### ****Frontend Flow (User Journey)****

1. **Home Page (Page 1)**:
   * The user arrives on the homepage where they can navigate to log in or sign up. They can also search for cars by entering a **location**, **from date**, and **to date**.
   * Users can explore different **car categories** (e.g., Economical, SUV, Luxury, Convertible).
2. **Authentication (Pages 2 and 3)**:
   * On **Page 2**, customers log in using their credentials.
   * On **Page 3**, customers sign up to create an account if they are new.
3. **Category Pages (Pages 4–7)**:
   * When a user selects a car category (like SUV), they are redirected to specific category pages where they can view available cars.
   * These pages display cars based on the selected category and their availability based on the location and dates entered.
4. **Search Function (Page 8)**:
   * From the homepage, users enter their **pickup location**, **from date**, and **to date** to search for available cars.
   * The system filters cars matching the user's search criteria (location, dates) and shows the available cars on **Page 8**.
5. **Car Booking**:
   * After viewing the available cars, users can click “Rent Now” to start the booking process.
   * The booking form includes the car details, pickup location, and booking duration.
   * Upon submission, the booking is stored in the **Bookings** table, and a **Booking ID** is generated. The initial status of the booking is set to **pending**.

### ****Admin Dashboard Flow****

1. **Admin Login**: Admins log in through a separate portal.
2. **Admin Actions**:
   * Admins can **view all cars**, **add/edit/delete cars**, and **view all booking requests**.
   * They can approve or reject bookings based on availability
   * Admins can also **view payments** and manage records related to payments.

### ****1. Database Structure (Collections/Tables)****

Here’s a breakdown of the database collections and the data they store:

#### ****Users Table****

* **Purpose**: Stores information about customers
* **Fields**:
  + Username: username can be used
  + email: Email address for login/communication.
  + password: Encrypted password for authentication.

#### ****Cars Table****

* **Purpose**: Stores details of the cars available for rental.
* **Fields**:
  + carId: Unique identifier for each car.
  + carName: The name/model of the car (e.g., "Hyundai i20").
  + category: Category of the car (e.g., **Economical**, **SUV**, **Luxury**, **Convertible**).
  + location: The **pickup location** where the car is available.
  + pricePerDay: Rental price per day for that car.
  + availability: Boolean value indicating whether the car is available for rent (true = available, false = unavailable).
  + image: Optionally, a URL or path to the car's image for display on the website.
  + description: Additional details about the car (e.g., features, specifications).

#### ****Bookings Table****

* **Purpose**: Stores booking details for each car rental.
* **Fields**:
  + bookingId: Unique identifier for each booking.
  + userId: The **userId** of the customer who made the booking.
  + carId: The **carId** of the car being rented.
  + pickupLocation: The pickup location for the booking (can be different from the car’s location).
  + fromDate: The start date of the rental period.
  + toDate: The end date of the rental period.
  + status: The status of the booking, which can be **pending**, **approved**, or **rejected**. Initially set to **pending**.
  + totalPrice: Total cost of the booking, calculated as pricePerDay \* numberOfDays.
  + paymentStatus: (Optional) Indicates whether the payment for the booking has been completed.

#### ****Payments Table****

* **Purpose**: Stores payment records for bookings.
* **Fields**:
  + paymentId: Unique identifier for each payment.
  + bookingId: Links the payment to a specific booking.
  + paymentAmount: The amount paid for the booking.
  + paymentDate: The date the payment was made.
  + paymentStatus: Whether the payment was successful or failed.

### ****2. API Interactions****

APIs are used for communication between the **frontend** (the user interface) and the **backend** (the database). Below are the main API calls and what happens behind the scenes:

#### ****Login API****

* **Endpoint**: /api/login (POST)
* **Purpose**: Authenticates the user.
* **Backend Flow**:
  1. The user provides their email and password.
  2. The backend searches the **Users** table for a matching email.
  3. If a match is found, the password is checked (via encryption or hashing).
  4. If authentication is successful, the system generates a **session token** or **JWT token** to manage the session.
  5. The response includes a success message and the user’s role (admin or customer).

#### ****Registration API****

* **Endpoint**: /api/register (POST)
* **Purpose**: Registers a new user.
* **Backend Flow**:
  1. The user provides details (name, email, password, contact).
  2. The backend creates a new record in the **Users** table.
  3. Password is hashed and stored securely.
  4. The new user is successfully created, and a success message is returned.

#### ****Search Cars API****

* **Endpoint**: /api/cars/search (POST)
* **Purpose**: Filters cars based on user search criteria (location, dates).
* **Backend Flow**:
  1. The user submits the search request with **location**, **from date**, and **to date**.
  2. The backend queries the **Cars** table, filtering by:
     + location matching the user's search.
     + Checking the **availability** field to ensure the car is available during the requested dates.
  3. The backend also checks the **Bookings** table to ensure no existing bookings overlap with the requested dates.
  4. A list of available cars is returned to the frontend.

#### ****Car Category API****

* **Endpoint**: /api/cars/category/:type (GET)
* **Purpose**: Retrieves cars from a specific category (e.g., SUV, Luxury).
* **Backend Flow**:
  1. The backend receives the category type (SUV, Luxury, etc.) as a parameter.
  2. It queries the **Cars** table and filters cars based on the selected category.
  3. The available cars of that category are returned to the frontend.

#### ****Booking API****

* **Endpoint**: /api/booking (POST)
* **Purpose**: Creates a new booking for a car.
* **Backend Flow**:
  1. The user selects a car to rent and submits the **pickup location**, **from date**, and **to date**.
  2. The backend checks the **Cars** table to verify the car's availability for the requested dates.
  3. The backend creates a new record in the **Bookings** table with:
     + userId (from logged-in user).
     + carId (from selected car).
     + pickupLocation, fromDate, toDate.
     + status set to **pending**.
  4. A **Booking ID** is generated and returned to the frontend.

#### ****Admin Approval API****

* **Endpoint**: /api/booking/approve/:id (PATCH)
* **Purpose**: Admin approves or rejects a booking.
* **Backend Flow**:
  1. The admin selects a booking from the admin dashboard and approves or rejects it.
  2. The backend updates the **Bookings** table:
     + Sets status to **approved** or **rejected**.
  3. If the booking is approved, the car's **availability** is updated to false in the **Cars** table, marking it as unavailable for that period.

### ****3. Example Workflow (Booking a Car)****

Let’s walk through a specific scenario:

1. **User Visits the Home Page**:
   * User searches for a car in **Udupi** from **April 20–22**.
2. **Search Cars API Call**:
   * The system queries the **Cars** table for cars with location = "Udupi" and **availability = true**.
   * It cross-references the **Bookings** table to ensure no bookings overlap with **April 20–22**.
   * The filtered cars are displayed on the frontend.
3. **User Selects a Car**:
   * User clicks “Rent Now” on a **Hyundai i20** and fills out the booking form.
4. **Booking API Call**:
   * The booking details (car ID, user ID, dates, pickup location) are submitted to the backend.
   * A new record is created in the **Bookings** table with status = pending.
5. **Admin Approval**:
   * The **Admin** logs in and sees the new booking with status **pending**.
   * The admin approves the booking, and the status in the **Bookings** table is updated to **approved**.
   * The **Cars** table's **availability** for the Hyundai i20 is updated to **false** for **April 20–22**.
6. **User Notification**:
   * Once approved, the system sends a notification to the user confirming the booking.

### ****Conclusion****

This backend workflow is structured to manage users, cars, and bookings efficiently. By using a relational database (or appropriate storage) to handle entities like users, cars, bookings, and payments, the system ensures smooth car rental operations, from searching for available cars to booking and approval. The API endpoints allow for seamless communication between the **frontend** and **backend**, ensuring a responsive user experience.