BULME

HÖHERE TECHNISCHE BUNDESLEHRANSTALT GRAZ-GÖSTING

MamaFood

Marketplace for Homeccoked Foods

Supervisor: Simon Gunakar

Author: Fatemeh Abolhassanlou

INFORMATIK TAGESKOLLEG

Date: June, 2025

Dedicated

To those who empower others and bring equal opportunities to all.
To those who nurture talent, not suppress it.
To those who stand against discrimination.

Abstract

MamaFood is an online platform that allows people to share and sell their homemade food. The platform is built using MySQL for the database, SQLAlchemy for backend data interaction, and Vue.js for the frontend - technologies that provide a solid, secure, and user-friendly foundation. From the very beginning, the platform was designed with usability and data protection in mind. MamaFood is more than just a food ordering site, it encourages entrepreneurship - especially among women - by offering a flexible way to earn money through their culinary skills. At the same time, it gives customers the opportunity to discover homemade food and enjoy diverse cultural flavors, and also prevents food waste by considering the pre-order process and order quantity.

Contents

1 Introduction

Mamafood is an innovative online platform that aims to connect customers directly with local home cooks. It provides a convenient, structured, and user-friendly platform for ordering, preparing, and delivering home-cooked meals. Mamafood's main focus is to provide opportunities for people who may not be able to work full-time, especially housewives, but who have a talent and passion for cooking.
Mamafood allows these people to earn money from home with great flexibility. Customers can also browse profiles and diverse menus to order their favorite food for the next day. This pre-ordering process, in addition to increasing productivity and more accurate planning, leads to reduced food waste and higher customer satisfaction.

1.1 Problem Statement

With the expansion of modern life, people have less time to cook at home, and at the same time, the desire to consume healthier and homemade foods is increasing. On the other hand, immigration, cultural diversity, and different food styles have made people interested in experiencing local and international foods.
Meanwhile, many women - especially immigrants or women with family responsibilities - do not have the possibility to actively participate in the labor market due to various reasons, such as the lack of sufficient capital to establish a restaurant, the responsibilities of home and children, and the inflexibility of traditional work environments.
On the other hand, food waste has also become one of the serious environmental and economic problems in the food industry, which mainly originates from overproduction, inaccurate demand forecasting and the absence of smart ordering mechanisms. Mamafood aims to help reduce this waste by creating a demand-based and pre-order model.

1.2 Objectives

- Develop a user-friendly web platform where users can easily register, browse, and select home-cooked meals, and have a simple and enjoyable food ordering experience.
- Enable easy registration for chefs and create a personal profile that includes a menu, contact information, available times, and food images.
- Create a search feature based on location (city) and provide filters based on food type (Iranian, Austrian, Arabic, etc.), dietary features (vegetarian, halal, gluten-free, etc.).

- Use a pre-order system (next-day delivery) to help chefs manage resources and ingredients optimally and prevent waste.
- To maintain separate sign-up processes for chefs and customers, simplifying the initial onboarding experience for each type of user.

Given that this platform is initially designed for local implementation, efforts have been made to integrate functionalities common in established market platforms such as Foodora or Lieferando, thereby aiming to achieve these objectives effectively. Furthermore, the platform's potential for success can be evaluated in the future through online deployment and by gathering feedback from expert users.

1.3 Motivation and Background

This project was formed from a combination of several main motivations. First, supporting women's empowerment and promoting small home businesses as a means of creating financial and social independence. Many women with excellent cooking skills are unable to enter the job market for various reasons.
On the other hand, the demand for healthier, more traditional and more diverse cuisine is growing, especially in immigrant-friendly cities. This demand provides an opportunity for home cooks to share their art with the public.
On the technological side, this project aligns with my personal interest in web systems design, user experience (UX) and the development of digital platforms. The project covers a mix of social, environmental and technical concerns.

1.4 Related Work

Platforms such as Lieferando, Foodora, and Mjam primarily work with restaurants and focus on speedy food delivery. These platforms often limit their services to professional restaurants and ignore home cooking or independent chefs.
In contrast, some new platforms such as Cookunity and Shef in the United States have attempted to introduce independent and home-cooked cooks to the market. By providing technical, marketing, and other infrastructure, these types of platforms have paved the way for local cooks to generate income.
In Iran, there is also a platform called Mamapez that uses a similar approach to connect home cooks – mostly women – with customers. Mamapez was one of the first serious attempts to build a home-cooked food sales ecosystem in Iran by providing home-cooked meals to organizations, offices, and regular users.
By following the examples mentioned above, Mamafood is trying to reduce food waste through careful planning of orders by adding features such as daily pre-ordering and introducing people to the food of other nations.

2 System Analysis and Design

System analysis is the very first step in any system development and the critical phase where developers come together to understand the problem, needs, and objectives of the project. It is the first critical step in understanding and defining the requirements necessary to achieve the goals of the Mamafood platform.
This section presents a detailed and practical analysis of the system requirements and strategic design decisions behind Mamafood, a home-cooked meal marketplace. The goal is to develop a platform that is intuitive, user-friendly, and responsive. The following are the steps involved in the SA/SD process:

- Requirements gathering
- Structured Analysis
- Data Modeling
- Process Modeling

The analysis focuses on understanding the needs of all stakeholders—including customers, home chefs (Koch), and administrators—by clearly defining system functionalities and designing a robust architecture and database structure. Key features include menu and food categorization, chef verification, order management, and pickup coordination. Additionally, the system has been designed with future expansion in mind, including support for secure online payment integration in later development stages.

2.1 Requirements Analysis

The first step in the SA/SD process is to gather requirements from stakeholders, including users, customers, and business partners. This section explores the full scope of the problem by identifying stakeholder needs—such as those of customers, home chefs, and administrators—while clearly outlining the system's boundaries.
Requirements analysis is an essential process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.
Functional requirements define the specific behavior or functions of a system. In contrast, non-functional requirements specify how the system performs its tasks.

2.1.1 Functional Requirements

User Registration and Login as customer and chef

There are two user roles: Customer, Chef/Restaurant (Koch). The role of Admin and its dashboard is for future development. Users can register as a customer or

chef and log in using their email and password credentials. Role assignment occurs during registration and determines system access and features.
The registration process flowchart for the user (see Figure 1 User Registration Process) as a customer and as a chef/Restaurant (see Figure 2 Chef Registration Process) is as follows. The users as customers are activated automatically, but the chef will be activated after the document review.

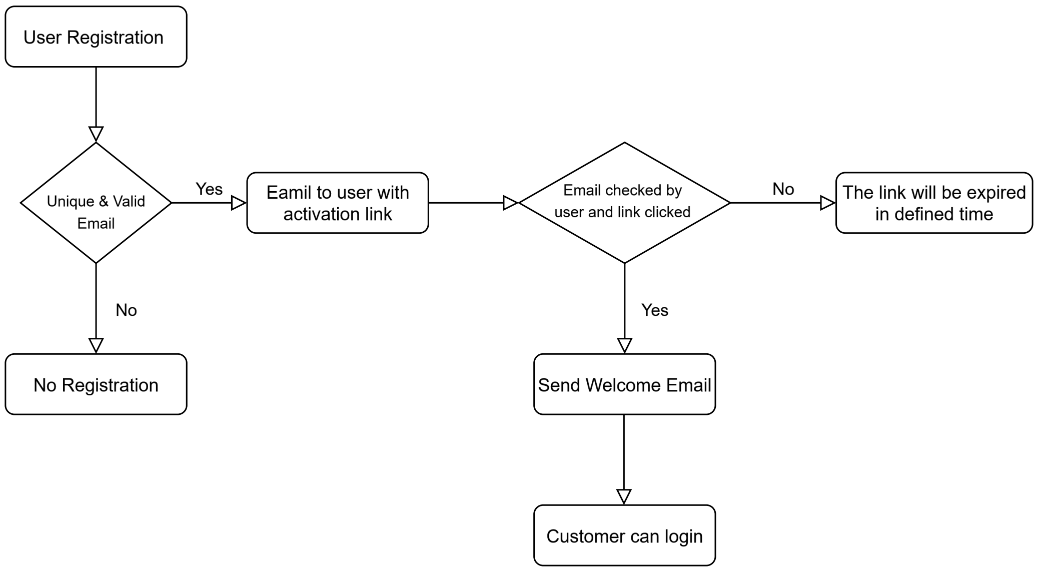

User Registration
Unique & Valid Email
Yes
No
No Registration
Eamil to user with activation link
Email checked by user and link clicked
No
The link will be expired in defined time
Yes
Send Welcome Email
Customer can login

Figure 1: User Registration Process

Chef (Koch) - Restaurant Management: Chefs (Koch) are able to manage their restaurant profiles, which include the restaurant or chef name, a description, the physical address, and the associated cuisine type (e.g., Austrian, Persian).

Chefs - Food Management: Chefs have comprehensive food management tools that allow them to create, edit, and delete food items; each item includes a name, categories (e.g. vegan, main course), inventory status (quantity control), approximate preparation or delivery time, detailed descriptions such as ingredients, and related images.

Discounts and Offers: Chefs can configure discounts based on: Quantity of items purchased or Total order value

Food Quantity Order Management

It will be designed for future development:
- Chefs can define purchase limits per order—for example, restricting each customer to order only a certain percentage (e.g., 10%) of the available stock—to ensure fair distribution and risk management.

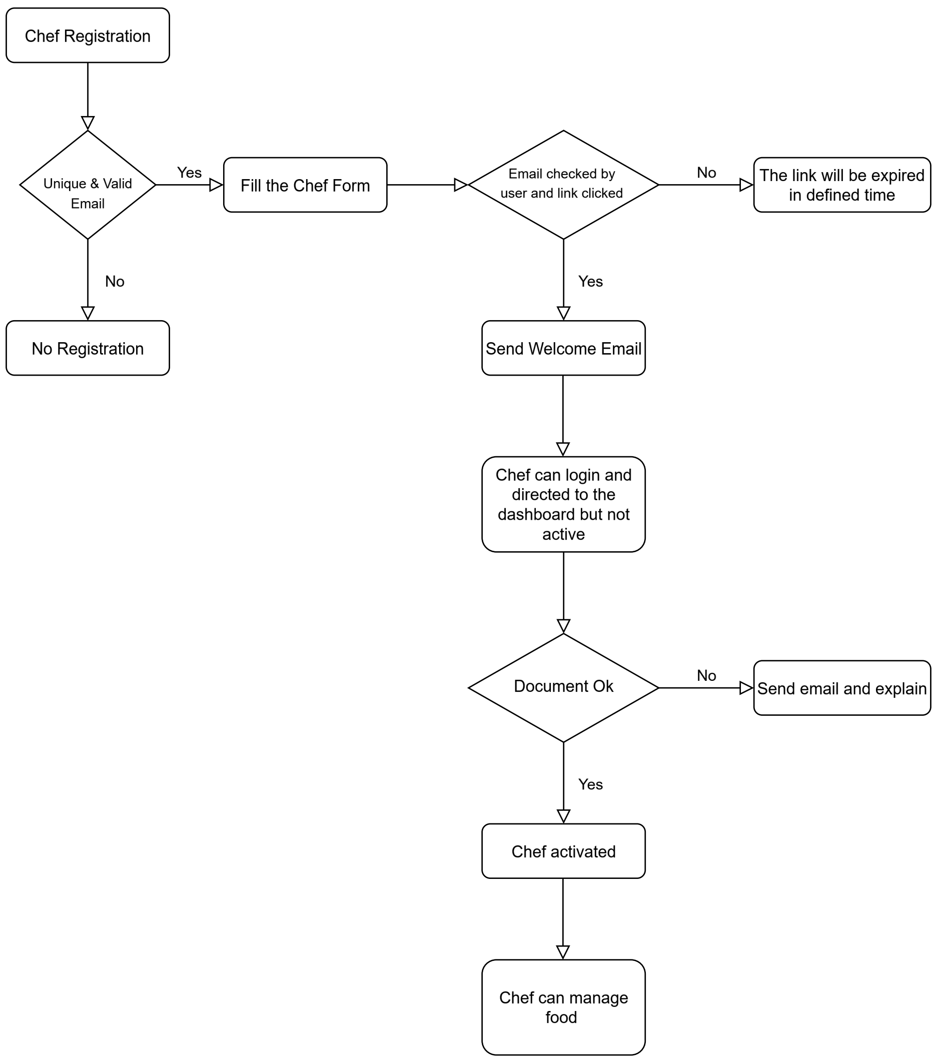


Figure 2: Chef Registration Process

Search Options: Customers can browse restaurants by location (e.g., city) or name and filter food by:

- Food type
- Category (e.g., vegan, dessert)
- Cuisine type
- Price range

Order Placement

- Customers can place orders from selected restaurant menus.
- Orders are directly routed to the respective Chef (Koch).
- Customers pick up their orders; no delivery service is currently supported.For more detail see Figure 3.

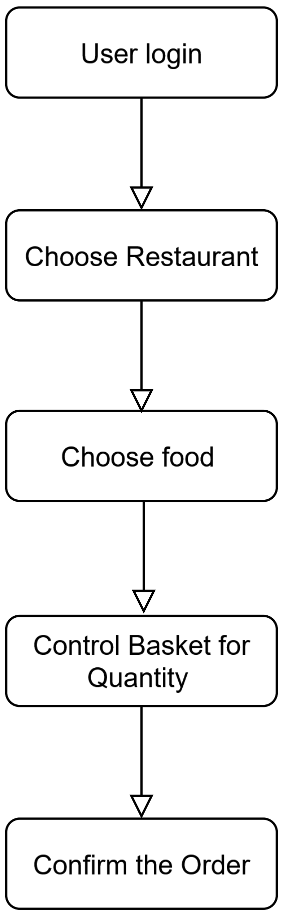


Figure 3: Customer-Order Diagram

Category and Cuisine Management

- Admins/Website Owner manage standardized food categories (e.g., Dessert, vegan) and cuisine types (e.g., Austrian, Iranian).

Admin Controls

- Admins /Website Owner directly through database management:
  - Enabling/disabling restaurant profiles
  - Managing platform-wide content (categories, cuisines)

2.1.2 Non-Functional Requirements

These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements.

- The app should be user friendly.
- The app should prevent SQL injection vulnerabilities.
- Access levels should be enforced based on user roles (e.g., customer, chef).
- The addition of new restaurants and food items should be possible without requiring changes to the database structure.
- The platform should be responsive.
- The system will initially support German and be built to easily accommodate multiple languages in the future (e.g., English).
- The system must comply with all relevant local food regulations and other related legal rules.

2.1.3 Stakeholders

Stakeholder Overview
There are two types of stakeholders: Internal and external, and it is important to analyze their behavior and power of influence.
The successful development and long-term operation of the Mamafood platform rely on addressing the unique needs and expectations of its key stakeholders.

Customers

Customers are the end-users who seek convenient, local, home-cooked meal options.

Chefs (Koch / Restaurants)

Chefs include home cooks and households offering meals through the platform.

Administrator/Website Designer

Administrator roles are crucial to ensuring the platform operates smoothly, securely, and as expected, by managing user registration, verifying chefs, and overseeing content such as food categories, cooking styles, and menus.

2.2 System Design

System Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating user requirements into a detailed blueprint that guides the implementation phase. The goal is to create a well-organized and efficient structure that meets the intended purpose while considering factors like scalability, maintainability, and performance.

This section outlines the architectural design and key decisions that shape the development of the Mamafood food ordering platform. It translates the functional and non-functional requirements into a concrete system structure, detailing how components interact, both logically and over time.
The system design is mainly divided into two parts: the backend and the frontend. Table 1 provides a comprehensive list of used Technology and relevant components and Figure 4 shows Relations/Dependencies between the components:

Component	**Technology**	**Responsibilities**
Frontend	Vue.js, HTML, Bootstrap	User interface, input handling, responsive design, API calls to backend
Authentication & Authorization	OAuth2, JWT	User authentication, token-based access control, role-based permissions
Database	MySQL	Persistent data storage (users, orders, menus), schema relationships, transactions
Containerization	Docker	Local development environment, isolated containers for backend and database
API Structure	Modular FastAPI Routers	Organizing API endpoints, clear routing for guest and authenticated access
Backend API	FastAPI (Python), SQLAlchemy	Business logic, data validation, secure APIs, database interaction

Table 1: List of used Technology

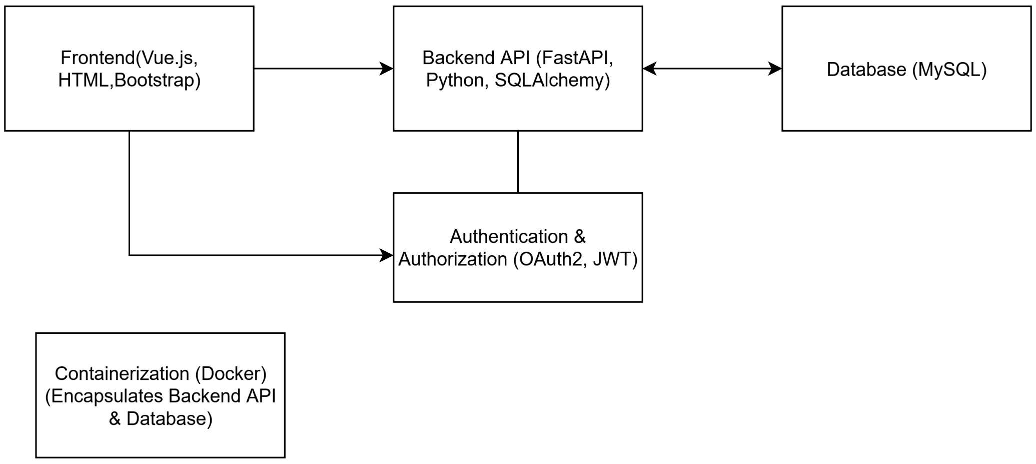

Frontend(Vue.js, HTML,Bootstrap)
Backend API (FastAPI, Python, SQLAlchemy)
Database (MySQL)
Authentication & Authorization (OAuth2, JWT)
Containerization (Docker) (Encapsulates Backend API & Database)

Figure 4: Relations/Dependencies between the components

2.2.1 Database Design

The database serves as the backbone of the MamaFood platform, responsible for storing and managing all persistent data. This includes information about users, restaurants, foods, categories, cuisines, orders, and addresses.
To provide a clear vision of these relationships, two key diagrams have been prepared:

- The Entity-Relationship Diagram (ERD), shown in Figure 5, Figure 6 and Figure 7, illustrates the connections and relationships between the tables related to the restaurant, food management, and order management in our database schema.
- The UML Class Diagram, presented in Figure 8, primarily represents the system's entities, their attributes, and relationships for restaurants. While it can conceptually depict high-level operations like CRUD (Create, Read, Update, Delete), its main role here is to serve as a blueprint for data models. This blueprint informs SQLAlchemy ORM mappings and defines Pydantic models for FastAPI's API request and response validation.

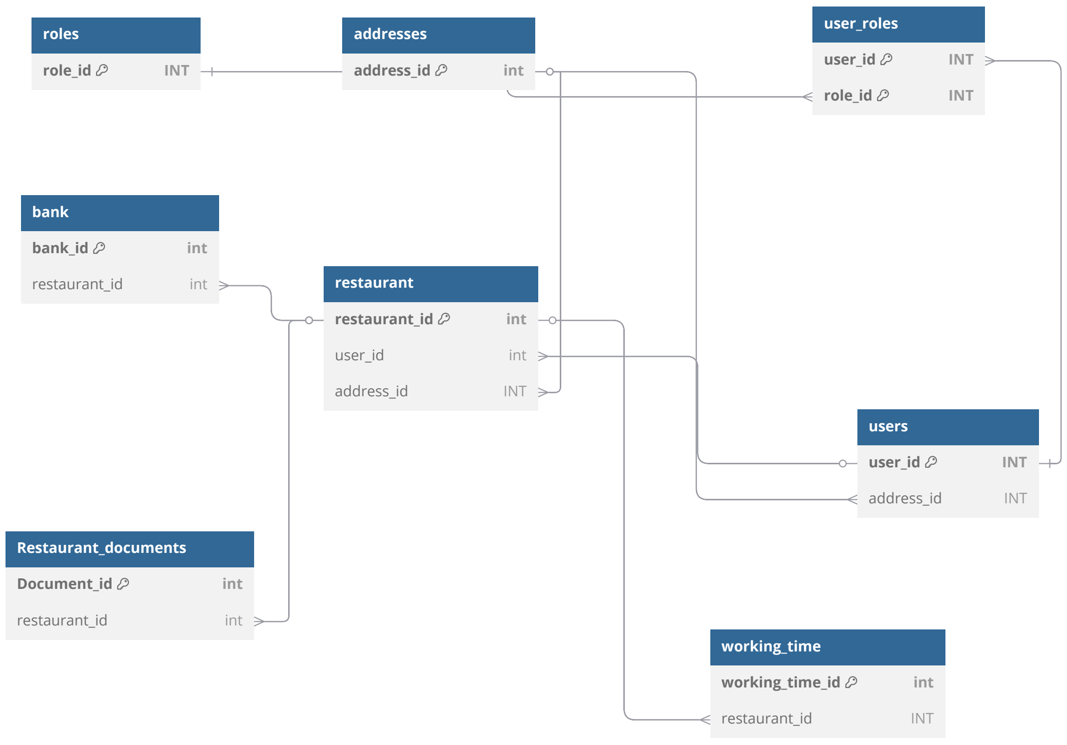


Figure 5: ERD Chef Diagram

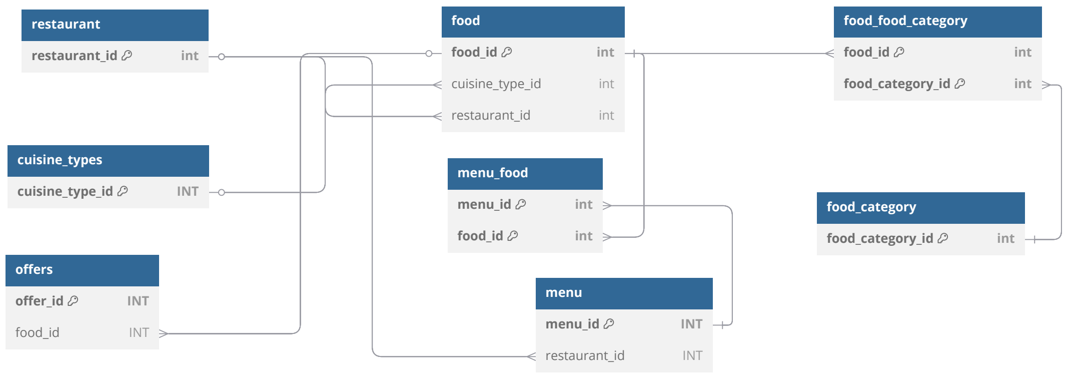


Figure 6: ERD Food Diagram

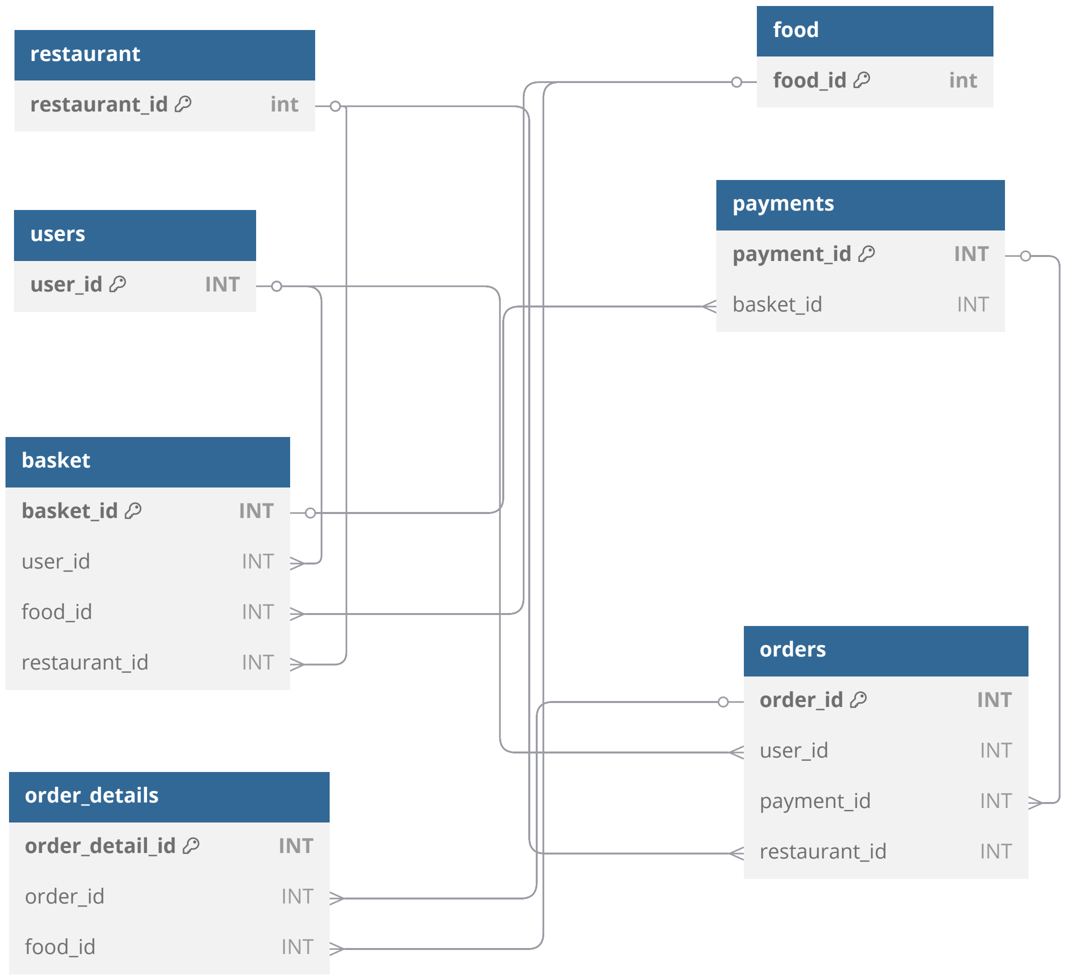


Figure 7: ERD Order/Payment Diagram

The data structure is defined, and the relationships between entities are established. A relational database is ideal for this purpose, as it effectively manages these relationships using foreign keys, ensuring data integrity and consistency. Since the types of data are stable and not subject to frequent changes, the fixed schema provided by a relational database is efficient and suitable for the system.

MySQL is a good choice due to its broad community support, mature ecosystem, and availability of robust tools for development and maintenance. While there are other database options such as PostgreSQL and SQLite, the widespread adoption of MySQL by large companies, including its use by Oracle, demonstrates its strength and makes it a viable choice for the MamaFood platform. For a more detailed discussion, see Section 3.1.3.

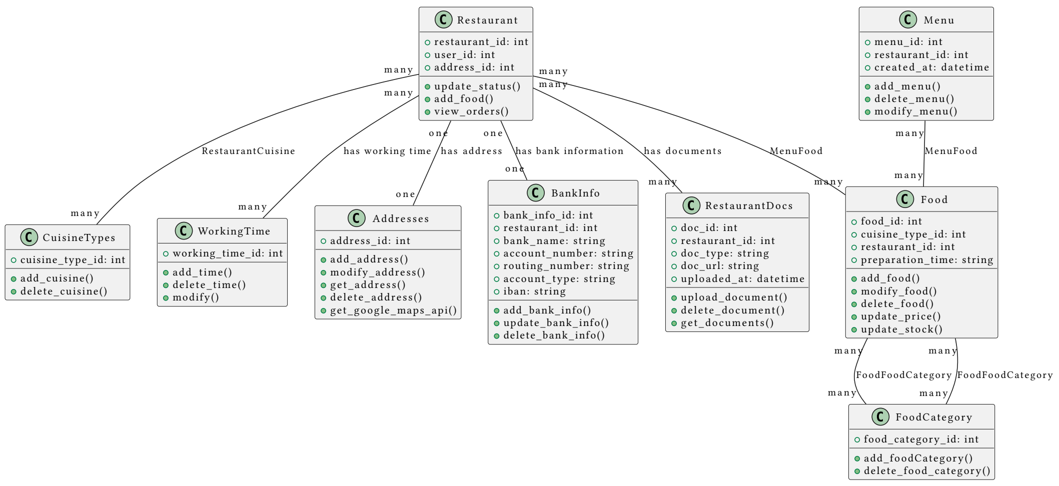


Figure 8: Restaurant Class Diagram

2.2.1.1 Tables

The tables are as below :

- roles: Stores the different roles of Users
- addresses: Stores address of chefs/Restaurants(Koch)
- users: Stores user information
- user_roles: Manages the many-to-many relationship between users and roles.
- cuisine_types: Lists available cuisine types (e.g., Italian, Persian, Indian) to categorize food and restaurants
- restaurant :Stores data about restaurants (chefs or kochs)
- restaurant_cuisine: Links restaurants to their offered cuisine types
- working_time: Defines the opening and closing hours for each restaurant for every day of the week.
- bank: Stores bank details for each restaurant .
- Restaurant_documents: Stores uploaded legal or operational documents (e.g., Health Certificate) for restaurant verification.
- food: Stores detailed information about food items such as name, description, prices, stock, and preparation time.
- food_categoryStores food categories (e.g., Main dish, Dessert, Vegan)

- food_food_category: Establishes a many-to-many relationship between food items and food categories.
- offers: Stores promotional offers applicable to food items or orders, including type, value, and duration.
- menu: Represents a menu that belongs to a restaurant, consisting of various food items grouped under it.
- menu_food: Links food items to menus (many-to-many relationship).
- basket: Temporary shopping cart where users add food items before placing an order.
- payments: Tracks payments made by users for items in their basket, including status and amount.(for future development)
- orders: Records order details
- order_details: Contains the detailed list of food items within a specific order, along with quantity and price.

2.2.1.2 Model Development(ORM)

In software development, there are two main approaches to managing the relationship between application code and database schema: code-first and database-first (or model-first).
Code-first: With this approach, the database schema is defined directly in the ORM models, and then the database is extracted.
Database-first (or model-first): In this approach, the database schema is first designed and created, usually using a database design tool or straight SQL.
For the MamaFood platform, the database-first approach was followed carefully. We started with a thorough analysis of the relationships and entities required for the system data. This led to the creation of an ERD (Entity-Relationship Diagram) that visually depicted these relationships. Based on this detailed design, the actual database tables were designed and implemented. After creating a robust database, we started generating application code.

Using SQLAlchemy allows us to interact with the MySQL database through Python objects, ensuring better maintainability.“

```
python -m sqlacodegen_v2 mysql+pymysql://root@localhost/mamafood >
models.py
```

This command creates a models.py file containing SQLAlchemy ORM classes such as User, Restaurant, Address, Token, CuisineType, Food, and more, all based on the database schema.
In the next step, it is necessary to create Pydantic schemas. These are used for request and response validation in FastAPI.
Finally, CRUD parts are created based on the class diagram.

2.2.1.3 FastAPI Endpoints

REST API endpoints are created in main.py using FastAPI. These endpoints use Pydantic schemas for data validation and utilize functions from the CRUD layer to interact integrally with the database.
It's possible to test these REST API routes using FastAPI's interactive documentation (Swagger UI) or Postman. An example with Swagger UI is shown in Figure 9.

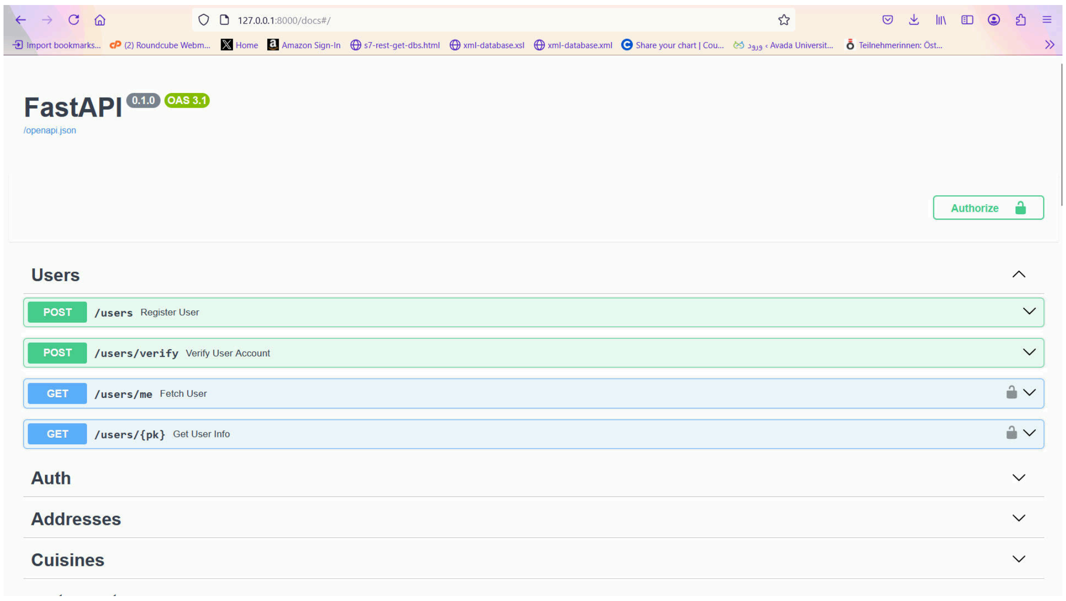


Figure 9: Fastapi End Point

2.2.2 Security Considerations

Security is essential for protecting user data and maintaining platform integrity.

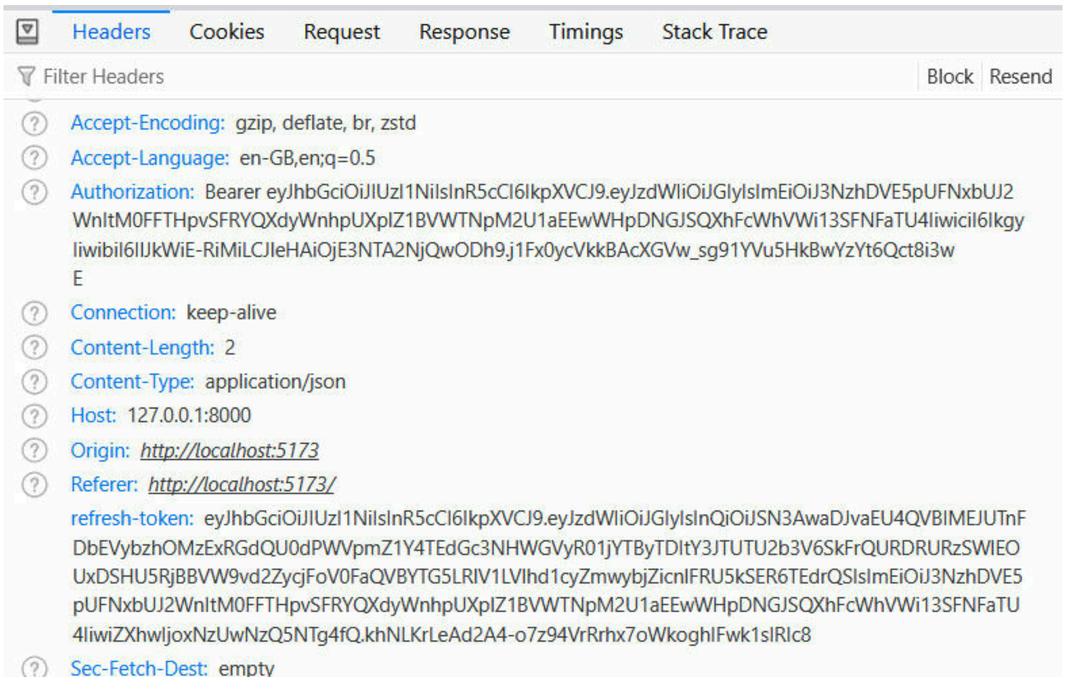
2.2.2.1 Data Protection

Password Security: To provide strong data protection and reduce the risk of password disclosure in the event of a hack or system theft, all user passwords are securely hashed using bcrypt before being stored in the database. This step prevents the password from being exposed in plain text and significantly increases security.

Authentication, Authorization, and Token Management

The MamaFood platform uses token-based authentication with JSON Web Tokens (JWT) to ensure secure access. The system uses two main types of tokens:
Access Tokens: Short-term tokens used for immediate API authorization.
Refresh Tokens: Long-term tokens used only to obtain new access tokens after the current token expires.

When a user logs in, the backend issues both an access token and a refresh token. When the access token expires, the client automatically uses the long-term refresh token to obtain a new access token without requiring the user to log in again.
This design increases security by limiting the lifetime of the frequently used access token and reducing the exposure of the refresh token. This avoids the need to repeatedly send user credentials.
Figure 10 provides an example of an authorization token (bearer) and refresh token.



```
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-GB,en;q=0.5
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJGlyIsImEiOiJ3NzhDVE5pUFNxbUJ2
WnItM0FFTHpvSFRYQXdyWnhpUXpIZ1BVWTNpM2U1aEEwWHpDNGJSQXhFcWhVWi13SFNFaTU4Iiwicil6Ikgy
IiwibiI6IlJkWiE-RiMiLCJleHAiOjE3NTA2NjQwODh9.j1Fx0ycVkkBAcXGVw_sg91YVu5HkBwYzYt6Qct8i3w
E
Connection: keep-alive
Content-Length: 2
Content-Type: application/json
Host: 127.0.0.1:8000
Origin: http://localhost:5173
Referer: http://localhost:5173/
refresh-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJGlyIsInQiOiJSN3AwaDJvaEU4QVBIMEJUTnF
DbEVybzhOMzExRGdQU0dPWVpmZ1Y4TEdGc3NHWGVyR01jYTByTDItY3JTUTU2b3V6SkFrQURDRURzSWlEO
UxDSHU5RjBBVW9vd2ZycjFoV0FaQVBYTG5LRlV1LVlhd1cyZmwybjZicnlFRU5kSER6TEdrQSIsImEiOiJ3NzhDVE5
pUFNxbUJ2WnItM0FFTHpvSFRYQXdyWnhpUXpIZ1BVWTNpM2U1aEEwWHpDNGJSQXhFcWhVWi13SFNFaTU
4IiwiZXhwIjoxNzUwNzQ5NTg4fQ.khNLKrLeAd2A4-o7z94VrRrhx7oWkoghlFwk1sIRIc8
Sec-Fetch-Dest: empty
```

Figure 10: Authorization and Refresh Tokens

2.2.3 Status Codes

To effectively handle feedback on the success or failure of API requests, various HTTP status codes are used. Status codes in the 2xx range indicate success, while other ranges indicate different results (e.g., 4xx for client errors, 5xx for server errors). Below are definitions of some common status codes, along with screenshots of sample code on the Mamafood platform:

Status Code: 200 (OK)

- When a request has succeeded and system is returning the requested data. It is commonly used with GET, PUT, or DELETE requests when the server successfully processes the request and sends a response with the data or a success message.
- Example Usage:

    - Returning a list of restaurants or users
    - Successfully processing a deletion (although 204 can also be used here)

Status Code: 201 (Created)
- When a new resource is successfully created on the server. It is typically used with POST requests. The server should return the newly created resource or a reference to it.
- Example Usage:
    - Creating a new user
    - Adding a new restaurant or menu
    - Registering a new customer or chef (koch)

Status Code: 404 (Not Found)

- When the requested resource could not be found on the server. It indicates that the client may be using an incorrect ID or endpoint.
- Example Usage:
    - Looking for a restaurant by ID that does not exist
    - Requesting user details with a non-existent user ID

2.2.4 Data Flow

This section draws simplified data flow diagrams for key processes in the MamaFood platform. These diagrams show how data moves between the frontend, backend, and database for various operations.

User Registration & Login

- Customer/Seller/Admin provides email and password to Frontend.
- Frontend sends credentials to Backend Authentication API.
- Backend interacts with Firebase Authentication to create/verify user.
- If registration, Backend creates a new document in users collection and user_roles collection based on chosen role.
- If login, Backend verifies credentials and returns authentication token to Frontend.
- Frontend stores token and updates UI based on user role.

Order Placement (Customer)

- Customer browses restaurants and fooditems via Frontend.
- Customer adds food_items to basket (local state initially, then synchronized with baskets table).

- When ready, Customer initiates checkout on Frontend.
- Frontend sends basket contents to Backend checkout API.
- Backend validates basket (stock, availability, pricing) against food_items and restaurants table.
- Backend initiates a payment record in payments collection (status: ‘pending’).
- Backend communicates with Payment Gateway (external API, not shown here).
- Upon successful payment, Backend updates payment status to ‘completed’.
- Backend creates an order document in orders collection and corresponding order_details documents, transferring data from the basket.
- Backend notifies the relevant restaurant (via real-time update/push notification) about the new order.
- Backend clears the basket for the user.
- Frontend displays order confirmation to Customer.

  Figure 11 provides a comprehensive illustration of the complete data flow for the user registration.

  Figure 12 provides a comprehensive illustration of the complete data flow for adding food by restaurant.

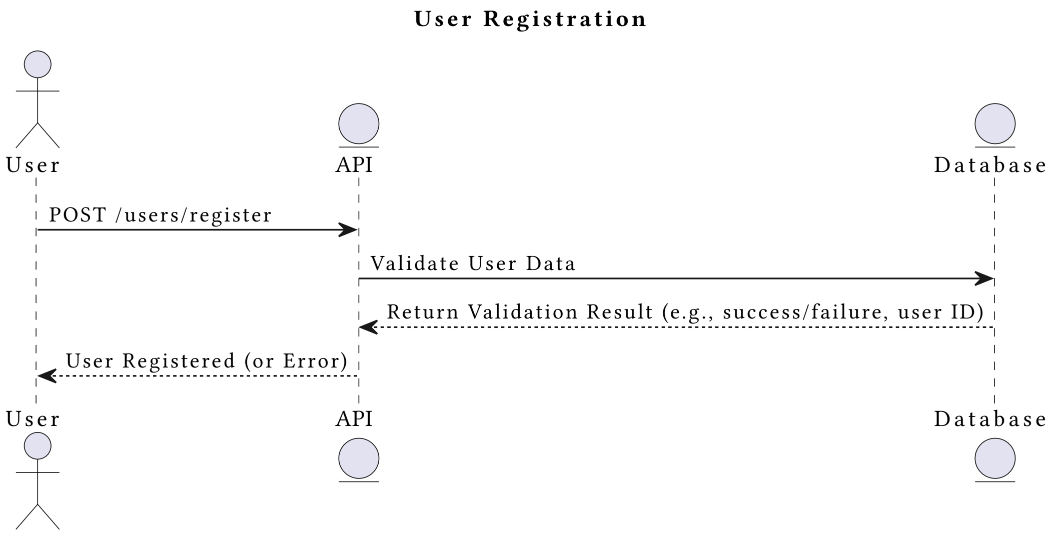


Figure 11: Dataflow of Food Adding .

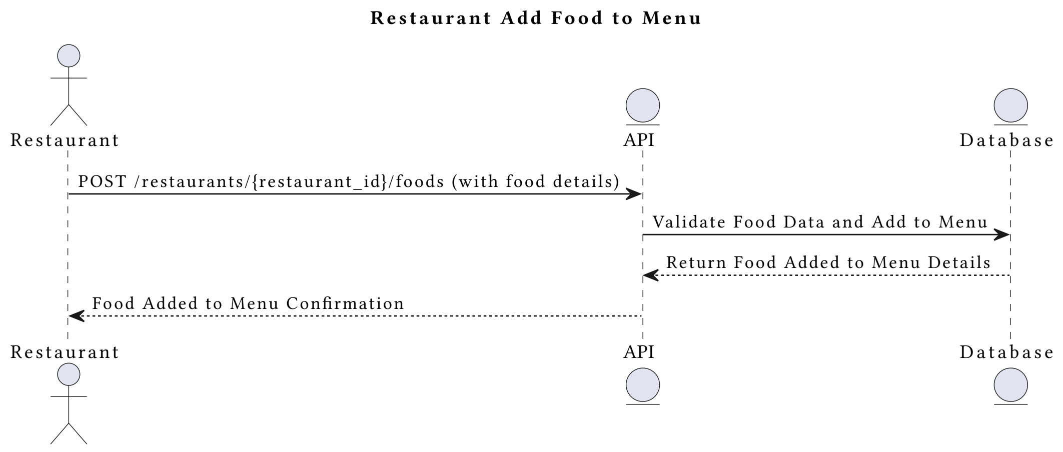

Restaurant Add Food to Menu
Restaurant
API
Database
POST /restaurants/{restaurant_id}/foods (with food details)
Validate Food Data and Add to Menu
Return Food Added to Menu Details
Food Added to Menu Confirmation
Restaurant
API
Database

Figure 12: Dataflow of Food Adding .

Restaurant Order Processing (Seller)

- Seller Frontend receives real-time updates for new orders from the orders collection (filtered by restaurant_id).
- Seller views order_details and updates order status (e.g., 'accepted', 'preparing', 'ready_for_pickup') via Frontend.
- Frontend sends status update to Backend update_order_status API.
- Backend updates status field in the orders table.
- Customer Frontend receives real-time updates on order status changes.

2.2.5 UX Considerations

User interface

UI design plays a pivotal role in the success of any website or digital product. Its importance stems from several key aspects, as highlighted in:

- User Experience: A well-designed UI ensures that users can interact directly and efficiently with the platform, resulting in a positive overall experience.
- Engagement: An engaging UI encourages users to spend more time on the platform and use its features effectively.
- Branding: UI design is crucial for creating a consistent and recognizable brand identity.
- Efficiency: A clear and logical UI enables users to perform tasks quickly and with minimal effort.

Figure 13 shows the homepage of the MamaFood website on wider screens.

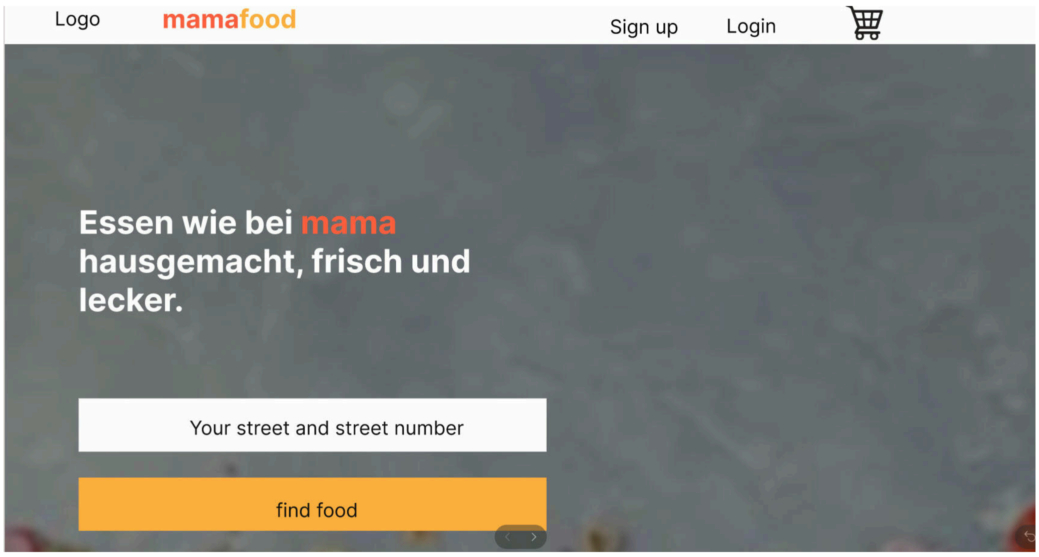

Logo
mamafood
Sign up
Login
Essen wie bei mama hausgemacht, frisch und lecker.
Your street and street number
find food

Figure 13: A Widescreen Website view.

General UI Requirements

- Responsive layout for both mobile and desktop devices
- Easy to User
- Having UX features (search and filter, dashboard for chef/ restaurants and customer)

3 Implementation

The Mamafood implementation is based on the principles of modern web development with a focus on modularity, scalability, security, and simplicity. The application consists of a front-end and a back-end. Each component has been selected based on its performance, maintainability, and support for rapid development from a developer ecosystem.

3.1 Backend

"FastAPI is a modern, fast (high-performance) web framework for building APIs with Python, leveraging standard Python type hints. Its performance is notably high, on par with technologies like NodeJS and Go, primarily due to its reliance on Starlette and Pydantic. This makes it one of the fastest Python frameworks available.
With FastAPI, development is significantly quicker, and the risk of human error is reduced. This is attributed to excellent editor support and comprehensive code completion, leading to less time spent debugging. The framework is intuitive, making it easy to learn and use, and requiring minimal time to consult its documentation. Furthermore, FastAPI is robust, facilitating the creation of production-ready code with automatic, interactive documentation. FastAPI is fully compatible with well-known standards of APIs (i.e. OpenAPI and JSON schema).

FastAPI Features:
- Automatic Documentation
- Python Type Hints
- Pydantic for the data validation
- Dependency Injection
- Asynchronous Support
- Security Features

3.1.1 FastAPI Features Overview

Automatic Documentation

Interactive API documentation and exploration web user interfaces. As the framework is based on OpenAPI, there are multiple options, 2 included by default. Swagger UI, with interactive exploration, call and test API directly from the browser.
Swagger UI allows anyone — be it development team or end consumers — to visualize and interact with the API's resources without having any of the imple-

mentation logic in place. It's automatically generated from OpenAPI (formerly known as Swagger) Specification, with the visual documentation making it easy for back end implementation and client side consumption.

Python Type Hints

One of FastAPI's best features is its use of Python's type hints. These "type hints"—sometimes called "type annotations"—are a special way Python lets one declare the kind of data a variable should hold.
By explicitly stating variable types, editors and other development tools provide much better support. When annotating function parameters and return types with these hints, it doesn't just make code easier to read. It also enables FastAPI to automatically validate incoming data and create perfect API documentation, all on its own.

Pydantic for the data validation

Pydantic is the most widely used data validation library for Python.Pydantic is powered by Python's type hints, meaning schema validation and data serialization are seamlessly controlled by type annotations. This approach reduces the amount of code to write, minimizes the learning curve, and provides excellent integration with IDEs and static analysis tools.
Its core validation logic is written in Rust, making it one of the fastest data validation libraries for Python. Pydantic can also produce JSON Schema, which significantly simplifies integration with various other tools. It offers both strict and lax modes, allowing it to either precisely enforce data types or attempt to coerce data types where appropriate. Furthermore, it supports standard Python library types, including dataclass and TypedDict.
Pydantic provides powerful customization options, enabling developers to modify validators and serializers to alter data processing in numerous ways. It is a widely adopted library within the Python ecosystem, with over 8,000 packages on PyPI relying on it, including FastAPI. Pydantic itself is downloaded over 360 million times per month and is utilized by all FAANG companies. Installation is straightforward: simply run pip install pydantic.

Dependency Injection

FastAPI supports dependency injection, a system that's easy to use for declaring dependencies for endpoints. This really helps keep code modular, testable, and maintainable. It's an powerful system.
The framework handles things automatically. All dependencies can ask for data from requests and even augment the path operation constraints and the automatic documentation. Dependencies can also have their own dependencies, creating a

hierarchy or “graph” of dependencies, and seamlessly inject things like database connections, authentication, and more into routes.
There’s automatic validation too, even for path operation parameters defined within these dependencies. The system supports complex features such as user authentication and database connections.

Asynchronous Support

Modern versions of Python have support for “asynchronous code” using something called “coroutines”.
It’s called “asynchronous” because the computer / program doesn’t have to be “synchronized” with the slow task, waiting for the exact moment that the task finishes, while doing nothing, to be able to take the task result and continue the work.
The Keywords are used to write asynchronous endpoints, making it well-suited for handling I/O-bound tasks (Tasks like reading from a database, making external API calls, or reading/writing files) and improving the overall responsiveness of application or web.

Security Features

There are many ways to handle security, authentication and authorization.FastAPI provides several tools to deal with Security easily, rapidly, in a standard way. Support for OAuth2, JWT (JSON Web Tokens), and automatic validation of request data to prevent common security vulnerabilities like SQL injection and cross-site scripting (XSS) attacks

3.1.2 FastAPI Security Features and tools

It supports various standard methods like OAuth2, OAuth 1, OpenID Connect, and OpenAPI (formerly Swagger) for defining security. FastAPI is fundamentally based on OpenAPI.

OpenAPI Specification and Security Schemes

FastAPI is built on top of OpenAPI. OpenAPI provides several features, including automatic interactive API documentation and automatic code generation. OpenAPI also provides a way to define different “security schemes”.
by using these schemes, it is possible to take full advantage of all the tools based on the OpenAPI standards, including these interactive documentation systems.
OpenAPI defines the following security schemes:

- API Key: An application-specific key that can be passed via: Query parameters, a header ,a cookie.
- http (HTTP Authentication): Standard HTTP authentication schemes, including:
  - bearer (Token Bearer): A token in the authorization header with the Bearer prefix (derived from OAuth2).
  - HTTP Basic Authentication.
  - HTTP Digest and similar.
- oauth2: All OAuth2 methods for managing security (known as "flows"). Several of these flows are suitable for building OAuth 2.0 authentication providers (e.g. Google, Facebook, Twitter): implicit, clientCredentials, authorizationCode. However, one specific flow, password, is very useful for managing authentication directly within the application itself.
- openIdConnect: Provides a way to automatically discover OAuth2 authentication data (defined in the OpenID Connect specification).

OAuth2, JWT, and Password Hashing

OAuth2 with Password (and Hashing), Bearer with JWT tokens
JWT stands for "JSON Web Tokens." It's a standard for encoding a JSON object into a long, dense string without spaces.
While a JWT is not encrypted (meaning its information can be recovered), it is signed. This signature allows a recipient to verify that the token was genuinely issued by the expected sender.
Password hashing involves converting a password into a unique, one-way sequence of characters (a hash). If a database is stolen, only these hashes are exposed, not the original plain-text passwords.
Handling JWT tokens involves creating and using a random secret key to sign and verify the tokens.
The steps for implementing this in FastAPI are:
Install PyJWT: pip install pyjwt Install passlib: pip install "passlib[bcrypt]" Handle JWT tokens: This includes creating, encoding, and decoding tokens using the secret key. Update dependencies: Modify existing functions (like get_current_user) to work with JWTs. Update token path operation: Configure the /token endpoint to issue JWT access tokens upon successful login. Check FastAPI - Swagger UI: Verify that the API documentation reflects the new security scheme. Check the developer tools: Inspect network requests to confirm tokens are being sent and received correctly.

FastAPI Tools for Security

FastAPI significantly simplifies the implementation of security mechanisms by providing multiple tools for each of these security schemes in the fastapi.security module.
There are tools for:
- API Key Security Schemas
- HTTP Authentication Schemes
- OAuth2 Security Schemes (and related forms)
- OpenID Connect
- Utility for Scopes

3.2 ORM

ORM is concerned with helping application to achieve persistence. Persistence simply means that we would like our program's data to outlive the current process.ORM helps us keep persistent state in a relational database.
Object-relational mapping (ORM) is a key concept in the field of Database Management Systems (DBMS), addressing the bridge between the object-oriented programming approach and relational databases. ORM is critical in data interaction simplification, code optimization, and smooth blending of applications and databases. With Object-Relational Mapping, it becomes much easier to work with an object-oriented programming language and relational database. Fundamentally, it acts as a translator, translating data between the database and the application without any hitch. ORM enables developers to work with objects in their programming language which are mapped to corresponding database entities, such as tables, views, or stored procedures.
SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

Tools (ORM Mapped Class Configuration) :

ORM Mapping Styles: The Declarative Mapping is the typical way that mappings are constructed in modern SQLAlchemy.

3.3 Database

A relational database, or relational database management system (RDMS), stores information in tables. Often, these tables share information, forming a relationship between them. This is where a relational database gets its name.

3.3.1 Advantages of relational databases

There are advantages to using relational databases, such as ACID compliance, data accuracy, normalization, and simplicity. However, when flexibility, scalability, and higher performance are required—especially due to the nature and structure of the data—it may be necessary to use non-relational databases, which are better suited for those needs.
Table 2 clearly illustrates the main differences between relational and non-relational databases.

Aspect	**Non-Relational database (NoSQL)**	**Relational database (SQL)**
Data model	Uses a variety of data models such as document, key-value, wide-column, and graph.	Uses a tabular data model with rows and columns.
Schema	Typically schema-less or flexible schema. Allows for varied data structures within the same database.	Requires a predefined schema with a fixed structure.
Scalability	Designed for horizontal scalability, can easily scale out by adding more nodes.	Primarily scales vertically, requiring more powerful hardware for scaling.
Query language	No standard query language. Query methods vary based on the type of NoSQL database.	Uses structured query language (SQL) for defining and manipulating data.
Data integrity	Typically eventual consistency. Focus on availability and partition tolerance (CAP theorem).	Strong focus on data integrity with ACID compliance.
Complexity of transactions	Better suited for simpler transactions. Complex transactions can be more challenging to manage.	Ideal for complex transactions and operations requiring joins and multi-record updates with consistency.
Handling of big data	Excellent for handling large volumes of unstructured or semi-structured data.	Can handle large volumes of data but may become complex and less efficient with extremely large datasets.
Flexibility	Highly flexible in terms of data models and schema changes.	Less flexible; schema changes can be complex and disruptive.
Relationships	Not inherently built for data relationships. Relationships can be modeled but often with more complexity.	Excellently manages data relationships through foreign keys and joins.
Use cases	Ideal for unstructured data, real-time applications, big data analytics, and rapidly evolving data requirements.	Well-suited for structured data with clear relationships, requiring complex queries and high data integrity.

Table 2: Comparison between Relational and Non-Relational Databases

3.3.2 MySQL

MySQL is an open source relational database management system (RDBMS) that's used to store and manage data. Its reliability, performance, scalability, and ease of use make MySQL a popular choice for developers. In fact, it is at the heart of demanding, high-traffic applications such as Facebook, Netflix, Uber, Airbnb, Shopify, and Booking.com.
The database chosen was MySQL.MySQL provides concurrency handling, robust ACID compliance, transaction management, and foreign key support—key for managing relational data between users, orders, and menus in production.
MySQL is the world's most popular open-source database management system. Databases are the essential data repositories for all software applications. For example, whenever someone conducts a web search, logs into an account, or completes a transaction, a database stores the information so it can be accessed in the future. MySQL excels at this task.
SQL, which stands for Structured Query Language, is a programming language that's used to retrieve, update, delete, and otherwise manipulate data in relational databases. As the name suggests, MySQL is a SQL-based relational database designed to store and manage structured data.
MySQL is known for being easy to set up and use, yet reliable and scalable enough for organizations with very large data sets and vast numbers of users.

3.4 Frontend

JavaScript frameworks are an essential part of modern front-end web development, providing developers with tried and tested tools for building scalable, interactive web applications. Many modern companies use frameworks as a standard part of their tooling.
A framework is a library that offers opinions about how software gets built. These opinions allow for predictability and homogeneity in an application; predictability allows the software to scale to an enormous size and still be maintainable; predictability and maintainability are essential for the health and longevity of software. The advent of modern JavaScript frameworks has made it much easier to build highly dynamic, interactive applications.

The real problem is this: every time we change application's state, we need to update the UI to match.
JavaScript frameworks were created to make this kind of work a lot easier — they exist to provide a better developer experience. They don't bring brand-new powers to JavaScript; they give developers easier access to JavaScript's powers so developers can build for today's web.

Every JavaScript framework offers a way to write user interfaces more declaratively. That is, they allow developers to write code that describes how UI should look, and the framework makes it happen in the DOM behind the scenes.
By using Vue, there is no need to write functions for building the UI; the framework will handle that in an optimized, efficient way. The only effort here is to describe to Vue what each item should look like.
Table 3 provides a detailed comparison of these frameworks based on various criteria.

Framework	Browser support	Preferred DSL	Supported DSLs
Angular	Modern	TypeScript	HTML-based; TypeScript
React	Modern	JSX	JSX; TypeScript
Vue	Modern (IE9+ in Vue 2)	HTML-based	HTML-based, JSX, Pug
Ember	Modern (IE9+ in Ember version 2.18)	Handlebars	Handlebars, TypeScript

Table 3: Comparison of Frameworks

To choose the appropriate frame work , some items should be considered:
- What browsers does the framework support?
- What domain-specific languages does the framework utilize?
- Does the framework have a strong community and good docs (and other support) available?

3.4.1 Vue

Vue (pronounced /vju:/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries (opens new window).
Core features of the vue:
- Declarative Rendering: Vue extends standard HTML with a template syntax that allows us to declaratively describe HTML output based on JavaScript state.
- Reactivity: Vue automatically tracks JavaScript state changes and efficiently updates the DOM when changes happen.

3.4.2 Vue Router

Vue Router is the official client-side routing solution for Vue. Client-side routing is used by single-page applications (SPAs) to tie the browser URL to the content seen by the user. As users navigate around the application, the URL updates accordingly, but the page doesn't need to be reloaded from the server. Vue Router is built on Vue's component system. The developer configures routes to tell Vue Router which components to show for each URL path.

3.4.3 Vue-Axios

There are many times when building application for the web that developer may want to consume and display data from an API. There are several ways to do so, but a very popular approach is to use axios, a promise-based HTTP client.
The frontend was developed using Vue.js, a progressive JavaScript framework. Vue was preferred over React or Angular because of its smaller bundle size, ease of integration, gentle learning curve, and two-way binding features. Vue allowed the development of reusable, reactive components for dashboards, food displays, and order tracking interfaces. Axios was used for making HTTP requests from Vue to the backend.

3.5 HTTP Status Codes

In HTTP, a numeric status code of 3 digits as part of the response are sent. HTTP status codes were utilized consistently to communicate the outcome of requests:
200 - 299 are for "Successful" responses. These are used the most.
- 200 is the default status code, which means everything was "OK". Another example would be 201, "Created". It is commonly used after creating a new record in the database.

400 - 499 are for "Client error" responses.
- An example is 404, for a "Not Found" response.

500 - 599 are for server errors. they are used rarely. When something goes wrong at some part in application code, or server, it will automatically return one of these status codes.

3.5.1 Technologies summary and Alternatives

Technologies

The selection of core technologies for the MamaFood platform was made based on a balance of performance, development efficiency, maintainability, and community support.

For the backend, Python (FastAPI), along with PyJWT and passlib, was chosen. FastAPI's modern asynchronous support, high performance, and automatic documentation generation were key factors. Its growing adoption and strong community support on platforms like GitHub make it a solid choice for the future. PyJWT and passlib were integrated due to their reliability and comprehensive documentation, which ensures strong and secure authentication with JSON Web Tokens (JWT) and strong password hashing.
Benefits of this backend stack: FastAPI offers excellent asynchronous capabilities, automatic data validation, and superior performance. PyJWT and passlib collectively ensure secure and standardized user authentication and password management.
Cons: As a relatively newer framework, the FastAPI ecosystem may still be maturing, and some of the specialized libraries or community resources may be less extensive than more established frameworks like Django.
For the front-end, Vue.js was chosen primarily for its advanced compatibility, gentle learning curve, and robust ecosystem that facilitates rapid development of a responsive, user-friendly interface. While alternatives like React and Angular offer powerful solutions, Vue.js provided the optimal balance for the scope of the project and the team's familiarity.
The back-end data persistence is managed by MySQL. It was chosen based on its proven reliability, maturity, strong community support, and efficient management of relational data critical for user information, ordering, and consistent menus. Alternatives like PostgreSQL offer similar benefits, but MySQL's widespread adoption and extensive tooling made it a viable choice.
Finally, SQLAlchemy was integrated as an object-relational mapper (ORM). This choice simplifies database interactions by allowing for native Python object manipulation, significantly increasing code readability, maintainability, and reducing repetitive SQL. It provides a robust and flexible abstraction layer over MySQL, essential for efficient backend development.

Alternatives

Table 4 outlines the primary technology stack chosen for the MamaFood application, alongside key alternative technologies that could be utilized for each component.

Component	MamaFood (Chosen)	Key Alternatives
Backend	Python (FastAPI)	Django, Flask, Node.js (Express, NestJS), Go (Gin), Java (Spring Boot), PHP
Frontend	Vue.js	React, Angular, Svelte, jQuery
Database	MySQL	PostgreSQL, SQLite, MongoDB, Oracle, Redis, Neo4j
ORM	SQLAlchemy	Django ORM, Sequelize, Prisma, Hibernate, Eloquent
Authentication	JWT	OAuth2, OpenID, Session, API Key, SAML, MFA

Table 4: Core Technology Stack and Key Alternatives

3.6 Other tools

Beyond the core technology stack, several development and deployment tools were utilized to increase productivity, streamline workflows, and ensure the reliability of the MamaFood platform.
Development Tools: VS Code, GitHub, Docker, Postman
Why they were used: These tools represent industry standards, chosen for their robust features, broad community support, and proven effectiveness in modern software development.
VS Code was chosen for its robust plugin ecosystem, excellent editor support, and large, active user community, all of which collectively increase coding and debugging efficiency.
GitHub was essential for communicating with team members, facilitating collaborative development, maintaining comprehensive version control, and effectively managing project issues.
Docker was used due to its suitability for local development with technologies like Python. It also aids in simulating functionalities, such as sending emails synchronously with the backend, and in viewing data through exposed ports.
Postman was essential for the efficient development, testing, and debugging of API endpoints.

3.7 User Interface Design

The user interface was designed to be clean, modern, and intuitive. Vue.js was used to create dynamic components that rendered data in real-time. The app featured role-based dashboards: one for customers to browse menus and place orders, and one for Chefs/Restaurants to manage foods , offerings and track orders.

Menus were categorized by type, cuisine, and dietary labels (vegan, halal, spicy, etc.). Customers could filter foods, and specify sizes and quantities. Chefs had CRUD access to their food items and could see real-time status updates on orders. The UI was responsive and accessible on mobile, tablet, and desktop.

4 Testing and Evaluation

Functional Testing

Testing tools: Pytest (unit tests), Postman (manual API tests)

Pytest is the standard Python testing tool. It is easy to write, extend, and run. Postman is widely used for manual API testing.

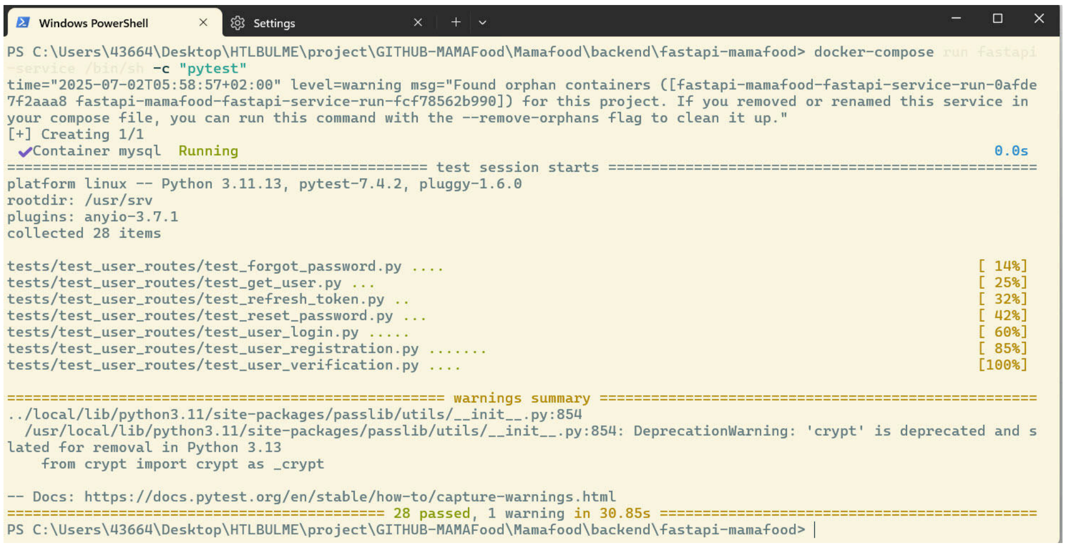
Automated Tests: Pytest covered endpoint testing, model logic, and error handling.The user registration and also email constrain checked with pytests.

Manual Testing: Postman was used to simulate real-world scenarios such as registration, login, token expiration, and CRUD operations.

4.1 Test Scenarios

tests/test_user_routes/test_forgot_password.py
tests/test_user_routes/test_get_user.py
tests/test_user_routes/test_refresh_token.py
tests/test_user_routes/test_reset_password.py
tests/test_user_routes/test_user_login.py
tests/test_user_routes/test_user_registration.py
tests/test_user_routes/test_user_verification.py

Figure 14 shows the outcome of the automated tests related to user login functionality.

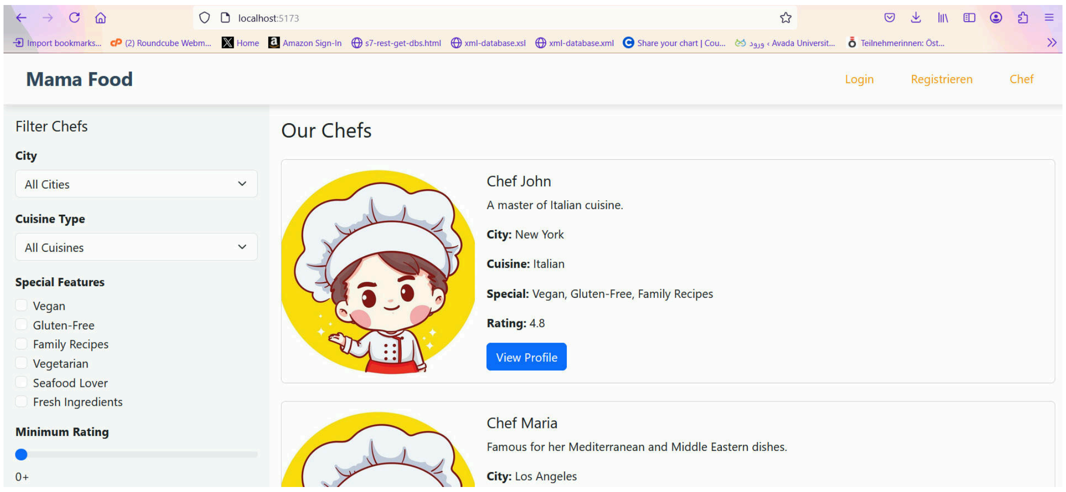


```
PS C:\Users\43664\Desktop\HTLBULME\project\GITHUB-MAMAFood\Mamafood\backend\fastapi-mamafood> docker-compose run fastapi
-service /bin/sh -c "pytest"
time="2025-07-02T05:58:57+02:00" level=warning msg="Found orphan containers ([fastapi-mamafood-fastapi-service-run-0afde
7f2aaa8 fastapi-mamafood-fastapi-service-run-fcf78562b990]) for this project. If you removed or renamed this service in
your compose file, you can run this command with the --remove-orphans flag to clean it up."
[+] Creating 1/1
 ✔Container mysql  Running                                                                                        0.0s
================================================= test session starts =================================================
platform linux -- Python 3.11.13, pytest-7.4.2, pluggy-1.6.0
rootdir: /usr/srv
plugins: anyio-3.7.1
collected 28 items

tests/test_user_routes/test_forgot_password.py ....                                                              [ 14%]
tests/test_user_routes/test_get_user.py ...                                                                      [ 25%]
tests/test_user_routes/test_refresh_token.py ..                                                                  [ 32%]
tests/test_user_routes/test_reset_password.py ...                                                                [ 42%]
tests/test_user_routes/test_user_login.py .....                                                                  [ 60%]
tests/test_user_routes/test_user_registration.py .......                                                         [ 85%]
tests/test_user_routes/test_user_verification.py ....                                                            [100%]

=================================================== warnings summary ==================================================
../local/lib/python3.11/site-packages/passlib/utils/__init__.py:854
  /usr/local/lib/python3.11/site-packages/passlib/utils/__init__.py:854: DeprecationWarning: 'crypt' is deprecated and s
lated for removal in Python 3.13
    from crypt import crypt as _crypt

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
============================================ 28 passed, 1 warning in 30.85s ===========================================
PS C:\Users\43664\Desktop\HTLBULME\project\GITHUB-MAMAFood\Mamafood\backend\fastapi-mamafood> |
```

Figure 14: User login test result.

Non-Functional Testing

Figure 15 shows how the website is user-friendly and easy to use.



Filter Chefs

City

All Cities

Cuisine Type

All Cuisines

Special Features

- Vegan
- Gluten-Free
- Family Recipes
- Vegetarian
- Seafood Lover
- Fresh Ingredients

Minimum Rating

0+

Our Chefs

Chef John

A master of Italian cuisine.

City: New York

Cuisine: Italian

Special: Vegan, Gluten-Free, Family Recipes

Rating: 4.8

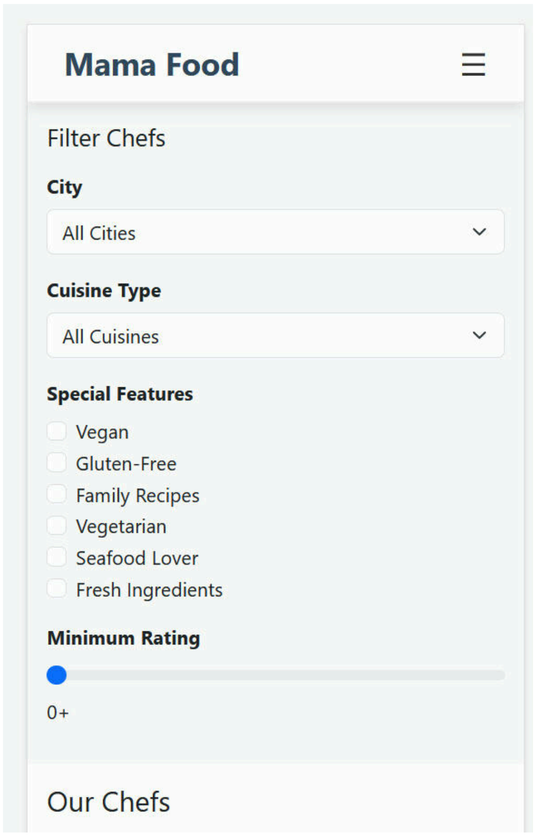
Chef Maria

Famous for her Mediterranean and Middle Eastern dishes.

City: Los Angeles

Figure 15: User login test result.

Figure 16 shows the responsiveness of the website



Filter Chefs

City

All Cities

Cuisine Type

All Cuisines

Special Features

- Vegan
- Gluten-Free
- Family Recipes
- Vegetarian
- Seafood Lover
- Fresh Ingredients

Minimum Rating

0+

Our Chefs

Figure 16: User login test result.

5 Conclusion

Mamafood successfully delivers a scalable, role-based food ordering platform. Its foundation on FastAPI, Vue.js, and MySQL ensures speed, reliability, and maintainability. Authentication is robust through JWT, and SQLAlchemy simplifies data manipulation.

By clearly separating customer and Koch functionality, Mamafood supports real-world food business workflows. Order tracking and role-based control all contribute to the robustness of the system.

In summary, the project demonstrates how modern open-source technologies can power a fully functional online food marketplace. With thoughtful UI design, clean architecture, and a scalable backend, Mamafood sets the foundation for future enhancements such as delivery, live messaging, payment integration, and a mobile app. The current implementation, although limited in a few areas, shows strong potential for real-world deployment with additional investment.

5.1 Future Work

To enhance the user experience and overall functionality of the platform, several additional features are planned for future development. These include:

Payment Integration: Enabling secure and seamless online payments to streamline order processing and reduce manual handling.

Chef Reservation System: Allowing users to reserve specific chefs for private events or scheduled cooking sessions.

Customizable Food Ingredients: Giving users the ability to personalize their meals by selecting or excluding specific ingredients based on dietary preferences or allergies.

Rating and Review System: Introducing a feedback mechanism where customers can rate meals and chefs, helping maintain quality and build trust within the community.

While these features are integral to the long-term vision of the platform, they are being considered for future releases and not included in the initial release due to the following factors:

Significant development effort: Each of these features, particularly the payment integration and the chef reservation system, requires significant development time, with individual estimates exceeding 180 hours of dedicated effort.

Requirement for detailed specifications: Features such as the payment integration require very detailed and robust specifications to ensure security, accuracy, and compliance. Developing these with the required precision requires a focused effort that was not possible for the initial iteration.

Team resource constraints: The initial project planning envisioned a larger development team. However, due to unforeseen circumstances, the team was reduced to a single developer.

6 References:

1. https://en.wikipedia.org/wiki/Structured_analysis Retrieved July 2, 2025
2. https://en.wikibooks.org/wiki/Systems_Analysis_and_Design/Introduction Retrieved July 2, 2025
3. FelixSchwab , Ingrid Schwab-Matkovits , Systemplannung und Projekt-Entwicklung Page 62.
4. https://en.wikipedia.org/wiki/Principles_of_user_interface_design Retrieved July 2, 2025
5. FastAPI. FastAPI – tiangolo.com. Retrieved June 21, 2025, from https://fastapi.tiangolo.com
6. https://json-schema.org/ Retrieved July 2, 2025
7. https://github.com/OAI/OpenAPI-Specification Retrieved July 2, 2025
8. https://fastapi.tiangolo.com/features/ Retrieved July 2, 2025
9. https://docs.pydantic.dev/latest/ Retrieved July 2, 2025
10. https://fastapi.tiangolo.com/Retrieved July 2, 2025
11. https://github.com/swagger-api/swagger-ui Retrieved July 2, 2025
12. https://fastapi.tiangolo.com/features/#dependency-injection Retrieved July 2, 2025
13. https://fastapi.tiangolo.com/async/ Retrieved July 2, 2025
14. https://fastapi.tiangolo.com/tutorial/security/ Retrieved July 2, 2025
15. https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/ Retrieved July 2, 2025
16. https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/#recap Retrieved July 2, 2025
17. https://www.sqlalchemy.org/ Retrieved July 2, 2025
18. https://hibernate.org/orm/what-is-an-orm/ Retrieved July 2, 2025
19. https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases Retrieved July 2, 2025
20. https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases Retrieved July 2, 2025
21. https://www.oracle.com/ca-en/mysql/what-is-mysql/ Retrieved July 2, 2025
22. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries Retrieved July 2, 2025
23. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Introduction Retrieved July 2, 2025
24. https://vueframework.com/guide/introduction.html#what-is-vue-js Retrieved July 2, 2025
25. https://vuejs.org/guide/introduction.html Retrieved July 2, 2025
26. https://v2.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html?redirect=true Retrieved July 2, 2025
27. https://fastapi.tiangolo.com/tutorial/response-status-code/#about-http-status-codes Retrieved July 2, 2025
28. https://atlan.com/non-relational-database-vs-relational/#non-relational-database-vs-relational-database--understanding-the-basics Retrieved July 2, 2025

7 List of Figures and Tables