[HTL BULME]

MamaFood

Marketplace for Homeccoked Foods

Supervisor: Simon Gunakar

Author: Fatemeh Abolhassanlou

IT /Tageskollegue

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.
- Separate the chef and customer registration process: To simplify the registration process

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 sustainable 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

2.1. System Analysis

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. [1] 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 "[1]

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.

2.1.1. Requirements Analysis

Requirements gathering: The first step in the SA/SD process is to gather requirements from stakeholders, including users, customers, and business partners.[1] 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.[2]

Functional requirements define the specific behavior or functions of a system. In contrast, non-functional requirements specify how the system performs its tasks, focusing on attributes like performance, security, scalability, and usability.[2]

Non Functional requirement covers items like: Portability , Security , Maintainability, Reliability, Scalability , Performance , Reusability , Flexibility [2]

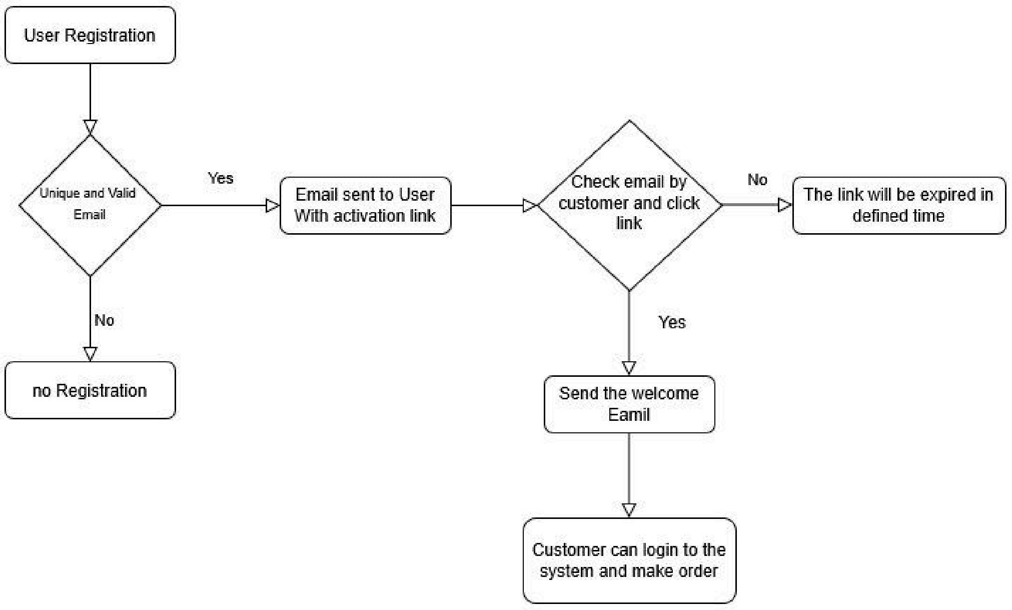
Functional Requirements

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.[2]

1. **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 as a customer and as a chef is as follows. The users as customers are activated automatically, but the chef will be activated after the document review.

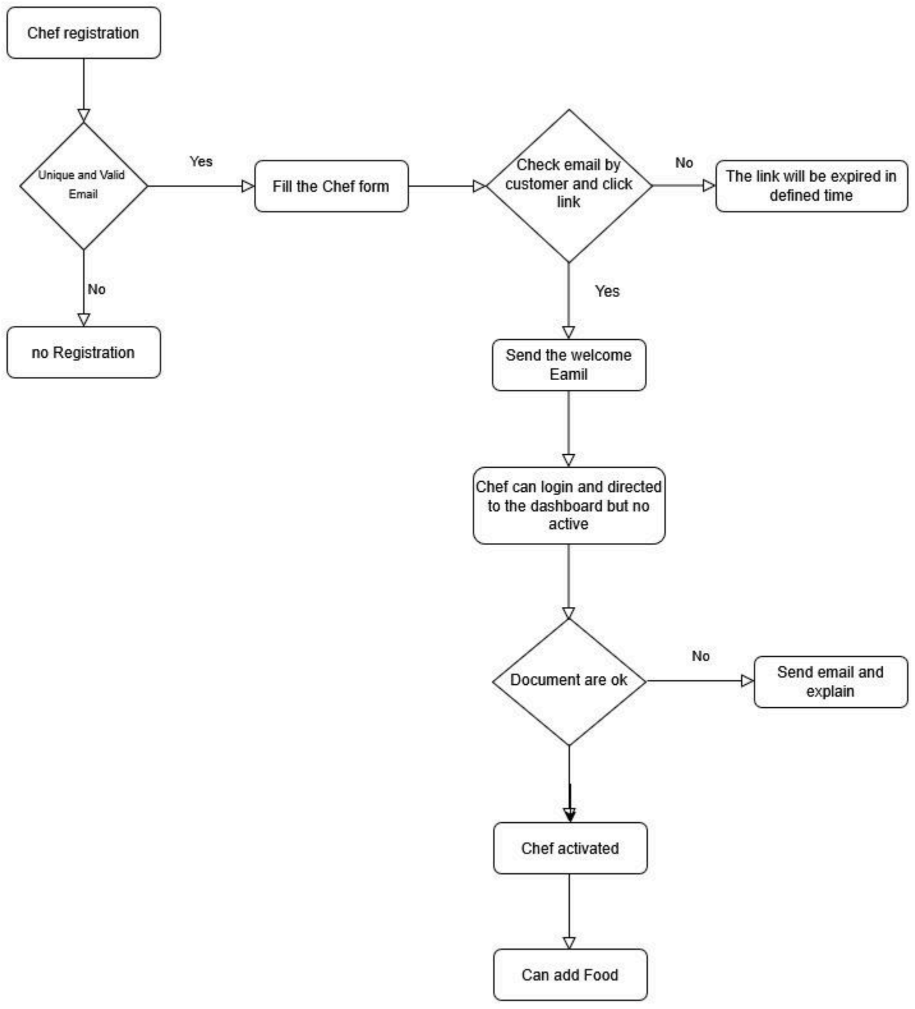
Figure 2.1- User Registration Process


User Registration
Unique and Valid Email
Yes
No
no Registration
Email sent to User With activation link
Check email by customer and click link
No
The link will be expired in defined time
Yes
Send the welcome Eamil
Customer can login to the system and make order

2. **Chef (Koch) - Restaurant Management**
    - Chefs can create and manage their profiles, including:
        - Restaurant/Chef name
        - Description and physical address
        - Associated cuisine type (e.g., Austrian, Persian, etc.)

3. **Chefs - Food Management**
    - Chefs can create, edit, and delete food items.
        - Each food item includes:
        - Food name and its categories (e.g., vegan, main course).
        - Availability status (Quantity Control) and estimated preparation/pick up time.
        - Detailed descriptions like food ingredients and images.

Figure 2.2 - Chef Registration Process



4. **Discounts and Offers**
    - Chefs can configure discounts based on:
        - Quantity of items purchased
        - Total order value

5. **Food Quantity Order Management**

It will be designed for future development:

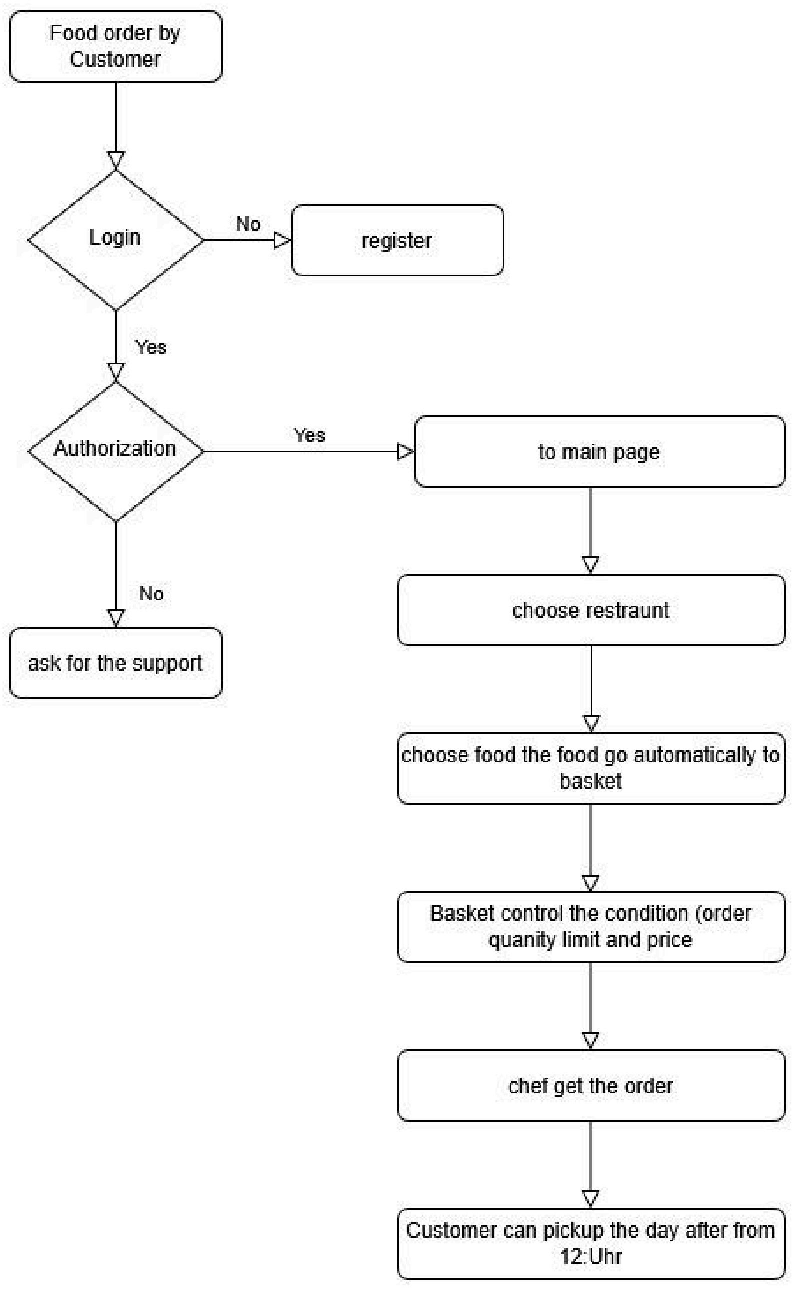
- Chefs can define purchase limits per order—for example, restricting each customer to order only a certain percentage (e.g., 10%) of the available stock—to ensure fair distribution and risk managemnet.

6. **Category and Cuisine Management**

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

7. **Customer - Menu Browsing and Viewing**
    - Customers can browse restaurants by location (e.g., city) and filter food by:
        - Category (e.g., vegan, dessert)
        - Cuisine type

Figure 2.3 - Customer-Order Diagram


Food order by Customer
Login
No
register
Yes
Authorization
Yes
to main page
No
ask for the support
choose restraunt
choose food the food go automatically to basket
Basket control the condition (order quanity limit and price
chef get the order
Customer can pickup the day after from 12:Uhr

8. **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.

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

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.[2]

- The app should load the restaurant menu within an appropriate time.
- The app should be easy to use for first-time users, with an intuitive interface.
- The app should save passwords using hashing algorithms to prevent physical attacks.
- The app should prevent SQL injection vulnerabilities.
- Access levels should be enforced based on user roles (e.g., customer, chef).
- For secure session management, authentication and authorization should be based on OAuth 2.0 and JWT tokens.
- The addition of new restaurants and food items should be possible without requiring changes to the database structure.
- The platform should be responsive and available most of the time.
- 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.

2.1.2. Stakeholders

Stakeholder Overview

There are two types of stakeholders: Internal and external, and it is important to analyze their behavior and power of influence [4]

The successful development and long-term operation of the Mamafood platform rely on addressing the unique needs and expectations of its key stakeholders.

1. **Customers**

    Customers are the end-users who seek convenient, local, home-cooked meal options.
2. **Chefs (Koch / Restaurants)**

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

3. **Administrator/Website Designer**
    - Managing user registrations and verifying Chefs' identities
    - Monitoring content such as food categories, cuisines, and listings

Their roles are crucial to ensuring the platform operates smoothly, securely, and as intended.

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.[4]

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. Below is a summary of the technology and relevant components:

Table 2.1 List of used Technology

Component	Technology	Responsibilities
Frontend	Vue.js, HTML, CSS, Bootstrap	User interface, input handling, responsive design, API calls to backend
Backend API	FastAPI (Python), SQLAlchemy	Business logic, data validation, secure APIs, database interaction
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

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.

The data structure is well-defined, and the relationships between entities are clearly 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 an appropriate choice due to its large community support, mature ecosystem, and availability of robust tools for development and maintenance.

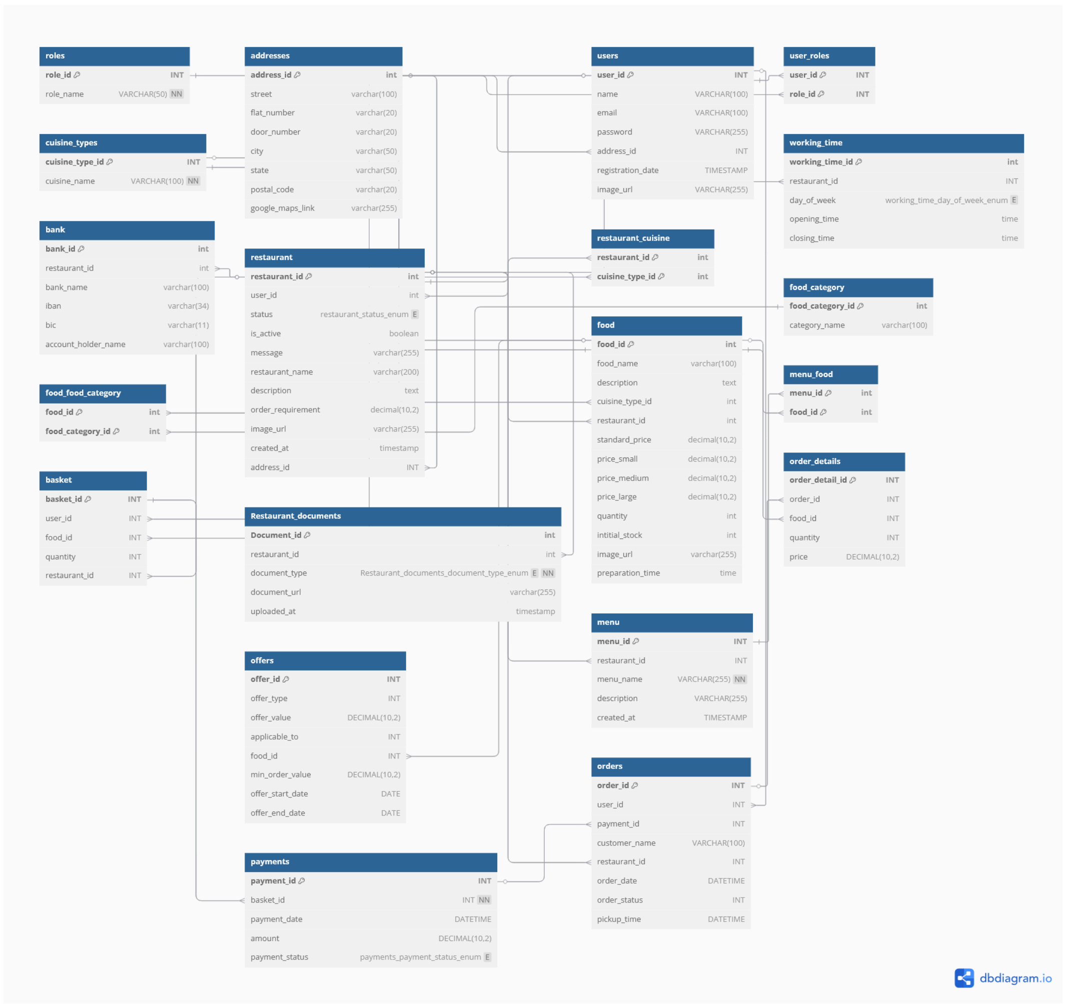
The tables are as below :

1. roles: Stores the different roles of Users
2. addresses: Stores address of chefs/Restaurants(Koch)
3. users: Stores user information
4. user_roles: Manages the many-to-many relationship between users and roles.
5. cuisine_types: Lists available cuisine types (e.g., Italian, Persian, Indian) to categorize food and restaurants
6. restaurant :Stores data about restaurants (chefs or kochs)
7. restaurant_cuisine: Links restaurants to their offered cuisine types
8. working_time: Defines the opening and closing hours for each restaurant for every day of the week.
9. bank: Stores bank details for each restaurant .
10. Restaurant_documents: Stores uploaded legal or operational documents (e.g., Health Certificate) for restaurant verification.
11. food: Stores detailed information about food items such as name, description, prices, stock, and preparation time.
12. food_categoryStores food categories (e.g., Main dish, Dessert, Vegan)
13. food_food_category: Establishes a many-to-many relationship between food items and food categories.
14. offers: Stores promotional offers applicable to food items or orders, including type, value, and duration.
15. menu: Represents a menu that belongs to a restaurant, consisting of various food items grouped under it.
16. menu_food: Links food items to menus (many-to-many relationship).
17. basket: Temporary shopping cart where users add food items before placing an order.
18. payments: Tracks payments made by users for items in their basket, including status and amount.(for future development)
19. orders: Records order details

20. order_details: Contains the detailed list of food items within a specific order, along with quantity and price.

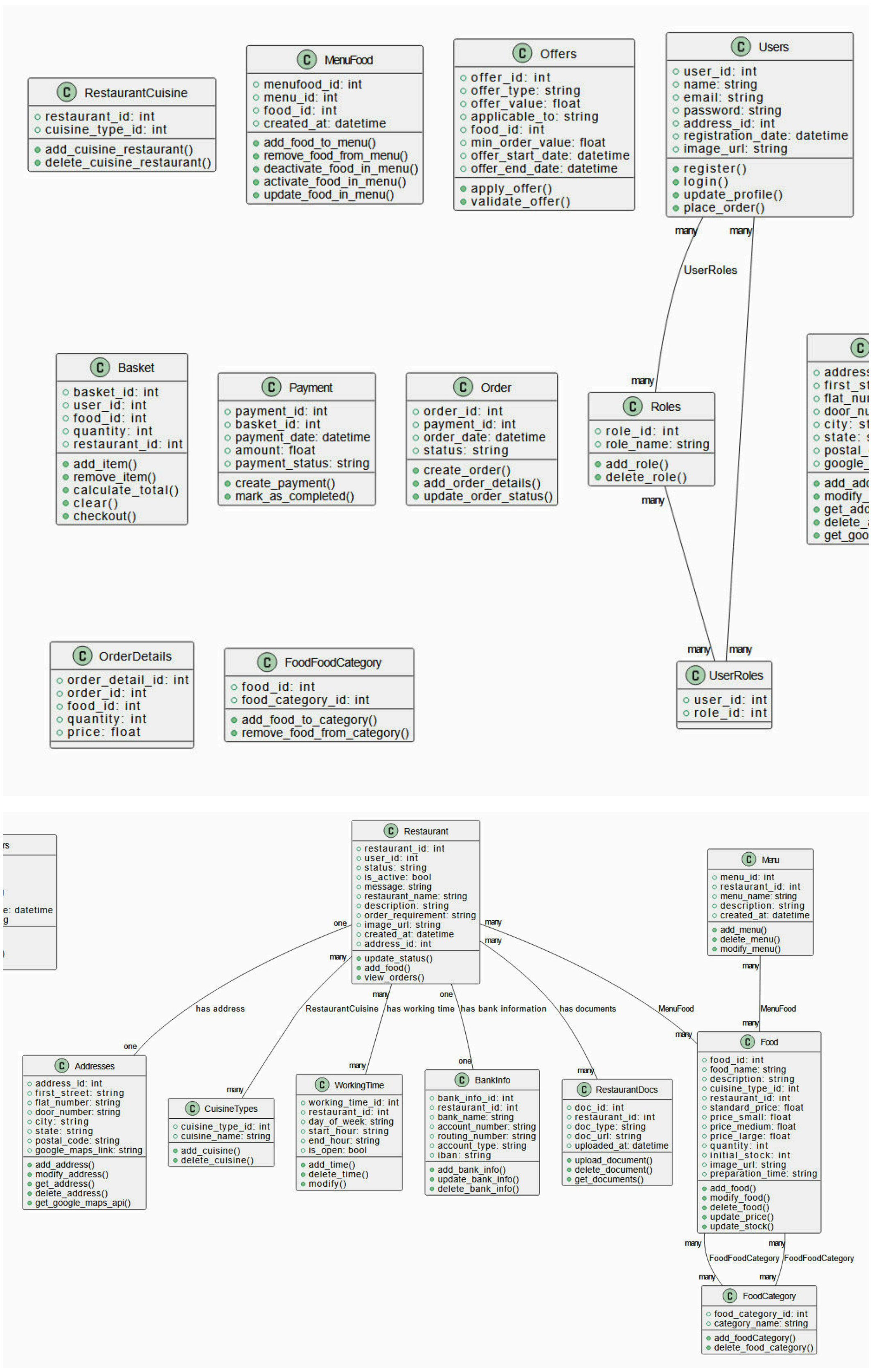
ERD Diagram shows the relationship between different tables.

Figure 3.4. ERD Diagram



A UML Class Diagram primarily represents system entities, their attributes, and relationships. While it can conceptually depict high-level operations like CRUD, its main role here is as a blueprint for data models: informing SQLAlchemy ORM mappings and defining Pydantic models for FastAPI's API request/response validation.

Figure 2.5 - Class Diagram



Model Development with Sqlalchemy (ORM)/API

By using the following command, we can automatically generate SQLAlchemy model classes from a MySQL database. This is a fundamental step in setting up the routing and logic for the API:

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

This 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.

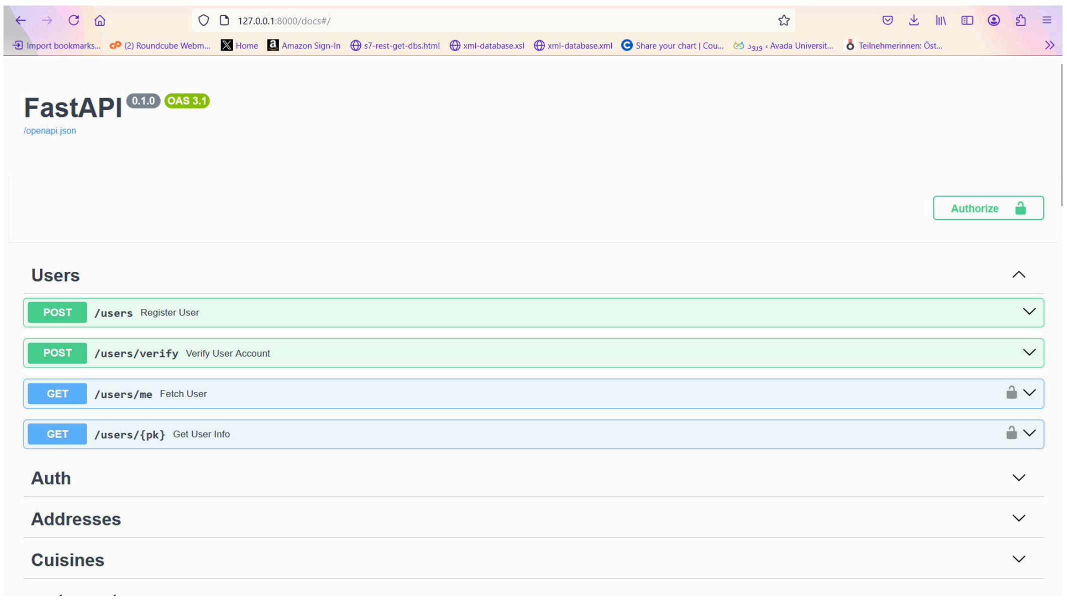
In the next step, it is necessary to create Pydantic schemas. These are used for request and response validation in FastAPI.

Finally, CRUD parts are created based on the class diagram.

Build FastAPI Endpoints "Through main.py, REST API endpoints using FastAPI are created. These endpoints use Pydantic schemas and call functions from the CRUD layer to interact with the database."

It is possible to test the REST API routes using FastAPI's interactive docs or Postman.

Figure 2.6 - REST API with FASTAPI



Users

POST /users Register User

POST /users/verify Verify User Account

GET /users/me Fetch User

GET /users/{pk} Get User Info

Auth

Addresses

Cuisines

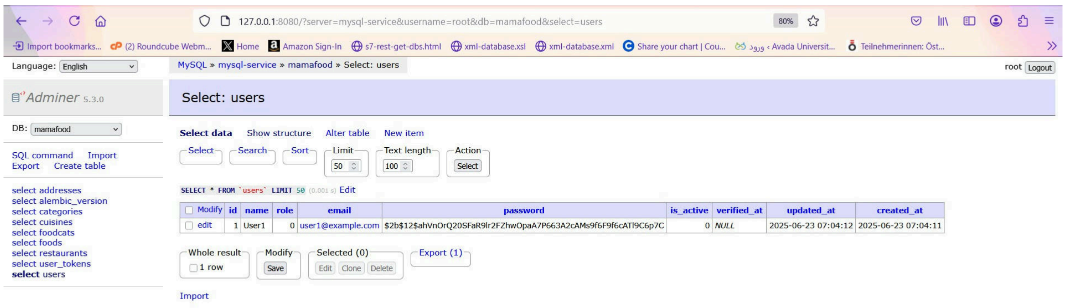
2.2.2. Security Considerations

Security is paramount for protecting user data, and maintaining platform integrity.

Data Protection

- Password Security: All passwords are hashed securely using bcrypt or Argon2 before storing.

Figure 2.7: Password in Hash format in the DB

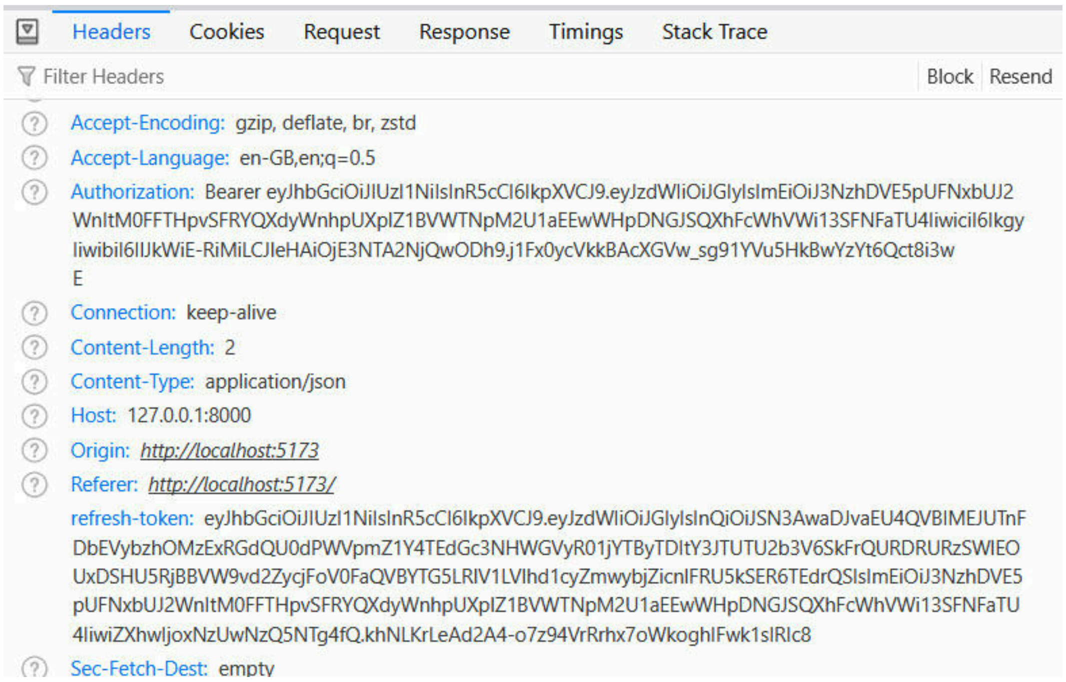


```
SELECT * FROM `users` LIMIT 50 (0.001 s) Edit
```

Modify	id	name	role	email	password	is_active	verified_at	updated_at	created_at
edit	1	User1	0	user1@example.com	$2b$12$ahVnOrQ20SFaR9lr2FZhwOpaA7P663A2cAMs9f6F9f6cATl9C6p7C	0	NULL	2025-06-23 07:04:12	2025-06-23 07:04:11

- **Authentication and Authorization & Session & Token Management**
  - Token-Based Auth: Uses JWT (JSON Web Tokens) for secure access. Tokens include expiration time and refresh logic.
  - Short-lived access tokens with automatic refresh.

Figure 2.8: Token (Authorization and Refresh)



```
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-GB,en;q=0.5
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJGlyIsImEiOiJ3NzhDVE5pUFNxbUJ2
WnltM0FFTHpvSFRYQXdyWnhpUXplZ1BVWTNpM2U1aEEwWHpDNGJSQXhFcWhVWi13SFNFaTU4Iiwicil6Ikgy
liwibil6IlJkWiE-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
DbEVybzhOMzExRGdQU0dPWVpmZ1Y4TEdGc3NHWGVyR01jYTByTDltY3JTUTU2b3V6SkFrQURDRURzSWIEO
UxDSHU5RjBBVW9vd2ZycjFoV0FaQVBYTG5LRlV1LVlhd1cyZmwybjZicnlFRU5kSER6TEdrQSIsImEiOiJ3NzhDVE5
pUFNxbUJ2WnltM0FFTHpvSFRYQXdyWnhpUXplZ1BVWTNpM2U1aEEwWHpDNGJSQXhFcWhVWi13SFNFaTU
4IiwiZXhwIjoxNzUwNzQ5NTg4fQ.khNLKrLeAd2A4-o7z94VrRrhx7oWkoghlFwk1sIRIc8
Sec-Fetch-Dest: empty
```

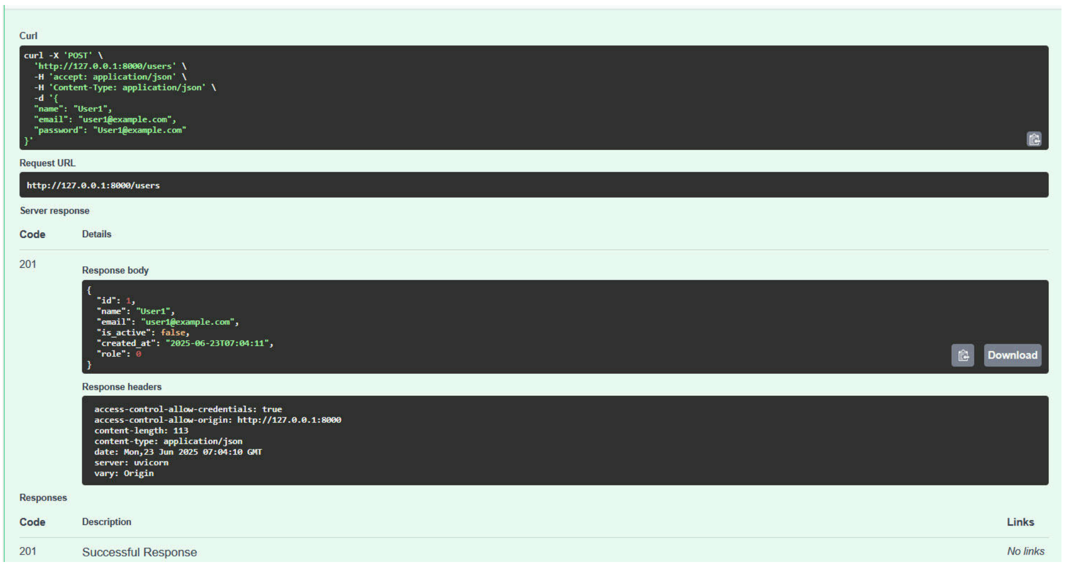
Status Code: 200

- Name: 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

- Name: 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)

Figure 2.9 : Status code 201



Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/users' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "User1",
  "email": "user1@example.com",
  "password": "User1@example.com"
}'
```

Request URL

```
http://127.0.0.1:8000/users
```

Server response

Code Details

201

Response body

```
{
  "id": 1,
  "name": "User1",
  "email": "user1@example.com",
  "is_active": false,
  "created_at": "2025-06-23T07:04:11",
  "role": 0
}
```

Response headers

```
 access-control-allow-credentials: true
 access-control-allow-origin: http://127.0.0.1:8000
 content-length: 113
 content-type: application/json
 date: Mon,23 Jun 2025 07:04:10 GMT
 server: uvicorn
 vary: Origin
```

Responses

Code	Description	Links
201	Successful Response	*No links*

Status Code: 404

- Name: 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

2.2.3. Data Flow

Here are simplified data flow diagrams for key processes:

1. 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.

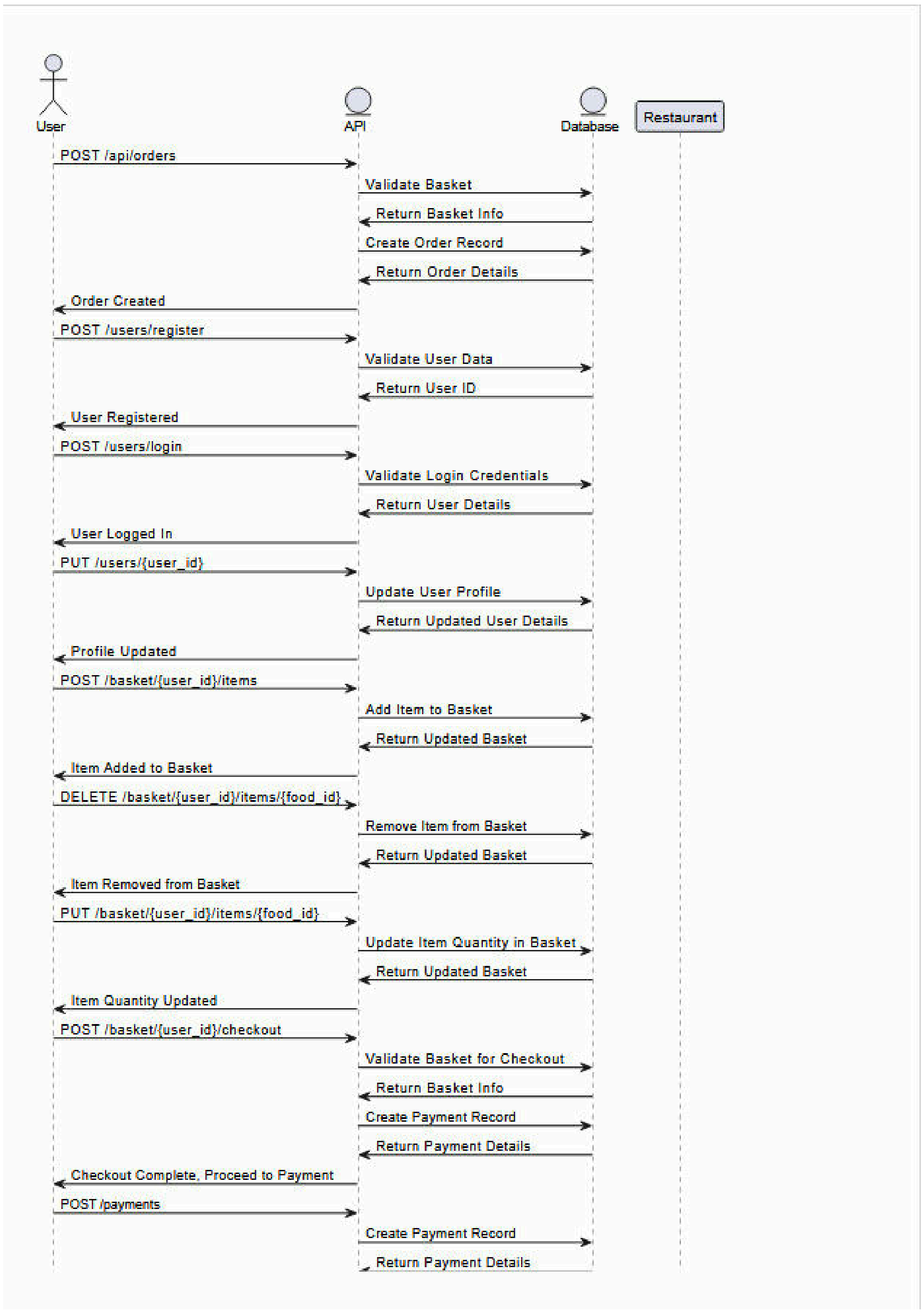
2. Order Placement (Customer):

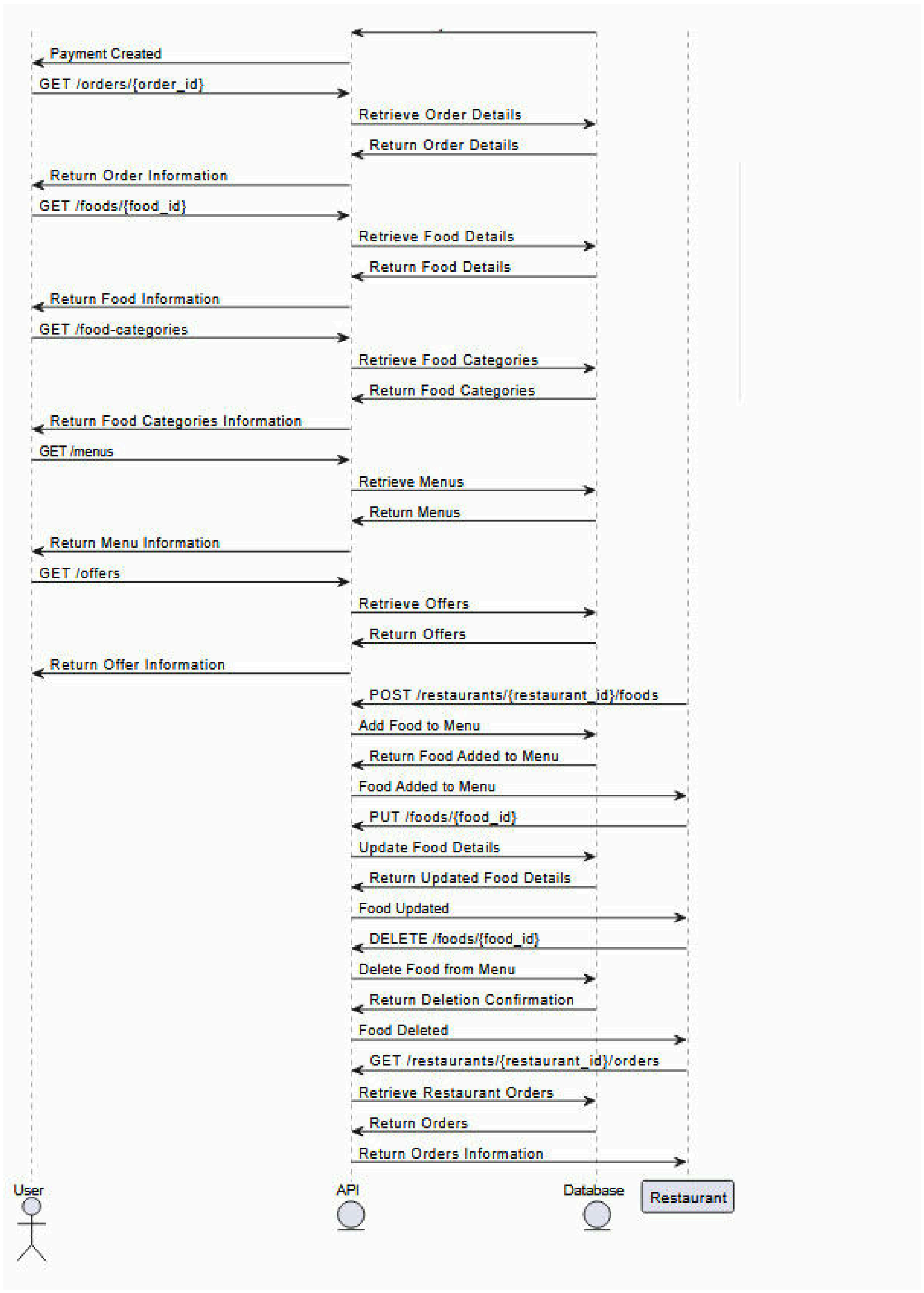
- Customer browses restaurants and food_items via Frontend.
- Customer adds food_items to basket (local state initially, then synchronized with baskets table).
- When ready, Customer initiates checkout on Frontend.
- Frontend sends basket contents to Backend checkout API.
- Backend validates basket (stock, 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.

3. 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.

Figure 2-10 - Data flow





2.2.4. UX Considerations

User Interface (UI) design plays an important role in the success of a website or a digital product for the following reasons[5]:

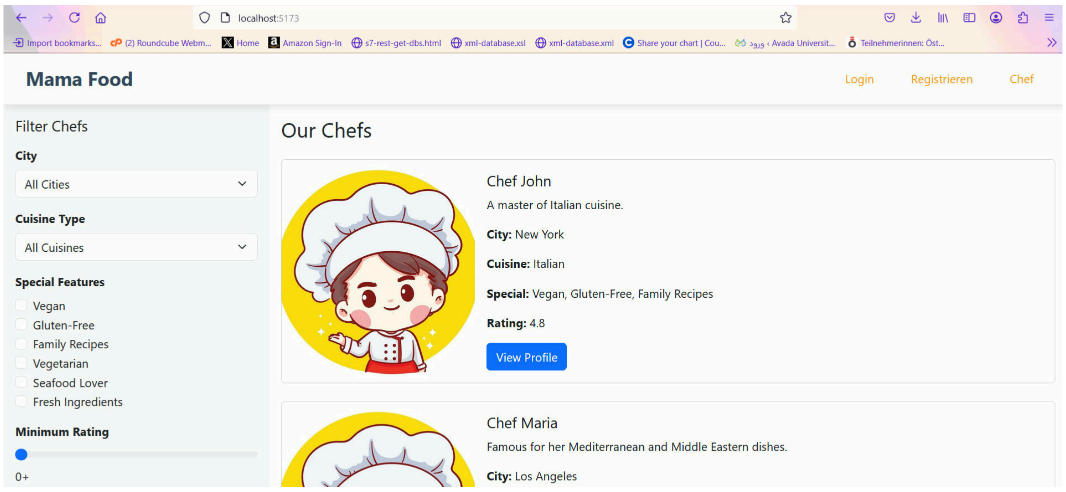
- User Experience
- Engagement
- Branding
- Efficiency

General UI Requirements

- Fully responsive layout for both mobile and desktop devices

- Easy to User
- Having Functional UX features (search and filter , dashboard for chef/ restaurants and customer)

Figure 2-11: Mama Food Website-Firspage



Chef John

A master of Italian cuisine.

City: New York

Cuisine: Italian

Special: Vegan, Gluten-Free, Family Recipes

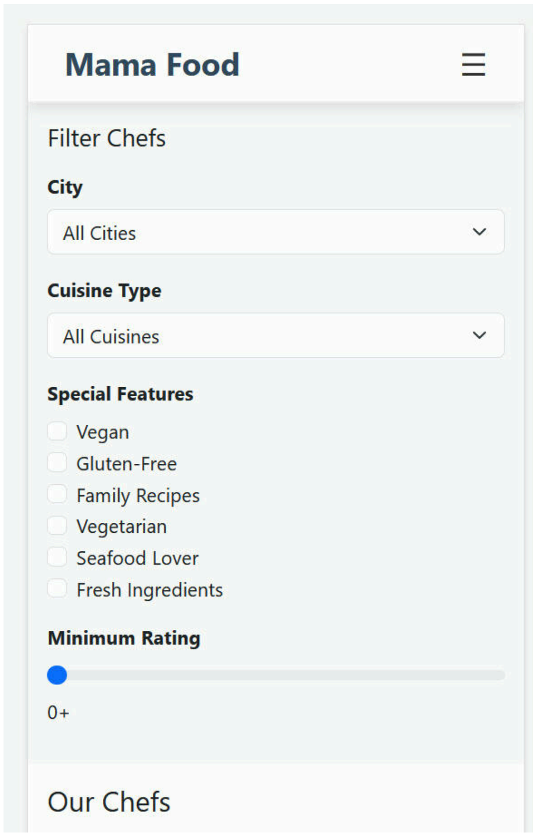
Rating: 4.8

Chef Maria

Famous for her Mediterranean and Middle Eastern dishes.

City: Los Angeles

Figure 2-12: Moblie Responsive view



2.3. References:

1. https://www.geeksforgeeks.org/structured-analysis-and-structured-design-sa-sd/

2.https://www.geeksforgeeks.org/system-design/system-analysis-system-design/ #system-analysis

3. FelixSchwab , Ingrid Schwab-Matkovits , Systemplannung und Projekt-Entwicklung Page 62.
4. https://www.geeksforgeeks.org/system-design/system-analysis-system-design/#system-analysis

5.https://www.geeksforgeeks.org/techtips/principles-of-ui-ux-design/

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 three main layers: front-end, back-end, and database. Each component has been selected based on its performance, maintainability, and support for rapid development from a developer ecosystem.

3.1. Technologies Used

Backend

"FastAPI is a modern, fast (high-performance) web framework for building APIs with Python, leveraging standard Python type hints [1]. 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 [2] and JSON schema[3]).[4]."

Link to download and further information : https://pypi.org/project/fastapi/

FastAPI Features: [5] [6]

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

Servers

- Uvicorn:

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 your 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 implementation 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.[7]

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. This means the code will have fewer mistakes and pretty much explains itself.

Pydantic for the data validation

Pydantic is the most widely used data validation library for Python.[8] "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." [8]

Dependency Injection

FastAPI supports dependency injection, a system that's extremely easy to use for declaring dependencies for endpoints. This really helps keep code modular, testable, and maintainable. It's an extremely powerful system.

The framework handles everything 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. It makes no compromise with databases or frontends, but instead offers easy integration with all of them.[9]

Asynchronous Support

Modern versions of Python have support for “asynchronous code” using something called “coroutines”, with async (async code :a way to tell the computer / program that at some point in the code, it will have to wait for something else to finish somewhere else) and await syntax.[10] 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 and tools

There are many ways to handle security, authentication and authorization.FastAPI provides several tools to deal with Security easily, rapidly, in a standard way, without having to study and learn all the security specifications. 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 There are OAuth2, OAuth 1,OpenID Connect,OpenID (not “OpenID Connect”),OpenAPI (Previously Swagger) methods. FastAPI is based on OpenAPI [11]

OpenAPI

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:

- apiKey (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 with Password (and hashing), Bearer with JWT tokens 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 [12].

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) [12]. 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. [12]

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.[13]

There are tools for:

- API Key Security Schemas
- HTTP Authentication Schemes
- OAuth2 Security Schemes (and related forms)
- OpenID Connect
- Utility for Scopes

```
from fastapi.security import (
    APIKeyCookie,
    APIKeyHeader,
    APIKeyQuery,
    HTTPAuthorizationCredentials,
    HTTPBasic,
    HTTPBasicCredentials,
    HTTPBearer,
    HTTPDigest,
    OAuth2,
    OAuth2AuthorizationCodeBearer,
    OAuth2PasswordBearer,
    OAuth2PasswordRequestForm,
    OAuth2PasswordRequestFormStrict,
    OpenIdConnect,
    SecurityScopes)
```

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.Hibernate ORM helps us keep persistent state in a relational database.[14]

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. [15]

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. [16]

Tools (ORM Mapped Class Configuration) :

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

Installtion: pip install SQLAlchemy

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.[17] **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 3-1.Comparison between Relational and Non-Relational Database[18]

Feature	Non-Relational	Relational
Availability	High	High
Horizontal Scaling	High	Low
Vertical Scaling	High	High
Data Storage	Optimized for huge data volumes	Suitable for medium to large data
Performance	High	Low to Medium
Reliability	Medium	High (ACID-compliant)
Complexity	Low	Medium (Joins)
Flexibility	High	Low (Strict Schema)
Suitability	OLAP and OLTP	Primarily OLTP

source: [17]

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, you'll find it at the heart of demanding, high-traffic applications such as Facebook, Netflix, Uber, Airbnb, Shopify, and Booking.com.[19]

The database chosen was **MySQL**. Unlike SQLite, which is lightweight and local, 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.[19]

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.

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. [19]

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. [20]

Why frameworks are out there? The real problem is this: every time we change our application's state, we need to update the UI to match.[19]

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.[19]

Thanks to Vue, no need to write our own functions for building the UI; the framework will handle that for us in an optimized, efficient way. Our only role here is to describe to Vue what each item should look like.

Table 3-2 Comparison of Frameworks[20]

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

To choose the appropriate frame work , some items should be considered:[21]

- 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?

Vue (pronounced /vjuː/, 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).[22] Two core feature of the vue: [23]

- 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.

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. You configure routes to tell Vue Router which components to show for each URL path.

Vue-Axios There are many times when building application for the web that you 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. [24]

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.

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 the ones you would use 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.

For generic errors from the client, There is code 400.

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. [25]

Technologies and Alternatives-Summary

1. Backend: Python (FastAPI), PyJWT, passlib

- **Why it was used**: FastAPI is known for its modern async support, fast execution, and automatic documentation generation. It is increasingly adopted by developers, with a strong GitHub community and official support. PyJWT and passlib are widely adopted libraries with good documentation, ensuring secure authentication and password management.

- **Advantages**: FastAPI offers asynchronous support, automatic validation, and high performance. PyJWT and passlib ensure secure authentication and password handling.
- **Disadvantages**: FastAPI is relatively new, and some libraries or community support may be limited compared to older frameworks like Django.

System Compatibility Table

This table illustrates general compatibility. “Works well” implies common and well-supported integrations. “Possible with effort” means it’s technically feasible but might require more custom work or bridging solutions. “Rarely/Not directly” indicates either a very uncommon pairing or direct incompatibility for common use cases (e.g., PHP and Python typically interact via HTTP APIs, not directly sharing code).

Table 3-3. System Compatibility Table

Component	**Backend**	**Frontend**	**Database**	**ORM**	**Auth.**
Mamafood	Python (FastAPI)	Vue.js	MySQL	SQLAlchemy	JWT
Alternatives	Django, Flask, Node.js (Express, NestJS), Go (Gin), Java (Spring Boot), PHP	React, Angular, Svelte, jQuery	PostgreSQL, SQLite, MongoDB, Oracle, Redis, Neo4j	Django ORM, Sequelize, Prisma, Hibernate, Eloquent	OAuth2, OpenID, Session, API Key, SAML, MFA
Backend comp.	Integrates well with Python ORMs like SQLAlchemy	Communicates with any backend via REST or GraphQL	Compatible with all backends via drivers	Works with FastAPI, Flask, Django, etc.	Implementable in any backend
Frontend comp.	Compatible with all modern frontends (Vue, React, Angular) via REST/	—	—	—	Frontend stores tokens or manages cookies

	GraphQL APIs				
Database comp.	Connects to any SQL/NoSQL DB using appropriate drivers	—	—	Connects to supported databases	Stores user/password, tokens in DB
ORM comp.	Uses language-specific ORMs like SQLAlchemy, Sequelize, etc.	—	Works with ORMs like SQLAlchemy, Hibernate, etc.	Language/framework-specific	ORM models include user credentials and session-related data
Auth. comp.	Supports JWT, OAuth2, session, API keys, etc.	Handles tokens (JWT in storage) or cookies for sessions	Stores user credentials (hashed), sessions, and roles	Manages user models, hashed passwords, roles, etc.	—

5. **Dev Tools: VS Code, GitHub, Docker, Postman**

- **Why it was used**: These tools are industry-standard. VS Code has strong plugin support and an active user community. GitHub facilitates collaboration and version control. Docker provides reliable containerized environments, and Postman is essential for API development and debugging with a large base of shared test collections.
- **Advantages**: Enhance productivity, code versioning, containerization, and API testing.
- **Disadvantages**: Requires familiarity with multiple tools and proper environment setup.

3.2. 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 fully responsive and accessible on mobile, tablet, and desktop.

3.3. Key Features

JWT Authentication: Stateless, secure login sessions
Role-Based Access: Custom dashboard for customers and Kochs
Menu Management: Kochs can add foods with sizes, categories, and tags
Order System: Customers place orders; Kochs prepare and mark them as picked up
Offer Mechanism: Fixed, percentage, and combo-based discounts
Real-Time Tracking: Orders update status dynamically (via polling)
Daily Reports: Kochs can view sales reports and payment statuses
Status Codes Handling: Returns 200, 201, 401, 403, or 404 for feedback and debugging

\

3.4. References

1. FastAPI. FastAPI – tiangolo.com. Retrieved June 21, 2025, from https://fastapi.tiangolo.com
2. https://json-schema.org/
3. https://github.com/OAI/OpenAPI-Specification
4. https://www.geeksforgeeks.org/python/fastapi-introduction/
5. https://fastapi.tiangolo.com/features/
6. https://docs.pydantic.dev/latest/
7. https://www.geeksforgeeks.org/python/fastapi-introduction/
8. https://github.com/swagger-api/swagger-ui
9. https://fastapi.tiangolo.com/features/#dependency-injection
10. https://fastapi.tiangolo.com/async/
11. https://fastapi.tiangolo.com/tutorial/security/
12. https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/
13. https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/#recap
14. https://www.sqlalchemy.org/
15. https://hibernate.org/orm/what-is-an-orm/
16. https://www.geeksforgeeks.org/dbms/what-is-object-relational-mapping-orm-in-dbms/
17. https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases
18. https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases
19. https://www.oracle.com/ca-en/mysql/what-is-mysql/
20. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries
21. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Introduction
22. https://vueframework.com/guide/introduction.html#what-is-vue-js
23. https://vuejs.org/guide/introduction.html
24. https://v2.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html?redirect=true
25. https://fastapi.tiangolo.com/tutorial/response-status-code/#about-http-status-codes

4. Testing and Evaluation

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

Why it was used: 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

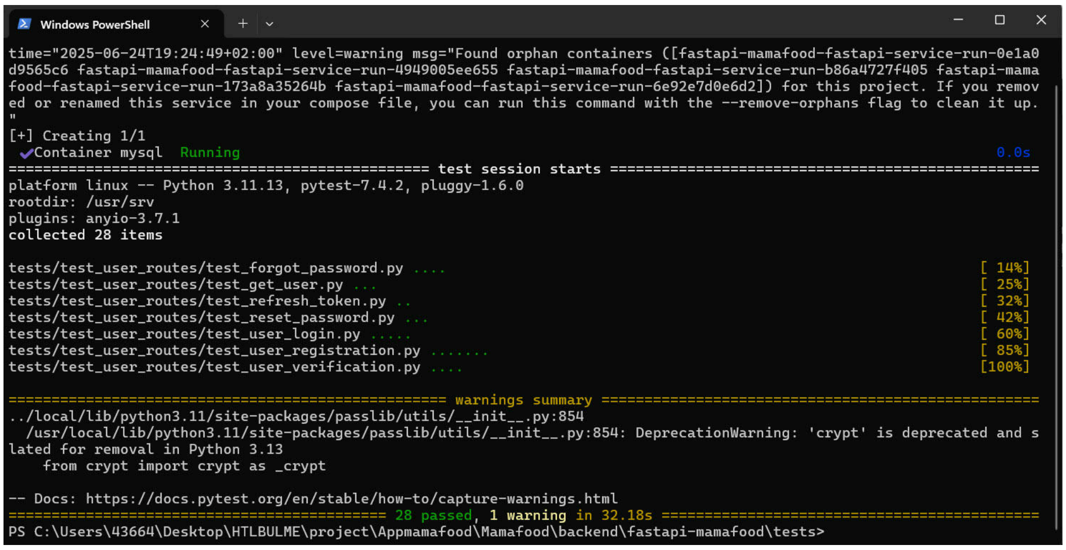
tests/test_user_routes/test_reset_password.py

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 4-1. Test Result



```
time="2025-06-24T19:24:49+02:00" level=warning msg="Found orphan containers ([fastapi-mamafood-fastapi-service-run-0e1a0
d9565c6 fastapi-mamafood-fastapi-service-run-4949005ee655 fastapi-mamafood-fastapi-service-run-b86a4727f405 fastapi-mama
food-fastapi-service-run-173a8a35264b fastapi-mamafood-fastapi-service-run-6e92e7d0e6d2]) for this project. If you remov
ed 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 32.18s ============================================
PS C:\Users\43664\Desktop\HTLBULME\project\Appmamafood\Mamafood\backend\fastapi-mamafood\tests>
```

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.

.

6. List of Figures and Tables