Meal Planner

Course Title: Agile Webanwendungen mit Python

Team Members:

- Frank Bänecke (Master Informatik, 2251624)

- Erik Jaramillo Fernández (Bachelor Informatik, 2245968)

- Veera Goutham Venkatesh (Bachelor Technische Informatik, 2127743)

Instructor: Erich Seifert

Date: 18.07.2001

Table of Contents

1. Motivation and Requirements

1.1 Project Idea and Goals

The core idea behind **Mealano** was to create a smart and user-friendly web application for people who want to eat consciously — not only in terms of calories but also with attention to macronutrients like protein, carbs, and fat.

The motivation came from the need for flexibility and variety in diet planning. Many existing tools offer rigid plans or focus on a single diet type. **Mealano** allows users to:

- Select their preferred diet (e.g., vegan, keto, paleo, etc.),

- Set a daily calorie goal,

- Choose how many meals they want,

- Decide how healthy the meals should be,

- And save their favorite plans for future use.

As a bonus, **Mealano** offers a fun feature to generate **cheat meals** like desserts and drinks within a custom calorie limit — perfect for indulging a little without going completely off track.

To support day-to-day cooking and inspire variety, **Mealano** also features a powerful **recipe search** interface. Users can input ingredients using smart autocomplete search parameters, making it easy to find relevant recipes that match, for example, what they already have at home. Each recipe comes with a **detailed view** showing ingredients, preparation steps, and a link to the

original source. Users can **save recipes** they like and access them anytime via their personal

Saved Recipes page — simplifying meal planning and grocery shopping alike.

This system helps users cook meals at home that align with their nutritional goals — giving them

both inspiration and control. This is especially useful for fitness enthusiasts, bodybuilders, and

gym-goers who want to stick to strict calorie intake and macronutrient targets.

The ultimate goal is to empower users to generate personalized meal plans and explore fitting

recipes based on their exact preferences — all without needing to be a nutrition expert.

1.2 Requirements

a) Functional Requirements

- **Meal Plan Generation**

  The system must allow users to generate meal plans based on their preferences, including

  dietary needs, calorie targets, or specific ingredients.

- **Save Meal Plans**

  Users must be able to save generated meal plans for later access and reuse.

- **Include Drinks and Desserts**

  The application should support the generation of meal plans that include not only main

  dishes but also drinks and dessert options.

- **Recipe Search Functionality**

  Users must be able to search for recipes using keywords, ingredients, or dietary filters.

- **View Recipe Details**

  The system should provide detailed information for each recipe, including ingredients, instructions, preparation time, and nutritional information.

- **Save Recipes**

  Users must be able to save their favorite recipes to a personal collection for quick future access.

- **Unsave Recipes**

  Users must be able to remove saved recipes from their personal collection.

- **View Saved Recipes**

  Users should be able to view all recipes they have saved in a dedicated section.

- **User Credential Management**

  Users must be able to change their credentials, including username, password, and email address.

- **User Registration**

  The application must allow new users to register by providing necessary credentials such as username, email, and password.

- **User Login**

  The system must provide a login functionality for existing users to access their personalized content and saved data.

b) Non-functional Requirements

- **Minimize API Calls**

  The system should be optimized to reduce the number of API calls to external services, improving performance and minimizing costs.

- **Centralized API Handling**

  All API access must be handled through a single class (e.g., SpoonacularAPI) to ensure consistent and maintainable code.

- **Separation of Concerns (HTML/CSS/JS)**

  The codebase must follow a clear separation of concerns by keeping HTML, CSS, and JavaScript in separate files to ensure maintainability.

- **Logical Folder Structure**

  The project must use a clean and logical folder structure to enhance readability, maintainability, and scalability.

- **Unique Recipes in Meal Plans**

  Each generated meal plan should contain different recipes to provide variety and avoid repetition.

- **Follow the 3-Click Rule**

  Navigation throughout the application should follow the 3-click rule, ensuring that users can reach any feature within three clicks from the homepage.

- **Error Handling**

  The system must implement proper error-handling mechanisms to gracefully manage API failures, form errors, or missing data.

- **Unique Username and Email Validation**

  During registration or when updating credentials, the system must check whether the provided username or email is already in use.

- **Custom 404 Error Page**

  The application must display a user-friendly custom 404 error page when a requested resource is not found.

- **Fast Meal Plan Generation**

  The system must generate a complete meal plan in less than 10 seconds to ensure a smooth user experience.

- **Memory Usage Limit**

  The application should not consume more than 500 MB of RAM during normal operation to ensure efficiency and compatibility with hosting environments.

- **Cross-Browser Compatibility**

  The system must be compatible with all major web browsers (e.g., Chrome, Firefox, Safari, Edge) to ensure accessibility for a wide user base.

- **Comprehensive Documentation**

  The entire system must be supported by comprehensive documentation, including setup instructions, API references, and developer guides.

1.3 Research and Existing Solutions

Eat This Much is an automatic meal planning application that allows users to generate personalized meal plans based on their dietary goals, calorie targets, and preferences. Mealano's dashboard is inspired by Eat This Much's one.

[The Automatic Meal Planner - Eat This Much](#)

There exists a suite of web and mobile **applications built on top of the powerful Food API from Spoonacular**. These apps provide functionality such as recipe search, ingredient analysis, and meal planning, among others. Mealano includes the mentioned functionalities.

[Apps built with the spoonacular API](#)

The **Spoonacular Meal Planner** is a feature-rich web tool that enables users to create and customize meal plans using the Spoonacular API.

[Free Weekly Meal Planner with Grocery List, Food Tracker, Recipes, and Products](#)

Restegourmet is a web application that helps users find recipes based on the ingredients they have at home. Its core feature is an intuitive recipe search powered by an autocomplete ingredient search: as users type ingredients into the search bar, suggestions appear in real time. Once ingredients are selected, the app retrieves matching recipes.

This makes it easy for users to reduce food waste by discovering meals they can cook with leftover ingredients.

[Restegourmet App – Rezeptsuche nach Zutaten & Wochenplaner](#)

2. Planning and Design

2.1 Task Distribution and Collaboration

- **Meetings** talking about current status, next plans and task distribution with 2-week sprint
  - Meeting 1: A brainstorming session was held to discuss the initial vision of the project and to define the first set of functional and non-functional requirements.
  - Meeting 2: A face-to-face meeting took place to introduce all team members to one another, helping to establish personal connections and improve collaboration going forward.
  - Meeting 3: This meeting focused on discussing the main issues encountered during the development process up to that point. Common knowledge was shared among the team to align understanding, and concrete solutions and goals for the next phase were established.
  - Meeting 4: The team met to make key decisions regarding documentation. A collaborative programming session was also conducted to gain insight into each member's current workload and challenges. Additionally, a retrospective analysis of the application was performed to evaluate progress and future improvements.

- **Communication Channels:**

  A WhatsApp group and a Discord server were used as fast, direct communication channels for daily coordination and quick decision-making.

- **Task Management:**

  GitHub Issues were utilized to document, assign, and track tasks throughout the development process, helping the team stay organized and accountable.

- **Product Feedback:**

  Since the project does not involve any external companies or official stakeholders, product reviews and usability feedback were obtained from relatives and friends who expressed interest in the application.

2.2 Chosen Technologies

- **Flask-Framework**

  - Version 3.1.1

  - Forms: Flask-WTF 1.2.2

  - Database-Access: SQL Alchemy 2.0.41, Flask SQL Alchemy 3.1.1

  - User Management: Flask-Login 0.6.3

- **Database:** SQLite 3.26.0

- **Languages:**

  - Python 3.1.1

  - HTML5

  - Jinja2

       ○ CSS

       ○ JS 1.5

    ● **IDE:** IntelliJ, VS Code

2.3 Architecture and Design Decisions

Integration with Spoonacular API

To access recipe and ingredient data, the system uses the Spoonacular API. Since the API has request limits and access constraints, a local caching strategy is implemented. Fetched data from the API (e.g., recipes, ingredients) is stored in the local database to minimize redundant API calls and improve performance.

Internal System

The Mealano system is built with three main components:
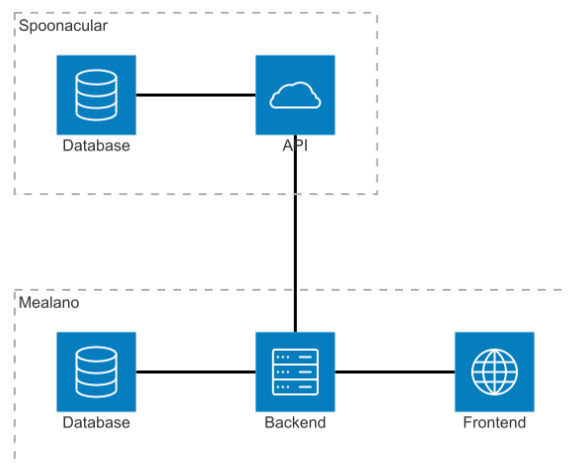
- **Database**: Stores both cached API data and user-specific content, such as saved recipes, meal plans, and authentication information.

- **Backend**: Handles business logic, manages API communication, and renders templates of the frontend. It acts as a bridge between the frontend and the data layer.

- **Frontend**: Provides the user interface for interacting with recipes, meal plans, and other features. It communicates with the backend to get data and perform actions.

This separation of concerns improves maintainability, scalability, and allows independent development of frontend and backend components.

2.4 Database and Data Model

The application's database schema is designed to efficiently manage users, recipes, ingredients and meal plans. Below is an overview of the main entities, their relationships, and the key attributes used throughout the system.

Icon-Legend: *https://www.jetbrains.com/help/idea/2025.1/creating-diagrams.html#icons*
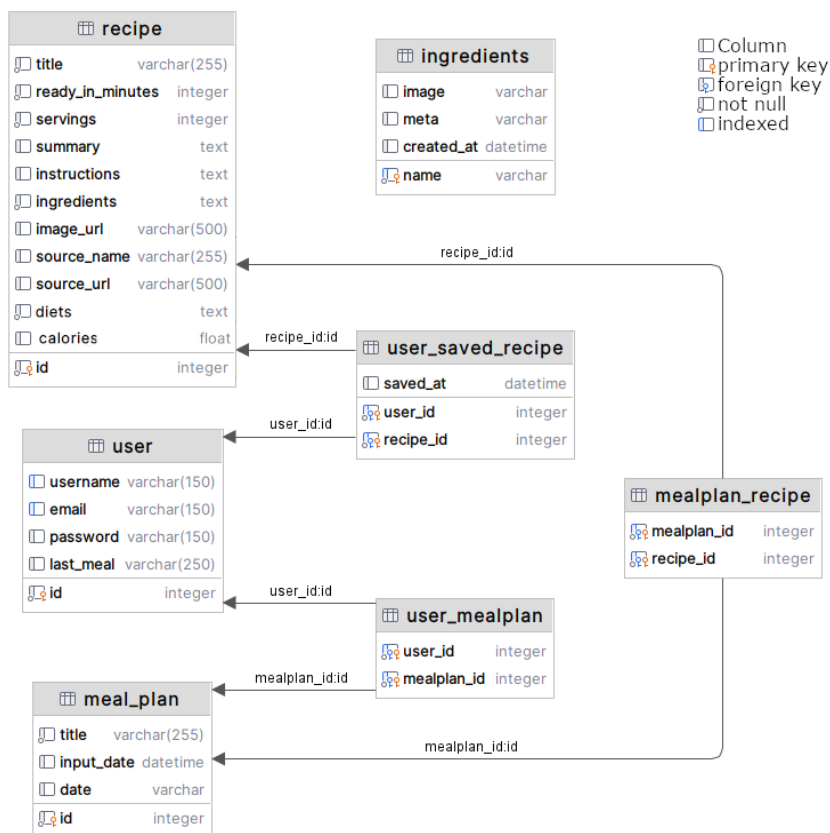
- **Users (**`user`**)**

  Represent individual accounts within the system. Users can save recipes and create

  personalized meal plans.

- **Recipes (**`recipe`**)**

  Contain all relevant details about a dish, including preparation time, ingredients,

  instructions, and optional metadata like images, source information and diet-infos.

- **Ingredients (**`ingredients`**)**

  Provide structured data for common ingredients. Ingredients are cached to more

  efficiently provide the autocompletion of ingredients on the search page, while reducing

  API-calls Additionally they provide the basis for future enhancements like ingredient

  analysis or grocery list generation.

- **Saved Recipes (**`user_saved_recipe`**)**

  Track which users have bookmarked or saved specific recipes for future reference.

- **Meal Plans (**`meal_plan`**)**

  Allow users to group multiple recipes under a specific plan or schedule. Meal plans can

  be dated and titled for organizational purposes.

- **Meal Plan Recipes (**`mealplan_recipe`**)**

  Define the association between recipes and meal plans, enabling many-to-many

  relationships.

- **User Meal Plans (**`user_mealplan`**)**

  Link users to their respective meal plans, allowing shared or individual ownership.

3. Development

3.1 Implementation Overview
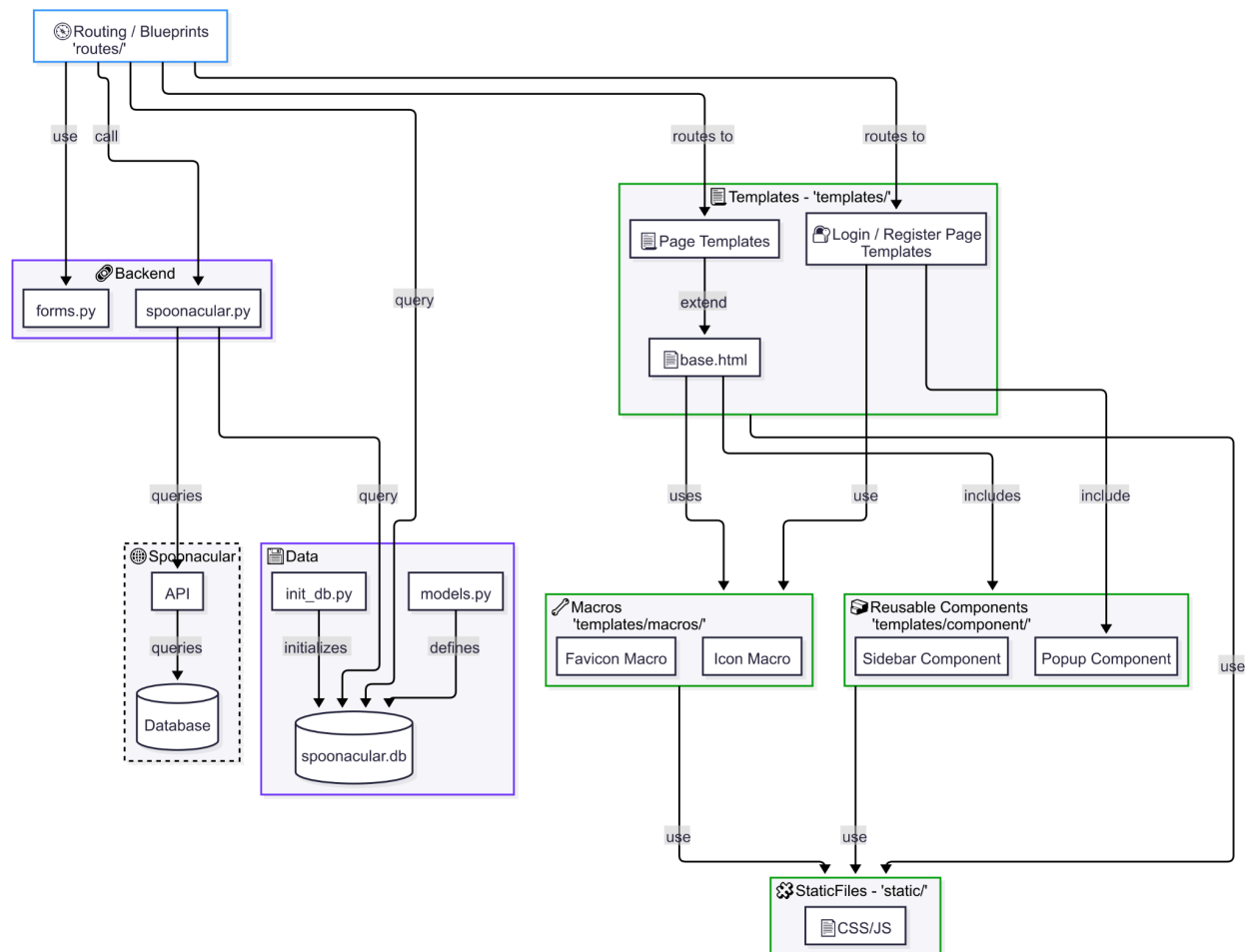
(Diagram with routing-/template-files listed separately, can be found in the Appendix)

The application is structured using modular Flask blueprints and a layered architecture that cleanly separates concerns such as routing, templating, backend logic, and database access.

Routing and Blueprints (`routes/`)

 All application routes are organized across multiple blueprints, each handling a distinct part of

the application logic:

- `auth.py`, `account.py`: Handle authentication and user account functionality.

- `dashboard.py`, `meal_plan.py`: Serve the home dashboard and meal planning tools.

- `recipes.py`, `browse.py`, `dessert_drinks_page.py`, etc.: Provide functionality for
  browsing and searching recipes.

Each blueprint routes requests to a corresponding template under `templates/`.

Templating (`templates/`)

Templates are based on a shared `base.html` layout, which is extended by all page templates,

apart from Login/Register to ensure consistency in design and structure. The key pages rendered

via these templates include:

- **Authentication Pages**: Login, Register (not based on `base.html`)
- **User Pages**: Account, Your Meal Plans, Saved Recipes
- **Content Pages**: Dashboard/Meal Planner, Desserts & Drinks, Recipe Search, Recipe
  Details

Elements such as the **Sidebar Component** and **Popup Component** are included through

separate files under `templates/component/`, while macros for inlining svg-icons and adding

favicon-meta-data are defined under `templates/macros/`.

Static Assets (`static/`)

CSS, JavaScript and icon files reside in the `static/` directory and are used across all templates to style and power UI behavior.

Backend Logic

- `forms.py` provides user input forms (login, register, account).

- `spoonacular.py` encapsulates logic for querying the external **Spoonacular API**, which serves recipe data used throughout the application.

Database and Models (`data/`)
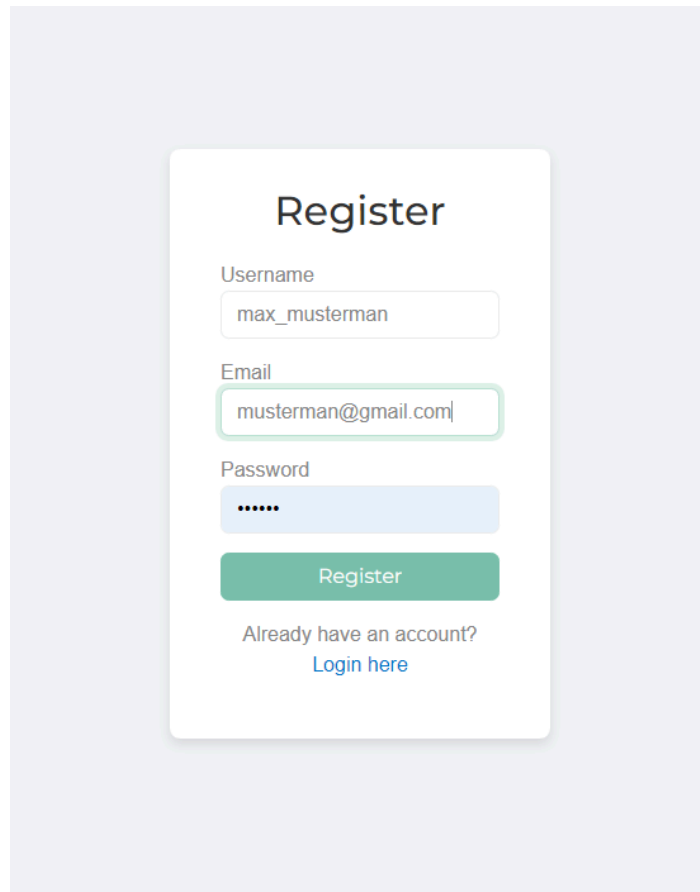
- `init_db.py` initializes the local SQLite database `spoonacular.db`.

- `models.py` defines ORM models used to store and manage application data, such as saved recipes, meal plans and user data.

This modular structure facilitates maintainability, testing, and future expansion, with clearly defined responsibilities for each component in the system.

3.2 Features and Screenshots

a) Login / Register
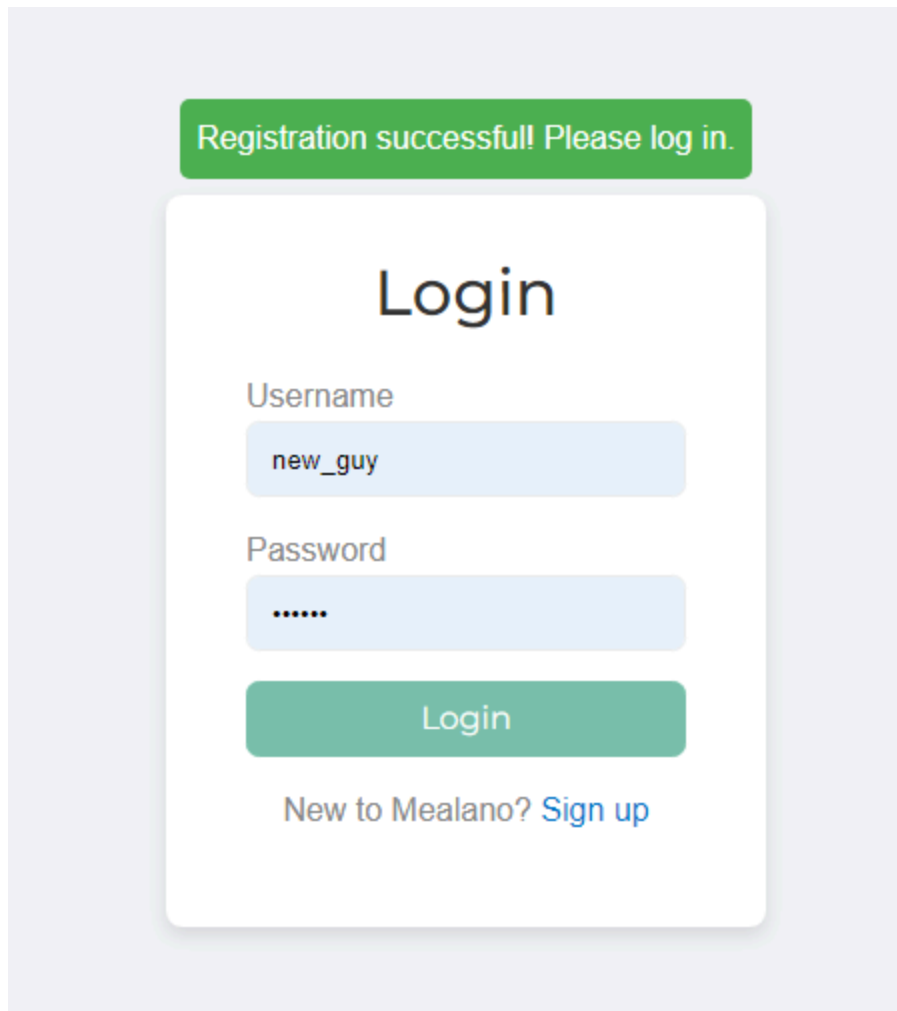
Registration-Page:.

New users can create an account by providing a unique username, an email address, and a
password.

Login-Page:

Registered users can log in using their credentials. After successful registration, a corresponding

confirmation message appears..

Implemented Features:

- Secure User Registration:

  - New users can register via a simple form.

  - The system checks whether the username or email address is already taken.

- ○ After successful registration, users are redirected to the login page with a success message

- User Login:

  - ○ Users can log in with their username and password.

  - ○ The credentials are verified against the database.

  - ○ On success, the user is authenticated via Flask-Login and redirected to the dashboard.

- Feedback via Flash Messages:

  - ○ Responses such as "Registration successful" or "Invalid login credentials" are displayed as colored notifications using a Popup-Component.

- Session Management:

  - ○ Sessions are managed using Flask-Login. This ensures that protected pages are accessible only to logged-in users.

- Logout-Function:

  - ○ Logged-in users can securely log out.

  - ○ The session is terminated, and the user is redirected to the login page.
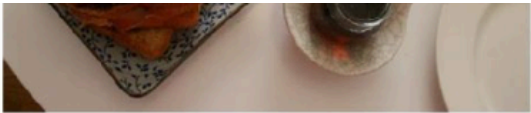
b) Dashboard – Meal plan in seconds:

In the dashboard, users can select their preferred diet type, such as "Vegan," "Keto," or "Paleo." Additionally, they can specify their daily calorie intake and the number of desired meals. Based on this input, the system automatically calculates appropriate macronutrient distributions (carbohydrates, fat, protein) and allows users to choose between different health preferences

(e.g., "Very healthy" or "I don't care"). With a single click on "Generate," a personalized daily meal plan is created.

Balthazar Brioche French Toast
1 serving

Dinner

~ 662 Calories

Baked Fried Chicken With Cauliflower Mash
1 serving

Lunch

~ 537 Calories

Chicken Verde Enchilada Casserole
1 serving

💾 Save Meal Plan

The result is presented to the user as a clear list – divided into breakfast, lunch, dinner, etc., each with calorie information, an image, a description, and a button to save it. Additionally, the recipe title acts as a clickable link that leads directly to the original page with detailed information, ingredients, and preparation steps.

Implemented Features:

- User input (calories, diet type, number of meals, etc.) is collected in the frontend and sent as JSON to the server via JavaScript.

- In the backend, the Spoonacular API is used to dynamically fetch suitable recipes.

- A smart selection logic evaluates the recipes based on health scores, nutrient balance, and user preferences.

- Results are filtered, randomly shuffled, and displayed as diversely as possible.

- Already used recipes are stored in the session to avoid duplicates.

- The backend sends the final recipe data back to the frontend, which presents it in an appealing layout.

- The recipe title links directly to the detailed original page with further information about the dish.

c) Dessert & Drinks – The Cheat-Meal-Page:

This section of the application is aimed at users who occasionally want to treat themselves — for example, with a dessert or a refreshing drink while still keeping their calorie intake in check. The user interface allows users to define a maximum calorie limit separately for desserts and drinks, and also select the number of suggestions they'd like to receive.

After clicking "Generate," suitable recipes are retrieved via the Spoonacular API. A random offset is used to ensure variety in the results. The results are sorted by popularity and displayed directly in the interface — including an image, calorie count, and title.

Implemented Features:

- **Filtering by Recipe Type:** Specifically searches for recipes categorized as dessert or drink.

- **Randomized Results:** Achieved through a randomized offset and sorting by popularity.

- **Calorie Limit:** Customizable per type (e.g., max. 400 kcal for desserts).

- **Result Display:** Recipe title acts as a clickable link to the original website with full details.

d) Your Meal Plans

Your Meal Plans

Birthday   View Recipes   ✎ Rename   🗑 Remove

Monday   View Recipes   ✎ Rename   🗑 Remove

looks good   View Recipes   ✎ Rename   🗑 Remove

Smoked Salmon Eggs Benedict With Lemon Dill Hollandaise
Ready in 45 minutes · Serves 2

Smoked Salmon Eggs Benedict With Lemon Dill Hollandaise could be just the **pescatarian** recipe you've been looking for. One serving contains **540 calories**, **41g of protein**, and **33g of fat**. This recipe serves 2 and costs $6.33 per serving. 74 people were impressed by this recipe. Head to the store and pick up eggs, lemon, butter, and a few other things to make it today. It is brought to you by Foodista. A couple people really liked this breakfast. From preparation to the plate, this recipe takes around **45 minutes**. Overall, this recipe earns a **tremendous spoonacular score of 92%**. If you like this recipe, take a look at these similar recipes: Smoked Salmon Eggs Benedict With Lemon Dill Hollandaise, Smoked Salmon Eggs Benedict With Lemon Dill Hollandaise, and Smoked Salmon Dill Eggs Benedict.

Sweet Mustard BBQ Pork Chops
Ready in 45 minutes · Serves 4

Sweet Mustard BBQ Pork Chops requires approximately **45 minutes** from start to finish. For **$2.15 per serving**, you get a main course that serves 4. One serving contains **445 calories**, **38g of protein**, and **16g of fat**. This recipe from Foodista requires soy sauce, dijon mustard, garlic, and coarsely ground pepper. 69 people were glad they tried this recipe. It is a good option if you're following a **gluten free and dairy free** diet. All things considered, we decided this recipe **deserves a spoonacular score of 73%**. This score is solid. Sweet Mustard BBQ Pork Chops, Honey Mustard BBQ Pork Chops, and Sautéed Pork Chops with Sweet Potato, Apples and Mustard Sauce are very similar to this recipe.

The *Your Meal Plans* page displays the meal plans the user has previously saved on the Dashboard / Meal Planner page. This allows users to save generated meal plans, containing the corresponding recipes, to revisit them across different days. Users can also remove plans, if needed.

e) Search Page

The *Search* page provides a powerful and user-friendly interface for finding recipes. It includes

many different search param fields, while providing autocompletion functionality for ingredients

and other relevant fields such as diets, cuisine and intolerances. This simplifies the search

process and helps users find relevant recipes with minimal typing, particularly useful when

cooking with leftovers or limited pantry items.

f) Recipe Details Page

The *Recipe Details* page presents comprehensive information about a selected recipe, including ingredients, instructions, and dietary information as available. Users can easily **save or unsave** recipes, or follow the link to the external source (e.g. original recipe website) for users who wish to see more details or preparation tips.

g) Saved Recipes
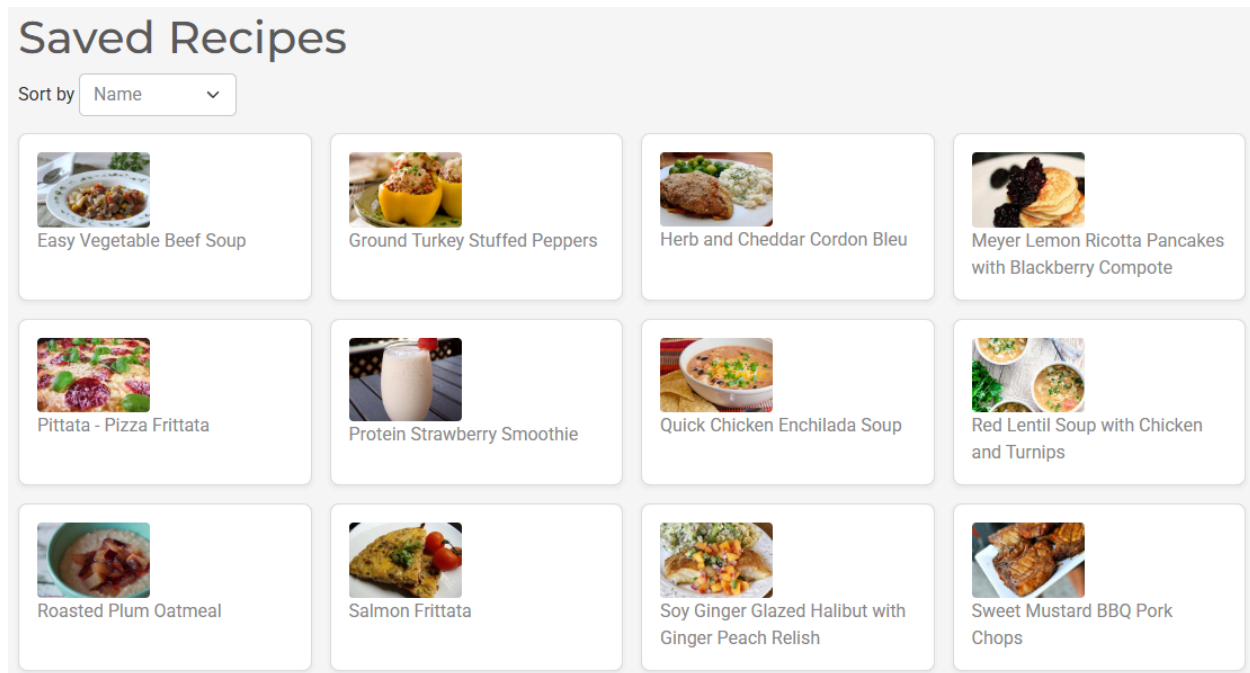
The *Saved Recipes* page acts as a personalized collection of bookmarked recipes. It allows users to quickly revisit dishes they've liked or want to try later. The list of recipes can be sorted by name or recency of when it has been saved.

h) Sidebar Component

The *Sidebar Component* serves as a consistent and accessible navigation tool across the application. It provides direct links to the different pages, including a search bar for direct queries. At the bottom, users can access their account settings or log out, making the sidebar an essential part of the app's usability and user experience.
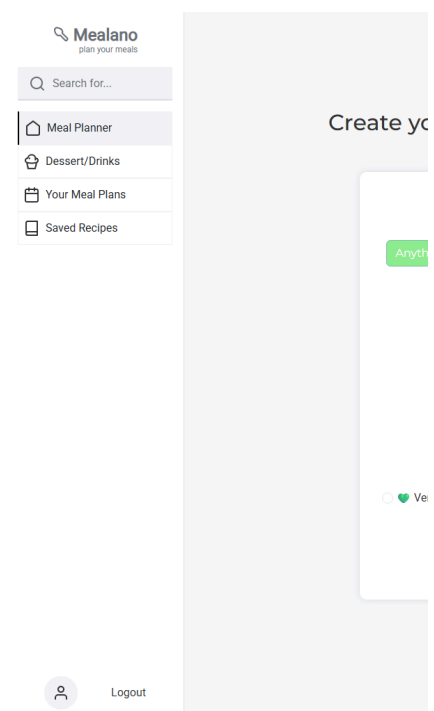
3.3 Technical Challenges and Solutions

- Free API access is rate-limited.

    ○ Recipes are cached to reduce redundant API calls.

    ○ The autocompletion of ingredients on the search page uses data returned by the

       `/food/ingredients/autocomplete` endpoint, as well as from the local database.

       The local database contains ingredients cached from the aforementioned endpoint,

       as well as from previously cached recipes.

- Some user preferences were not being properly considered during meal plan generation.

    ○ The generation process was adjusted to better reflect the user's preferences by

       properly transforming the input values.

    ○ The application now ensures that different recipes are generated for each meal.

- Meal plans were not being saved correctly.

    ○ Recipes included in the meal plans were, on some occasions, missing or

       incomplete, even though the data was available when saving the recipe

       individually.

    ○ The issue stemmed from the application's approach to fetching recipe data. It first

       checked the database for the recipe ID, and if found, it retrieved the data from

       there. However, the model did not contain enough fields to store all the required

       recipe information.

- ○ To resolve this, the `save_meal_plan` function in `dashboard.py` was refactored. It

  now saves each recipe in the meal plan with all missing data in the database, if it

  is not already stored.

- ○ The system now correctly retrieves recipe data for frontend display.

- Previously, some recipes lacked critical data needed by the application, such as the

  image, health score, instructions, ingredients, source URL, and diet type.

  - ○ This issue originated from the same process used in meal plan generation.

  - ○ The solution was to update the database model to include the missing fields and

    adjust the output of the API call functions to ensure that all necessary data is

    returned.

4. Source Code and Commissioning

The source code can be found on GitHub: https://github.com/KireStrings/Meal-Planner

Follow these steps to install, configure, and run the application locally:

- Install dependencies

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

- Initialize the database

```
python init_db.py
```

- Configure API key

  a. Get your Spoonacular API key.

  b. Copy the .env.dist file and rename it to .env.

  c. Open .env and paste your API key like so:

  ```
  SPOONACULAR_API_KEY=your_api_key_here
  ```

- Run the application

```
flask run
```

5. Conclusion

5.1 Summary of Achievements

The implemented application meets the core functional and technical requirements outlined at the beginning of the project. The key achievements set by the course were fulfilled:

- **User Login System**

  A fully functional authentication system was developed using Flask-Login. It supports user registration, login, and session management. Templates for login and registration pages are provided, and user credentials are securely handled via form validation and integration with the database.

- **Database Integration**

  A local SQLite database (`spoonacular.db`) was successfully integrated. It stores user accounts and application data such as saved recipes and meal plans. The database schema is defined using SQLAlchemy models, and an initialization script (`init_db.py`) ensures consistent setup across environments.

- **Python-Based Web Framework**

  The entire system is built on the Flask framework, leveraging its blueprint architecture for modular routing, Jinja2 templating for dynamic page rendering, and integration with external APIs and a local database. This provides a clean separation of concerns and a scalable structure for further development.

Overall, the implementation demonstrates a working full-stack web application with modern

features such as templated layouts, reusable components, and persistent user data.

5.2 Task distribution

Joint tasks		
● Brainstorming of project ideas ● Conceptual design of the application ● Documentation	● Bugfixing ● Planning of tasks and their distribution ● …	
Frank Bänecke (2251624)	**Erik Jaramillo Fernández (2245968)**	**Veera Goutham Venkatesh (2127743)**

● GitHub-Repo
  ○ Conflict resolving
● requirements.txt for project dependencies
● dotenv-handling for spoonacular-API-Key
● Design-Template on Figma
● Mealano-Logo
● Modularization of Templates
  ○ base.html
  ○ SideBar-/Popup-Component
  ○ Separation of HTML/CSS/JS
● `icon()`-Macro for using svg-icons
● flash-message-Popups (showing Login-/register-/account-infos/-errors)
● SpoonacularAPI-class, to access spoonacular API
● refactor/clean up CSS, templates, routes (login required, blueprints), API-Access
● clean up design of buttons/forms
  ○ added classes
  ○ removed redundant CSS covered by Bootstrap
  ○ bootswatch minty-theme (Bootstrap-Theme)
● RecipeSearch
  ○ Autocomplete ingredients
	● GitHub-Repo
	  ○ Security measures (.gitignore for secrets, set default value for API keys in the code)
	  ○ Cleaning the repository (.gitignore for __pycache__, databases, databases data)
	  ○ Conflict resolving and branch merging
	● updated requirements.txt
	● Modularization of Templates
	  ○ Separation of HTML/CSS/JS
	  ○ Frontend - Backend integration (JS modules call backend for data retrieving and storing)
	● UserSavedRecipes
	  ○ Save recipes (from MealPlanner and Recipe details)
	  ○ Show recipes saved by the current user
	● MealPlan
	  ○ Save MealPlans from MealPlanner (providing name)
	  ○ Show meal plans saved by the user
	  ○ Meal Plan Generating Form
	  ○ Store data about the recipes the Meal Plan which is saved contains
		Project Setup
		
		● Created initial Flask project structure with Blueprints and modular file organization.
		
		● Configured routing and `__init__.py` to register Blueprints cleanly.
		
		User Authentication System
		
		● Implemented user registration and login functionality with hashed passwords.
		
		● Integrated Flask-Login for session management and user authentication.
		
		● Connected user data with SQLAlchemy models.
		
		Dashboard & Meal Generation
		
		● Built frontend and backend logic for the **dashboard page**.
		
		● Created dynamic meal generation flow using

(in-/exclude) by calling database and API-Endpoint ○ Autocomplete cuisines, diets, intolerances against static lists ● RecipeDetails ○ unsave Recipes ○ reload after save/unsave ○ fallback to spoonacular link when external link is missing ● Cache Recipes ● SavedRecipes ○ shows all cached Recipes (old) ○ sort by name vs last saved ● MealPlan ○ link to RecipeDetails ● Account-Settings ○ check if anything changed ○ change username (check if unique) ○ change email (check if unique) ○ change password (only updated if field not empty) ● 404-page ● favicon ● Diagrams in Documentation	● Dashboard ○ Save buttons for generated recipes and meal plans ● RecipeDetails, RecipeSearch ○ Alert users if they attempt to save recipes they have already saved. ● Add tables for each data object the application has ● Add relationship tables to access filtered data by user, recipe, and meal plan ● Add real-time control for timestamp fields	Spoonacular API. ● Implemented logic to handle dietary filters, calorie limits, and health preferences. **Desserts & Drinks Pages** ● Developed backend routes and frontend templates for **Dessert** and **Drinks** categories. ● Integrated API calls to fetch and display results for these categories. **Session-based Logic** ● Used Flask session to store and track previously used recipe hashes per user session. ● Ensured new recipe suggestions do not repeat in consecutive generations.

5.3 Future Work

Login & Registration

- ***Password Hashing:*** *Currently, passwords are stored in plain text. Adding hashing (e.g., using bcrypt) would significantly improve security.*

- ***Email Verification:*** *Introduce email confirmation to reduce fake accounts and enhance trust.*

- ***Social Login Integration:*** *Allow users to log in using Google, Apple, or Facebook for convenience.*

- ***Forgot Password Functionality:*** *Add a password reset system via email link for better usability.*

Dashboard (Meal Plan Generation)

- ***Auto-Generated Shopping List:*** *Generate a shopping list from the selected meal plan.*

- ***Ratings & Comments:*** *Let users leave feedback or reviews on recipes.*

- ***Multilingual Support:*** *Support more languages (e.g., English and German) for broader usability.*

Dessert & Drinks Page

- ***Type Filters (e.g., Vegan, Gluten-Free):*** *Let users filter desserts and drinks based on dietary restrictions.*

- ***Random Surprise Mode:*** *Users can opt for surprise cheat meals without entering any preferences.*

- ***Personalized Recommendations:*** *Use previous activity or favorite ingredients to recommend desserts and drinks.*

- ***Ratings & Comments:*** *Let users leave feedback or reviews on recipes.*

6. Sources

- Script – https://smt.sytes.net/fha/2025ss/pyweb/skript/index.html

- Recordings – https://moodle.hs-augsburg.de/course/section.php?id=118464

- Flask Migrate – https://flask-migrate.readthedocs.io/en/latest/

- Spoonacular API Docs – https://spoonacular.com/food-api/docs

- SQLite – https://www.sqlite.org/docs.html

- Bootstrap – https://getbootstrap.com/docs/5.3/getting-started/introduction/

Appendix

Appendix A: Code-Structure: