

"ALEXANDRU IOAN CUZA" UNIVERSITY OF IASI  
**FACULTY OF COMPUTER SCIENCE**



BACHELOR THESIS

**ChefsCorner - Web Application for Restaurants  
for Employee Training and Recipe Management**

proposed by

**Andrei Iacob**

**Session: june/july, 2023**

Scientific Coordinator

**Assoc. Prof. Panu Andrei**

**"ALEXANDRU IOAN CUZA" UNIVERSITY OF IASI**  
**FACULTY OF COMPUTER SCIENCE**

**ChefsCorner - Web Application for  
Restaurants for Employee Training and  
Recipe Management**

**Andrei Iacob**

**Session: june/july, 2023**

**Scientific Coordinator**

**Assoc. Prof. Panu Andrei**

# Contents

<b>Introduction</b>	<b>3</b>
<b>Contributions</b>	<b>5</b>
<b>1 Application description</b>	<b>7</b>
1.1 Addressed problem . . . . .	7
1.2 Proposed solution . . . . .	7
1.3 Application features . . . . .	8
1.3.1 Sign in and sign up . . . . .	8
1.3.2 Search premade recipes . . . . .	9
1.3.3 Upload recipes with video instructions . . . . .	10
1.3.4 Allow access to other users . . . . .	11
1.3.5 Requests access to recipe . . . . .	11
1.3.6 Quantities update on recipe size and ingredients count . . . . .	13
1.3.7 Price information with recommended add-ons . . . . .	13
1.3.8 Ingredients provider tracklist . . . . .	15
1.3.9 Creating a menu . . . . .	16
1.3.10 Export menu as PDF . . . . .	16
1.3.11 Recipes recommendation for specific ingredients . . . . .	17
1.4 Similar solutions . . . . .	18
1.4.1 Getmeez . . . . .	18
1.4.2 eatthismuch . . . . .	19
1.4.3 Tasty . . . . .	20
<b>2 System Architecture</b>	<b>22</b>
2.1 General architecture . . . . .	22
2.2 Application backend . . . . .	23

2.3	Application frontend . . . . .	31
2.4	Data storage . . . . .	33
2.5	Security aspects . . . . .	35
<b>3</b>	<b>Use cases</b>	<b>38</b>
3.1	Posting a new recipe . . . . .	39
3.2	Price estimation for preparing a meal . . . . .	40
3.3	Managing all ingredients prices . . . . .	42
3.4	Managing access permissions over recipe . . . . .	43
3.5	Creating and populating menus . . . . .	44
3.6	Using the application as a new employee for training purposes . . . . .	45
	<b>Conclusions and Future Improvements</b>	<b>47</b>
	<b>Bibliography</b>	<b>49</b>
	<b>Appendix 1. Application deployment</b>	<b>52</b>
	Backend server . . . . .	52
	Frontend server . . . . .	53

# Introduction

Cooking is a work of art, it gives us the opportunity to experience new feelings, and creative ideas by making the food a part of our own selves. It lets us explore the art of taste, the art which unites competence, creativity, and communication. A delicious recipe can be the reason which brings together families and friends. Whenever we cook something mouth-watering, the reward we get is the joy we see in others' eyes.

This thesis approaches a common and old problem of every restaurant: managing the expenses for ingredients and training new employees with the right equipment by desired standards. This problem has been around since the appearance of the first inns and is still a common issue to this day.

The restaurant industry is very competitive, so, having good management of your staff is very important because time is the key to success. Digital solutions constantly develop, bringing new features and ways to ease unnecessary human work.

This application proposes a simple-to-use web application that gives the users a solution for managing the costs of every meal, allowing the users to scale the quantities of ingredients, thus having a more realistic view of the final cost of preparing a recipe. Before, managing the costs of individual ingredients was a difficult task, one that was predisposed to human errors, errors that could make a difference in this competition.

Showing every new employee how to prepare a new meal using the professional tools available is a time-consuming task, one that can be avoided by providing video instructions for each important step.

This document is structured into four chapters describing the proposed application from use cases to architectural decisions. Chapter 1 discusses the addressed problem, also giving information about the proposed solutions and the features which the current implementation has. Chapter 2 describes the application from a technical point of view, presenting the architectural decisions made for the backend and the frontend components of the applications, and the choices made for data storage and

security implementations. Chapter 3 offers two use-case scenarios of the application, depending on the role that the user has in a restaurant: as an owner or admin, and as an employee. Chapter 4 enumerates the steps to be followed to run the application, and what packages and modules are needed for the correct usage of the application.

# Contributions

The idea of this project was to create more than a recipe library, but a tool that could help cooks around the world to be more time efficient and more precise. The main idea of this application is not new, but along with the features that can be found in similar solutions, bringing them all into one place, an application where chefs and restaurant owners could manage their recipes and costs more easily, training their staff in a more time efficient way or to edit and generate a menu list just with some clicks.

The presented solution significantly contributes to the domain of recipe management and employee training by discussing important points in the managing area:

- Identifying the needs of businesses from the gastronomic field and restaurants, as they face great challenges during the crucial stage of transition towards digitalization, in managing the recipes and team training in a constant and time efficient way;
- Evaluation of existing solutions with regard to the existing problem, which represents an important part towards defining and implementing of a new solution. The current solutions offer a good perspective of the actual state of technology in the culinary domain, offering information about the state of merging of these two different work fields;
- Defining a complete solution and establishing objectives, which take into account the already identified obstacles and the analyzed competition solutions. Once the solution has been defined, it will server as a starting point for the implementation of the system.
- Implementing the predefined solution and personalizing it according to the described requirements and special necessities of the problem. Once fully implemented, the final system will serve as a tool for businesses to manage recipes and

expenses, and train the newcomers in an efficient and interactive way, leading to a great improvement in product quality.



# Chapter 1

## Application description

### 1.1 Addressed problem

The idea of the application came from the basic need of humans to store more data, representing recipe instructions and ingredients needed in order to prepare a meal in the best way possible. The idea evolved into a more business-oriented solution, targeting the idea of saving as much time and money as possible. As solutions already exist, all of them have advantages and disadvantages, some being more home-cook oriented, others being limited in terms of the number of features.

As such, defining and creating a system that is also easy to use and powerful at the same time would be the defining solution for the problem that is present right now.

### 1.2 Proposed solution

The proposed solution described and documented in this thesis consists of a system composed of a frontend server and a backend processing server. The frontend server will be the graphical interface that presents the user with the information, being a bridge between the user and the processes that compute the data. It will offer a user-friendly interface, in order to navigate and take advantage of each feature of the proposed solution. The backend server will expose an API in order to receive requests, requests that will execute the business logic in order to deliver the desired data.

This project can be crucial in a working environment, solving some of the most common problems, such as managing your team's knowledge and your expenses is key in order to have a successful business. On a base level, every solution for these

problems is almost the same, only that my solution tries to combine defining features from competitor solutions.

## **1.3 Application features**

The presented solution was developed around some predefined and studied problems, in order to define and create solutions, that took the shape of application features. The features and characteristics that will be presented in this section are what stand out in this application, creating the ways a user interacts with the system. Below each feature will be described, how it was implemented, and the part it has in different use cases.

### **1.3.1 Sign in and sign up**

As for most web applications, the presented solutions have an authentication and authorization system. It is composed of two main components: the registration part, and the login component.

In order to access the platform, a new user has to create an account, following the presented form, which contains the following inputs:

- Personal data about the user, requiring first name, last name, and the option of either being a business account or personal account;
- An image for the profile account, but the picture is not mandatory, the system will provide a default image if none is present;
- A unique email address, which works as the unique identification factor in the system. The email must be a valid one because it must be confirmed later by the user;
- A password, with a minimum length of 6, containing at least one number, one capital letter, and one special symbol. The password must be confirmed in the following input space.

If the provided data respect all the input requirement, the account will be added to the system, and a verification email will be sent to the user (Figure 1.3.1.1 - created following the template provided by [4]). The validation email will expire in 30 minutes,

and will not be valid. In order to get another validation email, the user has to login into the account, which will send another email. Accessing the route provided by the validation email, the account will be completely registered, and the user can access the platform through the login form.

## Confirm Your Email Address

Hi Ana Maria,

Thank you for registering. Please click on the below link to activate your account:

Confirm your email

Link will expire in 30 minutes.

If that doesn't work, copy and paste the following link in your browser:

<http://localhost:8080/api/auth/register/confirm?token=26f3f18d-c054-4565-a2ba-3cf570b74c55>

Cheers,  
ChefsCorner Team

Figure 1.3.1.1 : Activate account email

After the account registration, the user can access the login component, where it is required to input a unique email and the corresponding password. If the given credentials do not match any account stored in the system database, an error will be shown to the user and will require to reenter some valid credentials. If the provided credentials are valid, the user will be granted access to the system, and be redirected to the main page of the application.

### 1.3.2 Search premade recipes

Any user has the option to search through the presented database of recipes. In order to search for a recipe, users can choose one of the two available options:

Using the search input placed at the top bar of the graphical interface (Figure

1.3.2.1). There, you can search for recipes by keywords or patterns in the meal's title. In order to ask for recipes, the user has to press ENTER. The system will provide recipes that match the search criteria, showing a maximum of 20 options. Clicking on any of the results will redirect the user to the presentation page of that specific meal.

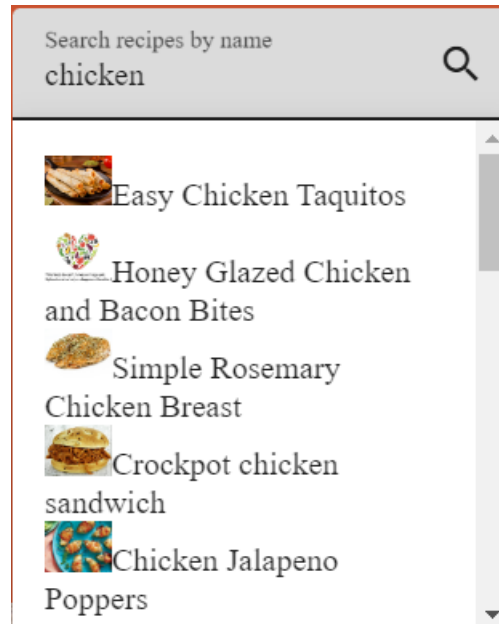


Figure 1.3.2.1 : Search recipe by name

### 1.3.3 Upload recipes with video instructions

Alongside public recipes, that everyone can access, a user is able to create and publish a personal recipe. In order to do that, the user must complete and send a form. The form for creating a recipe is composed of three sub-form, as follows:

The first form contains general information about the recipe, such as the title, a small description, the number of servings, time values for preparation time, and cooking time. Also, the user is requested to upload a suggestive photo of the final product. All inputs are required in order to proceed to the next form.

The second form requires information about the ingredients needed, alongside the quantities. The user can search for an ingredient that is already present in the system data, by typing keywords that are in the name of the ingredient. If there is no match for the ingredient in the database, a new ingredient will be created when the recipe is posted. After the user decided what ingredient to add, he must provide a description of the ingredient, the unit of measurement, and a conversion from the specified unit to grams. In this form, there must be at least one ingredient provided,

with all the input completed. If there is more than one ingredient, all the inputs must be completed in order to go to the last form.

The final form takes in information about the steps of making the recipe. There must be at least one completed input, with the option of adding a supplementary video, which should bring more clarity. The user can add as many instructions as he pleases. If all the inputs are completed, the confirmation panel can be accessed.

In the final scene, the user is asked to confirm the data input. The previous form can be reached in order to verify the information. After the confirmation button is pressed, a request is sent to the system and the recipe is created.

By default, every new recipe is private, only for the owner to access, but it can grant access to others, an option that will be discussed in the next feature. The newly created recipe can be found in the user's own recipe list.

#### **1.3.4 Allow access to other users**

As posted recipes are private in the beginning, it would be of no use if you could not share them with your team. On the presentation page of the recipe, there is a button that opens a dialog, containing options for modifying the access of other users (Figure 1.3.4.1).

After opening the dialog, there are two actions we can take: remove a permission or add a new permission. In order to remove permission, the "Check current permissions" panel must be extended. In line with every permission is a remove button. Clicking it will revoke the user's permission. In order to add a new permission, the new users must be searched using the input. If there are matches regarding the email search, a dropdown will be shown. On selection, the user will be granted access to the recipe.

#### **1.3.5 Requests access to recipe**

Another way for an unauthorized user to access a non-public recipe is to request access. If the user accesses a recipe by changing the id of a recipe from the URL, sometimes he can come across a private recipe. If that happens, a dialog popup will show, asking if he would like to request access (Figure 1.3.5.1).

The user can press the "NO" button to be redirected to the main page, or press the "YES" button in order to request access. Doing so, the system will send an email

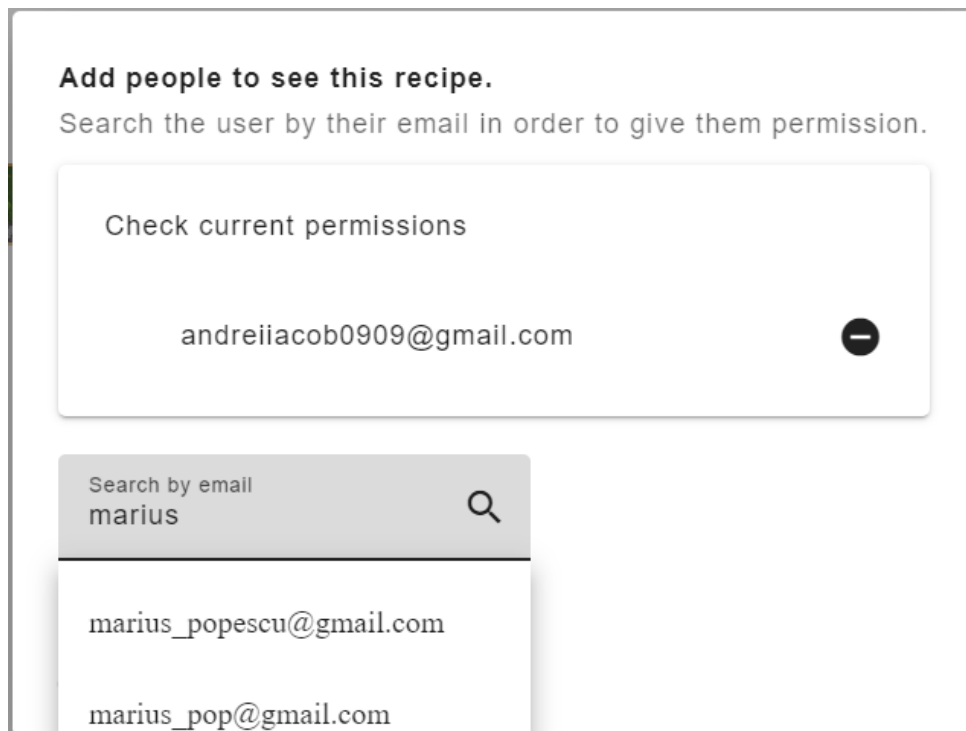


Figure 1.3.4.1 : Add permission dialog

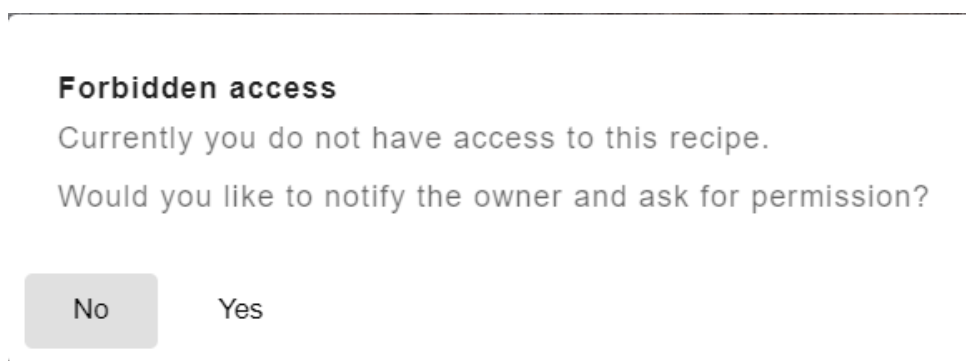


Figure 1.3.5.1 : Forbidden access dialog

(Figure 1.3.5.2 - created following the template provided by [4]) to the owner of the recipes, telling them that someone requested access to a specific recipe. The email will contain all the information about the requester and the recipe. The email has an expiration time of 30 minutes. If the request expires, the requester will have to make another request in order to be granted access.

Clicking the button in the mail will redirect the owner to another page. The page acts as an access request page, where will be presented two options: deny the request or grant access to the recipe.

## Allow access to recipe?

Hi Andrei iacob,

Ana Maria ([andreiicob0909@gmail.com](mailto:andreiicob0909@gmail.com)) would like to access 'Mici pe gratar'. Click the button bellow to allow access.

Grand access

Link will expire in 30 minutes.

If that doesn't work, copy and paste the following link in your browser:

<http://localhost:4200/#/recipe/confirm?token=307ca7ab-2030-4977-b891-d800832294a9>

Cheers,  
ChefsCorner Team

Figure 1.3.5.2 : Grand access email

### 1.3.6 Quantities update on recipe size and ingredients count

Knowing how much quantity you need in order to get the desired amount of product can be a frustrating task. So, bringing a way of generating an auto-scalable recipe description was an important requirement for the system.

Each recipe page follows a pattern for describing the ingredients used and their amount. Each has a specific unit of measurement, with a conversion to an international measurement unit, grams. The number of servings, and the units of measurement for ingredients act as inputs. The input only accepts numerical type, when it detects a change in one input, every other value will be automatically updated in order to match the previously updated ratio. The number of servings is also constantly updated, in order to have a realistic view of the final amount of food.

### 1.3.7 Price information with recommended add-ons

Knowing how much each ingredient costs is a crucial step in deciding the commercial price of a meal. On the recipe page, you can open a dialog that gives your

further information about the required ingredients (Figure 1.3.7.1).

The user is able to change the currency, being able to choose between RON, EURO, and USD, the conversion rates being updated once an hour [5]. For necessary ingredients, the price is computed based on the required quantity and the price per unit. The dialog uses the quantities set by the user using the previous feature. A total price will be shown, plus an editable percentage add-on, that would guarantee the restaurant a profitable trade.

For each ingredient, the user can add information about different providers, helping them keep track of the best deals. The input only requires the seller's name and the price per unit rate. Each seller's information can be edited and removed.

#### Estimated prices for ingredients:

Currency

RON

**Ketchup:**

1 \* 2.73 = 2.73 RON

Seller	Price/Unit	Action
Metro	RON 5	
Auchan	RON 4.5	
new seller	RON new price/unit	

**Beef Plate:**

20 \* 1.94 = 38.83 RON

Total: 41.56 RON

Total with profit: 72.72 RON

%

75%

Close

Figure 1.3.7.1 : Price ingredient dialog for recipe



### 1.3.8 Ingredients provider tracklist

For a restaurant is important to keep track of the expenses and the best prices for each product. Keeping all in one place shortens the work an employee has to do in order to update prices based on a real receipt.

The user is presented with all the ingredients he has added a seller to, being able to edit every information available or delete it (Figure 1.3.8.1). Removing all seller's information will not automatically remove the ingredient from the page, the user would still be able to add new data to it.

At the bottom of the table, a search input is present, which allows us to look for new ingredients for which to add new prices. Each new ingredient will be empty at first, with only the option to add new data.

Currency  
RON

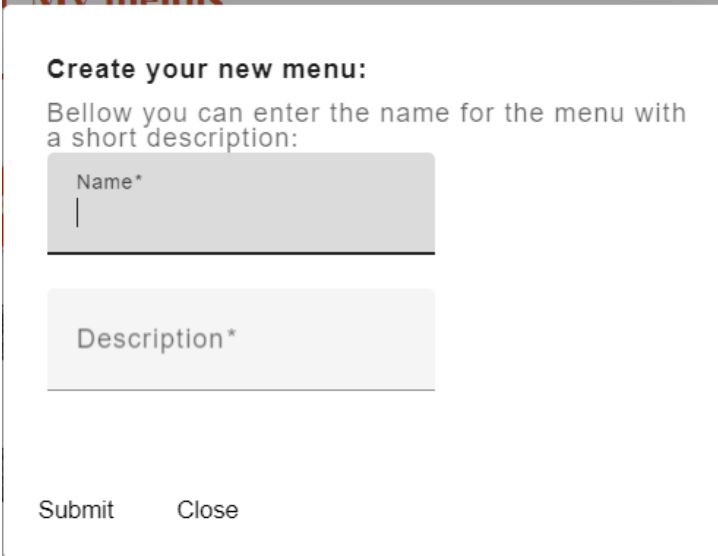
Crispy Corn Puffs Cereal:		
Seller	Price/Unit	Action
Kaufland	RON 27.82	⊖
Megalmage	RON 10.02	⊖ ✎
new seller	RON new price/unit	✎
Almond Milk:		
Ketchup:		
Chicken Breast:		
Potato:		

Search ingredients by name

Figure 1.3.8.1 : Ingredients providers tracklist

### 1.3.9 Creating a menu

Users can sort recipes in menus. In order to add a recipe to the menu, it must be created first (Figure 1.3.9.1). Each menu must have a title and a description. If the required inputs for creating the menu are completed correctly, then the menu is created and the user can start adding recipes to it.

A screenshot of a web application popup titled "Create your new menu:". Below the title, there is a text prompt: "Bellow you can enter the name for the menu with a short description:". There are two input fields: the first is labeled "Name\*" and the second is labeled "Description\*". At the bottom of the popup, there are two buttons: "Submit" and "Close".

**Create your new menu:**

Bellow you can enter the name for the menu with a short description:

Name\*

Description\*

Submit Close

Figure 1.3.9.1 : Create a new menu popup

### 1.3.10 Export menu as PDF

A menu would be of no use if it would be seen only by the employees. While on the menu page, you can click the "Export as PDF" button, which will generate a PDF based on the recipes on the menu (Figure 1.3.10.1). Recipes will be classified by their respective category. The price offered on the menu will be a recommendation based on an estimated value of every ingredient used, plus an editable in-menu page add-on rate, that would guarantee the restaurant a profitable trade. The currency of the menu can be chosen between RON, EURO, and USD, with conversion based on the latest rates. Ingredients used for every recipe are shown, along with the quantity of each component.

Menu de vara		
Lorem ipsum dolor sit amet, consectetur adipisicing elit. At beatae cumque deleniti deserunt dolor ducimus enim hic quisquam quo tenetur!		
MainDishes		
Almond Butter & Celery	4.03 RON	
Almond Butter, Celery		
Appetizers		
Penne with Roasted Asparagus and Balsamic	8.54 RON	
Asparagus, Olive Oil, Salt, Pepper, Whole Wheat Pasta, Balsamic Vinegar, Brown Sugar, Butter, Parmesan Cheese		
Corn Thins and Avocado	2.85 RON	
Avocados, Corn Thins, Pepper		
Avocado Hummus	25.25 RON	
Chickpeas, Avocados, Fresh Cilantro, Lime Juice, Water, Garlic, Salt		
Wilted Spinach with Nutmeg Butter	29.36 RON	
Nutmeg, Butter, Spinach		
Tomato Gratin	15.60 RON	
Garlic, Tomatoes, Bread Crumbs, Olive Oil, Basil, Parmesan Cheese		
Sesame Broccoli	16.07 RON	
Broccoli, Sesame Oil, Sesame Seeds, Soy Sauce		
Drinks		
Coconut Avocado Smoothie	8.05 RON	
Avocados, Vanilla Yogurt, Reduced Fat Milk, Coconut Cream, Ice Cubes		
Strawberry, Peach, and Chia Smoothie	9.42 RON	
Oranges, Peaches, Strawberries, Spinach, Chia Seeds, Ice Cubes		
Blueberry Watermelon Smoothie	19.30 RON	
Watermelon, Blueberries, Lime Juice		
Coconut Pineapple Orange Smoothie	9.38 RON	
Coconut Milk, Oranges, Banana, Pineapple, Plain Yogurt		
Banana Blueberry Smoothie	28.52 RON	

Figure 1.3.10.1 : Menu PDF exported

### 1.3.11 Recipes recommendation for specific ingredients

Not knowing what to cook with the available ingredients can be a frustrating thing. With this problem in mind, a system that recommends recipes based on those ingredients was created

In order to get recommended recipe, first you need to input the available ingredients (Figure 1.3.11.1). The user will search for ingredients using the given input, looking for them based on the keywords. A user can add as many ingredients as they want, but it requires at least three ingredients. The list of used foods can be edited, removing them anytime.

The system will try to retrieve only recipes with all the required ingredients, but if there are not any, it will try to search for recipes with at least the length of the list - 1 match.

The user will be presented with a maximum of 10 recipes, having the option the check every recipe, and add it to the desired menu.

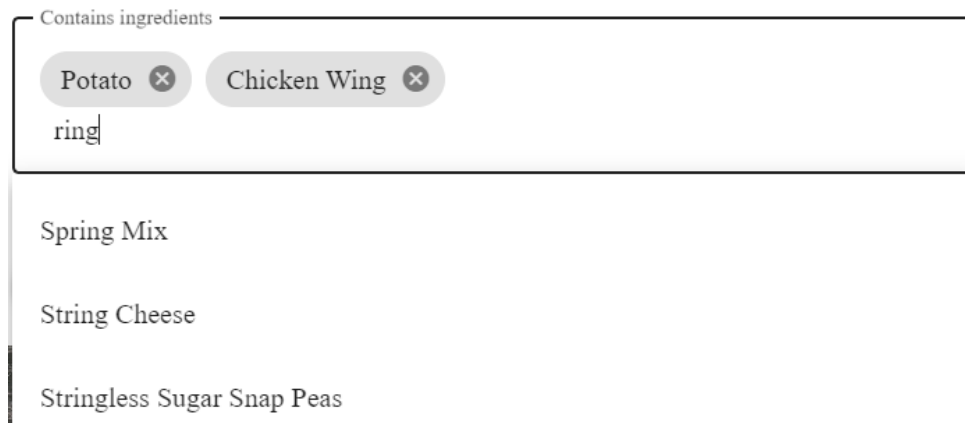


Figure 1.3.11.1 : Search recipes by ingredients

## 1.4 Similar solutions

In every domain, it is natural for problems to appear, so, for every requirement, there are tools for solving those problems. Our current problem is no different, currently existing multiple solutions for us to try, everyone with different features for different use cases. With a simple search on the “Google” search engine with “recipe manager”, we will be overwhelmed by numerous solutions. After careful studies of the available studies, three main solutions have been identified, with defining features: GetMeez, eatthismuch, and Tasty.

### 1.4.1 Getmeez

GetMeez application is a relatively new solution on the market, being released in 2020, after just two years of development, but saw an amazing rise in popularity among restaurants around the world. Meez is an “all-in-one” tool, a “first-of-its-kind culinary operating system created for chefs by chefs” [1], that is oriented for business use, offering support in all areas related to managing a restaurant. Some of their highlighted features are as follows:

- Food costing offers users the ability to better estimate the costs of preparing a meal, offering a view of the lost quantities during the process of cooking;
- Staff training, offering the new employees constant pieces of information about every instruction, with suggestive pictures and videos added by the training team;

- Scaling and conversion of ingredients to reduce waste in order to offer desired quantities of meals;
- Recipe access and sharing allow teams to share recipes with each other, being a useful tool for training new staff;
- Nutrition analysis offers precious information about calorie counts for menus and pieces of information about allergens;
- Kitchen management allows chefs to create their own recipes, share them with the rest of the team, offer real-time updates, and reduce misunderstandings.

As GetMeez is oriented toward business use, it has the downside of being limited by subscription-based use, with each new level unlocking new features. In appearance a disadvantage, behind the subscription is one of the most important features, constant feedback from the consultants of the GetMeez team and constant updates with more features.

In conclusion, GetMeez offers much-appreciated features in a professional kitchen, connecting teams in real-time, and being a must-have companion.

### **1.4.2 eatthismuch**

Eatthismuch [2] application started as *swole.me*, a simple web tool that is oriented to help athletes calculate and generate new meals within a specified amount of calories. Later, *swole.me* evolved into *eatthismuch*, a web tool with an increased interest in diets, nutrition strategies, and planning meals. Over the years, the platform evolved into a big repository of recipes, with in-depth data about ingredients in meals. Some of their distinctive features are the following:

- Meal plan generator, a tool that helps you generate a meal within a given amount of calories per day, meals that are separated into a given number of meals, taking into consideration the previous feedback;
- Food recipe browser, allowing the users to filter recipes by categories and within limits of calories;
- Scalable and conversion of the amount of ingredients needed, with the scalable number of servings;

- Grocery list, allowing users to create a cart of groceries that would be delivered to their home by a third-party seller.

In conclusion, eatthismuch is a useful tool for persons that want to try new things around the kitchen, especially for athletes that must respect diets for the best performance. The amount of information given about each recipe make this tool unique, with in-depth analytics about nutrients and micronutrients. This application also has a paid subscription, a meal planning service, that will connect the user with professional nutritionists ready to make a personalized and flexible menu.

### 1.4.3 Tasty

Tasty [3] is a very popular web application, following the structure of a blog. It is managed by the BuzzFeed team, which helped this site have continuous development and growth. It stands out in front of the other solutions because it is built as a blog, where users can tell their experiences and feelings about certain recipes, or just give some piece of advice for basic cooking skills to the most advanced techniques. The following characteristics are what make Tasty different from the others:

- Create your own recipe and post it, and share it with others can gain feedback from it;
- Review and star system, where users can tell their own opinion about a certain recipe, let others know if there is anything to improve, or give small pieces of advice;
- Various recipe repositories, with constant growth with the help of the community, Tasty has become one of the biggest solutions in this category. The number of meals would be wasted if not for the complex filtering system;
- Recommended recipes based on the user previous cooking history, or recommendations based on the popularity of a recipe during a week time;
- Grocery list, where each recipe is connected to a third-party seller that can provide users with the exact ingredients needed for a specific recipe;
- Tips and tricks pages, where users can add their own tricks for everyone to see and use.

Tasty is a popular web application that is happy to have the constant growth and contribution from the community that a blog can have. Another big advantage is that the application is free to use, and does not require an account in order to search for more delicious recipes.

# Chapter 2

## System Architecture

This chapter will discuss the technical implementation of the system presented. This chapter describes architectural decisions and principles applied, presenting libraries and frameworks used, in order to create the system that built the working solution. This section will be presented the following: an overview of the general architecture, backend server, frontend server, the structure used for storing data, and the security aspects.

### 2.1 General architecture

The solution is built as a full-stack web application, a client-server architecture, composed of a client application, required to handle the graphical user interface, a backend server, that processes the user requests, and the data storage system (Figure 2.1.1.). The communication between the two sides was realized through HTTP protocol, exposing a RESTful API.

The frontend server of the application sends requests using the HTTP protocol, using one of the endpoints exposed by the API. The backend side of the application receives the requests and executes the business and logic computes in order to return the required data back to the frontend server, which will present the user with the requested information.

The frontend server was created using Angular, a JavaScript framework, while the backend server was written in Java, using the SpringBoot framework, for its wide range of features and functionalities.



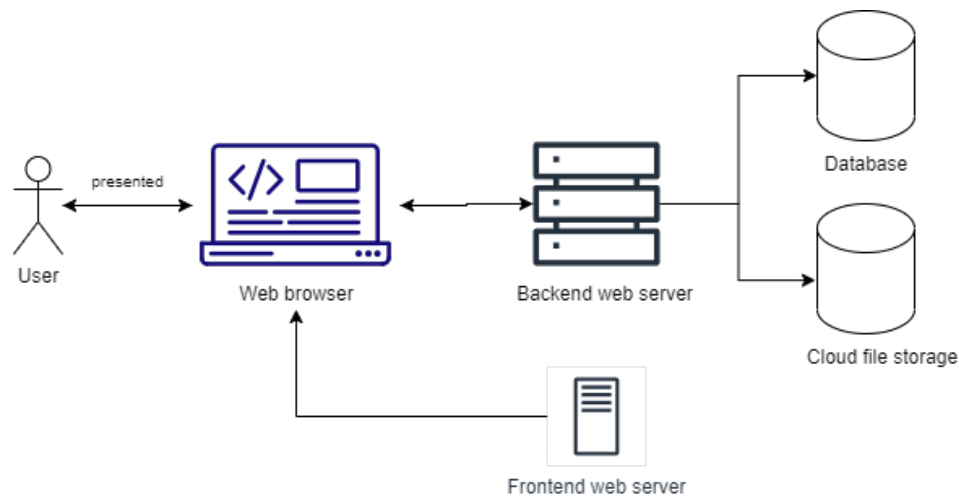


Figure 2.1.1 : General architecture diagram

## 2.2 Application backend

The backend side of the application was built in Java, taking advantage of the SpringBoot framework, being a RESTful application, that exposes an API. The solution provided is based on a microservice system, with a number of 8 entities, presented in the next image (Figure 2.2.1):

- Eureka server, which works as a discovery server. It is used in order to keep track of the services that are running, each service is required to register to the Eureka server before starting;
- API Gateway, which is used as an input point for the network system. It also works as a load balancer, thanks to the information received from the Eureka server;
- Authentication service, which is used to register and validate users' credentials. This entity is responsible for generating and validating the JWT tokens used for the authentication and authorization of the users;
- Mail service is responsible for sending emails for validation and requests. It is not exposed by the API Gateway but rather used in the internal communication between microservices;
- Storage service, which is responsible for uploading and downloading files from a storage server;

- User service, which is responsible for managing users' credentials and access permissions of users for different recipes;
- Recipe service that takes care of all recipe-related business logic, such as retrieving and modifying data. This service relies heavily on communication with the other service;
- Ingredient service, similar to the recipe service, takes care of business logic related to ingredients;

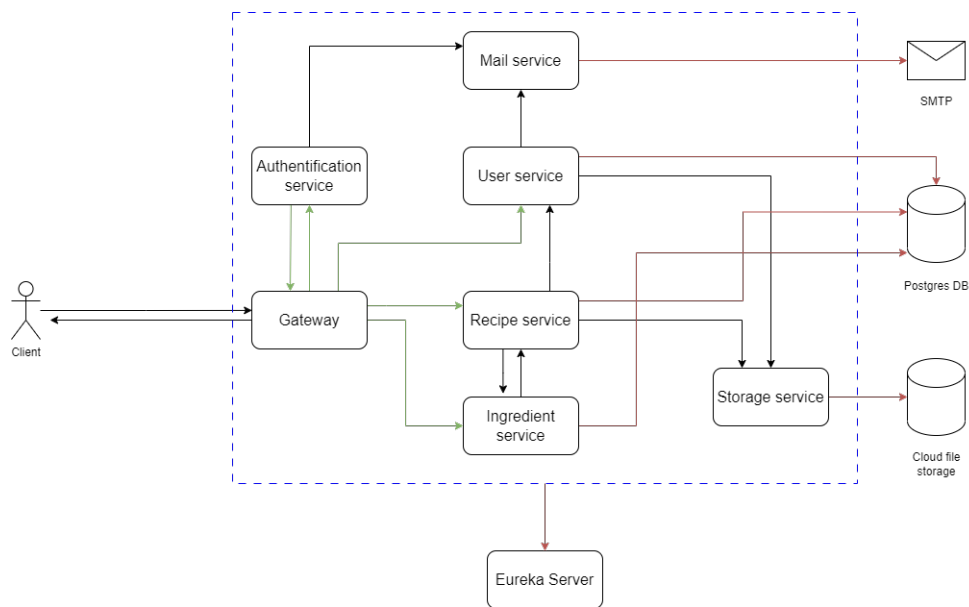


Figure 2.2.1 : Microservices diagram

All of the services created in the system follow the MVC - model view controller architecture. That is because SpringBoot allows for a good separation of concerns. The key features of this architecture are preserved in the SpringBoot framework as follows:

- Model: it contains the objects that store the data and the application business logic;
- View: it is a representation of the data that is returned to the requester. In this case, the services will end the requests with a DTO - data transfer object, in order to reduce the number of calls;
- Controller: is the component responsible for handling the users' requests. It does not contain any business logic, only determining the appropriate methods that will handle the request.

SpringBoot applications that follow the MVC architecture also have other characteristics, such as a dispatcher servlet, which works as an intermediary between the controller and the requester, as the servlet checks which is the correct controller to redirect the request to, allows users to use annotations for configuring classes. Spring is also a lightweight framework, which allows for faster development and deployment. Dependency injections help with the reusability of business code. Spring also allows users to easily generate unit tests, with JavaBeans classes, that are further injected into tests.

In order to allow the system to work properly, first every service will have to register itself to the Eureka server. After registering itself, it will receive data about the other services. Checking the Eureka server dashboard will give us information about all the registered services (Figure 2.2.2.). Periodically, the Eureka server will send back to the services the updated registry list.

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-JDL17HQ:api-gateway</a>
AUTH-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-JDL17HQ:auth-service:0</a>
INGREDIENT-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-JDL17HQ:ingredient-service:0</a>
MAIL-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-JDL17HQ:mail-service:0</a>
RECIPE-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-JDL17HQ:recipe-service:0</a>
STORAGE-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-JDL17HQ:storage-service:0</a>
USER-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-JDL17HQ:user-service:0</a>

Figure 2.2.2 : Dashboard Eureka server

The Eureka server was created using two main dependencies: the SpringBoot Starter and Spring Cloud Starter Eureka Server [6]. In order for the server to work as a Eureka server, the main method must have annotated with “@EnableEurekaServer”. Every service that will register to the server uses the same dependency, only that the annotation will be “@EnableDiscoveryClient”, with the necessary configuration that will be discussed in section 4.

The application gateway was implemented using the Spring Cloud Gateway dependency [7]. For the application to act as a gateway, some configuration must be set in order to define the routes for the services, such as the unique identifier provider by the microservice in the configuration file, the URI where the request must be redirected, the predicates used and the filters that will be verified for the request. The filter used

acts in two ways. If the request is for the authentication service, it will pass, to let the user get an access token. But if the request is for another service, the filter will verify if the header of the request contains the Bearer token. If there is no token, the request will be returned with an unauthorized code, and if the token is present, the filter will validate the token, in order to check if it is authentic or not. If the token provided is valid, the request will be redirected to the desired service.

The authentication service was created in order to process authentication and authorization using the JWT, with the help of SpringBoot Security [8]. This service takes care of the registration of users, and signing in process, where the business logic will verify if the credentials given by the users are valid and match an user account. If the credentials are valid, the user will receive a token user for authentication in order to access the other services. The service was documented using OpenAPI (Figure 2.2.3).

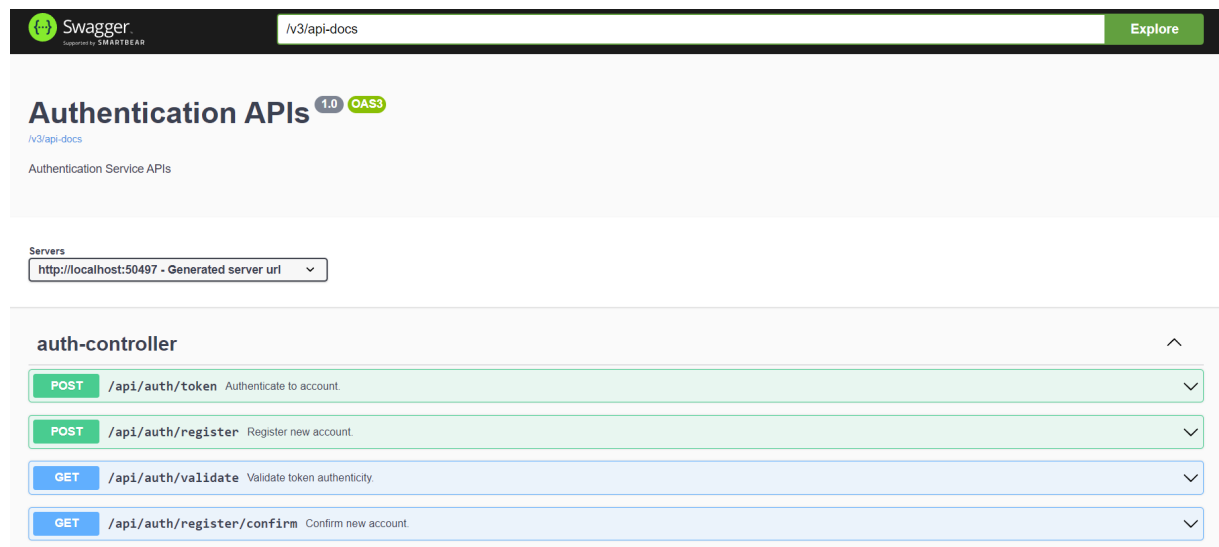


Figure 2.2.3 : Authentication service OpenAPI documentation

Mail Service was created just to send emails that were requested by the other microservices. In order to send emails, it is used the Java Mail Sender dependency [9] and the SpringBoot Mail, which allows the application to register to an email account and send emails through the SMTP. Using the JavaMailSender only requires the provider credential, such as the host, port, username, and password for the email address. Once the configurations are set, an email can be created using the JavaMailSender. As for the content of the email, it was implemented using Thymeleaf [10] as a template engine. The service was documented using OpenAPI (Figure 2.2.4).

Storage Service is responsible for managing the files located on the storage server. As the solution of choice for the storage provider was Amazon Web Services, with the

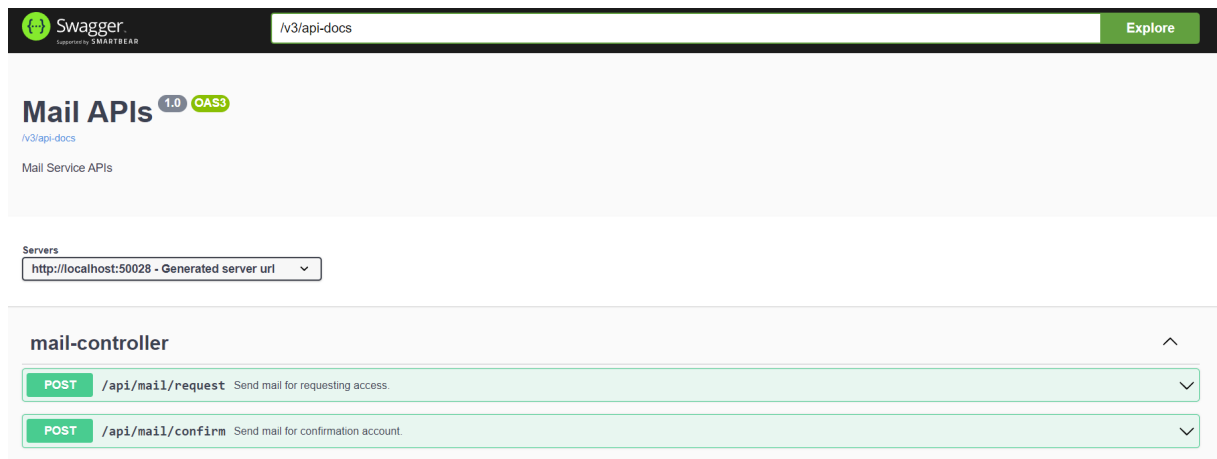


Figure 2.2.4 : Mail service OpenAPI documentation

S3 Bucket as the solution, the Java AWS SDK [10] was necessary in order to ease the work, a tool that would allow me to create an S3Client, necessary for uploading and downloading files from the online storage. Using dependency injection, an S3Client was created in the configuration, and further use it in the service class. Every file uploaded had been named with a random string, in order to remove redundancy of naming. That name would rather be saved into the database, alongside the object's other attributes. The service was documented using OpenAPI (Figure 2.2.5).

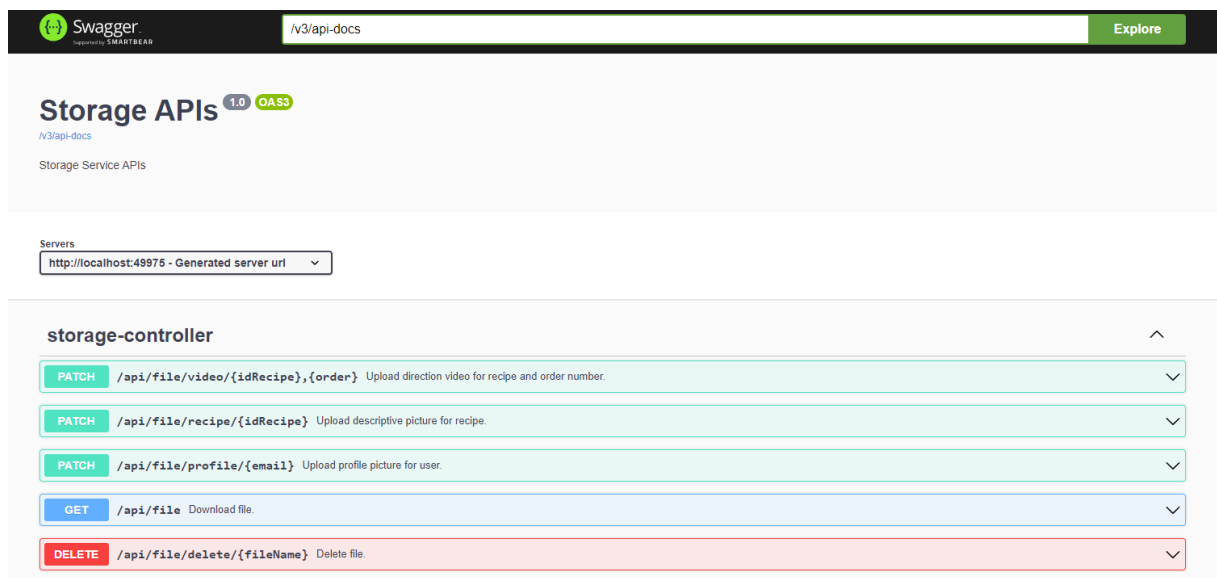


Figure 2.2.5 : Storage service OpenAPI documentation

In order to realize the inner communication between services, the SpringBoot WebClient [11] was used. In the WebService class, a WebClient.Builder object is injected, object that was configured before, specifying to opt for load balancing options. The WebClient works as an entry point that allows the performing of web requests. Using

WebClient allowed me to make requests to the other service. It integrates with the Eureka server, as in order to target a specific service, it had to be referred to it as in the registered information. For the deserialization of the response, primarily, the function “bodyToMono()” was used. All the services implemented this communication type, as it is important to separate the jobs of every service.

The recipe, ingredient, and user service are generally used services, focused on the business logic, rather than doing a specific thing, as in the case of the mail or storage services. Like the other services, they are Eureka clients, that register themselves at the beginning of their lifecycle. Each service was documented using OpenAPI, as follows: recipe service (Figures 2.2.6, 2.2.7), ingredient service (Figure 2.2.8), and user service (Figures 2.2.9, 2.2.10).

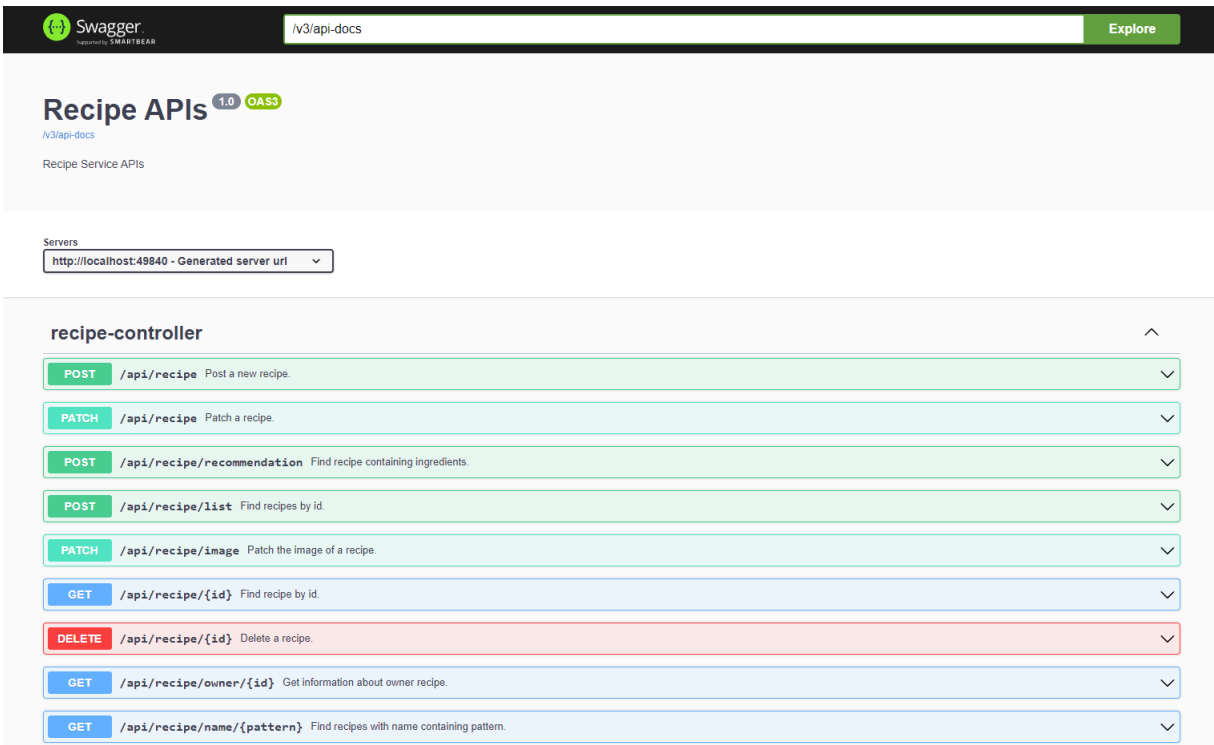


Figure 2.2.6 : Recipe service OpenAPI documentation

GET	/api/recipe/name/{pattern}	Find recipes with name containing pattern.	▼
GET	/api/recipe/for-user	Find users recipes.	▼
<b>menu-controller</b> ^			
POST	/api/menu	Post a menu.	▼
POST	/api/menu/add	Add recipe to menu.	▼
GET	/api/menu/{idMenu}	Get menu information.	▼
GET	/api/menu/owned	Find menus of owner.	▼
DELETE	/api/menu/remove	Remove recipe from menu.	▼
<b>category-controller</b> ^			
POST	/api/category/complete-menu	Add recommended recipes to menu.	▼
GET	/api/category/{category},{offset}	Get recipes by category and page number.	▼
<b>direction-controller</b> ^			
PATCH	/api/direction/video	Patch video instruction for direction.	▼

Figure 2.2.7 : Recipe service OpenAPI documentation


<div>  <span>Swagger</span> </div> <div> <input type="text" value="/v3/api-docs"/> <span>Explore</span> </div>			
<b>Ingredient APIs</b> <span>1.0</span> <span>OAS3</span> <small>/v3/api-docs</small> Ingredient Service APIs			
Servers <input type="text" value="http://localhost:50541 - Generated server url"/>			
<b>ingredient-controller</b> ^			
POST	/api/ingredient/recommendation	Get recommendation for ingredients.	▼
POST	/api/ingredient/from-recipe/list	Find ingredients for recipes.	▼
POST	/api/ingredient/add/{idRecipe}	Add ingredients to recipe.	▼
POST	/api/ingredient/add-price	Add price for ingredient.	▼
PATCH	/api/ingredient/update-price	Patch price for ingredient.	▼
GET	/api/ingredient/{pattern}	Find ingredients by pattern.	▼
GET	/api/ingredient/list-prices	Find prices of user.	▼
GET	/api/ingredient/from-recipe/{id}	Find ingredients in recipe.	▼
DELETE	/api/ingredient/delete-price/{id}	Delete price for ingredient.	▼

Figure 2.2.8 : Ingredient service OpenAPI documentation

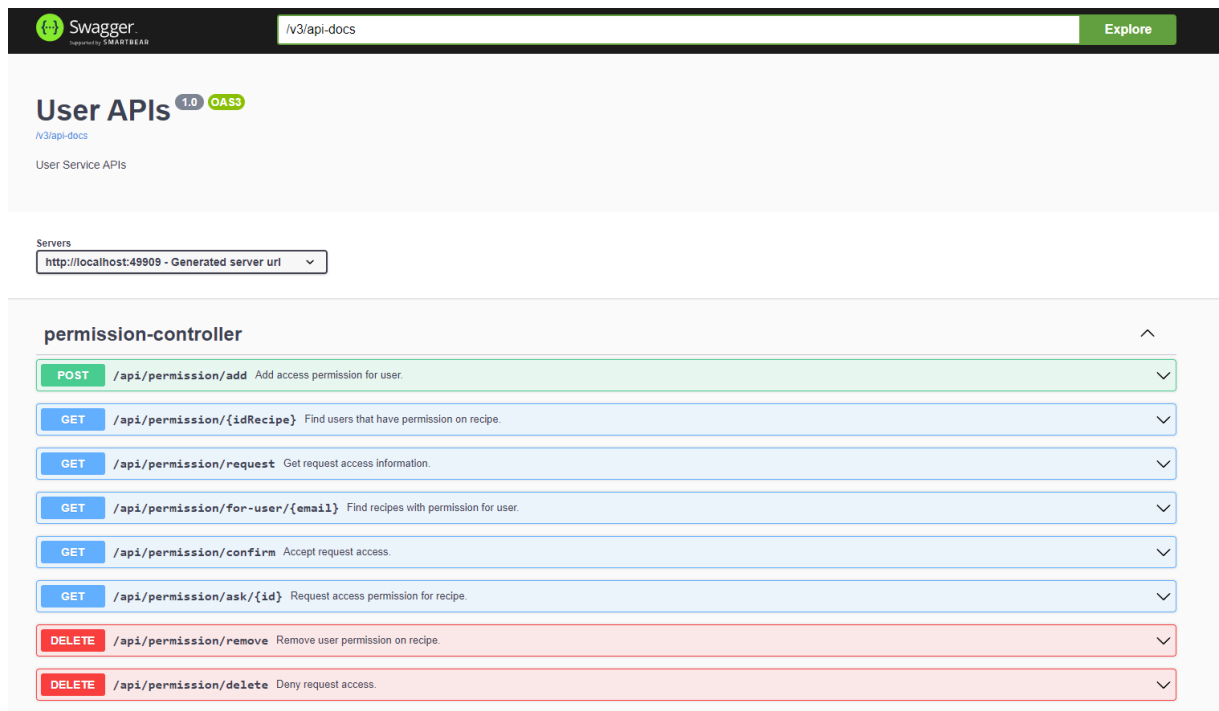


Figure 2.2.9 : User service OpenAPI documentation

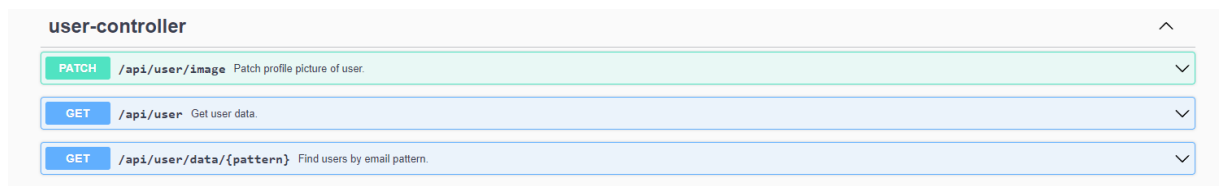


Figure 2.2.10 : User service OpenAPI documentation



## 2.3 Application frontend

The frontend server of the client application was created using the Angular framework, which runs on a NodeJS server, which delivers the client application, a popular single-page application, as every page exists in one HTML file, thanks to modularisation. This web application follows the Model-View-Controller architecture, as application data are managed by the models, and the views are represented by the display of the data. The controller is the part that connects the services that provide the models with the views. Also, Angular is a less code framework, that helps the process of linking the MVC layers, doing this automatically.

Another great feature of the Angular framework is the two-way data binding, which is achieved through the sharing of the same data between the view and the component class. If data will be changed in one component, this will be reflected in the other one.

Dependency injection is one of the fundamental features of this framework. Dependency injection is a design pattern, in which a class uses dependency from outside sources, rather than creating them. In Angular, this is used primarily for the use of services, in order to fasten the process of testing, overriding, and development [13].

As modularization is one of the key features of the framework, in this project used them intensively in the development process. A module is a container that holds components, services, and other pieces of code that are specific to a workflow. My application is divided into five main modules: HomeModule, AuthenticationModule, RecipeModule, MenuModule, and PageNotFoundModule. The first four modules are protected using the “CanActivate” route guard, which allows the user to access them only if is provided a valid authentication token. All modules are children from the main module of the application, the AppModule, with each module exposing other modules or components.

Alongside the exposed modules, two shared modules were created: DialogModule and TopBarModule, which contain shared components for the other components. In order to use the shared components in other modules, the component had to be grouped into modules and imported, in order to eliminate cyclic imports. Having a shared module that stores components reduce redundant code.

In order to make requests to the backend server of the application, the built-in the HttpClient service was used, which is based on the XMLHttpRequest. The HttpClient

calls return an Observable object, that we have to subscribe in order to receive data from the server [25]. An Observable is an object that helps with the asynchronous code execution. When we subscribe to an Observable, the request to the desired endpoint will be initialized. If the Observable is not subscribed, nothing will happen. In order to stop receiving data from the Observable, we have to unsubscribe, removing the possibility of memory leaks. The HTTP request accepts multiple parameters, the one that was preferred the most was by providing the URL, the body of the request, if needed, and for the last parameter the headers options. The header options are optional, as HttpClient can add them automatically.

Thanks to the HttpInterceptor interface, the behavior of the web application could be modified while the requests were in progress. The interface intercepts every request made, allowing me to change the information from the Observable provided by the LoaderService, which will display a loading spinner or not, based on the information provided by the Observable.

In order to reduce redundant code for requesting data from the backend server, services were created to take care of the logic. Services are grouped based on the module they are located in, or the use case they have. Services are injected into classes. Every service works as an intermediary between the class from which the call was made, and the main API service, which contains all the requests to the server.

When using the HttpClient, if the call was successful, it will return a JSON object. In order to remove redundant work, every response from the backend server is deserialized accordingly into models. This is a commonly used solution in order to not work with raw JSON objects, which can be a hard task when talking about big data structures. The models used are all declared in a shared component, from which every other component can use them.

When requests made through HttpClient happen to fail, an error is always thrown. As such, it is important to have a valid error-handling system. For every request made, a pipe operator was added, which will make a copy of that Observable, without changing the context. Inside the pipe operator, we will have at dispose of the original Observable. Such, we can make a preliminary request to the backend server, in order to check if the request will be executed successfully or not. The error handler will not intervene if the call was made successfully, but if an error is produced, it will be checked and treated accordingly. For example, if the user requests data about a recipe, it is possible that the recipe to not exist, and the backend server will return a 404 error with

the appropriate error details. In this case, the user will be redirected by the error handler to the not found page, and an informative message will be given to the user. It is important to treat errors when they accrue, in order to reduce further errors from happening.

For UI components, the Angular Material library was the chosen library, developed by Google [14]. This library was chosen because it has good integration with other Angular libraries, reducing the possibility of conflicts. It provides pre-built components, from which the most used components are the following: input, autocomplete panels, table, select, and others. The library comes with three pre-built color schemes but allows the developers to create their own theme. The color theme that was created was inspired by the color scheme of the application, with the orange shade as the primary color, and the black as the accent color.

Another great library that helped me in the process of development was the `ngrx-toastr` [15]. This library offers the developers the option to use a prebuild and customizable notification popup. In the application module, where the `ToastrModule` was imported, being able to override some predefined settings of the `toastr`, such as the position and some behavioral options. This library was used in order to provide the user with information about the state of requests made to the backend server, if any error accrued, or if the request was successful.

## 2.4 Data storage

For the data storage, was used a PostgreSQL database server. The database server that is used is not locally hosted mainly because that would require more configuration on my part, and so, choosing a cloud hosting provider would guarantee a more qualitative experience. Having a dedicated hosting server for the database would guarantee automatic backups, more security options such as encrypted data transfer, and server downtime are immediately resolved, as the server would be replaced with a new node, and the option to scale the hardware based on requirements.

The schema used (Figure 2.4.1.) consists of 12 tables, each table having a unique identifier, generated using a numerical sequence starting from 1. The foreign keys in the tables make the connection between them, even if the referenced key is an identifier or other attribute. Upon update or deletion of the referenced key, the referencing key will cascade in order to maintain the relations and consistency.

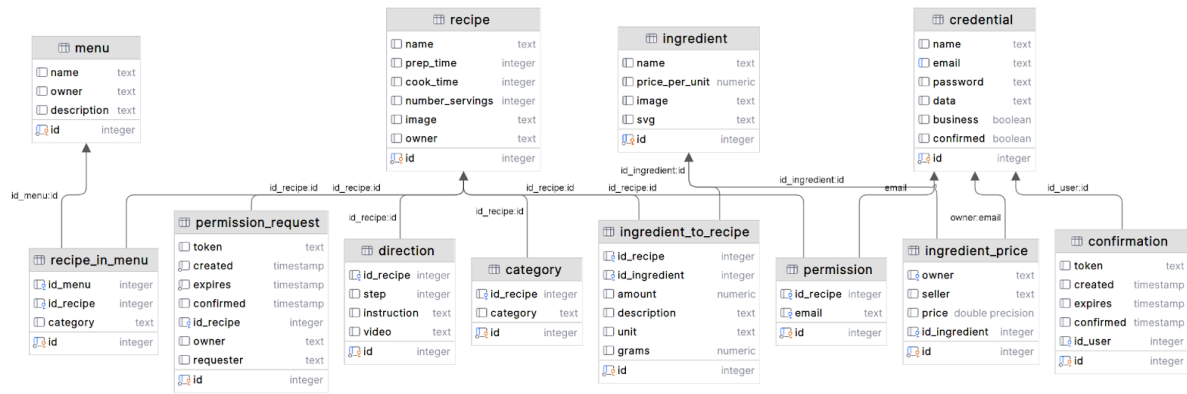


Figure 2.4.1 : Database relationships between tables

In the presented schema we can distinguish two types of relationships: one-to-many and many-to-many. For the one-to-many relationship, we have the following table structuring:

- “direction” table, where “id\_recipe” column references the “id” from “recipe” table, in order to know what are the instructions for a specific recipe;
- “category” table, where “id\_recipe” column references the “id” from “recipe” table, in order to catalog the recipe after their categories;
- “confirmation” table, where “id\_user” column references the “id” from “credential” table, to log the moment when a new account has been validated.

For the many-to-many relationships, we have the following table structuring:

- “recipe\_in\_menu” table, where “id\_menu” references the “id” from “menu” table, while the “id\_recipe” references the “id” from “recipe” table, where is reference can be found in multiple rows;
- “ingredient\_to\_recipe” table, where “id\_recipe” references the “id” from “recipe” table, while “id\_ingredient” references “id” from “ingredient” table, making the relation between recipes and ingredient used;
- “ingredient\_price” table, as “id\_ingredient” references the “id” of “ingredient”, while the “owner” references the “email” from credentials, storing informations about each deal a user has for a specific ingredient;
- “permission” table, where “id\_recipe” references the “id” from “recipe” table, and the “email” references the “email” from “credential” table.

Preliminary, “recipe”, “ingredient”, “direction” and “ingredient\_to\_recipe” tables have been populated using a public API provided by [21, 22, 23].

For the file storage, the best solution was a dedicated cloud storage solution from Amazon Web Services, the S3 Bucket. This option was chosen because it has many facilities, such as scalability, security, and performance. The cloud storing option was better than the local storage because it removes redundant work for maintaining the files, it also benefits microservice architecture, because it allows for better decoupling of the system from the local storage. The storage provider allows us to use a ready-to-use pre-configured storage server. The storage is used in order to keep images for users’ profile pictures, video instructions for specific steps, or the representative image of the final product.

While uploading the file to the cloud, each file gets a unique identifier as a name. For example, for the video instructions, in the column “video”, save that unique identifier, in order to have a clear location of the file.

The upload and download process is realized using the AWS Java SDK [16], each operation requires a valid AWS account, that has S3 Bucket service available. The files can be made publicly or not, depending on the use case.

## 2.5 Security aspects

Securing the system was done using a JWT (JSON Web Token) implementation, used for authentication and authorization of the user. This securing method uses a base 64 encoded token, using the hash message authentication code, that takes in a hash function and a secret key.

The JWT is composed of three components, each storing different information: header, which contains the algorithm used for encryption, and the type of the token, payload, containing the claims, and other optional data about the user [17]. Typically, in the payload of a JWT token, it is stored information such as the issuer, expiration date, and the subject, refers to the entity to which the token was created. The final part is the signature, which is used to validate the token using the secret key.

In order to secure the backend side of the application, the Spring Web Security framework was used in order to secure the application. The application gateway acts as the main layer of security. Every request made through the gateway is filtered in order to check if the request was made by an authorized entity. In the configuration

files, routes toward the authentication service are all allowed, in order to let the user authenticate.

In the authentication service, a user can request to authenticate, by providing his credentials. If the provided credentials are valid and match a user account, the service will return a signed token, valid for 30 minutes, a token that can be used in order to request further data from the system.

But if the requested route is not in the authentication service, the next filter of the gateway will run. It will check if the header of the request contains a token, stored in the “Bearer” schema. If there is no such token, the request will be denied, returning the status of unauthorized. If there is a token, the filter will validate the token using the same secret key used while creating the token, checking if the third part of the token is valid, by encrypting the payload using the algorithm from the header, while using the secret key. If the token is valid, the filter will be finalized, and the request will be redirected toward the desired service.

For securing the frontend server of the application, has been used the “CanActivate” interface, which works as a guard for specified routes. The class which implements the interface must implement the “canActivate” method. Routes are protected through JWT token validations. First, the service will verify if the token has expired or not, by decoding the payload. If the token is not present or is invalid, the user will be redirected to the authentication component, despite accessing different routes.

The JWT token provided by the backend server is stored in the local storage of the application. For every backend request, the bearer token will be automatically written into the header thanks to the “HttpInterceptor” interface, which allows us to intercept every request and modify its headers.

Even though my application uses authentication and authorization of the user, there is still room for vulnerabilities and attacks from malicious users. Injection attacks known as SQL injections are a common type of attack, that can happen when the inputs are not sanitized corresponding, allowing the attackers to insert malicious SQL in order to receive unauthorized data. One way to avoid SQL injection is to use SpringBoot Data JPA Repository, or to sanitize the input accordingly.

XSS attacks are commonly used for web servers, as the attackers try to input malicious scripts in the input boxes, scripts that will be executed automatically, in order to receive data. Angular framework use as a defense mechanism the “contextual escaping” in order to not interpret the input as HTML files [21]. As long as we do not

intervene with security models implemented by Angular, in order to minimize the possible attacks of XSS.

# Chapter 3

## Use cases

The presented solutions come with solutions for specific use cases, that were studied and analyzed before. This section contains descriptions for those use cases, that follow the diagram from the Figure 3.1. The possible use cases mainly target the administrators of restaurants, focusing on reducing the time wasted on repetitive tasks and helping new employee complete their training in a modern, fast-paced environment.

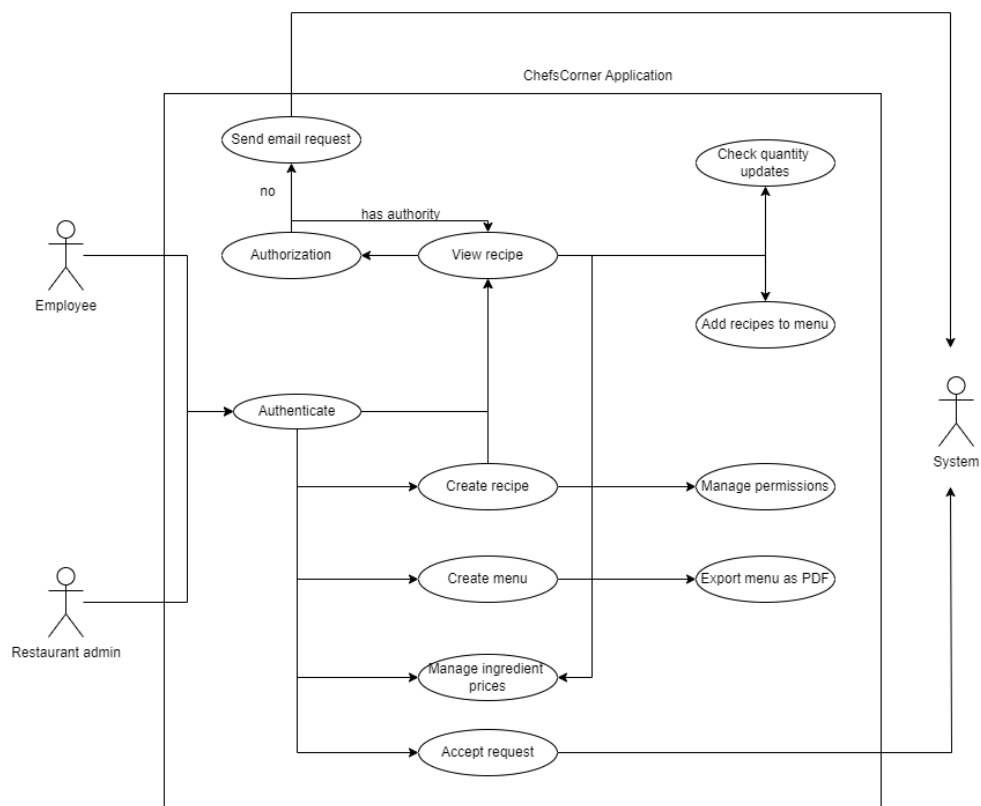


Figure 3.1 : Use case diagram



### 3.1 Posting a new recipe

One of the most essential things in a restaurant is to store recipes, keeping everything together. With this issue in mind, the solution came with the option of posting recipes. In order to create new recipes, the user has to authenticate himself. After the authentication process is done, from the menu drawer, he can select the “Create recipe” option (Figure 3.1). This will take the user to the process of creating a new recipe.

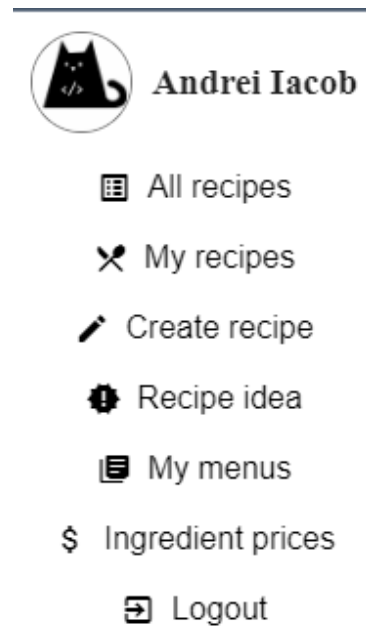


Figure 3.1.1 : Selecting Drawer options

From there, the user has to complete the required data (Figure 3.1.2): name, description, category of the recipe, cooking and preparation time, and add a photo of the final product.

Following the recipe details, the user has to select the required ingredients for the recipe (Figure 3.1.3). Here, the user can search for predefined ingredients, or create new ones, by simply writing a new name.

Last, the user has to introduce information about the preparation steps (Figure 3.1.4). The minimum number of steps is one. Alongside text steps, the user has to option to add video instructions. In the end, the user has to confirm the imputed data, so that the process of posting a new recipe would be complete.

Recipe name\*

Clatite cu nutella

Categories\*

Desserts

Prep time (minutes)\*

10

Cook time (minutes)\*

15

Number of servings\*

10

Clatite-cu-Nutella-si-banane-151332662.jpg


Figure 3.1.2 : Recipe details

Search ingredients by name*	Description*	Amount*	Unit of measurement*	Grams per unit*
Wheat Flour	new	10	tsb	20
Search ingredients by name*	Description*	Amount*	Unit of measurement*	Grams per unit*
Eggs	new	2	piece	50
Search ingredients by name*	Description*	Amount*	Unit of measurement*	Grams per unit*
Dark Chocolate	good	1	tablet	150
Search ingredients by name*	Description*	Amount*	Unit of measurement*	Grams per unit*
		1		1

Figure 3.1.3 : Ingredients required for recipe

## 3.2 Price estimation for preparing a meal

One important task before setting the prices for meals would be to check an estimated price for preparing that recipe. To do so, the user has to access the page of any recipe. In the recipe, panel, under the title, are four quick-access buttons (Figure 3.2.1). By clicking the first button, the user will be shown a popup (Figure 3.2.2), where the user would get access to some predefined prices for the ingredients. Here, the user will have the option to change the currency in use, by selecting another value from the first dropdown. The computed prices for each ingredient use the preselected quantities from the recipe page. As every quantity is editable, the user will have a realistic view of the prices of ingredients. Furthermore, the user will have the option to add their own prices for ingredients.

Instruction\*

Mix the flour with

⊖

WIN\_20230609\_18\_04\_59\_Pro.mp4

📎

Instruction\*

⊖

No file uploaded yet.

📎

Figure 3.1.4 : Add instructions for recipe

Alongside the prices of the ingredients, the user is given a recommended selling price for the recipe, based on the quantity of the ingredients used, and also offers a This use-case is very useful for any business, helping them keep track of ingredient prices and making realistic evaluation upon a good selling value.

#### Estimated prices for ingredients:

Currency

RON

▼

Ketchup:

1 \* 2.73 = 2.73 RON

Seller	Price/Unit	Action
Metro	RON 5	⊖
Auchan	RON 4.5	⊖
new seller	RON new price/unit	✎

Beef Plate:

20 \* 1.94 = 38.83 RON

Total: 41.56 RON

Total with profit: 72.72 RON

%

75%

▼

Close

Figure 3.2.2 : Ingredient prices popup

# Clatite cu nutella

my recipes

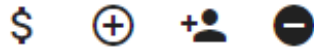


Figure 3.2.1 : Quick access menu on recipe page

## 3.3 Managing all ingredients prices

Similarly to the previous use case, the admin of a business is also able to keep track of all the ingredients prices he has added throughout the recipes. In order to access this component, from the menu drawer (Figure 3.1.1), the “Ingredient prices” option has to be selected.

There, the user will see all the ingredients that contain deals from different shops (Figure 3.3.1). As an administrator, periodically he could modify these prices, based on previous receipts. Here, he could check every ingredient used, update deals, or delete some. If any new ingredients are needed, he can search for them using the search box.

Currency  
RON

Seller	Price/Unit	Action
Kaufland	RON 27.82	
Megalmage	RON 10.02	
new seller	RON new price/unit	

Crispy Corn Puffs Cereal:

Almond Milk:

Ketchup:

Chicken Breast:

Potato:

Search ingredients by name

Figure 3.3.1 : Manage prices on ingredients

### 3.4 Managing access permissions over recipe

As an administrator, another task that would be required is to manage the access permissions of the employees over the restaurant recipes. In the recipe page, from the quick access panel (Figure 3.2.1), by accessing the third icon, the admin will access the permission popup (Figure 3.4.1). Here, the user has two options: add new permissions or remove others.

Currency  
RON

Crispy Corn Puffs Cereal:

Seller	Price/Unit	Action
Kaufland	RON 27.82	⊖
Megalmage	RON 10.02	⊖ ✎
new seller	RON new price/unit	✎

Almond Milk:

Ketchup:

Chicken Breast:

Potato:

Search ingredients by name

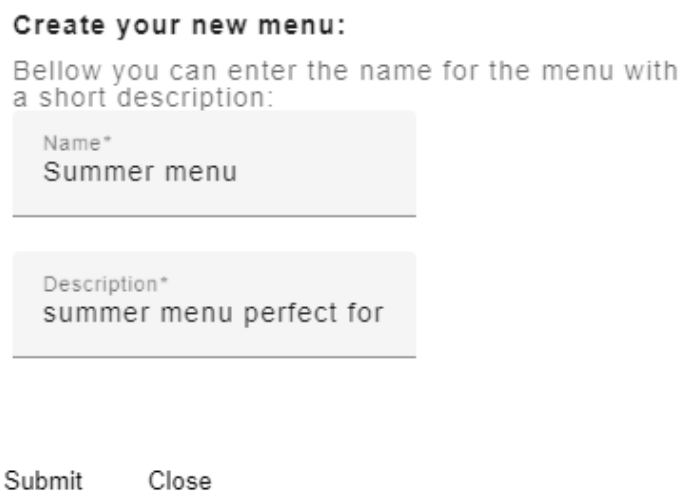
Figure 3.3.1 : Recipe permission panel

For example, if an employee leaves the team, he would not need access to the recipe anymore, so, using the delete button from the right of his email, the permission would be removed.

In the other way, if a new member joins the team, his email can be searched in the search box, in order to grant him permission to access the recipe and begin his training as a new employee.

## 3.5 Creating and populating menus

From time to time, restaurants change their menu, which can be an irritating task, modifying the pre-existing ones, and changing manually all the prices. But, this system comes with a solution for this use case. The person responsible for changing the real-use menu can create a new menu in the application (Figure 3.5.1), by providing a title and a suitable description. From there, any recipe can be added to the menu, using the quick access (Figure 3.2.1) inside a recipe.



**Create your new menu:**

Bellow you can enter the name for the menu with a short description:

Name\*  
Summer menu

Description\*  
summer menu perfect for

Submit    Close

Figure 3.5.1 : Creating a new menu

Clicking the icon will open a popup (Figure 3.5.3), in which the user has to select the desired menu and the category in which the recipe will be placed. Finally, clicking the “Submit” button will complete the process of adding the recipe to the menu.

As most businesses prefer to have printed menus, this is possible due to the printing feature. While on the menu page, you can click the “Generate PDF” (Figure 3.5.2) button, which will generate a PDF based on the recipes on the menu. Recipes will be classified by their respective category. The price offered on the menu will be a recommendation based on an estimated value of every ingredient used, plus an editable in-menu page add-on rate, that would guarantee the restaurant a profitable trade. The currency of the menu can be chosen between RON, EURO, and USD, with conversion based on the latest rates. Ingredients used for every recipe are shown, along with the quantity of each component.

### Add this recipe to your menu:

Bellow you can select the menu in which you can add this recipe:

Menu  
Summer menu ▼

Category  
MostlyMeat ▼

Add

Close

Figure 3.5.3 : Adding a recipe to the menu

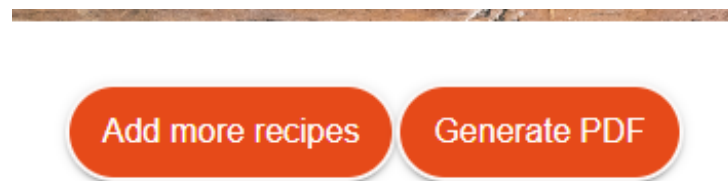


Figure 3.5.2 : Functions button in the menu page

## 3.6 Using the application as a new employee for training purposes

As a new employee, it is crucial to understand and learn how each recipe is made in a professional kitchen. This process can be sped up by following the specially made video instructions, but in order to watch them, the new employee needs access to the recipe. One way to gain access was discussed in section 3.4, but another way would be to access the recipe directly. There, the user would be prompted with a popup that would ask the user if he would like to request access to the recipe (Figure 3.6.1). By selecting “Yes” as the option, the recipe admin would get an email request.

In this way, the access permission can be given faster, without any confusion. As the user gains access, where he could study each detail from the steps instructions and the demonstrative videos.

---

**Forbidden access**

Currently you do not have access to this recipe.

Would you like to notify the owner and ask for permission?

No

Yes

---

Figure 3.6.1 : Request access to recipe



# Conclusions and Future Improvements

The key to success and growth in the culinary industry is based on digitalization, providing an efficient way of managing recipes and costs, and also the training process. Using the proposed platform helps to solve the discussed problem. Nowadays, the way to grow a business is based on the development of the individual at the workplace

As with any other application, there is always room for improvements and future features. Further, we will be present some features that would improve the current implementation.

One way in which users can be confused while updating the prices of ingredients would be to have multiple sellers with similar names, a situation that can happen to anyone due to a typo. In order to prevent that from happening, predefined options for sellers would be the perfect solution. But as a starting solution that would not be possible, due to low data income. As time gets by, the system would have gained enough data in order to present the user with default options for sellers.

Another great feature that would help the platform would be the option to add ingredients needed into a cart, choosing the required amount for every ingredient. Each ingredient should be available on an online shopping platform. Finalizing a cart in the ChefsCorner platform should redirect the user to the third-party seller, where a pre-filled shopping cart would be created, with the selected ingredients.

Momentary, when exporting a menu as PDF in order to print it, the user can not decide how the items are aligned and stylized. In the future, when clicking the “Export as PDF” button from the menu page, the user would be presented with options for the template used. Furthermore, the user would be able to select styling options, such as the used font or the font size.

In conclusion, this thesis integrates harmoniously the idea of facile and efficient management of recipes and training process of the employees, helping to significantly reduce business expenses through the optimal and up-to-date solution provided by the

proposed application. The developed platform offers an important advantage for businesses from the culinary industry, as it simplifies the usual tasks through the structured guidance of the employees.

# Bibliography

1. GetMeez, *About Us*, 2023, URL:  
<https://www.getmeez.com/about-us>
2. Eatthismuch, *About Us*, 2023, URL:  
<https://www.eatthismuch.com/about/>
3. Tasty, 2023, URL:  
<https://tasty.co/>
4. MD Khokon, *CodePen - Email Confirmation HTML Template*, URL:  
<https://codepen.io/md-khokon/pen/bPLqzV>
5. FreecurrencyAPI, *Guide*, 2023, URL:  
<https://freecurrencyapi.com/docs/#official-libraries>
6. Spring Cloud Netflix, 2023, URL:  
<https://cloud.spring.io/spring-cloud-netflix/reference/html/>
7. Spring Cloud Gateway, 2023, URL:  
<https://spring.io/projects/spring-cloud-gateway>
8. Spring Security, 2023, URL:  
<https://docs.spring.io/spring-security/reference/getting-spring-security.html>
9. Thymeleaf, 2023, URL:  
<https://www.thymeleaf.org/>
10. Baeldung, *Spring Email*, 2021,  
<https://www.baeldung.com/spring-email>

11. Baeldung, *Spring WebClient*, 2022,  
<https://www.baeldung.com/spring-5-webclient>
12. Spring Data JPA, 2023,  
<https://spring.io/projects/spring-data-jpa>
13. Ranjeeta Borah, *Angular Features: What It Brings to Us?*, 2023,  
<https://www.clariontech.com/blog/angular-features-what-it-brings-to-us>
14. Google, *Angular Material*, 2023,  
<https://material.angular.io/>
15. Scttcper, *ngx-toastr*, 2023,  
<https://www.npmjs.com/package/ngx-toastr>
16. AWS, *AWS Java SDK*, 2023,  
<https://github.com/aws/aws-sdk-java-v2/#using-the-sdk>
17. JWT, *Introduction*, 2023,  
<https://jwt.io/introduction>
18. Oracle, *Java*, 2023,  
<https://www.oracle.com/java/technologies/downloads/>
19. Apache, *Maven*, 2023,  
<https://maven.apache.org/download.cgi>
20. NodeJS, 2023,  
<https://nodejs.org/en/download>
21. StackHawk, *Angular XSS Guide*, 2021,  
<https://www.stackhawk.com/blog/angular-xss-guide-examples-and-prevention/>
22. eatshismuch, *Recipe API*, 2023,  
<https://www.eatthismuch.com/api/v1/recipe/>
23. eatshismuch, *Ingredients API*, 2023,  
<https://www.eatthismuch.com/api/v1/ingredients/>

24. eatshismuch, *Directions API*, 2023,  
<https://www.eatthismuch.com/api/v1/directions/>
25. Jeetendra Gund, *Angular Basics: How To Use HttpClient in Angular*, 2023,  
<https://www.telerik.com/blogs/angular-basics-how-to-use-httpclient>

# Appendix 1. Application deployment

## Backend server

The backend server of the application was created using Java. In order to run the backend application, you need to install JDK [18]. The application was written using JDK 19, so any new version will be compatible. After installing the developer kit, you will need the Maven package manager [19].

Before running each microservice, you must set constant variables in the “application.properties” of each module:

- API Gateway:
  - “secret.key” - must contain a 256-bit secret key written in hex;
- Authentication service:
  - “secret.key” - must contain a 256-bit secret key written in hex, the same key used in the API Gateway;
  - “spring.datasource.password = username”, “spring.datasource.password = mypassword”, “spring.datasource.password = url” - credentials of the database that respect the schema presented in chapter 2.4;
- Recipe, Ingredient, and User service:
  - “spring.datasource.password = username”, “spring.datasource.password = mypassword”, “spring.datasource.password = url” - credentials of the database that respect the schema presented in chapter 2.4;

- Mail service:
  - “spring.mail.username” and “spring.mail.password” - credentials of the mail address used;
  - “spring.mail.host=smtp.gmail.com” - the host provider can be changed based on usage.
- Storage service:
  - “aws.data.accessKey” and “aws.data.accessSecret” - credential to the AWS account, can be changed based on the service provider;
  - “aws.data.region” - an optional, region of the bucket server location;
  - “aws.data.bucketname” - the name of the bucket used for storing files.

After all the variables have been set, the microservices can be started. It is important to start the eureka server first, in order to allow the other services to register, as discussed in chapter 2.

## Frontend server

The frontend server of the application was created using the Angular framework. First, you have to install the NodeJS [20] runtime environment. After the installation of NodeJS, you will have to install the Angular CLI, by running the following command:

```
npm install -g @angular/cli@15.1.0
```

Before running the frontend server, you must set environmental parameters. In the “src/environments/environment.ts” configuration file (Figure 4.2.1.), the following constants must be modified:

- keyCurrencyAPI: a valid key for the currency API [4] must be provided;
- apiBaseUrl: the URL of the backend server gateway;
- storageUrl: the URL to the bucket used for file storage.

After all the constants have been modified, in order to run the frontend server, you have to run the following commands:

```
npm install
npm start
```

```
const key = {
  keyCurrencyAPI: 'key',
}

export const environment = {
  production: false,
  apiBaseUrl: 'http://localhost:8080',
  imageUrl: 'https://images.eatthismuch.com/',
  storageUrl: 'https://chefs-corner-data.s3.eu-central-1.amazonaws.com/',
  currencyAPI: `https://api.freecurrencyapi.com/v1/latest?apikey=${key.keyCurrencyAPI}&currencies=EUR%20USD%20RON`,
};
```

Figure A1.1 : environment.ts configuration file