**Food Management System**

**Requirements Document**

Roshini Seelamsetty, Nida Tazeen

Purdue University

ACS 560: Software Engineering

Matthew Parker

9th October 2024

Version Summary: Version 1.0 – This project focuses on analysing food delivery data to gain insights into customer preferences, restaurant performance, and delivery efficiency.

Table Of Contents

[Introduction: 2](#_Toc1460367016)

[What will the project do? 2](#_Toc1090111839)

[Who will benefit from the project? 3](#_Toc775932709)

[What data is used? 3](#_Toc1410874743)

[Glossary: 4](#_Toc1231567570)

[User Requirements 5](#_Toc420099632)

[System Requirements 6](#_Toc1224766382)

[System Architecture: 6](#_Toc71255620)

[1. Client-Server Architecture 6](#_Toc598251976)

[2. Layered Architecture 6](#_Toc212477348)

[3. Repository (Façade) Pattern 6](#_Toc1238375637)

[System Models: Class Diagram, Use Case Diagram and Sequence Diagram 8](#_Toc749279008)

[Class Diagram 9](#_Toc1539929837)

[Use Case Diagram 10](#_Toc722084650)

[Sequence Diagrams 12](#_Toc1551894786)

[1. Retrieve All Orders 13](#_Toc589395940)

[2. Delete an Order 13](#_Toc176315969)

[3. Update an Order 14](#_Toc286977478)

[4. Add a New Order 15](#_Toc1520629904)

[5. Retrieve Order by Cuisine 16](#_Toc1109032517)

[6. Retrieve Order by Rating 17](#_Toc568018282)

[7. Retrieve Order by Restaurant Name 18](#_Toc1054731813)

[8. Analyse Delivery Times 21](#_Toc1880170330)

[9. Monitor Food Preparation Time 22](#_Toc982434228)

[10. Evaluate Customer Satisfaction 23](#_Toc677346690)

[11. Identify Trends 23](#_Toc98968680)

[12. Generate Reports 24](#_Toc406834870)

## **Introduction**:

The Food Management System project aims to analyse food delivery orders from various restaurants, providing valuable insights into customer behaviour, restaurant preferences, and service efficiency. By evaluating key metrics such as delivery time, order cost, and food preparation time, this system strives to enhance decision-making processes for both customers and restaurants, leading to improved service quality and customer satisfaction.

This system will benefit restaurant owners, delivery services, and customers by offering comprehensive insights based on the data. The dataset used in the project includes variables such as order ID, restaurant name, cuisine type, order cost, customer ratings, food preparation time, and delivery time, ensuring a holistic understanding of the food delivery process.

## **What will the project do?**

The Food Management System will analyse food delivery orders from multiple restaurants, providing insights into various aspects of the food delivery process. Specifically, the system will:

* Analyse Delivery Times: Calculate and analyse the average delivery times for each restaurant and cuisine type, allowing for performance comparisons.
* Monitor Food Preparation Time: Track the time taken by restaurants to prepare food, helping to identify bottlenecks in the process.
* Evaluate Customer Satisfaction: Use customer ratings to assess service quality and food satisfaction, offering insights into areas for improvement.
* Identify Trends: Analyse data to highlight trends such as peak order times, popular cuisines, and the busiest days for restaurants.
* Generate Reports: Produce detailed reports on restaurant performance, customer preferences, and delivery efficiency, aiding restaurant owners and managers in optimizing their services.
* Support Decision-Making: Provide restaurant owners and delivery services with data-driven insights that can improve operational efficiency and enhance the customer experience.

## **Who will benefit from the project?**

The Food Management System will benefit the following groups:

* Restaurant Owners: By providing insights into order trends, delivery times, and customer satisfaction, restaurant owners can optimize their operations, improve food preparation efficiency, and enhance customer service.
* Delivery Services: The system helps delivery services identify peak hours, improve delivery times, and address any logistical challenges, resulting in faster and more reliable deliveries.
* Customers: Customers benefit from better service as restaurants and delivery services use data-driven insights to reduce wait times, improve food quality, and enhance the overall delivery experience.
* Restaurant Managers: Managers can use the system to monitor restaurant performance, track key metrics, and make informed decisions about staffing, inventory management, and service improvements.
* Business Analysts: Analysts can leverage the data to provide strategic insights, improving both short-term and long-term business decisions for restaurants and delivery companies.

## **What data is used?**

The Food Management System utilizes the following data to provide valuable insights and optimize operations:

* Order ID: A unique identifier for each food delivery order.
* Customer ID: A unique identifier for each customer placing an order.
* Restaurant Name: The name of the restaurant fulfilling the order.
* Cuisine Type: The type of cuisine being ordered (e.g., Korean, Japanese, American).
* Cost of the Order: The total amount spent on each order.
* Day of the Week: The day when the order was placed, categorized as a weekday or weekend.
* Customer Rating: Feedback or rating provided by the customer for the food or service (if available).
* Food Preparation Time: The time taken by the restaurant to prepare the order before it is ready for delivery.
* Delivery Time: The time it takes for the order to be delivered to the customer after preparation.

This data helps to analyse various aspects of the food delivery process, including customer preferences, restaurant performance, and service efficiency.

## **Glossary:**

* Order ID: A unique numerical identifier assigned to each food delivery order.
* Customer ID: A unique identifier for each customer placing an order, used to track customer-specific data.
* Restaurant Name: The name of the restaurant responsible for preparing and delivering the food.
* Cuisine Type: A category or style of food offered by the restaurant (e.g., Korean, Japanese, American).
* Cost of the Order: The total price of the food order, including any applicable taxes or delivery fees.
* Day of the Week: A designation indicating whether the order was placed on a weekday or weekend, useful for analyzing trends.
* Rating: A score or feedback provided by the customer, reflecting their satisfaction with the food and delivery service.
* Food Preparation Time: The amount of time (in minutes) taken by the restaurant to prepare the food after receiving the order.
* Delivery Time: The amount of time (in minutes) it takes for the food to be delivered to the customer after preparation is complete.
* Busiest Day: The day of the week with the highest number of orders, used to identify peak operational periods.
* Average Delivery Time: The average amount of time taken to deliver food across all orders, useful for measuring efficiency.
* Peak Hours: The times during the day when the highest number of food orders are placed.

## **User Requirements**

* Order Management: The system should allow users to input, track, and manage food orders from various restaurants.
* Inventory Tracking: The system must keep track of inventory levels of ingredients, alerting users when items are running low.
* Menu Management: Users should be able to update and manage the restaurant's menu, including adding, editing, and removing items.
* Order Cost Calculation: The system should automatically calculate the total cost of each order, including taxes and any applicable discounts.
* Customer Information Management: Users should be able to store and manage customer information, such as contact details and order history.
* Preparation Time Tracking: The system should track and display the time taken to prepare each order.
* Daily Sales Report: Users should be able to generate daily reports on total sales and order volumes.
* Order Status Updates: The system must allow users to update the status of each order (e.g., in preparation, ready for delivery, delivered).
* Employee Scheduling: The system should allow users to schedule kitchen staff and delivery personnel based on projected order volumes.
* Supplier Management: Users should be able to track and manage supplier information, including order histories and contact details.
* Waste Management: The system should track and report on food wastage to help users identify inefficiencies.
* Feedback and Reviews: The system should allow users to collect and manage customer feedback and reviews on food quality and service.

## **System Requirements**

* Scalable Database: The system shall be built using a scalable database to handle large volumes of order data efficiently.
* User-Friendly Interface: The system shall have a user-friendly interface for both restaurant owners and customers, ensuring ease of navigation and usability.
* Web Accessibility: The system shall be accessible via web applications, allowing users to access the platform from various devices with internet connectivity.
* Data Analytics Capabilities: The system shall include data analytics capabilities to generate insights from order data, such as trends in customer preferences and restaurant performance.

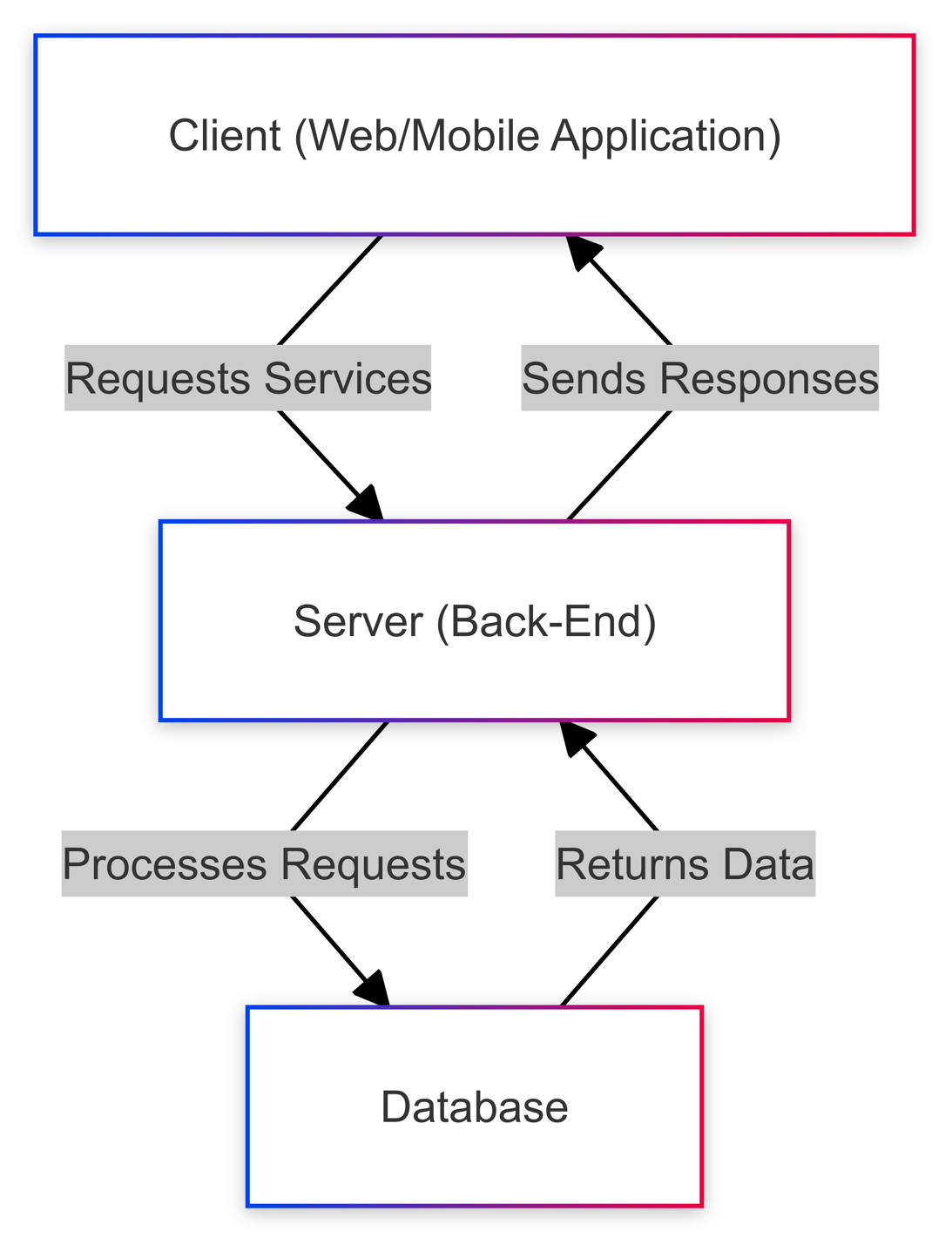
## **System Architecture:**

**Recommended Architectural Patterns**

1. **Client-Server Architecture**
2. **Layered Architecture**
3. **Repository (Façade) Pattern**

### **1. Client-Server Architecture**

Overview: The client-server architecture is a foundational pattern where the system is divided into clients (front-end) and servers (back-end). Clients request services, and servers provide them.



**Relevance to Project:**

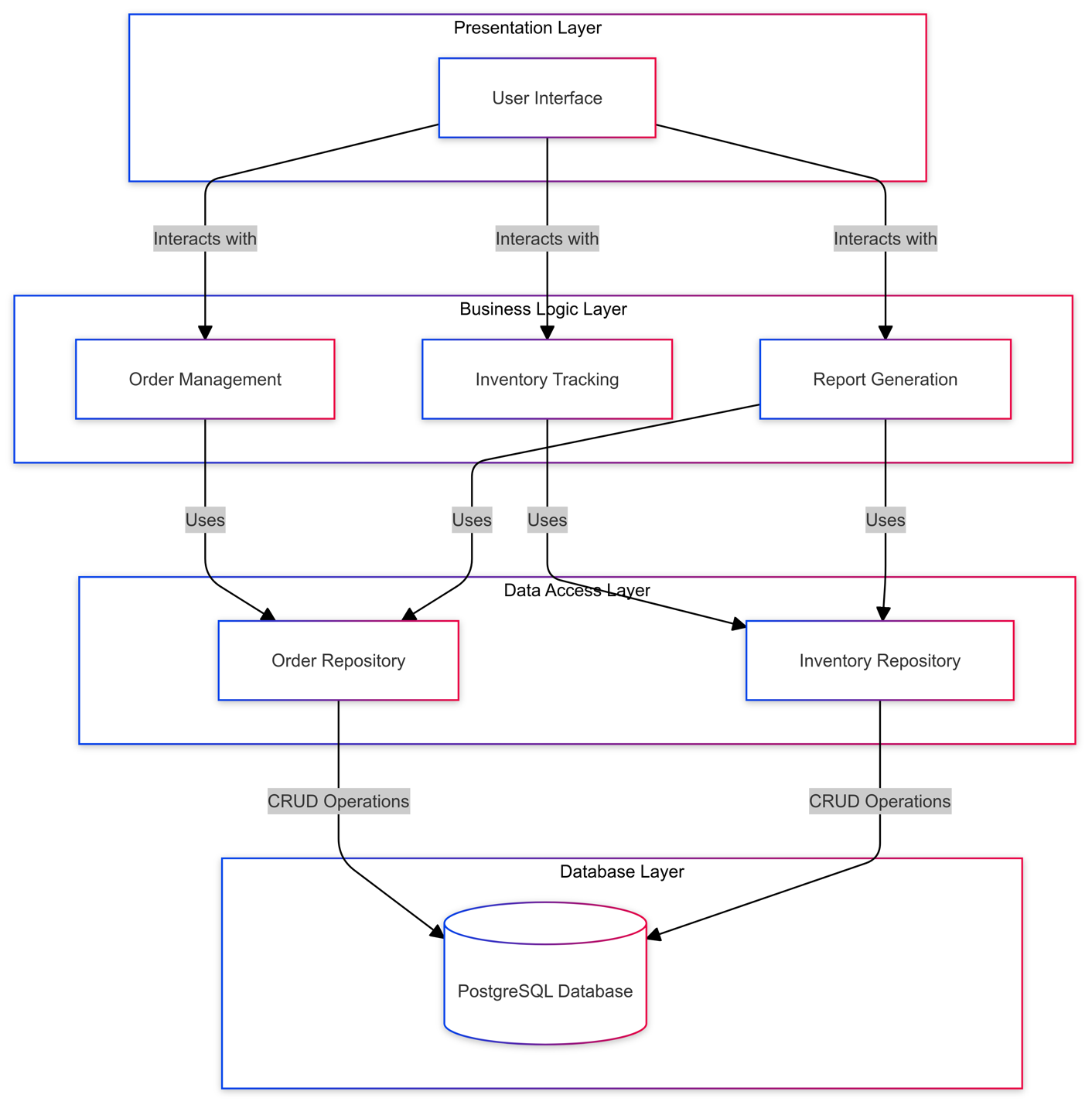
* Separation of Concerns: Clearly distinguishes between the user interface and the business logic/data management.
* Scalability: Allows independent scaling of client and server components based on demand.
* Security: Centralizes data management, making it easier to implement security measures.

**Implementation in Project:**

* Client Layer: Web-based application or mobile app for users (customers, restaurant owners, admins).
* Server Layer: Handles business logic, processes requests, interacts with the database, and performs data analytics.

### **2. Layered Architecture**

Overview: Layered architecture organizes the system into distinct layers, each with specific responsibilities. Common layers include Presentation, Business Logic, Data Access, and Database.



Relevance to Project:

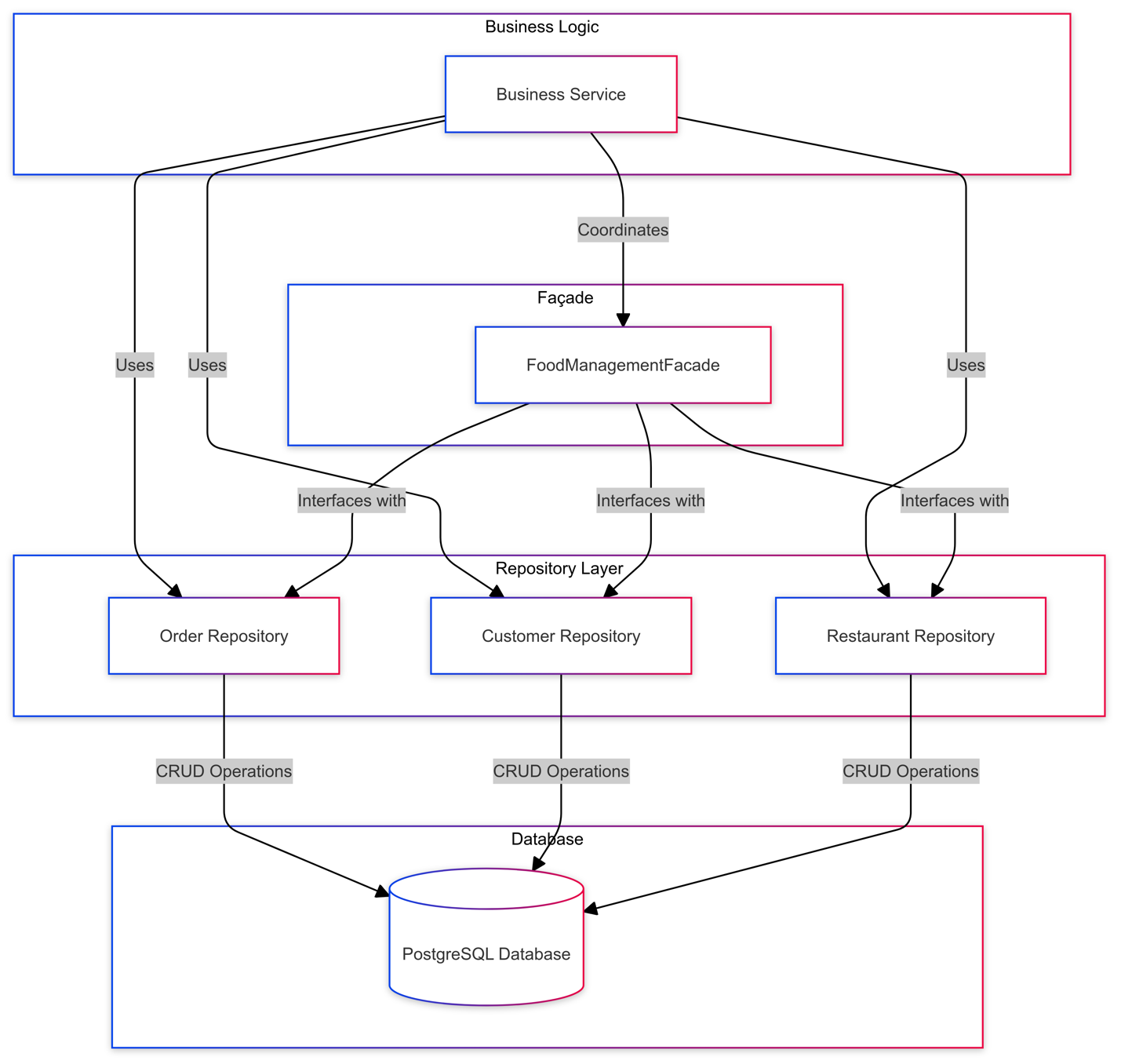
* Modularity: Enhances maintainability by allowing developers to work on different layers independently.
* Reusability: Promotes reuse of components within the same layer or across different layers.
* Ease of Testing: Facilitates unit testing by isolating functionality within layers.

Implementation in Project:

* Presentation Layer:
  + Handles the user interface and user interactions.
  + Technologies: HTML/CSS, JavaScript frameworks (e.g., React, Angular).
* Business Logic Layer:
  + Contains core functionalities like order management, inventory tracking, analytics, and report generation.
  + Technologies: Server-side languages (e.g., Node.js, Python, Java).
* Data Access Layer:
  + Manages data retrieval and storage, interacting with the database.
  + Implements the Repository Pattern for data abstraction.
* Database Layer:
  + Stores all persistent data related to orders, customers, restaurants, and inventory.
  + Technologies: Relational databases (e.g., MySQL, PostgreSQL).

### **3. Repository (Façade) Pattern**

**Overview:** The Repository Pattern provides an abstraction over the data layer, encapsulating data access logic and promoting a clean separation between business logic and data storage.



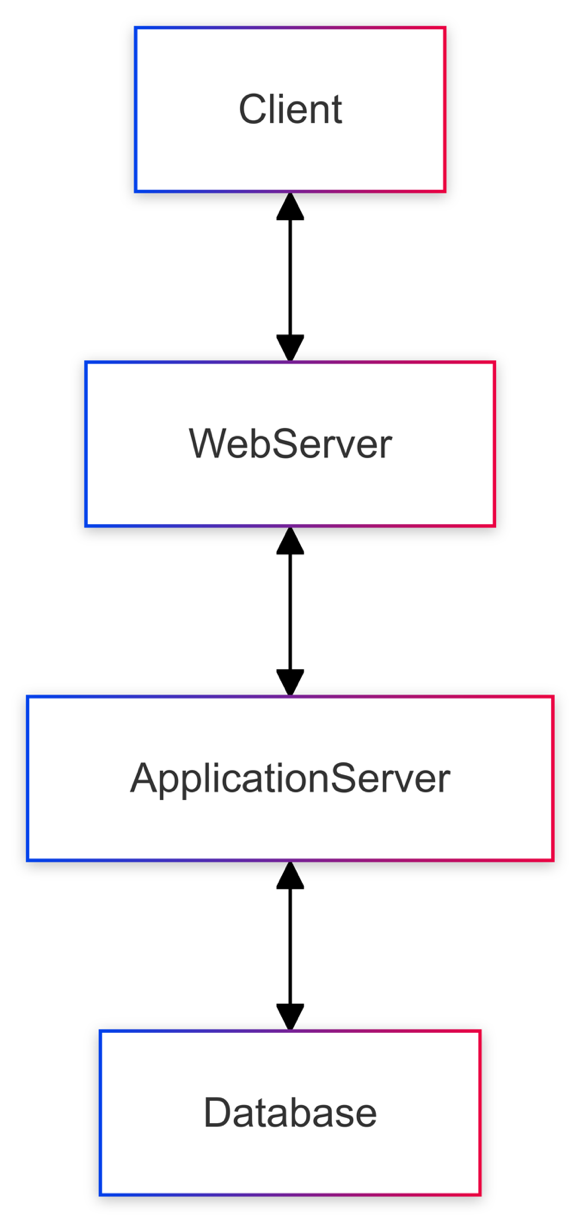
**Relevance to Project:**

* **Decoupling:** Separates business logic from data access, making the system more modular and easier to maintain.
* **Flexibility:** Allows swapping out data storage mechanisms without affecting business logic.
* **Testability:** Facilitates mocking data access in unit tests.

**Implementation in Project:**

* **Repositories:**
  + Create repository classes for each entity (e.g., OrderRepository, CustomerRepository, RestaurantRepository).
  + Implement CRUD operations and complex queries within these repositories.
  + Interface with the Data Access Layer to perform database interactions.
* **Façade:**
  + Provide a simplified interface to the complex subsystems (e.g., multiple repositories).
  + Example: A FoodManagementFacade that coordinates between different repositories and business logic components.

For this project, we will implement a **Client-Server Architecture**, as it perfectly aligns with our requirements by offering a clear separation between the front-end (client) and back-end (server). This architecture enhances scalability, maintainability, and performance, ensuring seamless interaction between users and the system while efficiently handling business logic, data processing, and communication.



Client-server architecture

**1. Client-Server Architecture:**

**Client Layer:** This layer consists of user interfaces that allow users (customers, restaurant owners, and administrators) to interact with the system. It can be a web-based application or a mobile app, designed for ease of use and accessibility.

**Server Layer:** This layer includes the application logic, data processing, and business rules. It is responsible for handling requests from clients, performing operations on the database, and returning responses.

**2. Components:**

* **User Interface (UI):**
  + Web Application
* **API Layer:**
  + RESTful API that handles all client requests
  + Processes requests and communicates with the database
* **Business Logic Layer:**
  + Handles core functionalities such as order management, inventory tracking, and analytics
  + Implements algorithms for calculating average delivery times, food preparation times, and generating reports
* **Database Layer:**
  + A scalable relational database (e.g., MySQL, PostgreSQL) for storing all data related to orders, customers, restaurants, and inventory
  + Includes tables for Orders, Customers, Restaurants

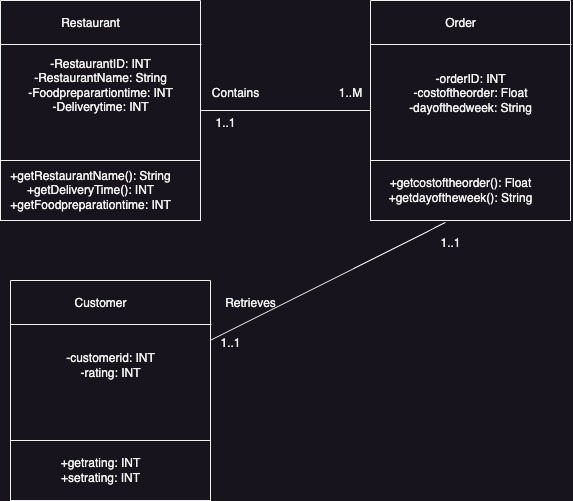
**3. Data Flow:**

* **User Interaction:**
  + Users interact with the UI to perform various tasks (e.g., placing an order, generating reports etc).
* **Request Handling:**
  + The UI sends requests to the REST API when users perform actions.
* **Business Logic Execution:**
  + The API forwards the requests to the business logic layer, which processes the requests.
* **Database Interaction:**
  + The business logic layer communicates with the database to retrieve or store data.
* **Response to Client:**
* The API sends the response back to the client, which updates the UI accordingly.

## **System Models:**

## Class Diagram, Use Case Diagram and Sequence Diagram

## Class Diagram



This class diagram illustrates the relationships between three main entities in a system related to restaurants, orders, and customers.

Classes:

* Restaurant
* Attributes:
* RestaurantID: INT: A unique identifier for each restaurant.
* RestaurantName: String: The name of the restaurant.
* Foodpreparationtime: INT: The time it takes for the restaurant to prepare food.
* Deliverytime: INT: The delivery time associated with the restaurant.
* Methods:
* getRestaurantName(): String: A method to retrieve the restaurant's name.
* getDeliveryTime(): INT: A method to retrieve the delivery time.
* getFoodpreparationtime(): INT: A method to retrieve the food preparation time.
* Order
* Attributes:
* orderID: INT: A unique identifier for each order.
* costof the order: Float: The total cost of the order.
* dayoftheweek: String: The day of the week when the order was placed.
* Methods:
* getcostoftheorder(): Float: A method to retrieve the cost of the order.
* getdayoftheweek(): String: A method to retrieve the day of the week for the order.
* Customer
* Attributes:
* customerid: INT: A unique identifier for each customer.
* rating: INT: A rating given by the customer.
* Methods:
* getrating(): INT: A method to retrieve the customer's rating.
* setrating(INT): A method to set or update the customer's rating.

Relationships:

* Restaurant and Order:
* Association: A Restaurant "contains" one or more Order instances (1..M), meaning each restaurant can have multiple orders, but each order belongs to only one restaurant.
* Multiplicity: The multiplicity is defined as 1..1 on the restaurant side and 1..M on the order side, indicating that each restaurant can handle many orders.
* Customer and Order:
* Association: A Customer retrieves one or more Order instances (1..M), meaning each customer can place multiple orders, and each order is associated with only one customer.
* Multiplicity: The multiplicity is 1..1 on the customer side and 1..1 on the order side, indicating that an order is placed by one customer, and each customer can have multiple orders.

## Use Case Diagram

A diagram of a restaurant

Description automatically generated

Actors:

* Customer: A user who interacts with the system to search and retrieve information about restaurants.
* Admin: A user responsible for managing the restaurant database, including adding, updating, and deleting restaurant information.

Use Cases:

* Retrieve Restaurant: Both customers and admins can retrieve general restaurant details from the system.
* Retrieve Restaurant by Name: Customers and admins can search for restaurants based on their name.
* Retrieve Restaurant by Range: Customers and admins can filter restaurants based on a specified distance or geographic location.
* Retrieve Restaurant by Rating and Cuisines\*\*: Users can retrieve restaurants based on a combination of their ratings and the cuisines they offer.
* Retrieve Restaurant by Filtering Cuisines: Customers and admins can retrieve restaurants by applying filters based on specific cuisines.
* Add New Restaurant: Only admins have the privilege to add new restaurants into the system database.
* Update Restaurant Ratings: Admins can update or modify the ratings of restaurants.
* Delete a Restaurant: Admins can remove a restaurant from the system.
* Generate Reports: Admins can generate reports about the restaurants, such as performance summaries.
* Identify Trends: Admins can use the system to identify emerging trends based on restaurant ratings, cuisine popularity, and customer interactions.
* Evaluate Customer Satisfaction: Admins can assess customer satisfaction based on feedback and ratings collected by the system.
* Analyze Delivery Times: Admins can review and analyze restaurant delivery times for optimization.

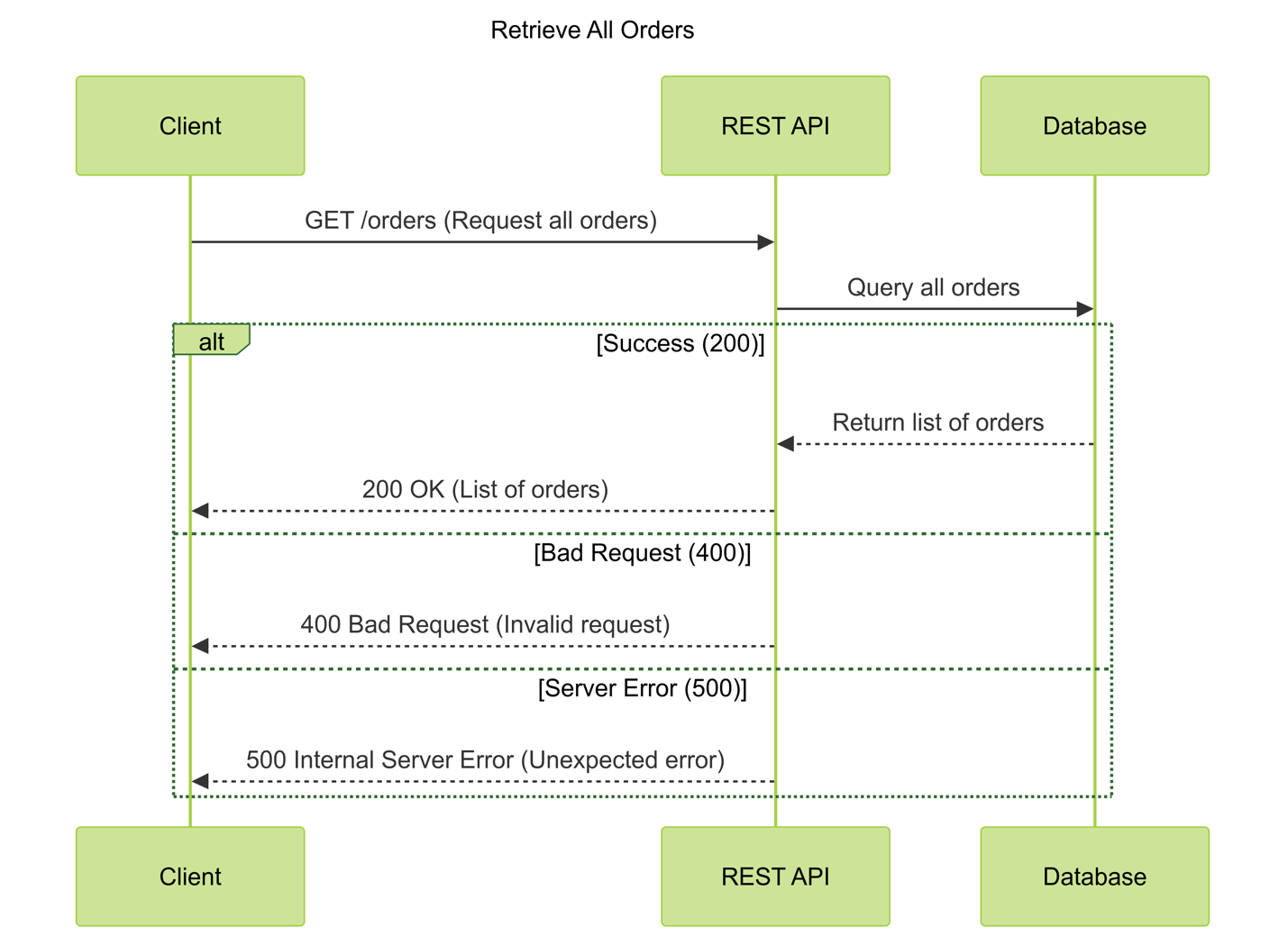
Flow of Interactions:

* Customer: Customers mainly interact with the system to retrieve various types of restaurant information, such as by name, location, rating, and cuisine filters. They are limited to viewing the restaurant data.
* Admin: Admins have extended functionalities beyond retrieval. They can add, update, and delete restaurant records, generate reports, evaluate trends, and perform customer satisfaction analyses.

This high-level diagram outlines the roles of both customers and admins within the restaurant management system, showing how they interact with the system's use cases.

## Sequence Diagrams

### **1. Retrieve All Orders**



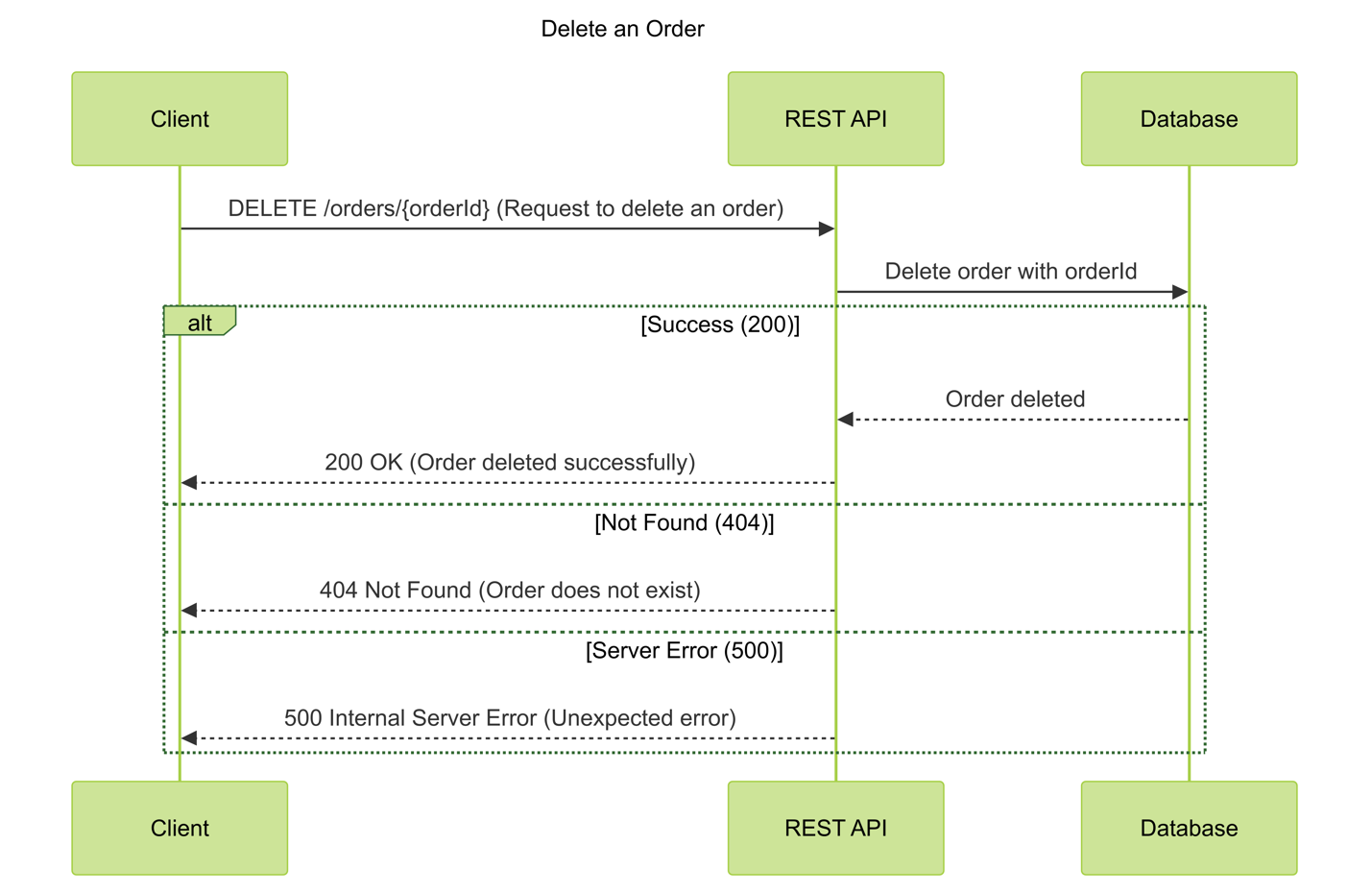
This demonstrates how all orders are retrieved from the system:

**Actors:** Client, REST API, Database

**Sequence:**

1. The client requests all orders via the API.
2. The REST API processes the request and queries the database.
3. The database returns the list of orders.
4. The API sends the list of orders back to the client.

### **2. Delete an Order**



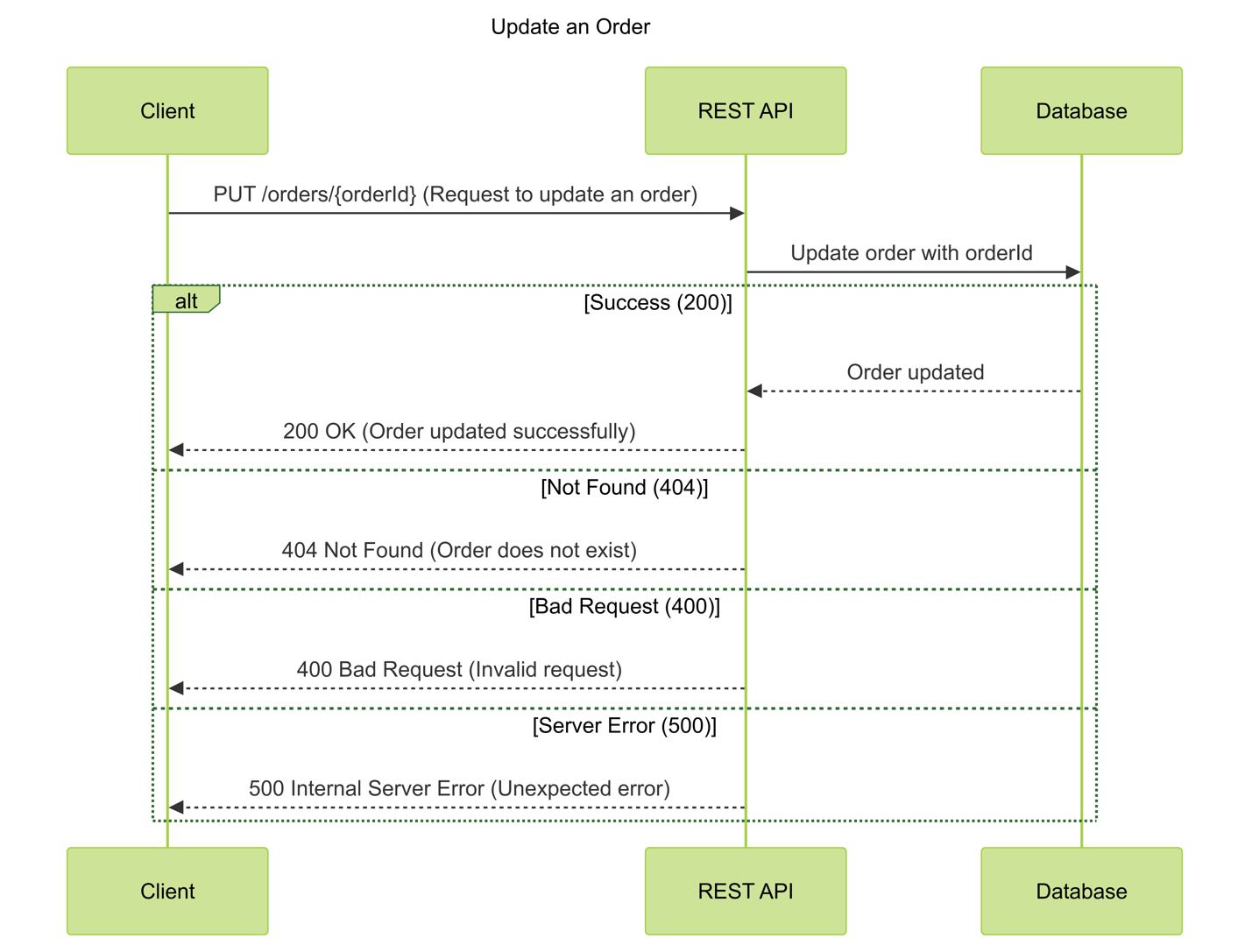
This demonstrates how an order is deleted from the system:

**Actors:** Client, REST API, Database

**Sequence:**

1. The client sends a request to delete an order via the API.
2. The REST API processes the delete request.
3. The API queries the database to delete the order.
4. The database confirms the deletion.
5. The API sends a response back to the client indicating success or failure.

### **3. Update an Order**



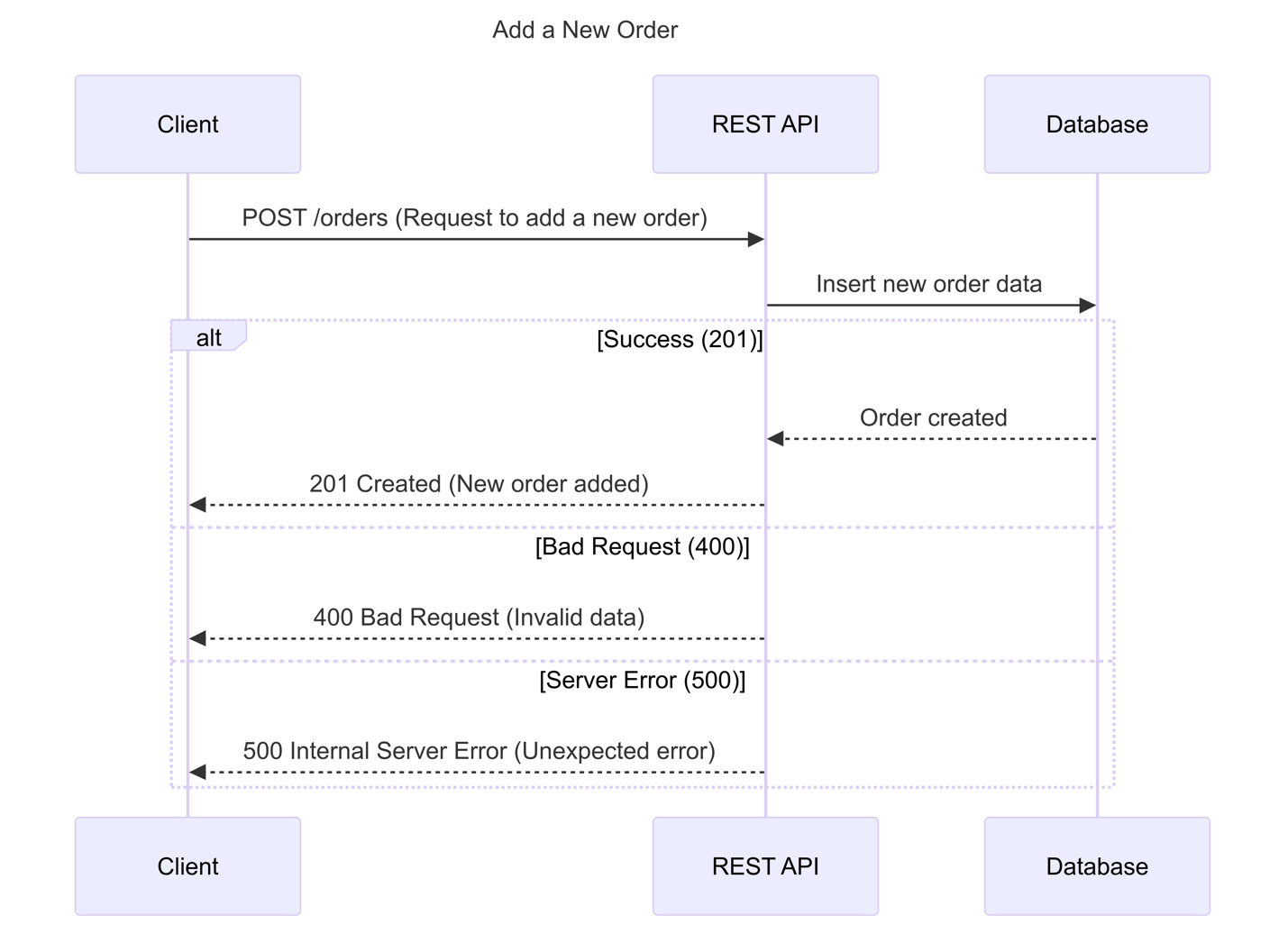
This demonstrates how an existing order is updated:

**Actors:** Client, REST API, Database

**Sequence:**

1. The client sends a request to update an order via the API.
2. The REST API processes the update request.
3. The API updates the order details in the database.
4. The database confirms the update.
5. The API sends a response back to the client indicating success or failure.

### **4. Add a New Order**



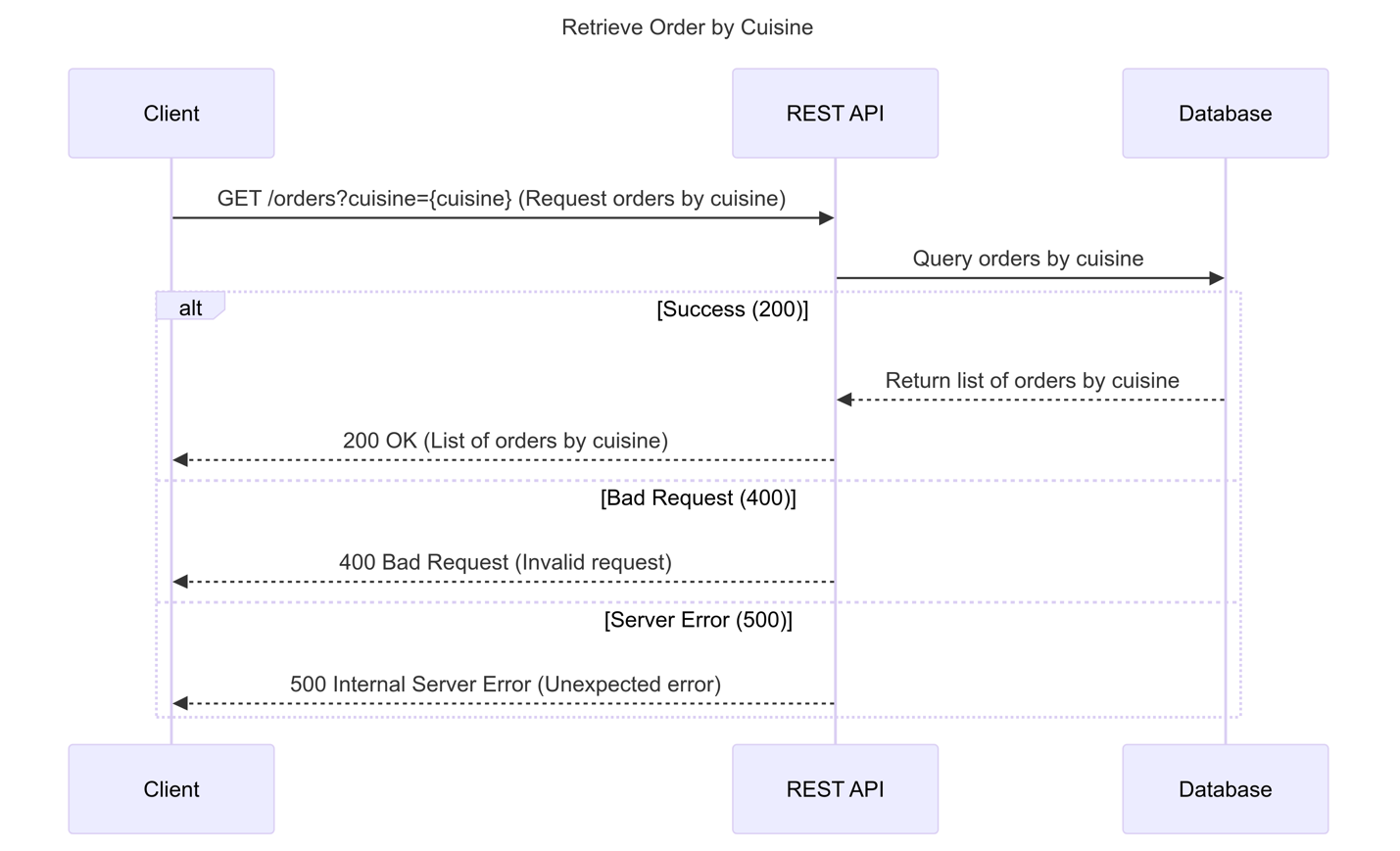
This demonstrates how a new order is added to the system:

**Actors:** Client, REST API, Database

**Sequence:**

1. The client sends a request to add a new order via the API.
2. The REST API processes the request and validates the data.
3. The API saves the new order in the database.
4. The database confirms the order creation.
5. The API sends a response back to the client indicating success or failure.

### **5. Retrieve Order by Cuisine**



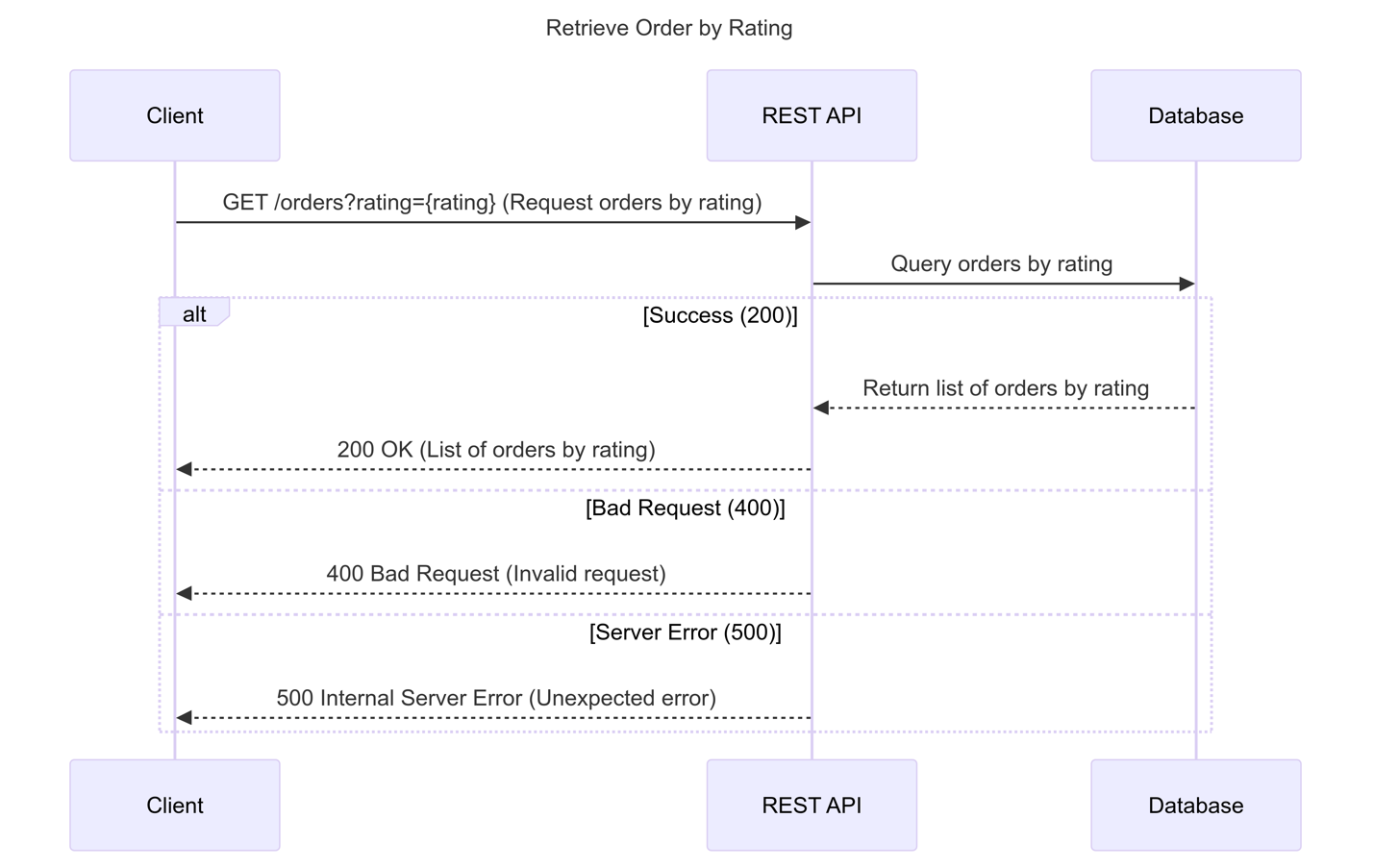
This demonstrates how orders are retrieved by cuisine type:

**Actors:** Client, REST API, Database

**Sequence:**

1. The client requests orders filtered by cuisine type via the API.
2. The REST API processes the request and queries the database.
3. The database returns the orders for the specified cuisine.
4. The API sends the list of orders back to the client.

### **6. Retrieve Order by Rating**



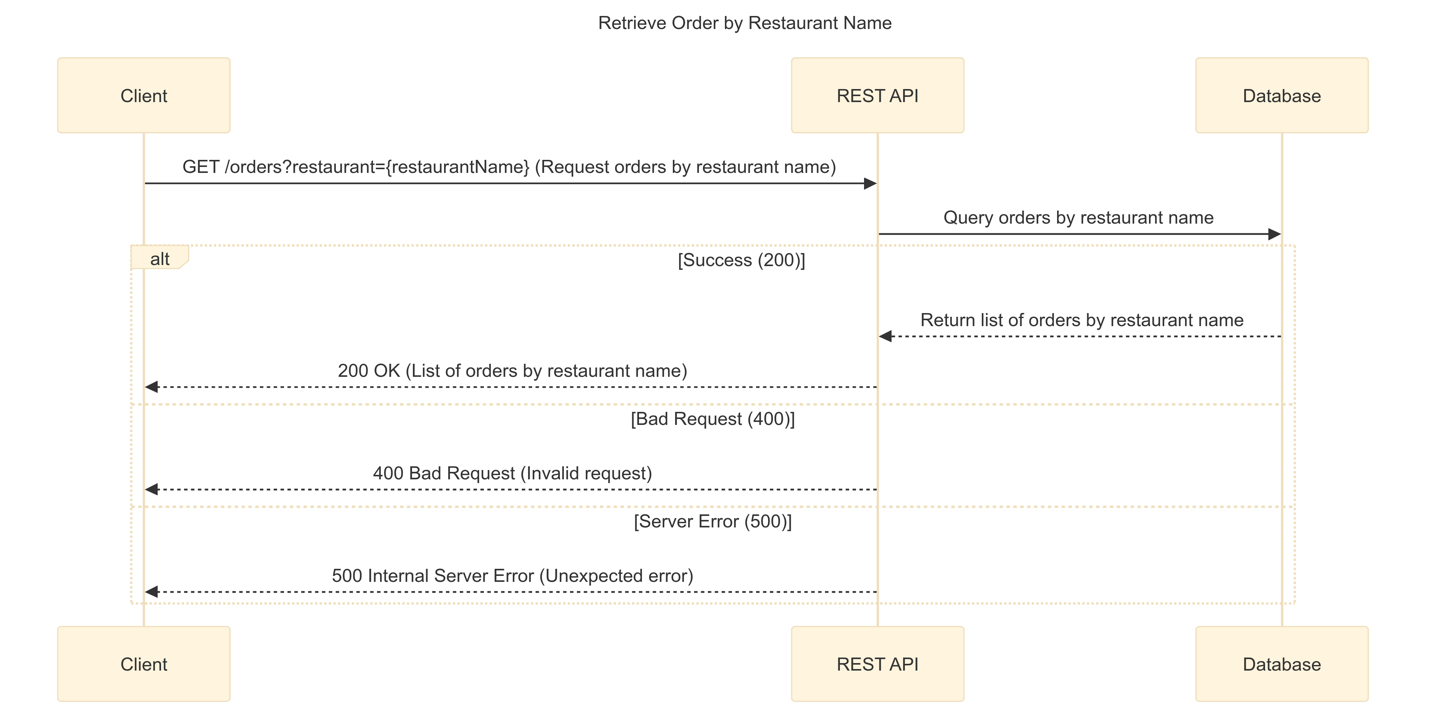
This demonstrates how orders are retrieved by customer rating:

**Actors:** Client, REST API, Database

**Sequence:**

1. The client requests orders filtered by rating via the API.
2. The REST API processes the request and queries the database.
3. The database returns the orders with the specified rating.
4. The API sends the list of orders back to the client.

### **7. Retrieve Order by Restaurant Name**

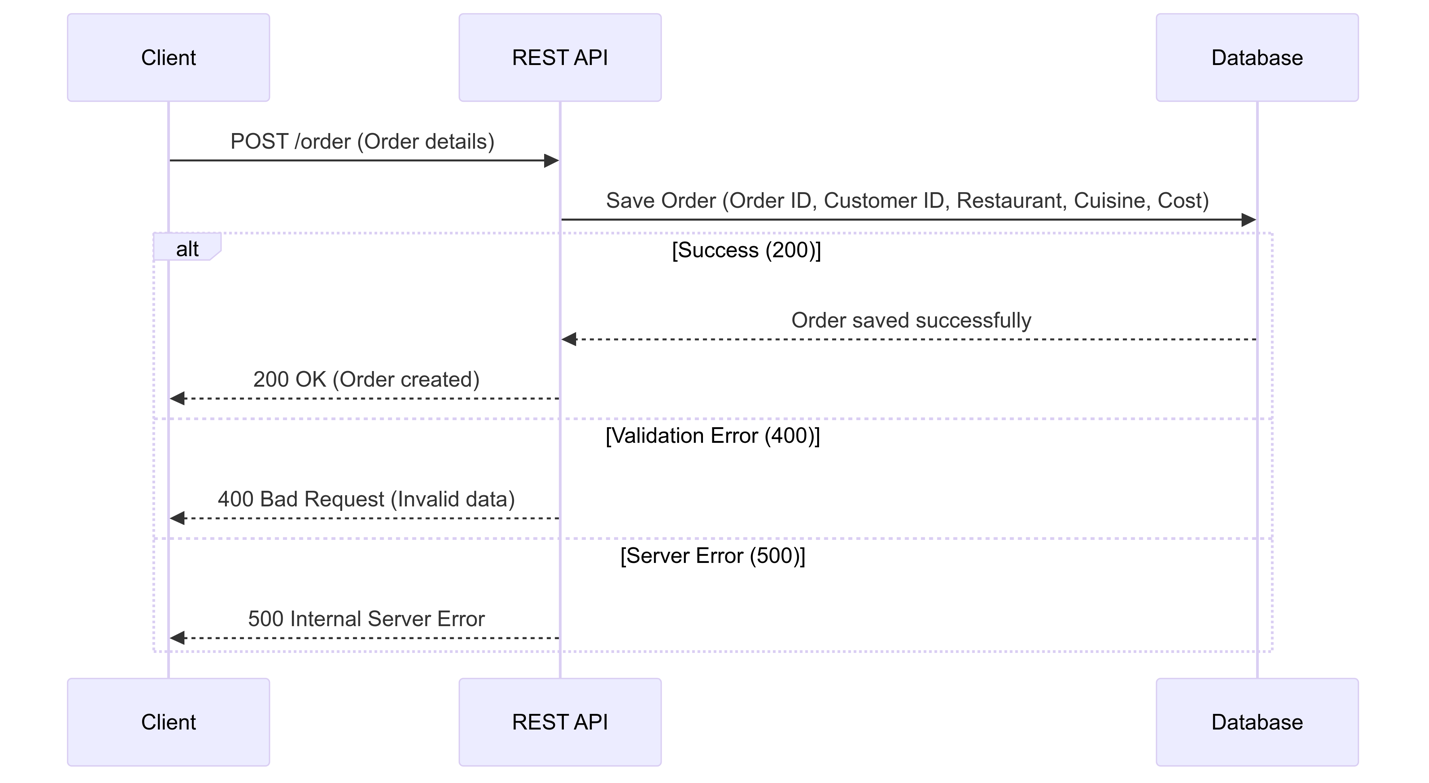


This demonstrates how orders are retrieved by restaurant name:

**Actors:** Client, REST API, Database

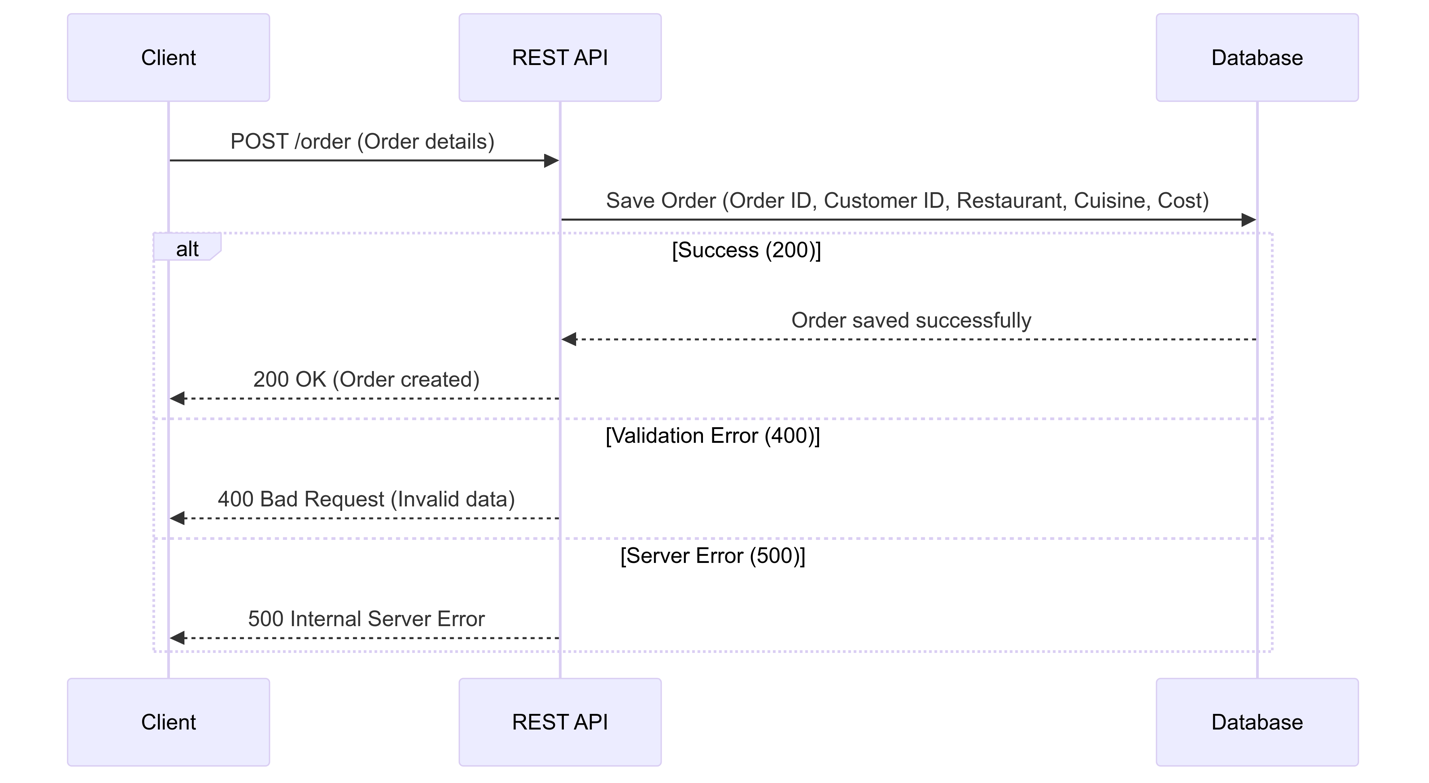
**Sequence:**

1. The client requests orders filtered by restaurant name via the API.
2. The REST API processes the request and queries the database.
3. The database returns the orders for the specified restaurant.
4. The API sends the list of orders back to the client.

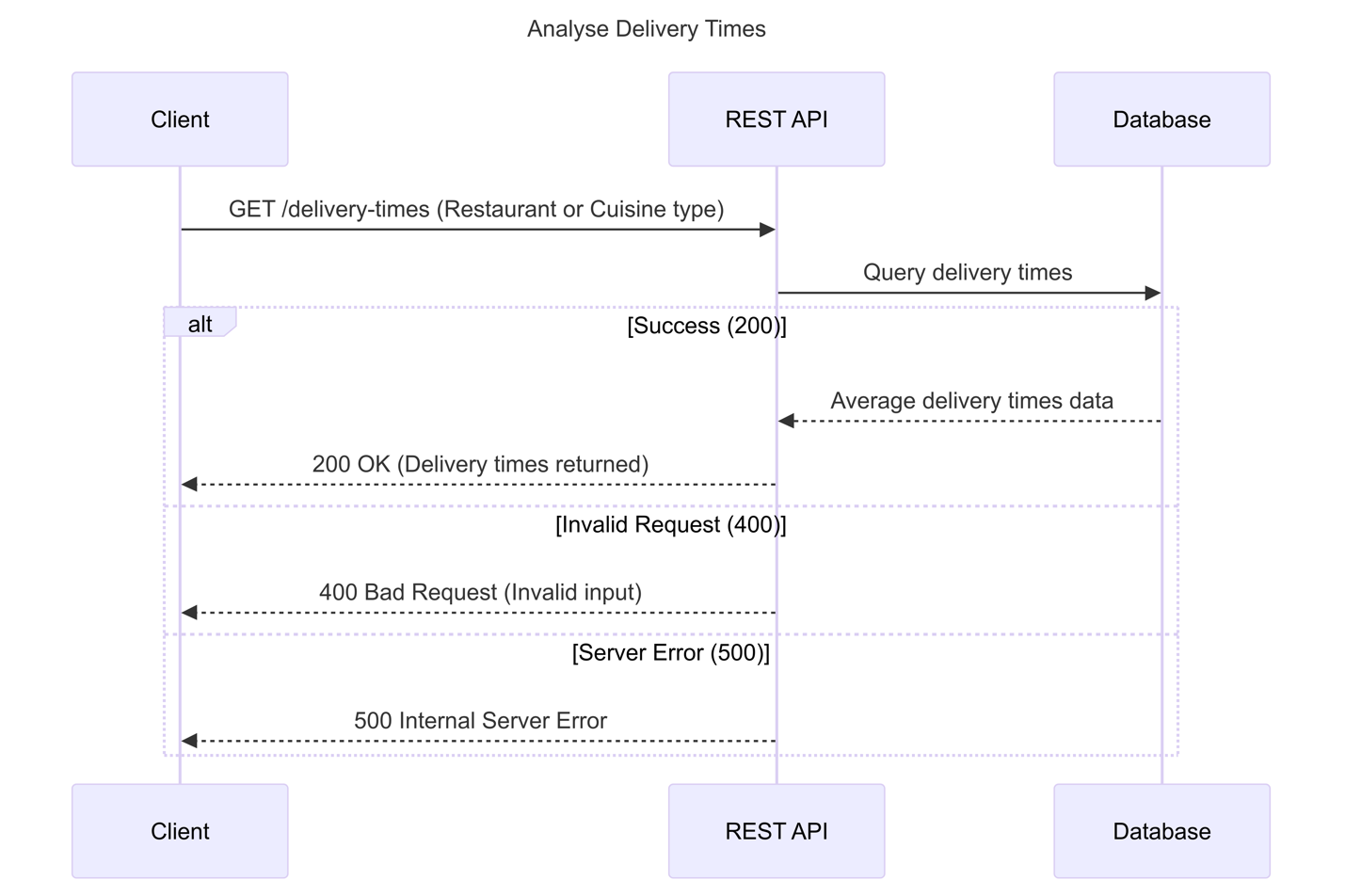


This diagram will depict how orders are tracked and stored:

* **Actors**: Client, REST API, Database
* **Sequence**:
  + Client submits order (order ID, customer ID, restaurant, cuisine type, order cost) to the API.
  + REST API processes the request.
  + API records the order in the database.
  + A success response is returned to the client.



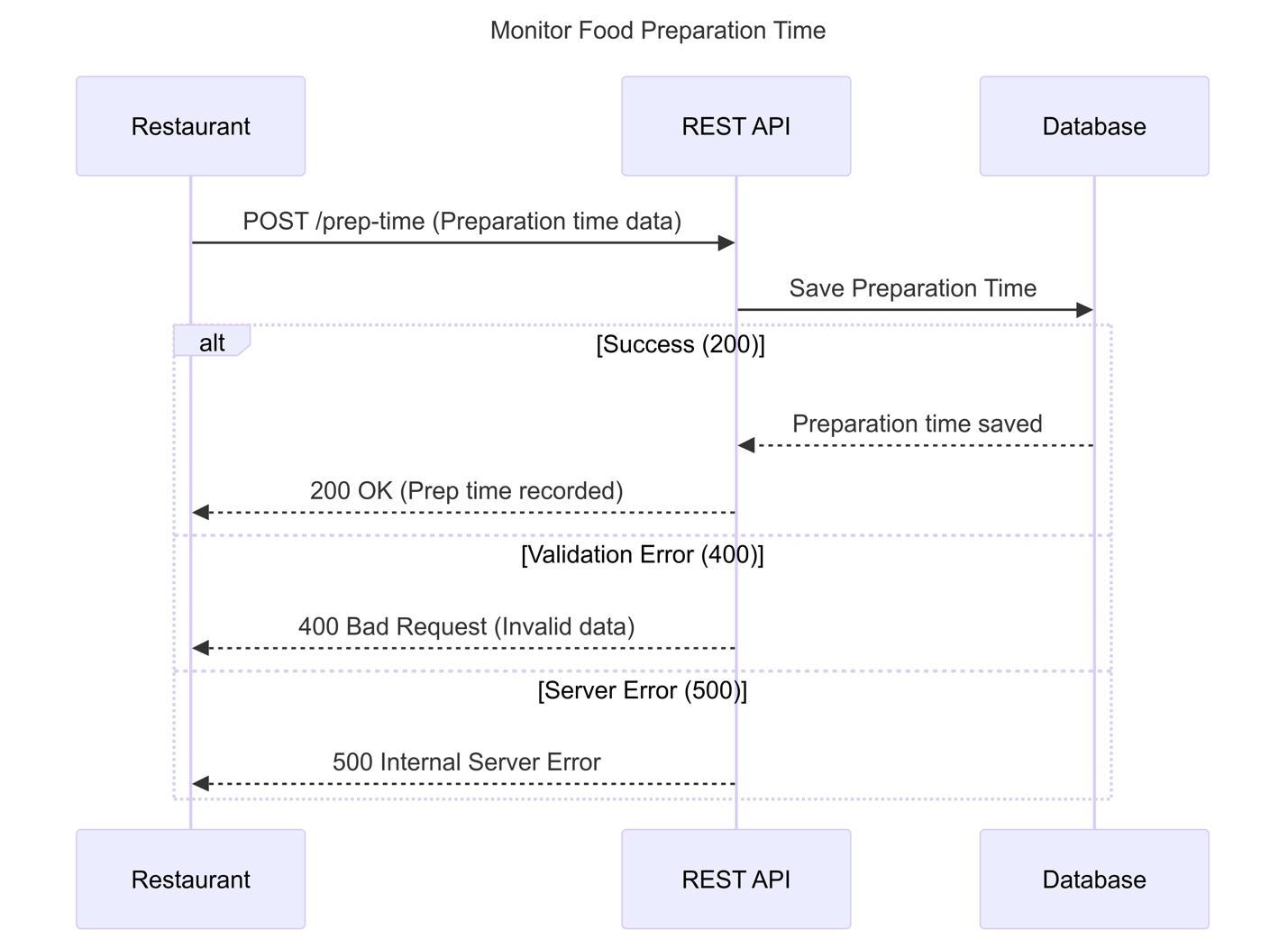
### **8. Analyse Delivery Times**



This shows how delivery times are calculated:

* **Actors**: Client, REST API, Database
* **Sequence**:
  + Client requests average delivery times for a restaurant or cuisine type.
  + REST API fetches order history from the database.
  + API calculates average delivery times for each restaurant/cuisine type.
  + API returns the calculated data to the client.

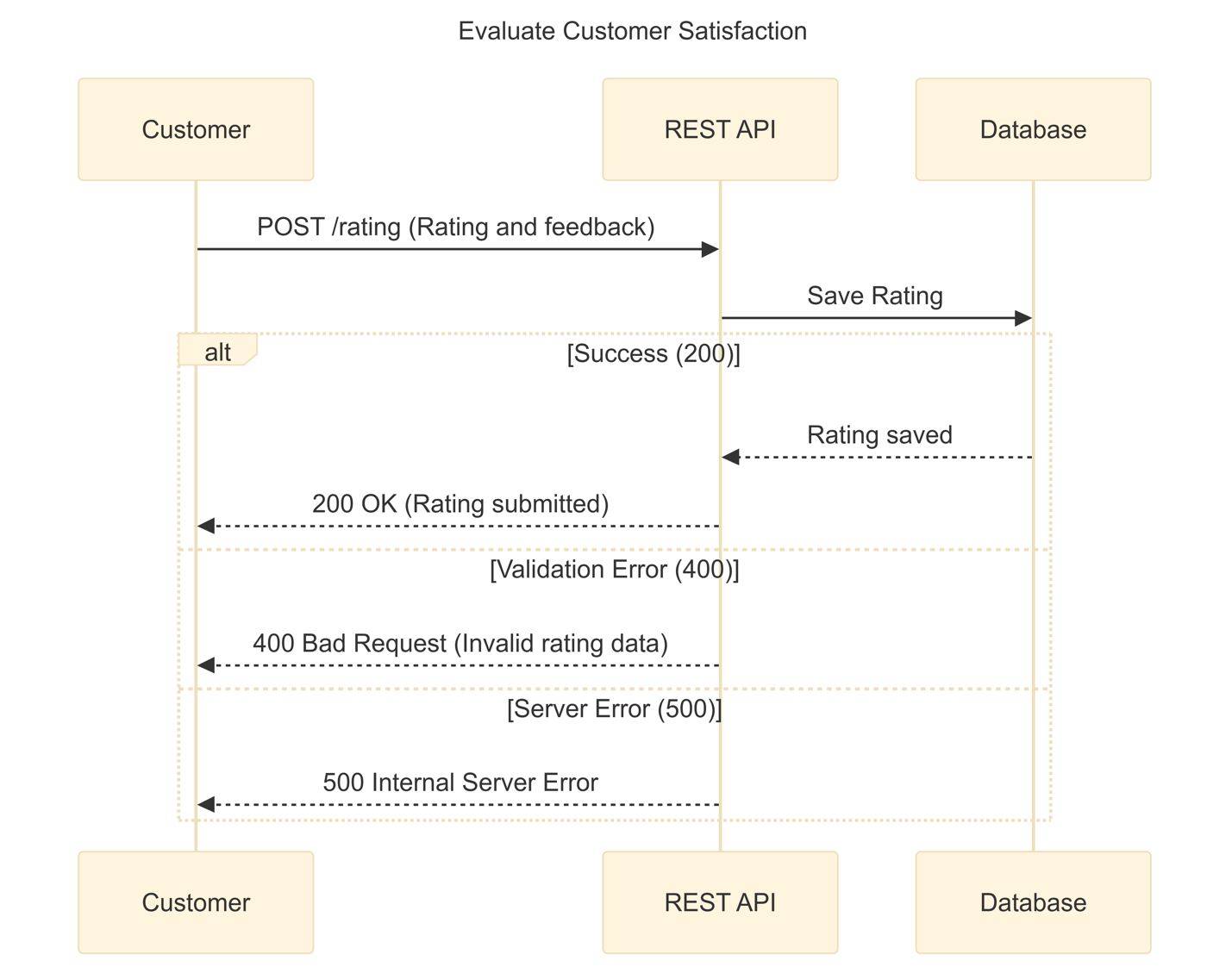
### **9. Monitor Food Preparation Time**



This diagram illustrates how preparation time is monitored:

* **Actors**: Client (or restaurant), REST API, Database
* **Sequence**:
  + Restaurant submits food preparation time to the API.
  + REST API processes the preparation time.
  + API stores the preparation time data in the database.
  + A confirmation is sent back to the restaurant/client.

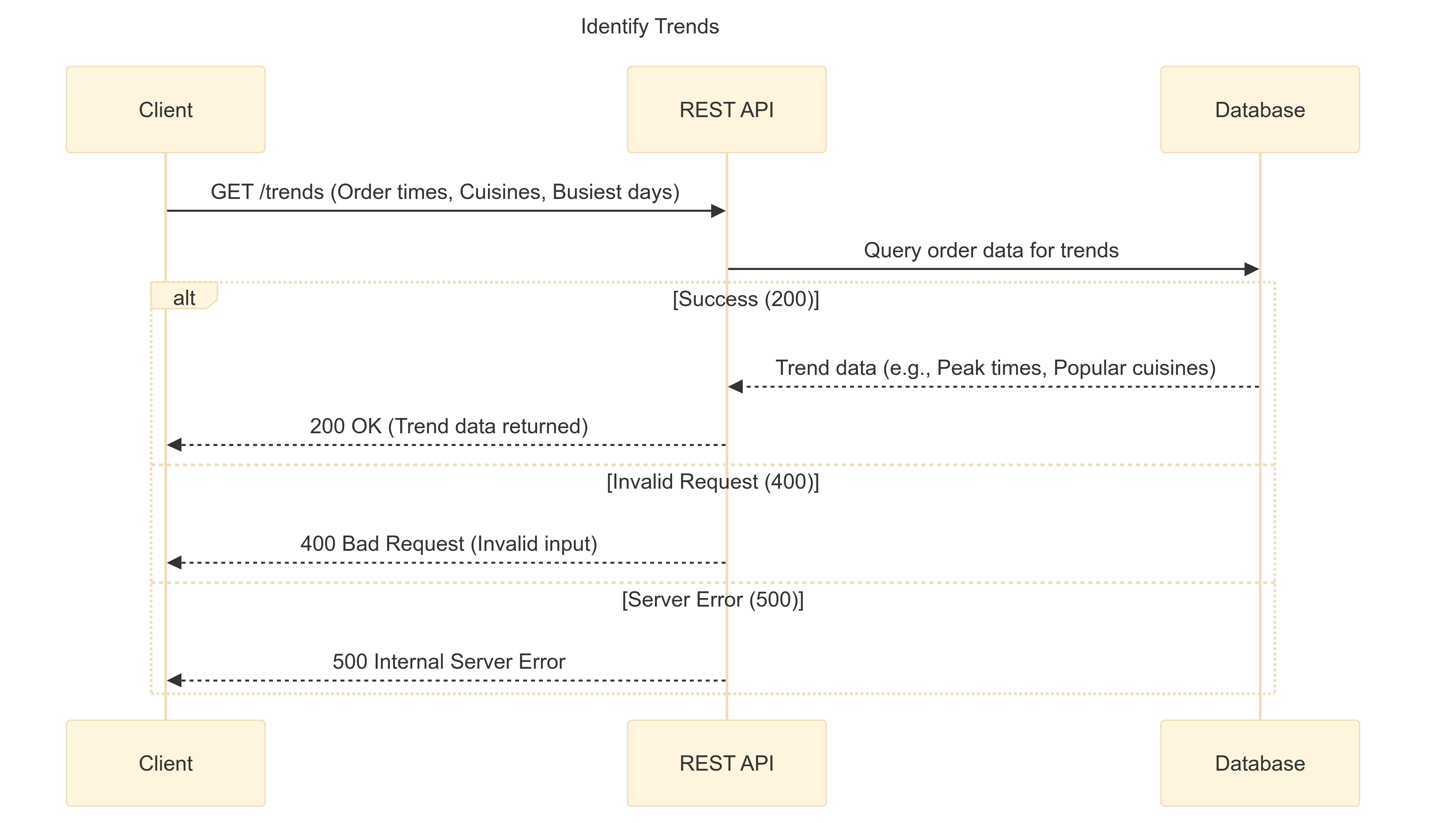
### **10. Evaluate Customer Satisfaction**



This demonstrates how customer ratings are gathered and evaluated:

* **Actors**: Customer, REST API, Database
* **Sequence**:
  + Customer submits a rating and feedback via the client.
  + REST API receives and processes the rating.
  + API stores the rating in the database.
  + API analyses the rating data and returns a summary (optional).

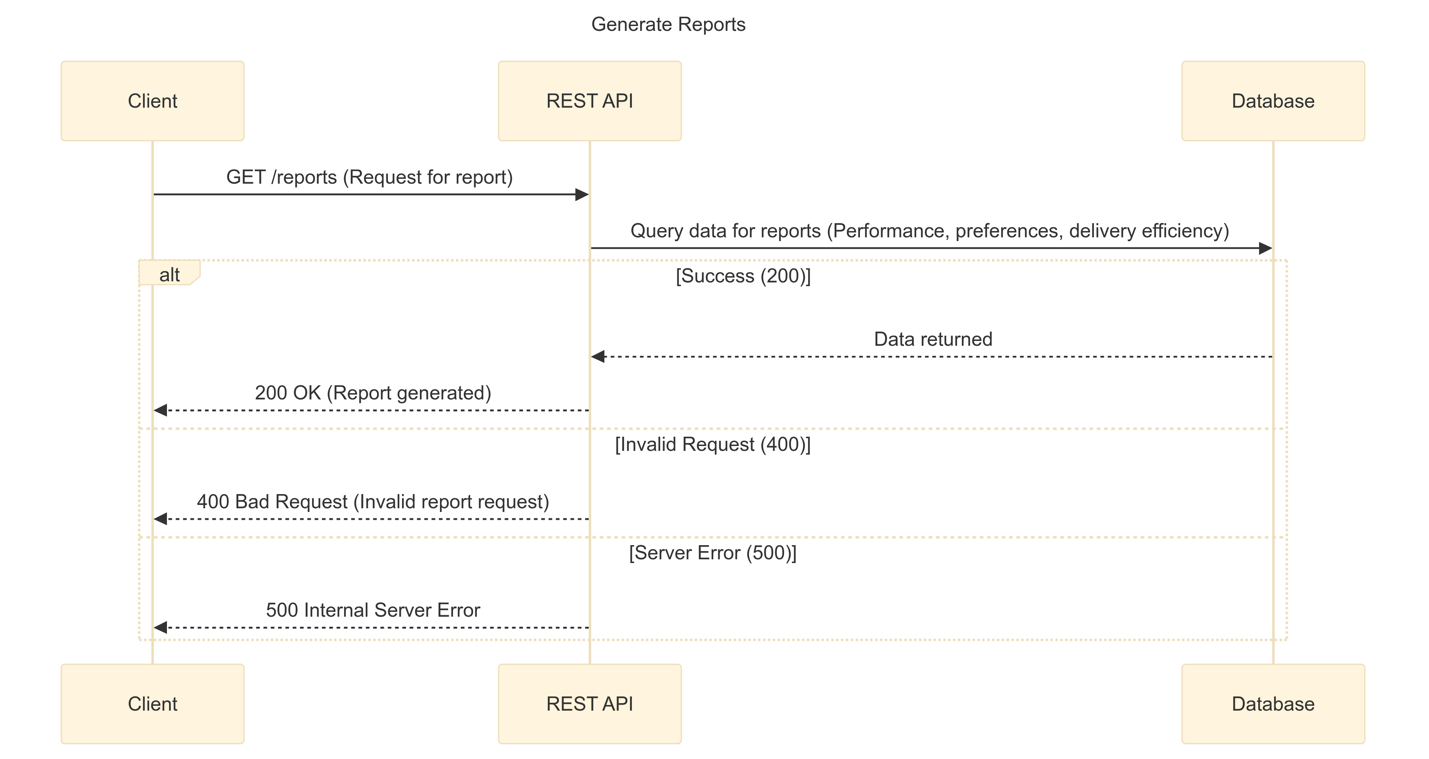
### **11. Identify Trends**



This sequence shows how trends are identified:

* **Actors**: Client, REST API, Database
* **Sequence**:
  + Client requests data to identify trends.
  + REST API queries order data (times, cuisines, popular restaurants) from the database.
  + API processes the data to identify trends (peak order times, busiest days, etc.).
  + API returns trend data to the client.

### **12. Generate Reports**



This sequence will illustrate how reports are generated:

* **Actors**: Client, REST API, Database
* **Sequence**:
  + Client requests a performance or efficiency report.
  + REST API gathers order, delivery, and customer feedback data from the database.
  + API processes and compiles a detailed report.
  + Report is returned to the client in the requested format (PDF, CSV, etc.).

**GitHub Repository**:

<https://github.com/tazeenida/Food-Management-System.git>