1. Project Overview

A **Car Rental Website** where users can browse cars, view details, make payments, and manage rentals. Admins can manage listings, bookings, and users via a dashboard.

---

2. Tech Stack

Frontend

- **Framework**: Next.js (React-based for SSR & SSG)
- **Styling**: Tailwind CSS (for a utility-first design)
- **Icons**: React Icons (for UI enhancements)
- **Forms & Validation**: React Hook Form
- **Image Handling**: Next.js Image Optimization
- **Maps & Location**: Google Maps API or OpenStreetMap
- **Payments**: Stripe API for secure transactions

---

3. Pages & Functional Requirements

1. Home Page

- Hero Section with a search bar (filter cars by location, date, and type)
- Featured Cars Section
- FAQs Section
- CTA buttons for Sign-Up/Login

2. Car Listing Page

- Grid/List view of available cars
- Filters (Price, Car Type, Fuel Type, Transmission)
- Sorting (Price, Popularity, Latest)
- Pagination

3. Car Description Page

- High-quality car images
- Car details (Model, Year, Features, Fuel, Transmission, Price per day)
- Availability calendar
- Booking Form
- Reviews & Ratings

4. Payment & Rent Confirmation Page

- User details form
- Car rental duration selection
- Price breakdown
- Payment gateway integration (Stripe)
- Rental confirmation page with invoice download

5. Admin Dashboard (Protected Route)

- **Car Management**: Add, Edit, Delete Cars
- **Booking Management**: View, Approve, Cancel Rentals
- **User Management**: View, Ban Users
- **Analytics**: Revenue & Booking Statistics

---

4. UI/UX Design Considerations

- **Responsive**: Mobile-first approach
- **Dark Mode Support**
- **Accessibility**: WCAG-compliant design
- **Fast Loading**: Optimized images & lazy loading

---

5. API & Backend Communication

- **Data Fetching**: Next.js API Routes or external backend
- **Authentication**: NextAuth.js (OAuth, JWT)
- **Database**: Sanity or MongoDB (for car listings & bookings)

---

6. Performance Optimization

- **Static Site Generation (SSG)** for fast page loads
- **Server-Side Rendering (SSR)** for dynamic car data
- **Incremental Static Regeneration (ISR)** for up-to-date listings
- **Code Splitting & Lazy Loading**

---

7. Security Measures

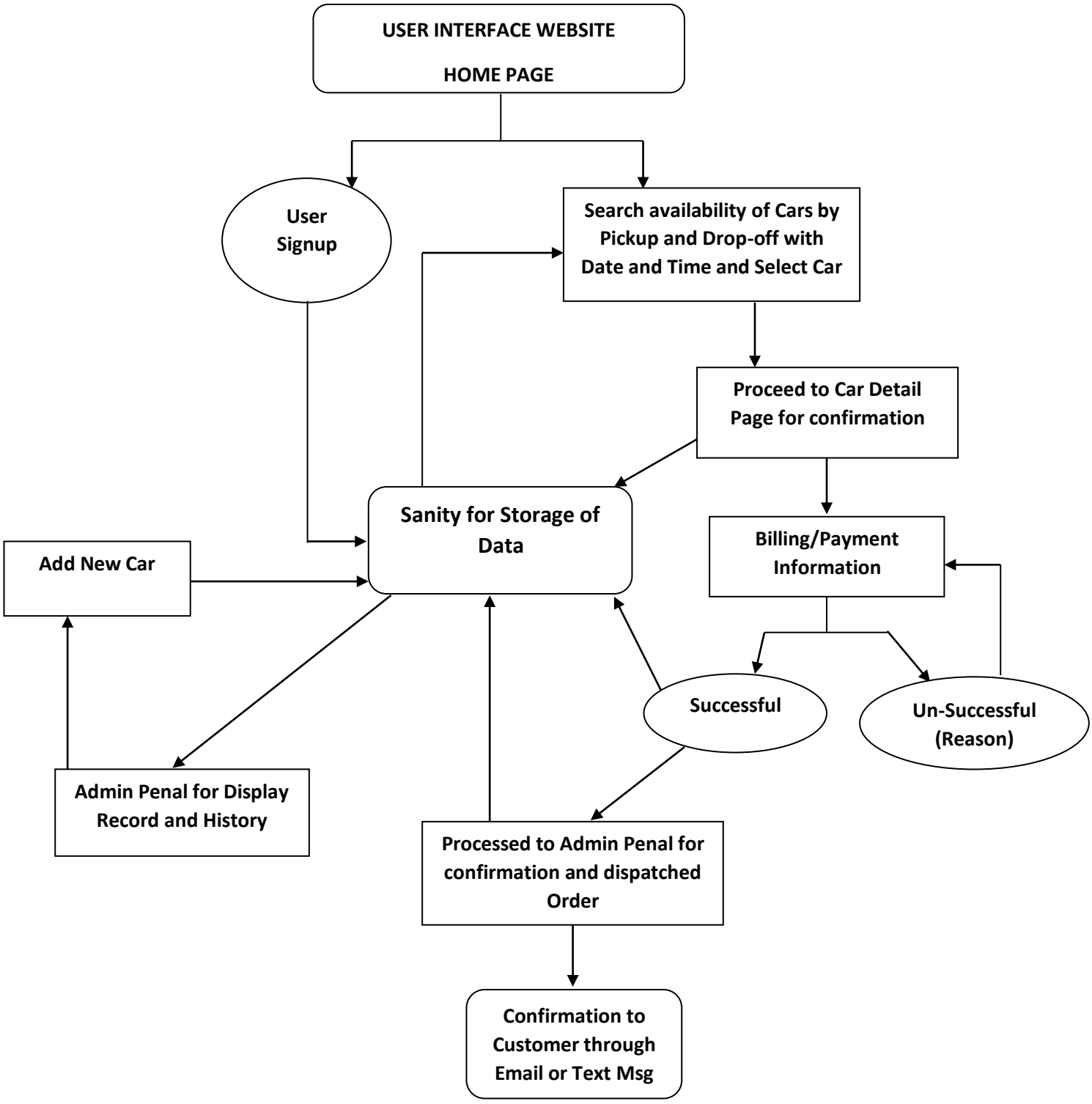
- User authentication & authorization
- Secure payments with Stripe
- Rate limiting for API protection
- Data validation & sanitization

Here's a high-level **System Architecture Diagram** for your **Car Rental Website** using **Next.js, Tailwind CSS, and APIs** for shipment tracking and payment gateway integration.

System Components:

1. **Frontend (Next.js + Tailwind CSS)**
    - Home Page
    - Car Listing
    - Car Description
    - Rent a Car (Payment Page)
    - Admin Dashboard
2. **Backend (Next.js API Routes + Node.js)**
    - User Authentication (NextAuth.js or Firebase)
    - Car Rental Management (CRUD operations for cars, rentals)
    - Payment Processing (Stripe, PayPal)
    - Admin Panel (Manage bookings, users, payments)
3. **Database (Sanity CMS / MongoDB / PostgreSQL)**
    - User Information (Profile, Payment history)
    - Car Inventory (Car details, availability, pricing)
    - Rental Transactions (Booking history, status)
4. **Third-Party Integrations**
    - **Payment Gateway (Stripe, PayPal, Razorpay)**
    - **Authentication (Google OAuth, Firebase Auth, NextAuth.js)**

WEBSITE DESIGN ARCHITECTURE / WORK FLOW



Key Interactions

1. **User Browsing** → Next.js renders **car listings** from Sanity/MongoDB.
2. **User Authentication** → Handled via **NextAuth.js** or **Firebase**.
3. **Car Rental & Payment** → User selects a car and pays via **Stripe/PayPal**.
4. **Booking Confirmation** → Database updates **rental status**.
5. **Admin Panel** → Admin manages **bookings, users, payments**.

Tech Stack

- **Frontend**: Next.js, Tailwind CSS, React Icons
- **Backend**: Next.js API Routes, Node.js
- **Database**: Sanity CMS / MongoDB
- **Auth**: NextAuth.js / Firebase
- **Payments**: Stripe, PayPal

Here is a **Sanity schema** for the **Car Rental Website** using Next.js and Tailwind CSS.

(Main Schema File) schema.js

This file imports all the schemas and exports them as an array.

```
import car from './car';
import booking from './booking';
import user from './user';
import review from './review';
import admin from './admin';

export default [car, booking, user, review, admin];
```

2. car.js (Car Schema)

Defines all car details.

```
export default {
  name: 'car',
  title: 'Cars',
  type: 'document',
  fields: [
    {
      name: 'name',
      title: 'Car Name',
      type: 'string',
      },
    {
      name: 'brand',
      title: 'Brand',
      type: 'string',
    },
    {
      name: 'image',
      title: 'Car Image',
      type: 'image',
      options: { hotspot: true },
    },
    {
      name: 'pricePerDay',
      title: 'Price Per Day',
      type: 'number',
    },
    {
      name: 'type',
      title: 'Car Type',
      type: 'string',
      options: {
        list: ['SUV', 'Sedan', 'Hatchback', 'Truck', 'Convertible', 'Electric'],
      },
```

```
    },
    {
      name: 'fuelType',
      title: 'Fuel Type',
      type: 'string',
      options: {
        list: ['Petrol', 'Diesel', 'Electric', 'Hybrid'],
      },
    },
    {
      name: 'transmission',
      title: 'Transmission',
      type: 'string',
      options: {
        list: ['Automatic', 'Manual'],
      },
    },
    {
      name: 'seats',
      title: 'Seats',
      type: 'number',
    },
    {
      name: 'availability',
      title: 'Availability',
      type: 'boolean',
    },
    {
      name: 'description',
      title: 'Description',
      type: 'text',
    },
  ],
};
```

3. booking.js (Booking Schema)

Stores booking details.

```
export default {
  name: 'booking',
  title: 'Bookings',
  type: 'document',
  fields: [
    {
      name: 'car',
      title: 'Car',
      type: 'reference',
      to: [{ type: 'car' }],
    },
    {
      name: 'user',
      title: 'User',
      type: 'reference',
      to: [{ type: 'user' }],
    },
    {
```

```
    name: 'startDate',
    title: 'Start Date',
    type: 'datetime',
   },
   {
    name: 'endDate',
    title: 'End Date',
    type: 'datetime',
   },
   {
    name: 'totalAmount',
    title: 'Total Amount',
    type: 'number',
   },
   {
    name: 'paymentStatus',
    title: 'Payment Status',
    type: 'string',
    options: {
    list: ['Pending', 'Completed', 'Failed'],
    },
   },
  ],
};
```

4. user.js (User Schema)

Manages user data.

```
export default {
  name: 'user',
  title: 'Users',
  type: 'document',
  fields: [
   {
    name: 'name',
    title: 'Full Name',
    type: 'string',
   },
   {
    name: 'email',
    title: 'Email',
    type: 'string',
   },
   {
    name: 'phone',
    title: 'Phone Number',
    type: 'string',
   },
   {
    name: 'profileImage',
    title: 'Profile Image',
    type: 'image',
   },
   {
    name: 'role',
    title: 'User Role',
```

```
      type: 'string',
      options: {
        list: ['Customer', 'Admin'],
      },
    },
  ],
};
```

5. review.js (Car Review Schema)

Stores customer reviews for cars.

```
export default {
  name: 'review',
  title: 'Reviews',
  type: 'document',
  fields: [
    {
      name: 'car',
      title: 'Car',
      type: 'reference',
      to: [{ type: 'car' }],
    },
    {
      name: 'user',
      title: 'User',
      type: 'reference',
      to: [{ type: 'user' }],
    },
    {
      name: 'rating',
      title: 'Rating',
      type: 'number',
    },
    {
      name: 'comment',
      title: 'Comment',
      type: 'text',
    },
    {
      name: 'createdAt',
      title: 'Created At',
      type: 'datetime',
      options: {
        dateFormat: 'YYYY-MM-DD',
        timeFormat: 'HH:mm',
      },
    },
  ],
};
```

6. admin.js (Admin Schema)

For managing admins.

```
export default {
  name: 'admin',
  title: 'Admins',
  type: 'document',
  fields: [
    {
      name: 'name',
      title: 'Admin Name',
      type: 'string',
    },
    {
      name: 'email',
      title: 'Email',
      type: 'string',
    },
    {
      name: 'role',
      title: 'Role',
      type: 'string',
      options: {
        list: ['Super Admin', 'Manager'],
      },
    },
  ],
};
```

Conclusion

This schema provides a **structured way** to manage **cars, bookings, users, and reviews**. It integrates smoothly with **Next.js** for a dynamic **Car Rental Website**.

To integrate APIs for Payment gateways in your Next.js car rental website, follow these steps:

Payment Gateway Integration

For payments, you can use:

- **Stripe** (Stripe API)
- **PayPal** (PayPal API)

Final Steps

- **Secure API keys** using .env.local
- NEXT_PUBLIC_STRIPE_PUBLIC_KEY=your_stripe_public_key
- STRIPE_SECRET_KEY=your_stripe_secret_key
- **Test APIs using Postman** or API testing tools.
- **Deploy with environment variables** for production security.

API Endpoints for Car Rental Website (Based on Sanity Schema)

Below is a structured API design for your **Car Rental Website**, defining endpoints for **Users, Cars, Rentals, Payments, and Shipment Tracking**.

1. Authentication & User Management

Endpoint Name	Method	Description
/api/auth/signup	POST	Register a new user
/api/auth/login	POST	Authenticate user and return JWT token
/api/auth/logout	POST	Logout user and destroy session
/api/auth/me	GET	Get logged-in user details
/api/users/:id	GET	Get user profile by ID
/api/users/:id	PUT	Update user profile
/api/users/:id	DELETE	Delete user account

2. Car Management

Endpoint Name	Method	Description
/api/cars	GET	Get all available cars
/api/cars/:id	GET	Get car details by ID
/api/cars	POST	Add a new car (Admin only)
/api/cars/:id	PUT	Update car details (Admin only)
/api/cars/:id	DELETE	Remove a car listing (Admin only)

3. Rental Management

Endpoint Name	Method	Description
/api/rentals	GET	Get all rental bookings
/api/rentals/:id	GET	Get rental details by ID
/api/rentals	POST	Create a new car rental
/api/rentals/:id	PUT	Update rental details (Admin only)
/api/rentals/:id	DELETE	Cancel rental booking

4. Payment Processing

Endpoint Name	Method	Description
/api/payments/checkout	POST	Initiate a car rental payment
/api/payments/status/:id	GET	Get payment status by rental ID
/api/payments/refund/:id	POST	Process a refund (Admin only)

Payment Providers: Stripe, PayPal

5. Admin Dashboard (Restricted)

Endpoint Name	Method	Description
/api/admin/overview	GET	Get admin dashboard stats
/api/admin/users	GET	Get list of all users
/api/admin/rentals	GET	Get all rental records
/api/admin/payments	GET	Get all payment transactions
/api/admin/shipments	GET	Get all shipment details

Access Control: Only authenticated admins can access these endpoints.

Next Steps

1. Implement these API endpoints using **Next.js API routes**.
2. Use **JWT Authentication** for protected routes.
3. Integrate **Stripe API** for payments.
4. Connect to **Sanity CMS** for dynamic content management.