Software Design Specifications

for

ReserV8
1.0

Prepared by: Team ReserV8

Document Information

Title: Software Design specifications	Document Version No:1.0
Project Manager: Professor	Document Version Date: 07-04-2025
Prepared By: Team ReserV8	Preparation Date: 27-03-2025

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename
1.0	07-04-2025	ReserV8	Prepared the document after team discussion	Software Design Specifications

Table of Contents

1. Introduction

This Software Design Specification (SDS) document presents a comprehensive design blueprint for **ReserV8**, a web-based multi-purpose reservation system intended for restaurant bookings, university facility scheduling, and meal pre-ordering. This document builds upon the Software Requirements Specification (SRS) and defines how the system's functional and non-functional requirements are realized in design.
The SDS outs the structure and behavior of ReserV8 through use case realizations, design models, data storage perspectives, exception handling mechanisms, and quality of service requirements. It serves as a guide for developers during implementation, testers for validation, and project managers for planning and monitoring.

The document is structured into the following major sections:

- **Section 1: Introduction** – Purpose, scope, key terms, and references.
- **Section 2: Use Case View** – Central system use cases with supporting diagrams.
- **Section 3: Design Overview** – Goals, assumptions, and system decomposition.
- **Section 4: Logical View** – Class and module-level design with use case realizations.
- **Section 5: Data View** – Persistent data model and domain relationships.
- **Section 6: Exception Handling** – Errors and system responses.
- **Section 7: Configurable Parameters** – Tunable system-level settings.
- **Section 8: Quality of Service** – Availability, security, performance, and monitoring.

This introduction sets the stage for the detailed software design that follows, ensuring each component is aligned with ReserV8's goals of modularity, scalability, user-friendliness, and maintainability.

1.1 Purpose

The purpose of this Software Design Specification (SDS) is to describe the high-level and detailed design of the **ReserV8** system, including its structure, components, interactions, and implementation strategies. It translates the functional and non-functional requirements outlined in the Software Requirements Specification (SRS) into a concrete technical design.

This document is intended for the following audiences:

- **Developers** – to guide system implementation based on the described architecture, components, and interfaces.
- **Testers** – to validate the system against use case realizations and design constraints.

- **Project Managers** – to ensure design alignment with project goals and to manage timelines, dependencies, and resource allocation.
- **System Architects** – to verify conformance with the overall software engineering principles and modular architecture.
- **Instructors/Evaluators** – to review the design decisions, completeness, and adherence to software engineering methodologies.

The SDS ensures that all team members have a consistent understanding of the system design and can work collaboratively toward its successful development and deployment.

1.2 Scope

This Software Design Specification applies to the complete design of **ReserV8**, a modular and scalable web-based reservation platform. The document outlines the architectural design and component-level details that influence the system's development, deployment, testing, and maintenance.

The scope includes the following design aspects of ReserV8:

- Reservation system for restaurant tables and university facilities.
- Meal pre-ordering functionality integrated with reservation workflows.
- Role-based access controls for users and administrators.
- Real-time availability tracking and conflict resolution.

This SDS serves as a reference point for all design decisions affecting implementation, integration with external components and configuration of core system modules.

1.3 Definitions, Acronyms, and Abbreviations

SRS – Software Requirements Specification: outlines the functional and non-functional requirements of the system.
SDS – Software Design Specification: this document translates the SRS into a detailed technical design.
UI – User Interface: the front-end interface through which users interact with ReserV8.
UX – User Experience: the overall experience a user has when interacting with the system.
DB – Database: structured storage used to persist system data such as reservations, users, and menus.
RBAC – Role-Based Access Control: permission management system distinguishing between user roles (e.g., Admin, Customer).

CRUD – Create, Read, Update, delete: basic operations performed on data entities.
Module – A self-contained component of the software performing specific functionality, e.g., Reservation Module.

1.4 References

The following documents and resources have been referred to in the preparation of this Software Design Specification:

- ReserV8 Software Requirements Specification (SRS) Document
- IEEE Standard for Software Design Descriptions (IEEE 1016-2009)
- Course Materials from Software Engineering (Spring 2025)
- UI/UX Guidelines for Web Applications
- Database Design Best Practices

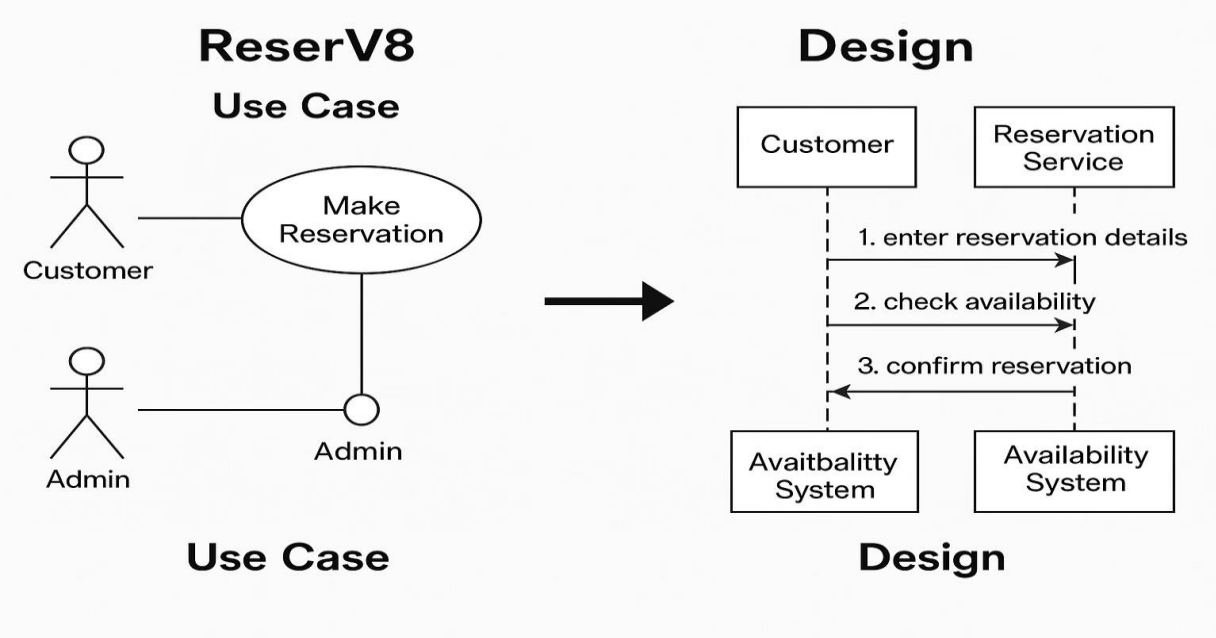
2. Use Case View

Based on the Software Requirements Specification for ReserV8, the following use cases represent significant and central functionality of the system. These use cases cover core features that define the system's design, such as managing reservations, handling availability, processing meal orders, and booking university facilities. Each of these use cases exercises multiple system modules and helps to illustrate both normal and exceptional design flows.

The following key use cases are addressed in this design:

- Reserve a Restaurant Table
- Manage Restaurant Availability
- Cancel a Reservation
- Pre-Book Meals and Pay
- Reserve a University Facility

Each use case has been selected because of its extensive coverage of backend logic, user interactions, role-based permissions, and data persistence mechanisms.



2.1 Use Case

2.1.1 Use Case: Reserve a Restaurant Table

Actors: User, System
Brief Description: This use case allows a user to log in, view nearby restaurants, and reserve a table for a selected date and time.

Usage Steps:

1. User logs into the system.
2. User selects the “Reserve Table” option.
3. System presents a list of restaurants.
4. User selects a restaurant and time slot.
5. System checks for availability and confirms the reservation.
6. System stores the reservation and sends a confirmation to the user.

2.1.2 Use Case: Manage Restaurant Availability

Actors: Admin, System
Brief Description: This use case allows an admin to manage the availability of their restaurant by modifying available booking slots or blocking out specific dates.

Usage Steps:

1. Admin logs into the system.
2. Admin accesses the availability dashboard.
3. Admin selects a restaurant and modifies availability.
4. System validates the input.
5. System updates the database.
6. The system notifies users if existing reservations are affected.

2.1.3 Use Case: Cancel a Reservation

Actors: User, System, Admin
Brief Description: This use case allows a user to cancel a confirmed reservation and receive a refund if applicable.

Usage Steps:

1. User logs in and navigates to their reservation list.
2. User selects a reservation to cancel.
3. System checks if the reservation is eligible for a refund.
4. System cancels the reservation and updates availability.
5. The system sends notifications to both the user and admin.
6. System initiates refund processing, if applicable.

2.1.4 Use Case: Pre-Book Meals and Pay

Actors: User, System, Restaurant Admin
Brief Description: This use case allows a user to pre-select meals while reserving a table and complete payment securely.

Usage Steps:

1. User selects a restaurant and reserves a table.
2. User browses the menu and selects items.
3. System prompts for payment.
4. User completes the payment.
5. System stores the pre-order with the reservation.
6. System notifies the restaurant about the order.

2.1.5 Use Case: Reserve a University Facility

Actors: User, System, Facility Admin
Brief Description: This use case allows a user to book university facilities such as classrooms or sports halls.

Usage Steps:

1. User logs into the system.
2. The user opens the facility booking section.
3. The system displays a list of available facilities and time slots.
4. User selects a facility and time slot.
5. System checks availability and confirms the booking.
6. The system stores the reservation and sends a confirmation.

3. Design Overview

The design of **ReserV8** is based on a modular, multi-tier architecture that supports role-based access, scalability, and ease of maintenance. It translates the requirements from the SRS into implementable components organized across three layers: presentation (UI), application logic (controllers/services), and data (database).

ReserV8 supports two primary roles: **Users** (restaurant customers or university students) and **Admins** (restaurant/facility managers). Each role accesses system features via dedicated modules.

Key architectural principles include:

- Loose Coupling and High Cohesion for easy maintenance.
- Reusability of shared services like authentication and session management.
- Extensibility to support new features with minimal redesign.

Major System Modules:

- User Module – authentication and session handling
- Reservation Module – create, update, cancel reservations
- Meal Preorder Module – manage menus and attach orders to bookings
- Admin Dashboard – manage restaurant/facility availability

The following sections elaborate on design goals, use case realizations, data structures, and quality attributes of the ReserV8 system.

3.1 Design Goals and Constraints

Design Goals:

- **Modularity:** Each major function (e.g., reservation, meal pre-ordering) is implemented as a separate module to ensure clarity and maintainability.
- **Scalability:** The system should support a growing user base and potential expansion to additional domains (e.g., sports facilities).
- **Simplicity and Usability:** Interfaces and workflows are designed for minimal user effort, especially for first-time users.
- **Maintainability:** Codebase should be organized to support easy debugging, testing, and future updates.

Constraints:

- **Technology Stack:** The system is implemented using HTML/CSS, JavaScript, React for frontend, and MySQL database.
- **Limited Team Size:** The system is built by a small development team, which impacts the number of features implemented within the project timeline.
- **Academic Scope:** Certain features (e.g., complex payment integration, real-time notifications) are excluded due to project constraints.
- **Static Data Models:** Menu items and facilities are managed manually via admin inputs rather than dynamic syncing from external sources.

3.2 Design Assumptions

- Users will access the system through modern web browsers that support JavaScript and responsive layouts.
- Internet connectivity is required at all times to perform core functions such as booking, updating availability, and accessing menus.
- Admin users (restaurant/facility managers) will manually input and update availability and menu data through dedicated interfaces.
- The database is assumed to handle concurrent read/write operations without requiring manual conflict resolution.
- Each user will have a unique authenticated session, and session handling will be managed at the application level.

3.3 Significant Design Packages

The ReserV8 system follows a full-stack modular architecture, divided into frontend and backend packages. The design emphasizes scalability, separation of concerns, and maintainability.

Frontend - (React + Vite + TypeScript):

The frontend is developed in **React** using **TypeScript**, bundled with **Vite**, and styled using **Tailwind CSS**. It features a **page-based routing structure** and reusable components, enabling a clean and responsive user experience.

Key Pages:

- LandingPage – Main entry point; displays login modal and navigation to dining/facility views
- DiningPage – Fetches restaurant data and displays restaurant cards with filters
- RestaurantPage – Full reservation UI with booking form and customer reviews
- FacilitiesPage, CourtBookingPage, RoomBookingPage – Placeholder and booking interfaces for university facilities
- LoginPage, Signup – Authentication interfaces

Components:

- Header, Footer – Navigation and layout wrappers
- RestaurantCard, RestaurantHeader – Restaurant listing and details display

Structure Highlights:

- Routing managed with react-router-dom
- Real-time fetch from backend (/restaurants)
- Filter logic for sorting by rating, location, etc.

Backend - (Node.js + Express + MySQL):

The backend serves as the API layer using Express.js and MySQL. It supports user registration, login, and restaurant/facility data operations.

Modules and Responsibilities:

- **server.js, start-server.js:** Initializes Express server and manages shutdown events
- **Routes**: /register, /login, /restaurants, etc.
- **Database**: User and restaurant data is stored and queried

3.4 Dependent External Interfaces

The table below lists the public interfaces this design requires from other modules or applications.

External Application and Interface Name	Module Using the Interface	Functionality / Description
MySQL Database	Reservation Module, User Module	Executes queries to store, retrieve, and update data for users, restaurants, and reservations.
REST API Endpoints (/restaurants, /login)	React Frontend Pages & Components	Enables frontend to fetch restaurant data, handle authentication, and submit reservations.
TailwindCSS	All UI Components and Pages	Applies consistent, responsive, utility-based styling to the frontend interface.
React Icons	RestaurantPage, RestaurantCard	Adds icons for ratings, pricing, and location to enhance the user experience.

3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces that the ReserV8 system makes available for other components (such as the frontend application) to consume.

Interface Name	Module Implementing the Interface	Functionality / Description

/login (POST)	Auth Controller (backend)	Authenticates users using provided credentials and responds with login status.
/register (POST)	Auth Controller (backend)	Registers a new user by inserting details into the MySQL database.
/restaurants (GET)	Restaurant Controller (backend)	Returns a list of restaurants for the frontend to display in dining search pages.
/restaurants/:id (GET)	Restaurant Controller (backend)	Fetches details for a specific restaurant used in the RestaurantPage frontend.
/health (GET)	Server Utility Module (backend)	Returns server status for monitoring uptime or checking live status.

4. Logical View

Logical View presents the detailed internal design of the ReserV8 system. It is organized into layers, starting with high-level module interactions that implement core use cases, and followed by lower-level views showing how individual components and functions collaborate within each module.

This layered approach reflects both the architectural separation (frontend/backend) and internal responsibilities (such as booking, user management, and admin controls). The system uses a model-view-controller (MVC) inspired approach on the backend and component-service routing on the front-end.

Top-Level Module Interaction

At the top level, modules interact to implement the following key use cases:

Use Case	Frontend Component	Backend Endpoint / Module
Reserve a Restaurant Table	RestaurantPage.tsx	POST /reservations → Reservation Handler
Manage Restaurant Availability	AdminDashboard (planned)	PUT /restaurants/:id/availability
Cancel a Reservation	ReservationHistoryPage	DELETE /reservations/:id
Pre-Book Meals and Pay	MenuSelector.tsx (planned)	POST /preorders (future implementation)

Reserve a University Facility	RoomBookingPage.tsx, CourtBookingPage.tsx	POST /facilities/book (future implementation)

Each of these interactions triggers calls from React UI components to Express.js backend routes, which then handle business logic and database transactions.

Layered Design Detail

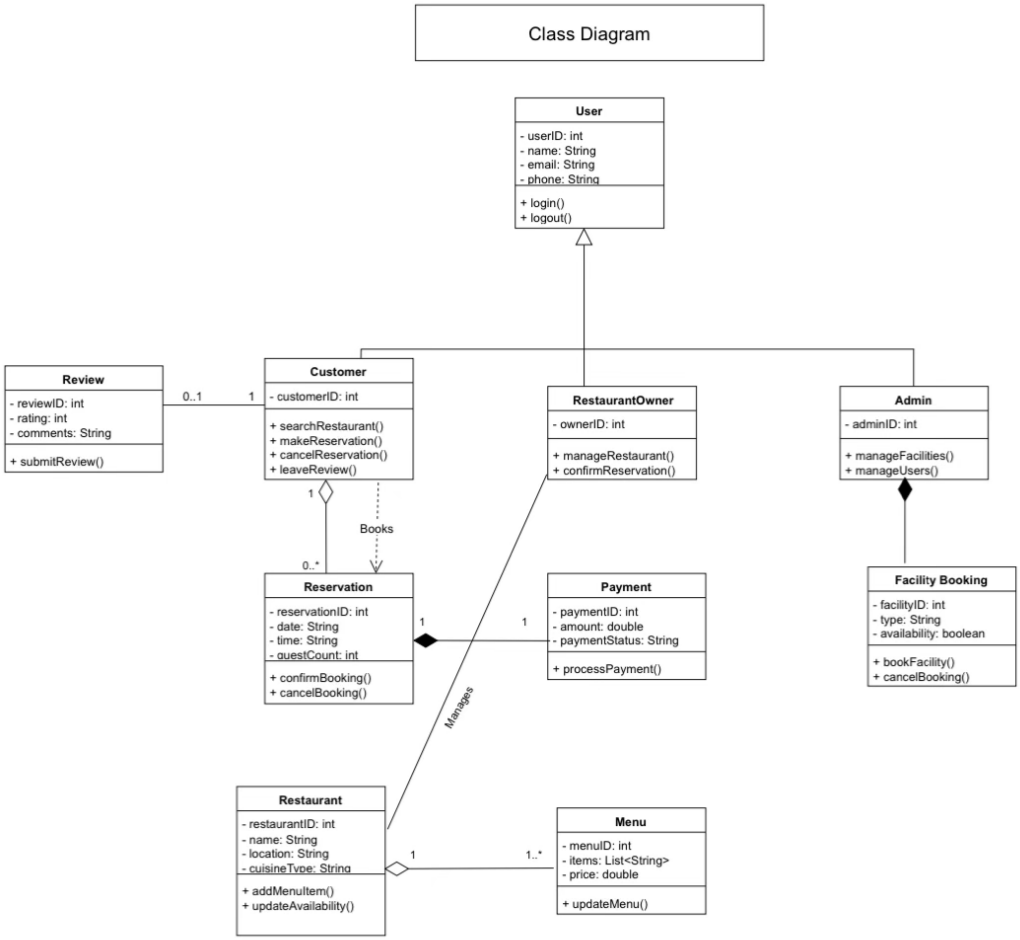
Frontend - (React + Vite):

- **Component Layer:**
  Reusable components like RestaurantCard, RestaurantHeader, Header, and Footer manage UI display logic.
- **Page Layer:**
  Page-level components like DiningPage, RestaurantPage, and RoomBookingPage define route-based views and user flows.
- **Routing Layer:**
  Handles navigation between pages using react-router-dom.
- **State/Auth Context:**
  AuthContext provides global access to login state and user info.

Backend - (Node.js + Express):

- **Routing Layer:**
  Defines endpoints like /login, /register, /restaurants, and /reservations.
- **Controller Layer:**
  Each route maps to a controller function that contains business logic (e.g., validating input, querying DB).
- **Data Access Layer:**
  Uses mysql2 to perform structured CRUD operations on MySQL tables.
- **Startup & Middleware:**
  start-server.js and Express middlewares handle environment setup, parsing, and error management.

4.1 Design Model



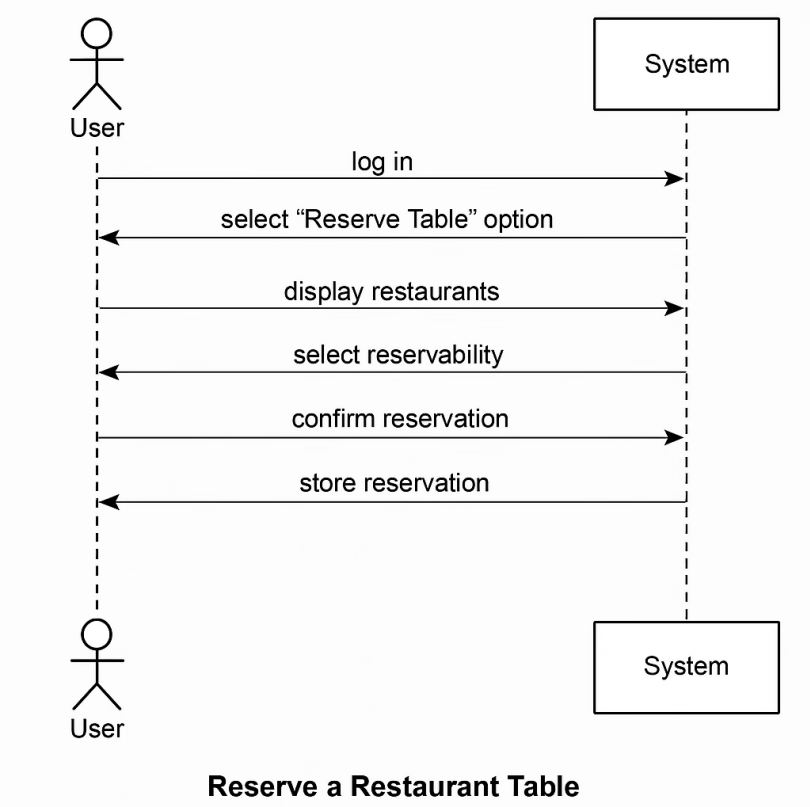
4.2 Use Case Realization

Use Case Realization: Reserve a Restaurant Table

Actors Involved:
User, RestaurantPage (Frontend), Reservation Controller, Reservation Class, Database

Sequence Steps:

1. User opens the RestaurantPage and selects reservation details (restaurant, date, time, guests).
2. The RestaurantPage sends a reservation request to the Reservation Controller (backend).
3. The Reservation Controller creates and validates a Reservation object.
4. The Reservation class inserts the reservation into the Database.
5. The controller sends a confirmation response back to the RestaurantPage.
6. The RestaurantPage displays a success message to the user.



Reserve a Restaurant Table

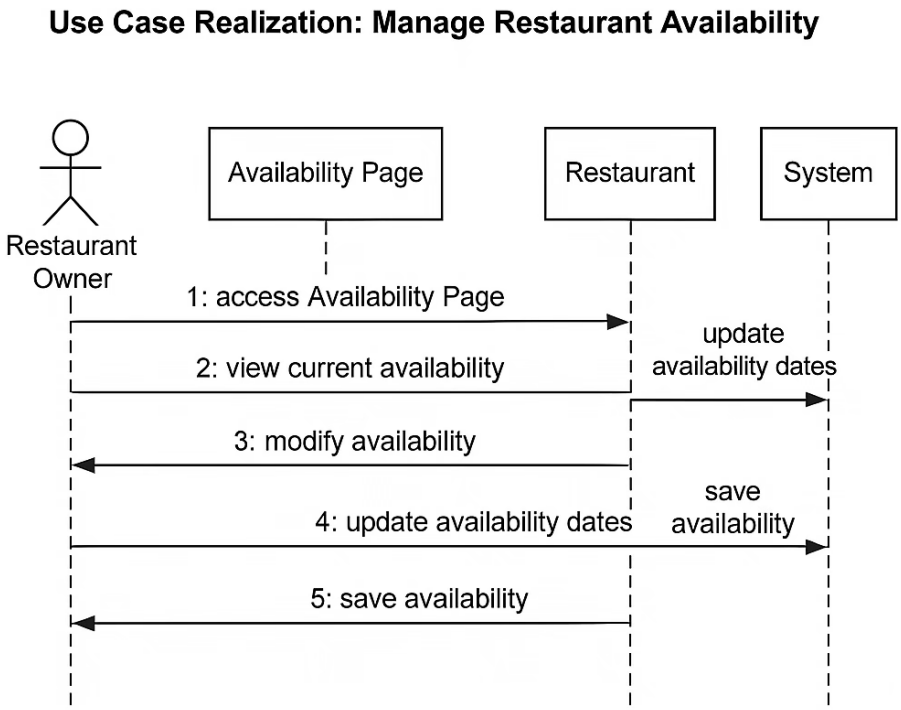
Use Case Realization: Manage Restaurant Availability

Actors Involved:
Restaurant Owner, Availability Page (Frontend), Availability Controller, Restaurant Class, Database

Sequence Steps:

1. The Restaurant Owner logs in and accesses the Availability Page.

2. The Availability Page fetches and displays current availability data.
3. The Restaurant Owner modifies availability (e.g., blocks date, adds slots).
4. The Availability Page sends the updates to the Availability Controller.
5. The controller invokes the Restaurant class to process the changes.
6. The Restaurant class updates the Database.
7. A success confirmation is returned to the Availability Page and shown to the owner.



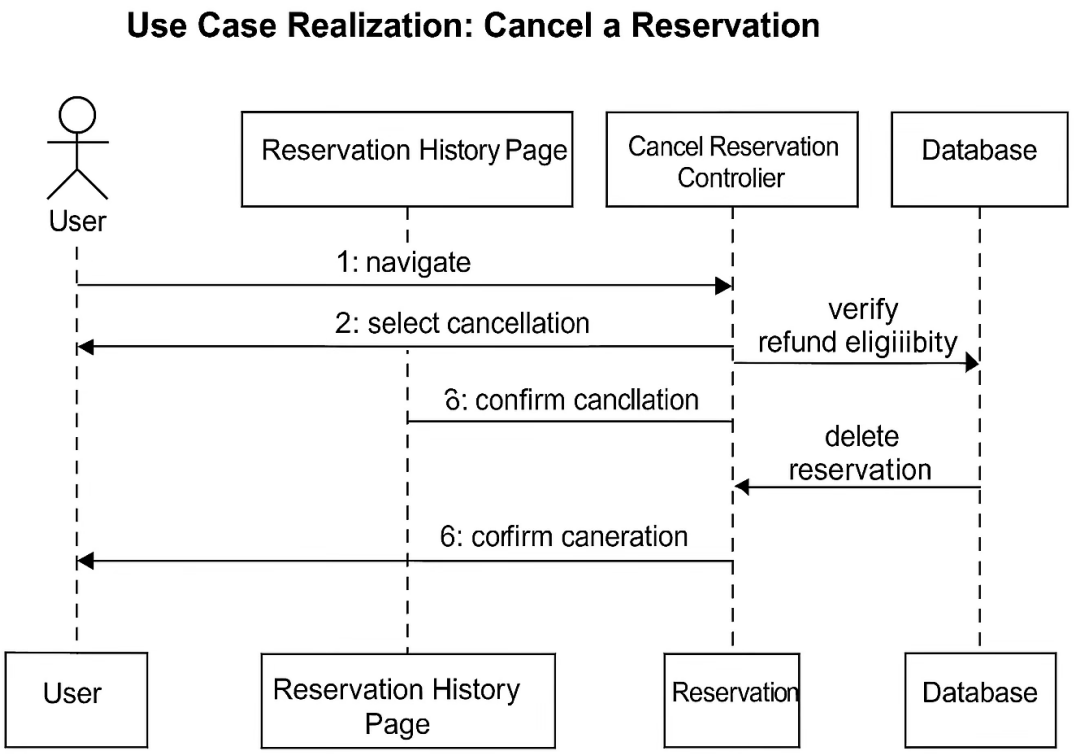
Use Case Realization: Cancel a Reservation

Actors Involved:
User, Reservation History Page (Frontend), Cancel Reservation Controller, Reservation Class, Database

Sequence Steps:

1. User navigates to the Reservation History Page.

2. User selects an existing reservation to cancel.
3. The Reservation History Page sends a DELETE request to the Cancel Reservation Controller.
4. The controller verifies the cancellation policy and checks refund eligibility.
5. The Reservation class processes the cancellation.
6. The system updates the Database to reflect the cancellation.
7. The controller returns a cancellation confirmation to the frontend.
8. The Reservation History Page displays the result to the user.



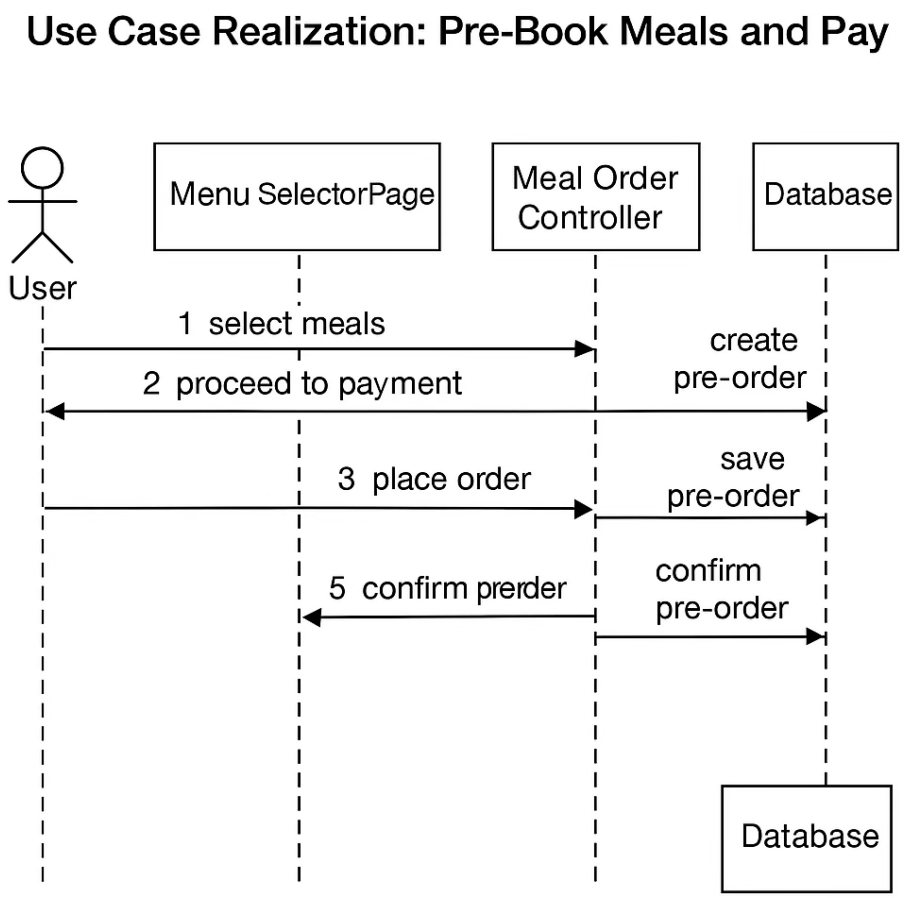
Use Case Realization: Pre-Book Meals and Pay

Actors Involved:
User, Menu Selector Page (Frontend), Meal Order Controller, Meal Class, Database

Sequence Steps:

1. The user selects a restaurant and confirms a table reservation.
2. User navigates to the Menu Selector Page to view the menu.
3. User selects meals and proceeds to checkout.

4. The Menu Selector Page sends the selected items to the Meal Order Controller.
5. The controller calls the Meal class to process the order.
6. The Meal class inserts the meal pre-order into the Database.
7. The controller returns a successful response to the frontend.
8. The frontend displays a confirmation message and links it to the reservation.



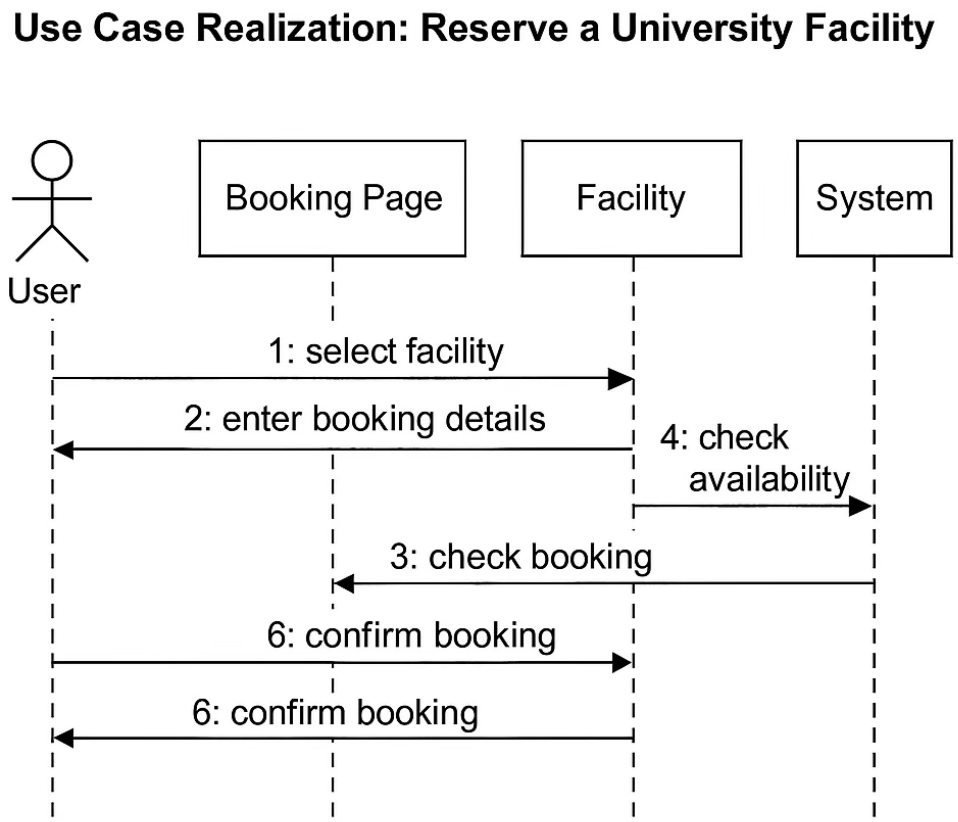
Use Case Realization: Reserve a University Facility

Actors Involved:
User, Facility Booking Page (Frontend), Facility Controller, Facility Class, Database

Sequence Steps:

1. User opens the Facility Booking Page and views a list of rooms or courts.
2. User selects a facility, date, and time slot.
3. The frontend sends a reservation request to the Facility Controller.
4. The controller creates a Facility object and validates slot availability.

5. If available, booking details are stored in the Database.
6. A confirmation message is returned and displayed to the user.



5. Data View

The domain model directly maps to the database design with minimal abstraction, making the transition between the two models trivial. However, we provide a brief outline of the entities and data structures for completeness.

5.1 DOMAIN MODEL

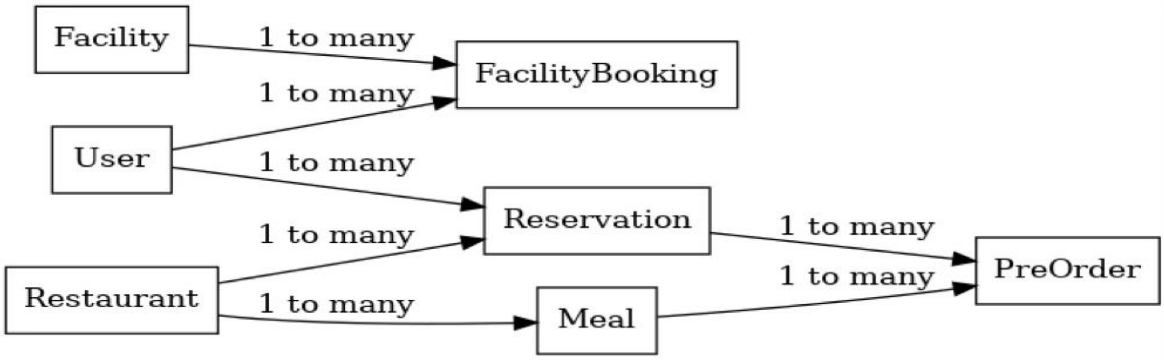
The domain model represents key real-world concepts in the ReserV8 system and the relationships between them. This model helps in translating user requirements into a structured design without focusing on technical implementation.

Entities:

- **User** – Represents any system participant (e.g., customer, admin)
- **Restaurant** – A food service location available for reservation
- **Reservation** – A booking by a user for a specific restaurant at a set time
- **Meal** – A menu item offered by a restaurant
- **PreOrder** – A selected set of meals linked to a reservation
- **Facility** – University facility available for booking
- **FacilityBooking** – A confirmed booking of a facility by a user.

Relationships:

- A User can make multiple Reservations
- A Reservation is associated with one Restaurant
- A Reservation can include multiple PreOrders
- A PreOrder includes one or more Meals
- A User can make multiple FacilityBookings
- Each FacilityBooking links one User and one Facility



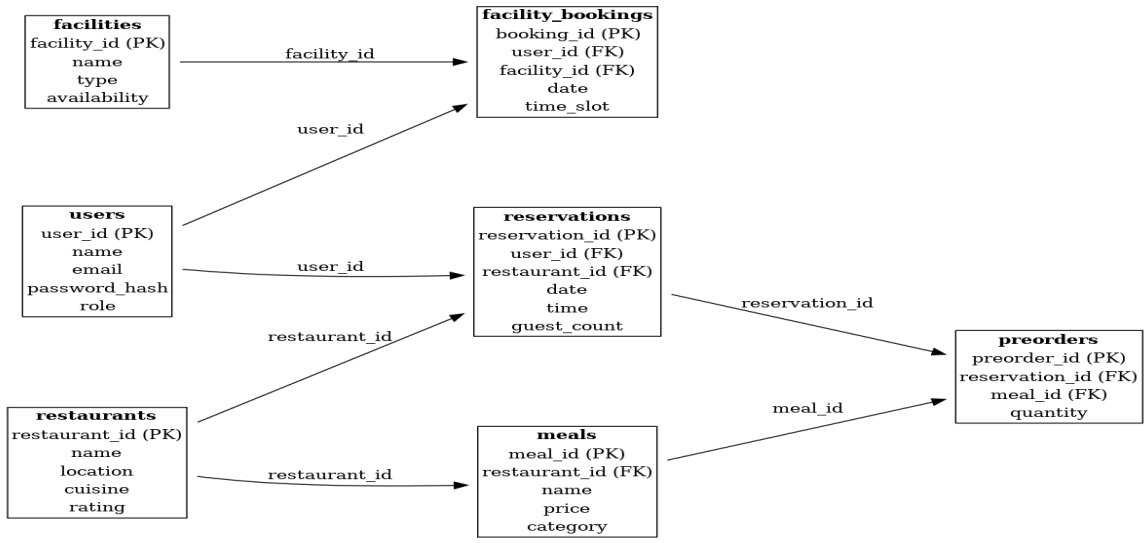
5.2 Data Model (persistent data view)

This is the logical implementation of the domain model in the database schema. It consists of normalized relational tables:

Table	Key Fields	Foreign Keys
users	user_id (PK), name, email, role	—

restaurants	restaurant_id (PK), name, cuisine	—
reservations	reservation_id (PK)	user_id → users, restaurant_id → restaurants
meals	meal_id (PK), name, price	restaurant_id → restaurants
preorders	preorder_id (PK), quantity	reservation_id → reservations, meal_id → meals
facilities	facility_id (PK), name, type	—
facility_bookings	booking_id (PK), date, time_slot	user_id → users, facility_id → facilities

The model is normalized to 3NF, avoids data redundancy, and uses auto-incrementing primary keys.



5.2.1 Data Dictionary

Table	Attribute	Type	Description
users	user_id	INT (PK)	Unique identifier for each user
	name	VARCHAR	User's full name

	email	VARCHAR	User's email address (used for login)
	password_hash	VARCHAR	Hashed password
	role	VARCHAR	Role of the user (e.g., 'customer', 'admin')
restaurants	restaurant_id	INT (PK)	Unique identifier for each restaurant
	name	VARCHAR	Name of the restaurant
	location	VARCHAR	Address or area
	cuisine	VARCHAR	Type of cuisine
	rating	DECIMAL	Average rating value
reservations	reservation_id	INT (PK)	Unique ID for each reservation
	user_id	INT (FK)	Linked to the user who made the reservation
	restaurant_id	INT (FK)	Linked to the reserved restaurant
	date	DATE	Date of reservation
	time	TIME	Time of reservation
	guest_count	INT	Number of guests
meals	meal_id	INT (PK)	Unique identifier for each menu item
	restaurant_id	INT (FK)	Restaurant offering the meal
	name	VARCHAR	Meal name
	price	DECIMAL	Meal price
	category	VARCHAR	Meal type/category (e.g., appetizer, dessert)

preorders	preorder_id	INT (PK)	Unique identifier for a pre-order
	reservation_id	INT (FK)	Linked to a reservation
	meal_id	INT (FK)	Linked to a meal
	quantity	INT	Number of items ordered
facilities	facility_id	INT (PK)	Unique identifier for each facility
	name	VARCHAR	Facility name
	type	VARCHAR	Type (e.g., room, court)
	availability	TEXT	Available time slots
facility_bookings	booking_id	INT (PK)	Unique booking identifier
	user_id	INT (FK)	User who booked the facility
	facility_id	INT (FK)	Booked facility
	date	DATE	Booking date
	time_slot	VARCHAR	Selected time slot

6. Exception Handling

This section outlines the exceptions defined within the ReserV8 application, the scenarios that trigger them, logging procedures, and the appropriate follow-up actions.

6.1 Defined Exceptions and Circumstances

Exception Name	Description	Trigger Conditions	Handled By

UserAuthenticationError	Occurs when login credentials are incorrect, or token is invalid	Invalid username/password, expired/invalid token	Authentication Module
TableBookingError	Raised when a booking cannot be completed due to conflicts or invalid input	Table already booked, invalid date/time format, table unavailable	Booking Service
DatabaseConnectionError	Signals a failure to connect to the database	Database server down, incorrect DB URL, network failure	Backend API Layer
InputValidationError	Raised when user-submitted data doesn't meet format or logic constraints	Empty required fields, invalid date/time, incorrect format	Frontend & Backend Validation
UnauthorizedAccessError	Attempt to access restricted resources without proper permissions	Role-based access violation	Access Control Middleware

6.2 Logging Strategy

All exceptions are logged using a centralized logging service with the following details:

- Timestamp
- User session ID
- Exception type and message
- Request payload (excluding sensitive data)
- Stack trace (for developers)

Logs are stored in a secure and access-controlled logging system, and critical errors trigger alerts to the development team via email or messaging integrations.

6.3 Follow-up Actions

Exception	Follow-up Action
UserAuthenticationError	Prompt user to retry login or reset password
TableBookingError	Show user-friendly message and suggest alternate times/tables
DatabaseConnectionError	Retry after exponential backoff; show fallback page if retries fail

InputValidationError	Highlight invalid fields and prompt for correction
UnauthorizedAccessError	Redirect to login or home page with appropriate message

7. Configurable Parameters

This section outlines the configurable parameters used in the ReserV8 application. These parameters enable flexibility across different environments (development, staging, production) and system behavior adjustments. Parameters marked as Dynamic can be changed at runtime without restarting the application.

Simple Configuration Parameters

Configuration Parameter Name	Definition and Usage	Dynamic?
MAX_RESERVATIONS_PER_SLOT	Maximum number of table reservations allowed per time slot per restaurant	Yes
DB_CONNECTION_STRING	Connection string used to access the MySQL database	No
FIREBASE_API_KEY	API key for accessing Firebase Authentication and Firestore services	No
PAYMENT_TIMEOUT_SECONDS	Timeout duration (in seconds) for payment confirmation via the payment gateway	Yes
MAINTENANCE_WINDOW_START	Time at which the application will initiate scheduled maintenance	Yes
SESSION_TIMEOUT_MINUTES	Time after which inactive sessions are automatically logged out	Yes
EMAIL_SENDER_ADDRESS	Default sender email address used for reservation and notification emails	Yes
ENABLE_DEBUG_LOGGING	Enables or disables detailed logging for debugging purposes	Yes

RESERVATION_CUTOFF_HOURS	Minimum number of hours in advance a reservation must be made	Yes
ALLOWED_LOGIN_ATTEMPTS	Number of failed login attempts allowed before locking a user account	Yes

8. Quality of Service

This section outlines how the ReserV8 system ensures high standards of service quality in terms of availability, security, performance, and system monitoring.

8.1 Availability

ReserV8 is designed to maintain high availability to support real-time restaurant table bookings and management.

- **Redundant Cloud Hosting:** The application is deployed on cloud infrastructure with failover and load balancing to ensure minimal downtime.
- **Database Backups:** Daily automated backups are configured to minimize data loss in case of system failure.
- **Scheduled Maintenance:** Maintenance tasks, such as data cleanup and updates, are scheduled during non-peak hours to avoid user disruption.
- **Graceful Degradation:** In case of partial system failures (e.g., payment gateway outage), core functionalities such as viewing menus or reservations remain accessible.

8.2 Security and Authorization

Security is integral to ReserV8, covering both data privacy and secure feature access.

- **User Authentication:** The system uses Authentication to securely validate user credentials and manage sessions.
- **Role-Based Access Control (RBAC):** Access is differentiated between users (customers), restaurant managers, and admins. Managers can only access data relevant to their restaurant.
- **Data Encryption:** Sensitive data (e.g., payment info) is encrypted in transit (HTTPS) and at rest.
- **Security Reviews:** Regular reviews and vulnerability scans are scheduled to ensure protection against evolving threats.

8.3 Load and Performance Implications

The system is designed to handle varying traffic loads, especially during peak hours (e.g., weekends, holidays).

- **Expected Load:**
  - ~100 concurrent users per restaurant during peak hours
  - ~5,000 reservation transactions/day (across all restaurants).

- **Performance Goals:**
  - Page load time: < 3 seconds for most screens
  - Booking confirmation response time: < 2 second

- **Database Growth:** Reservation records are expected to grow by 100 entries/month. Partitioning and indexing strategies are used to maintain query performance.

- **Scalability:** The backend is horizontally scalable, using containerized services to manage increased request volumes.

8.4 Monitoring and Control

The system incorporates real-time monitoring and control mechanisms to ensure reliable operation.

- **Health Checks:** All microservices have health endpoints checked periodically by an orchestration layer.
- **Process Monitoring:** Background jobs such as table availability updates, periodic cleanup, and notification delivery are monitored via a job scheduler.
- **Metrics and Logs:**
  - Metrics such as response time, error rate, and system load are collected.
  - Logs are centralized and categorized (info, warning, error) to aid in debugging and alerts.
- **Alerts:** Notifications are triggered for anomalies such as API failure spikes or database latency issues.