in developing a system by presenting the threat model, along with the risk caused by these threats.

In the design chapter, the three-tier architecture diagram was presented, along with packages for each of the tiers. Technology choices were explained and the class path that data goes through to reach the Data tier from the Presentation tier during the process of user registration. Also looked once again at the threat model and tried to look at possible preventive measures.

In the implementation Part frameworks and libraries used to develop the project were discussed and example code snippets relating to the User registration use case. Again looked at the data path from the Presentation tier to the data tier, along with a concrete example of data transmitted.

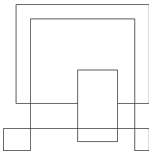
The test chapter described the approach for testing and test specification. Tested each of the tiers for the process of user registration.

In the result and discussion chapter, the implementation status for user stories was stated, along with the requirement fulfillment and existing vulnerabilities.



To conclude, the core functionality was successfully implemented and the project requirements were fulfilled, if not accounting for security. If security was accounted for then the system can barely be usable. But the improvement points were clearly stated.

If the project was started from the beginning once again there would be a different approach on the architecture, as for now the business tier is more like an adapter between the presentation tier and data tier, the security features implementation would be taken into a higher priority.



[Daniel, Mark]

Sources of information

Dysart, M., 2020. Understand SMART Project Objectives & Learn How To Write Them For Your Projects (With Examples). [online] The Digital Project Manager. Available at: <https://thedigitalprojectmanager.com/project-objectives/> [Accessed June 3, 2021].

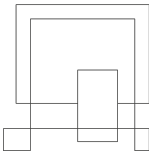
Haughey, D. MoSCoW Method. [online] Project Smart. Available at: <https://www.projectsmart.co.uk/moscow-method.php> [Accessed June 3, 2021].

Refactoring.guru. 2021. *Flyweight*. [online] Available at: <https://refactoring.guru/design-patterns/flyweight> [Accessed 3 June 2021].

Refactoring.guru. 2021. *Singleton*. [online] Available at: <https://refactoring.guru/design-patterns/singleton> [Accessed 3 June 2021].

JRebel by Perforce. 2021. *SOLID Principles in Java Application Development | JRebel by Perforce*. [online] Available at: <https://www.jrebel.com/blog/solid-principles-in-java> [Accessed 3 June 2021].

Petrakovich, J., 2021. *The WHY Series: Why should you use the repository pattern?*. [online] Making Loops. Available at: <https://makingloops.com/why-should-you-use-the-repository-pattern/#:~:text=The%20repository%20pattern%20is%20a,list%20items%20in%20a%20table.> [Accessed 3 June 2021].



Appendices

[Dimitrian, Mark]

Appendix A Project Description

Appendix B Use case descriptions

Appendix C Architecture - GodEats diagrams

a. DataAccess Tier package diagram
b. Business Tier package diagram
c. Presentation Tier package diagram
d. Architecture GodEats diagram

Appendix D GodEats diagrams

a. DataAccess Tier class diagram
b. Business Tier class diagram
c. Presentation Tier class diagram
d. Domain Model
e. Use case diagram
f. Sequence diagram
g. Activity diagram

Appendix E Source code , executables and installation guide

Appendix F User guide

Appendix G Source code documentation

Appendix H Test specification