System Documentation

Car Rental System

Thanushka Praveen Wickramarachchi
Student ID: 270599091

Yoobee College
Master of Software Engineering

Course: MSE800 Professional Software Engineering
Assignment 1: Object-Oriented Programming
Assignment Type: Individual Project

Date: February 3, 2025

Contents

1 Introduction

The Car Rental System automates the rental process by replacing manual operations with an efficient and scalable software solution. The system improves efficiency, enhances customer satisfaction, and ensures scalability for future needs.

Objectives:

- Automate rental processes.
- Provide a user-friendly system for admins and customers.
- Ensure scalability and maintainability using advanced software design principles.

2 Class Diagram

Purpose: The Class Diagram represents the static structure of the system by illustrating its classes, their attributes, methods, and relationships. This helps in understanding the data model and interactions within the system.

2.1 Key Classes and Relationships

2.1.1 UserType Class

Attributes:

- **user_type (str):** The type or category of the user.
- **is_active (int):** A flag indicating whether the user type is active (1 for active, 0 for inactive). Defaults to 1.
- **user_type_id (int):** A unique identifier for the user type. Assigned after insertion into the database.

Methods:

- `__init__(user_type, is_active=1, user_type_id=None)`: Initializes a UserType object with the specified attributes.
- `insert(db, user_type)`: Inserts a UserType object into the database and updates its user_type_id.
- `update(db, user_type)`: Updates an existing UserType record in the database.
- `deactivate(db, user_type)`: Marks the UserType as inactive in the database.
- `delete(db, user_type)`: Deletes the UserType from the database.
- `select(db, user_type=None)`: Retrieves UserType records. Returns a list of UserType objects.

Relationships:

- **UserType → User:** One-to-Many (1..*): A user type can apply to multiple users.

2.1.2 User Class

Attributes:

- **user_type_id (int):** Foreign key referencing the UserType class.
- **user_name (str):** Name of the user.
- **user_email (str):** Email address of the user.
- **user_phone_number (str):** Contact phone number for the user.
- **user_password (str):** Password for user authentication.
- **is_active (int):** Indicates whether the user is active (1 for active, 0 for inactive). Defaults to 1.
- **user_id (int):** Unique identifier for the user. Assigned after insertion into the database.

Methods:

- `__init__(...)`: Initializes a User object with the provided attributes.
- `insert(db, user)`: Inserts a new user into the database and updates the user_id.
- `update(db, user)`: Updates an existing user's details in the database.
- `deactivate(db, user)`: Marks the user as inactive.
- `delete(db, user)`: Deletes the user record from the database.
- `select(db, user=None)`: Retrieves all users or a specific user by ID. Returns a list of User objects.
- `find_by_email_and_password(db, user_email, user_password=None)`: Finds a user by email and optionally password.

Relationships:

- **User → Booking:** One-to-Many (1..*): A user can have multiple bookings.

2.1.3 CarType Class

Attributes:

- **car_type (str):** Represents the type or category of the car.
- **is_active (int):** Indicates whether the car type is active (1 for active, 0 for inactive). Defaults to 1.
- **car_type_id (int):** Unique identifier for the car type. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes a CarType object.

- `insert(db, car_type)`: Inserts a new car type into the database and updates its car_type_id.
- `update(db, car_type)`: Updates an existing car type.
- `deactivate(db, car_type)`: Marks the car type as inactive.
- `delete(db, car_type)`: Deletes the car type from the database.
- `select(db, car_type=None)`: Retrieves car types. Returns a list of CarType objects.

Relationships:

- **CarType → CarBrandModel:** One-to-Many (1..*): A car type can include multiple brand models.

2.1.4 CarBrandModel Class

Attributes:

- **car_type_id (int):** Foreign key referencing CarType.
- **brand_name (str):** The name of the car brand.
- **model_name (str):** The name of the specific car model.
- **is_active (int):** Indicates whether the brand and model are active. Defaults to 1.
- **car_brand_model_id (int):** Unique identifier for the brand model. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes a CarBrandModel object.
- `insert(db, car_brand_model)`: Inserts a new car brand model and updates its ID.
- `update(db, car_brand_model)`: Updates an existing brand model.
- `deactivate(db, car_brand_model)`: Marks the brand model as inactive.
- `delete(db, car_brand_model)`: Deletes the brand model from the database.
- `select(db, car_brand_model=None)`: Retrieves brand models. Returns a list of CarBrandModel objects.

Relationships:

- **CarBrandModel → Car:** One-to-Many (1..*): A brand model can apply to multiple cars.

2.1.5 CarStatus Class

Attributes:

- **car_status_type (str):** Description of the car status (e.g., "Available", "Under Maintenance").
- **is_active (int):** Indicates whether the status is active. Defaults to 1.
- **car_status_id (int):** Unique identifier for the car status. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes a CarStatus object.
- `insert(db, car_status)`: Inserts a new car status into the database.
- `update(db, car_status)`: Updates an existing car status.
- `deactivate(db, car_status)`: Marks the car status as inactive.
- `delete(db, car_status)`: Deletes the car status from the database.
- `select(db, car_status=None)`: Retrieves car statuses. Returns a list of CarStatus objects.

Relationships:

- **CarStatus → Car:** One-to-Many (1..*): A car can have one status, but a status can apply to many cars.

2.1.6 Car Class

Attributes:

- **car_brand_model_id (int):** Foreign key to CarBrandModel.
- **car_status_id (int):** Foreign key to CarStatus.
- **number_plate (str):** License plate of the car.
- **model_name (str):** Model name of the car.
- **daily_rate (float):** Daily rental rate for the car.
- **year (int):** Year of manufacture.
- **mileage (int):** Total mileage of the car.
- **min_rental_period (int):** Minimum rental period in days.
- **max_rental_period (int):** Maximum rental period in days.
- **is_active (int):** Indicates if the car is active. Defaults to 1.
- **car_id (int):** Unique identifier for the car. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes a Car object.
- `insert(db, car)`: Inserts a new car into the database.
- `update(db, car)`: Updates an existing car's details.
- `deactivate(db, car)`: Marks the car as inactive.
- `delete(db, car)`: Deletes the car record.
- `select(db, car=None)`: Retrieves car records. Returns a list of Car objects.

Relationships:

- **Car → Booking:** One-to-Many (1..*): A car can be booked multiple times.

2.1.7 BookingStatus Class

Attributes:

- **status_type (str):** Name of the booking status (e.g., "Pending", "Confirmed").
- **is_active (int):** Indicates whether the status is active. Defaults to 1.
- **booking_status_id (int):** Unique identifier for the status. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes a BookingStatus object.
- `insert(db, booking_status)`: Inserts a new booking status into the database.
- `update(db, booking_status)`: Updates an existing status.
- `deactivate(db, booking_status)`: Marks the status as inactive.
- `delete(db, booking_status)`: Deletes the status by ID.
- `select(db, booking_status=None)`: Retrieves statuses. Returns a list of BookingStatus objects.

Relationships:

- **BookingStatus → Booking:** One-to-One (1..1): Each booking has a unique status.

2.1.8 Booking Class

Attributes:

- **user_id (int):** Foreign key referencing the User.
- **car_id (int):** Foreign key referencing the Car.
- **booking_status_id (int):** Foreign key referencing BookingStatus.
- **start_date (str):** Start date of the booking.
- **end_date (str):** End date of the booking.
- **total_amount (float):** Total booking cost.
- **note (str):** Additional information about the booking.
- **is_active (int):** Indicates if the booking is active. Defaults to 1.
- **booking_id (int):** Unique identifier for the booking. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes a Booking object.
- `insert(db, booking)`: Inserts a new booking into the database.
- `update(db, booking)`: Updates an existing booking.
- `deactivate(db, booking)`: Marks the booking as inactive.
- `delete(db, booking)`: Deletes the booking by ID.
- `select(db, booking=None)`: Retrieves bookings. Returns a list of Booking objects.

Relationships:

- **Booking → Invoice:** One-to-One (1..1): Each booking generates one invoice.

2.1.9 AdditionalServices Class

Attributes:

- **services_description (str):** Description of the service (e.g., "GPS", "Child Seat").
- **services_amount (float):** Cost of the service.
- **is_active (int):** Indicates if the service is active. Defaults to 1.
- **additional_services_id (int):** Unique identifier for the service. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes an AdditionalServices object.
- `insert(db, service)`: Inserts a new service into the database.
- `update(db, service)`: Updates an existing service.
- `deactivate(db, service)`: Marks the service as inactive.
- `delete(db, service)`: Deletes the service from the database.
- `select(db, service=None)`: Retrieves services. Returns a list of AdditionalServices objects.

Relationships:

- **AdditionalServices → BookingAdditionalServices:** One-to-Many (1..*): A service can apply to multiple bookings.

2.1.10 BookingAdditionalServices Class

Attributes:

- **booking_id (int):** Foreign key referencing the Booking.
- **additional_services_id (int):** Foreign key referencing AdditionalServices.
- **is_active (int):** Indicates if the record is active. Defaults to 1.
- **booking_additional_charge_id (int):** Unique identifier for the record. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes a BookingAdditionalServices object.
- `insert(db, booking_service)`: Inserts a new record linking a booking and service into the database.
- `update(db, booking_service)`: Updates an existing record.
- `deactivate(db, booking_service)`: Marks the record as inactive.
- `delete(db, booking_service)`: Deletes the record from the database.
- `select(db, booking_service=None)`: Retrieves records. Returns a list of BookingAdditionalServices objects.

Relationships:

- **BookingAdditionalServices → Booking:** Many-to-One (*..1): Each record is linked to a booking.
- **BookingAdditionalServices → AdditionalServices:** Many-to-One (*..1): Each record is linked to a service.

2.1.11 Invoice Class

Attributes:

- **booking_id (int):** Foreign key referencing the Booking.
- **user_id (int):** Foreign key referencing the User.
- **amount (float):** Total invoice amount.
- **payment_method (str):** Payment method (e.g., "Credit Card", "PayPal").
- **payment_date (str):** Payment date in a string format (e.g., "YYYY-MM-DD").
- **is_paid (int):** Indicates if the invoice is paid (1 for paid, 0 for unpaid). Defaults to 0.
- **is_active (int):** Indicates if the invoice is active. Defaults to 1.
- **invoice_id (int):** Unique identifier for the invoice. Assigned after insertion.

Methods:

- `__init__(...)`: Initializes an Invoice object.
- `insert(db, invoice)`: Inserts a new invoice into the database.
- `update(db, invoice)`: Updates an existing invoice in the database.
- `deactivate(db, invoice)`: Marks the invoice as inactive.
- `delete(db, invoice)`: Deletes the invoice from the database.
- `select(db, invoice=None)`: Retrieves invoices. Returns a list of Invoice objects.

Relationships:

- **Invoice → Booking:** One-to-One (1..1): Each invoice is tied to a booking.
- **Invoice → User:** Many-to-One (*..1): Multiple invoices can belong to a single user.

2.2 Summary of Relationships

- **UserType → User:** One-to-Many (1..*).
- **User → Booking:** One-to-Many (1..*).
- **CarType → CarBrandModel:** One-to-Many (1..*).
- **CarBrandModel → Car:** One-to-Many (1..*).
- **CarStatus → Car:** One-to-Many (1..*).
- **Booking → Invoice:** One-to-One (1..1).
- **Invoice → User:** Many-to-One (*..1).

- **AdditionalServices → BookingAdditionalServices:** One-to-Many (1..*).
- **BookingAdditionalServices → Booking:** Many-to-One (*..1).
- **BookingAdditionalServices → AdditionalServices:** Many-to-One (*..1).

2.3 Class Diagram

Figure 1: Class Diagram

3 Use Case Diagram and Descriptions

Purpose: The Use Case Diagram illustrates user interactions with the system, highlighting its functionalities and roles of different users.

Actors:

- **Admin\Customer:** Register\Login
- **Admin:** Manage Cars data, users, and bookings, Approve/Reject Bookings, Manage Services.
- **Customer:** Booking Cars, View Bookings, and views invoices, Pay invoice.

Use Cases:

- **Admin:**
  - Register/Login
  - Manage Cars
  - Approve/Reject Bookings
  - Manage Customers
  - Manage Services
- **Customer:**
  - Register/Login
  - Booking a Car
  - View Bookings
  - View Invoices

3.1 Use Case Diagram

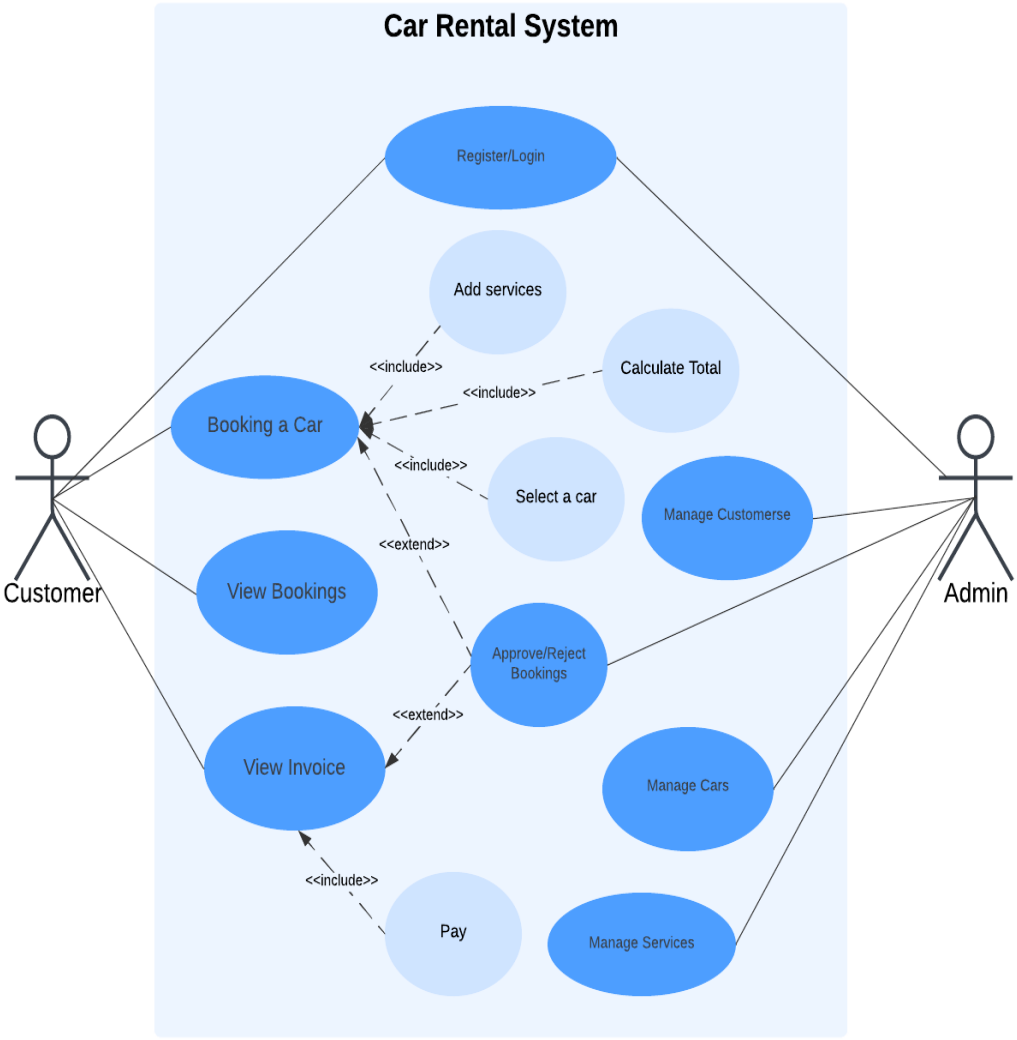


Figure 2: Use Case Diagram

3.2 Use Case Descriptions

Use case name:	Register/Login
Scenario:	Login: Verify existing customer or admin credentials, including email and password. Register: Create a new account for a customer or admin and input the provided data into the system's database.

Triggering event:	• A new customer wants to book a car and needs to create an account. • An admin needs to create an account to manage users, cars, and bookings.
Brief description:	To interact with the car rental system, users (customers and admins) must register and log in. Registration involves creating an account by providing details such as name, email, phone number, and password. After registration, customers can book cars, and admins can manage the system. Logging in allows users to access the system based on their role and privileges.
Actors:	• Customer: Wants to book a car. • Admin: Manages users, cars, and bookings.
Related use cases:	• Makes Booking: Customers book cars. • Manage Management: Admin approves or rejects rental requests.
Stakeholders:	• Car Rental Company: Seeks a streamlined and automated system for car rentals. • Customers: Want an easy and error-free process for booking cars. • Admins: Require an efficient way to manage cars, users, and bookings.
Preconditions:	• The system is online and operational. • Users have access to the car rental platform. • A valid database exists for storing and retrieving user credentials.

<table>
<tr><td>Postconditions:</td><td colspan="2">Success:
Customers are redirected to the booking interface.
Admins are redirected to the admin dashboard.
Failure:
Users receive appropriate error messages (e.g., invalid credentials, email already in use).</td></tr>
<tr><td rowspan="2">Flow of activities:</td><td>Actor (Customer/Admin):</td><td>System:</td></tr>
<tr><td>Register:
Provide details: name, valid email address, phone number, and password.
Submit the registration form.

Login:
Enter registered email address and password.
Submit the login form.</td><td>Register:
Validate the email format and check if it already exists.
Validate the phone number format.
Save the user data with the appropriate role (customer or admin).
Navigate the user to the appropriate home screen based on their role.

Login:
Verify the provided email and password with database records.
Identify the user's role (customer or admin).
Navigate the user to the appropriate dashboard.</td></tr>
<tr><td>Exception conditions:</td><td colspan="2">Registration:
Invalid email format or already in use.
Phone number is not valid.
Password does not meet security requirements.
Login:
Email not registered.
Incorrect password.
Database connectivity issues.</td></tr>
</table>

Use case name:	Booking a Car
Actors:	• Customer: Selects and books a car for rental. • Admin: Approves or rejects rental requests and manages invoices.
Related use cases:	• View Bookings: Customers view their booking details. • View Invoice: Customers view the invoice after admin approval. • Approve/Reject Bookings: Admin approves or rejects customer rental requests.
Stakeholders:	• Car Rental Company: Aims to improve efficiency in car rental processes. • Customers: Require a straightforward process to book cars and view booking details. • Admins: Need tools to manage bookings, approvals, and invoices.
Preconditions:	• The system is online and operational. • Users (customers and admins) have access to the car rental platform. • A valid database exists for storing and retrieving data, including cars, users, bookings, and additional services.

<table>
<tr><td>Postconditions:</td><td colspan="2">Success:
Customers successfully submit their booking request and await admin approval before paying the invoice.
Failure:
Users receive error messages if fields are incomplete, invalid, or data constraints (e.g., booking duration, date format) are not met.</td></tr>
<tr><td rowspan="2">Flow of activities:</td><td>Actor (Customer):</td><td>System:</td></tr>
</table>

	• Log in to the car rental system. • View the list of available cars. • Select a car from the displayed options. • Enter the number of booking days (minimum 1, maximum 30). • Input the rental start date in the format YYYY-MM-DD. • Specify whether additional services are required (yes/no). • If "yes," select the desired additional services.	• Display the list of available cars. • Validate the customer's input: – Ensure booking duration is within the 1–30 days range. – Verify the date format is YYYY-MM-DD. • Show available additional services if the customer selects "yes." • Calculate the total booking amount: – Multiply the daily rate by the number of booking days. – Add prices for selected additional services. • Display booking details to the customer, including: – Car details. – Rental start date and duration. – Additional services and their costs. – Total amount payable. • Save the booking request in the database for admin approval.

Exception conditions:	**Registration:** • Invalid email format or already registered email. • Phone number is invalid. • Password does not meet security standards. **Login:** • Email not registered. • Incorrect password. • Database connectivity issues. **Booking:** • Invalid date format (must be YYYY-MM-DD). • Booking duration outside the allowed range (1–30 days). • Missing or incomplete mandatory fields (e.g., car selection, start date). **Additional Services:** • Invalid service selection (row number must follow proper format and comma-separated values).

Use case name:	View Bookings
Actor:	Customer: Wants to view their past and pending booking details.
Related use cases:	• Booking a Car: Creates the bookings that the customer can view. • Approve/Reject Bookings: Admin action that affects the status of bookings visible to the customer. • View Invoice: The customer can access invoices for approved bookings.

<table>
<tr><td>Stakeholders:</td><td colspan="2">Customers: Need a clear view of their booking details to manage rentals.
Car Rental Company: Aims to enhance customer satisfaction by providing transparency and accessibility.</td></tr>
<tr><td>Preconditions:</td><td colspan="2">The customer is logged into the system.
Booking details are stored in the database.
The system is operational and can retrieve data.</td></tr>
<tr><td>Postconditions:</td><td colspan="2">Success:
The customer views their booking details, including pending, approved, or rejected bookings.
Failure:
The customer is notified if no bookings are available or if an error occurs during data retrieval.</td></tr>
<tr><td>Flow of activities:</td><td>Actor (Customer):</td><td>System:</td></tr>
</table>

	• Log in to the car rental system. • Navigate to the "View Bookings" section. • Select the type of bookings to view (e.g., all, pending, approved, or rejected).	• Verify the customer's identity and fetch their booking records. • Display the list of bookings with the following details: – Booking ID. – Car details (make, model, etc.). – Booking status (pending, approved, or rejected). – Rental start date and end date. – Total booking amount (if applicable). • If no bookings are found, display a message indicating no records are available. • Allow the customer to view additional details for each booking (e.g., invoice for approved bookings).
Exception conditions:	• **No Bookings Found:** If the customer has not made any bookings, display a message: "No bookings found." • **Database Error:** If the system cannot retrieve booking details, display an error message"	

Use case name:	Manage Cars
Scenario:	Admins manage car information by adding, editing, or deleting car records. They can also view all cars, available cars, and booked cars in the system.
Triggering event:	The admin decides to add a new car or update existing car records.
Brief description:	Admins can manage car-related information in the car rental system. They have the ability to: • View all cars. • View available cars. • View all booked cars. • Add new cars. • Edit existing car records. • Delete cars no longer available for rental.
Actor:	Admin: Manages car records in the system.
Related use cases:	Booking a Car: Customers can view and book updated or newly added cars.
Stakeholders:	• Customers: Gain access to updated car details or newly added cars, ensuring a better rental experience. • Car Rental Company: Enhances customer satisfaction and operational efficiency by maintaining accurate car records.
Preconditions:	• The admin is logged into the car rental system. • For editing, car details must already exist in the database. • The system is operational and able to retrieve and store data.

<table>
<tr><td>Postconditions:</td><td colspan="2">Success:
New cars are available for customers to view and book.
Updated car details are visible to customers.
Failure:
The car data is not saved if incorrect details are entered, and the admin is notified of the error.</td></tr>
<tr><td rowspan="2">Flow of activities:</td><td>Actor (Admin):</td><td>System:</td></tr>
<tr><td>Log in to the car rental system.
Navigate to the "Manage Cars" section.
Perform one of the following actions:
View all cars.
View available cars.
View booked cars.
Add a new car by entering details such as make, model, number plate, car type, and daily rate.
Edit existing car details such as the car's availability, booking status, or daily rate.
Delete cars that are no longer available for rental.</td><td>Verify the admin's identity and authorization.
Fetch and display the list of cars with the following details:
Car ID.
Make and model.
Car status (available, booked, or not available).
Booking duration (if applicable).
Daily rental rate.
Validate the admin's input when adding or editing car records:
Ensure details like number plate, car type, and model are valid.
Save the new or updated car details to the database.
Reflect the changes in the system for customers and admins.</td></tr>
</table>

Exception conditions:	• **Invalid Car Details:** If the admin enters incorrect information such as an invalid number plate, car type, or model, the system displays an error message: "Invalid car details. Please check the entered information." • **Database Error:** If the system cannot save the car details due to a database issue, an error message is shown.

Use case name:	Approve/Reject Bookings
Scenario:	Admins are responsible for managing booking requests by accepting or rejecting them based on car availability and other criteria. Once approved, the system generates an invoice and notifies the customer.
Triggering event:	The admin receives a booking notification via email and accesses the system to approve or reject the booking request.
Brief description:	Admins review booking requests and verify car availability. If a car is already booked for overlapping dates, the system highlights the conflict in red. The admin can then decide to approve or reject the booking. Upon approval, an invoice is generated, and the customer is notified. If rejected, the system updates the status and sends an email to the customer.
Actor:	Admin: Manages booking requests, approves or rejects them, and oversees car availability.
Related use cases:	• View Bookings: Customers can view their booking details and updated status. • Invoice: Customers receive a generated invoice for payment after their booking request is approved.

<table>
<tr><td>Stakeholders:</td><td colspan="2">Customers:
Receive prompt feedback on booking requests.
Can view updated booking status and, if approved, an invoice.
Car Rental Company:
Ensures efficient management of car bookings.
Enhances customer satisfaction by providing timely updates on booking requests.</td></tr>
<tr><td>Preconditions:</td><td colspan="2">The admin is logged into the car rental system.
The system has pending booking requests stored in the database.
Customers have submitted booking requests with all necessary details.</td></tr>
<tr><td>Postconditions:</td><td colspan="2">Success:
The system successfully updates the booking status to "Approved" or "Rejected."
Customers receive notifications via email with the updated status and, if approved, an invoice.
Redundant or conflicting bookings are highlighted for admin decision-making.
Failure:
Invalid or conflicting booking requests remain unresolved.
Customers are notified of rejection or delays due to errors.</td></tr>
<tr><td>Flow of activities:</td><td>Actor (Admin):</td><td>System:</td></tr>
</table>

	• Log in to the car rental system. • Navigate to the "Manage Bookings" section. • Access the "View Booking Requests" subsection. • Review the booking requests and check for conflicts. • Conflicting bookings (overlapping dates) are highlighted in red. • Decide to approve or reject the booking request: – Approve: Confirm the booking and generate an invoice. – Reject: Provide a reason and update the booking status.	• Verify the admin's identity and authorization. • Fetch and display pending booking requests. • Highlight overlapping bookings in red in the booking request table. • Update the booking status based on the admin's decision: – **If approved:** ∗ Generate an invoice. ∗ Send an email notification with the invoice to the customer. – **If rejected:** ∗ Update the booking status as "Rejected." ∗ Notify the customer via email. • Save the updated booking status and associated details to the database.

Exception conditions:	• **Invalid Booking Request Acceptance:** – The admin attempts to accept a booking without checking conflicts or errors. – The system prompts the admin to review highlighted overlaps before confirming. • **System Errors:** – If the system encounters an issue while generating an invoice or sending emails, display an error message.

Use case name:	View Invoice
Scenario:	Customers can view the invoice generated by the system after their booking request has been approved by the admin.
Triggering event:	The admin approves the customer's booking request, and the system generates an invoice for the customer.
Brief description:	Once a booking is approved, the system generates an invoice detailing the booking information and charges. Customers can access this invoice to review details such as car rental costs, additional services, and payment instructions.
Actor:	Customer: Views and reviews the invoice for approved bookings.
Related use cases:	• Approve/Reject Bookings: The admin approves the booking, which triggers invoice generation. • View Bookings: Customers can navigate to the approved booking and view the associated invoice.
Stakeholders:	• **Customers:** Receive a detailed invoice for payment and can verify all charges. • **Car Rental Company:** Provides transparent billing and ensures accurate invoicing.

Preconditions:	• The customer has logged into the car rental system. • The booking request has been approved by the admin. • The system has generated an invoice and stored it in the database.	
Postconditions:	**Success:** • The customer views the invoice details, including all charges and payment information. **Failure:** • If no invoice is available or an error occurs, the customer is notified appropriately.	
Flow of activities:	**Actor (Customer):**	**System:**

	• Log in to the car rental system. • Navigate to the "View Bookings" section. • Select an approved booking. • Click on the "View Invoice" option.	• Verify the customer's identity and authorization. • Retrieve the invoice associated with the selected approved booking. • Display the invoice with the following details: – Invoice ID. – Booking ID. – Customer name. – Car details (make, model, etc.). – Rental start and end dates. – Daily rental rate. – Number of days booked. – Additional services and their costs (if applicable). – Total amount payable. • Notify the customer if no invoice is available for the selected booking.
Exception conditions:	• **No Invoice Found:** If no invoice exists for the selected booking, display a message: "No invoice is available for this booking." • **System Error:** If an error occurs while retrieving the invoice, display a message: "Unable to fetch invoice details. Please try again later."	

3.3 Sequence Diagram and Description

Purpose: The Sequence Diagram depicts interactions between objects in a sequential flow for specific scenarios, such as booking a car.

3.3.1 Register/Login Use Case

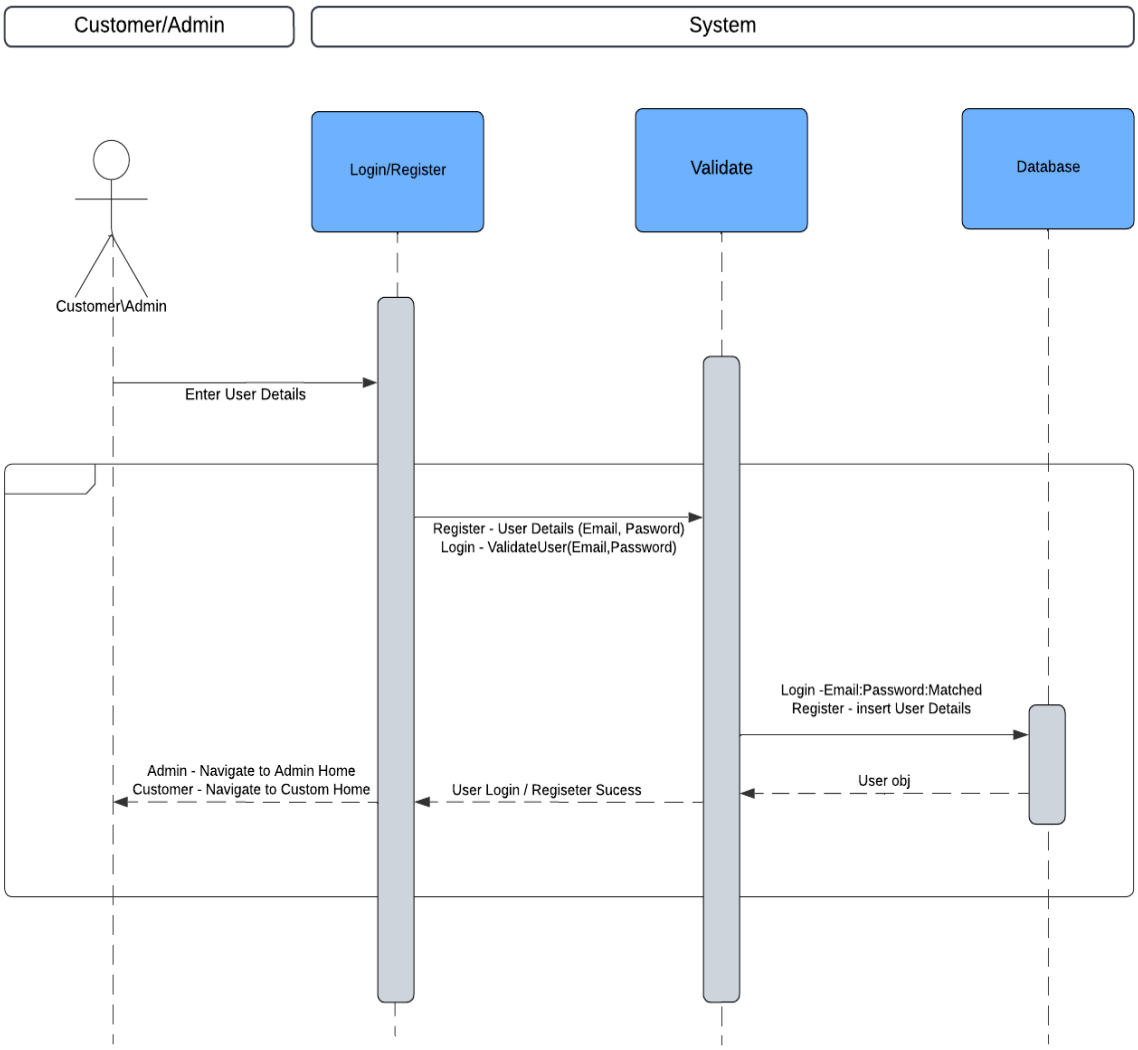


Figure 3: Sequence Diagram: Register/Login Use Case

Actors

- **User**: Provides credentials to register or log in.
- **System**: Processes user input, validates data, and handles registration or login.
- **Database**: Stores and retrieves user credentials.

Interactions

Login

1. User enters email and password and submits them.
2. System validates the format of the input.
3. System queries the Database for matching credentials.
4. Database returns the result to the System:
    - **Success**: User is navigated to the dashboard/home.
    - **Failure**: System displays an error message.

Register

1. User provides details (name, email, phone, password) and submits them.
2. System validates all input fields (e.g., email format, phone number, password strength).
3. System sends data to the Database for storage.
4. Database stores the data and returns the result:
    - **Success**: System confirms registration and redirects the User to the login screen.
    - **Failure**: System shows an error message for corrections.

Exception Handling

Invalid Input

- System prompts corrections if data (e.g., email or password) is invalid.

Login Failure

- System notifies: "Invalid email or password."

Registration Failure

- System alerts if email exists or validation fails, highlighting the error fields.

Actors

- **User**: Provides credentials to register or log in.
- **System**: Processes user input, validates data, and handles registration or login.
- **Database**: Stores and retrieves user credentials.
- **Customer**: Initiates the booking process by searching for available cars and providing booking details, such as dates and additional services.
- **Database (for Booking)**: Stores and retrieves car availability and booking details.

Interactions

Login

1. User enters email and password and submits them.
2. System validates the format of the input.
3. System queries the Database for matching credentials.
4. Database returns the result to the System:
    - **Success**: User is navigated to the dashboard/home.
    - **Failure**: System displays an error message.

Register

1. User provides details (name, email, phone, password) and submits them.
2. System validates all input fields (e.g., email format, phone number, password strength).
3. System sends data to the Database for storage.
4. Database stores the data and returns the result:
    - **Success**: System confirms registration and redirects the User to the login screen.
    - **Failure**: System shows an error message for corrections.

3.3.2 Booking a Car Use Case

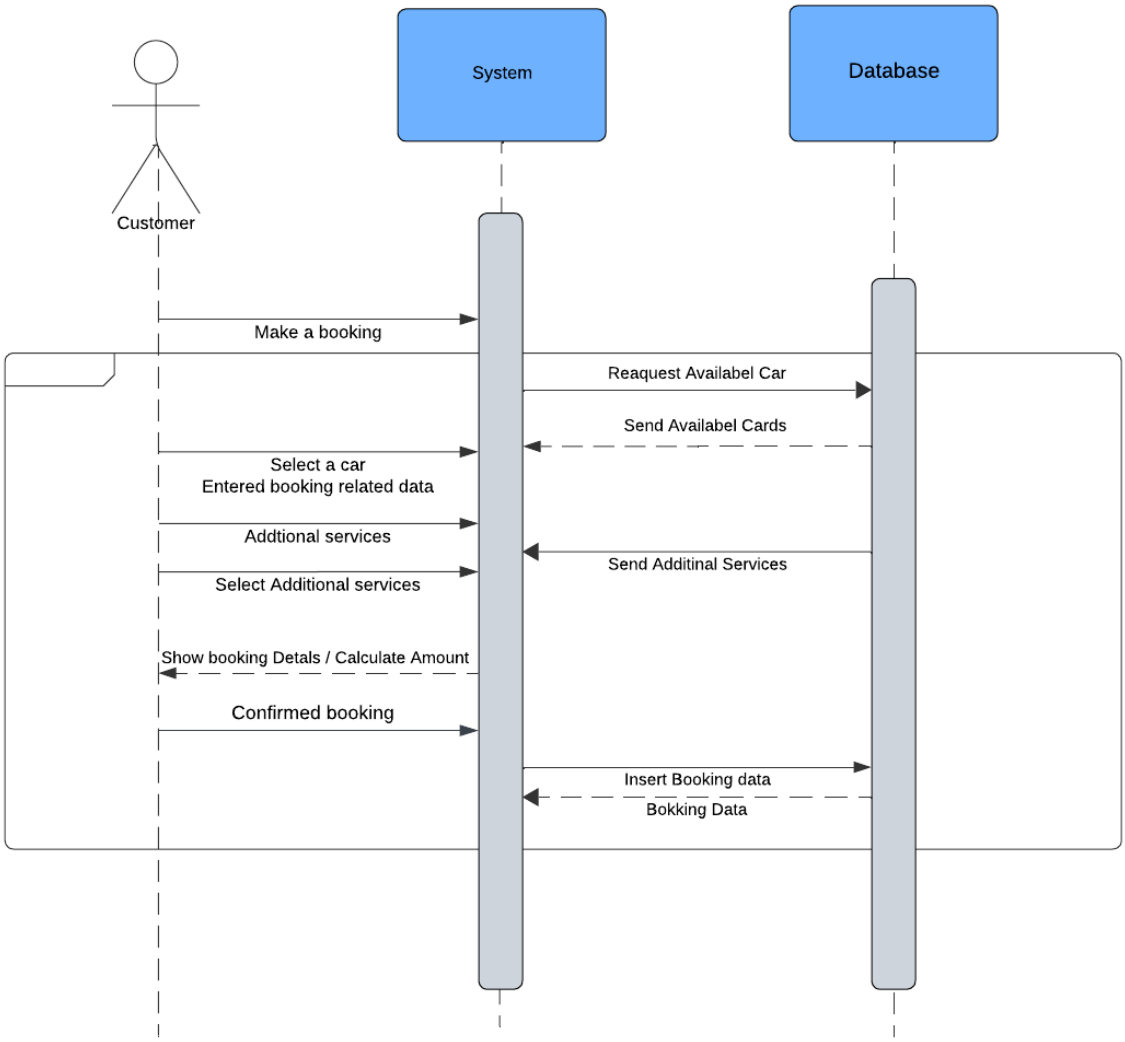


Figure 4: Sequence Diagram: Booking a Car Use Case

Select a Car

1. Customer searches for available cars.
2. System queries the Database for car availability.
3. Database returns a list of available cars to the System.
4. System displays the available cars to the Customer.

Input Booking Details

1. Customer selects a car and provides booking details, such as rental start and end dates.

2. System validates the input data.

3. saves the booking data locally.

Add Additional Services (Optional)

1. Customer optionally selects additional services.

2. System saves the selected services to the Database and updates the total booking cost.

3. Additional services are saved.

Display Booking Confirmation

1. System displays the final booking details, including car details, rental dates, additional services, and total cost, to the Customer.

Invalid Input

- System prompts corrections if data (e.g., email or password, booking details) is invalid.

Login Failure

- System notifies: “Invalid email or password.”

Registration Failure

- System alerts if email exists or validation fails, highlighting the error fields.

No Available Cars

- System show the Customer: “No cars available ”

Database Error

- System displays an error message

3.3.3 Manage Cars Use Case

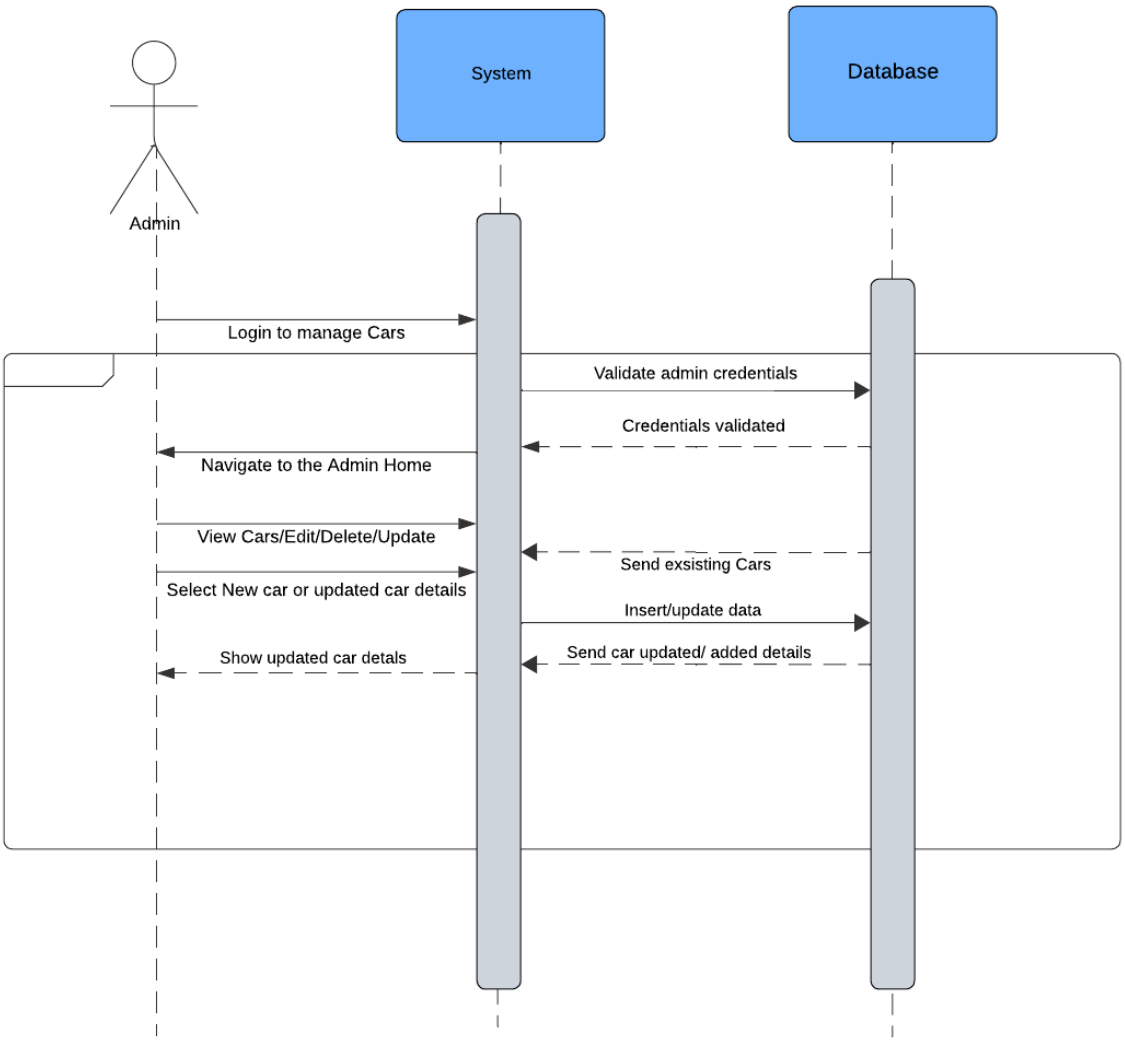


Figure 5: Sequence Diagram: Register/Login Use Case

Log in to the System

1. Admin logs in to access the car management functionality.
2. System validates the admin credentials via the Database.
3. If validated, the Admin gains access to the "Manage Cars" section.

View Cars

1. Admin requests to view all cars.
2. System fetches car records from the Database.
3. System displays the car records to the Admin.

Add/Edit/Delete Car Details

1. Admin provides details for a new car, modifies existing car information, or deletes a car record.
2. System validates the provided details.
3. System sends the request to the Database to save changes.
4. Database updates the records and confirms the changes.

Confirmation

1. System notifies the Admin of the success or failure of their action.

Invalid Input

- System displays an error: "Invalid car details. Please check your input."

Database Error

- System notifies the Admin: "Unable to save changes. Please try again later."

Unauthorized Access

- If the Admin's session expires, the System prompts them to log in again.

3.3.4 Approve/Reject Bookings Use Case

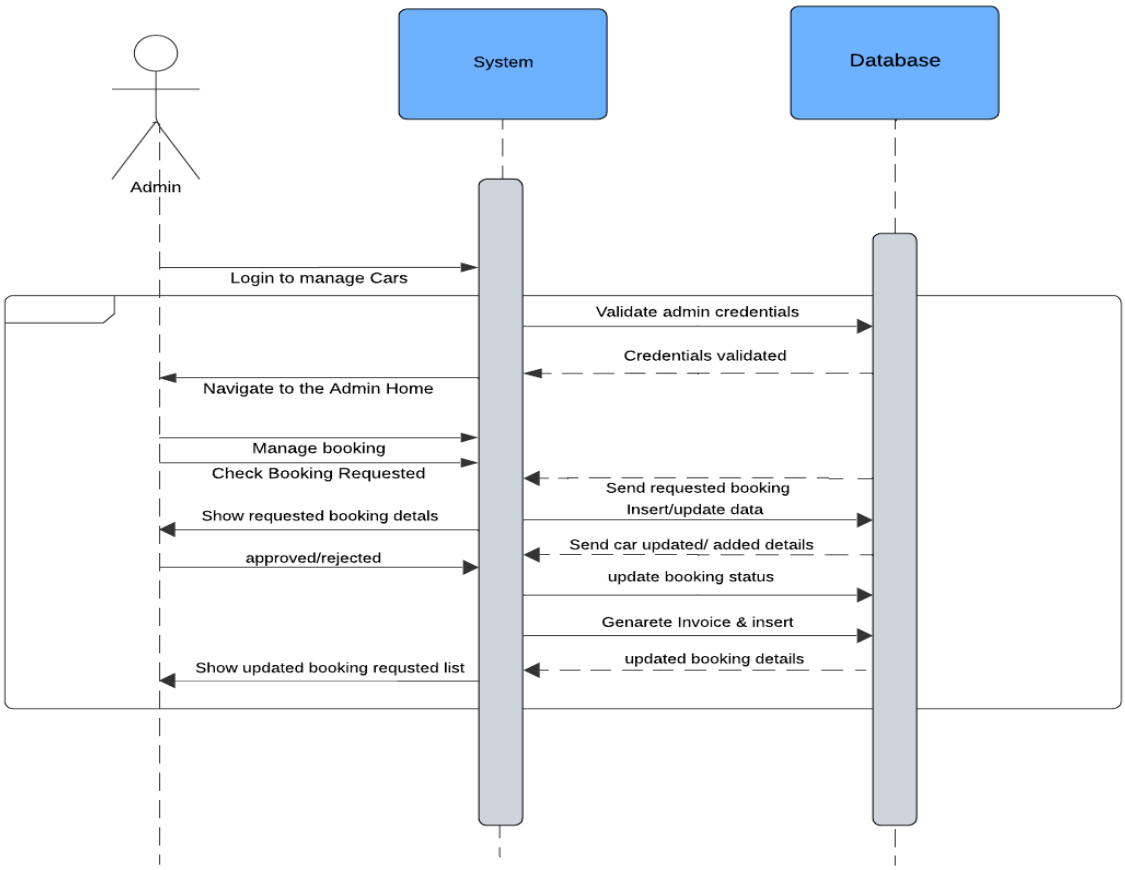


Figure 6: Sequence Diagram: Register/Login Use Case

Log In to the System

1. Admin logs in to access the booking management functionality.
2. System validates the admin credentials via the Database.
3. If validated, the Admin gains access to the "Manage Bookings" section.

View Booking Requests

1. Admin requests to view pending booking requests.
2. System queries the Database for pending booking records.
3. Database returns the booking details.
4. System displays the booking details to the Admin.

Approve or Reject Booking

1. Admin selects a booking and chooses to approve or reject it.
2. System updates the booking status in the Database accordingly.
3. If approved:
    - System generates an invoice.
    - System updates the customer record.
4. System notifies the Admin of the successful status update.

Notify Customer

1. If approved:
    - System notifies the customer via email.
    - System includes the invoice in the notification.
2. If rejected:
    - System informs the customer of the rejection.
    - System updates the booking status.

Exception Handling

Invalid Booking Request

- If the Admin tries to approve a booking with overlapping dates, the System notifies the Admin of the conflict.

Database Error

- If the Database fails to update the status, the System displays an error message: "Unable to update booking status. Please try again."

Unauthorized Access

- If the Admin's session expires, the System prompts them to log in again.

4 Features and Functionalities

4.1 Admin Features

- **Manage Users:** Add, update, deactivate users.
- **Manage Cars:** View, add, update, delete cars.
- **Manage Bookings:** Approve or reject bookings.
- **Manage Invoices:** Track payments and update statuses.

4.2 Customer Features

- **View Cars:** Browse available cars with details.
- **Book Cars:** Select a car, provide booking details, and confirm.
- **View Invoices:** Access and pay for bookings.

5 Database Design

5.1 Database Schema

5.1.1 UserType

Purpose: Defines roles for users, such as Admin and Customer.
Columns:

- `user_type_id` (Primary Key): Unique identifier for user types.
- `user_type`: Name of the user type.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.2 User

Purpose: Stores information about system users.
Columns:

- `user_id` (Primary Key): Unique identifier for users.
- `user_type_id` (Foreign Key): Links to `UserType(user_type_id)`.
- `user_name`, `user_email`, `user_phone_number`, `user_password`: User details.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.3 CarType

Purpose: Maintains categories of cars (e.g., Sedan, SUV).
Columns:

- `car_type_id` (Primary Key): Unique identifier for car types.
- `car_type`: Name of the car type.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.4 CarBrandModel

Purpose: Stores brand and model details of cars.
Columns:

- `car_brand_model_id` (Primary Key): Unique identifier for brand and model entries.
- `car_type_id` (Foreign Key): Links to `CarType(car_type_id)`.
- `brand_name`, `model_name`: Details of car brands and models.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.5 CarStatus

Purpose: Tracks the status of cars (e.g., Available, Rented).
Columns:

- `car_status_id` (Primary Key): Unique identifier for car statuses.
- `car_status_type`: Status type.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.6 Car

Purpose: Stores car inventory information.
Columns:

- `car_id` (Primary Key): Unique identifier for cars.
- `car_brand_model_id` (Foreign Key): Links to `CarBrandModel(car_brand_model_id)`.
- `car_status_id` (Foreign Key): Links to `CarStatus(car_status_id)`.
- Other details: `number_plate`, `model_name`, `daily_rate`, `year`, `mileage`, `min_rental_period`, `max_rental_period`.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.7 BookingStatus

Purpose: Tracks the status of bookings (e.g., Pending, Confirmed).
Columns:

- `booking_status_id` (Primary Key): Unique identifier for booking statuses.
- `booking_status_type`: Status type.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.8 Booking

Purpose: Manages customer bookings.
Columns:

- `booking_id` (Primary Key): Unique identifier for bookings.
- `user_id` (Foreign Key): Links to `User(user_id)`.
- `car_id` (Foreign Key): Links to `Car(car_id)`.
- `booking_status_id` (Foreign Key): Links to `BookingStatus(booking_status_id)`.
- Other details: `start_date`, `end_date`, `total_amount`, `note`.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.9 AdditionalServices

Purpose: Tracks optional services for bookings (e.g., GPS, child seat).
Columns:

- `additional_services_id` (Primary Key): Unique identifier for additional services.
- `services_description`, `services_amount`: Service details.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.10 BookingAdditionalServices

Purpose: Links additional services to bookings.
Columns:

- `booking_additional_charge_id` (Primary Key): Unique identifier for booking additional charges.
- `booking_id` (Foreign Key): Links to `Booking(booking_id)`.
- `additional_services_id` (Foreign Key): Links to `AdditionalServices(additional_services`
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.11 Invoice

Purpose: Manages payment records.
Columns:

- `invoice_id` (Primary Key): Unique identifier for invoices.
- `booking_id` (Foreign Key): Links to `Booking(booking_id)`.
- `user_id` (Foreign Key): Links to `User(user_id)`.
- Other details: `amount`, `payment_method`, `payment_date`, `is_paid`.
- `is_active`: Indicates if the record is active.
- `create_at` and `updated_at`: Timestamps for record creation and modification.

5.1.12 Entity Relationship Diagram

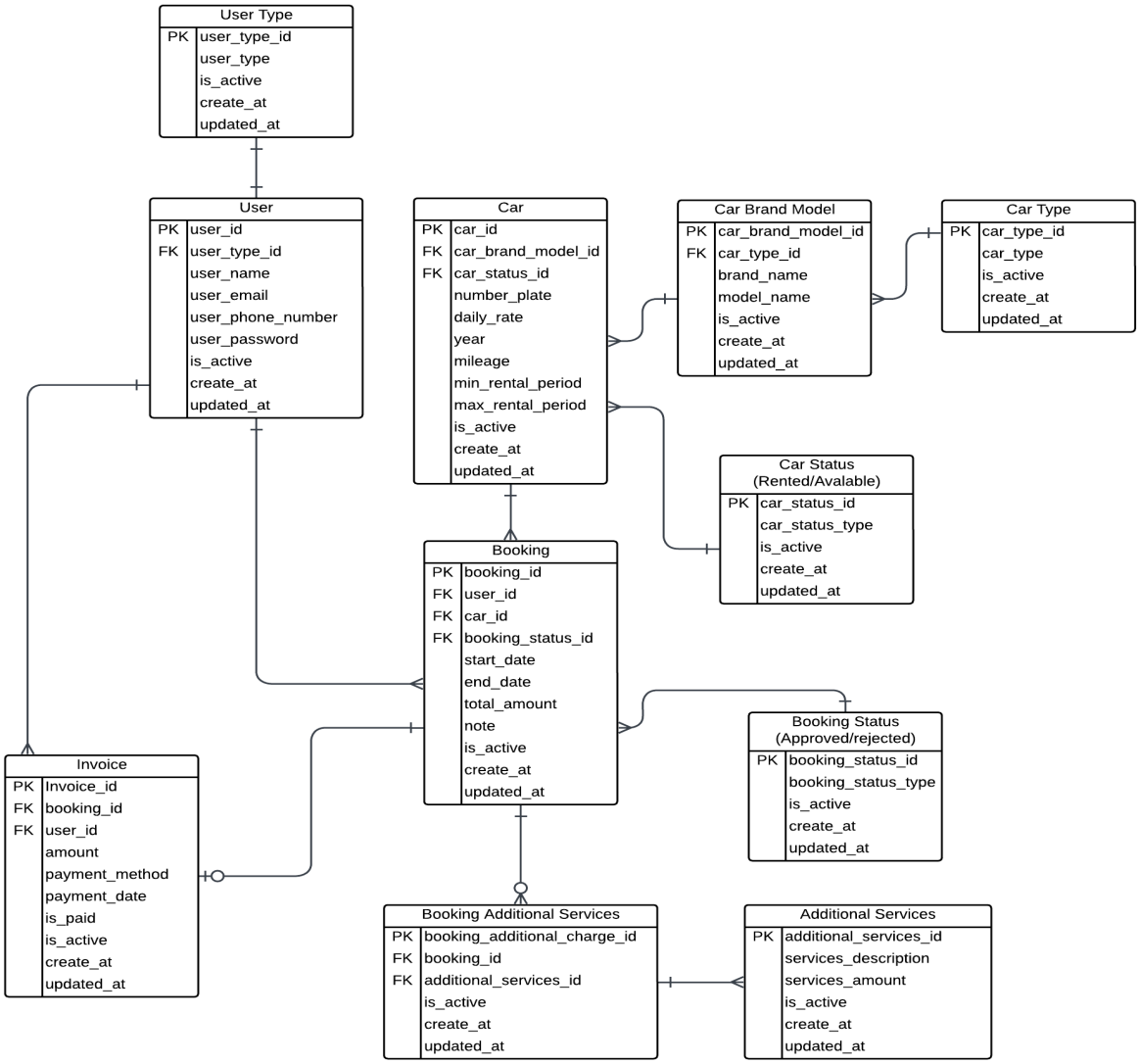


Figure 7: ERD Diagram

5.2 Relationships Between Tables

5.2.1 User Management

- **UserType → User:** `UserType(user_type_id)` is referenced in `User(user_type_id)` to define user roles.

5.2.2 Car Management

- **CarType → CarBrandModel:** `CarType(car_type_id)` is referenced in `CarBrandModel(car_t` to associate car brands/models with types.
- **CarBrandModel → Car:** `CarBrandModel(car_brand_model_id)` is referenced in `Car(car_brand_model_id)` to link cars to their brand/model.
- **CarStatus → Car:** `CarStatus(car_status_id)` is referenced in `Car(car_status_id)` to track car availability.

5.2.3 Booking System

- **User → Booking:** `User(user_id)` is referenced in `Booking(user_id)` to associate bookings with customers.
- **Car → Booking:** `Car(car_id)` is referenced in `Booking(car_id)` to track the booked car.
- **BookingStatus → Booking:** `BookingStatus(booking_status_id)` is referenced in `Booking(booking_status_id)` to manage booking progress.

5.2.4 Additional Services

- **Booking → BookingAdditionalServices:** `Booking(booking_id)` is referenced in `BookingAdditionalServices(booking_id)` to link additional services to bookings.
- **AdditionalServices → BookingAdditionalServices:** `AdditionalServices(additional_ser` is referenced in `BookingAdditionalServices(additional_services_id)`.

5.2.5 Invoices

- **Booking → Invoice:** `Booking(booking_id)` is referenced in `Invoice(booking_id)` for payment tracking.
- **User → Invoice:** `User(user_id)` is referenced in `Invoice(user_id)` to track customer payments.

5.2.6 Table Relationships Summery

- **Users ↔ Booking:** One user can have many bookings.
- **Cars ↔ Booking:** One car can have multiple bookings.
- **Booking ↔ Invoice:** Each booking can have one associated invoice.

- **Users ↔ Invoice:** One user can have multiple invoices.
- **UserType ↔ Users:** One user type can be associated with multiple users.
- **CarType ↔ CarBrandModel:** One car type can be associated with multiple car brands and models.
- **CarBrandModel ↔ Cars:** One car brand/model can have multiple cars.
- **CarStatus ↔ Cars:** One car status can be associated with multiple cars.
- **BookingStatus ↔ Booking:** One booking status can be associated with multiple bookings.
- **Booking ↔ BookingAdditionalServices:** One booking can have multiple additional services.
- **AdditionalServices ↔ BookingAdditionalServices:** One additional service can be associated with multiple bookings.

5.3 Database Screenshots

5.3.1 UserType Table

user_type_id	user_type	is_active	create_at	updated_at
2	User	1	1738468031	1738468031
1	Admin	1	1738468031	1738468031
NULL	NULL	NULL	NULL	NULL

Figure 8: serType Table

5.3.2 User Table

user_id	user_type_...	user_name	user_email	user_phone_numb...	user_password	is_active	create_at	updated_at
1	1	Alice Smith	admin@admin.com	1234567890	$2b$12$v/stMUOZxVsP3064XTz.R.r/nPS5dT/lHCFr.vJ.X7Tqxif6DZ.2C	1	1738468031	1738468031
2	2	Bob Johnson	user@user.com	0987654321	$2b$12$tLCKGEAXNG6Oves0cgWro.j9Ly.77Ng8D4.nm0XsYtqgkZzZ8IYXi	1	1738468031	1738468031
3	2	Charlie Brown	charlie@brown.com	1122334455	$2b$12$lnb6Sr1rwHTN3rqOAZk40eC5jeGWzUY5URX5sJUtlIPmq8ml8x1KS	1	1738468031	1738468031
4	2	Diana Prince	diana@prince.com	9988776655	$2b$12$DOoY/U7K/vrYUYgDQAtVPerahy1ASV5/.2rhVyLgfJcKgRN/lP16G	1	1738468031	1738468031
5	2	Evan Parker	evan@parker.com	6677889900	$2b$12$q8FlZCxILX5TR0FshQyaqOaahpWZlyR96E7oUya8ThzMuj.RLMExy	1	1738468031	1738468031
6	2	Thanushka Praveen	thanushkawickramarachchi@gmail.com	+64204489222	$2b$12$lJ.QhlicvnD2/ppPbO6sGehY7BawAN1zuGpoNLJVp/Al53FT/tr0.	1	1738479339	1738479339

Figure 9: User Table

5.3.3 CarType Table

car_type_id	car_type	is_active	create_at	updated_at
1	Car	1	1738468031	1738468031
2	Motorcycle	1	1738468031	1738468031
3	Truck	1	1738468031	1738468031
4	Bus	1	1738468032	1738468032
5	Van	1	1738468032	1738468032
6	SUV	1	1738468032	1738468032
NULL	NULL	NULL	NULL	NULL

Figure 10: CarType Table

5.3.4 CarBrandModel Table

car_brand_model...	car_type_id	brand_name	model_name	is_active	create_at	updated_at
1	1	Toyota	RAV4	1	1738468032	1738468032
2	1	Toyota	Corolla	1	1738468032	1738468032
3	1	Honda	Civic	1	1738468032	1738468032
4	2	Harley-Davidson	Sportster	1	1738468032	1738468032
5	2	Yamaha	MT-07	1	1738468032	1738468032
6	3	Volvo	FH16	1	1738468032	1738468032
7	3	Mercedes-Benz	Actros	1	1738468032	1738468032
8	4	Volvo	9400	1	1738468032	1738468032
9	4	Scania	Touring	1	1738468032	1738468032
10	5	Ford	Transit	1	1738468032	1738468032
11	5	Mercedes-Benz	Sprinter	1	1738468032	1738468032
12	6	Jeep	Grand Cher...	1	1738468032	1738468032
13	6	Ford	Explorer	1	1738468032	1738468032
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 11: CarBrandModel Table

5.3.5 CarStatus Table

car_status_Id	car_status_ty...	is_active	create_at	updated_at
1	Available	1	1738468032	1738468032
2	Unavailable	1	1738468032	1738468032
NULL	NULL	NULL	NULL	NULL

Figure 12: CarStatus Table

5.3.6 Car Table

car_Id	car_brand_model...	car_status_Id	number_plate	model_name	daily_rate	year	mileage	min_rental_peri...	max_rental_peri...	is_active	create_at	updated_at
1	1	1	ABC-1234	Toyota RAV4	120.50	2021	10000	1	30	1	1738468032	1738468032
2	2	1	XYZ-5678	Toyota Corolla	100.00	2020	15000	1	30	1	1738468032	1738468032
3	3	2	LMN-3456	Honda Civic	110.00	2019	20000	2	30	0	1738468032	1738478338
4	4	1	DEF-7890	Ford Explorer	130.75	2022	5000	1	30	1	1738468032	1738468032
5	5	1	ABB-1212	Yamaha - 1111	99.0	2023	1222	5	25	1	1738468032	1738477568
6	1	1	ACD-3242	Rave - MZ	139.0	2004	19388	4	30	1	1738469412	1738469412
7	3	1	ABC-1221	Civic - FD2 RS	200.0	2022	20000	10	30	1	1738469638	1738476290
8	5	1	ALM-1234	Yamaha - LXR	98.0	2022	50050	5	20	1	1738475408	1738475408
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

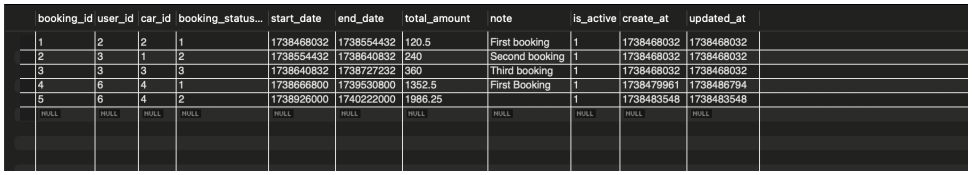
Figure 13: Car Table

5.3.7 BookingStatus Table

booking_status...	booking_status_t...	is_active	create_at	updated_at
1	Confirmed	1	1738468032	1738468032
2	Pending	1	1738468032	1738468032
3	Cancelled	1	1738468032	1738468032
NULL	NULL	NULL	NULL	NULL

Figure 14: BookingStatus Table

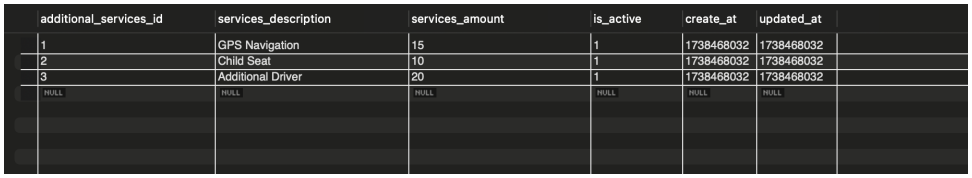
5.3.8 Booking Table



booking_id	user_id	car_id	booking_status...	start_date	end_date	total_amount	note	is_active	create_at	updated_at
1	2	2	1	1738468032	1738554432	120.5	First booking	1	1738468032	1738468032
2	3	1	2	1738554432	1738640832	240	Second booking	1	1738468032	1738468032
3	3	3	3	1738640832	1738727232	360	Third booking	1	1738468032	1738468032
4	6	4	1	1738666800	1739530800	1352.5	First Booking	1	1738479961	1738486794
5	6	4	2	1738926000	1740222000	1986.25		1	1738483548	1738483548
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 15: Booking Table

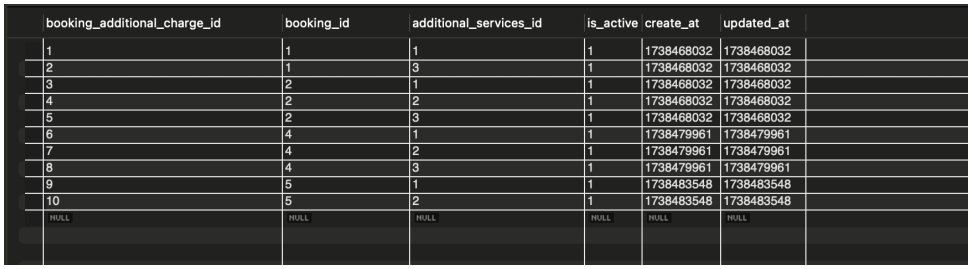
5.3.9 AdditionalServices Table



additional_services_id	services_description	services_amount	is_active	create_at	updated_at
1	GPS Navigation	15	1	1738468032	1738468032
2	Child Seat	10	1	1738468032	1738468032
3	Additional Driver	20	1	1738468032	1738468032
NULL	NULL	NULL	NULL	NULL	NULL

Figure 16: AdditionalServices Table

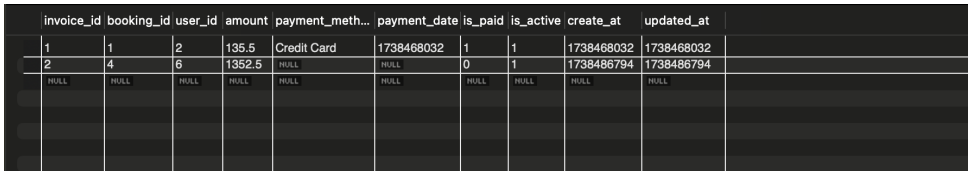
5.3.10 BookingAdditionalServices Table



booking_additional_charge_id	booking_id	additional_services_id	is_active	create_at	updated_at
1	1	1	1	1738468032	1738468032
2	1	3	1	1738468032	1738468032
3	2	1	1	1738468032	1738468032
4	2	2	1	1738468032	1738468032
5	2	3	1	1738468032	1738468032
6	4	1	1	1738479961	1738479961
7	4	2	1	1738479961	1738479961
8	4	3	1	1738479961	1738479961
9	5	1	1	1738483548	1738483548
10	5	2	1	1738483548	1738483548
NULL	NULL	NULL	NULL	NULL	NULL

Figure 17: BookingAdditionalServices Table

5.3.11 Invoice Table



invoice_id	booking_id	user_id	amount	payment_meth...	payment_date	is_paid	is_active	create_at	updated_at
1	1	2	135.5	Credit Card	1738468032	1	1	1738468032	1738468032
2	4	6	1352.5	NULL	NULL	0	1	1738486794	1738486794
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 18: Invoice Table

NOTE: All database-related queries can be checked in the `sql_statement` file.

6 System Workflow

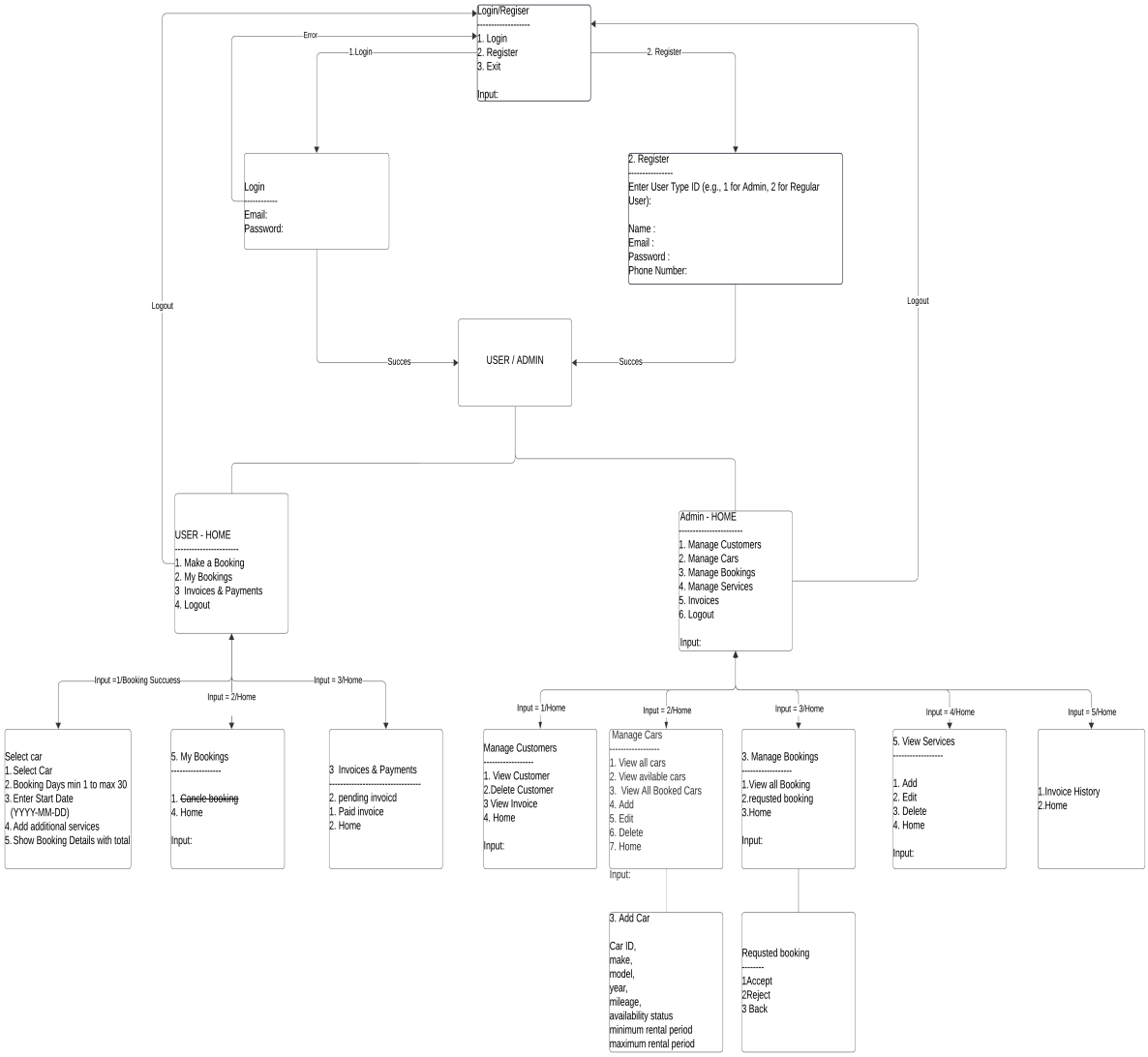


Figure 19: System Workflow

6.1 Customer Workflow

1. Register and log in.
2. Browse available cars.
3. Make a booking.
4. Add optional services.
5. Confirm and pay.

6.2 Admin Workflow

1. Log in as Admin.
2. Manage users, cars, and bookings.
3. Approve or reject bookings.
4. Monitor payments and invoices.

7 Architecture Patterns

- **MVC: Model-View-Controller (MVC)** Architecture.
- **Model:** Data & Business Logic - The Model represents the data, business logic, and rules of the application. It communicates with the database and processes the data.
- **View:** User Interface (UI) - The View is responsible for displaying data and rendering the user interface. It interacts with the Model indirectly through the Controller.
- **Controller:** Controller (C) - Handles User Interaction - The Controller acts as a mediator between the Model and the View. It processes user input, retrieves data from the Model, and updates the View accordingly.

8 Design Patterns

- **Singleton Pattern**: Ensures that only one database connection instance exists at any given time, maintaining consistency and preventing redundant connections.

9 Software Design Principles

- **Object-Oriented Programming:** Encapsulation, inheritance, polymorphism, and abstraction.
- **Modularity:** Logical separation for users, cars, bookings, and invoices.

10 Conclusion

The Car Rental System effectively automates rental processes, enhancing user experience and operational efficiency. With a scalable design, the system is prepared for future enhancements like mobile app integration and advanced analytics.