CAR RENTAL SYSTEM

Methodology / Application Development

The development of the Car Rental System followed a structured and user-centered approach. The aim was to design a system that not only automates the process of renting a car but also enhances the experience for both users and administrators. This chapter walks through the methodology we followed, from initial planning to final testing and deployment, giving a behind-the-scenes look at how the project evolved.

3.1 Project Overview

The Car Rental System is a web-based application that allows users to search for available cars, book rentals for specific time durations, and make payments online. At the same time, it provides administrators with a dashboard to manage users, vehicle inventory, and booking records efficiently.

Our goal was to transition from the manual, paperwork-heavy car rental process to a streamlined, secure, and user-friendly digital platform that reduces time and errors while improving service availability.

3.2 Development Methodology: Agile Model

For this project, we adopted the **Agile methodology**, specifically the **Scrum** framework. Agile allowed us to break down the development cycle into manageable sprints (typically two weeks each) where we planned, developed, tested, and reviewed features iteratively. The choice of Agile gave us the flexibility to adapt to new requirements as they emerged during user feedback sessions.

Each sprint included:

- **Sprint Planning**: Identifying key tasks and assigning them.
- **Daily Standups**: Quick progress checks and blockers.
- **Sprint Review**: Demo of completed features.
- **Sprint Retrospective**: Reflecting on what went well and what could improve.

3.3 Technology Stack

Choosing the right technology stack was critical to ensuring smooth development and scalability. Here's what we used:

- **Frontend**: HTML5, CSS3, JavaScript
- **Database**: SQLite (for simplicity during development)
- **Version Control**: Git and GitHub

3.4 Application Architecture

The architecture of the Car Rental System follows a three-tier model:

1. **Presentation Layer (Frontend)**
   This layer handles user interaction. It includes web pages for login, registration, car listings, booking forms, and the admin dashboard.

2. **Business Logic Layer (Backend)**
   This layer manages user input, processes booking logic, verifies user sessions, and handles system rules such as availability checking and cost calculation.

3. **Data Layer (Database)**
   This layer stores all application data including user credentials, vehicle details, booking records, and admin data.

3.5 Database Design

The heart of the application is its database, which contains multiple tables as follows:

- **User Table**: Stores registered user details.
- **Vehicle Inventory Table**: Stores available cars and their status.
- **Booking Table**: Stores rental history and live bookings.
- **Admin Table**: Stores credentials of system administrators.
- **Payment Table (optional)**: Placeholder for transaction logs in future integration.

Each table is connected using foreign keys to maintain relational integrity. For example, the `booking` table references both `user_id` and `vehicle_id` to form a link between who booked what car.

3.6 Application Modules

3.6.1 User Registration & Login

We created a secure sign-up and login system where user passwords are hashed before storing them. Flask-Login was used to manage sessions securely, ensuring users stay logged in without compromising security.

3.6.2 Car Browsing & Booking

Users can browse the list of available vehicles with filters like vehicle type and price. Once a user selects a car, they can book it for a specified time frame. The system checks for overlapping bookings and availability before confirming the reservation.

3.6.3 Admin Panel

Admins can:

- View and manage users
- Add or remove cars from the system
- See current and past bookings
- Change booking statuses (e.g., active, completed)

The admin dashboard is protected by an extra layer of authentication and role verification.

3.6.4 Error Handling

Error messages were implemented across the app to ensure user clarity. For instance, if a user tries to book an already rented car, the system provides a clear message with alternate suggestions.

3.7 User Interface Design

The UI was designed to be clean, intuitive, and mobile-responsive. Bootstrap helped achieve consistency and responsiveness across pages. Key design considerations included:

- Clear call-to-action buttons
- Easy-to-navigate car listings
- Minimalist admin dashboard with analytics-ready layout

We also maintained design contrast and accessibility for users with visual impairments by following WCAG guidelines wherever possible.

3.8 Testing and Debugging

The application underwent multiple layers of testing:

3.8.1 Unit Testing

Each module, such as user login or booking form submission, was tested independently to ensure it works in isolation.

3.8.2 Integration Testing

We checked if modules worked together — for example, after booking a car, does the inventory update correctly?

3.8.3 User Testing

A few peers tested the system and provided feedback on usability, bugs, and suggestions for improvement. This helped us iron out issues that weren’t obvious during development.

3.8.4 Error Logging

Python's `logging` module was used to track errors and exceptions, which was helpful during debugging.

3.9 Deployment

The project was initially run locally for development and testing purposes. For future expansion, we plan to deploy it on a cloud platform like Heroku or AWS with a shift from SQLite to PostgreSQL for better scalability.

3.10 Challenges Faced

- **Date Conflict Logic**: Implementing booking conflict checks for overlapping dates required careful logic and testing.
- **Session Management**: Ensuring user sessions remained secure and valid after multiple redirects.
- **Design Uniformity**: Making sure the UI was consistent across devices and browsers took some extra fine-tuning.

3.11 Future Enhancements

- **Online Payments**: Integration with Razorpay or Stripe to handle rental payments.
- **Email Notifications**: Automatic email confirmation upon booking or cancellations.
- **Google Maps API**: To show pickup and drop-off locations.
- **Mobile App**: A Flutter or React Native app for better accessibility.

Conclusion

The Car Rental System is a complete, modular, and scalable application that bridges the gap between traditional car rental methods and modern online platforms. The development process was guided by the principles of clarity, simplicity, and responsiveness. Using the Agile methodology allowed us to continuously improve the product through short cycles and feedback. The resulting system is not just a product of code, but of thoughtful planning, real-time problem-solving, and a strong commitment to improving user experience.