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 или **stu2301681018@uni-plovdiv.bg**
Факултетен номер: **2301681018**

СЪДЪРЖАНИЕ:

1. Въведение в Mealise

Mealise е иновативно уеб приложение, създадено с цел да улесни потребителите в намирането и създаването на кулинарни рецепти, които отговарят на техните предпочитания и налични съставки. Това интелигентно решение използва мощта на Flask — лек и гъвкав Python уеб фреймуърк — за изграждане на модулна, мащабируема и лесна за поддръжка платформа, която съчетава локално съхранение на данни с външни API интеграции, предоставяйки богата и персонализирана функционалност.

1.1. Основна архитектура и цели на приложението

Проектът Mealise е структурирано като класическо Flask приложение, което следва най-добрите практики за разделяне на кода в различни модули — с цел повишаване на четимостта, повторната използваемост и улеснена поддръжка. Това позволява на разработчиците бързо да добавят нови функционалности или да оптимизират съществуващи, без да нарушават цялостната структура на системата.

Ключовите цели на Mealise включват:

- **Интелигентно генериране на рецепти** — чрез използването на OpenAI модели, Mealise създава креативни и персонализирани рецепти на база въведени съставки от потребителя;
- **Откриване на рецепти чрез външни API-та** — интеграция със Spoonacular API позволява достъп до стотици професионално подбрани рецепти с възможност за филтриране по кухня, диетични изисквания, популярност и други;
- **Управление на потребители и рецепти** — регистрация, вход, и авторизация на потребители, които могат да съхраняват, преглеждат и изтриват свои рецепти в локалната база данни;
- **Модулна структура** — ясно разделение на логиката за автентикация, сървърни маршрути, модели, шаблони и разширения за опростено развитие и тестване.

1.2. Технологичен стек и основни компоненти

Mealise използва следния технологичен стек и компоненти:

Компонент	Описание
Python & Flask	Лек и мощен уеб фреймуърк за създаване на уеб приложения и RESTful API интерфейси.
Виртуална среда	Директория venv/ съдържа локални пакети и зависимости, изолирани от системната среда.
SQLite база данни	Лек, файл-базиран SQL двигател за съхранение на потребителски данни и рецепти (mealise.db).
Flask-Login	Управление на сесии и автентикация на потребители.

Компонент	Описание
Flask-SQLAlchemy	ORM слой за работа с базата данни, позволяващ обектно ориентиран достъп.
Flask-Bcrypt	За криптиране и защита на пароли в базата.
Външни API-та	Spoonacular API за търсене на рецепти, OpenAI за генериране на нови рецепти на база съставки.

1.3. Роля на виртуалната среда и зависимостите

Една от ключовите практики при разработката на Mealise е използването на виртуална среда (`venv/`), която инсталира всички необходими Python библиотеки изолирано от глобалната инсталация. Това гарантира стабилност, защитава от конфликти между различни проекти и улеснява обновяване и възстановяване на средата по време на разработка и продукция.

1.4. Модулна и разширяема структура

Една от силните страни на Mealise е модулната организация, която значително се улеснява от Flask Blueprints и отделянето на отделни функционални зони в проекта:

- **Модели (`models/`)** — дефинират основните структури на данни (потребители и рецепти), като използват SQLAlchemy обекти;
- **Маршрути (`routes/`)** — организирани в самостоятелни модули за автентикация, управление на рецепти, домашна страница и откриване на съдържание чрез Spoonacular;
- **Шаблони (`templates/`)** — HTML/Jinja2 файлове, които разделят синтаксиса на фронтенда от бекенда, осигурявайки повторна употреба и лесно кастомизиране на интерфейса;
- **Допълнителни разширения (`extensions.py`)** — централизирана инициализация на базата данни и механизми за сигурност.

Тази архитектура позволява лесно добавяне на нови функционалности като нови API интеграции, персонализирани филтри или разширения в интерфейса, без да е необходима сериозна намеса в основния код.

1.5. Интеграция с външни API-та за богато потребителско изживяване

Mealise съчетава локалното съхранение и обработка на потребителски рецепти с мощни външни източници на кулинарно съдържание. Интеграцията с **Spoonacular API** осигурява достъп до голямо разнообразие от рецепти от световната кухня, които могат да бъдат търсени и филтрирани по множество критерии. Това значително обогатява възможностите на потребителите и ги улеснява в откриването на нови идеи за готвене.

За генериране на персонализирани рецепти, приложението използва **OpenAI API** чрез OpenRouter, който базира създаването на рецепти върху въведени от потребителя съставки. Този интелигентен подход дава уникални и креативни предложения, които могат да включват точно описани инструкции и списък на необходимите продукти, адаптирани към конкретните наличности.

1.6. Предимства на Mealise като модулно, гъвкаво приложение

- **Лесна поддръжка и разширяване** — благодарение на разделената архитектура, нови модули и функционалности могат да бъдат добавяни без негативно въздействие върху съществуващия код;
- **Сигурност и управление на потребителите** — интеграция с Flask-Login и криптиране на пароли с Bcrypt осигуряват надеждна автентикация и сесийно управление;
- **Универсалност при разгръщане** — използването на конфигурационни файлове и променливи на средата позволява лесно адаптиране към различни среди — например разработка, тестове и продукция;
- **Отворен код и документация** — ясното документиране и разделяне на функционалностите подпомагат както настоящите, така и бъдещите разработчици да навлязат бързо и ефективно в проекта.

В заключение, Mealise представлява иновативна и практична платформа, която съчетава модерни технологии и интелигентни решения за улесняване на кулинарния процес, като едновременно с това остава лесна и удобна за разработка, поддръжка и разрастване.

2. Структура на проекта Mealise

Проектът Mealise е изградена с идеята за простота, модулност и разширяемост, което позволява както лесна поддръжка, така и можливост за бързо разрастване на функционалностите. В тази секция ще разгледаме подробно структурата на директориите и ключовите файлове, които съставят ядрото на приложението. Това включва обяснение за виртуалната среда, управлението на конфигурационните файлове, моделите за данни, маршрутизацията, шаблоните (templates) и разширенията, които гарантират сигурна и ефективна работа на Mealise.

2.1. Виртуална среда – venv/

Директорията venv/ съдържа виртуалната среда, която е основен компонент на всеки модерен Python проект. Създаването и използването на виртуална среда позволява на разработчиците да инсталират специфичните зависимости за проекта, без да променят системните пакети на операционната система. Това дава възможност за:

- Изолация на пакетите и зависимости, използвани в Mealise;
- Предотвратяване на конфликти между различни проекти;
- Лесна инсталация и поддръжка на специфичните версии на библиотеките.

Благодарение на тази изолация, при всяко обновяване или разгръщане на Mealise, средата остава стабилна и предвидима, което е от ключово значение за продукционните среди.

2.2. Конфигурации и базата данни – instance/

Директорията instance/ съдържа конфигурационни файлове и локалната база данни (файлът mealise.db). Този подход се използва за съхранение на настройки, които са специфични за дадена инстанция на приложението, като например връзки към бази данни, тайни ключове или други чувствителни данни. Основните характеристики на тази директория включват:

- Отделяне на конфигурационната логика от основния код, което спомага за защитата и удобството при настройка;
- Съхранение на SQLite база данни – лек SQL двигател, който е подходящ за малки и средно големи приложения;
- Улесняване на настройката на различни среди – разработка, тестове и продукция, чрез използване на отделни конфигурационни файлове.

2.3. Модели за данни – models/

Директорията models/ съдържа дефинициите на основните структури на данните, използвани в Mealise. Тук се намират класове като User и Recipe, които са реализирани с помощта на Flask-SQLAlchemy. Основните модели, използвани в приложението, включват:

- **User**: Този модел управлява данните на потребителите. Той съдържа полета като id, username, email и password, като гарантира уникалността и целостта на данните чрез подходящи ограничения. Моделът също така използва Flask-Login чрез UserMixin, за да улесни управлението на сесиите.
- **Recipe**: Този модел описва рецептите, генерирани или запазени от потребителите. Той съдържа информация като идентификатор, свързан с потребителя (чрез username), списък със съставки, пълния текст на рецептата, заглавие и дата на създаване.

Използването на ORM слой (Object Relational Mapping) чрез SQLAlchemy гарантира, че всички операции с базата данни са извършвани по обектно ориентиран начин, което прави манипулацията на данните по-лесна и четима.

2.4. Маршрути и логика на приложението – routes/

Структурата на директорията routes/ е създадена така, че да раздели различните функционални зони на приложението в отделни модули. Главните файлове и тяхната роля са:

- **auth.py**: Отговаря за всичко свързано с автентикация – регистрация, вход, изход и управление на потребителските сесии. Използвайки Flask-Login, този модул гарантира, че само автентифицирани потребители имат достъп до защитени маршрути.
- **discover.py**: Този модул служи за откриване на рецепти чрез интеграция със Spoonacular API. Тук се задават логиката за търсене, филтриране и подреждане на резултатите, което осигурява богат потребителски опит.
- **home.py**: Този файл управлява началната страница на приложението, често наричана landing page. Той представя първоначалното съдържание и основна навигация, насочвайки потребителя към другите функционалности на Mealise.
- **recipes.py**: Модулът recipes отговаря за генериране на рецепти чрез вътрешната логика и интеграция с OpenAI (или неговата услуга, реализирана чрез OpenRouter API). Тук се обработват заявките за генериране, запазване, показване и изтриване на рецепти. Този модул интегрира всички предходни компоненти за да създаде завършено потребителско изживяване.

Всеки от тези модули използва Flask Blueprints, които позволяват лесно организиране и реорганизация на маршрути, като същевременно се запазва яснотата и разделението на функционалностите. Това е от ключово значение за ефикасността и поддръжката на кода в дългосрочен план.

2.5. HTML шаблони – templates/

Директорията templates/ съдържа всички HTML файлове, използвани за рендериране на потребителския интерфейс чрез Jinja2 шаблони. Тези шаблони са от основно значение за разделянето на представителната логика от бизнес логиката, предоставяйки по-ясен и модулен подход към изграждането на уеб страниците. Основните компоненти в тази директория са:

- Основният шаблон **base.html**: Това е основната структура, върху която се изграждат всички останали страници. Той съдържа елементи като хедър, футър, навигация и основна разкладка, която гарантира еднообразен вид на целия сайт.
- Специализирани страници, като **login.html**, **register.html**, **landing.html**, **discover.html** и **recipe_page.html**: Всяка от тези страници обслужва специфична функционалност, като по този начин подобрява потребителското изживяване. Например, страницата за генериране на рецепти (recipe_page.html) позволява на потребителя да въведе съставки и да види генерираната рецепта.

- Партиални шаблони: Тези малки, пренаправяеми компоненти могат да бъдат включвани в по-големите шаблони, за да се избегне повторението на кода и за улесняване на поддръжката.

Тази организация позволява на разработчиците и дизайнерите лесно да добавят или променят елементи във външния вид на приложението без да засягат функционалната му част.

2.6. Конфигурационни променливи – .env файл

Файлът .env съдържа важни конфигурационни променливи, които са ключови за сигурността и правилната работа на приложението. Сред тях се включват:

- Секретни ключове за Flask сесии и криптиране;
- URI адреси на базата данни (например, към SQLite файловете);
- API ключове за външни услуги, като Spoonacular и OpenRouter (за генериране на рецепти чрез OpenAI).

Използването на .env файла позволява на разработчиците да конфигурират приложението лесно без да модифицират кода, като по този начин се улеснява и управлението на различни среди (например, разработка, тестове и продукция).

2.7. Основната точка за стартиране – app.py

Файлът app.py е главната точка за стартиране на приложението и осъществява следните основни функции:

- Инициализация на Flask апликацията и зареждане на конфигурационните променливи от .env файла;
- Регистриране на всички blueprints, които управляват различните маршрути (auth, recipes, discover и home);
- Инициализация на разширения като Flask-SQLAlchemy и Flask-Bcrypt, осигуряващи работа с базата данни и криптиране на пароли;
- Конфигуриране на Flask-Login за управление на потребителските сесии;
- Динамично създаване на таблици в базата данни, ако те не съществуват, чрез използване на контекста на приложението.

Този подход гарантира, че с всяко стартиране на Mealise, се изпълняват всички необходими инициализации и се подготвя средата за сигурна и стабилна работа.

2.8. База данни – database.db

SQLite базата данни, обикновено именувана като mealise.db, съхранява всички данни, свързани с потребителите и рецептите в приложението. Въпреки че е файл-

базирана система, SQLite предлага достатъчно надеждност и ефективност за малки до средно големи проекти. Този тип база данни се отличава с:

- Лесна настройка и поддръжка, без нужда от сложна конфигурация или управление на сървър;
- Висока производителност при малък обем от данни;
- Отлична интеграция с Flask-SQLAlchemy, позволявайки обектно ориентиран достъп до данните.

2.9. Разширения и тяхната инициализация – extensions.py

Файлът extensions.py съдържа кода, който инициализира външни разширения, необходими за работата на Mealise. Сред основните разширения са:

- **Flask-SQLAlchemy**: За максиализиране на възможностите за достъп и манипулация на базата данни чрез ORM слой.
- **Flask-Bcrypt**: За криптиране на пароли преди тяхното запазване в базата данни, което значително повишава сигурността на потребителските данни.

Тази централна инициализация позволява лесното добавяне на други библиотеки и разширения в бъдеще, подпомагайки поддръжката на проекта и улеснявайки адаптацията спрямо нуждите на приложението.

2.10. Взаимодействие и сътрудничество между модулите

Всички компоненти на Mealise са изградени с внимание към принципите на SOLID и разделението на отговорностите. Връзката между модулите е осигурена чрез внимателно дефинирани интерфейси и разделение на логиката:

Модулите в directories като models/ и routes/ комуникират чрез базата данни и контекста на Flask приложението. Всеки път, когато е необходим достъп до данни, маршрутите използват ORM методите, дефинирани в models/.

Конфигурационните променливи, зададени в .env, оказват влияние върху начина, по който се стартира приложението в app.py, като настройват параметрите за сигурност, връзка към базата данни и API интеграциите.

Шаблоните в templates/ работят в синхрон с логиката в routes/. Чрез използването на Jinja2 механизъм се гарантира, че данните, създадени от backend логиката, се визуализират правилно в потребителския интерфейс.

Тази интегрирана архитектура позволява на Mealise да бъде едновременно модулен и гъвкав. При добавяне на нови функционалности – независимо дали става дума за нови API интеграции, допълнителни модели за данни или разширение на съществуващите шаблони – промените могат да се реализират без да се нарушава

стабилността на системата. Разработчиците могат лесно да проследяват потока от данни и да проследяват грешки, което води до по-бързо откриване и отстраняване на потенциални проблеми.

3. Главният файл app.py и инициализация на приложението

Файлът `app.py` в проекта Mealise е централната точка за стартиране и конфигуриране на уеб приложението. Той изпълнява ключова роля по зареждане на настройките, свързване на структурни компоненти и подготовка на средата за работа. В тази секция ще разгледаме подробно основните функционалности, реализирани в този файл, с акцент върху процеса на инициализация, управлението на конфигурациите и настройката на разширенията, които осигуряват сигурността и стабилността на приложението.

3.1. Зареждане на конфигурационни променливи чрез dotenv

В началото на `app.py` се използва библиотеката `dotenv`, която зарежда конфигурационни променливи от `.env` файл. Това е обичайна практика в съвременните уеб проекти, която осигурява следните преимущества:

- **Сигурност:** Чувствителни данни като ключове и пароли не се записват директно в кода.
- **Гъвкавост:** Лесно превключване между различни среди (разработка, тест, продукция) чрез смяна на `.env` конфигурацията.
- **Улеснена поддръжка:** Промените по конфигурацията не изискват редактиране на основните с изходен кодове файлове.

```
from dotenv import load_dotenv
…
load_dotenv()
```

С функцията load_dotenv() в паметта на приложението се зареждат променливи като FLASK_SECRET_KEY и DATABASE_URI, използвани по-нататък за конфигурация.

3.2. Функцията create_app() – сърцето на инициализацията

В основата на `app.py` стои функцията `create_app()`, която създава и конфигурира екземпляра на Flask приложението. Този подход е широко препоръчван във Flask проекти, тъй като позволява по-гъвкаво управление, лесно създаване на тестови приложения и поддръжка на множество екземпляри.

Основните стъпки в `create_app()` включват:

3.2.a. Създаване на инстанция на Flask:

```
app = Flask(__name__)
```

3.2.b. Настройка на тайните ключове и база данни:

- app.secret_key осигурява сигурността на сесийната информация във Flask.
- SQLALCHEMY_DATABASE_URI посочва адреса на базата данни (в случая SQLite, дефинирана в .env).
- SQLALCHEMY_TRACK_MODIFICATIONS се изключва за оптимизация.

```
app.secret_key = os.getenv('FLASK_SECRET_KEY')
app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('DATABASE_URI')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

3.2.c. Регистриране на Blueprints:

Flask Blueprints са модулен начин за организиране на маршрутите и логиката. В Mealise те са разделени по функционалности – auth_bp за автентикация, recipes_bp за рецепти, home_bp за начална страница и spoonacular_bp за интеграция със Spoonacular API.

Регистрирането им в основното приложение позволява централизирана маршрутизация:

```
app.register_blueprint(auth_bp)
app.register_blueprint(recipes_bp)
app.register_blueprint(home_bp)
app.register_blueprint(spoonacular_bp)
```

3.2.d. Инициализация на разширения:

Проектът използва външни библиотеки като Flask-SQLAlchemy (за работа с БД) и Flask-Bcrypt (за криптиране на пароли). В extensions.py те се инициализират глобално, а тук се свързват с конкретното приложение:

```
db.init_app(app)
bcrypt.init_app(app)
```

3.2.e. Настройка на Flask-Login мениджъра:

Flask-Login се грижи за управлението на потребителските сесии – вход, изход и текущ статус. В `create_app()` се създава и конфигурира инстанция на `LoginManager`:

```
from flask_login import LoginManager

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'auth.login'
```

login_view указва крайна точка, към която потребителят се пренасочва при опит за достъп до защитени ресурси без влизане.

3.2.f. Функция за зареждане на потребител по ID:

Flask-Login изисква дефиниране на `user_loader` функция, която преобразува ID (запазено в сесията) в обект на потребителя (от базата данни). Това гарантира, че при всяка HTTP заявка се осигурява достъп до текущия потребител.

```
@login_manager.user_loader
def load_user(user_id):
    return db.session.get(User, int(user_id))
```

3.2.g. Създаване на базата данни при нужда:

Ключов момент за удобство при разработка и разгръщане е автоматичното създаване на базата данни и всички таблици, дефинирани в моделите, ако те липсват:

```
with app.app_context():
    db.create_all()
```

Това се прави само веднъж в контекста на приложението и премахва необходимостта от ръчна настройка при първо стартиране.

3.3. Мениджърът за вход и неговата роля

Мениджърът за вход (LoginManager) е съществен компонент за контрол на достъпа и сигурността. Той изпълнява следните функции:

- Осигурява **декоратори** като @login_required, които предпазват защитени маршрути, като препращат неавтентифицираните потребители към страницата за вход.
- Зарежда и съхранява текущата сесия на автентикиран потребител, поддържайки свързаност и статус при различни HTTP заявки.
- Позволява лесно дефиниране на логика за зареждане на потребителския обект чрез функцията user_loader.
- Основава се на app.secret_key, което гарантира сигурността на сесийните данни.

В Mealise този мениджър е настроен така, че при неуспешен достъп до защитена страница, потребителят се пренасочва към /login (по маршрута в auth_bp).

3.4. Стартиране на приложението с debug режим

В низовата част на app.py се дефинира условие, което позволява пускането на приложението в debug режим, когато файлът е изпълнен директно:

```
if __name__ == "__main__":
    app = create_app()
    app.run(debug=True)
```

Стартирането с debug=True включва:

- Автоматично презареждане при промяна на кода в проекта.
- Показване на детайлна информация при възникнали грешки, което улеснява отстраняването им.
- Забрана за използване в продукционна среда, тъй като разкрива чувствителна информация.

3.5. Работа със сесии, сини маршрути и мобилност

Благодарение на организирането на Flask Blueprints и конфигурацията в app.py, приложението Mealise поддържа ясна и логична маршрутизация, която позволява лесно мащабиране и добавяне на нови функционалности.

- Всеки blueprint обслужва отделен набор от URL адреси, което подобрява четливостта и устойчивостта на кода.
- Инициализацията на разширенията и мениджъра на вход в create_app() осигурява централизирано управление и обединяване на функционални компоненти.
- Автоматичното създаване на базата данни осигурява, че след всяко обновяване или разгръщане няма нужда от отделна миграция, подходящо за малки и средни приложения.

4. Модели на данни: User и Recipe

В тази секция ще разгледаме детайлно основните модели на данни в приложението Mealise — **User** и **Recipe**. Тези модели са основата за съхранение и управление на потребителската информация и генерираните рецепти. Описанието обхваща структурата на базата данни, типовете данни на колоните, връзките между моделите, както и интеграцията с Flask-Login чрез наследяване на UserMixin и методите, осигуряващи правилното управление на автентикация и сесии.

4.1. Основен модел User

4.1.a. Основна цел

Моделът User съхранява данните за регистрираните потребители на приложението. Той се използва за идентификация, автентикация и управление на сесиите в Mealise.

4.1.b. Наследяване от UserMixin

Класът наследява от UserMixin (част от модула flask_login), което осигурява стандартни методи и свойства, необходими за работата на Flask-Login, като например:

- get_id() — връща уникалния идентификатор за потребителя, използван от Flask-Login за сесийното управление.
- Свойствата is_authenticated, is_active и is_anonymous, които обозначават статуса на потребителя и дали сесията е валидна.

Това наследяване максимално улеснява имплементацията на системата за вход и осигурява съвместимост със стандартните механизми на Flask-Login.

4.1.c. Колони (атрибути) в базата данни

Колона	Тип данни	Описание	Ограничения
id	Integer (primary key)	Уникален идентификатор на потребителя, автоматично генериран.	Първичен ключ, уникален
username	String(120)	Потребителско име, използвано за логин и показване в приложението.	Уникално, не може да е празно
email	String(120)	Имейл адрес на потребителя.	Уникален, не може да е празно
password	String(120)	Хеширана парола на потребителя.	Не може да е празно

4.1.d. Технически бележки:

- Типът String(120) дефинира максималната дължина на текстовото поле, което е достатъчно за повечето имена и имейли.
- Паролата не се съхранява в явен текст, а е криптирана с помощта на Flask-Bcrypt, което гарантира безопасност при съхранение.
- Уникалните ограничения на username и email предотвратяват създаването на дублирани акаунти и поддържат цялостта на данните.

4.1.e. Методи и свойства

Методът __repr__() предоставя удобен низов представител на обекта, полезен за дебъгване и логване:

```
def __repr__(self):
    return f'<User {self.username}>'
```

Методът get_id() връща ID-то на потребителя като стринг, което е изискуемо от Flask-Login:

```
def get_id(self):
    return str(self.id)
```

Свойствата за автентикация се дефинират като *read-only* (чрез @property), стандартно връщайки True или False в зависимост от статуса, като по този начин Flask-Login може да определи дали сесията е валидна.

4.2. Модел Recipe

4.2.a. Основна цел

Моделът Recipe описва отделните рецепти, създадени и съхранени от потребителите в приложението. Той има за цел да структурира информацията за всяка рецепта по начин, подходящ за последващо визуализиране и манипулация.

4.2.b. Колони (атрибути) в базата данни

Колона	Тип данни	Описание	Ограничения
id	Integer (primary key)	Уникален идентификатор на рецептата.	Първичен ключ, уникален
username	String(150)	Потребителското име, свързано с автора на рецептата (чужд ключ към User).	Чужд ключ, задължително
ingredients	String(500)	Текстово поле, съдържащо списък със съставки (обикновено с префикс --).	Задължително, за по-удобно парсиране
recipe	String(1000)	Текст с инструкции за приготвяне на рецептата.	Задължително
title	String(150)	Заглавие или име на рецептата.	Задължително
created_at	DateTime	Дата и час на създаване на рецептата.	Задължително

4.2.c. Технически бележки:

Връзката между Recipe и User е осъществена чрез username като чужд ключ към User.username. Това позволява лесен достъп до всички рецепти, създадени от даден потребител.

Полето ingredients е съхранено като дълъг текст с отделни редове (например, префикс с --), което улеснява парсирането и визуализацията под формата на списък.

created_at използва DateTime, което позволява сортиране по дата и показване на хронологията.

4.3. Връзки между моделите

- Връзката между User и Recipe е базирана на атрибут username в модела Recipe като **foreign key** към User.username.
- Тази връзка осигурява, че всяка рецепта е свързана с един потребител — автора на рецептата.
- Този подход улеснява заявки като търсене на всички рецепти, създадени от конкретен потребител, и осигурява целостта на данните, като не позволява рецепти без свързан потребител.

4.4. Методи за проверка на автентикация и сесии

В модела User, наследяването от UserMixin осигурява необходимата инфраструктура за Flask-Login, но за по-голяма сигурност и коректност се дефинират някои важни параметри:

Свойство/Метод	**Описание**	**Върнат тип/стойност**
is_authenticated	Връща дали настоящият потребител е автентикиран (влязъл в системата).	True при валиден вход
is_active	Показва дали акаунтът е активен и не е блокиран/деактивиран.	По подразбиране True
is_anonymous	Отразява дали потребителят е анонимен (не е влязъл в системата).	Връща False за регистриран потребител
get_id()	Връща уникален идентификатор на потребителя за сесията.	Стринг с ID на потребителя

Тази функционалност се използва от **Flask-Login** за контрол на достъпа и управление на сесии във всички защитени части от приложението.

4.5. Сигурност и типове данни

Полетата като password се съхраняват като криптирани хешове чрез библиотеката Flask-Bcrypt. Никога не се пази парола в ясен текст.

Всички текстови полета използват ограничена дължина със SQLAlchemy String — това предотвратява неочаквано големи записи и потенциални проблеми със сигурността.

Полетата, които са ключови (като username и email), са маркирани като уникални и задължителни с nullable=False, за да гарантират правилна структура и ограничение на база данни.

5. Модул за автентикация (auth.py)

Модулът за автентикация в Mealise е ключов компонент, който отговаря за регистрацията, влизането и изхода на потребителите. Този модул управлява всички операции, свързани с потребителските данни и поведението на сесиите, като се основава на сигурни практики за валидиране на вход, криптиране на пароли и директно използване на Flask-Login за контролиране на достъпа до защитени ресурси.

5.1. Основни маршрути и техните функции

В модулът auth.py се дефинират няколко основни маршрута, сред които са:

- **/register** – маршрут за нова регистрация на потребител;
- **/login** – маршрут за вход в системата;
- **/logout** – маршрут за изход от системата.

Всеки от тези маршрути има свои специфични задачи и отговаря за различни аспекти на потребителската автентикация.

5.1.a. 1. Регистрация (/register)

При посещение на страницата за регистрация, потребителят има възможност да въведе данни като потребителско име, имейл и парола (заедно с потвърждение на паролата). Основната логика в този маршрут включва следните стъпки:

- **Валидация на входящите данни:**
  При получаване на POST заявка се проверява дали всички задължителни полета (username, email, password) са попълнени. Ако някое от тях липсва, се извежда съобщение за грешка чрез метода flash, което уведомява потребителя, че някои данни са задължителни.
- **Съвместимост на паролите:**
  Преди създаването на нов акаунт се проверява дали въведената парола съвпада с потвърдената версия. При разминаване между двете стойности, потребителят се уведомява, че паролите не съвпадат, което предотвратява потенциални грешки при впоследствие влизане.
- **Проверка за съществуващи потребители:**
  За да се избегне дублиране на потребителски имена и имейл адреси, преди записването на новия потребител в базата данни, се извършва заявка, която

проверява дали някой вече съществува с въведения username или email. Ако се открие такъв потребител, се извежда подходящо съобщение за грешка и регистрацията се спира.

- **Криптиране на пароли с Flask-Bcrypt:**
  Преди съхранението на паролата в базата данни, тя преминава през процеса на хеширане с помощта на Flask-Bcrypt. Това гарантира, че дори при неоторизиран достъп до базата данни, паролата няма да бъде изложена в ясен текст. Хешираният резултат се записва вместо оригиналната парола.
- **Запис на новия потребител и автоматично логване:**
  При успешно преминаване през всички проверки, се създава нов обект на модела User с въведените данни и хеширана парола. След това този потребител се запазва в базата данни и се използва Flask-Login за автоматично влизане, като по този начин потребителят получава валидна сесия веднага след регистрация.
- **Обработка на грешки и съобщения:**
  В случай на неуспешна регистрация (липса на задължителни данни, несъответствие на пароли или вече съществуващ потребител) се използва flash съобщенията, насочващи потребителя към повторна регистрация. Това позволява лесно откриване и коригиране на грешки от страна на потребителя.

  Примерната логика изглежда по следния начин:

- Получаване на данни чрез request.form.
- Извършване на проверки (наличие на username, email, password и съвпадение на паролите).
- Използване на User.query.filter((User.username == username) | (User.email == email)).first() за проверка на съществуващи потребители.
- Хеширане на паролата чрез bcrypt.generate_password_hash(password).decode('utf-8').
- Записване и комитване на новия потребител в базата данни.
- Използване на login_user(new_user) от Flask-Login за установяване на сесията.

5.1.b. 2. Вход (/login)

Входът в системата е също толкова важен процес за осигуряване на достъп само на валидни потребители. В логиката на маршрута за вход се предприемат следните действия:

- **Получаване на входящите данни:**
  Потребителят въвежда своя username и парола, които се предават чрез POST заявка към маршрута.
- **Проверка за съществуване на потребителя:**
  Използва се заявка към базата данни за намиране на потребител с дадения

username. Ако потребителят не съществува, потребителят получава съобщение за невалидни данни.

- **Декриптиране и проверка на паролата:**
  При намерен потребител, въведената парола се сравнява с хешираната парола, съхранявана в базата данни. Функцията bcrypt.check_password_hash(user.password, password) извършва тази проверка. При съвпадение потребителят се логва, а при неуспех – се извежда съобщение за грешка.

- **Управление на сесиите чрез Flask-Login:**
  При валиден вход, функцията login_user() от Flask-Login се извиква, за да се създаде сесия за потребителя и да се запази неговата автентикация за бъдещи заявки. Освен това, логиката на маршрута позволява пренасочване след успешен вход – ако в URL-а има параметър "next", потребителят се препраща към тази страница. В противен случай, той се насочва към началната страница за работа с рецепти.

- **Обработка на невалидни опити за вход:**
  Ако предоставените данни не съвпадат с никакъв запис в базата данни или паролата не минава проверката, се извежда съобщение за грешка чрез flash съобщения. Това помага за индикация на потребителя какъв е проблемът и го насърчава да опита отново.

  Примерната логика в маршрута за вход е структурирана така:

- Извличане на данните от request.form (username и password).
- Намиране на потребителя с помощта на User.query.filter_by(username=username).first().
- Извикване на bcrypt.check_password_hash(user.password, password) за сравнение на паролата.
- При успех, извикване на login_user(user), flash съобщение „Login successful!” и пренасочване към следващата страница (чрез next параметър ако е наличен).

5.1.c. 3. Изход (/logout)

Изходът от системата е сравнително опростен, но жизненоважен за сигурността при затваряне на сесията:

- **Логика за изход:**
  При извикване на маршрута за изход, функцията logout_user() от Flask-Login премахва сесията на настоящия потребител. След това потребителят получава flash съобщение, което го уведомява, че е излязъл успешно, и се пренасочва към страницата за вход.

- **Защита на маршрута:**
  Маршрутът е защитен чрез декоратора @login_required. Това означава, че само автентифицирани потребители могат да го използват. Ако някой се опита да

достъпи тази точка без да бъде логнат, Flask-Login го пренасочва към логин страницата.

Примерната имплементация включва:

- Извикване на logout_user().
- Flash съобщение „You have been logged out.“
- Пренасочване към URL адреса, свързан с логина.

5.2. Използване на Flask-Login за управление на сесии и защита

Като сърцевина на сигурността в този модул е Flask-Login, който предлага готови инструменти и методи за работа с потребителски сесии:

- **User Loader функцията:**
  С помощта на декоратора @login_manager.user_loader се дефинира функция, която зарежда потребителски обект въз основа на неговия уникален идентификатор. Тази функция се извиква автоматично от Flask-Login при всяка заявка, така че текущият потребител винаги е наличен чрез променливата current_user.
- **Декораторът @login_required:**
  Този декоратор ограничава достъпа до защитени маршрути. Ако потребителят не е логнат, той се пренасочва към страницата за вход с указание за необходимостта от автентикация.
- **Запазване и управление на сесиите:**
  С помощта на сесийните механизми на Flask, данни за потребителите се съхраняват сигурно с използване на секретен ключ (app.secret_key), като така се гарантира цялостта и сигурността на данните през времето на активната сесия.

5.3. Детайли за криптиране и валидиране на пароли

Криптирането на пароли е съществен аспект от сигурността. В Mealise се използва Flask-Bcrypt, която предлага лесен и ефективен начин за генериране на сигурни хешове:

- Преди паролата да бъде съхранена, тя се преобразува чрез bcrypt.generate_password_hash(password) и получената стойност се декодира във формат utf-8.
- При вход, използва се bcrypt.check_password_hash(stored_hash, password) за сравнение на въведената парола с хешираната стойност. Този процес осигурява, че оригиналният текст на паролата никога не се съхранява или сравнява директно, като така се минимизира рискът от компрометиране на личната информация.

5.4. Обработка на грешки и логване на инциденти

При регистрация и вход се използва механизъм за обработка на грешки, който гарантира, че:

- Във всеки един случай, при който входящите данни не отговарят на изискванията (липса на задължителни полета, несъответствие на пароли, невалиден потребител или грешна парола), се извеждат подходящи flash съобщения.
- Чрез логиране на важни събития (например опит за вход с неправилни данни) се подпомага откриването и отстраняването на бъгове и потенциални атаки.
- Потребителят получава ясни инструкции за корекция или повторно опитване, като се гарантира, че опитът за вход не остава без реакция и системата реагира адекватно на всяка непредвидена ситуация.

5.5. Безопасност и стандарти при разработка

Модулът за автентикация в Mealise следва редица добро практики и стандарти:

- **Разделение на отговорностите:**
  Всяка функция и маршрут имат ясна и дефинирана отговорност – от валидацията до криптирането и пренасочването, което улеснява поддръжката и разширението на модула.
- **Използване на външни библиотеки със сигурност:**
  С помощта на признати библиотеки като Flask-Bcrypt и Flask-Login се гарантира, че криптирането и управлението на сесиите покриват съвременните стандарти за сигурност.
- **Проверка на съществуващи записи:**
  При регистрация винаги се извършват проверки за уникалност на потребителските имена и имейли, което предпазва от създаване на дублирани акаунти и поддържа целостта на базата данни.
- **Централизиран контрол и лесна дебъгване:**
  Системата използва централизирана фабрика за създаване на приложението (create_app()), където се инициализират всички компоненти, свързани с автентикацията. Това осигурява ясно разделение между логиката на сесиите, маршрутизацията и конфигурацията на разширенията.

6. Интеграция с Spoonacular API за откриване на рецепти (discover.py)

В този раздел ще разгледаме подробно модула discover.py, който реализира функционалностите за извличане, филтриране и показване на рецепти чрез интеграция със Spoonacular API – една от популярните външни услуги за рецензирани кулинарни

рецепти. Модулът използва Flask Blueprint структура за организиране на маршрутите и логиката, както и requests библиотеката за комуникация с външния API. Ще опишем основните методи, параметри на заявките, динамичното зареждане на рецепти и начина, по който потребителят взаимодейства с интерфейса.

6.1. Blueprint структура и организация на модула

Файлът discover.py дефинира Flask Blueprint с име spoonacular_bp, който обособява всички маршрути за откриване и търсене на рецепти в отделен логически модул. Това позволява:

- Ясно разделяне на отговорностите на кода – интеграцията със Spoonacular е капсулирана и лесна за поддръжка;
- Лесно регистриране в главния файл app.py и интеграция в цялото приложение;
- Използване на общ префикс или групиране на маршрути, което подобрява структурираността на URL адресите.

Определянето на Blueprint става чрез следния кодов фрагмент:

```
from flask import Blueprint

spoonacular_bp = Blueprint('spoonacular', __name__)
```

След това маршрутите се декларират с декораторите @spoonacular_bp.route(...).

6.2. Комуникация със Spoonacular API чрез requests

За извличане на данни от Spoonacular API се използва популярната Python библиотека requests, която изпраща HTTP GET заявки към публичния API endpoint на Spoonacular. Това позволява динамично изтегляне на рецепти според зададените критерии и параметри.

Ключови моменти:

- API ключът се зарежда от .env файла чрез os.getenv, което гарантира сигурност, като не се излага публично.
- Използва се endpointът /recipes/complexSearch на Spoonacular, който поддържа разширено търсене с параметри за филтриране.
- Параметрите на заявката включват брой рецепти за извличане, офсет (за постранично зареждане), ключова дума, тип кухня, диета и критерии за сортиране.
- Отговорът от API е в JSON формат, като основният интерес е в масива results, съдържащ списък с рецепти.

Примерен код за извличане на рецепти:

```
import requests
import os
from dotenv import load_dotenv

load_dotenv()
API_KEY = os.getenv("SPOONACULAR_API_KEY")

def fetch_recipes(page=1, query='', cuisine='', diet='', sort=''):
    offset = (page - 1) * 10
    url = "https://api.spoonacular.com/recipes/complexSearch"
    params = {
        "number": 10,
        "offset": offset,
        "addRecipeInformation": True,
        "apiKey": API_KEY,
        "query": query,
        "cuisine": cuisine,
        "diet": diet,
        "sort": sort
    }
    response = requests.get(url, params=params)
    return response.json().get("results", [])
```

6.3. Обработка на заявки – GET и POST методи

6.3.a. Маршрут /discover

Този маршрут обслужва както GET, така и POST заявки и реализира основния интерфейс за търсене и показване на рецепти. Основната функционалност:

- При GET заявка зарежда начална страница със списък рецепти по подразбиране (например първата страница без филтри).
- При POST заявка получава потребителски въведени стойности за търсене, вид кухня, диета и сортиране чрез формуляр.
- Извиква функцията fetch_recipes() с въведените или подразбиращите се параметри, след което подава списъка от рецепти към HTML шаблон discover.html.

Примерен код:

```
from flask import render_template, request

@spoonacular_bp.route('/discover', methods=['GET', 'POST'])
def unified_discover():
    page = 1
    query = ''
    cuisine = ''
    diet = ''
```

```
    sort = ''

    if request.method == 'POST':
        query = request.form.get('query', '')
        cuisine = request.form.get('cuisine', '')
        diet = request.form.get('diet', '')
        sort = request.form.get('sort', '')

    recipes = fetch_recipes(page, query, cuisine, diet, sort)
    return render_template('discover.html', recipes=recipes, current_page=
page,
                           query=query, cuisine=cuisine, diet=diet, sort=s
ort)
```

Параметрите на зареждането на рецепти се задават през POST формата, което позволява зарастване на резултатите според желанията на потребителя.

6.4. Динамично зареждане на рецепти с бутон "Зареди още"

За по-добро потребителско изживяване и избягване на презареждания на цялата страница, в Mealise е реализирана функционалност за динамично зареждане на още рецепти:

- Създаден е отделен маршрут /discover/load_more, който обслужва заявки с метод GET.
- Този маршрут приема параметри за страница (page), ключова дума, кухня, диета и сортиране от URL параметрите.
- Извлича нов списък с рецепти за посочената страница, изпращайки резултата под формата на HTML фрагмент (partial шаблон _recipes.html).
- От страната на клиента бутонът "Load More Recipes" изпраща AJAX заявка към този маршрут с необходимите параметри.
- При получаване на отговора новите рецепти се добавят в края на текущия списък, без презареждане на цялата страница.
- Бутонът се презарежда с нов номер на страница, позволявайки многократно добавяне на следващи порции с рецепти.

Имплементация на маршрута:

```
@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)
    return render_template('partials/_recipes.html', recipes=recipes)
```

Във фронтенда при бутон "Load More" се използва JavaScript код, който изпраща динамично GET заявка с параметрите и добавя получения HTML към основния контейнер с рецепти.

6.5. Параметри за търсене и филтриране

Потребителят може да зададе няколко ключови параметъра, които влияят на търсенето и показването на рецепти:

Параметър	**Описание**	**Стойности/Възможности**
query	Ключова дума за търсене в заглавието и описанието на рецептите	Текст (напр. "паста", "шоколад")
cuisine	Вид кухня	Примерно: "Italian", "Mexican", "Indian", "Thai"
diet	Вид диета	Възможности: "vegetarian", "vegan", "gluten free"
sort	Критерий за сортиране	Възможности: "popularity", "healthiness", "price"

Тези филтри се изпращат до API-то и намаляват резултатите според желанията на потребителя, което прави търсенето по-прецизно и полезно.

6.6. Представяне на резултатите в discover.html

Резултатите от API-то се визуализират чрез динамичен HTML шаблон, при който се показват основни атрибути на рецептата:

- Заглавие;
- Време за приготвяне (ако е налично);
- Кратко описание или резюме;
- Бутон съдържащ линк към източника на рецептата (sourceUrl).

Използват се условни конструкции, за да се покажат само наличните данни (например изображение или време за приготвяне). Шаблонът позволява лесно добавяне на още данни или промени в дизайна.

7. Модул за домашна страница (home.py) и базови шаблони

7.1. Blueprint home.py – управление на началната страница

В проекта Mealise файловият модул home.py осъществява функционалността, свързана с основната начална страница (landing page) на уеб приложението. Той е дефиниран като **Flask Blueprint** с име home_bp, което позволява лесно регистриране и интеграция в главното приложение чрез механизма на blueprints.

Основната роля на home.py е да предостави началната точка за потребителите, която:

- Представя приветствено съобщение и кратка информация за приложението;
- Предлага две важни опции за навигация към основните функционалности: **Generate a Recipe** (генериране на рецепта) и **Discover Recipes** (откриване на рецепти);
- Прави плавен преход към следващите стъпки в използването на Mealise.

Маршрутът дефиниран в home.py е прост и ясен:

```
from flask import Blueprint, render_template

home_bp = Blueprint('home', __name__)

@home_bp.route('/')
def landing():
    return render_template('landing.html')
```

Това означава, че при отваряне на кореновия URL адрес на приложението (/), потребителите получават визуално оформление, което ги насочва към основните действия. Няма сложна логика или обмен на данни — само рендериране на HTML шаблон landing.html.

7.2. landing.html – приветствена страница с навигация

Шаблонът landing.html използва наследяване от базовия шаблон base.html (описан по-долу), с което гарантира единен външен вид и структура на приложението. Тази страница съдържа:

- *Заглавие* („Welcome to Mealise“), което първоначално представя апликацията;
- *Кратък описателен текст*, който обяснява накратко целта и ползите от Mealise;

- *Два големи бутона* за навигация, които препращат към ключовите функционалности:
    - **Generate a Recipe** – води към страницата за генериране на рецепти, където потребителят въвежда наличните съставки;
    - **Discover Recipes** – насочва към интерфейса за търсене и разглеждане на рецепти от Spoonacular API.

Тези бутони са реализирани чрез стандартни бутонни HTML елементи с класове от Bootstrap, което гарантира, че изглеждат съвременно и са адаптивни за различни устройства.

Примерен ключов HTML код във landing.html:

```
{% extends 'base.html' %}
{% block content %}
<section class="…" style="…">
  <div class="…">
    <h1 class="…">Welcome to Mealise</h1>
    <p class="…">Your smart assistant for generating delicious recip
es based on your preferences and pantry.</p>
    <a href="{{ url_for('recipes.gen_recipe_page') }}" class="…">Gen
erate a Recipe</a>
    <a href="{{ url_for('spoonacular.unified_discover') }}" class="…
">Discover Recipes</a>
  </div>
</section>
{% endblock %}
```

7.3. base.html – основна структура и оформление на интерфейса

Шаблонът base.html в templates/ е фундаменталният HTML документ върху който се надграждат всички останали страници на Mealise. Той съдържа основната разметка на интерфейса и включва използването на CSS framework Bootstrap 5, който осигурява адаптивност, модерен дизайн и удобна работа с компоненти и оформление.

7.3.a. Бокова странична навигация (Sidebar)

На лявата страна на страницата (колона с фиксирана ширина) е разположена вертикална навигационна лента (<nav>):

7.3.b. Основно съдържание (Main Content Area)

Втората по-голяма колона е резервирана за динамичното съдържание на страницата, което различните шаблони поставят в Jinja2 блока {% block content %}. Това

позволява всяка страница да дефинира своя уникален UI, но в рамката и структурата на main layout-a.

7.3.c. Условно показване на потребителска информация и бутони

В base.html се използват Jinja2 условни конструкции, които позволяват:

Показване на приветствие с потребителско име, ако потребителят е логнат:

```
{% if current_user.is_authenticated %}
    <p class="px-3 fw-lighter">Welcome, <b>{{ current_user.username }}</b>
</p>
{% endif %}
```

Показване на бутон за изход (Logout), ако потребителят е автентикиран, или бутон за вход (Login), ако не е:

```
{% if current_user.is_authenticated %}
    <a class="nav-link" href="{{ url_for('auth.logout') }}">Logout</a>
{% else %}
    <a class="nav-link" href="{{ url_for('auth.login') }}">Login</a>
{% endif %}
```

Това условно представяне подобрява потребителското изживяване и осигурява контекстно навигиране.

7.4. Използване на Jinja2 шаблонната система

Mealise разчита на мощта на **Jinja2**, стандартния шаблонен език, който Flask използва за динамично генериране на HTML. Това позволява:

- Въвеждане на Python променливи и данни директно в HTML с шаблонни маркери ({{ }});
- Условно рендиране с логика ({% if %}, {% for %}, {% block %} и т.н.);
- Наследяване на шаблони — базов base.html, който дефинира основната рамка, и страници, които разширяват тази рамка чрез {% extends "base.html" %};
- Разделяне на повторно използваеми HTML компоненти (partials), което улеснява поддръжка и модификации.

По този начин всеки шаблон използва {% block content %}, където поставя своето уникално съдържание в общата рамка на страницата.

Следващи части на документацията ще разгледат по-подробно други модули и техните рендериращи шаблони, както и взаимодействието между бекенд и фронтенд логиката.

8. Генериране и управление на рецепти (recipes.py)

Модулът за рецепти (recipes.py) е ключова част от Mealise, тъй като отговаря за цялостното управление на потребителски генерирани рецепти. Този модул обединява логиката за генериране на нови рецепти чрез OpenAI модел, съхраняването им в базата данни, извличането им с помощта на постранично зареждане (pagination) и изтриването им, като гарантира, че достъпът до всяка рецепта е защитен и е свързан само с потребителя, който е създал рецептата. В този раздел ще разгледаме подробно основните функционалности на recipes.py и начина, по който те съчетават различни технологии и добри практики за разработка на уеб приложения.

8.1. Генериране на рецепти чрез интеграция с OpenAI

Една от основните функции на модула е генерирането на рецепти на база въведени от потребителя съставки. Процесът е следният:

```
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.
Format your answer exactly like this:

$part-title
[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
"""
    response = client.chat.completions.create(
        model="mistralai/mistral-7b-instruct",
        temperature=0,
        messages=[
            {"role": "user", "content": prompt}
        ]
    )
```

```
    return response.choices[0].message.content



@recipes_bp.route('/generate_recipe', methods=['POST'])
def generate_and_save_recipe():
    if not current_user.is_authenticated:
        return redirect(url_for('auth.login'))

    ingredients = request.form['ingredients']
    raw_recipe_text = generate_recipe(ingredients)
    print(raw_recipe_text)

    parts = raw_recipe_text.split("$part-")
    title = ""
    parsed_ingredients = ""
    instructions = ""
    for part in parts :
        if part.startswith("title") :
            title = part.replace("title", "").strip()
        elif part.startswith("ingredients") :
            parsed_ingredients = part.replace("ingredients", "").strip()
        elif part.startswith("instructions") :
            instructions = part.replace("instructions", "").strip()

    print(title, ingredients, instructions)

    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()

    return redirect(url_for('recipes.gen_recipe_page'))
```

- **Събиране на входни данни:**
  Потребителят попълва формата, намираща се в шаблона recipe_page.html, като въвежда съставки в текстово поле (textarea). Често се препоръчва въвеждане на съставките с кратки описания, например "яйца, брашно, мляко", за да се получи по-точен резултат.
- **Обработка на заявката:**
  При изпращане на формата, данните се засичат в POST заявка към маршрута /generate_recipe. Този маршрут първо проверява дали потребителят е влязъл в

системата, използвайки декоратора @login_required, който гарантира, че само автентифицирани потребители могат да генерират нови рецепти.

- **Извикване на OpenAI API:**
  Функцията generate_recipe(ingredients) конструира подканващ текст (prompt), съдържащ изброените съставки и инструкции за форматиране на резултата. API заявката към OpenAI (чрез OpenRouter) е конфигурирана с фиксирани параметри, като температурата на модела е зададена на 0, за да се гарантира детерминистичен отговор и последователност в генерираните резултати.

- **Парсване на резултатите:**
  Отговорът от OpenAI моделът съдържа текст, който е структуриран с предварително определени разделители – например, "$part-title", "$part-ingredients" и "$part-instructions". Модулът анализира текста, като разделя отделните секции и отстранява излишните префикси. Това парсване позволява лесно извличане на заглавието, списъка със съставки и подробните инструкции за приготвяне, които после се използват за визуализиране в интерфейса.

- **Създаване на нова рецепта:**
  След успешно парсване на резултатите, нов обект от тип Recipe се създава с помощта на данните – потребителското име (current_user.username), парсирани съставки, инструкции и заглавие. Допълнително времевият печат (timestamp) се задава автоматично с помощта на SQLAlchemy функцията db.func.current_timestamp().

- **Записване в базата данни:**
  Новата рецепта се добавя към сесията на базата данни, като след това се извиква db.session.commit(), за да се запази промените постоянно. Този процес осигурява, че всяка генерирана рецепта става част от личната колекция на потребителя.

8.2. Постранично зареждане (Pagination) и визуализация

За да се гарантира добро потребителско изживяване при разглеждане на многобройни рецепти, системата използва техника за постранично зареждане:

- **Извличане на рецепти:**
  В маршрута /recipes се използва функцията paginate на SQLAlchemy, за да се извлекат рецепти от базата данни, създадени от текущия потребител. Параметрите включват номера на страницата (page) и брой записи на страница (per_page), например по 4 рецепти на страница. Запитването е формулирано чрез метод като
  Recipe.query.filter_by(username=username).order_by(Recipe.created_at.desc()).paginate(page=page, per_page=per_page).

- **Предаване на данните към шаблона:**
  Резултатът от заявката – обектът с рецепти, който съдържа информация за

текущата страница, общия брой страници, и други – се подава към шаблона recipe_page.html. В този шаблон с помощта на Jinja2 се визуализират рецептите като списък от карти (card), всяка от които съдържа заглавие, списък със съставки и инструкции.

- **Контроли за навигация:**
  Страниците са снабдени с бутони за навигация, които позволяват на потребителя да преминава към предишна или следваща страница. При клик на тези бутони URL параметрите, като Връзка към предишна или следваща страница, се актуализират, а новите данни се подлагат на заявка към базата, без да е необходимо презареждане на цялата страница.

8.3. Изтриване на рецепти с контрол на потребителския достъп

За да се предотвратят нежелани действия и да се гарантира, че потребителите могат да управляват само своите рецепти, модулът включва функция за изтриване:

```
@recipes_bp.route('/delete_recipe/<int:recipe_id>', methods=['POST'])
@login_required
def delete_recipe(recipe_id):
    recipe = Recipe.query.get_or_404(recipe_id)

    # Make sure the recipe belongs to the logged-in user
    if recipe.username != current_user.username:
        return "Unauthorized", 403

    db.session.delete(recipe)
    db.session.commit()
    return redirect(url_for('recipes.gen_recipe_page'))
```

Маршрутът /delete_recipe/<int:recipe_id> се използва за изтриване на конкретна рецепта. Този маршрут е защитен с декоратора @login_required, което означава, че само влезли потребители могат да извършват тази операция.

Преди изтриването модулът извлича рецептата от базата данни чрез Recipe.query.get_or_404(recipe_id). След това следва валидация, която сравнява username на рецептата с current_user.username. Ако двата не съвпадат, това означава, че потребителят няма право да изтрие рецептата, създадена от друг потребител. В такъв случай се връща съобщение "Unauthorized" с HTTP статус 403.

При успешна проверка, рецептата се изтрива чрез db.session.delete(recipe) и след това се потвърждава чрез db.session.commit(). Потребителят след това се пренасочва обратно към страницата за рецепти, където промяната е видима.

8.4. Сигурност чрез @login_required и управление на сесии

Всички операции в този модул, които имат влияние върху данните, като генериране и изтриване, са защитени с Flask-Login декоратора @login_required. Това гарантира, че:

- Само автентифицирани потребители имат достъп до функционалностите за добавяне, преглед и модификация на данни.
- Всяко действие, което модифицира данни в базата, се извършва в контекста на валидна потребителска сесия, като по този начин се запазва цялостта на системата и предотвратява неоторизиран достъп.
- При опит за достъп до защитен маршрут, не-влезли потребители бъдат автоматично пренасочени към страницата за вход, както е конфигурирано в настройките на Flask-Login (login_view = 'auth.login').

9. Шаблони на потребителски интерфейс (templates)

В уеб приложението Mealise, потребителският интерфейс се реализира чрез набор от HTML шаблони, използващи мощната система на Flask – **Jinja2**. Тази шаблонна система позволява вграждане на динамично съдържание, условна логика и наследяване на основна структура, което осигурява гъвкав и поддържаем фронтенд код. В допълнение към това, Mealise използва **Bootstrap 5**, който предоставя адаптивен, модерен и достъпен дизайн на страниците.

9.1. base.html – основна рамка на приложението

base.html е базовият шаблон, на който се основават всички останали страници в Mealise. Той дефинира общата структура на интерфейса, включително страничната навигация, главния контейнер за съдържание, поддръжката на стилове и JavaScript библиотеки.

9.1.a. Основни елементи:

- **Главна структура (HTML5, meta tags и Bootstrap):**
  В <head> секцията са зададени кодиране, viewport и линк към CDN на Bootstrap 5, осигуряващ стилове за целия интерфейс.
- **Странична навигация (Sidebar):**
- **Основно съдържание:**
  Използва се Jinja2 блокът {% block content %}, който позволява на наследник шаблоните да поставят своето уникално съдържание в централната част на приложението.

- **Условна логика с Jinja2:**
  В base.html е реализирана условна видимост за представяне на различни елементи в зависимост от автентикационния статус на потребителя – например приветствен текст и бутон за изход.

9.2. landing.html – начална страница (Landing page)

landing.html е първата страница, която вижда потребителят при отваряне на приложението. Тя е фокусирана върху кратко и ясно представяне на основната цел на Mealise и бърз достъп до ключовите функционалности.

9.2.а. Основни компоненти:

- **Наследяване от base.html:**
  Това позволява единния стил и навигация да бъдат автоматично приложени.
- **Приветствен блок с Bootstrap класове:**
  Горещ раздел, който центрира съдържанието вертикално и хоризонтално, с голямо заглавие „Welcome to Mealise“, подзаглавие и два големи бутона.
- **Бутони за навигация.**

9.3. discover.html – търсене и преглед на рецепти

discover.html обслужва функционалността за търсене, филтриране и разглеждане на рецепти, получени от Spoonacular API. Този шаблон е проектиран с акцент върху удобството и гъвкавостта при намирането на кулинарни идеи.

9.3.а. Основни елементи:

- Форма за разширено търсене:
  - Ключова дума (текстово поле).
  - Вид кухня (select).
  - Диетични предпочитания (select).
  - Начин на сортиране (select).
    Форма използва метод POST и изпраща данните към същия маршрут, където бекенд логиката обработва филтрите и зарежда резултатите.
- Динамично зареждане на рецепти:
  - Началният набор от рецепти се показва чрез рендериране на данните от backend.
  - Под рецепти се намира бутон „Load More Recipes“, който чрез JavaScript/AJAX зарежда още рецепти без презареждане на страницата.

- AJAX заявките използват GET метода към маршрута /discover/load_more, където се връща HTML фрагмент _recipes.html, който се прибавя към текущия списък.

- Визуално представяне на рецепти:
  - Всяка рецепта е в отделен Bootstrap card, съдържащ заглавие, време за приготвяне (по възможност), кратко резюме и бутон за външно насочване към източника (sourceUrl).
  - Изображенията (ако са налични) се показват вляво, придавайки по-добър визуален ефект.
- Използване на Jinja2 условия и цикли:
  - Тези конструкции позволяват генериране на списък с рецепти в зависимост от наличните данни.
  - Също така условни блокове предпазват от показване на празни или липсващи полета.
- JavaScript за взаимодействие:
  - Скриптът въвежда логиката на бутона за „Clear Filters“ (изчистване на филтрите и зареждане на първоначалния списък).
  - Също така управлението на зареждането на новите рецепти се контролира от JS, който добавя асинхронно новото съдържание в контейнери.

9.4. login.html и register.html – форми за автентикация

Тези два шаблона осигуряват интерфейсите за **вход** и **регистрация** на потребители.

9.4.a. Общи характеристики:

- И двата са наследници на base.html и използват Bootstrap класове за подредба, стил и реактивност.
- Имат центриран формуляр с подходящи полета, валидирани на ниво браузър (required).

9.4.b. login.html

- Съдържа полета за потребителско име и парола.
- Кнопка „Login“ с Bootstrap стил.
- Под формата се предлага линк към страницата за регистрация.

- Използва flash съобщения за показване на грешки или успех при опити за вход (осигурено от бекенд логиката).

9.4.c. register.html

- Съдържа полета за име, email, парола и потвърждение на паролата.
- При възникване на грешки (напр. несъвпадение на пароли или вече съществуващ потребител) се показват flash съобщения.
- Под формата има линк към страницата за вход.
- Използва Bootstrap card с ясна структура и линии, които отделят съдържанието.

9.5. recipe_page.html – страница за управление на рецепти

Това е ключовия шаблон за потребители, които вече са логнати и искат да генерират или да разглеждат своите рецепти.

9.5.a. Основни секции:

- Форма за генериране на рецепти:

Текстово поле (textarea) за въвеждане на съставки. Бутон за изпращане на заявката към бекенда. При натискане на бутона в страницата се появява loader с анимация и полупрозрачен overlay, предоставящи визуална обратна връзка.

- Списък с генерирани рецепти:

Визуализира всички рецепти, създадени от текущия потребител, подредени в cards. Всяка рецепта включва: заглавие, дата на създаване, списък със съставки, инструкции. Парсирането на съставките и инструкциите е направено чрез разделяне на текстовите полета по дефинирани символи (напр. ingredients.split('--')). За всяка рецепта има бутон за изтриване („Delete“), който използва POST заявка и потвърждение от потребителя (javascript confirm диалог).

- Пагинация:

Под списъка с рецепти е реализирана пагинация, която позволява навигация между различните страници с рецепти. Използват се бутони за предишна и следваща страница, а също и конкретни номера на страници. Активната страница е визуално маркирана с клас bootstrap active.

10. Заключение и възможности за развитие

Уеб приложението **Mealise** представлява иновативно решение в сферата на кулинарните платформи, като съчетава няколко ключови технологии и подходи за изграждане на качествен продукт, ориентиран към потребителското изживяване и лесна

поддръжка от страна на разработчиците. В тази последна част на документацията ще обобщим основните предимства на приложението, неговата архитектура, както и перспективите за бъдещо развитие и подобрения.

10.1. Предимства на Mealise като модулно Flask приложение с API и AI интеграция

Mealise се отличава с архитектура, изградена върху модулния подход на **Flask** framework, който позволява ясно структуриране на проекта в отделни модули (blueprints) с конкретни отговорности. Това разделение на логиката, шаблоните и разширенията дава следните ключови предимства:

Лесна мащабируемост и поддръжка:

Чрез използване на Flask Blueprints и модулност, добавянето или модифицирането на функционалности става без сериозен риск от нарушаване на цялостната стабилност на приложението. Това улеснява работата на екипа и позволява бързо развитие.

Интеграция с външни услуги и AI модели:

Mealise внася многократно повишена стойност чрез свързването със **Spoonacular API** – осигуряващ богата база от проверени рецепти, и **OpenAI API** (чрез OpenRouter) – за интелигентно и персонализирано генериране на нови рецепти на базата на налични продукти. Тази двойна интеграция разширява възможностите на приложението далеч отвъд локалното съхранение и базова логика.

Сигурност и персонализация:

С използването на **Flask-Login** и **Flask-Bcrypt** е осигурена надеждна система за автентикация и криптиране на потребителски пароли, което гарантира безопасност на потребителските данни. Потребителите имат възможност да създават свои персонални профили и да управляват собствената си колекция с рецепти, което повишава ангажираността.

Гъвкав и интуитивен интерфейс:

Благодарение на ясното разделяне на HTML шаблоните, построени върху **Jinja2** и **Bootstrap 5**, Mealise предлага модерен, адаптивен и приятен за използване интерфейс. Методите за динамично зареждане на съдържание (например „Зареди още рецепти“) подобряват UX и намаляват необходимостта от пълно презареждане на страниците.

Автоматична настройка и конфигурация:

Конфигурационният модел, базиран на .env файлове и централизирана фабрика за създаване на приложението (create_app() в app.py), улеснява разгръщането и адаптирането на приложението в различни среди – разработка, тест или продукция.

10.2. Как архитектурата позволява лесно разширяване и подобрение

Модулната структура и разделението по функционални зони в Mealise формират солидна основа за добавяне на нови възможности с минимални усилия и риск от непредвидени проблеми. Сред ключовите аспекти, които я правят особено подходяща за бъдещо развитие, са:

Ясни интерфейси между компонентите:

Всеки модул (например auth.py, recipes.py, discover.py, home.py) комуникира чрез добре определени входни и изходни данни, базирани на Flask routing, ORM модели и шаблони. Това намалява зависимостите и улеснява паралелна работа и промени.

Лесно добавяне на нови API интеграции:

Понастоящем Mealise използва Spoonacular за откриване на рецепти и OpenAI за генерация, но архитектурата позволява въвеждане на нови външни източници (като други кулинарни бази данни, социални мрежи или локални услуги) без големи промени.

Интеграция на нови потребителски функционалности:

Създаване на нови филтри, добавяне на социални компоненти (оценяване, коментари) или системи за препоръки може да бъде реализирано като отделни blueprint модули.

Гъвкава работа с базата от данни:

Като използва SQLAlchemy ORM, приложението може лесно да разшири моделите с допълнителни атрибути (например категории рецепти, бележки, нива на сложност) или да мигрира към по-мощна база с минимални промени по логиката.

Персонализация и профилиране:

Лесното управление на потребителите отговаря на изграждането на профили, които биха позволили персонализирани препоръки, история на генерирани рецепти, запазени любимци и други.

10.3. Заключителни бележки

Mealise представлява добре замислена и реализирана платформа, която успешно съчетава модерни технологии за уеб разработка към момента на създаването ѝ. Архитектурата ѝ осигурява стабилна основа за ефективна разработка, лесна поддръжка и гъвкаво разрастване, като същевременно предлага сложни функционалности на крайния потребител — от сигурно управление на профили, през интегриране на външни данни, до интелигентно генериране на рецепти.

С пускането и преминаването на Mealise от прототип към продукт, в бъдеще могат да се реализират множество подобрения и нови функции, които ще превърнат приложението в пълноценен кулинарен асистент с висока добавена стойност за

потребителите. Този проект илюстрира как с добър архитектурен дизайн, използване на модерен технологичен стек и ефективна интеграция на външни услуги, може да се създаде значимо и иновативно уеб приложение.