

ПРОГРАМИРАНЕ В ИНТЕРНЕТ С PHP И MYSQL

# ТЕХНИЧЕСКА ДОКУМЕНТАЦИЯ НА УЕБ ПРИЛОЖЕНИЕ “MEALISE”

Разработен от **Айхан Х. Хасанов** като проект по дисциплината “Програмиране в интернет с PHP и MySQL” по специалността „Софтуерни технологии и дизайн“ – 2. курс към ФМИ, Пловдивски университет „Паисий Хилендарски“.

Имейл за обратна връзка: [ayhan.hsn.01@gmail.com](mailto:ayhan.hsn.01@gmail.com) или **stu2301681018@uni-plovdiv.bg**

Факултетен номер: **2301681018**

## СЪДЪРЖАНИЕ:

<b>1. Увод .....</b>	<b>3</b>
1.1. Въведение.....	3
<b>2. Проектиране на приложението .....</b>	<b>3</b>
2.1. Предметна област .....	3
2.2. Лични мотиви за избора на предметната област .....	4
2.3. Функционалности.....	4
2.4. Wireframe. Визуална структура и дизайн.....	4
2.5. Описание на страниците .....	6
2.6. Архитектура на приложението.....	6
2.7. Структура и логика на приложението .....	7
2.8. База от данни.....	8
<b>3. Реализация .....</b>	<b>9</b>
3.1. Използвани технологии .....	9
3.2. Функционалност: Автентикация.....	9
3.3. Функционалност: Генериране на рецепти.....	11
3.4. Функционалност: Търсене на съществуващи рецепти чрез Spoonacular API.....	13
3.5. Функционалност: Съхранение и управление на рецепти. ....	14
3.6. Екрани.....	18
3.7. Примерни стъпки за генериране на рецепта.....	21
3.8. Инструкции за инсталация .....	22
<b>4. Заключение .....</b>	<b>23</b>
<b>5. Използвани източници. ....</b>	<b>24</b>
5.1. Използване на генеративен изкуствен интелект.....	24
5.2. Документация .....	24

# 1. Увод

## 1.1. Въведение

### 1.1.a. Проблем

В съвременното ни ежедневие все повече хора водят забързан начин на живот, рядко намирайки време за приготвяне на храна в домашни условия, което от своя страна е предпоставка за нездравословно хранене.

### 1.1.b. Основа цел

Основната цел на Mealise е да реши точно този проблем. Бидейки уеб приложение, Mealise цели да улесни потребителите в намирането и създаването на кулинарни рецепти, които отговарят на техните предпочитания и налични съставки. Това решение използва Flask — лек Python-базиран уеб фреймуърк — за изграждане на модулна, мащабируема и лесна за поддръжка уеб платформа. Системата работи с локална бази от данни с външни API интеграции.

### 1.1.c. Подобни системи и вдъхновение

Съществуват множество подобни приложения в онлайн пространството, предлагащи рецепти и съвети за хранене. Някои такива приложения, които са били източник на вдъхновение за този проект, са:

- [Supercook.com](https://www.supercook.com/)
- [Realfood.tesco.com](https://www.realfood.tesco.com/)

*Следващите раздели от документа ще разгледат подробно проектирането на приложението, включително функционалностите, архитектурата и структурата на базата данни. След това ще бъдат представени детайли за използваните технологии и етапите на реализация, придружени с примери на код и екранни снимки.*

## 2. Проектиране на приложението

Процесът на проектиране на приложението включва няколко ключови етапа, които гарантират неговата функционалност, използваемост и разширяемост.

### 2.1. Предметна област

Приложението е фокусирано върху генерирането на персонализирани рецепти с помощта на GenAI. Предметната област, както е споменато по-горе, обхваща кулинарията и предоставянето на потребителите на възможност да откриват нови и интересни рецепти, съобразени с техните предпочитания и налични съставки.

## 2.2. Лични мотиви за избора на предметната област

Като човек с интерес към готвенето, исках да обвържа разработката на уеб приложението с тема, която ме вълнува и в която имам личен опит. Това, от своя страна, ме мотивира да реализирам този проект. Темата за рецепти, хранене и кулинарни идеи е не само близка до мен, но и изключително актуална и широко достъпна. Поради тази причина избрах да създам приложение, което да служи като помощник в кухнята – място, където потребителите могат да откриват рецепти.

Освен това, исках да експериментирам с модерни технологии и да интегрирам елементи на Generative AI, които да обогатят функционалността на приложението.

## 2.3. Функционалности

Основните функционалности на приложението включват:

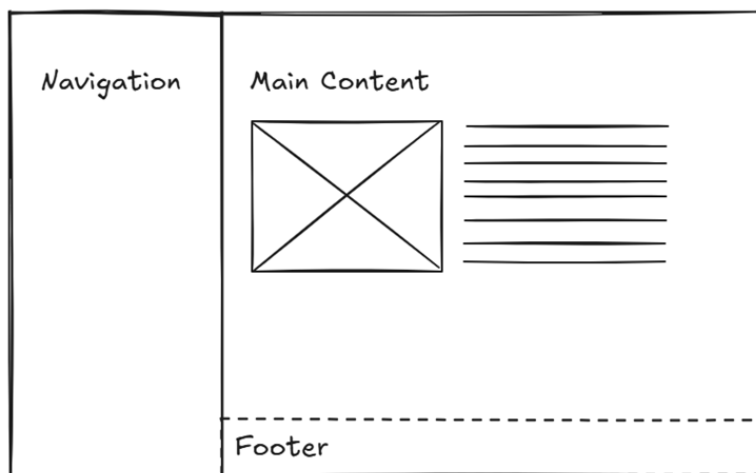
- Регистрация и автентикация на потребители;
- Търсене на съществуващи рецепти;
- Генериране на рецепти с помощта на GenAI;
- Съхранение и управление на генерираните рецепти.

*Отделните функционалности са по подробно разгледани в 3. Реализация.*

## 2.4. Wireframe. Визуална структура и дизайн.

Уеб приложението се придържа към визуалната структура, показана на Фигура 1.

Основната структура на страницата следва разделение на три основни части:



*Фигура 1. Wireframe на Mealise*

### 2.4.a. Навигационна лента отляво (sidebar):

Тя е фиксирана ("sticky") и позволява на потребителя лесно да се придвижва между основните функционалности – начална страница, търсене на рецепти, генериране на рецепти и вход/изход. Лявата лента използва светъл фон с подчертани връзки в син цвят, които се открояват на фона на минималистичния дизайн.

### 2.4.b. Основно съдържание (main):

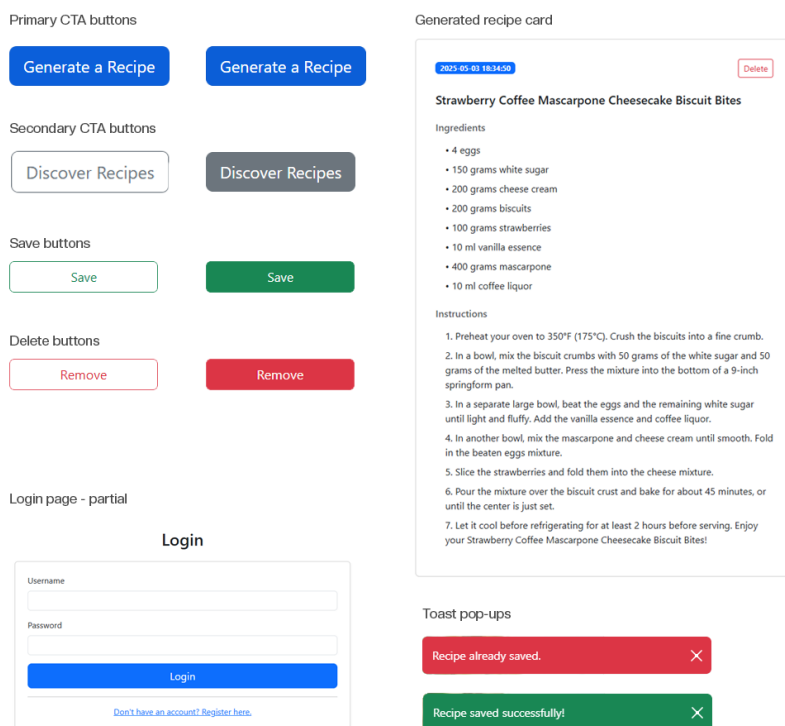
Централната част на приложението съдържа динамично сменящи се компоненти в зависимост от избрания маршрут – например landing страница, формулярите за вписване/регистрация, списъци с рецепти и др.

### 2.4.c. Футър (долна част):

Футърът е разположен в долната част на страницата и съдържа връзки към основните секции, както и информация за разработчика. Той използва същата цветова палитра и минималистичен стил, за да се впише хармонично с останалата част от дизайна.

Използвани са вградени компоненти на Bootstrap като container-fluid, row, col, card, btn, navbar, които значително улесняват подредбата и стилизирането на съдържанието.

Принципите на дизайна са ориентирани към удобство на потребителя. За целта е използвана последната версия на Bootstrap - v5.3, което предполага, че дизайнът ще е познат на крайния потребител, а навигирането из приложението - интуитивно. Основните части от дизайна са показани на Фигура 2.



Фигура 2. Основни елементи на дизайна, базирани на Bootstrap 5.3.

## 2.5. Описание на страниците

Приложението ще включва следните основни страници:

- **Начална страница:** представяне на приложението и с бутони с препратки към основните функционалности.
- **Списък с рецепти:** показва списък с всички рецепти, генерирани от потребителя.
- **AI генератор:** интерфейс за генериране на нови рецепти с помощта на GenAI. Изглед към вече генериране рецепти.
- **Регистрация:** форма за създаване на нов потребителски профил.
- **Вход:** форма за вход в съществуващ потребителски профил.

## 2.6. Архитектура на приложението

Схема на архитектурата е представена на Фигура 3. Уеб приложението се придържа към класическата трислойна MVC архитектура. Логиката е допълнително разширена чрез използване на външни API услуги. Всеки компонент в архитектурата има ясно дефинирана роля, както е показано в диаграмата:

### 2.6.a. Views (Изгледи):

Отговаря за визуализиране на данните към крайния потребител. Представя потребителския интерфейс и получава входни данни (заявки), които предава към слоя с бизнес логика.

### 2.6.b. Business Logic (Бизнес логика):

Това е основният координиращ слой, който обработва потребителските заявки. Получава вход от потребителския интерфейс (Views), прави заявки към моделите и базата данни, и ако е необходимо, осъществява връзка с външната API услуга за получаване на допълнителна информация.

### 2.6.c. Models (Модели):

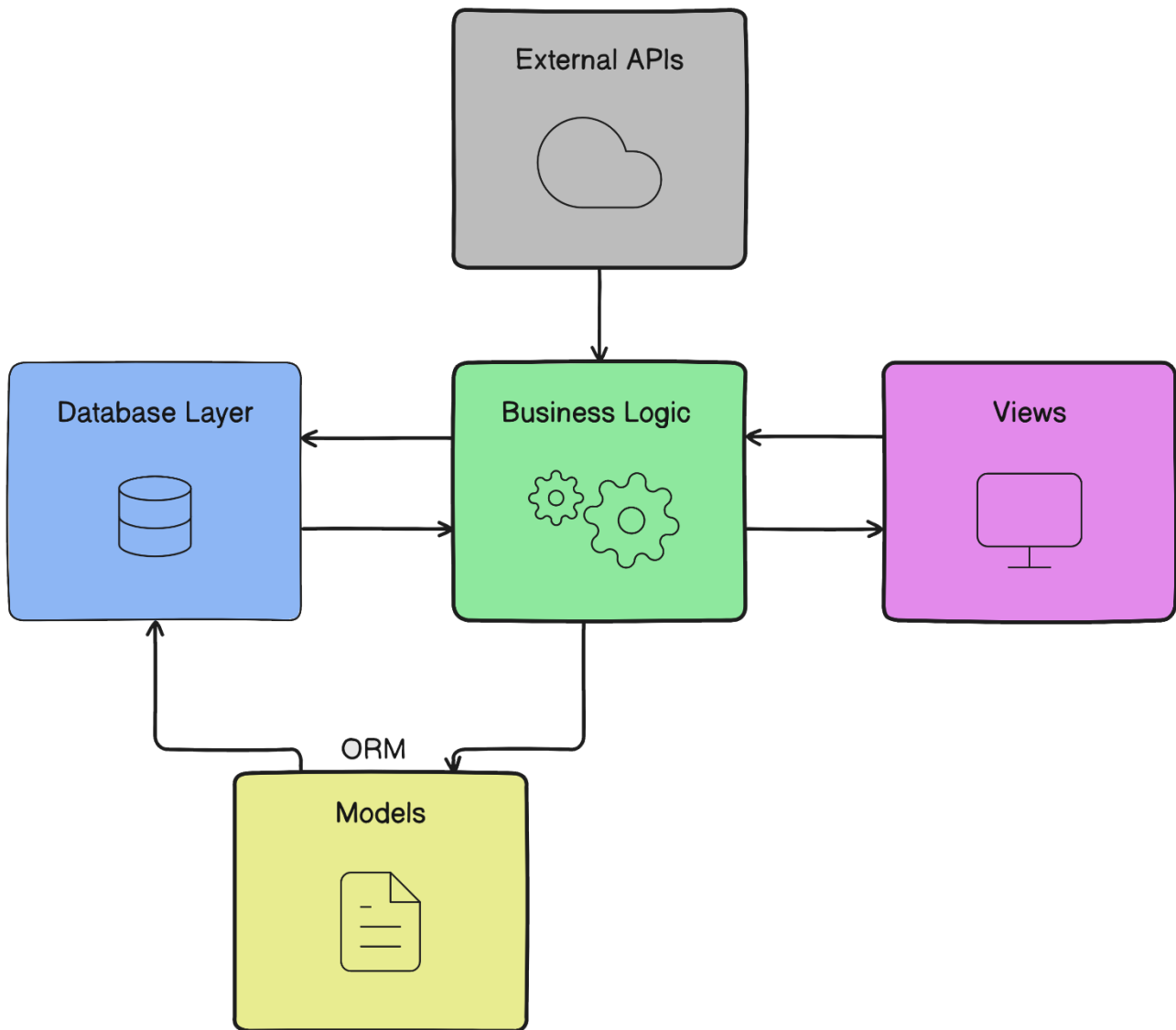
Представяват структурата на данните. Чрез ORM (Object-Relational Mapping) се свързва с базата данни, като позволява на бизнес логиката да чете и записва данни в нея по абстрактен и структуриран начин.

### 2.6.d. Database Layer (База данни):

Отговаря за съхранението на данните. Връзката между бизнес логиката, моделите и базата данни е осъществена чрез SQL Alchemy.

### 2.6.e. External API (Външни API услуги):

Допълват функционалността на приложението, като предоставят достъп до външни данни или услуги. Те се използват при нужда от допълнителна информация, която не се съхранява в локалната база данни.



Фигура 3. Схема на архитектурата на приложението.

## 2.7. Структура и логика на приложението

Структурата на проекта следва модулен подход, типичен за Flask уеб приложенията. Основните компоненти включват:

- **venv/**: виртуална среда, който е нужен за изолиране на пакетите и библиотеките, от които се нуждае приложението, за да функционира по очаквания начин.
- **instance/**: съдържа конфигурационни файлове и базата данни.

- **models/**: дефинира структурата на данните, които ще се пазят в базата от данни – в случая са два модела – User, Recipe.
- **routes/**: съдържа Python файлове, дефиниращи маршрутите на приложението и основната бизнес логика, реализирана в тях (auth.py, discover.py, home.py, recipes.py).
- **templates/**: съдържа HTML шаблони за визуализация на страниците.

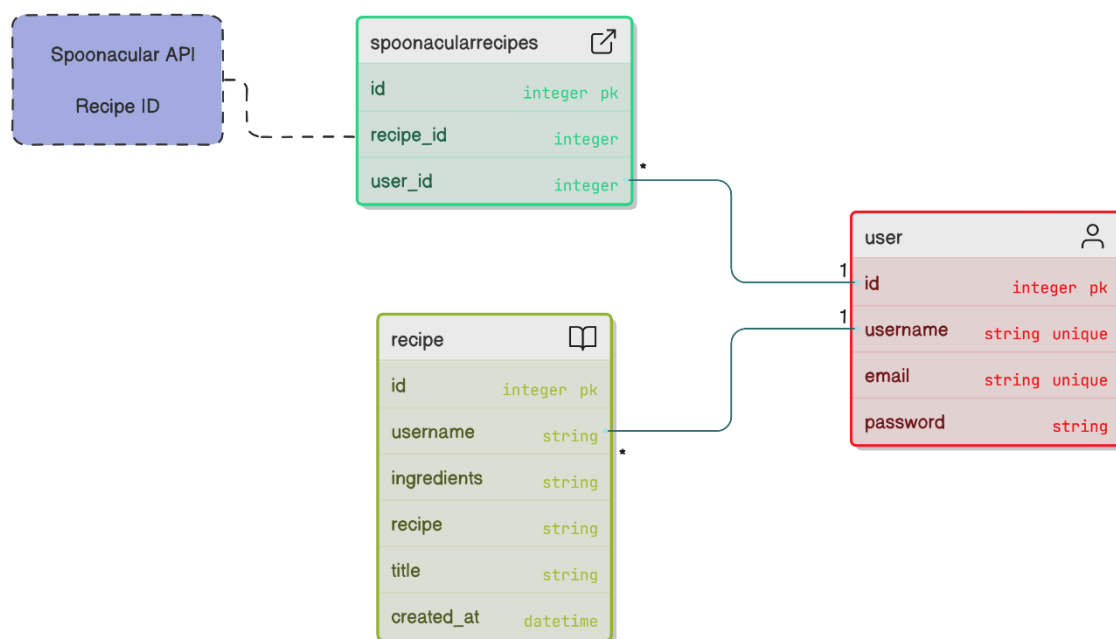
Основната логика на приложението включва обработка на потребителски заявки, комуникация с GenAI API за генериране на рецепти и взаимодействие с базата данни за съхранение и извличане на информация. Маршрутите определят как приложението реагира на различните URL адреси и HTTP методи.

## 2.8. База от данни

Базата данни е изградена с SQLite и съдържа следните таблици:

- **Users**: съхранява информация за потребителите. Атрибути: username, email, password.
- **Recipes**: съхранява информация за генерираните рецепти. Атрибути: title, ingredients, instructions, created\_at, username.
- **Spoonacular\_recipes**: междинна таблица, която съхранява информация за запазените рецепти от Spoonacular API. Атрибути: id, user\_id, recipe\_id

Връзката между обектите е "едно към много" - един потребител може да има много рецепти. А при регистрацията на нов потребител, паролата задължително се криптира, с оглед на защита на профилите.



Фигура 4. Схема на базата от данни.



## 3. Реализация

Този раздел описва детайлите по реализацията на уеб. В раздела е включена информация за използваните технологии, описание на функционалностите, страниците и примери от кода.

### 3.1. Използвани технологии

Mealise използва следния технологичен стек и компоненти:

КОМПОНЕНТ	ОПИСАНИЕ
<b>Python 3.13</b>	Основния език, на който е разработен целия проект.
<b>Flask</b>	Лек и мощен уеб фреймуърк за създаване на уеб приложения и RESTful API интерфейси.
<b>SQLite</b>	Лек, файл-базиран SQL-engine за съхранение на потребителски данни и рецепти.
<b>SQLAlchemy</b>	ORM (Object-Relational Mapper) библиотека, използвана за взаимодействие с базата данни. Позволява обектно-ориентиран развой.
<b>GenAI API</b>	API, използван за генериране на рецепти.
<b>HTML/CSS</b>	Използвани за структурата и стилизирането на уеб страниците.
<b>Jinja2</b>	Template engine, използван за генериране на динамични HTML страници.
<b>Външни API-та</b>	Spoonacular API - за търсене на рецепти, OpenAI - за генериране на нови рецепти на база съставки.
<b>JavaScript</b>	Използван за добавяне на интерактивност към уеб страниците.
<b>Bootstrap</b>	CSS framework, използван за бързо и лесно създаване на responsive дизайн.

### 3.2. Функционалност: Автентикация

Автентикацията се използва за управление регистрациите, вписването в и излизане от потребителските акаунти. Контролерът, отговарящ за автентикацията – Auth Controller (./auth), разчита на няколко ключови разширения на Flask, включително: flask\_login за управление на потребителски сесии, bcrypt за хеширане на пароли и SQLAlchemy за взаимодействията с базата от данни. Използва се и таен ключ за сесиите.

Функционалността се имплементира чрез няколко ключови маршрути:

### 3.2.a. Register (.../auth/register)

- Поддържа методи GET и POST, показвайки съответно формуляр за регистрация и обработвайки подадените от потребителя данни.
- Проверките за валидиране гарантират, че потребителското име, имейл адресът и паролата са предоставени и че паролата, въведена два пъти, е еднаква.
- Функцията, реализираща крайната точка, извършва търсене в базата данни, за да провери уникалността на потребителското име и имейл адреса, предотвратявайки дублиране на акаунти. Паролите се хешират с помощта на `bcrypt` преди да бъдат съхранени в базата. Пример за хеширане на потребителските пароли:

```
hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
```

- След успешна регистрация, новият потребител автоматично е вписан в системата и бива пренасочван към главната страница на приложението.

### 3.2.b. Login (.../auth/login)

- Поддържа също GET и POST методи;
- Маршрутът извлича подадените потребителски данни и праща запитване към базата данни за посоченото потребителско име.
- Ако съответстващия потребител е намерен, въведената парола се проверява спрямо съхранената хеширана парола, използвайки `bcrypt.check_password_hash`.
- Ако хешираните версии на паролите съвпадат – потребителят е успешно удостоверен. Той е вписан в системата чрез `flask_login.login_user`, като се пази информация и за текущата сесия. Извежда се и флаш съобщение за успех.
- Функцията, също така, поддържа параметър за следваща заявка, позволяващ пренасочване към страницата, до която потребителят първоначално е възнамерявал да осъществи достъп преди влизане. Например, ако потребителят се е опитал да достъпи страница, за която трябва да е автентикиран, то след успешно вписване, потребителят бива пренасочван към същата страница.
- Ако влизането е неуспешно (поради неправилно потребителско име или парола), флаш съобщение информира потребителя и страницата за вход се презарежда.

### 3.2.с. Logout (/auth/logout):

- Един потребител не може да излезе от акаунта си, ако изобщо не се е вписвал. Това се проверява чрез атрибута **@login\_required**, предотвратявайки неоторизиран достъп.
- За осъществяването на функционалността функцията използва `flask_login.logout_user`. Сесията бива изтрита и потребителят успешно е излязъл от профила си.,
- Потребителят се пренасочва обратно към страницата за вход след излизане.

## 3.3. Функционалност: Генериране на рецепти

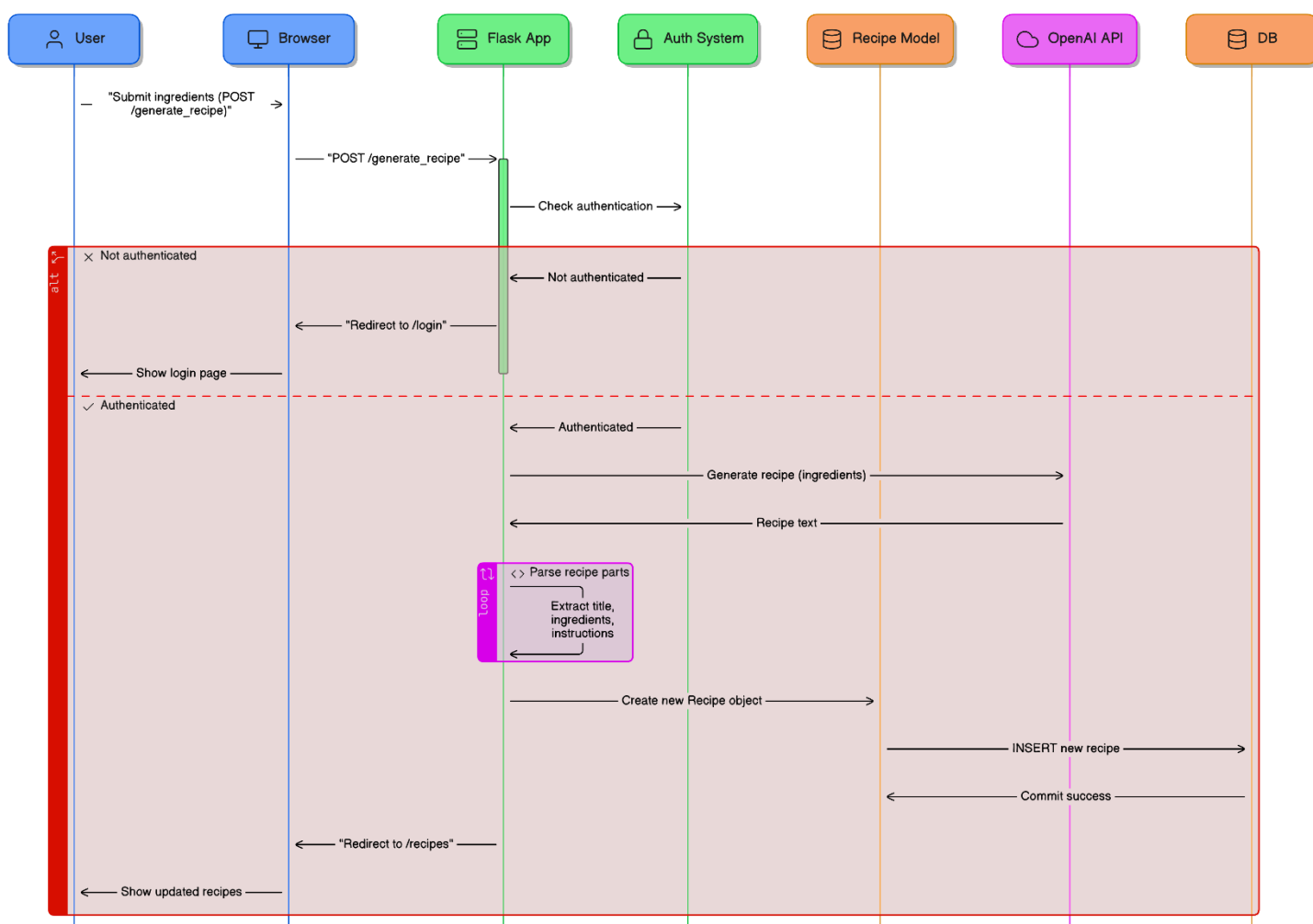
- Генерирането на рецепти се случва чрез маршрута `/recipes/generate_recipe`, който е осъществен, чрез функция, дефинирана да приема само POST заявки. Задължително се изисква автентификация на потребителя чрез атрибута **@login\_required**.
- Функцията очаква списък със съставки, които потребителя въвежда във формата. Ако потребителя се опита да създаде рецепта без да е въвел съставки, изпълнението на функцията спира и страницата се презарежда.
- След проверката за автентификация и наличност на съставки, маршрута извиква помощна функция **generate\_recipe(ingredients)**, която отговаря за осъществяването на връзка към AI модела и генерирането на рецептата.
- Помощната функция използва **mistralai/mistral-7b-instruct**, с предварително зададен и конкретно-оформен текст:

```
def generate_recipe(ingredients):  
    prompt = f"""  
You are a creative chef. Generate a recipe using the following ingredients:  
{ingredients}  
You don't need to include ALL of the ingredients.  
You can't use ingredients that are not in the list.  
Exclude listed items that aren't food.  
Format your answer exactly like this:  
  
$part-title\n[Insert a creative recipe title here]  
  
$part-ingredients  
-- ingredient 1  
-- ingredient 2  
-- ingredient 3  
  
$part-instructions  
1. Step one  
2. Step two  
3. Step three """
```

- Очакваният отговор е структуриран в три части, разделени с текстови маркери. Този подход улеснява визуализирането на информацията в HTML шаблоните:
  - **\$part-title** – заглавие на рецептата
  - **\$part-ingredients** – списък със съставки
  - **\$part-instructions** – инструкции за приготвяне
- Резултата (генерирания отговор от модела) се преработва и се създава нов запис в базата данни със съответните полета и текущото потребителско име.

```
@recipes_bp.route('/generate_recipe', methods=['POST'])
def generate_and_save_recipe():
    ...
    new_recipe = Recipe(
        username=current_user.username,
        ingredients=parsed_ingredients,
        recipe=instructions,
        title=title,
        created_at=db.func.current_timestamp()
    )
    db.session.add(new_recipe)
    db.session.commit()
```

- След успешно записване, потребителят се пренасочва обратно към списъка с рецепти.
- Подробна схема на стъпките, през които преминава алгоритъма, е представена на следващата страница на Фигура 5.



Фигура 5. Sequence диаграма на функционалност „генериране на рецепти“.

## 3.4. Функционалност: Търсене на съществуващи рецепти чрез Spoonacular API

### 3.4.a. Discover Recipes (/discover)

- Този маршрут отговаря както за GET, така и за POST заявки. Той позволява на потребителите да прилагат филтриране на рецептите чрез следните параметри: име на храна, вид кухня (пр. италианска), предпочитания (вегетарианска, веган) и сортиране. Стойностите по подразбиране гарантират, че страницата зарежда първоначалните резултати, когато не са предоставени никакви параметри.
- Маршрута използва помощна функция **fetch\_recipes** с подходящите филтри, ако има приложени такива, и показва резултатите.

### 3.4.b. Зареждане на още рецепти (/discover/load\_more)

- Този GET маршрут поддържа безкрайно превъртане (infinite scrolling) или механизъм за „зареждане на още“, като приема текущия номер на страницата и параметрите на филтъра, като съответно извлича следващия набор от рецепти.

```
@spoonacular_bp.route('/discover/load_more', methods=['GET'])
def load_more_recipes():
    page = int(request.args.get('page', 1))
    query = request.args.get('query', '')
    cuisine = request.args.get('cuisine', '')
    diet = request.args.get('diet', '')
    sort = request.args.get('sort', '')

    recipes = fetch_recipes(page, query, cuisine, diet, sort)
    ...
```

### 3.4.c. Запазване на рецепта (/save\_recipe)

Тази POST крайна точка позволява на удостоверени потребители да запазват рецепти в профила си. Преди запазване, функцията проверява дали рецептата вече е запазена, за да предотврати дублиране. При успех или неуспех, той връща JSON отговор, указващ резултата. JSON отговора се използва за визуализиране на поп-уп с подходяща информация за настъпилата промяна (запазена рецепта/рецептът вече е запазен).

## 3.5. Функционалност: Съхранение и управление на рецепти.

Важно уточнение – имплементацията на контролерите за управление на рецепти приема, че потребителите са удостоверени, извличайки информацията за текущия автентикиран потребител от прокси сървър `current_user` на `flask_login`.

### 3.5.a. Съхранение на рецепти

И генерираните рецепти и тези, които се извличат от Spoonacular се пазят в базата от данни в две таблици. Създават се инстанции на съответните модели `Recipe` и `SpoonacularRecipe`. `Recipe` моделът се използва за съхранение на информация за генерираните рецепти, докато `SpoonacularRecipe` – информация за рецептите, идващи от външното API.

```
new_recipe = SpoonacularRecipes(
    recipe_id=recipe_id,
    user_id=user_id,
)
db.session.add(new_recipe)
db.session.commit()
```

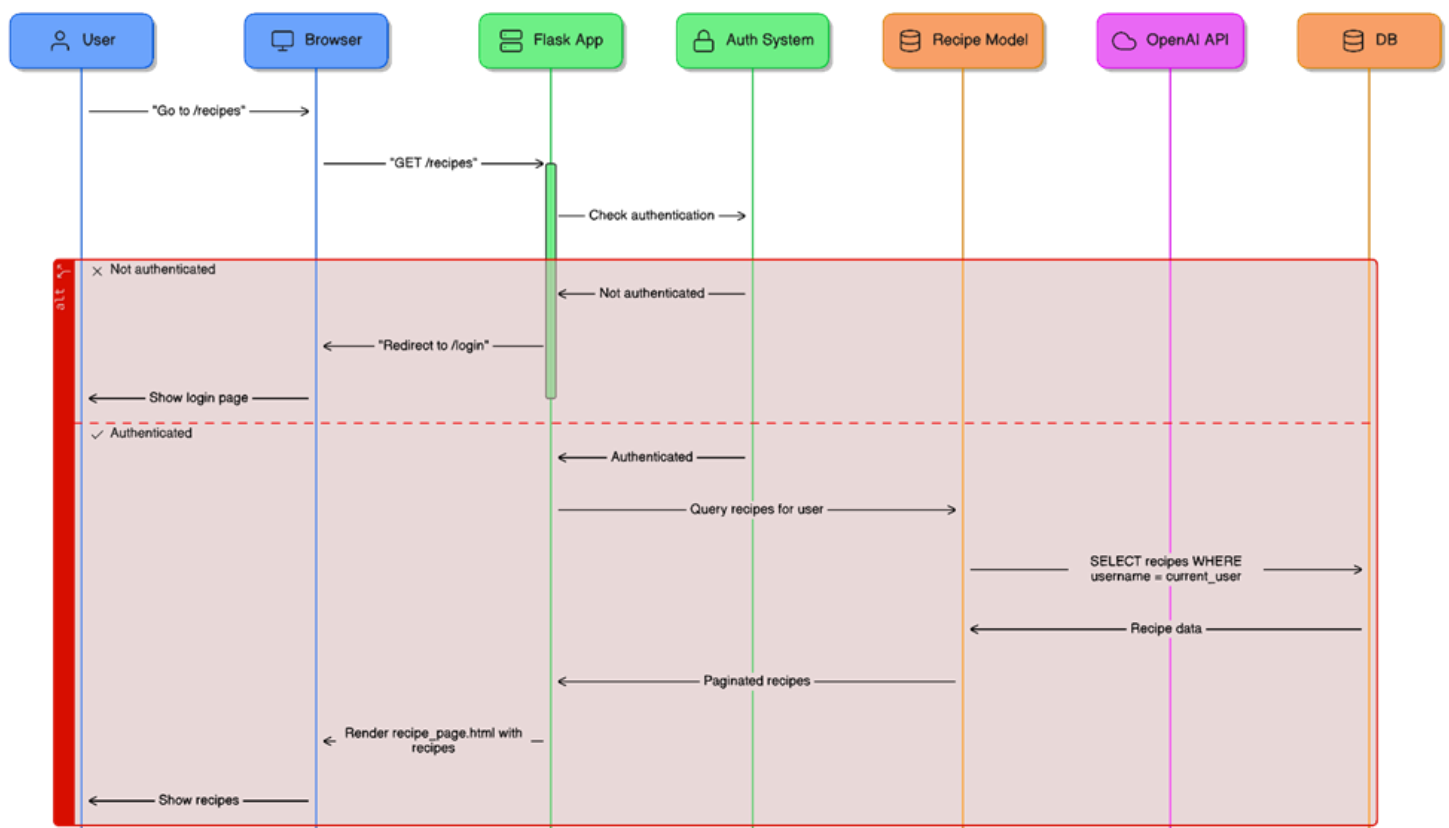
```

new_recipe = Recipe(
    username=current_user.username,
    ingredients=parsed_ingredients,
    recipe=instructions,
    title=title,
    created_at=db.func.current_timestamp()
)
db.session.add(new_recipe)
db.session.commit()

```

### 3.5.b. Преглед на запазени рецепти от Spoonacular (/saved)

- Потребителите могат да преглеждат запазените си рецепти чрез тази GET крайна точка.
- В базата данни се пази информация за идентификатор на потребителя и идентификатор на рецептата от Spoonacular API.
- Подробностите за рецептите се извличат Spoonacular API, използвайки запазените в базата идентификатори.



Фигура 6. Sequence диаграма на разглеждане на рецепти от Spoonacular.

### 3.5.с. Премахване на рецепти

- Тази функционалност се реализира чрез два POST маршрута, които позволяват на потребителите да премахнат запазените/генерираните рецепта от профила си. И двете функции, респективно към двата маршрута, проверява дали рецептата съществува, преди да я изтрие, и съответно връща отговор за успех или грешка.
- **Премахване на рецепта от Spoonacular:** Самото премахване става посредством POST заявка, съдържаща в себе си параметър `recipe_id`. След това, заявка към базата данни проверява дали рецептата с дадения идентификатор е запазена от този потребител в таблицата `SpoonacularRecipes`. Ако бъде намерен, записът се изтрива от таблицата и промените се записват в базата данни. Като отговор функцията връща JSON параметър. Ако рецептата не е била запазена преди това, се връща грешка 404 със съобщение за неуспех.

```
saved = SpoonacularRecipes.query.filter_by(user_id=user_id, recipe_id=recipe_id).first()
    if saved:
        db.session.delete(saved)
        db.session.commit()
        return {'success': True}

    return {'success': False}, 404
```

- **Премахване на рецепта, генерирана от потребител:** По подобие на функцията, описана преди тази, функцията, отговаряща за изтриването на рецепта, приема POST заявка с `recipe_id` като параметър. Проверява се дали рецептата съществува. Ако не - автоматично се генерира грешка 404. При неоторизиран опит за достъп се връща отговор 403. При успешна проверка на потребителя и рецептата, последната се изтрива от базата данни. След това потребителят се пренасочва към страницата за генериране на рецепти.

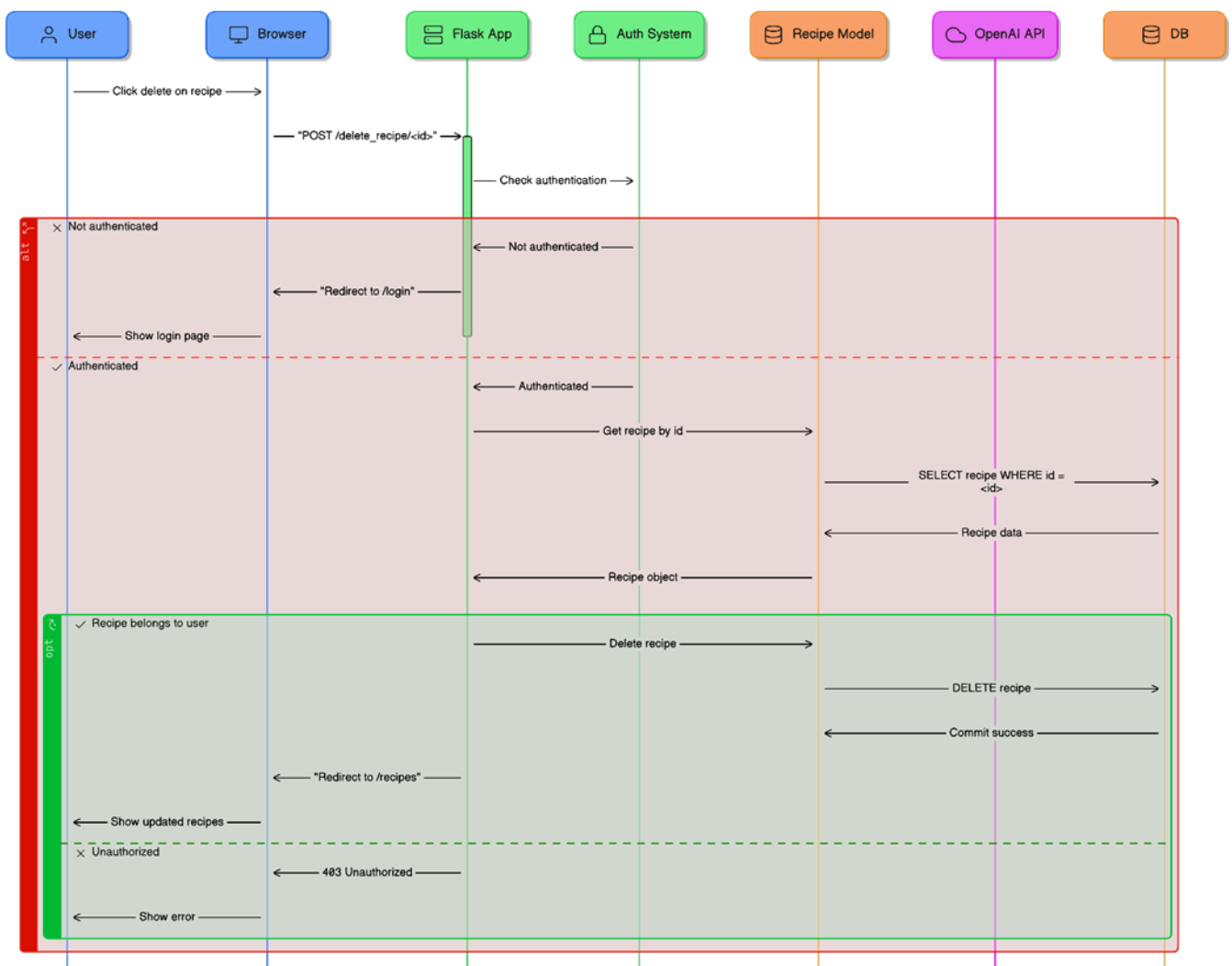
```
recipe = Recipe.query.get_or_404(recipe_id)

    if recipe.username != current_user.username:
        return "Unauthorized", 403

    db.session.delete(recipe)
    db.session.commit()
```

- Подробна схема на стъпките, през които преминава алгоритъма за изтриване на генерирана рецепта, е представена на следващата страница на Фигура 7.

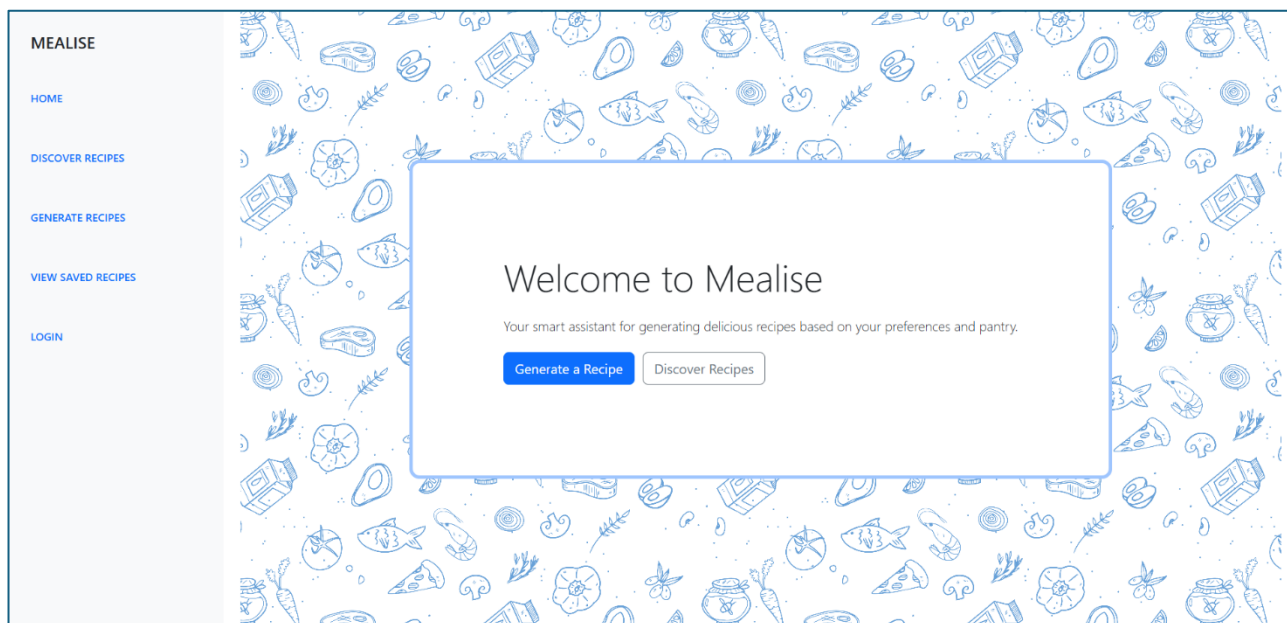




Фигура 7. Sequence диаграма на функционалност "изтриване на рецепта"

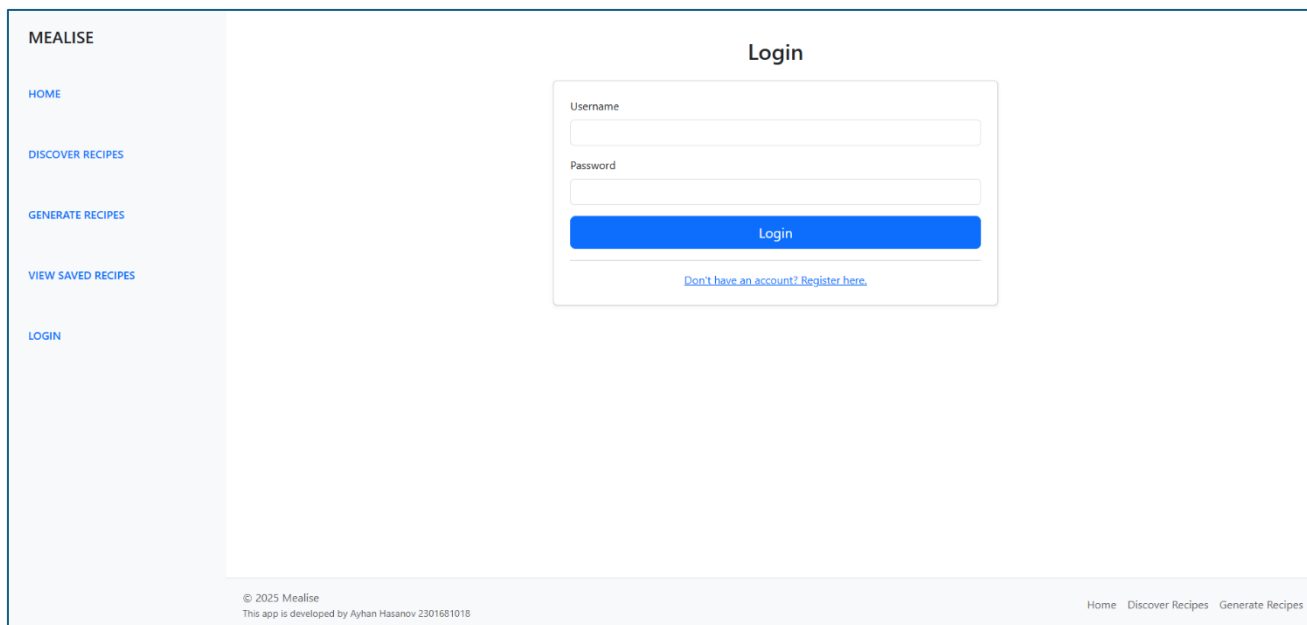
## 3.6. Екрани

### 3.6.a. Начален екран при невисан потребител



Фигура 8. Екран Homepage.

### 3.6.b. Екран за вписване в системата



Фигура 9. Екран Login страница.

### 3.6.c. Екран за регистрация

MEALISE

HOME

DISCOVER RECIPES

GENERATE RECIPES

VIEW SAVED RECIPES

LOGIN

Register

Username

Email

Password

Confirm Password

Register

[Already have an account? Login here.](#)

© 2025 Mealise  
This app is developed by Ayhan Hasanov 2301681018

Home Discover Recipes Generate Recipes

Фигура 10. Екран Registration страница.

### 3.6.d. Екран с каталог от рецепти от Spoonacular

MEALISE

WELCOME, AYHAN

HOME

DISCOVER RECIPES

GENERATE RECIPES

VIEW SAVED RECIPES

LOGOUT

Discover Recipes

Advanced Search

Keyword (e.g. pasta)

Any Cuisine


Any Diet

No Sorting

Search

Clear Filters

Ready in: 55 mins




Red Lentil Soup with Chicken and Turnips

View Recipe

Save

Ready in: 20 mins




Asparagus and Pea Soup: Real Convenience Food

View Recipe

Save

Ready in: 45 mins




Garlicky Kale

View Recipe

Save

Ready in: 490 mins




Slow cooker Beef Stew

View Recipe

Save

Ready in: 45 mins




Vegetable and Bean Stew

View Recipe

Save

Ready in: 30 mins



Quinoa and Vegetable Salad

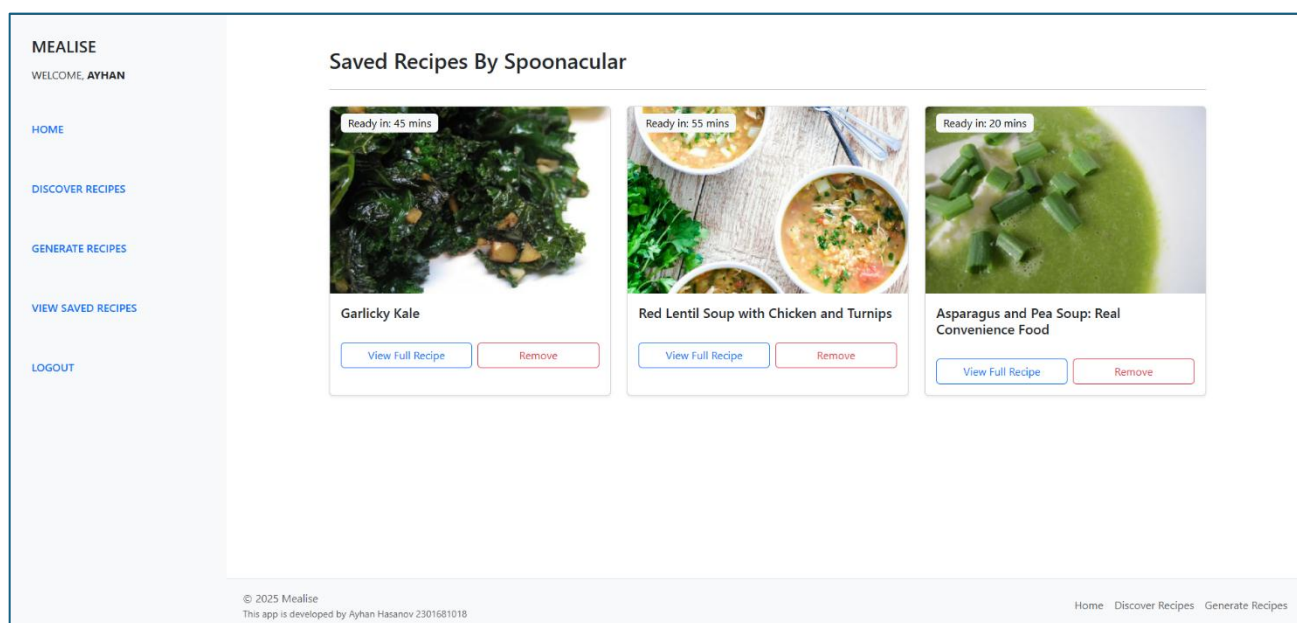
View Recipe

Save

Фигура 11. Екран Discover Recipes с каталог от рецепти от Spoonacular.

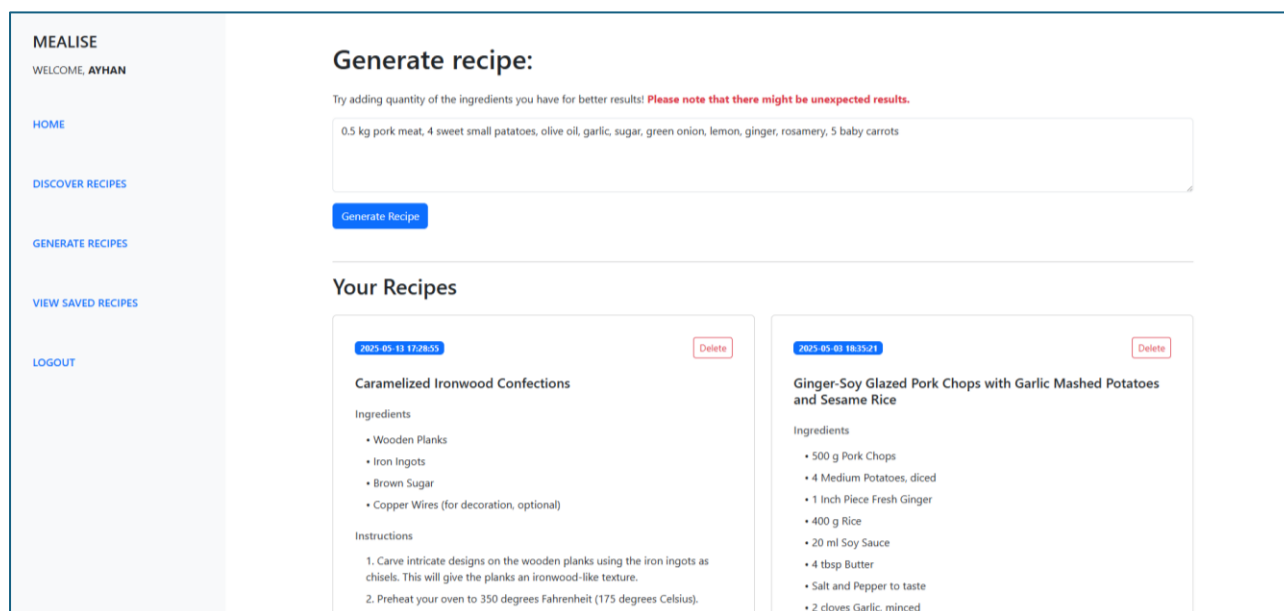
19

### 3.6.e. Екран със запазените рецепти от Spoonacular.



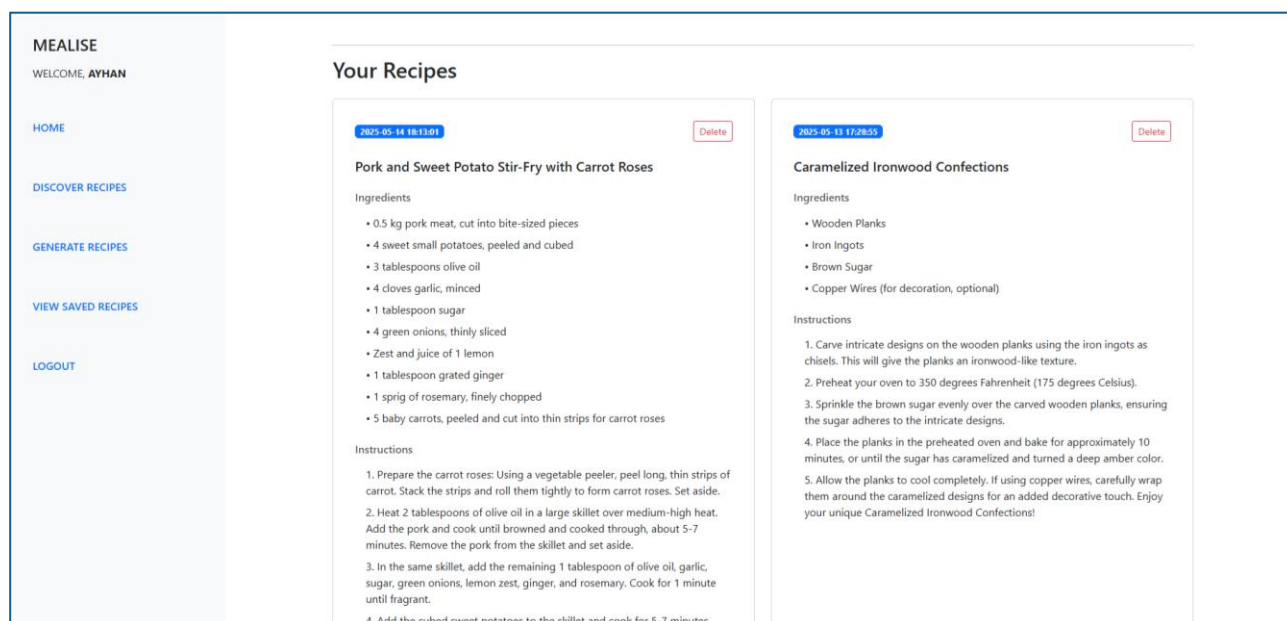
Фигура 12. Екран View Saved Recipes с каталог от запазените рецепти от Spoonacular.

### 3.6.f. Екран със страницата за генериране на рецепти и управление на генерираните рецепти.



Фигура 13. Екран Generate Recipes за генериране на рецепти.

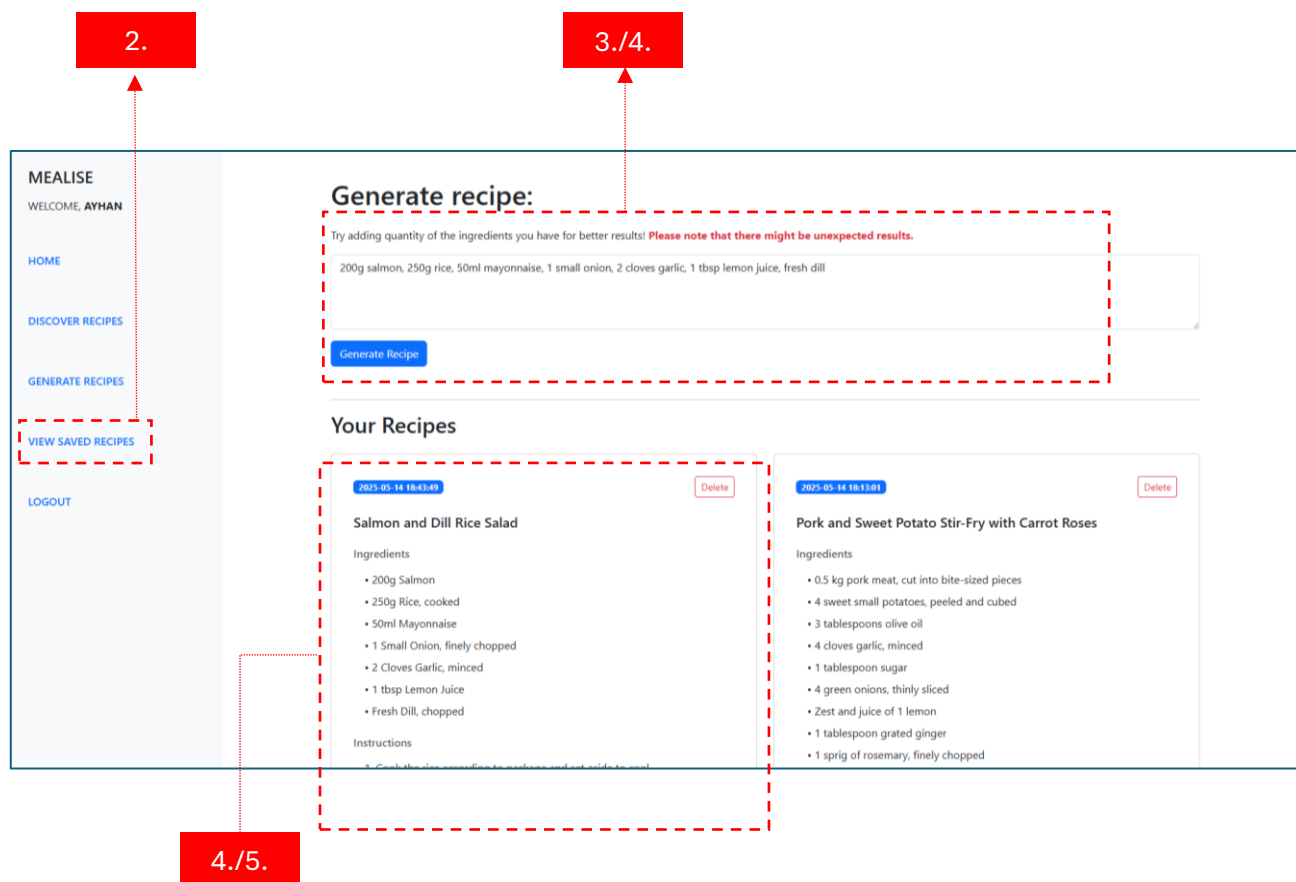
### 3.6.g. Екран със сащата страница от 3.5.f. с фокус към генерираните рецепти.



Фигура 14. Екран *Generate Recipes* - генерирани рецепти.

## 3.7. Примерни стъпки за генериране на рецепта

- 1. След стартирането на проекта, приложението се достъпва чрез, който и да е, браузър на следния URL адрес: 127.0.0.1:5000 (или localhost:5000).
- 2. За да генерираме рецепта, първо трябва да се регистрираме в системата като нови потребители или, ако вече имаме съществуващ акаунт – да се впишем. Това може да се случи чрез навигацията вляво – “Login”. Алтернативен вариант е да се избере “Generate Recipe” – системата автоматично ще пренасочи всички неотторизирани потребители към Login-страницата.
- 3. След като сме се вписали, ако вече не се намираме в нужния екран, се избира “Generate Recipe” от навигационното меню. След като отворим страницата най-отгоре ни посреща голямо текстово поле, в което ще се въвеждат съставките.
- 4. Въвеждат се всички съставки по предпочитание и/или наличност на потребителя (например: 200g salmon, 250g rice, 50ml mayonnaise, 1 small onion, 2 cloves garlic, 1 tbsp lemon juice, fresh dill).
- 5. След като всички съставки са въведени, се натиска бутонът “Generate Recipe”, намиращо се точно под текстовото поле. Рецептата е генерирана и се намира най-отгоре малко по-надолу от текстовото поле.



Фигура 15. Схема на основните елементи в страницата *Generate Recipes*.

## 3.8. Инструкции за инсталация

- Изтеглете ZIP файла на проекта от следния линк и го разархивирайте в желана от вас директория. Линк: [https://github.com/AyhanHasanov/Mealise\\_v1](https://github.com/AyhanHasanov/Mealise_v1)
- Инсталирайте Python 3.13. Ако имате инсталиран Python, можете да проверите версията, като изпълните следната команда чрез CMD (с администраторски права) или PowerShell:

```
python -V.
```

- Отворете CMD (с администраторски права) или PowerShell и навигирайте до директорията, където се намира папката на разархивирания проект.
- Уверете се, че сте с администраторски права и изпълнете следното (ако използвате Visual Studio Code, изпълнете командите в прозореца на терминала):

```
python -m venv venv
```

- Инсталирайте необходимите зависимости (уверете се, че сте във виртуалната среда)

```
$ pip install -r requirements.txt
```

- Тайните ключове, от които се нуждае приложение, за да функционира се намират в .env файл. Трябва да се създаде такъв .env файл в главната директория със следното съдържание:

```
FLASK_SECRET_KEY=your_flask_secret_key  
  
DATABASE_URI=sqlite:///instance/mealise.db  
  
SPOONACULAR_API_KEY=your_spoonacular_api_key  
  
OPENROUTER_API_KEY=your_openrouter_api_key
```

- Уверете се, че сте записали с действителните си ключове.
- Стартирайте приложението чрез следната команда (отново в CMD/Powershell или терминала на Visual Studio Code).

```
$ python app.py
```

- Приложението се достъпва чрез който и да е браузър на следния URL адрес: **http://127.0.0.1:5000/** или **localhost:5000**
- Обърнете внимание, че приложението работи локално, следователно всички стъпки, изброени тук, са необходими, за да работи приложението, както се очаква.

## 4. Заключение

Mealise е уеб приложение, което бъдейки базиран върху реален проблем, цели да бъде вид дигитален помощник в кухнята, предлагайки генерирани рецепти.

Системата използва съвременни технологии и разработен на актуална версия на Python, заедно с редица python-базирани пакети и библиотеки, които разширяват по един или друг начин приложението.

С ясна структура, сигурна автентикация и интеграция с AI, приложението позволява на потребителите да се регистрират, създават и откриват рецепти. Генерирането на рецепти се случва чрез OPENROUTER, докато каталога с готови рецепти използва Spoonacular API.

В процеса на разработката на уеб приложението, едно от основните предизвикателства беше интегрирането на GenAI API и обработването на резултатите и по-скоро намирането на модел, който да е безплатен. След дълго проучване и експерименти с редица модели, **mistralai**

беше избран за целите на функционалността, тъй като е безплатен и предлага това, което е нужно за целите на приложението.

## 5. Използвани източници.

### 5.1. Използване на генеративен изкуствен интелект

В процеса на разработка на проекта активно бяха използвани съвременни генеративни AI модели, по-специално ChatGPT на OpenAI и DeepSeek. Тези инструменти значително подпомогнаха работата в няколко ключови аспекта на проекта:

#### 5.1.a. Брейнсторминг и планиране

Моделите бяха използвани още в началните фази на проектиране – за изясняване и оформяне на идеята, както и за избор на подходящ технологичен стек, бази данни, фреймуърк и други ключови компоненти.

#### 5.1.b. Примерен разговор с ChatGPT.

**Prompt:** Is using MySQL in a Flask App possible and is it a viable solution for the set problem with the given requirements and context?

**Response:** Yes, Flask + MySQL is a great option!  
Flask is lightweight, easy to work with, and integrates well with MySQL using SQLAlchemy (ORM) or the mysql-connector-python package.  
Flask + MySQL Tech Stack  
Backend: Flask (Python)  
Database: MySQL (or MariaDB if needed)  
Frontend: HTML + JavaScript (or Vue.js if you want interactivity)  
Authentication: Flask-Login or JWT  
AI Integration: OpenAI API (or another AI-based recipe API)

#### 5.1.c. Дебъгване и отстраняване на грешки

Освен при планирането, ChatGPT и DeepSeek бяха използвани и като помощни средства при откриване и решаване на програмни грешки и бъгове по време на разработката. Моделите предоставяха разяснения, възможни причини за бъговете и конкретни решения, често съчетани с примерен код. Често предложените решения не разрешаваха настъпилите проблеми.

### 5.2. Документация

За схемите и фигурите в тази документация бяха използвани следните източници: eraser.io и excalidraw.