Hackathon Day2 Task2
Technical Plan Rental E-commerce

1. Platform Overview

The Car Rental Marketplace is an online platform that connects car rental providers with users. Users can book cars for different periods (hours, days, or weeks), choose from various car models, and pay via integrated payment systems.

Key Features:

- **Car Listings**: Display a variety of cars available for rent.
- **Booking**: Users can book a car for a specific date range.
- **Payment Integration**: Integration with payment systems like Stripe or PayPal,Jazzcash,Easypaisa.
- **Reviews**: Customers can leave reviews after renting.
- **User Accounts**: Users can sign up, log in, and manage bookings.

2. Frontend Development Plan (UI/UX)

Frontend Technologies:

- **Next.js**: For SSR (Server-Side Rendering) and optimized performance.
- **React**: For dynamic page rendering and state management.
- **Tailwind CSS**: For building responsive and modern UI components.
- **Axios**: To handle API requests from the backend.

Key Pages:

- **Home Page**: Overview of available cars, search filters (car type, price, availability).
- **Car Listings Page**: Display a list of cars with filtering and sorting options.
- **Car Detail Page**: Detailed view of each car (including pricing, availability, and booking options).
- **Checkout Page**: Collect user information, show car rental details, and allow payment.

- **User Dashboard**: Users can view their past bookings, reviews, and edit personal info.

3. Backend Development Plan

Backend Technologies:

- **Node.js (Express)**: For handling the server-side logic and API routes.
- **Sanity CMS**: To manage car listings, availability, and other dynamic content.
- **MongoDB**: For storing user and booking data (Document-based storage).
- **JWT (JSON Web Tokens)**: For user authentication and session management.

API Endpoints:

1. **/api/cars** (GET) – Fetch all cars with filtering options.
2. **/api/car/{carId}** (GET) – Fetch details for a specific car.
3. **/api/booking** (POST) – Create a new booking.
4. **/api/payment** (POST) – Process payments using Stripe or PayPal.
5. **/api/reviews** (POST) – Add reviews for a car.
6. **/api/user** (GET/POST) – Get or update user information.

User Authentication:

- **Login**: User will log in via email/password. Token-based authentication will be used.
- **Signup**: New users can register by providing their email, phone number, and address.
- **Password Reset**: Email link for password reset.

4. Data Models (Database Schema)

The Car Rental Marketplace will use MongoDB to store the data. Here are the data models and their relationships:

User Model:

```
const UserSchema = new mongoose.Schema({

  first_name: String,
```

```
  last_name: String,
  email: { type: String, unique: true },
  phone_number: String,
  address: String,
  created_at: { type: Date, default: Date.now },
});
```

Car Model:

```
const CarSchema = new mongoose.Schema({
  make: String,
  model: String,
  type: String, // e.g., Sedan, SUV
  price_per: Number, // Price per day/hour
  availability: Boolean,
  images: [String], // Array of image URLs
});
```

Booking Model:

```
const BookingSchema = new mongoose.Schema({
  user_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  car_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Car' },
  start_date: Date,
  end_date: Date,
  total_price: Number,
```

```
  status: { type: String, enum: ['pending', 'confirmed', 'cancelled'], default: 'pending' },
  created_at: { type: Date, default: Date.now },
});
```

Payment Model:

```
const PaymentSchema = new mongoose.Schema({
  booking_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Booking' },
  payment_method: String,
  amount: Number,
  payment_date: { type: Date, default: Date.now },
  payment_status: { type: String, enum: ['completed', 'pending', 'failed'], default: 'pending' },
});
```

Review Model:

```
const ReviewSchema = new mongoose.Schema({
  user_id: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  car_id: { type: mongoose.Schema.Types.ObjectId, ref: 'Car' },
  rating: { type: Number, min: 1, max: 5 },
  comment: String,
  created_at: { type: Date, default: Date.now },
});
```

5. API Design

Here are the detailed API routes for the platform:

1. /api/cars (GET)

- **Purpose**: Fetch all available cars.
- **Query Parameters**: make, type, price, availability
- **Response**:

```
[
  {
    "car_id": 101,
    "make": "Toyota",
```

```
  "model": "Corolla",
  "price_per": 40,
  "availability": true,
  "images": ["toyota_corolla1.jpg", "toyota_corolla2.jpg"]
 }
]
```


2. /api/car/{carId} (GET)

- **Purpose**: Fetch details for a specific car.
- **Response**:


```
{
  "car_id": 101,
  "make": "Toyota",
  "model": "Corolla",
  "type": "Sedan",
  "price_per": 40,
  "availability": true,
  "images": ["toyota_corolla1.jpg", "toyota_corolla2.jpg"],
  "description": "A reliable and fuel-efficient sedan perfect for city drives."}
```


3. /api/booking (POST)

- **Purpose**: Create a new booking.
- **Payload**:


```
{
  "user_id": "1",
  "car_id": "101",
  "start_date": "2025-01-15",
  "end_date": "2025-01-16",
  "total_price": 80}
```

- **Response**:

```
{
 "booking_id": "123",
 "status": "confirmed"}
```

4. /api/payment (POST)

- **Purpose**: Process payment for the booking.
- **Payload**:

```
{
 "booking_id": "123",
 "payment_method": "Credit Card",
 "amount": 80}
```

- **Response**:

```
{
 "payment_status": "completed",
 "payment_date": "2025-01-10"}
```

5. /api/reviews (POST)

- **Purpose**: Submit a review for a car.
- **Payload**:

```
{
 "user_id": "1",
 "car_id": "101",
 "rating": 5,
 "comment": "Great car, easy booking process!"}
```

- **Response**:

```
{
 "review_id": "123",
```

"status": "submitted"}

6. Payment Integration

For payments, we will integrate Stripe or PayPal or JazzCash or EasyPaisa Here's how the integration will work:

1. **Create a Payment Intent**: When the user proceeds to checkout, a payment intent is created via the Stripe API.
2. **Payment Confirmation**: After successful payment, the system updates the booking status and confirms the rental.
3. **Security**: Use **HTTPS** for secure communication between frontend and backend.

7. Security Considerations

- **JWT Authentication**: For secure user login and token-based sessions.
- **Data Encryption**: All sensitive data like user information and payment details will be encrypted.
- **Role-Based Access Control (RBAC)**: Ensure only authorized users can modify car data, bookings, etc.

8. Deployment and Hosting

For hosting, we can use services like:

- **Frontend**: Vercel or Netlify.
- **Backend**: AWS (EC2), DigitalOcean, or Heroku.
- **Database**: MongoDB Atlas for cloud-based database hosting.

9. Future Improvements

- **Car Availability**: Real-time car availability updates.
- **Push Notifications**: Send booking reminders and promotional offers.
- **Advanced Search Filters**: Add features like geolocation-based search, car categories, etc.

10. Visual Diagram: How the Project Works

Below is the sequence diagram of how the Car Rental Marketplace operates:

Car Rental Marketplace Flow (Table Representation)

Steps	Action	Frontend (User)	Backend API	External Payment Gateway
1	Visit Homepage	User visits the homepage.	Requests car data (available cars list).	
2	Request Car Listings	User selects filters and searches for cars.	Fetches filtered list of cars (using /api/cars).	
3	Display Car Listings	Displays car list with available filters.	Returns filtered car listings.	
4	View Specific Car	User selects a specific car to view.	Fetches detailed car information (using /api/car/{carId}).	
5	Fetch Car Details	Views detailed information of the selected car.	Returns specific car details.	
6	Book Car	User selects car and proceeds to book.	Receives booking request, creates a new booking (using /api/booking).	

7	Create Booking	Submits booking details (car, date, etc.).	Processes booking details and returns booking ID.	
8	Proceed to Checkout	User proceeds to checkout.	Prepares payment processing (creates a payment intent).	
9	Create Payment Intent		Creates a payment intent (using /api/payment).	Calls payment gateway (Stripe/PayPal/JazzCash/EasyPaisa).
10	Payment Processing		Receives payment confirmation.	Processes payment and confirms transaction status.
11	Payment Confirmation	Displays payment status.	Updates booking status (confirmed).	
12	Submit Review	User submits a review for the car after rental.	Receives review details (rating, comment).	
13	Store Review		Saves the review (using /api/reviews).	
14	Return Review Status	Displays review submission status.	Confirms review has been successfully submitted.	

Clarifying the Flow:

- **Step 1-3**: The user first interacts with the homepage, where they filter and view available cars. The frontend calls the backend API to fetch the car listings.

- **Step 4-5**: The user selects a car, and the frontend fetches the detailed information for that car.
- **Step 6-7**: After selecting the car, the user proceeds to book it. The backend creates a booking and returns a booking ID.
- **Step 8-11**: The user proceeds to checkout, and payment is processed. The frontend communicates with the backend to create a payment intent and finalize the transaction through an external payment gateway.
- **Step 12-14**: After the rental, the user submits a review. The frontend submits the review to the backend, which stores it in the database.