**An MVP for a car rental service**

****

**Sections**

1. **Overview of the MVP**
2. **Architecture**
3. **APIs and Methods**
4. **Data Modelling**

**Overview of the MVP**

1. **Vision**: To provide an accessible and convenient car rental service for users who need short-term transportation solutions.

**2. Core Features:**

**User Registration/Login**: Allow users to register or login to their accounts.

**Browse and Reserve Cars**: Users can browse available cars and reserve one for a specific date and time.

**User Profile Management**: Users can update their profile information and payment details.

**Basic Reservation Management**: Users can view and manage their current and upcoming reservations.

**Notifications**: Send basic notifications to users confirming their reservation and reminding them of upcoming rentals.

**Feedback and Support**: Allow users to provide feedback and access basic support.

**3. Prioritization:**

1. Must-have:

- User registration/login

- Browse and reserve cars

- User profile management

1. Should-have:

- Basic reservation management

- Notifications

1. Could-have:

- Advanced reservation options (e.g., additional equipment, insurance)

- Integration with map services for car location

1. Won't-have:

- Advanced analytics on user behaviour

**4. User Stories:**

- As a user, I want to register for an account so I can rent a car.

- As a user, I want to browse available cars and select one for rental.

- As a user, I want to reserve a car for a specific date and time.

- As a user, I want to update my profile information and payment details.

- As a user, I want to view and manage my current and upcoming reservations.

- As a user, I want to receive notifications confirming my reservation and reminding me of upcoming rentals.

- As a user, I want to provide feedback or contact support if I encounter issues.

5. **Wireframes or Mock-ups**:

- Simple interface showing login/register, car browsing, reservation form, and user profile.

6. **Technical Requirements**:

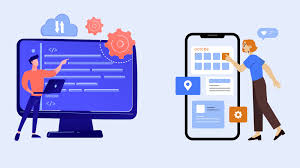
- Web or mobile application development.

- Backend server for user authentication, data storage, and processing reservation requests.

- Database for storing user information and reservations.

- Integration with payment gateway for handling transactions.

- Push notification service for sending notifications.



**7. Build the MVP:**

- Develop a basic web or mobile application with core features.

- Implement user registration/login functionality.

- Create a catalogue of available cars with basic details.

- Enable users to reserve a car for a specific date and time.

- Develop user profile management functionality.

- Implement basic reservation management features.

- Set up notifications for confirming reservations and reminders.

- Provide a basic feedback/support interface.

**8. Test and Iterate:**

- Test the MVP with users to ensure it meets basic functionality requirements.

- Gather feedback on user experience and any issues encountered.

- Iterate on the MVP to fix bugs and improve usability based on feedback.

**9. Launch and Measure:**

- Launch the MVP car rental service in a limited area or test environment.

- Measure key metrics such as user registration, reservation rates, and user satisfaction.

- Gather data on user interactions and any technical issues.

**10. Refine and Scale:**

- Based on feedback and data, refine the MVP by addressing any issues and adding features like advanced reservation options or better notifications.

- Gradually scale up the car rental service, expanding to more locations or adding more features, while continuing to gather feedback and iterate on improvements.

### Architecture

**Client Interface:**

-This is the user-facing part of the system, where users interact with the application.

-Web Interface: For users accessing the system via a web browser.

-Mobile Interface: For users accessing the system via a mobile app.

**Server:**

-This is where the core logic of the application resides.

-Web Server: Handles incoming requests from the client and serves web pages or data.

-Application Server: Implements the business logic of the application.

-Database Server: Stores and manages application data.

**Database:**

-This is where all the application data is stored.

-User Data: Stores information about users such as usernames, passwords, and profiles.

-Car Data: Stores information about available cars for rental.

-Reservation Data: Stores information about user reservations.

**External Services:**

-These are third-party services integrated into the application.

-Payment Gateway: Handles payment processing for rental transactions.

-Notification Service: Sends out notifications to users regarding their reservations.

**Client-Server Communication:**

-Communication between the client and server occurs via HTTP requests and responses.

-REST API: The server exposes a RESTful API that the client uses to interact with the server.

### APIs and Methods

1. **API Routes for Web Client to Web Server Communication:**
2. **/api/cars**

* -GET: Returns a list of available cars for rental.
* -POST: Adds a new car to the inventory.

1. **/api/reservations**

* -GET: Returns a list of reservations for a specific user.
* -POST: Creates a new reservation for a user.

1. **/api/users**

* -GET: Returns user information based on user ID.
* -POST: Creates a new user.

1. **Endpoints for External Clients:**
2. **User Class**

* -get\_user\_info(user\_id: int) -> dict: Retrieves user information based on user ID.
* -create\_user(user\_data: dict) -> int: Creates a new user and returns the user ID.

1. **Reservation Class**

* -get\_user\_reservations(user\_id: int) -> list: Retrieves reservations for a specific user.
* -create\_reservation(reservation\_data: dict) -> int: Creates a new reservation and returns the reservation ID.

**3. Car Class**

* -get\_available\_cars() -> list: Retrieves a list of available cars for rental.
* -add\_car(car\_data: dict) -> int: Adds a new car to the inventory and returns the car ID.

1. **3rd Party APIs:**
2. **Stripe API**

* Endpoints:
* POST /v1/charges: Process payments for rental transactions.
* GET /v1/customers/{customer\_id}: Retrieve customer information.
* POST /v1/customers: Create a new customer for payment processing.

1. **Google Maps API**

* Endpoints:
* GET /maps/api/geocode/json: Convert addresses into geographic coordinates.
* GET /maps/api/directions/json: Calculate directions between locations

1. **Twilio API**

* Endpoints:
* POST /Messages: Send SMS notifications to users.
* GET /Messages/{message\_id}: Retrieve message details.
* POST /Calls: Make voice calls to users.

### Data Modelling

1. **User**:
   * Represents users of the car rental service.
   * Attributes:
     + **user\_id**: Primary key, unique identifier for each user.
     + **username**: User's username for logging in.
     + **password**: User's password (encrypted for security).
     + **email**: User's email address.
     + **phone\_number**: User's phone number.
     + **created\_at**: Timestamp of when the user account was created.
2. **Car**:
   * Represents available cars for rental.
   * Attributes:
     + **car\_id**: Primary key, unique identifier for each car.
     + **brand**: Car brand (e.g., Toyota, Honda).
     + **model**: Car model (e.g., Camry, Civic).
     + **year**: Year of the car.
     + **color**: Color of the car.
     + **license\_plate**: Car's license plate number.
     + **availability**: Boolean indicating whether the car is available for rental.
     + **created\_at**: Timestamp of when the car was added to the inventory.
3. **Reservation**:
   * Represents reservations made by users.
   * Attributes:
     + **reservation\_id**: Primary key, unique identifier for each reservation.
     + **user\_id**: Foreign key referencing the User table.
     + **car\_id**: Foreign key referencing the Car table.
     + **start\_date**: Start date of the reservation.
     + **end\_date**: End date of the reservation.
     + **status**: Status of the reservation (e.g., pending, confirmed, canceled).
     + **created\_at**: Timestamp of when the reservation was made.

**Relationships:**

* Each User can have multiple Reservations (One-to-Many).
* Each Car can have multiple Reservations (One-to-Many).