### ****API Report for Food Hub****

#### ****Overview:****

The Food Hub API is a Python-based web API developed using the Flask framework. The API provides a backend for a food delivery platform, handling essential operations such as managing users, food items, categories, and orders. The API connects to a MySQL database and offers a range of RESTful endpoints to interact with the system, allowing users to add, retrieve, update, and delete records for users, food items, food categories, orders, payments, deliveries, reviews, and workers.

#### ****Technologies Used:****

* **Flask**: A lightweight Python web framework used for building the API.
* **SQLAlchemy**: An ORM (Object Relational Mapper) to interact with the MySQL database.
* **MySQL**: The relational database used to store all platform data.
* **Flask-SQLAlchemy**: An extension for Flask that adds support for SQLAlchemy.
* **JSON**: Used for structuring data in API responses and requests.

#### ****Database Structure:****

The following tables are used to represent the core data structure of the Food Hub:

1. **Users**
2. **Food Categories**
3. **Food Items**
4. **Orders**
5. **Order Items**
6. **Payments**
7. **Reviews**
8. **Addresses**
9. **Workers**
10. **Deliveries**

Detailed Description of API Endpoints:

#### ****1. Users****

**GET** /users

* + Retrieves a list of all users in the system.
  + **Response**: A JSON array of all user objects.

**POST** /users

* + Adds a new user to the system.

**Request Body**:fig1

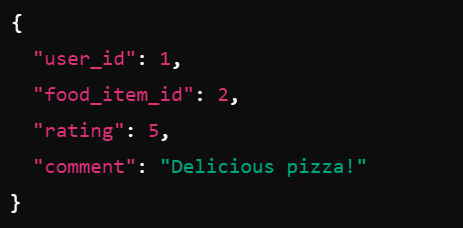


Fig1

#### ****2. Food Categories****

**GET** /categories

* + Retrieves a list of all food categories.
  + **Response**: A JSON array of all categories.

**POST** /categories

* + Adds a new food category to the system.
  + **Request Body**:(fig2)



Fig2

#### ****3. Food Items****

**GET** /food\_items

* + Retrieves a list of all food items.
  + **Response**: A JSON array of all food items.

**POST** /food\_items

* + Adds a new food item to the system.

**Request Body(fig3)**



Fig3

#### ****4. Orders****

**GET** /orders

* + Retrieves a list of all orders in the system.
  + **Response**: A JSON array of all order objects.

**POST** /orders

* + Creates a new order.

**Request Body(fig4)**

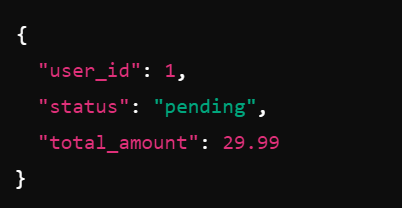


Fig4

#### ****5. Order Items****

**GET** /order\_items

* + Retrieves a list of all order items.
  + **Response**: A JSON array of all order items.

**POST** /order\_items

* + Adds items to an order.

**Request Body(fig5)**

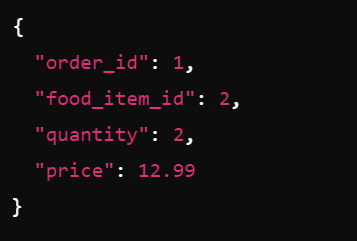


Fig5

#### ****6. Payments****

**GET** /payments

* + Retrieves a list of all payments.
  + **Response**: A JSON array of all payments.

**POST** /payments

* + Processes a payment for an order.

**Request Body(fig6)**

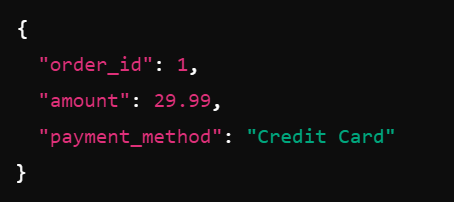


Fig6

#### ****7. Deliveries****

**GET** /deliveries

* + Retrieves a list of all deliveries.
  + **Response**: A JSON array of all delivery objects.

**POST** /deliveries

* + Creates a new delivery record.

**Request Body (fig7)**

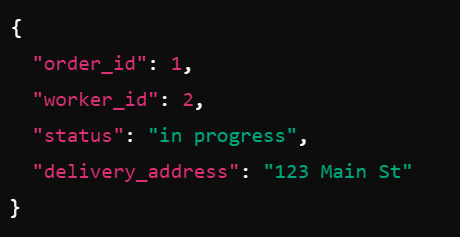


Fig7

#### ****8. Workers****

**GET** /workers

* + Retrieves a list of all workers.
  + **Response**: A JSON array of all worker objects.

**POST** /workers

* + Adds a new worker to the system.

**Request Body(fig8)**



Fig8

#### ****9. Reviews****

**GET** /reviews

* + Retrieves a list of all reviews.
  + **Response**: A JSON array of all reviews.

**POST** /reviews

* + Allows a user to leave a review for a food item.

**Request Body (fig9)**

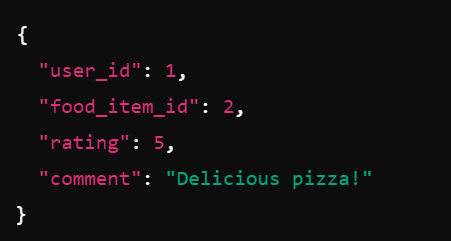


Fig9

#### ****10. Addresses****

**GET** /addresses

* + Retrieves a list of all addresses.
  + **Response**: A JSON array of all addresses.

**POST** /addresses

* + Adds a new address for a user.

**Request Body**:(fig10)

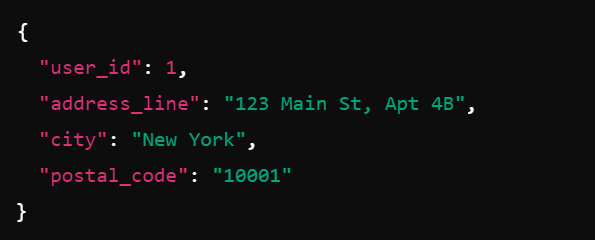


Fig10

### ****API Authentication and Security:****

The API does not currently implement authentication and authorization. However, it is strongly recommended to implement token-based authentication (e.g., JWT) to secure sensitive endpoints such as adding or modifying data.

### ****Error Handling:****

The API includes basic error handling, where invalid or missing input results in appropriate HTTP status codes (e.g., 400 Bad Request, 404 Not Found) along with error messages.

### ****Database Schema:****

Each model is mapped to a table in the MySQL database using SQLAlchemy. The relationships between models are clearly defined using foreign keys.

#### ****User Model:****

* Represents the user who places orders and reviews food.
* Fields: id, username, password, email, role.

#### ****Food Category Model:****

* Represents the categories to which food items belong.
* Fields: id, name.

#### ****Food Item Model:****

* Represents individual food items available for purchase.
* Fields: id, name, description, price, image\_url, category\_id (ForeignKey).

#### ****Order Model:****

* Represents a user's order.
* Fields: id, user\_id (ForeignKey), status, total\_amount.

#### ****Order Item Model:****

* Represents individual items in an order.
* Fields: id, order\_id (ForeignKey), food\_item\_id (ForeignKey), quantity, price.

#### ****Payment Model:****

* Represents payments for orders.
* Fields: id, order\_id (ForeignKey), amount, payment\_method.

#### ****Review Model:****

* Represents customer reviews for food items.
* Fields: id, user\_id (ForeignKey), food\_item\_id (ForeignKey), rating, comment.

#### ****Address Model:****

* Represents user addresses.
* Fields: id, user\_id (ForeignKey), address\_line, city, postal\_code.

#### ****Worker Model:****

* Represents delivery workers.
* Fields: id, name, vehicle\_type, status.

#### ****Delivery Model:****

* Represents deliveries of orders.
* Fields: id, order\_id (ForeignKey), worker\_id (ForeignKey), status, delivery\_address.

### ****Conclusion:****

This API provides all the necessary functionality to manage a food delivery service, handling users, orders, food items, payments, and more. By interacting with these endpoints, frontend applications or clients can manage the data and provide a complete food delivery experience.