Restaurant Management API Documentation

Muhammed Furkan Ataç - 150210304

Ömer Erdağ - 150210332 Serdar Biçici - 150210331
Batuhan Sal - 150210316

January 6, 2025

Contents

1 Project Overview 4

2 Data Model and Documentation 4
2.1 Data Model 4
2.1.1 Tables 4
2.1.2 Customer 5
2.1.3 Reservation 5
2.1.4 Menu 5
2.1.5 Order 5
2.1.6 Order_Item 6
2.1.7 Staff 6
2.1.8 Payment 6

3 ER Diagram 6

4 Relationships Between Tables 7
4.1 Customers and Reservations 7
4.2 Tables and Reservations 7
4.3 Reservations and Orders 8
4.4 Orders and Order_Items 8
4.5 Menu and Order_Items 8
4.6 Orders and Payments 8

5 Complex Queries **9**
5.1 Customer Spending Query . . . 9
5.1.1 Query . . . 9
5.1.2 Explanation . . . 9
5.1.3 Sample Output . . . 9
5.2 Popular Dishes Query . . . 10
5.2.1 Query . . . 10
5.2.2 Explanation . . . 10
5.2.3 Sample Output . . . 10
5.3 Pending Orders Query . . . 11
5.3.1 Query . . . 11
5.3.2 Explanation . . . 11
5.3.3 Sample Output . . . 11
5.4 Customers Who Spent Above Average . . . 11
5.4.1 Query . . . 11
5.4.2 Explanation . . . 12
5.4.3 Sample Output . . . 12

6 CRUD Operations and Implementations **13**
6.1 Analytics Endpoints . . . 13
6.1.1 Customer Spending Analytics . . . 13
6.1.2 Popular Dishes Analytics . . . 13
6.1.3 Pending Orders Details . . . 13
6.2 Customers Endpoints . . . 13
6.2.1 Add a New Customer . . . 13
6.2.2 Get All Customers . . . 13
6.2.3 Get a Specific Customer . . . 13
6.2.4 Update a Customer . . . 13
6.2.5 Delete a Customer . . . 13
6.3 Menu Endpoints . . . 13
6.3.1 Add a New Dish . . . 13
6.3.2 Get All Menu Items . . . 14
6.3.3 Get a Specific Dish . . . 14
6.3.4 Update a Dish . . . 14
6.3.5 Delete a Dish . . . 14
6.4 Order Items Endpoints . . . 14
6.4.1 Add a New Order Item . . . 14
6.4.2 Get All Order Items . . . 14
6.4.3 Get a Specific Order Item . . . 14
6.4.4 Update an Order Item . . . 14
6.4.5 Delete an Order Item . . . 14

6.5 Orders Endpoints 14
6.5.1 Add a New Order 14
6.5.2 Get All Orders 15
6.5.3 Get a Specific Order 15
6.5.4 Update an Order 15
6.5.5 Delete an Order 15
6.6 Payments Endpoints 15
6.6.1 Add a Payment 15
6.6.2 Update a Payment 15
6.6.3 Delete a Payment 15
6.7 Staff Endpoints 15
6.7.1 Add a Staff Member 15
6.7.2 Get All Staff Members 15
6.7.3 Get a Specific Staff Member 15
6.7.4 Update a Staff Member 16
6.7.5 Delete a Staff Member 16
6.8 Tables Endpoints 16
6.8.1 Add a New Table 16
6.8.2 Get All Tables 16
6.8.3 Get a Specific Table 16
6.8.4 Update a Table 16
6.8.5 Delete a Table 16

7 Challenges and Solutions 16

1 Project Overview

The Restaurant Management API is a web-based system designed to streamline restaurant operations. It enables efficient management of customers, reservations, menu items, orders, payments, and staff information. The system is built using Python's Flask framework and MySQL database, offering secure and scalable solutions for handling restaurant workflows.

Key features of the API include:

- Managing customer details and contact information.
- Handling table reservations with time constraints to prevent double bookings.
- Maintaining a dynamic menu with categories and pricing.
- Tracking orders and associated items for reservations.
- Recording payment transactions and supporting multiple payment methods.
- Managing staff roles and work shifts.

The API supports CRUD operations and integrates JWT-based authentication for secure access. It also includes analytical features and complex queries to generate reports, such as revenue tracking and pending orders. Swagger documentation is provided to ensure ease of testing and integration.

2 Data Model and Documentation

2.1 Data Model

2.1.1 Tables

Stores information about the restaurant tables, including their capacity and location.

- **Primary Key:** table_id (NOT NULL)
- **Attributes:** capacity (INT), location (VARCHAR(50))

2.1.2 Customer

Stores customer information, such as names and contact details.

- **Primary Key:** customer_id (NOT NULL)
- **Attributes:** name (VARCHAR(100)), contact_details (VARCHAR(100), UNIQUE)

2.1.3 Reservation

Tracks reservations, linking customers to tables with details like date, time, and status.

- **Primary Key:** reservation_id (NOT NULL)
- **Attributes:**
  - **Foreign Keys:** customer_id (FK to Customers, NOT NULL), table_id (FK to Tables, NOT NULL)
  - reservation_date (DATE)
  - reservation_time (TIME)
  - status (ENUM)
- **Constraint:** unique_table_time ensures no table is double-booked for the same time slot.

2.1.4 Menu

Maintains a catalog of dishes available in the restaurant.

- **Primary Key:** dish_id (NOT NULL)
- **Attributes:** dish_name (VARCHAR(100), NOT NULL), category (ENUM), price (DECIMAL(10,2))

2.1.5 Order

Links reservations to order details, including total amount and order status.

- **Primary Key:** order_id (NOT NULL)
- **Attributes:**
  - **Foreign Keys:** reservation_id (FK to Reservations, NOT NULL)
  - total_amount (DECIMAL(10,2))
  - order_status (ENUM)

2.1.6 Order_Item

Tracks individual items within an order.

- **Primary Key:** order_item_id (NOT NULL)
- **Attributes:**
  - **Foreign Keys:** order_id (FK to Orders, NOT NULL), dish_id (FK to Menu, NOT NULL)
  - quantity (INT)
- **Constraint:** unique_order_menu ensures no duplicate dish within a single order.

2.1.7 Staff

Stores staff details, including their roles and shifts.

- **Primary Key:** staff_id (NOT NULL)
- **Attributes:** name (VARCHAR(100), NOT NULL), role (ENUM), shift (ENUM)

2.1.8 Payment

Tracks payment transactions associated with orders.

- **Primary Key:** payment_id (NOT NULL)
- **Attributes:**
  - **Foreign Key:** order_id (FK to Orders, NOT NULL)
  - amount_paid (DECIMAL(10,2))
  - payment_method (ENUM)
  - payment_date (TIMESTAMP)

3 ER Diagram

Here is the ER Diagram:

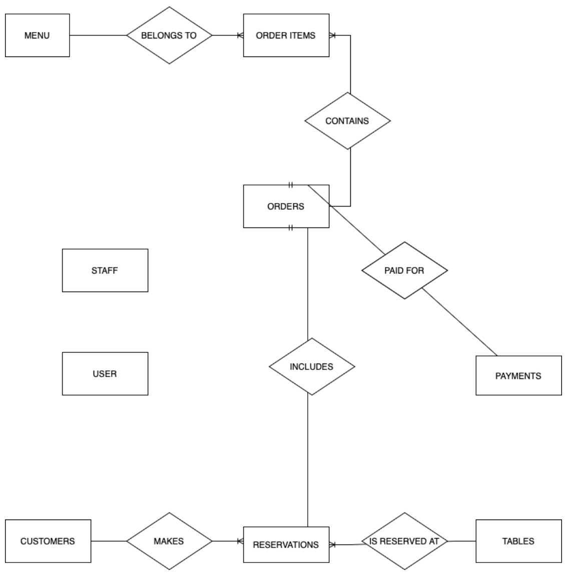

Figure 1: ER Diagram

4 Relationships Between Tables

4.1 Customers and Reservations

- **Type:** One-to-Many
- **Cardinality:** One customer can have multiple reservations, but each reservation is linked to a single customer.
- **Modality:**
  - Mandatory for Reservations (every reservation must have a customer).
  - Optional for Customers (not all customers have reservations).

4.2 Tables and Reservations

- **Type:** One-to-Many
- **Cardinality:** One table can be reserved multiple times, but each reservation is linked to one table.
- **Modality:**
  - Mandatory for Reservations (every reservation must have a table).
  - Optional for Tables (not all tables are reserved at a given time).

4.3 Reservations and Orders

- **Type:** One-to-Many
- **Cardinality:** Each reservation can result in multiple orders, and each order corresponds to one reservation.
- **Modality:**
  - Optional for both (a reservation may not lead to an order, and an order cannot exist without a reservation).

4.4 Orders and Order_Items

- **Type:** One-to-Many
- **Cardinality:** One order can have multiple order items, but each order item belongs to one order.
- **Modality:**
  - Mandatory for Order_Items (every order item must belong to an order).
  - Optional for Orders (not all orders have order items).

4.5 Menu and Order_Items

- **Type:** One-to-Many
- **Cardinality:** One dish from the menu can appear in multiple order items, but each order item corresponds to one menu item.
- **Modality:**
  - Mandatory for Order_Items (every order item must refer to a dish).
  - Optional for Menu (not all menu items are ordered).

4.6 Orders and Payments

- **Type:** One-to-Many
- **Cardinality:** Each order can have multiple payments, and each payment is linked to one order.
- **Modality:**

  - Mandatory for Payments (every payment must correspond to an order).
  - Optional for Orders (not all orders are paid immediately).

5 Complex Queries

5.1 Customer Spending Query

5.1.1 Query

```sql
SELECT c.name AS customer_name,
       SUM(o.total_amount) AS total_spent
FROM Customers c
JOIN Reservations r ON c.customer_id = r.customer_id
JOIN Orders o ON r.reservation_id = o.reservation_id
GROUP BY c.customer_id
ORDER BY total_spent DESC;
```

Listing 1: Customer Spending Query

5.1.2 Explanation

- **Purpose:** Calculate the total spending of each customer.
- **Tables Involved:** Customers, Reservations, and Orders.
- **Steps:**
  - Join the tables to link customers, reservations, and orders.
  - Calculate the total amount spent per customer using `SUM()`.
  - Group results by customer ID and sort in descending order to show top spenders.

5.1.3 Sample Output

```
| customer_name        | total_spent |
|----------------------|-------------|
| John Doe             | 500.75      |
| Jane Smith           | 320.50      |
| Alice Johnson        | 250.00      |
```

5.2 Popular Dishes Query

5.2.1 Query

```sql
SELECT m.dish_name,
       COUNT(oi.dish_id) AS total_orders
FROM Menu m
JOIN Order_Items oi ON m.dish_id = oi.dish_id
GROUP BY m.dish_id
ORDER BY total_orders DESC
LIMIT 5;
```

Listing 2: Popular Dishes Query

5.2.2 Explanation

- **Purpose:** Find the top 5 most popular dishes based on the number of orders.
- **Tables Involved:** Menu and Order_Items.
- **Steps:**
  - Join the tables to link dishes and order items.
  - Count the number of times each dish appears in orders using `COUNT()`.
  - Group results by dish ID and sort by order count in descending order.
  - Limit output to the top 5 dishes.

5.2.3 Sample Output

```
| dish_name                 | total_orders |
|---------------------------|--------------|
| Margherita Pizza          | 25           |
| Caesar Salad              | 20           |
| Grilled Chicken           | 15           |
| Spaghetti Bolognese       | 12           |
| Chocolate Lava Cake       | 10           |
```

5.3 Pending Orders Query

5.3.1 Query

```sql
SELECT o.order_id,
       m.dish_name,
       oi.quantity
FROM Orders o
JOIN Order_Items oi ON o.order_id = oi.order_id
JOIN Menu m ON oi.dish_id = m.dish_id
WHERE o.order_status = 'Pending'
ORDER BY o.order_id;
```

Listing 3: Pending Orders Query

5.3.2 Explanation

- **Purpose:** Retrieve all pending orders with dish names and quantities.
- **Tables Involved:** Orders, Order_Items, and Menu.
- **Steps:**
  - Join the tables to link orders, order items, and menu dishes.
  - Filter orders with status = 'Pending'.
  - Display results ordered by order ID.

5.3.3 Sample Output

```
| order_id | dish_name                 | quantity |
|----------|---------------------------|----------|
| 1        | Margherita Pizza          | 2        |
| 1        | Caesar Salad              | 1        |
| 2        | Grilled Chicken           | 3        |
| 3        | Spaghetti Bolognese       | 4        |
```

5.4 Customers Who Spent Above Average

5.4.1 Query

```sql
SELECT c.name AS customer_name,
       total_spent
FROM (
    SELECT r.customer_id,
           SUM(o.total_amount) AS total_spent
    FROM Reservations r
    JOIN Orders o ON r.reservation_id = o.reservation_id
    GROUP BY r.customer_id
) AS spending
JOIN Customers c ON c.customer_id = spending.customer_id
WHERE spending.total_spent > (
    SELECT AVG(total_spent)
    FROM (
        SELECT SUM(o.total_amount) AS total_spent
        FROM Reservations r
        JOIN Orders o ON r.reservation_id = o.reservation_id
        GROUP BY r.customer_id
    ) AS avg_spending
);
```

Listing 4: Customers Who Spent Above Average

5.4.2 Explanation

- **Purpose:** Retrieve customers who spent above the average spending.
- **Tables Involved:** Customers, Reservations, and Orders.
- **Steps:**
  - Compute total spending for each customer using a subquery.
  - Calculate the average spending with a nested query.
  - Filter and retrieve customers whose spending exceeds the average value.

5.4.3 Sample Output

```
| customer_name         | total_spent  |
|----------------------|-------------|
| John Doe              | 700.50       |
| Jane Smith            | 640.00       |
| Alice Johnson         | 620.00       |
```

6 CRUD Operations and Implementations

6.1 Analytics Endpoints

6.1.1 Customer Spending Analytics

GET /api/analytics/customer_spending

6.1.2 Popular Dishes Analytics

GET /api/analytics/popular_dishes

6.1.3 Pending Orders Details

GET /api/analytics/pending_orders_details

6.2 Customers Endpoints

6.2.1 Add a New Customer

POST /api/customers/add

6.2.2 Get All Customers

GET /api/customers

6.2.3 Get a Specific Customer

GET /api/customers/{customer_id}

6.2.4 Update a Customer

PUT /api/customers/{customer_id}

6.2.5 Delete a Customer

DELETE /api/customers/{customer_id}

6.3 Menu Endpoints

6.3.1 Add a New Dish

POST /api/menu/add

6.3.2 Get All Menu Items

GET /api/menu

6.3.3 Get a Specific Dish

GET /api/menu/{dish_name}

6.3.4 Update a Dish

PUT /api/menu/{dish_id}

6.3.5 Delete a Dish

DELETE /api/menu/{dish_id}

6.4 Order Items Endpoints

6.4.1 Add a New Order Item

POST /api/order_items/add

6.4.2 Get All Order Items

GET /api/order_items

6.4.3 Get a Specific Order Item

GET /api/order_items/{order_item_id}

6.4.4 Update an Order Item

PUT /api/order_items/{order_item_id}

6.4.5 Delete an Order Item

DELETE /api/order_items/{order_item_id}

6.5 Orders Endpoints

6.5.1 Add a New Order

POST /api/orders/add

6.5.2 Get All Orders

GET /api/orders

6.5.3 Get a Specific Order

GET /api/orders/{order_id}

6.5.4 Update an Order

PUT /api/orders/{order_id}

6.5.5 Delete an Order

DELETE /api/orders/{order_id}

6.6 Payments Endpoints

6.6.1 Add a Payment

POST /api/payments/add

6.6.2 Update a Payment

PUT /api/payments/{payment_id}

6.6.3 Delete a Payment

DELETE /api/payments/{payment_id}

6.7 Staff Endpoints

6.7.1 Add a Staff Member

POST /api/staff/add

6.7.2 Get All Staff Members

GET /api/staff

6.7.3 Get a Specific Staff Member

GET /api/staff/{staff_id}

6.7.4 Update a Staff Member

PUT /api/staff/{staff_id}

6.7.5 Delete a Staff Member

DELETE /api/staff/{staff_id}

6.8 Tables Endpoints

6.8.1 Add a New Table

POST /api/tables/add

6.8.2 Get All Tables

GET /api/tables

6.8.3 Get a Specific Table

GET /api/tables/{table_id}

6.8.4 Update a Table

PUT /api/tables/{table_id}

6.8.5 Delete a Table

DELETE /api/tables/{table_id}

7 Challenges and Solutions

- **Database Design:** Ensured normalization and defined relationships using ER diagrams. Added constraints and indexes for data integrity.
- **Authentication and Security:** Implemented JWT for secure authentication and encrypted sensitive data.
- **Complex Queries:** Optimized SQL queries with indexing and JOIN operations for analytics and reporting.
- **Validation and Error Handling:** Applied schema validation and meaningful error messages with HTTP status codes.

- **API Documentation:** Used Swagger for endpoint documentation, ensuring alignment with implementations.