

VIETNAM INTERNATIONAL UNIVERSITY – HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

**WEB APPLICATION DEVELOPMENT PROJECT**

**RESTAURANT MANAGEMENT SYSTEM**

By

Trần Gia Khôi - ITITIU21231

Trần Quốc Nam - ITITIU21250

Lê Khánh Nguyên - ITCSIU21211

Advisor: Nguyen Van Sinh

A report submitted to the School of Computer Science and Engineering in partial fulfillment of the requirements for the Final Project in Web Application Development course - Fall 2025

**TABLE OF CONTENT**

**I. Introduction……………………………………………………………………………1**

**1. Overview Requirements……………………………………………………………1**

**2. Product’s Information………………………………………………………………1**

**II. Purpose…………………………………………………………………………………2**

**III. Languages, Methods, Technology Stack………………………………….2**

**1. Languages…………………………………………………………………………….2**

**2. Methods………………………………………………………………………………..2**

**3. Technology Stack…………………………………………………………………….3**

**IV. Analysis and Design……………………………………………………………….3**

**1. Requirement Analysis……………………………………………………………….3**

**2. Design…………………………………………………………………………………..9**

**V. Workflow………………………………………………………………………………..24**

**1. Customer Order Flow………………………………………………………………..24**

**2. Kitchen Order Management………………………………………………………...24**

**3. Waiter Order Management…………………………………………………………..25**

**4. VIP Status Request Flow…………………………………………………………….25**

**5. Admin Dashboard & Management…………………………………………………25**

**6. Payment Workflow Detail……………………………………………………………25**

**7. Order Status Lifecycle……………………………………………………………….25**

**VI. Project Demo………………………………………………………………………….26**

**VII. References…………………………………………………………………………….31**

1. **INTRODUCTION**

**1. Overview Requirements**

Nowadays, technology has evolved and developed rapidly. Due to that, the service career needs to strive and adapt to today’s trend. The restaurant management system project is not an exception. It has been implemented to meet this requirement. The restaurant will be full of technology and as little human labor as possible.

The restaurant will equip on each table a computer with a touching screen. Customers who come to the restaurant will sit at the table and order the meal by themselves using this computer. Then the order will be transmitted to the computer of the cook in the kitchen.

At the kitchen, the cook will enter the estimated time for each order based on his schedule.

Then after he finished the order, he would send it to a delivery system to transport to the customer table. The most important thing is that customers can track the status of their order when they just simply sit at their table.

Then when customers finish their meal, they send a signal to the cashier over the system. The cashier will be notified and confirm the payment process.

For the manager, he/she will manage all of the operation of the restaurant, for example, be able to view and modify the menu, hire employees, and manage tables for customers. He/she uses the system to make those changes in the morning before the restaurant is opened.

By using this system, the manager can make changes by just some simple clicks and can do it anywhere he/she likes. Customers will also be very happy since they do not have to wait for the waiter to order any meal, they can just simply order by themselves at their table. Moreover, they can also view the status of their order to check whether it is finish or not, which is very convenient.

**2. Product’s Information**

The Riverside Restaurant Management System is a web-based application designed to streamline restaurant operations, including customer ordering, staff management, kitchen workflows, reservations, payments, and administrative tasks. The system is built as a full-stack web project with a backend API and frontend user interface. It supports multiple user roles such as customers, waiters (servers), chefs, staff, and admins. The backend handles data persistence using a PostgreSQL database, while the frontend provides intuitive interfaces for interactions.

The project follows modern web development practices, ensuring scalability, security, and efficiency. Key features include order placement, real-time status updates, VIP requests, and comprehensive admin dashboards.

**II. Purpose**

This project mainly served as a group project assignment for a Web Application Development course. By doing this project in a group of 2 to 3 people, we can apply the theory of basic web programming knowledge like HTML, CSS, and JavaScript to real world applications. Moreover, we can also experience with many frameworks and libraries, such as FastAPI, ReactJS to make the web application more professional, which is the key of this course.

**III. Languages, Methods, Technology Stack**

**1. Languages**

* **Python:** Used for the backend API development.
* **JavaScript/ TypeScript:** Used for the frontend application.
* **SQL:** Used for database schema definition and queries (PostgreSQL dialect).

**2. Methods**

This project follows the RESTful API architectural style for the backend. All endpoints are designed to be stateless, resource-oriented, and use standard HTTP methods (GET, POST, PUT, PATCH, DELETE) for CRUD operations. For example:

* GET for retrieving resources (e.g., GET /menu-items/ to fetch menu items).
* POST for creating resources (e.g., POST /orders/ to create a new order).
* PUT/PATCH for updating resources (e.g., PUT /orders/{id} to update order status).
* DELETE for removing resources (e.g., DELETE /menu-items/{id}).

The system also incorporates role-based access control (RBAC) for authentication and authorization, ensuring that actors like customers, staff, and admins have appropriate permissions.

**3. Technology Stack**

* **Backend:** FastAPI (Python framework for building APIs with automatic OpenAPI documentation and async support).
* **Frontend:** ReactJS (JavaScript library for building user interfaces), with React Router for navigation and Axios for HTTP requests.
* **Styling:** TailwindCSS (utility-first CSS framework for rapid UI development).
* **Database:** PostgreSQL (relational database for storing structured data like orders, customers, and menus).
* **Other Tools:** pgAdmin for database management, potentially SQLAlchemy/Pydantic for ORM and data validation in the backend.

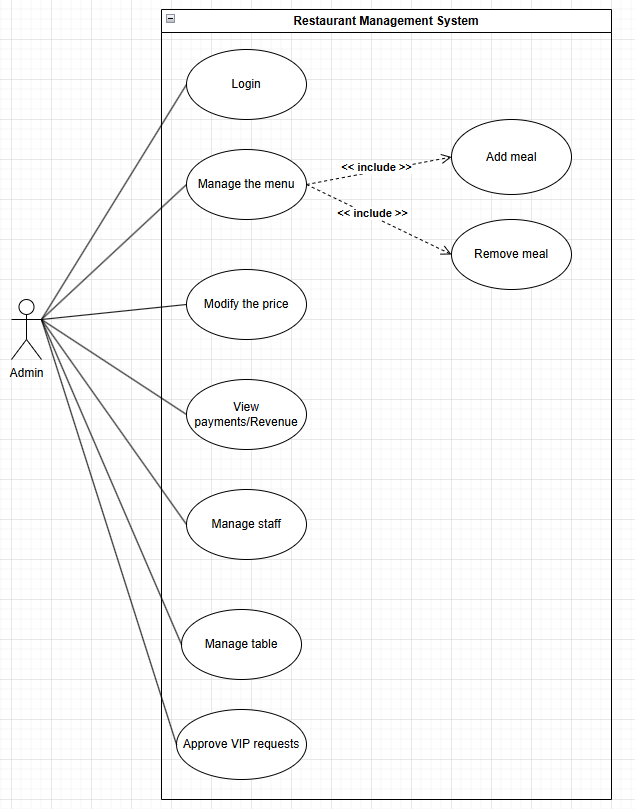
The application follows the MVC (Model-View-Controller) pattern:

* **Model:** Database tables and Pydantic models for data representation.
* **View:** React components for rendering UI.
* **Controller:** FastAPI routes and services for handling business logic.

**IV. Analysis and Design**

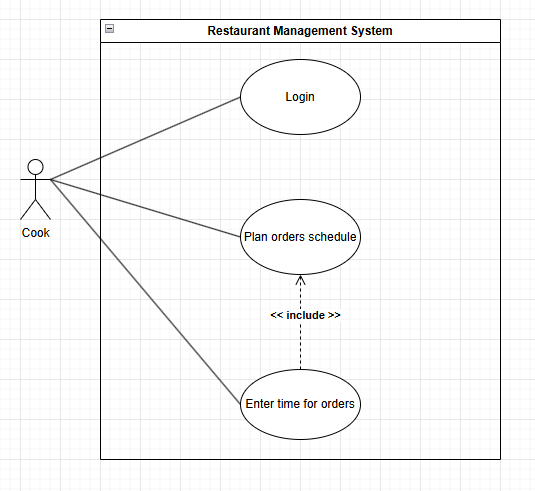
**1. Requirement Analysis**

**Actor: Admin**



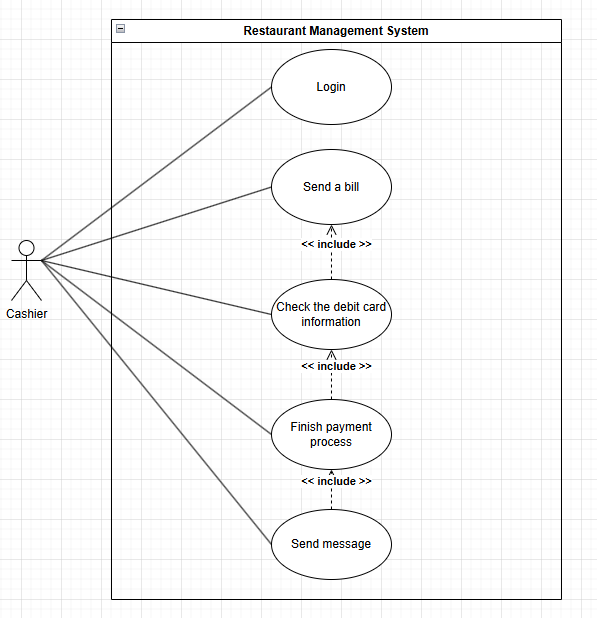
* The Admin manages the restaurant's information, prices, and staff records.
* Login: Authenticate with elevated privileges.
* Manage the menu: Updating the list of food offered by the restaurant. The action "Manage the menu" includes both "Add meal" and "Remove meal." This means that updating the menu is done specifically by performing these two actions.
* Add meal: Putting a new dish onto the menu.
* Remove meal: Taking a dish off the menu.
* Modify the price: Changing how much a meal costs.
* View payments/revenue: Access payment history and dashboards as well as checking how much money the restaurant is making.
* Manage staff: Managing or changing staff information
* Manage tables: Add/edit tables.
* Approve VIP requests: Review and approve/reject VIP requests.

**Actor: Cook**

****

* Login: Authenticate.
* Plan orders schedule: The cook organizes the sequence of cooking tasks based on priority.
* Enter time for orders: The action of inputting specific preparation durations for each dish.
* The relationship between these two is an «include» dependency. This means that "Planning the schedule" is incomplete without "Entering time." The system enforces that the timing data must be captured as a mandatory step within the scheduling process.

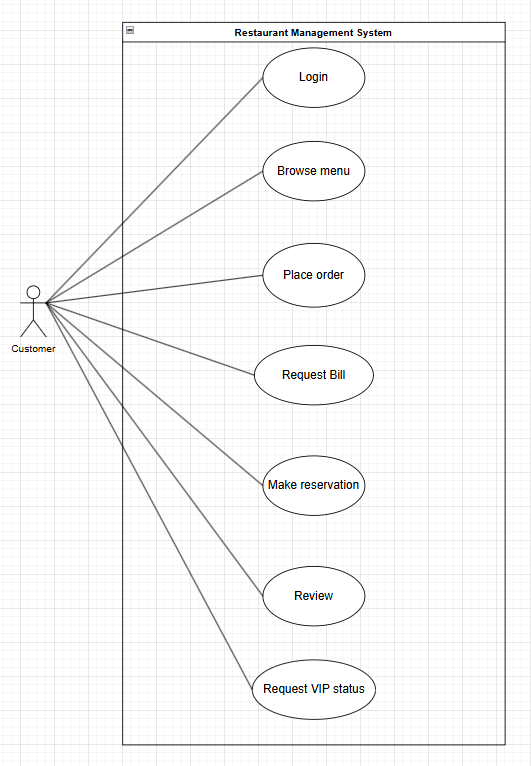
**Actor: Cashier**

****

The Cashier handles the final billing and payment verification

* Login: Authenticate
* Send a bill: Presenting the total amount to the customer.
* Check the debit card information: Verifying if the payment card is valid.
* Finish payment process: Mark order as paid after bill request.
* Send message: Sending a digital receipt or notification to the customer.
* These relationships show required steps. To "Send a bill," the Cashier must "Check the debit card info." To "Finish the payment process," the system must automatically "Send a message" to complete the task.

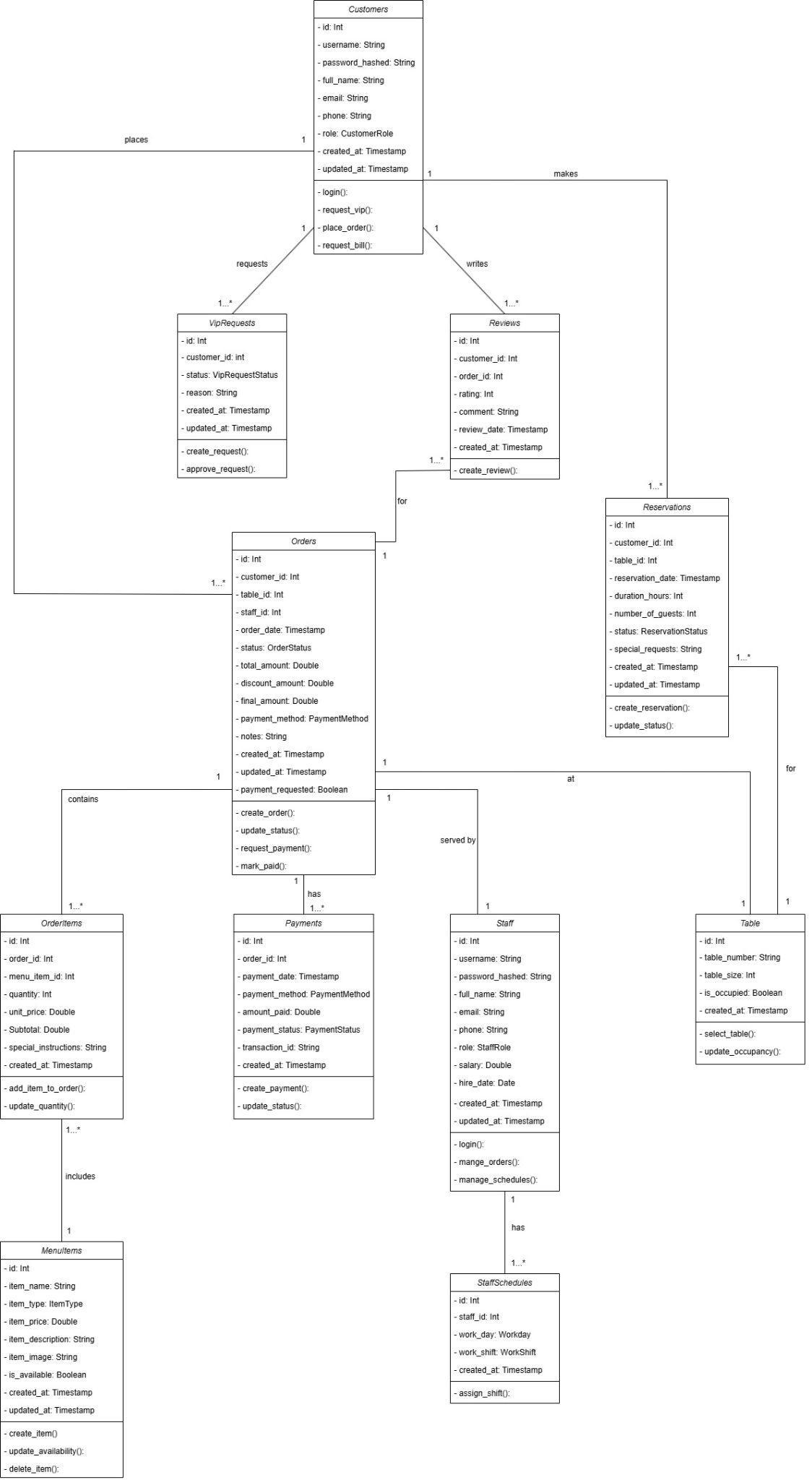
**Actor: Customer**

****

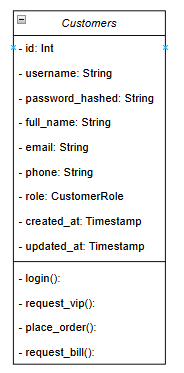
* Login: Authenticate to access personal features.
* Browse Menu: View available menu items.
* Place Order: Select items, choose table, create order.
* Request Bill: Select payment method and request payment.
* Make Reservation: Book a table for a specific date/time.
* Leave Review: Rate and comment on an order after completion.
* Request VIP Status: Submit request if spending threshold met.

**2. Design**

**Class Diagram**

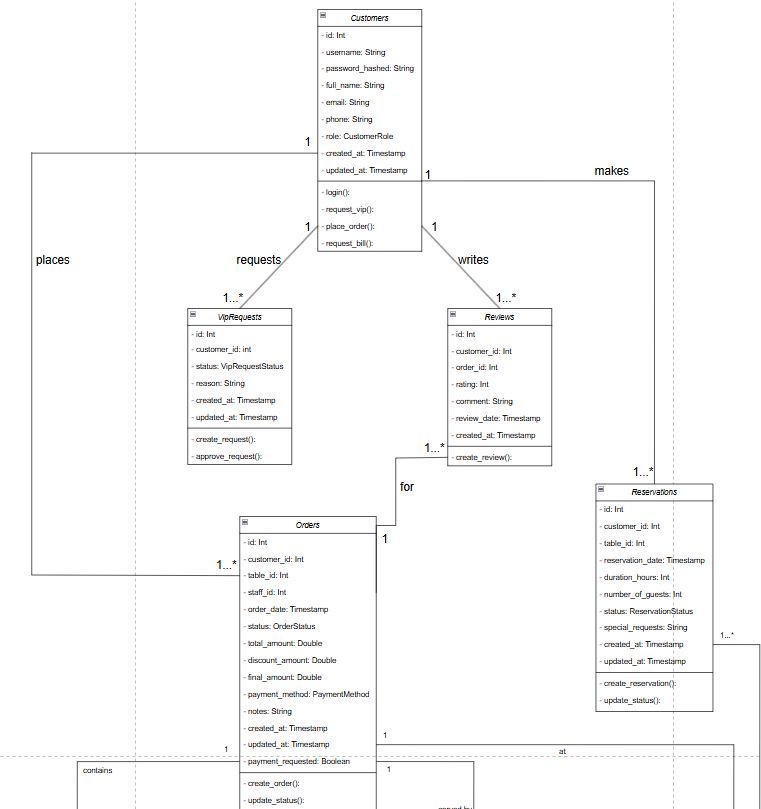
****

**Customer class**

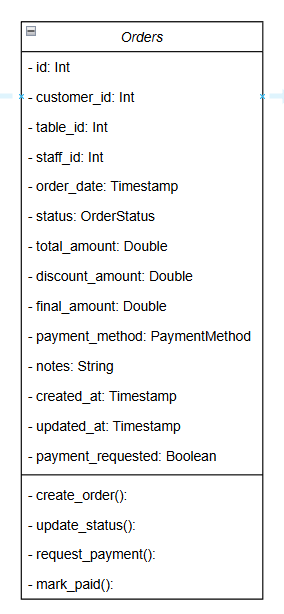
****

The Customer class models individuals who use the restaurant services.

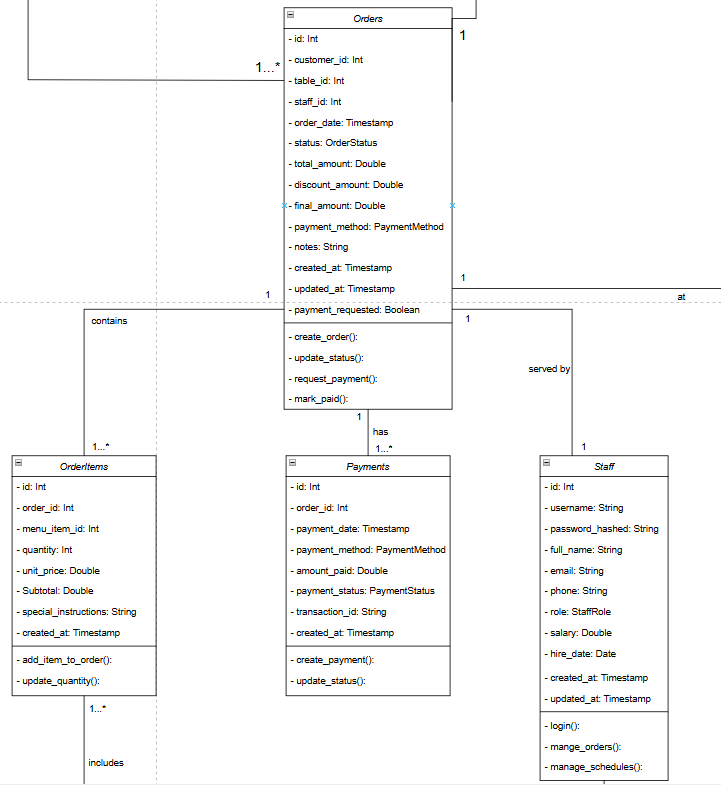
* Can initiate order and payment
* This class provides operations for browsing the menu, placing orders
* Association (Has-a) relationship with Orders class
* Association (Has-a) relationship with Reservations class
* Association (Has-a) relationship with Reviews class
* Association (Has-a) relationship with VipRequests class



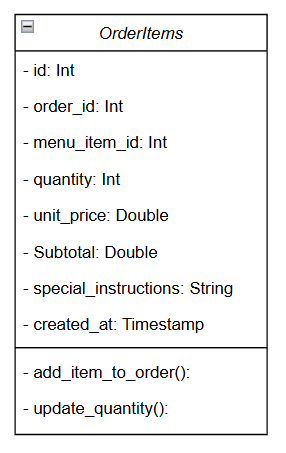
**Order class**

****

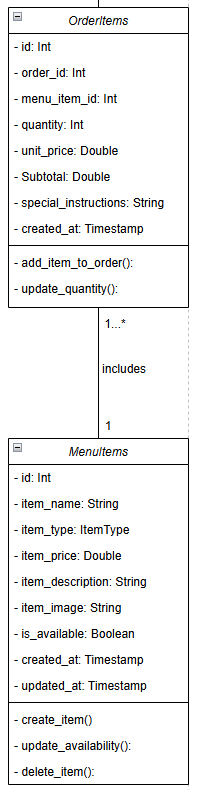
* The Order class represents a customer's order placed at the restaurant.
* It defines attributes such as order ID, order date/time, status (Enum: "Pending", "Preparing", "Ready", "Served", "Completed"), total amount, and special instructions.
* This class manages the lifecycle of an order from placement to completion and payment.
* Association (Has-a) relationship with Customer class
* Association (Has-a) relationship with Table class
* Association (Has-a) relationship with Staff class (served by)
* Composition (Part-of) relationship with OrderItems class
* Composition (Part-of) relationship with Payments class
* Association (Has-a) relationship with Review class

****

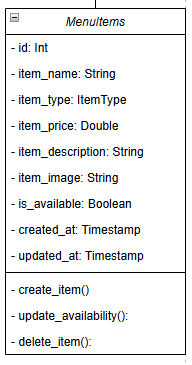
**OrderItems class**

****

* The OrderItem class represents individual items within an order.
* It defines attributes such as order item ID, quantity, unit price, subtotal, and special requests.
* This class links menu items to orders and calculates line item totals.
* Association (Has-a) relationship with Order class
* Association (Has-a) relationship with MenuItem class

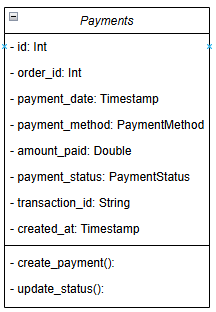
****

**MenuItem class**



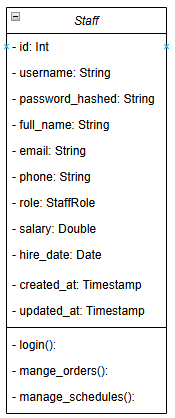
* The MenuItem class represents dishes and beverages available in the restaurant.
* It defines attributes such as item ID, name, description, category (Enum: "Appetizer", "Main Course", "Dessert", "Beverage"), price, availability status, and ingredients.
* This class provides operations for menu display and availability management.
* Admin role exclusive: add, update, delete menu items
* Association (Has-a) relationship with OrderItems class

**Payment class**

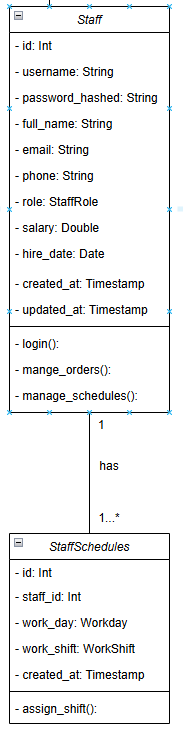


* The Payment class represents financial transactions for orders.
* It defines attributes such as payment ID, amount, payment method (Enum: "Cash", "Credit Card", "Debit Card", "Mobile Payment"), transaction date/time, and status (Enum: "Pending", "Completed", "Failed", "Refunded").
* This class handles payment processing and generates payment receipts.
* Composition (Part-of) relationship with Order class

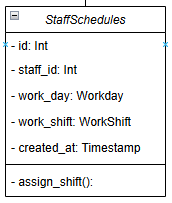
**Staff class**



* The Staff class acts as the superclass for all employee-related roles in the system.
* It defines common attributes such as staff ID, name, and role( Enum(type = String):”Cashier”,”Chef”,”Admin”,”waiter” ), schedule, which are shared by all types of staff members.
* In addition, this class provides general operations for meal management, order planning, order status, payment handling, and internal communication.
* Admin role exclusive : add table, manage schedule, manage menu
* Association (Has -a ) relationship with Staff schedule class
* Compositon(part of) relationship with order class

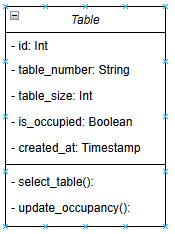


**StaffSchedules class**

****

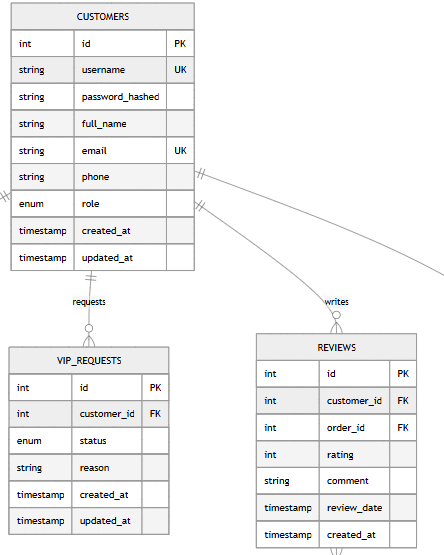
* A class that define schedule for staff
* Association (Has-a) relationship with Staff class

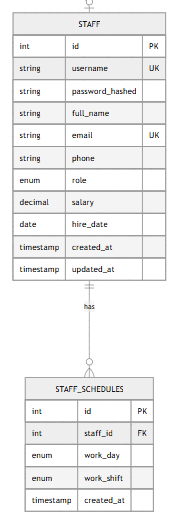
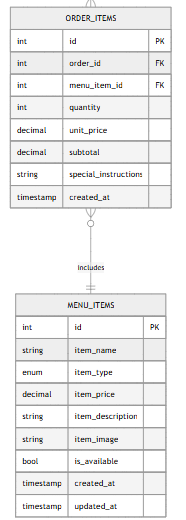
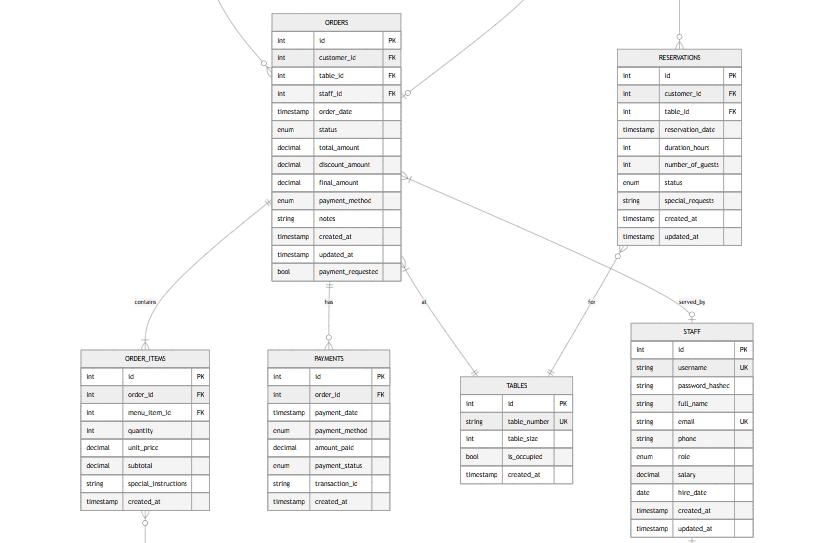
**Table class**

****

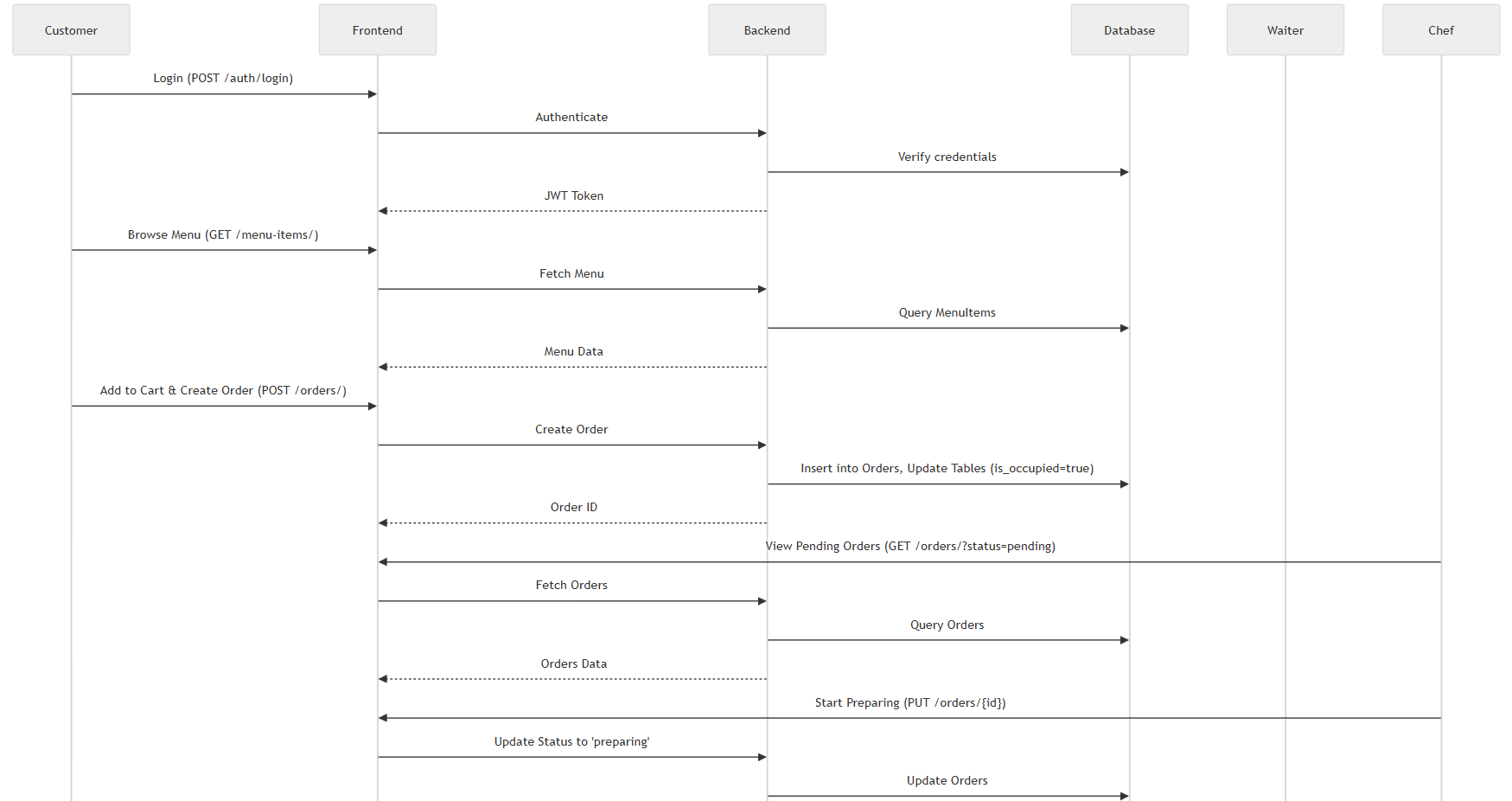
* The Table class represents physical tables in the restaurant.
* It defines attributes such as table ID, table number, seating capacity, location/section, and status (Enum: "Available", "Occupied", "Reserved").
* This class manages table allocation and availability for orders and reservations.
* Admin role exclusive: add, update, delete tables
* Association (Has-a) relationship with Orders class
* Association (Has-a) relationship with Reservations class

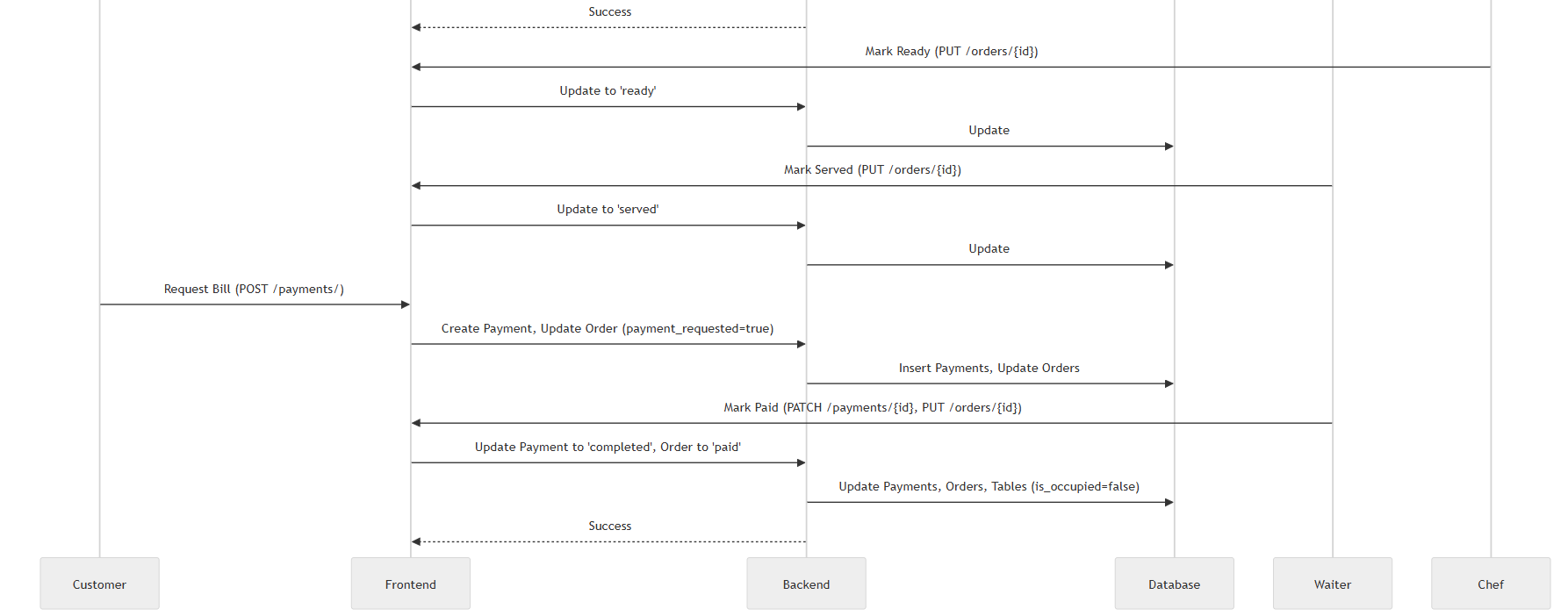
**Entity-Relationship Diagram (ERD)**





**Sequence Diagram**

****

****

**V. Workflow**

**1. Customer Order Flow**

* Customer logs in, browses menu, adds to cart, selects table, creates order.
* Order status: pending → preparing (Chef) → ready (Chef) → served (Waiter).
* Customer requests bill (selects payment method), creates payment record.
* Waiter marks as paid, updates order/table status.

**2. Kitchen Order Management (Cook)**

* Chef logs in, views pending orders.
* Updates to preparing, then ready (notifies Waiter).

**3. Waiter Order Management**

* Waiter logs in, selects table, creates/adds to order.
* Monitors orders, marks served when ready.
* Processes payment when requested.

**4. VIP Status Request Flow**

* Customer checks spending (≥ $500), requests VIP.
* Admin views pending requests, approves/rejects (updates customer role).

**5. Admin Dashboard & Management**

* Dashboard shows revenue, orders, etc.
* Modules for managing orders, menu, staff, schedules, tables, payments, customers/VIP.

**6. Payment Workflow Detail**

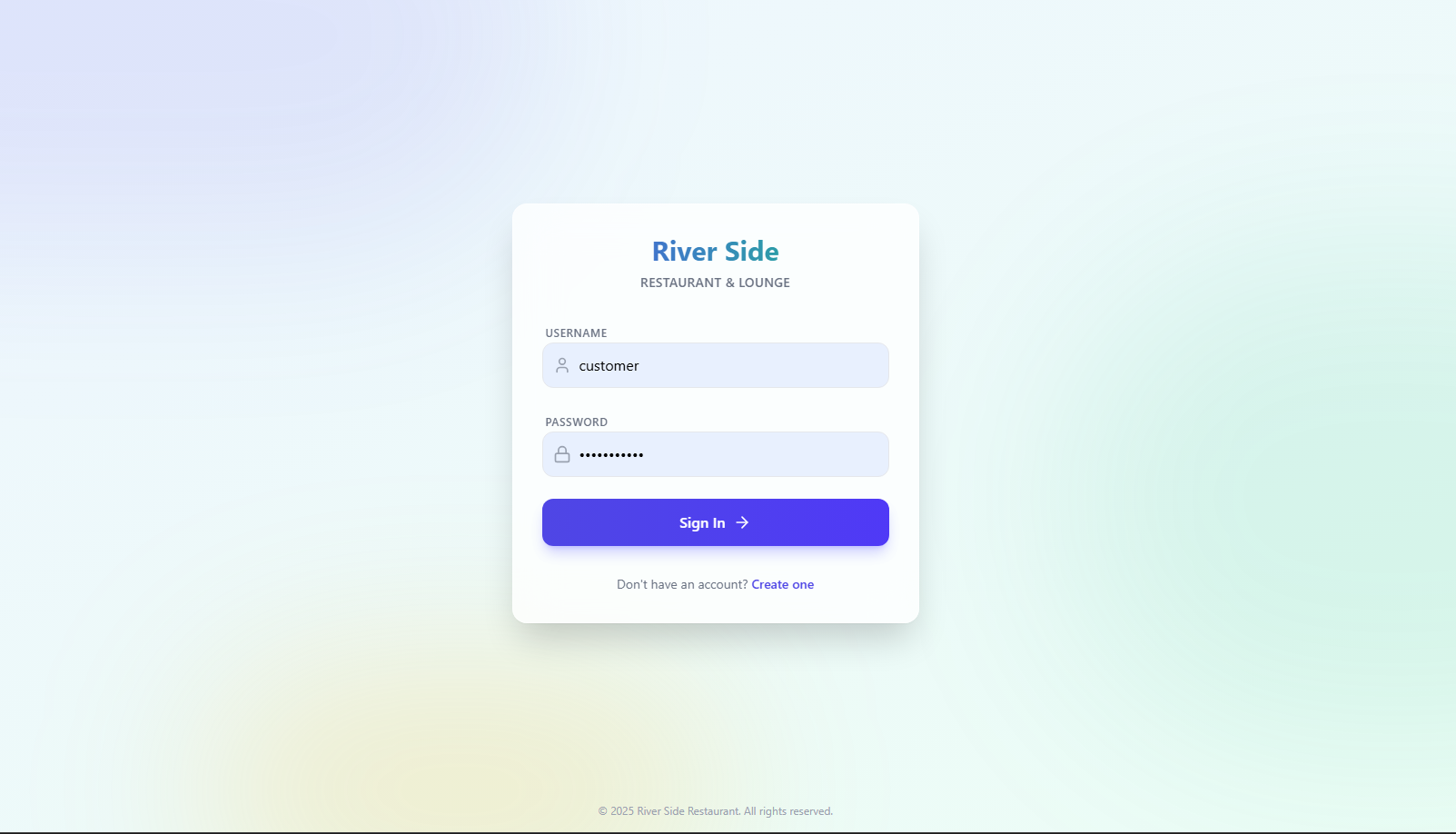
* Customer requests bill: Creates pending payment, sets payment\_requested=true.
* Waiter marks paid: Updates payment to completed, order to paid, frees table.

**7. Order Status Lifecycle**

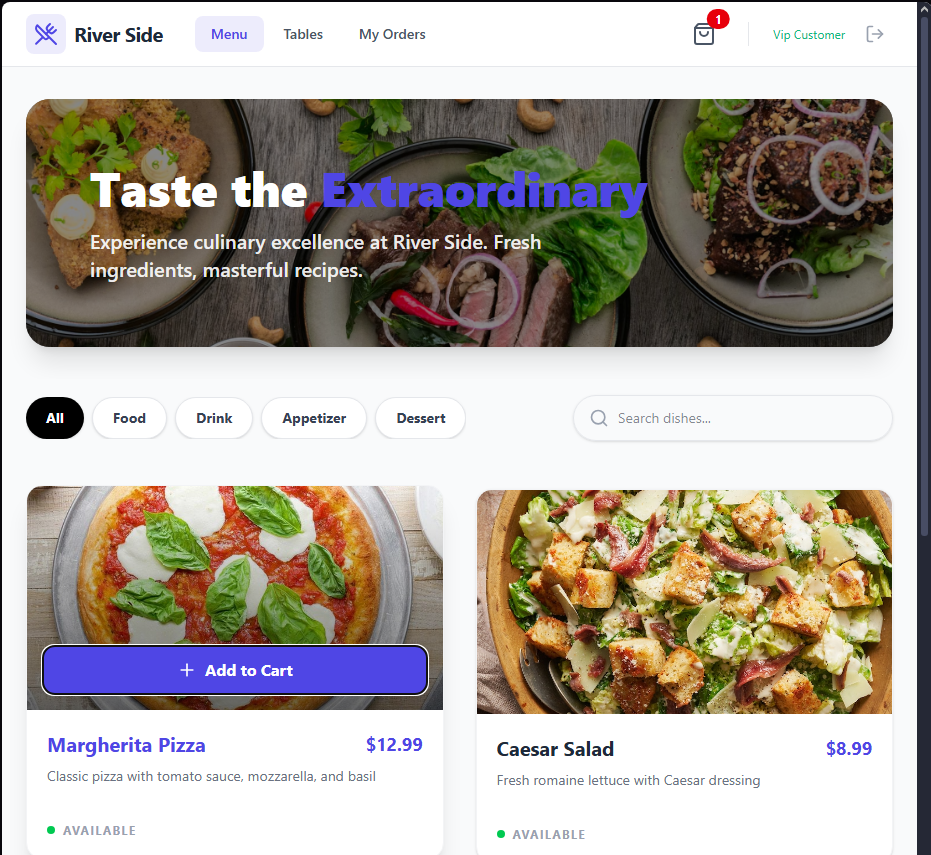
States: pending, preparing, ready, served, paid, cancelled.

Transitions: Chef (pending→preparing→ready), Waiter (ready→served→paid).

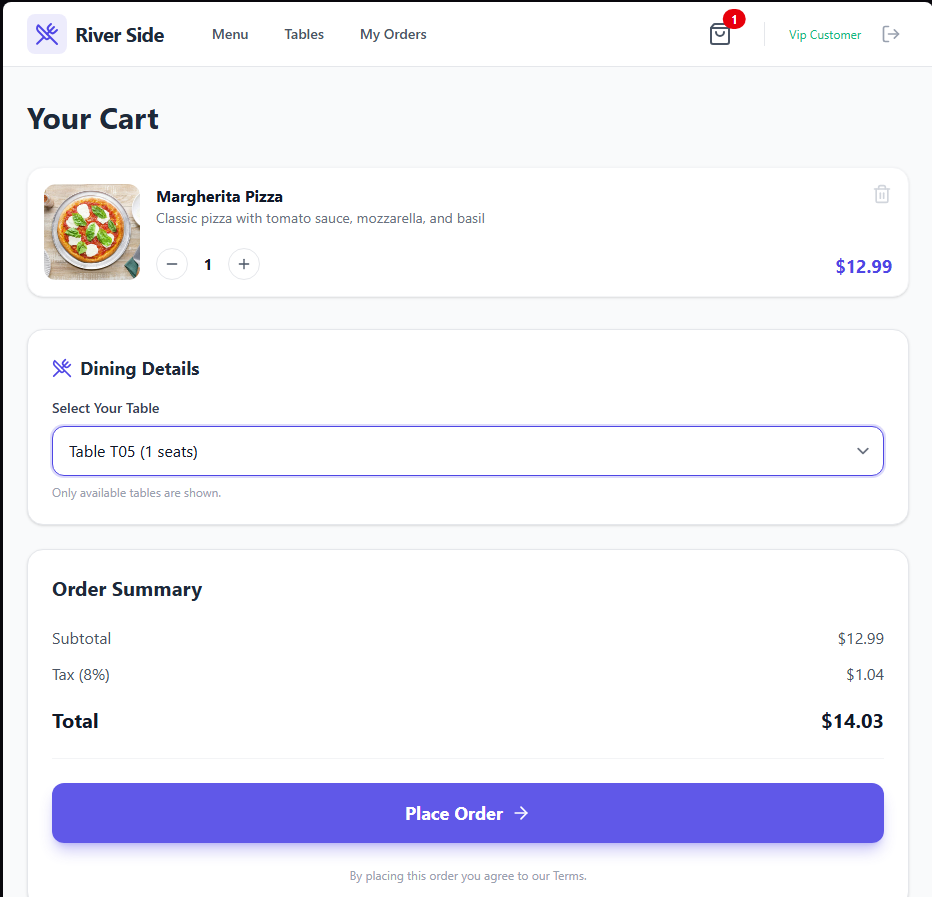
**VI. Project Demo**

****

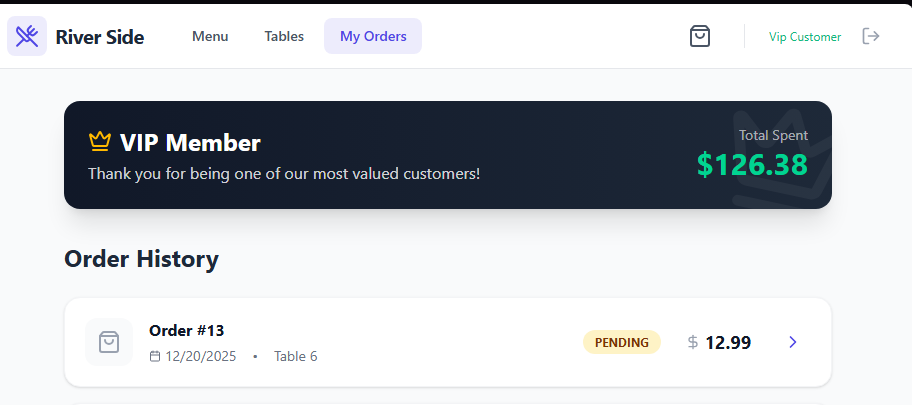
*Figure 5.1 Customer login*

****

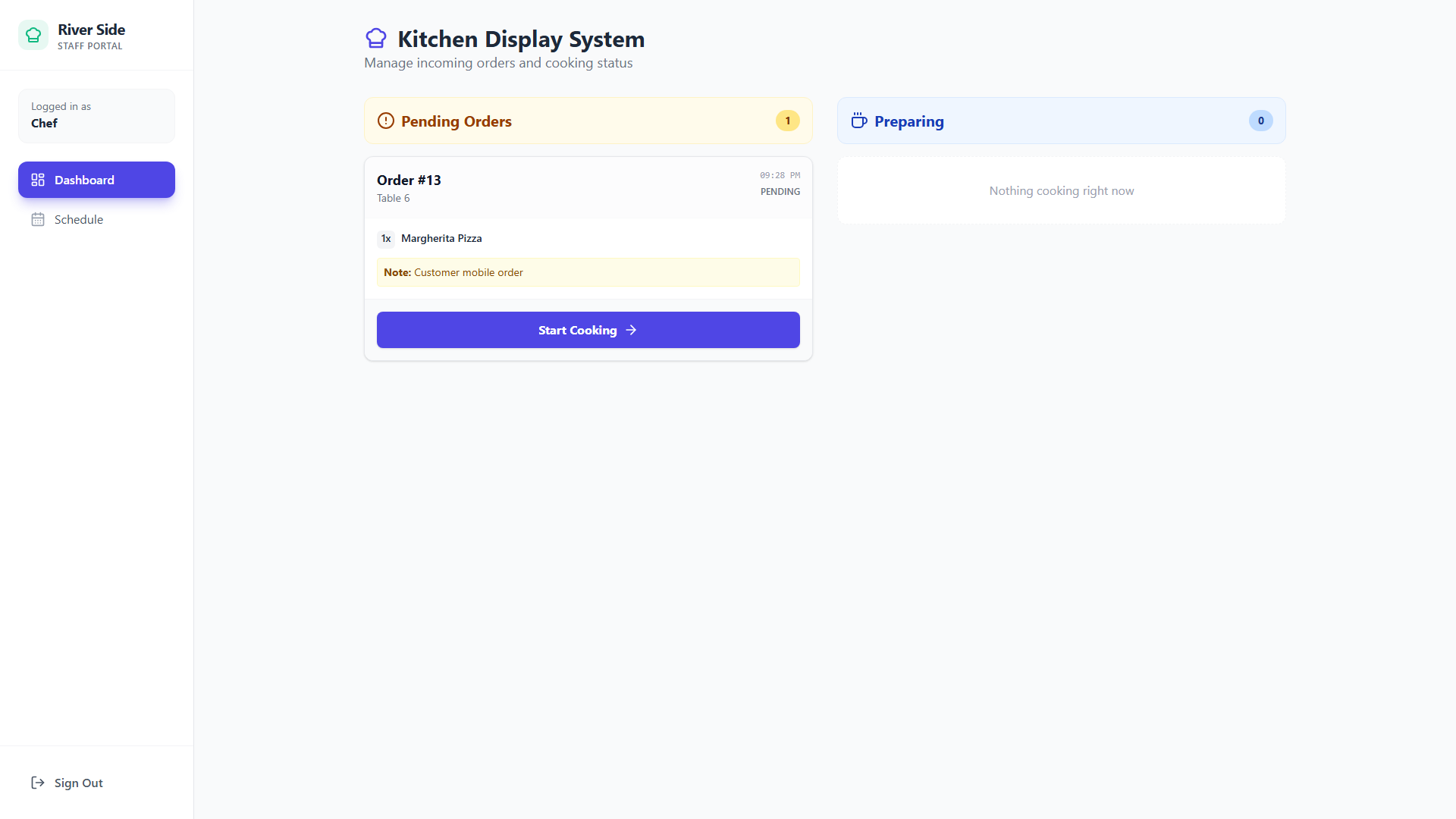
*Figure 5.2 customer browse menu and add item to cart*

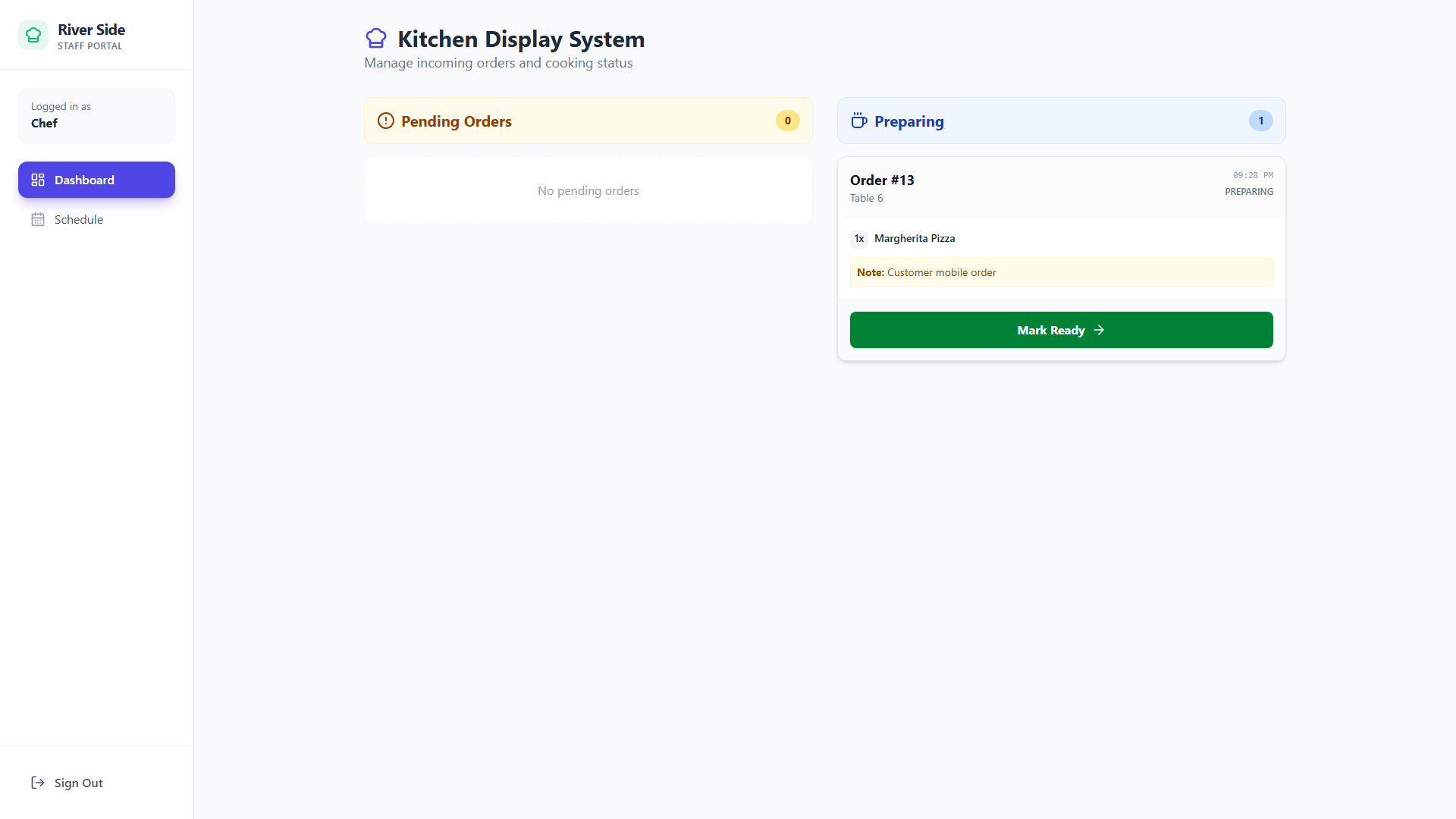
****

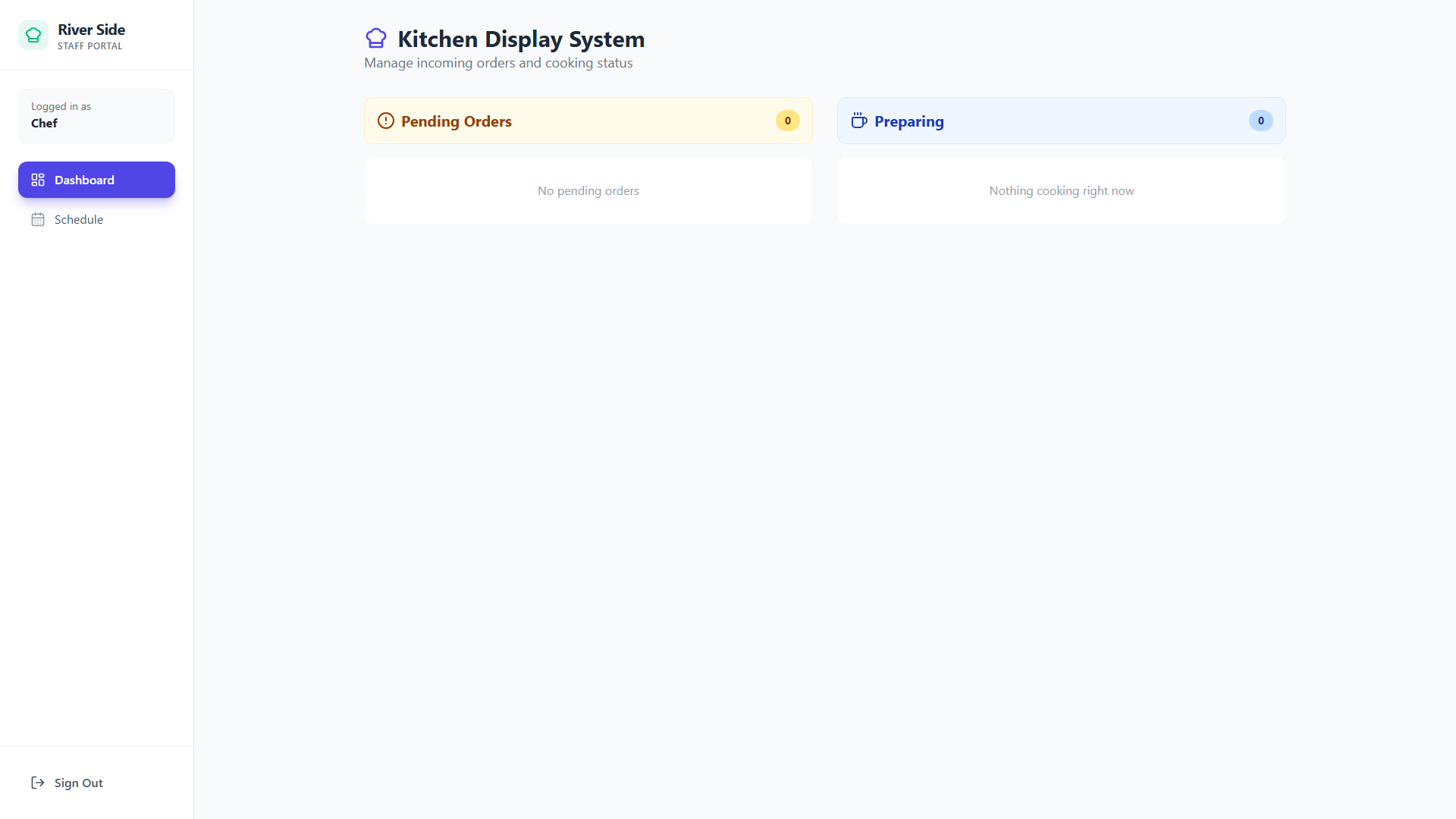
*Figure 5.3 customer select table and create order*

****

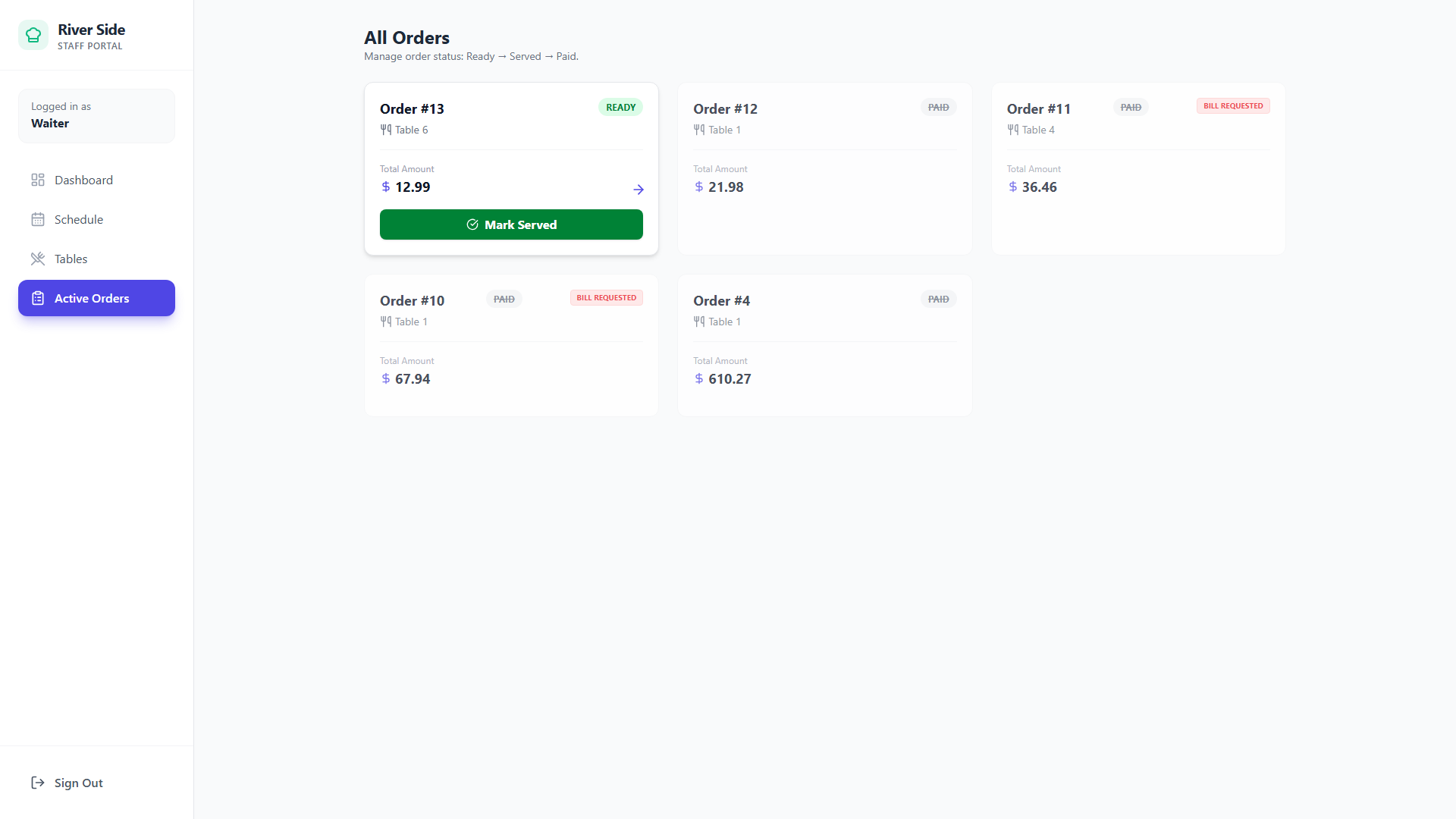
*Figure 5.4 Order created with pending status*

**

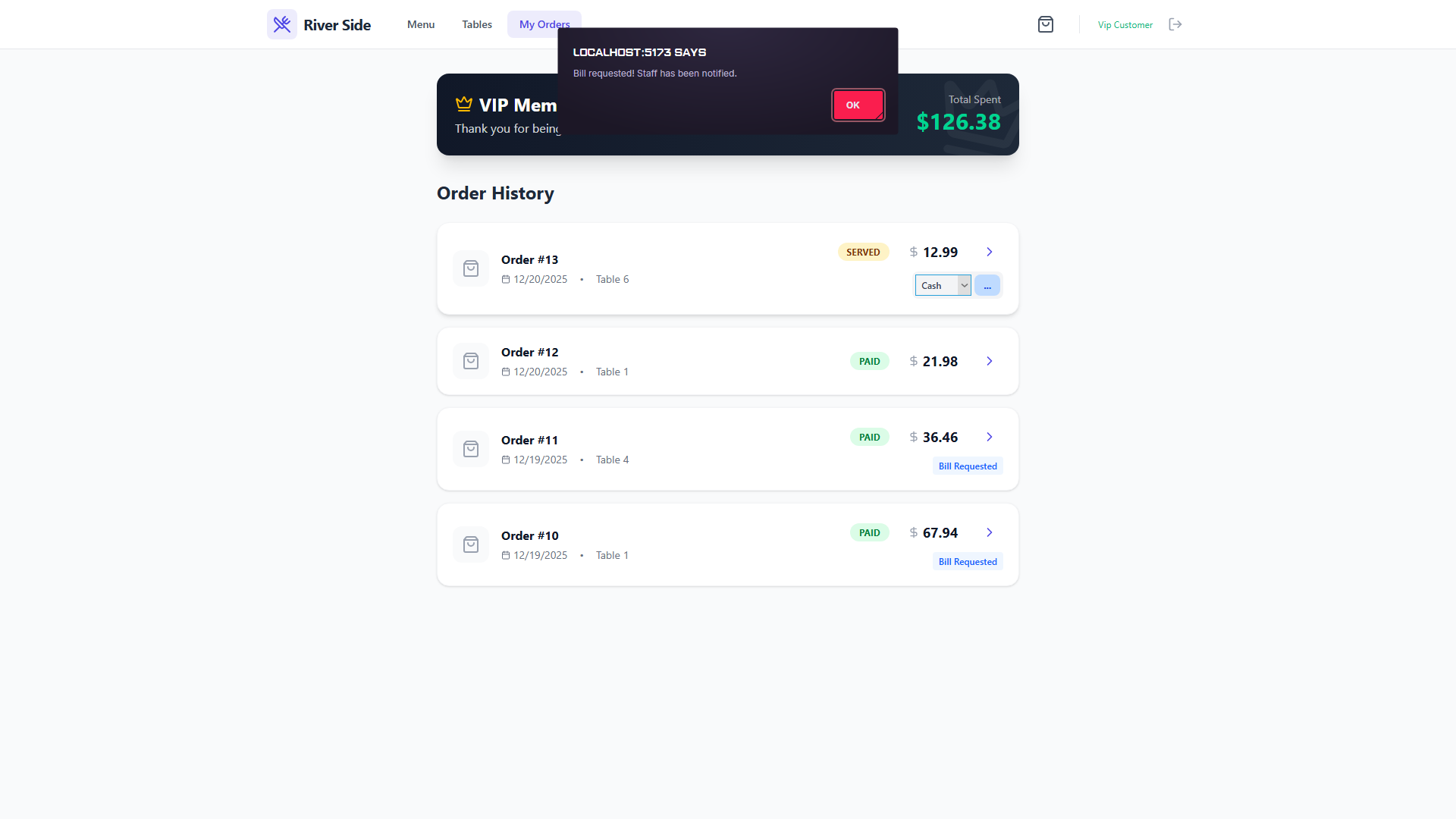
**

**

*Figure 5.4 ,5.5 , 5.6 chef update order status*

**

*Figure 5.7 waiter serves order*

**

*Figure 5.8 customer select payment method, request bill and create payment*

**

*Figure 5.9 waiter/staff update status of payment and order*

**VII. References**

1. FastAPI Documentation: <https://fastapi.tiangolo.com/>

2. ReactJS: <https://reactjs.org/>

3. React Router: <https://reactrouter.com/>

4. Axios: <https://axios-http.com/>

5. TailwindCSS: <https://tailwindcss.com/>

6. Web Application Structure (MVC Model): General reference to Model-View-Controller pattern in web development (e.g., <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>)