<u>Department of Computer Science-Software Engineering</u>

Software Design and Architecture

Assignment No: 1

COMSATS University Islamabad

Dhamtor campus

Submitted to:

Sir Mukhtiar Zamin

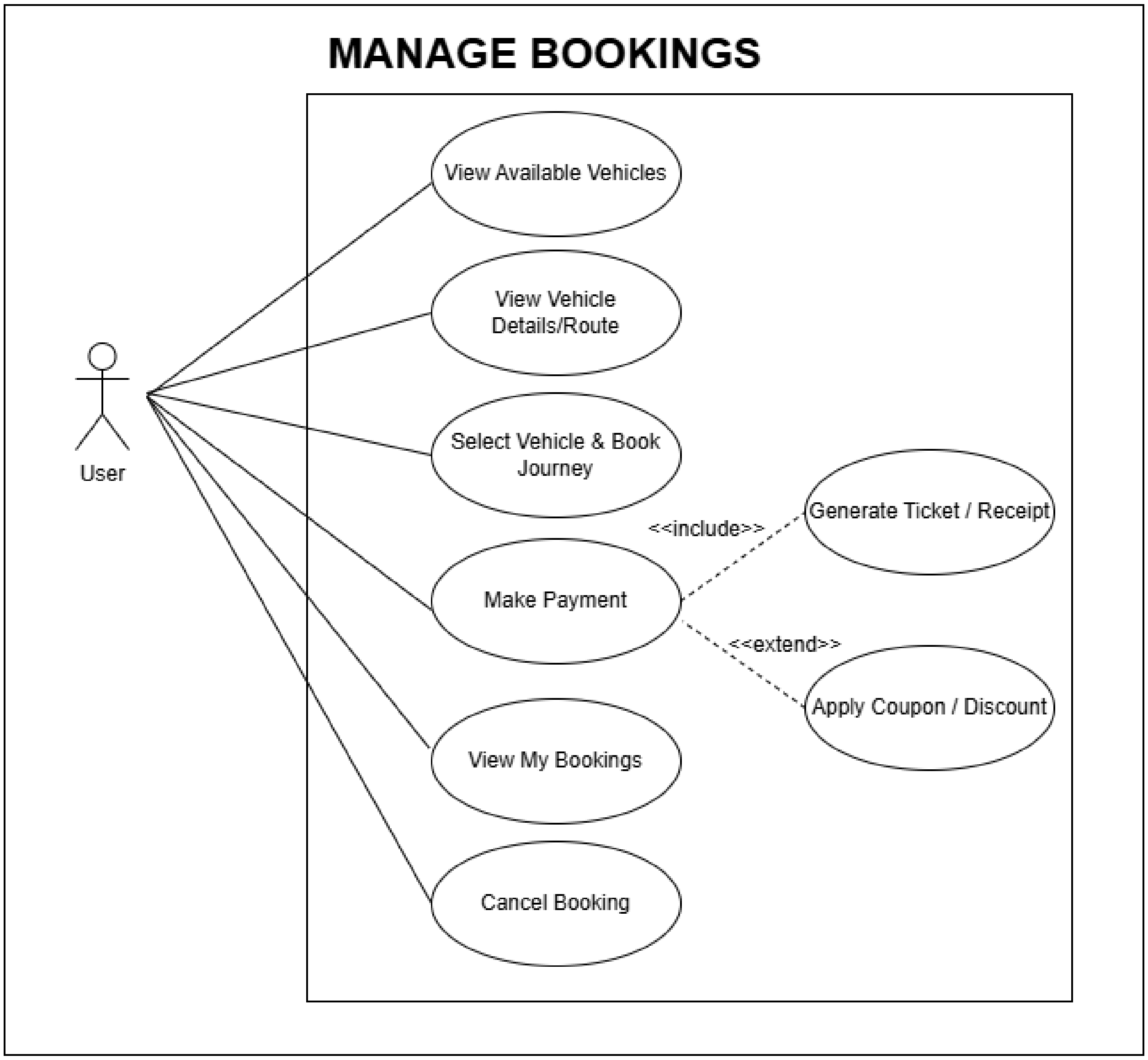
Submitted by:

Alaina Khan

Sp23-bse-069

Contents

Chapter 1: Use Case Diagram



Chapter 2: Fully Dressed Use Cases

USE CASE ID	**UC-01**
USE CASE NAME	View Available Vehicles
ACTOR	Registered User
DESCRIPTION	This use case allows a registered user to view a list of vehicles available for booking. The list is filtered based on input criteria such as location, vehicle type, and availability date.
TRIGGER	User selects “Book Vehicle” from the dashboard and then chooses the option to view available vehicles.
PRE-CONDITION	User must be logged into the system. System must have vehicle data available.
POST CONDITION	List of available vehicles is displayed based on selected criteria.
NORMAL FLOW	1. User logs into the system. 2. User selects "Book Vehicle" from dashboard. 3. System displays filter options (date, location, type). 4. User selects filters and submits. 5. System queries available vehicles. 6. List of available vehicles is shown to the user.
ALTERNATIVE FLOW	3a. User does not apply any filters → System displays all available vehicles.
EXCEPTIONS	1. No vehicles available → System displays “No vehicles available for selected criteria.” 2. System error during fetch → Show error message.
BUSINESS RULES	- Only available vehicles should be shown. - User can only view vehicles available in the selected location/date. - Unregistered or logged-out users cannot access this page.
ASSUMPTIONS	- Vehicle data is regularly updated. - User has stable internet access. - Vehicles are tagged properly with status and location in the database.

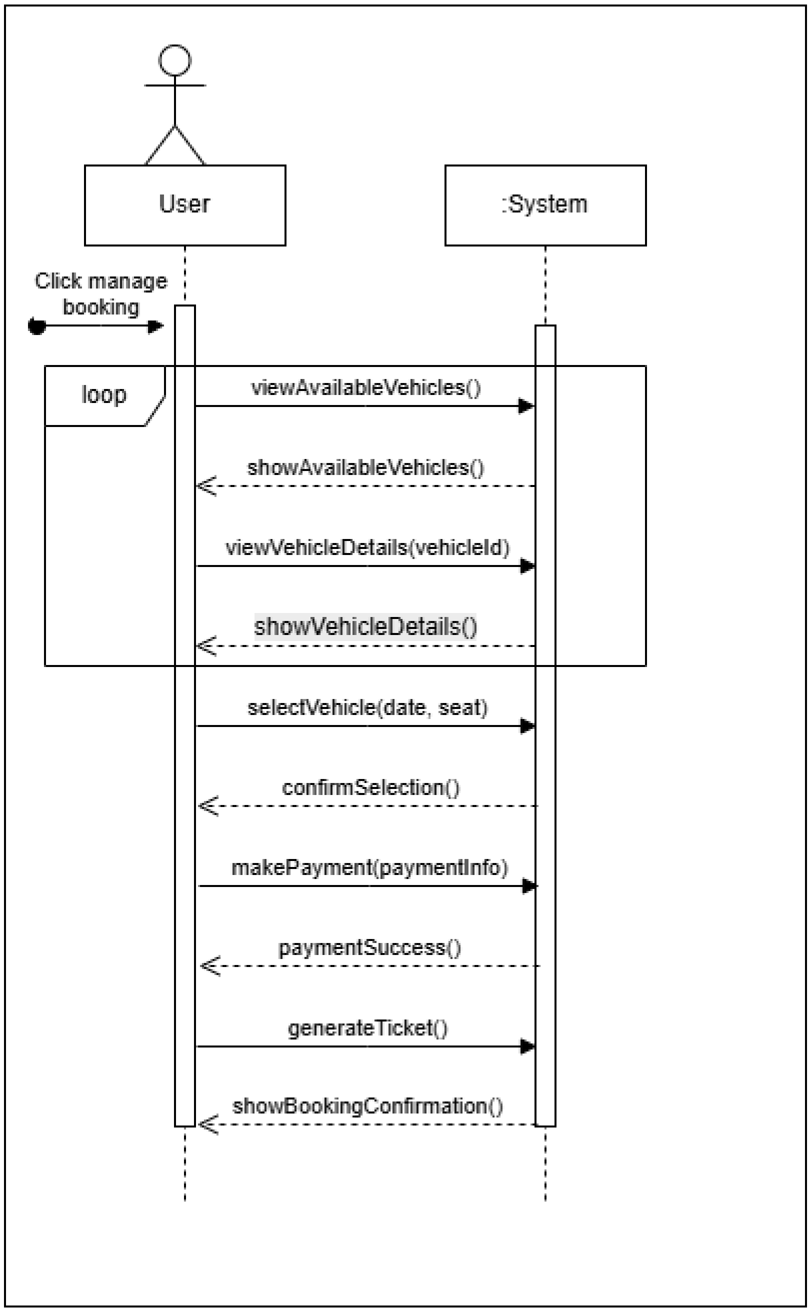
USE CASE ID	**UC-02**
USE CASE NAME	View Vehicle Details / Routes
ACTOR	Registered User
DESCRIPTION	After viewing the list of available vehicles, the user can select a vehicle to view more detailed information including its route, schedule, seat layout, and estimated timings.
TRIGGER	User clicks on a specific vehicle from the list of available vehicles.
PRE-CONDITION	- User must be logged in. - List of available vehicles must already be displayed.
POST CONDITION	Vehicle details, including route map and related info, are displayed to the user.

NORMAL FLOW	1. User views list of available vehicles.2. User selects a vehicle.3. System fetches vehicle information and route details.4. System displays vehicle details (type, number, features).5. System shows route map, stops, timings, and estimated travel time.6. Seat layout and availability are optionally shown.
ALTERNATIVE FLOW	4a. If route data is not available, system only displays basic vehicle info.6a. If seat layout is not supported, show general availability count.
EXCEPTIONS	- Selected vehicle no longer available → Show warning message. - Database timeout or error → Display technical error.
BUSINESS RULES	- User can only view details of available vehicles. - Routes must be displayed using map or list format. - Only valid and scheduled routes should be shown.
ASSUMPTIONS	- Route and vehicle data are pre-entered and accurate. - User interface supports map or route display features. - Internet connection is stable for map rendering.

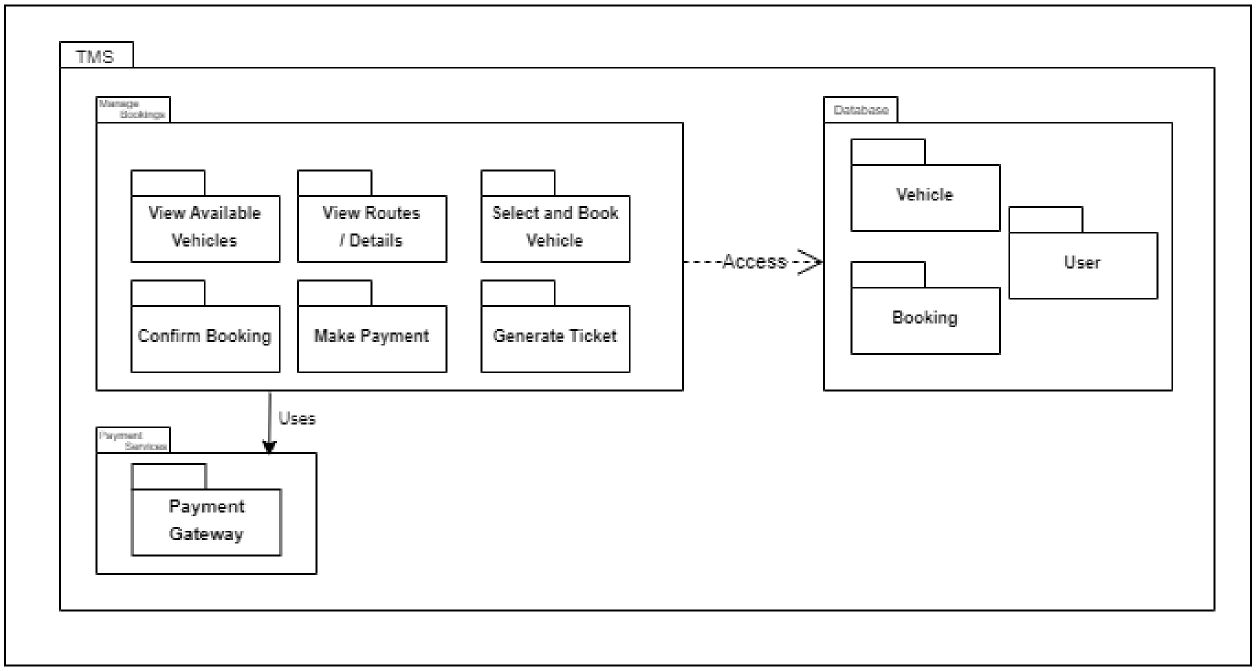
Use Case ID	UC-003
Use Case Name	Select Vehicle & Book Journey
Actor	**Primary Actor:** Customer **Secondary Actor:** Vehicle System **Supporting Actor:** Payment System
Description	This use case allows a customer to select a vehicle for their journey, review available options, confirm the booking, and complete the payment process.
Trigger	The customer initiates the vehicle selection and booking process by selecting the "Select Vehicle & Book Journey" option on the dashboard.
Preconditions	- The customer is logged into the system. - The customer has entered the destination and journey details. - The system has a list of available vehicles. - The customer has a valid payment method.
Postconditions	- The customer has successfully booked a vehicle for the specified journey. - The system has reserved the selected vehicle. - Payment has been processed and confirmed.
Normal Flow	1. Customer selects "Select Vehicle & Book Journey" from the dashboard. 2. The system displays available vehicles. 3. Customer reviews available vehicles and selects one. 4. System shows booking details (vehicle type, journey time, cost). 5. Customer confirms the booking. 6. The system processes the payment. 7. The system confirms the booking and sends a confirmation notification.
Alternative Flow	**A1: No Available Vehicles**: If no vehicles are available, the system will notify the customer. The customer can change preferences or try again later. **A2: Payment Failure**: If payment fails, the system notifies the customer, and they can retry with a different payment method.

	A3: Customer Cancels Booking: If the customer cancels, the system returns to the vehicle selection page.
Exception	**E1: Payment Gateway Unavailable**: If the payment gateway fails, the system notifies the customer and allows them to retry later. **E2: Technical Issue in Vehicle Selection**: If there's a technical issue, the system asks the customer to retry or contact support. **E3: Invalid Payment Information**: The system asks the customer to re-enter valid payment details.
Business Rules	- BR1: Vehicle must be available for the entire duration of the journey. - BR2: Booking is confirmed only after successful payment. - BR3: Vehicle prices may fluctuate based on demand and availability.
Assumptions	- The customer has a pre-registered account. - The customer knows the desired journey details (time, destination). - The system has access to up-to-date vehicle availability.

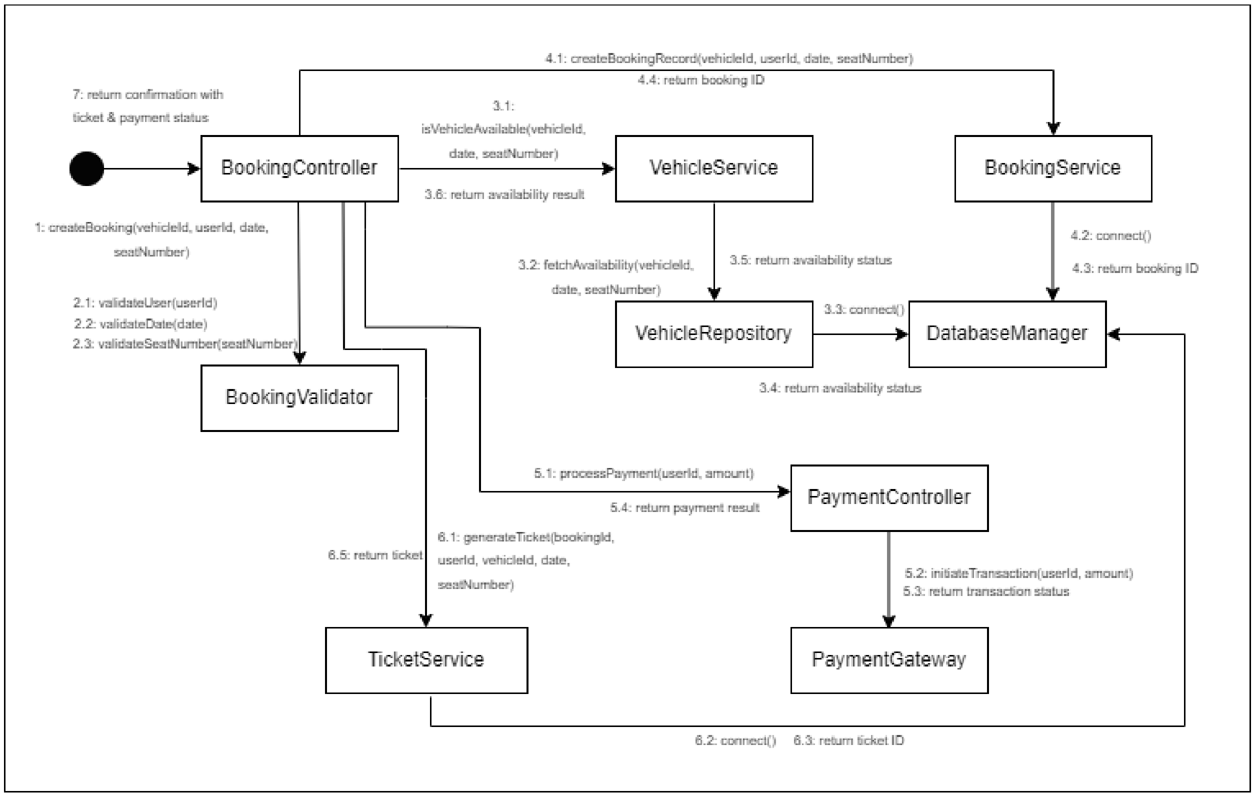
Chapter 3: SSDs



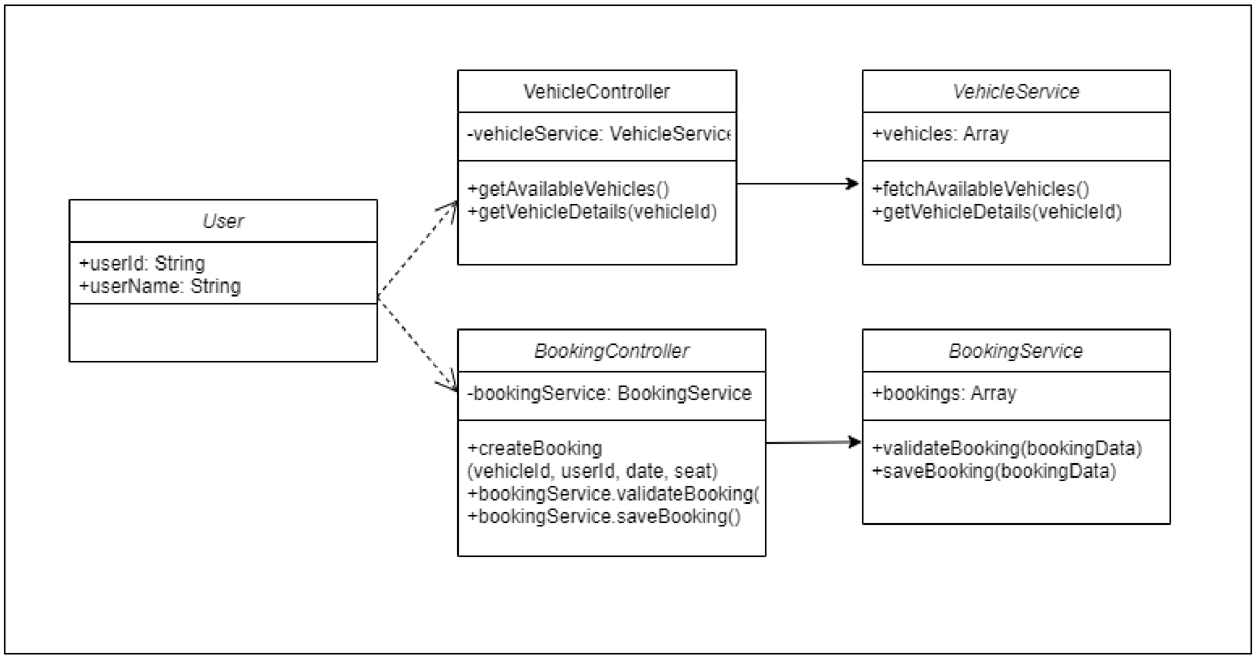
Chapter 4: Package diagram



Chapter 5: Collaboration Diagrams


4.1: createBookingRecord(vehicleId, userId, date, seatNumber)
4.4: return booking ID
7: return confirmation with ticket & payment status
3.1: isVehicleAvailable(vehicleId, date, seatNumber)
BookingController
VehicleService
BookingService
3.6: return availability result
1: createBooking(vehicleId, userId, date, seatNumber)
4.2: connect()
4.3: return booking ID
3.2: fetchAvailability(vehicleId, date, seatNumber)
3.5: return availability status
2.1: validateUser(userId)
2.2: validateDate(date)
2.3: validateSeatNumber(seatNumber)
3.3: connect()
VehicleRepository
DatabaseManager
BookingValidator
3.4: return availability status
5.1: processPayment(userId, amount)
PaymentController
5.4: return payment result
6.5: return ticket
6.1: generateTicket(bookingId, userId, vehicleId, date, seatNumber)
5.2: initiateTransaction(userId, amount)
5.3: return transaction status
TicketService
PaymentGateway
6.2: connect()
6.3: return ticket ID

Chapter 6: Class Diagram


User
+userId: String
+userName: String
VehicleController
-vehicleService: VehicleServic
+getAvailableVehicles()
+getVehicleDetails(vehicleId)
VehicleService
+vehicles: Array
+fetchAvailableVehicles()
+getVehicleDetails(vehicleId)
BookingController
-bookingService: BookingService
+createBooking
(vehicleId, userId, date, seat)
+bookingService.validateBooking(
+bookingService.saveBooking()
BookingService
+bookings: Array
+validateBooking(bookingData)
+saveBooking(bookingData)

Chapter 7: Coding Standards

Coding Standards for TMS Manage Bookings Project

I intend to follow a set of basic, consistent coding standards while developing the **Manage Bookings module** for the TMS (Transport Management System). These standards are aimed at making the code clear, readable, maintainable, and logically organized for both myself and others who may review it.

Since this is a learning exercise, I will focus on fundamental conventions and practices suitable for beginner to intermediate-level projects.

1. Naming Conventions

- **Class Names:**
  Class names will be written in **PascalCase** (each word starting with a capital letter).
  Example: BookingController, VehicleService.
- **Variables and Function Names:**
  Variable and function names will be written in **camelCase** (the first word lowercase, each subsequent word capitalized).
  Example: availableVehicles, validateBookingDetails().

- **File Names:**
  File names will be meaningful and consistent with the class or component they contain. For React components, the file name will match the component name.
  Example: ManageBookings.js, VehicleList.js.

2. Indentation and Formatting

- Code will be consistently indented using **4 spaces** per indentation level.
- Each logical block of code (functions, conditionals, loops) will be separated by one blank line to enhance readability.
- Opening braces { will be placed on the **same line** as the control structure or function declaration.

Example:

```
if (isAvailable) {
   confirmBooking();
}
```

3. Commenting

- Simple, clear comments will be included above functions and important code sections to describe their purpose.
- Comments will also be added for any non-obvious logic or processes.
- Comments will follow the single-line // format in JavaScript and React.

4. Function Design

- Functions will be **small and single-purpose**, focusing on doing one specific task.
- Where applicable, reusable utility functions will be placed in a separate utility file.

5. Error Handling

- Basic error checking will be implemented where required, such as validating input values before processing them.

- Error messages will be meaningful and assist in debugging.

6. Code Structure and Organization

- The codebase will follow a **modular structure**:
    - Components for UI elements.
    - Services for handling data and business logic.
    - Pages for complete views.
- Related files and classes will be grouped logically in folders.

Example Folder Structure:

```
/src
  /components
  /services
  /pages
  /assets
  App.js
  index.js
```

7. Consistency in Syntax and Practices

- Consistent use of **const** and **let** for variable declarations in JavaScript.
- Consistent arrow function syntax where appropriate.
- Removal of any unused code or variables during cleanup.
- Keeping code aligned with modern JavaScript (ES6+) and React conventions.

8. Object-Oriented Practices

- The project will implement **object-oriented programming (OOP) concepts** in JavaScript where suitable, particularly for services and controllers.
- Classes will have clearly defined attributes (properties) and methods (functions) based on their responsibilities.