## Technologies

This chapter is dedicated to the technologies used during this project and explains in a higher detail the purpose and the reasons why these tools were selected to achieve a specific properties and qualities of the product.

### JavaScript

JavaScript was selected as a main programing language for this project as it fits the needs of the single page web applications very well and is supported by most of the modern browsers. It is a high level, multi-paradigm, programming and scripting language suitable for object-oriented programming as well as functional programming. One of its highlights is the ability to dynamically update the content of the webpage without the need to refresh, embracing the interactivity it adds to the web pages. Another benefit is the full integration with the HTML and CSS giving the possibility to change and manipulate the HTML elements as well as to change their styling.

### AngularJS

In simplicity AngularJS is a JavaScript framework also written in JavaScript. Possibility to extend HTML attributes with AngularJS directives is one of the core functionalities among the others. Moreover, it offers option to create customize directives as well. These directives are used to create components which introduces another level of code reusability together with option to control behavior and complex DOM structures. Another advantage is the way AngularJS handles two-way data binding easily with directives or expressions. Furthermore, it implements the MVC pattern in HTML for the view and in JavaScript for the model with controller. This framework was selected to make a noticeable progress in shorter time, provide better code organization and readability.

Jasmine

Jasmine is a BDD (behavior-driven development) framework for JavaScript that was chosen as a tool for creating the unit tests in the client-side of the application. Jasmine is open-source and due to no external dependencies also adds value by its low overhead. Plus its build-in and optionally custom matchers make it relatively simple to implement a unit test in a short period of time.

### HTML

HTML also known as Hyper Text Markup Language is a standard markup language which was used to build a skeleton of the front-end part of this project. It structures and defines all elements on the website. This is done with the use of the so-called tags which are specific for each element type on the HTML page (e.g. images, links, divisions, paragraphs etc.).

### CSS and SCSS

To achieve a higher code organization and readability the SCSS was chosen instead of the classical CSS for the project. SCSS stands for Sassy CSS and is similarly like SASS (Syntactically Awesome Style Sheets) an extension of CSS. One of the main advantages of SCSS is a possibility to create variables which adds flexibility and more importantly maintainability to the code. Another advantage is a support of nested syntax which can be used to reflect the HTML tag hierarchy and provide a more natural structure of the code. And least not last is the option to divide the code into files separated by business logic which adds even better code structure. The important functionality is that these files are afterwards merged during the compilation into a single CSS file. This results into a single HTTP request to obtain all the styles from the server.

### Java

Java is object oriented, class-based, general-purpose and statically-typed programing language which is not compiled directly into the machine code but instead into the so-called byte code. This byte code is further processed by JVM (Java Virtual Machine) introducing platform independency. Java was used for the backend of the project and together with Spring Boot provide a solution to handle all the request from the client side (e.g. image storage in the file system, accessing the database etc.).

### Spring Boot

Spring Boot is a Java based, open source framework picked for this project since it reduces the development process of web applications and testing significantly. It is used together with Java to create RESTful web service which is the core of the project architecture. It also provides an embedded web server (Tomcat) by default. Even though this server is pre-configured we can change the configuration settings to fulfil desired behavior.

### Apache Tomcat

Apache Tomcat is an opensource Java server container used as a web server or

for hosting Java servlets.

### IntelliJ IDEA

IntelliJ IDEA is an integrated development environment. It was chosen for this project since it provides a powerful tools and support during the process of software development of a single page web application as well as its back-end side Spring Boot server. Essentially error checking, syntax highlighting and autocompletion contribute to accelerating the overall performance. Remote debugger and integration of the version control system speeds up the process tremendously.

### GitHub

GitHub is primarily a hosting service for repository of version control system Git. This tool’s main purpose is to ease the collaboration between the team members and any other contributors to the project. With the same importance it provides a system to control the versions of the system as well as track the progress and check the changes done. Among its leading functionalities is opportunity to make so called pull request which adds extra layer of quality assurance.

### MySQL and SQL

SQL (Structured Query Language) is a standard language designed for data manipulation. These data are stored in objects called relations or tables which consist of tuples / rows and attributes / columns. MySQL is an SQL relational database management system (RDBMS). Security and high reliability are one of the reasons why it became globally popular and created a massive community that a developer can benefit from speeding up the whole process. MySQL is opensource and in this project it serves as a web database benefiting from its high performance and good scalability.

Abstract

*As the research in the field of healthy lifestyle makes progress, people care about what they eat more and more. This results in the need for a system that would be a solution for handling related task. The aim of this project is to build a recipe management system which would be a cross platform solution that would eliminate the need for many different applications to satisfy the arising needs.*

*The solution is a system that consists of a single page web application and a RESTful server implemented in Spring Boot that handles the database and file system interactions. The recipe image compression is also handled by the server. The client side is implemented in AngularJS and its main functionalities are creating, deleting and updating recipes. Adding ingredients to a shopping list and searching through the recipes by applying filters as well as possibility to share the recipes with friends. Due to the responsive design of the system the application is available on multiple screen resolutions.*

*All requirements have been accomplished on time and tested to assure the proper behavior of the system.*

### Introduction

This document contains a detailed information about the steps taken during the process of development of the Food and Recipe assistant which is an advanced management system for recipes. The first four chapters are dedicated to the four stages of the development which are analysis, design, implementation and testing providing an insight to the decisions made during them. Analysis points out the actions behind the extraction and formulation of the software requirements specifications. Next chapter dives into the planning of the system design including the design of algorithms as well as architectural design. This chapter is followed by implementation describing how the designed functionalities were implemented from the perspective of writing the source code providing more information about what techniques and patterns were used and the reasoning behind them. This chapter also includes a section about the technologies used during the implementation phase and the considerations that led to these choices. Last of these four stages is testing. The chapter about the testing detailly describes proceeding towards the high quality and reliability of the product with respect to the testing models and methods selected to ensure these attributes. Chapter results and discussion states the outcome and goals achieved during these stages.

Finally, this report states the conclusion and discusses the future of the project.

### **Analysis**

The issues discussed in the Project description - **Appendix** were identified and broke down into the smaller tasks called requirements to gradually proceed towards the solution. These were retrieved from the customer to fulfil their expectations and needs throughout the process called requirements engineering since the main aim is to deliver a high quality software that satisfies the customer’s needs.

#### **Requirements**

This section contains the lists of requirements including both functional and nonfunctional. The functional requirements were prioritized after considering which ones are the core requirements and which ones are the most important. This was also done to ensure the correct flow of the development process as some requirements might be dependent on the others e.g. adding a user as a friend is a precondition of sharing a recipe with another user thus it has a higher priority and shall be implemented first.

**Functional Requirements:**

* FR-1.1: The user shall be able to create a new user account by filling in the required information and submitting it.
* FR-1.2: The user shall be able to log in to the application.
* FR-2.1: The user should be able to add a new recipe by filling in the required information and submitting.
* FR-2.2: New recipe should appear in the list of recipes immediately without the need to refresh the page.
* FR-2.3: When the user selects the recipe from the list, more details should be displayed.
* FR-3.1: The user shall be able to update an existing recipe by providing new values.
* FR-4.1: The user shall be able to remove an existing recipe from his list of recipes.
* FR-4.2: If the user shared this recipe with some of the other users - friend, it shall not be deleted from their lists.
* FR-5.1: The user shall be able to add other users as „friends“ by email address.
* FR-5.2: The user should be able to confirm a friend request.
* FR-5.3: The user should be able to reject a friend request.
* FR-5.4: If the fried request is confirmed, users should be able to mutually see each other in their „Friend lists “.
* FR-6.1: The user shall be able to send a recipe to other users from his „ Friend list “.
* FR-7.1: The user shall be able to search in the list of recipes by the recipe name.
* FR-7.2: The user shall be able to search in the list of recipes by the ingredient.
* FR-7.3: The user shall be able to search in the list of recipes by the category.
* FR-8.1: The user shall be able to get a recipe suggestion.
* FR-9.1: The user shall be able to add items to the shopping list directly from the list of ingredients of a recipe.
* FR-9.1: The user shall be able to remove items from the shopping list.
* FR-10.1: The user shall be able to receive notifications about friend requests.
* FR-10.2: The user shall be able to receive a notification about recipes received from friends.
* FR-11: Manage user profile

**Non-functional Requirements:**

Even though these requirements do not have a functional character they are still of a high importance as they ensure certain qualities of the system. They dictate the system attributes that need to be achieved by non-functional approach.

* **NFR-1:** The application should be independent of the operating system of the device.
* **NFR-2:** Client should have a responsive character.
* **NFR-3:** The response time of the system should be less than 3 seconds 90% of the time.
* **NFR-4:** The time to learn and operate the system shall be below 15 minutes.
* **NFR-5:** Menu of the system should be designed for easy use on touch screens.
* **NFR-6:** System language shall be English as it is most widely spoken world language.

**User Stories:**

The user stories were created in order to fulfil the requirements specified above and they reflect the interaction of the user with the system. Since there is need only for one type of the user, there are no restrictions in the system and all functionalities are accessible to all the users. All user stories are displayed in the use case diagram of the system in Figure 1. It pictures all allowed actions that the user can perform as well as the relationship between them.

1. As a user I want to be able to create a profile.
2. As a user I want to log in.
3. As a user I want to update the profile, so it contains an actual information.
4. As a user I want to add a recipe so I can access it whenever needed.
5. As a user I want to manage recipes so I can adjust them or remove them depending on my needs.
6. As a user I want to search through the recipes, so I do not waste time on scrolling through them manually.
7. As a user I want to add another user as a friend to keep track of the users I am interest in sharing recipes with.
8. As a user I want to share a recipe with other users, so the recipes do not need to be typed in manually.
9. As a user I want to manage received recipes, so I can decide whether to keep them or not.
10. As a user I want to get recipe suggestions, so it is easier to make a decision about what to cook.
11. As a user I want to receive notifications, so I am informed about the incoming recipes and friend requests.

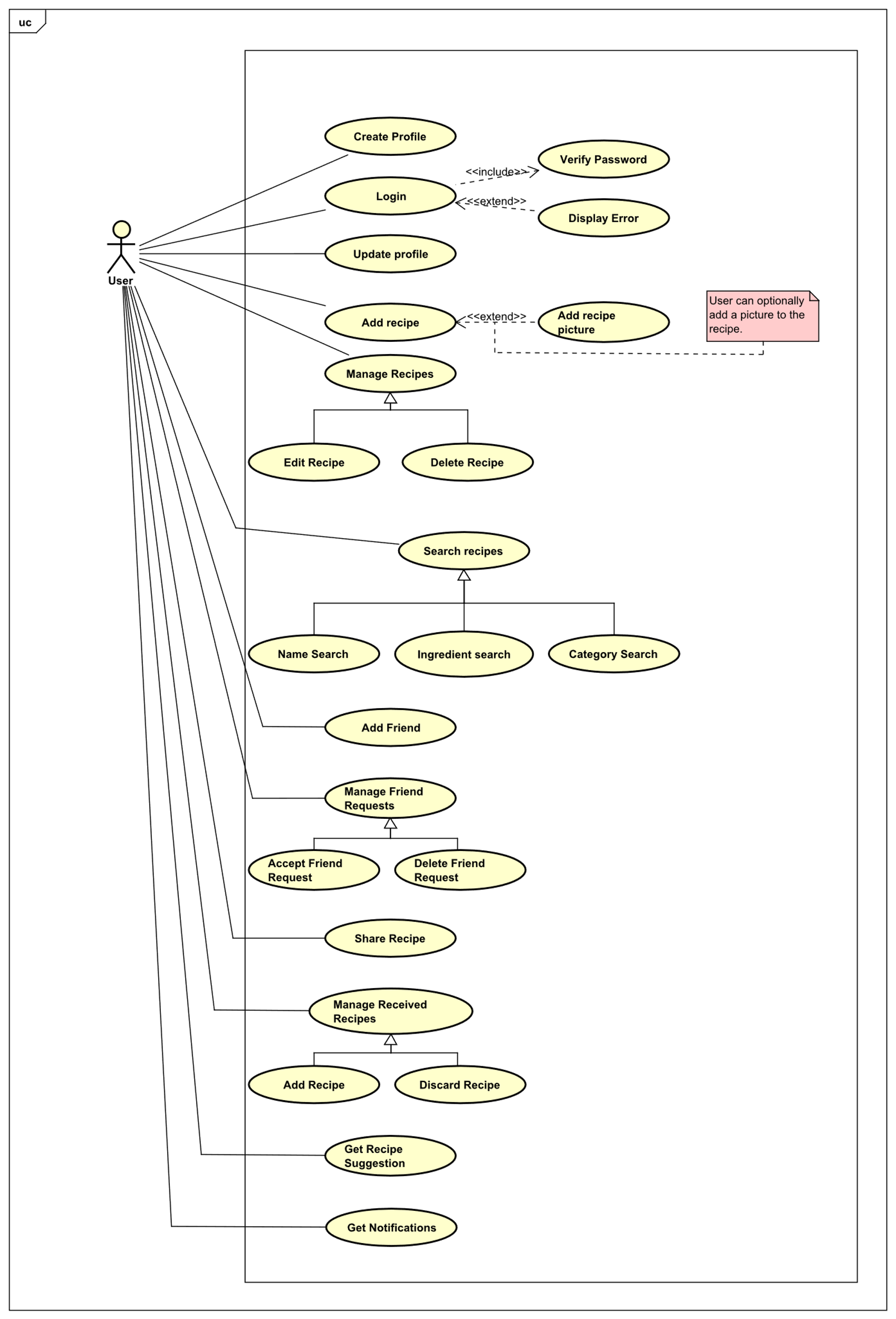


Figure 1 Use case Diagram

#### Use cases

The more detailed information about each of the use cases are present in the use case descriptions. These providea closer look at the steps behind each actions as well as the conditions to finish the action sucessfuly. Preconditions are the conditions that need to be fulfilled otherwise the action can not becompleted. Postconditions are the results of successfully proceedint through the steps in base sequence which lists the smallest units that the action consists of. When these steps are followed and the preconditions are met, the user will achieve the expected result. If the preconditions are not met, the exception will arise and the steps to solve the issue are listed in the exceptions sequence.

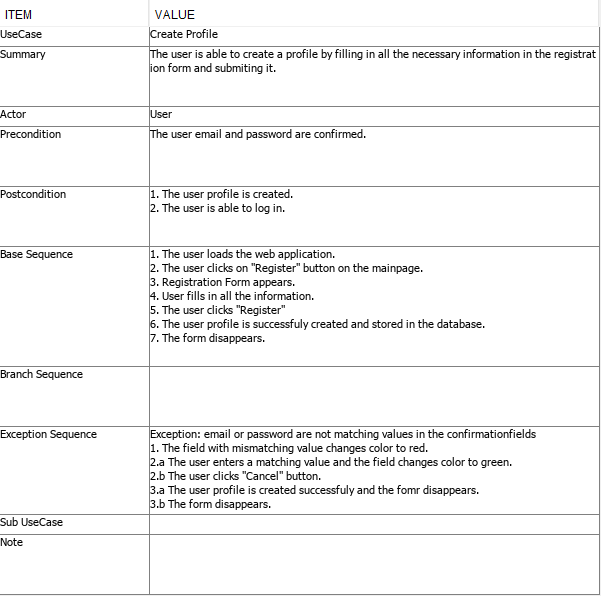
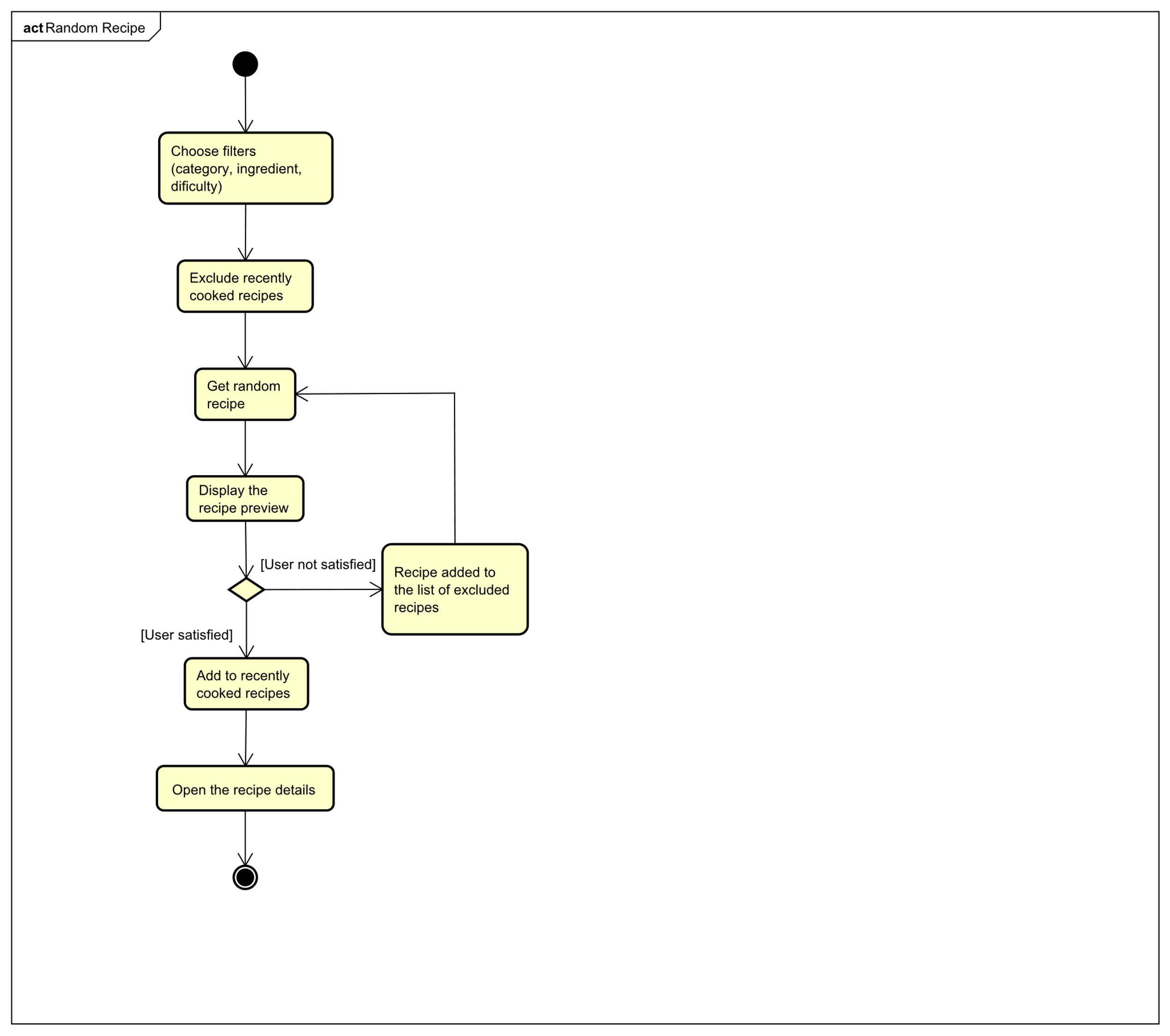


Figure 2 Use Case Description

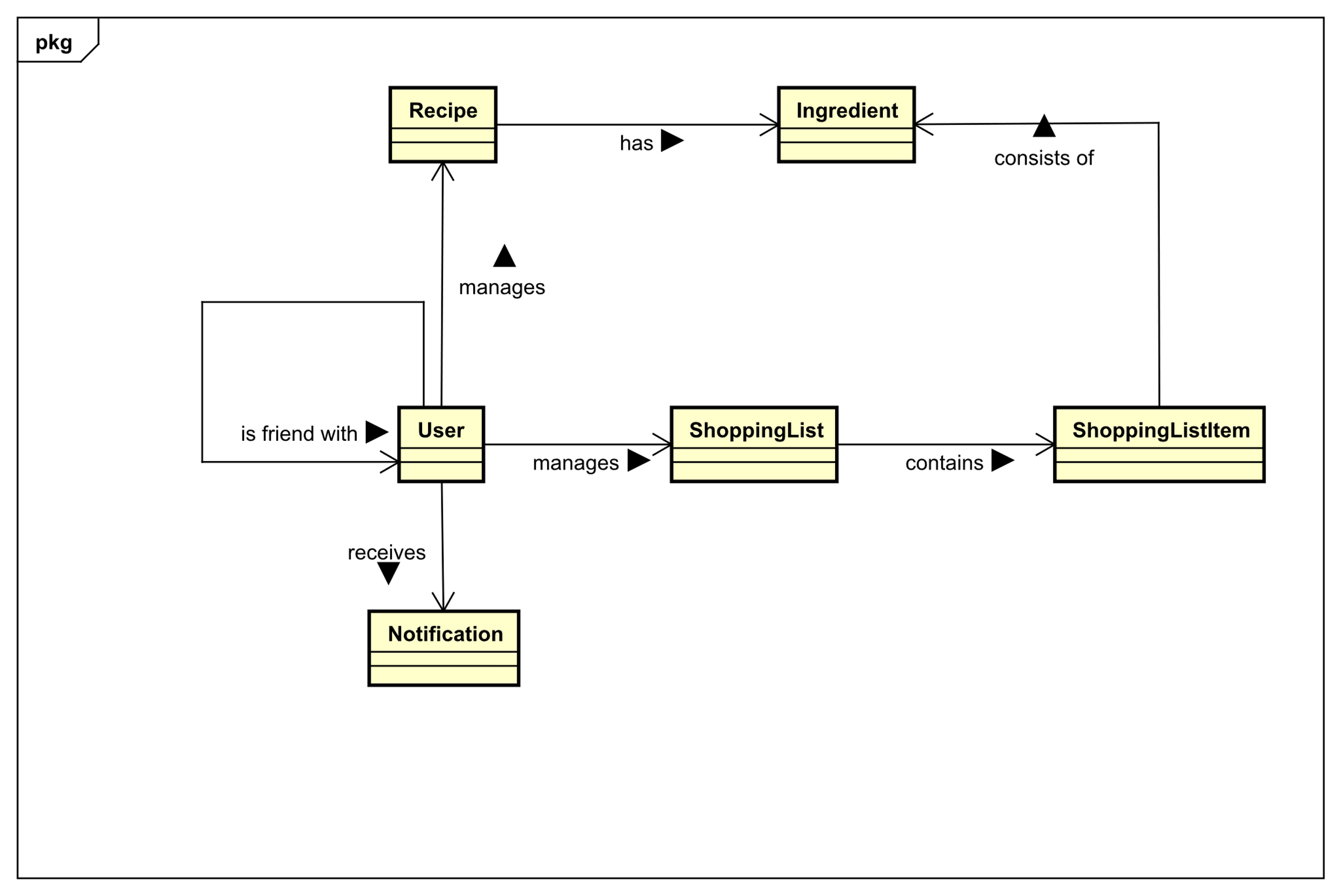
Another tool that was used to provide a better insight to the dynamics of the system are activity diagrams. Activity diagrams were used to depicture the flow between the individual activities. It also catches the behavior of the system specifically the actions that will follow after a certain activity was executed, depending on the users input. Following activity diagram **Figure x** shows activities involved in the recipe suggestion functionality.



The diagram starts with the user providing a filter parameters to give a more relevant results. After the irrelevant recipes are filtered out, the recently cooked recipes are removed as well as the rejected results, so the user does not get any unwanted suggestions. In the next step the system randomly picks a recipe that fulfils the criterion. The recipe preview is displayed to the user who can decide whether to open it to see the full details or can opt for a new recipe suggestion. If the user decides to get another suggestion, the displayed recipe is added to the list of rejected recipes to prevent repetition of the suggestions. If the user decides to open the recipe, the recipe is added to the list of recently cooked recipes and the full recipe information is displayed.

#### Domain model

Domain Model was created to understand the relationships between the object that take part of the system. The users can add other users as a friend so they can easily share recipes with each other. They also receive notifications, so they are up to date with the incoming events. Users can also add and manage their own recipes as they wish. These recipes consist of ingredients and these ingredients can also beaded to the shopping list. Each user has their own shopping list which they can manage any time. This can be seen in the **Figure x** below.



### Design

This chapter is dedicated to the decisions made when deciding upon the system architecture as well as the user interface design. It exaplains the reasons behind it and the steps that led towards them. Moreover it discusses individual parts of the system and their responsibilities.

#### System Architecture

The system consists of three main parts which are MySQL Database, Appliaction Server and finally Web Application Client. Since one of the non-functional requirements is that the client application should be platform independent, it targets all the standard screen resolutions, so it is available at personal computers, tablets and mobile phones as well. The approache and means to achieve this goal will be discussed in more detail later in the UI section. The Figure **X below**, displays the system architecture and the communication between the individual parts.

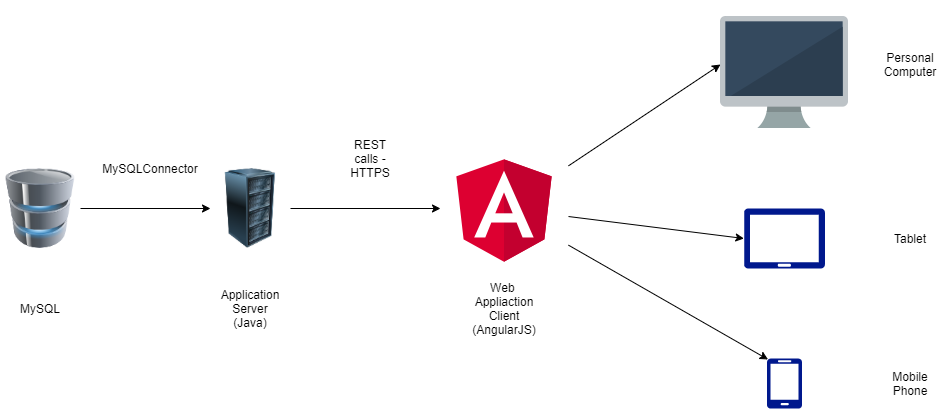


Figure 3System Architecture

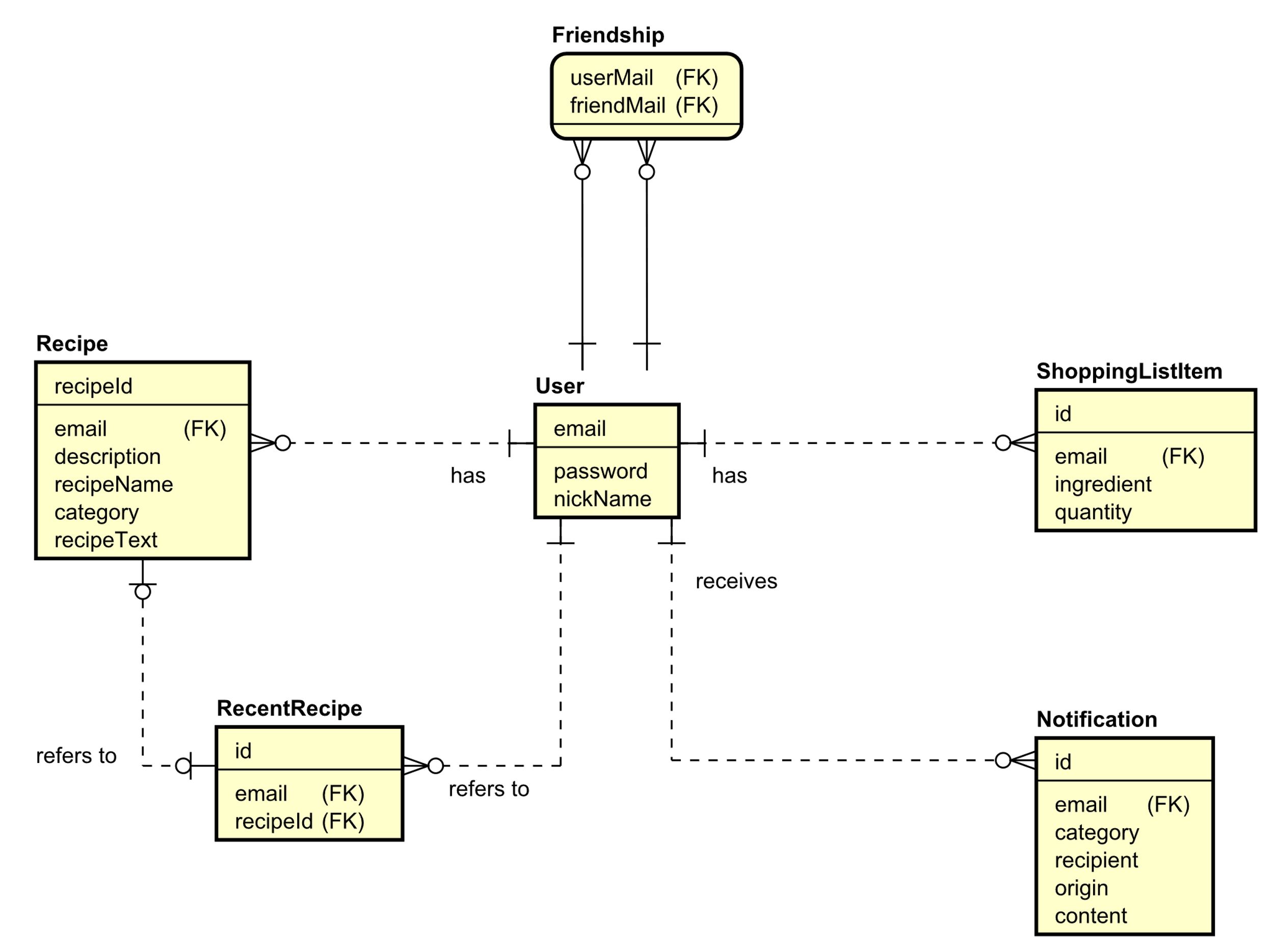
Firs part of the system MySQL database takes the responsibility for the handleling the data and enforces that data logic is persistant.

Second part of the system is the application server. Server hadndles the communication with the database as well as with the file storage system. Server is also responsible for the computationally heavier tasks so the devices with less resources are not overloaded.

Last but not least is the web application client which is also responsible for all the computationally lightweight tasks. The client is also responsible for the user interface and handleling the user interaction.

#### Database design

Database design was developed in order to fulfil the requirements stated in the analysis. A relational database was chosen as it fits the needs of this project. As the focus of this project is on the web application a relatively simple database was required. It stores mainly the recipes, shopping list items, notifications etc. Each of them is stored in separated relation – table. These tables are the result of examining the functional requirements. Entity relationship diagram in **figure x** demonstrates the relationships between the database entities and how they depend on each other.



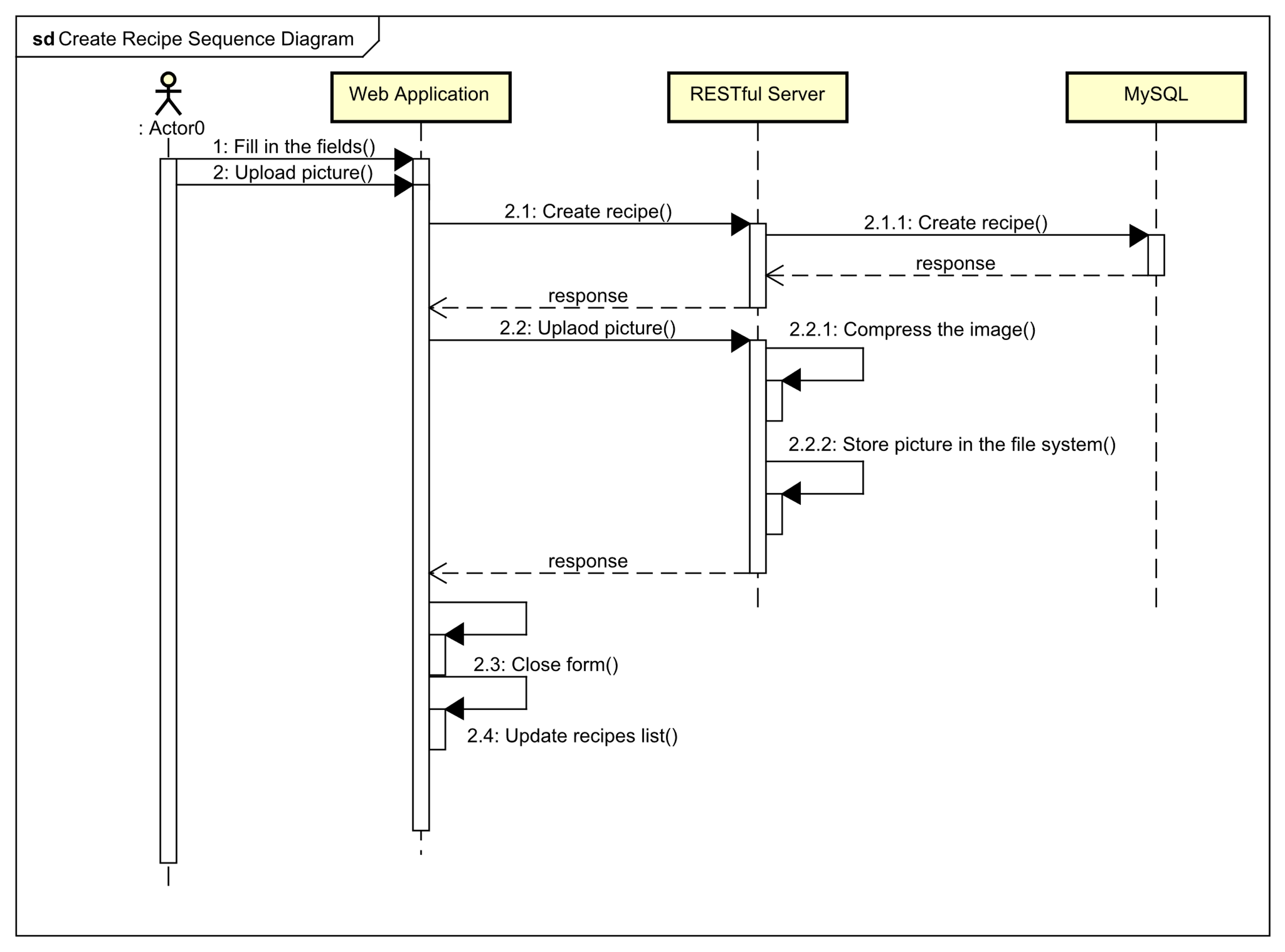
#### Class diagram

Class diagram of the back-end in the **figure x** provides an overview of the infrastructure of the system. All the data is being stored in the MySQL database except the images which are stored in the file system. The server is deigned as RESTful server and its end points are separated into the controllers. Each of the controllers is responsible for a specific functionality of the system using mainly GET, POST, PUT and DELETE which are HTTP requests to handle them. Also, they use objects that represent the data when communicating with the database to either store them or forward them to the client. This is done throughout the data access object (DAO) class which is responsible for the CRUD functions to manipulate the data. Achieving this goal is secured by executing the SQL statements. Another important part of the system is ImageManager class. Its purpose is to handle the images of the recipes, specifically to store, retrieve and remove them from the servers file system. In addition, ImageManager also performs compression of the images to reduce their size and thus help to save resources of the file system.



#### Sequence Diagram

To demonstrate the interactions between the individual layers of the system a sequence diagram was created. Sequence diagrams contain lifelines of objects and messages exchanged between them. They provide an overview of the order in which the functionalities are executed, depicturing the behavior of the system over the time. The diagram in the **Figure x** represents the flow and the messages exchanged between the layers involving the user when creating a new recipe in the system.



Firstly, the user fills in the form and uploads the picture from their device. When the form is submitted, the recipe information is forwarded to the server. Server then tries to store the information in the database and returns the result to the client with the id assigned to the recipe. If the database stores the recipe information successfully, the client uploads the picture linked to the recipe by id to the server where it is compressed and stored in the file system. On completion a response is sent to the client which acts on it by closes the form and updates the list of displayed recipes with the newly created one.

#### Design patterns

The focus of this section are the design patterns used in this project to contribute to a cleaner architecture. Design patterns are usually chosen as a standardized solution for certain type of issues. This means they are well known, and the further development is fast even when a new developers take over since they are already familiar with it.

#### Dao

Data access objects are one of the design patterns. Their purpose is to separate the application logic from the data persistence. This means that the application is independent of the implementation of the data retrieval from the database system.

#### MVC

Model view controller architectural design pattern is one of the standards in the software engineering. Its aim is to separate the application logic to provide readability and maintainability. One of the reasons the AngularJS which is a JavaScript framework was chosen in this project is that it enforces the MVC-like pattern in the application and also it makes its implementation easier.

The view consists of HTML templates which handle the interaction with the user. It takes the user input and displays the results of the user actions. It also communicates with the model and updates it when new information is obtained.

The controllers contain the functions and handle the responses to the users input. Controllers also manage the views and display the relevant once. They also define the initial state and behavior of the data. The main purpose of the controllers is to handle the business logic.

Model in AngularJS is implemented by default as a singleton. This means that there is only one instance containing the data. Model is also a single source of truth which is due to the two-way data binding implemented by AngularJS. This means that the view updates the model and vice versa the model updates the view in real time.

AngularJS Service

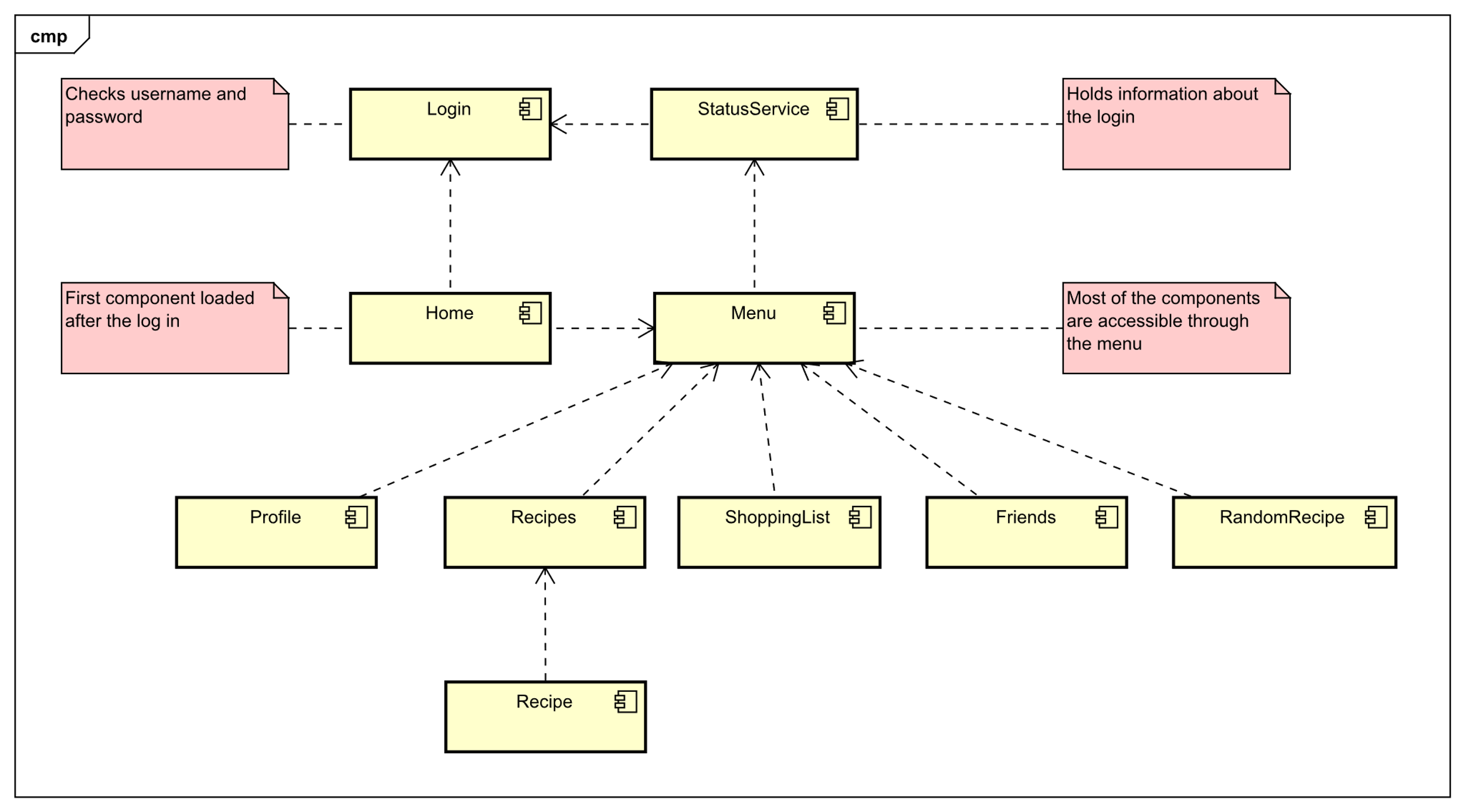
Switching between different views in the application results in destroying the controllers with the corresponding model. This means that a way to preserve the data is required. This is handled by creating an AngularJS service. The service is view independent and is able to transfer the data from one controller to another. This is used for example to verify if the user is logged in, when trying to access any route of the application.

Interfaces

Last design decision is use of the interfaces. Interfaces assure that certain functionalities of the system will be implemented without the need to modify any other code except its implementation. E.g. IImageManager interface was created and used in the rest endpoints. This means that the REST endpoint is independent of the solution to handle the image management. If the file system is changed to e.g. some external system, only implementation of the IImageManager Interface needs to be changed with the REST endpoints remining intact.

### Web Application Client Components

Component diagram in the **figure x** was created to show the individual components of the application and their relationships. Except the Status Service, each of the components is made of the HTML template with corresponding SASS styling and JavaScript controller. Status service holds the information about the user login as well as other data that need to be transferred among the components. Its value is set when the user logs in. As the menu is present in each part of the application after logging in, it checks for the user login every time there is a switch in views using the Status Service.



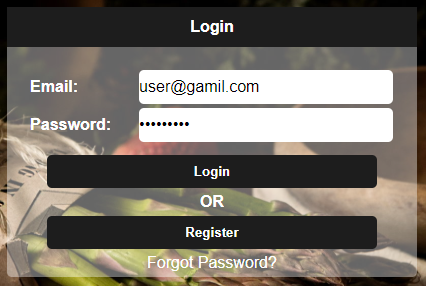
### Implementation

This chapter provides the technical details and answers the question about how the design was implemented in the code. Code snippets are present, to reveal how the system works under the hood and explain how the different divisions of the system interact with each other. Also it gives the reasoning behind the choices of the technologies.

#### AngularJS and JavaScript

#### Login

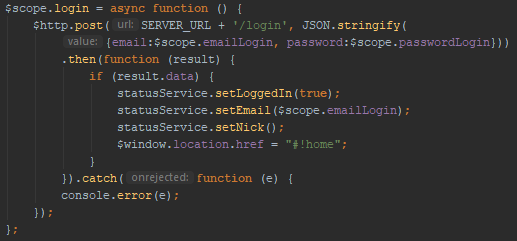
Login functionality was implemented to verify the registered users. The login form in **figure x** is located at the main page after visiting the web application. User can log in by filling in the email with password and pressing the Login button.



This functionality is contained in the login controller which was created with AnglurJS function as shown in the **figure x** below. Status service is added to the controller via dependency injection for a later use.



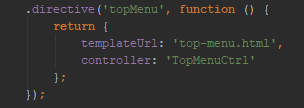
In the **figure x** we can see the implementation of the login function. Inside the function an HTTP post request sends the data encapsulated in an object to the server using the REST endpoint. If the user was verified successfully the status in the Status Service is set to true as well as additional information that are used in different parts of the applications like email and nickname. Last step is redirection to home view using $window directive.



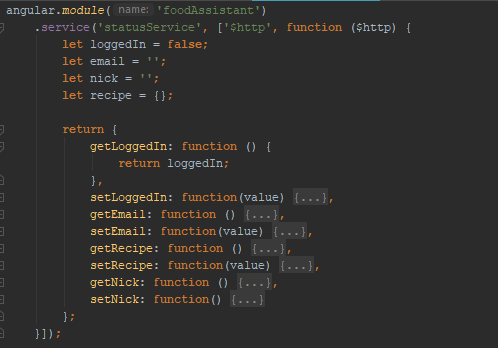
As displayed in the figure x, the direct accessing of the application components is not possible without logging in at first. The user will be redirected automatically to the main page.



This functionality is presented in the menu directive as it is present in all the components. Menu is a custom directive created in AngluarJS and is linked to its own HTML template as pictured in the following figure.

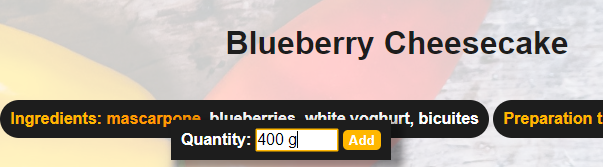


As mentioned in the design section, Status Service holds the data that are needed outside of the Angular controllers. It consists of a few fields and the functions to manipulate them.

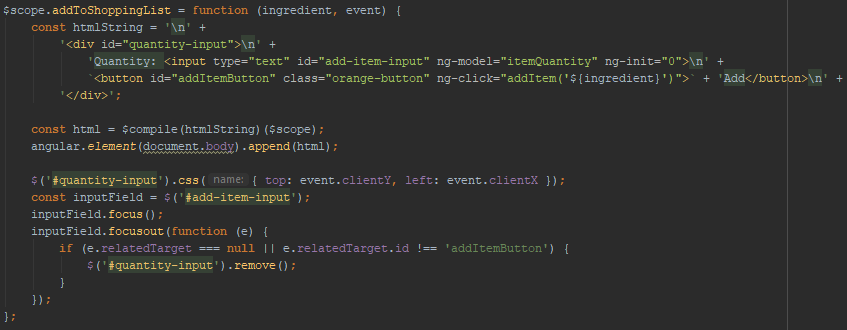


Shopping List

It is possible to add ingredients into the shopping list directly from the recipe description. If the user clicks on one of the ingredients a pop-up window appears with field for quantity as shown in the **figure x**.



This was implemented by creating an HTML template containing input and button elements. This HTML code is then compiled so the AngularJS is aware of the new code and afterwards it is appended to the document body. As result a pop-up window papers when the code is executed. CSS styling was used to position the pop-up box right next to the ingredient for more intuitive user experience. This is achieved by using event element which contains the information about the ingredients text position. The pop up disappears when it loses the focus or when the ingredient is added to the shopping list. This is captured in the **figurex**.

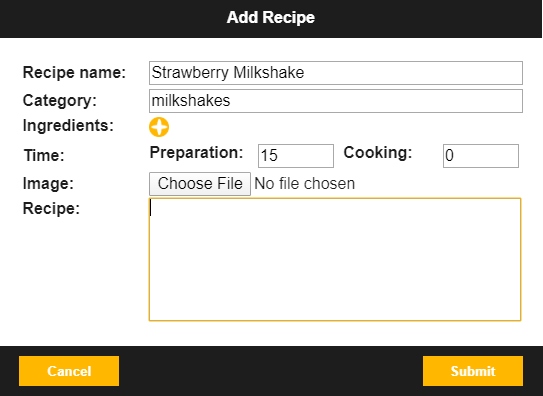
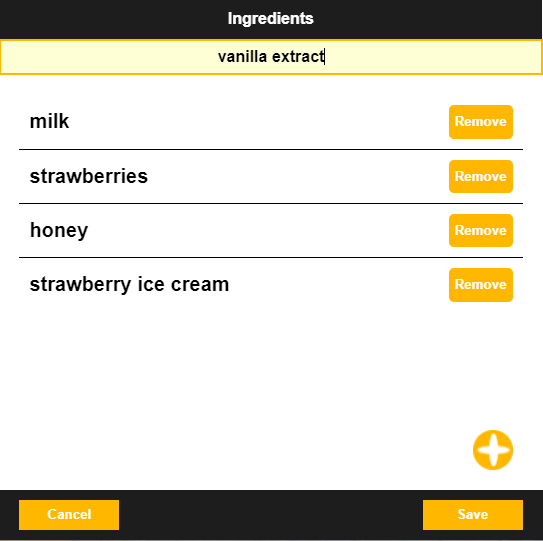


Create recipe

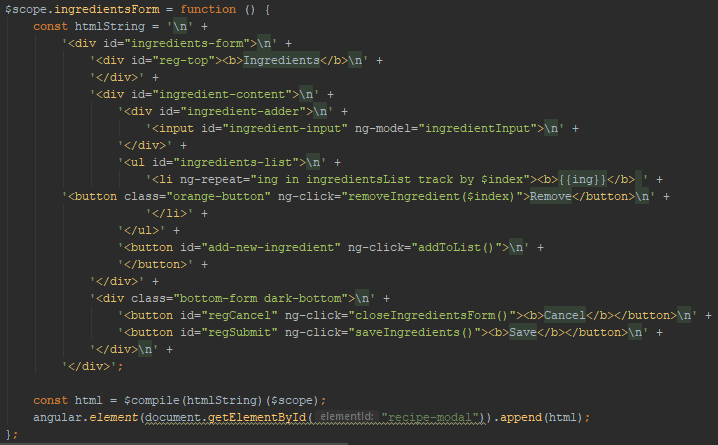
A new recipe can be created by filling in a recipe form **Figure x**. This can be accessed from any part of the system a as this functionality is part of the top menu as can be seen in the figure below.



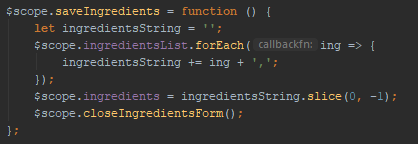
The form contains input fields as well as an option to upload an image. Also form for adding ingredients can be accessed from here – **Figure x**. The ingredients can be added by typing in the ingredient name and pressing the “plus” button in the right bottom corner.

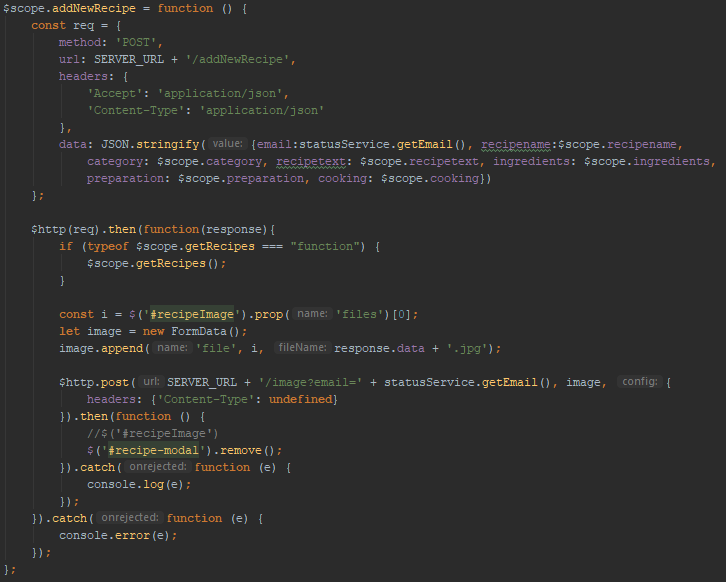
Similarly like the ingredient pop-up, the form for adding recipes and ingredients are HTML strings compiled and appended to the document body. This can be seen in the **figure x**.



When the ingredients are saved, they are converted to a single string holding all the recipe ingredients. This is done by concatenation of the ingredient strings and separating the by coma.

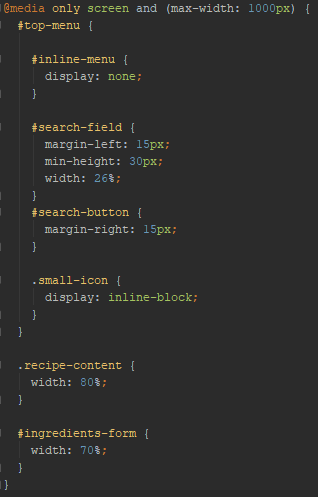
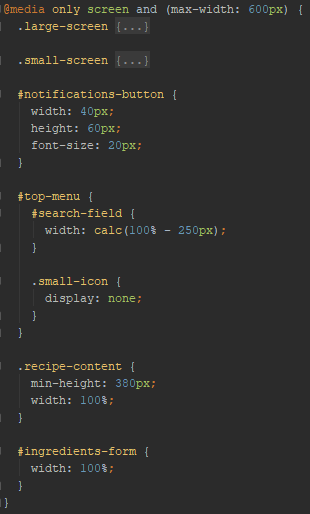


As shown in the sequence diagram in the previous chapter in **figure x**, when adding a recipe at first the recipe information is sent to the server. This is done via HTTP request consisting of defining attributes. Method specifies the request type to be POST as the intention is to add new data. URL sets the address of the server with the route for specific end point. Headers identify the application content and data attribute holds the information that the server will process further. The rest calls return a promise, if the promise is resolved successfully the list of recipes is updated. This is followed by selecting the recipe picture uploaded by the user and appending it to the form data object. Image in such format can be send through the subsequent POST request. If the request completes successfully the form is closed.

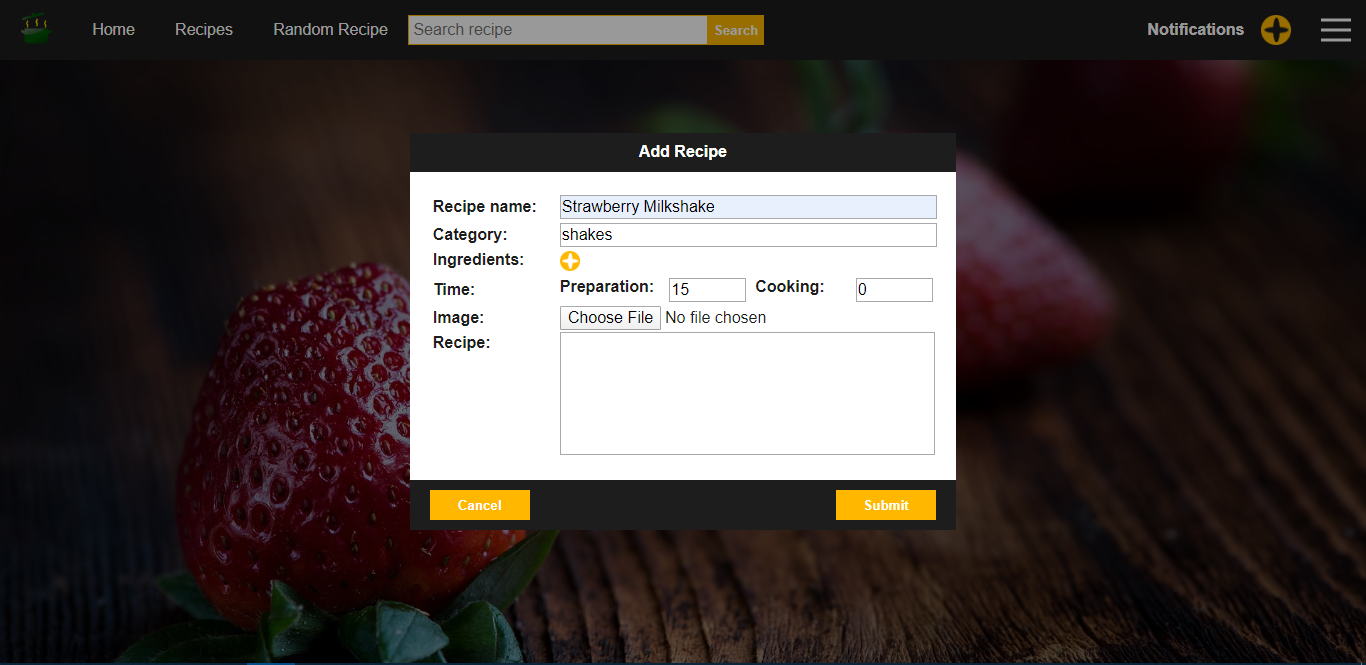


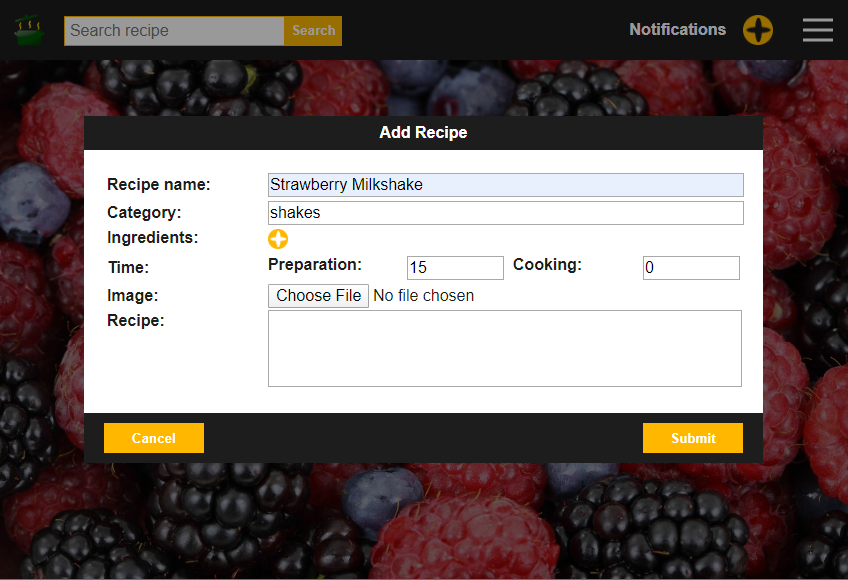
#### SASS

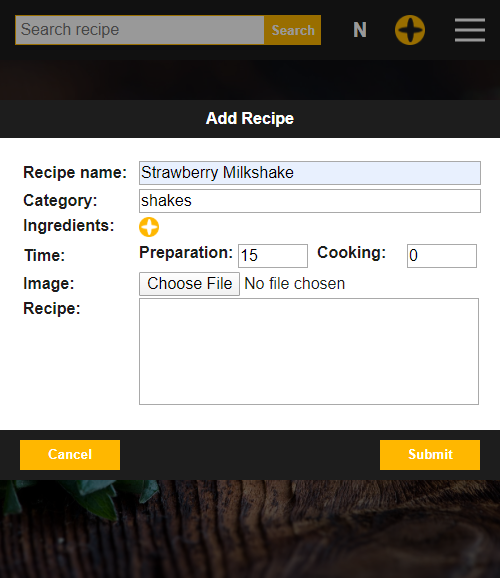
One of the non-functional requirements is that the system shall be responsive. This was achieved by implementing the media query technique using the @media rule. This way the properties of the HTML elements can be changed automatically when certain conditions are met. In this case the rule was the screen resolution below 1000px and 600px.The implementation in the code can be seen in the **figure x** for the resolution of 1000px and in the **figure x** for the resolution of 600px.

In the **figure x** you can see the form for adding recipe before any break point is reached.

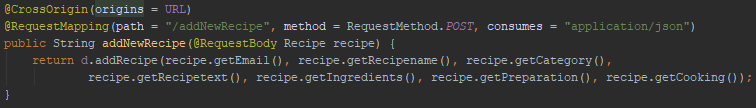






#### Java

The server implementation is done using Spring framework and consists of REST endpoints. Each of the endpoints has specific path mapping through which can be accessed from the client. The request method and type of the data it deals with is also defined here using @RequestMapping decorator. The function handling the request returns a string containing the id of the recipe created because it is needed for saving the linked picture of the recipe. In the function body we can see that DAO is used to handle the data.



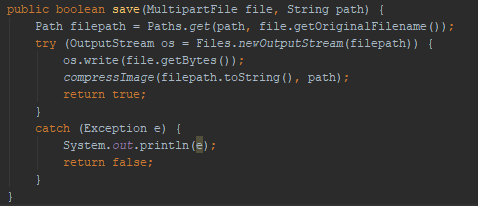
Inside the DAO class, the data is used to create an SQL statement which is then executed and stores the new data in the database. It is followed by query that retrieves the id of the newly created recipe. It needs to be done separately as the database assigns the IDs automatically.



Next the image is received through another endpoint. This on is implemented also in its own rest controller with a difference that it uses Image Manager class instead of the DAO as it is not accessing the database but the file system instead.

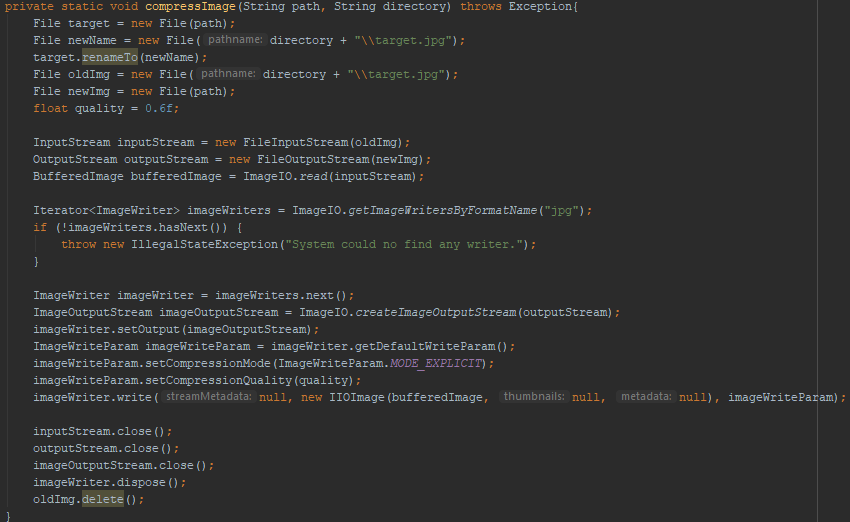


The image is saved to the folder created specifically for each user by their email address. Afterwards the image is compressed.



#### Compression

In the first step the picture is located, new name is created, and the target picture is renamed so the compressed version can keep the name linked to the recipe. New file object is created, and the compression quality is defined as 0.6. The compression is done using javax package specifically ImageIO class. After the initialization of the streams (input for the original image and output for the new one) a suitable image writer is found. The parameters are set on the ImageWriteParam object. The writer is used to write the compressed image and takes the ImageWriteParam as parameter. The result is a new compressed image. The final step is closing the streams and the old image deletion.



SQL

The **figure x** displays how the relation holding the recipes was created by following the **figurex**. Recipeid column, which is the primary key, is as mention before, handled by the database by autoincrementing. All the attributes cannot be null to assure all the functionalities of the system will be available to the user. All of the columns have a data type specified as well as restrictions on their sizes. The email in the recipe is a foreign key from the UserLogin table to assure a relationship between the users and their recipes.



### Test

This chapter talks about how the software was tested to assure that the product was implemented correctly in the preceding chapter and that the customer’s needs will be fulfilled.