1. **Frontend Workflow Diagram & Plan**
2. **Backend Workflow Diagram & Plan**
3. **Database Design (ER Diagram & Plan)**

**🎨 1. Frontend Workflow Diagram & Plan**

**✅ Frontend Technologies:**

* **Web**: React.js (SPA)
* **Mobile (optional)**: Flutter (cross-platform)
* **State Management**: Redux / Context API
* **API Integration**: Axios or Fetch
* **Routing**: React Router
* **Form Handling**: React Hook Form + Yup (for validation)

**🌐 Frontend Workflow Diagram**

[User Login/Signup Page]

|

v

[Home Dashboard]

| | |

| | |

[Buy] [Sell] [Services]

| | |

v v v

[Car [Add Car [Search Service Centers]

Listings] Details] | |

| | | |

[View Car] [Upload] [Compare Prices]

Details Images | |

| [Book Slot]

[Submit] |

[Track Booking Status]

**🎯 Frontend Pages / Components Plan**

| **Page / Component** | **Features** |
| --- | --- |
| LoginPage | User login, JWT token storage |
| SignupPage | Buyer/Seller/Service Provider registration with KYC option |
| DashboardPage | Overview of listings, bookings, alerts |
| CarListingsPage | Browse, filter, search cars |
| CarDetailPage | Price, specs, contact seller, view history |
| AddCarPage | Car info form, image upload, ML price suggestion |
| ServiceSearchPage | Filter service centers by type, location |
| BookingPage | Book appointments, choose date/time, pick/drop options |
| ProfilePage | View/edit user details, saved listings, service history |
| Notifications | Push notifications for updates |

**🛠️ 2. Backend Workflow Diagram & Plan**

**🔧 Backend Technologies:**

* **Language**: Java 17+
* **Framework**: Spring Boot 3+
* **Auth**: Spring Security + JWT
* **Communication**: REST APIs + OpenFeign (for microservices)
* **Database Access**: Spring Data JPA
* **Validation**: Hibernate Validator
* **API Docs**: Swagger / Springdoc OpenAPI

**⚙️ Backend Workflow Diagram**

[Frontend Request]

|

v

[API Gateway (Spring Cloud Gateway)]

|

+---------------------+

| Authentication MS | <---> [User DB]

+---------------------+

|

+---------------------+

| Car Service MS | <---> [Car DB]

+---------------------+

|

+---------------------+

| Service Center MS | <---> [Service DB]

+---------------------+

|

+---------------------+

| Pricing Engine MS | <---> [ML Model / API]

+---------------------+

|

+---------------------+

| Booking & Payment | <---> [Booking DB, Payment API]

+---------------------+

**🧩 Backend Modules Plan**

| **Module** | **Endpoints (Sample)** |
| --- | --- |
| **Auth Service** | POST /auth/signup, POST /auth/login, GET /user/me |
| **User Service** | GET /users/{id}, PUT /users/{id} |
| **Car Listing Service** | GET /cars, POST /cars, GET /cars/{id}, DELETE /cars/{id} |
| **Service Mgmt Service** | GET /services, POST /services, GET /services/{id} |
| **Booking Service** | POST /book, GET /bookings, PUT /bookings/{id}/status |
| **Pricing Engine** | POST /price/predict (inputs: make, model, mileage, year → output: price) |
| **Notification Service** | POST /notify/email, POST /notify/sms |

**🗃️ 3. Database Design (ER Diagram & Plan)**

**📌 Core Entities:**

1. **User**
2. **Car**
3. **ServiceCenter**
4. **Booking**
5. **Review**
6. **VehicleInspection**
7. **Notification**

**🔄 ER Diagram (Simplified Text Format)**

[User]

- id (PK)

- name

- email

- password

- role (BUYER | SELLER | SERVICE\_PROVIDER)

- phone

- verified (bool)

[Car]

- id (PK)

- owner\_id (FK to User)

- make

- model

- year

- mileage

- condition

- price

- images (JSON or S3 link)

[ServiceCenter]

- id (PK)

- user\_id (FK to User)

- name

- location

- services\_offered (JSON)

[Booking]

- id (PK)

- user\_id (FK to User)

- service\_center\_id (FK)

- service\_type

- date\_time

- status (BOOKED | COMPLETED | CANCELLED)

- price

[Review]

- id (PK)

- user\_id (FK)

- service\_center\_id (FK)

- rating (1-5)

- comments

[VehicleInspection]

- id (PK)

- car\_id (FK)

- inspection\_date

- report (text/file link)

[Notification]

- id (PK)

- user\_id (FK)

- message

- type (EMAIL/SMS/PUSH)

- status (SENT/FAILED)

**✅ Step 3: Database Design – ER Diagram & Detailed Plan**

**🧩 Core Entities (Tables)**

We’ll divide the design based on services. Here's the **Entity-Relationship Diagram (ERD)** structure grouped logically:

**🗃️ [1] User Service – users, roles**

┌────────────┐ ┌─────────────┐

| users | | roles |

└────────────┘ └─────────────┘

| id (PK) |◄────┐ | id (PK) |

| name | └──| name |

| email | └─────────────┘

| password |

| role\_id (FK) |

└────────────┘

* A user has one role: BUYER / SELLER / SERVICE\_PROVIDER
* Passwords are stored hashed.

**🚗 [2] Car Service – cars, car\_images, car\_reviews**

┌────────────┐ ┌──────────────┐

| cars |◄──────| car\_images |

└────────────┘ └──────────────┘

| id (PK) | | id (PK) |

| seller\_id | | car\_id (FK) |

| make | | image\_url |

| model | └──────────────┘

| year |

| mileage | ┌────────────────┐

| price |◄──────| car\_reviews |

| status | └────────────────┘

└────────────┘ | id (PK) |

| car\_id (FK) |

| reviewer\_id |

| rating (1-5) |

| comment |

└────────────────┘

**🛠️ [3] Service Center – service\_centers, services\_offered**

┌───────────────────┐ ┌────────────────────┐

| service\_centers |◄──────| services\_offered |

└───────────────────┘ └────────────────────┘

| id (PK) | | id (PK) |

| name | | center\_id (FK) |

| provider\_id (FK) | | service\_name |

| location | | base\_price |

| contact\_number | └────────────────────┘

└───────────────────┘

**📅 [4] Booking Service – bookings**

┌───────────────┐

| bookings |

└───────────────┘

| id (PK) |

| user\_id (FK) |

| service\_id (FK) |

| car\_id (FK) |

| date |

| time\_slot |

| status | --> BOOKED / CANCELLED / DONE

└───────────────┘

**🧠 [5] Pricing Engine – No RDBMS**

Handled by a separate ML model via REST. Stores predictions optionally for audit.

┌────────────────┐

| price\_history |

└────────────────┘

| id |

| car\_id (FK) |

| predicted\_price|

| timestamp |

└────────────────┘

**🔔 [6] Notification System – notifications**

┌────────────────────┐

| notifications |

└────────────────────┘

| id (PK) |

| user\_id (FK) |

| message |

| is\_read (boolean) |

| created\_at |

└────────────────────┘

**🧾 [7] Logs/Audit (Optional)**

┌──────────────┐

| audit\_logs |

└──────────────┘

| id |

| user\_id |

| action |

| timestamp |

| metadata |

└──────────────┘

**🧱 Database Type**

* **PostgreSQL** or **MySQL** for production
* **H2** for local development/testing

**🔐 Relationships Summary**

* **users (1) ↔ (M) cars**
* **users (1) ↔ (M) bookings**
* **users (1) ↔ (M) service\_centers**
* **cars (1) ↔ (M) car\_images**
* **cars (1) ↔ (M) car\_reviews**
* **service\_centers (1) ↔ (M) services\_offered**
* **services\_offered (1) ↔ (M) bookings**

**Front-End Overview**

**🔷 1. Tech Stack Overview**

| **Component** | **Technology Used** | **Purpose** |
| --- | --- | --- |
| UI Framework | React.js | Component-based frontend |
| Styling Framework | Tailwind CSS / Bootstrap | Modern UI design |
| HTTP Client | Axios | API communication |
| Routing | React Router DOM | Navigation between pages |
| Form Handling | React Hook Form / Formik | User input forms |
| State Management | Redux Toolkit / Context API | Global state (optional) |
| UI Library (optional) | Material-UI / Ant Design | Pre-built components |
| Icons | Heroicons / FontAwesome | Visual icons |

**🔷 2. Folder Structure**

src/

├── components/ # Reusable UI components

│ └── Navbar.jsx

│ └── CarCard.jsx

├── pages/ # Main page components

│ └── Home.jsx

│ └── CarList.jsx

│ └── ServiceBooking.jsx

│ └── Dashboard.jsx

├── services/ # API calls (Axios)

│ └── carService.js

│ └── userService.js

├── assets/ # Images, logos, styles

├── App.jsx # Root component with routes

├── main.jsx # React DOM root entry

└── routes/ # Route guards and route configs

**🔷 3. Core Pages and Functionalities**

**✅ 1. Home Page**

* **Components**: Search bar, featured listings, call-to-action.
* **Features**:
  + Browse top car deals
  + Navigation links: Buy | Sell | Service
  + Responsive design

**✅ 2. Login / Register**

* **Forms**: Built with React Hook Form + Validation
* **Features**:
  + Role-based login (buyer/seller/service provider/admin)
  + Token storage in localStorage
  + Error handling, form validation
* **Security**: Send JWT to backend, protect routes using React Router

**✅ 3. Car Listing Page**

* **API**: GET /cars
* **Features**:
  + Filter by brand, price, year, etc.
  + Sort by price/date
  + View car details
  + Add to wishlist (optional)

**✅ 4. Car Details Page**

* **API**: GET /cars/{id}
* **Features**:
  + Full specs, images, owner details
  + Contact seller (if buyer logged in)
  + Book a test drive (optional feature)

**✅ 5. Sell Car Page**

* **Form**: Upload details + images
* **API**: POST /cars
* **Features**:
  + Form validation
  + Upload images (multi-file input)
  + Show price recommendation via Pricing API

**✅ 6. Service Booking Page**

* **API**: GET /service-centers
* **Features**:
  + Browse service centers
  + Book maintenance slot (calendar + time)
  + Payment UI (optional)

**✅ 7. Dashboard (Role-Based)**

| **Role** | **Dashboard Features** |
| --- | --- |
| Buyer | My Bookings, Wishlist, History |
| Seller | My Cars, Add Car, Edit/Delete Listings |
| Service Provider | Manage Bookings, Add Services |
| Admin | Manage Users, Reports, View Logs |

**✅ 8. Admin Panel**

* User role management
* Block/unblock accounts
* View analytics (optional)

**🔷 4. Authentication Frontend Flow**

* On login → get JWT → store in localStorage
* On every API call → send JWT in Authorization header
* Use React Router Protected Routes for:
  + /dashboard
  + /sell
  + /service-booking

**🔷 5. Axios Setup (Example)**

// services/axiosInstance.js

import axios from 'axios';

const axiosInstance = axios.create({

baseURL: 'http://localhost:8080/api',

});

axiosInstance.interceptors.request.use(config => {

const token = localStorage.getItem("token");

if (token) config.headers.Authorization = `Bearer ${token}`;

return config;

});

export default axiosInstance;

**🔷 6. API Integration Plan**

| **Feature** | **Method** | **API Endpoint** |
| --- | --- | --- |
| Register | POST | /auth/register |
| Login | POST | /auth/login |
| Get Cars | GET | /cars |
| Add Car | POST | /cars |
| Get Services | GET | /service-centers |
| Book Service | POST | /bookings |
| Get Bookings | GET | /bookings/user/{id} |

**🔷 7. UI Libraries to Use**

* **React Hook Form**: Forms
* **SweetAlert2 / Toastify**: Notifications
* **React Datepicker**: Booking dates
* **React Icons**: Icons
* **React Image Gallery**: Car photo gallery

**🔷 8. Future Enhancements**

* PWA (Progressive Web App)
* Add Chat system (buyer ↔ seller)
* Payment gateway integration (Razorpay/Stripe)
* Mobile responsive layouts
* Admin analytics dashboard with charts

**🔷 Database Type: Relational Database (MySQL/PostgreSQL)**

**🔷 1. Main Entities**

Here's the core set of tables we'll design:

1. **Users**
2. **Cars**
3. **CarImages**
4. **Bookings**
5. **ServiceCenters**
6. **Services**
7. **Reviews**
8. **Roles**
9. **Payments (optional)**

**🔷 2. Entity Relationship Diagram (ERD)**

**[ERD Diagram Summary]**

Users (1)───< (M) Cars

Users (1)───< (M) Bookings

Cars (1) ───< (M) CarImages

ServiceCenters (1) ───< (M) Services

Users (1) ───< (M) Reviews

Roles (1) ───< (M) Users

**🔷 3. Detailed Table Structures**

**✅ users**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| name | VARCHAR | Full name |
| email | VARCHAR | Unique, indexed |
| password | VARCHAR | Hashed |
| phone | VARCHAR | Optional |
| role\_id | FK → roles | Role (buyer, seller, etc.) |
| created\_at | TIMESTAMP |  |

**✅ roles**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | INT (PK) | Role ID |
| name | VARCHAR | "Buyer", "Seller", etc. |

**✅ cars**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| owner\_id | FK → users | Seller who listed the car |
| brand | VARCHAR | Car brand (Honda, etc.) |
| model | VARCHAR | Car model |
| year | INT | Manufacturing year |
| price | DECIMAL | Selling price |
| mileage | VARCHAR | e.g., 50,000 km |
| condition | VARCHAR | New / Used |
| location | VARCHAR | Seller’s location |
| description | TEXT | Details about the car |
| created\_at | TIMESTAMP |  |

**✅ car\_images**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| car\_id | FK → cars | Reference to the car |
| image\_url | TEXT | Link to image |

**✅ service\_centers**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| name | VARCHAR | Center name |
| location | VARCHAR | City or address |
| contact\_info | VARCHAR | Email/Phone |

**✅ services**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| center\_id | FK → service\_centers | Service provider |
| name | VARCHAR | e.g., Oil Change |
| price | DECIMAL | Cost of service |
| duration | INT (mins) | Duration in minutes |

**✅ bookings**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| user\_id | FK → users | Who booked the service |
| service\_id | FK → services | What service was booked |
| date | DATE | Selected date |
| time\_slot | VARCHAR | e.g., 10:00 - 11:00 |
| status | VARCHAR | Pending / Completed / Canceled |

**✅ reviews**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| user\_id | FK → users | Who wrote the review |
| service\_id | FK → services | What service is being reviewed |
| rating | INT | 1 to 5 |
| comment | TEXT | Optional feedback |
| created\_at | TIMESTAMP |  |

**✅ payments (Optional)**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID / INT | Primary Key |
| booking\_id | FK → bookings | Paid booking |
| amount | DECIMAL | Payment amount |
| method | VARCHAR | UPI / Card / Netbanking |
| status | VARCHAR | Paid / Failed / Pending |

**🔷 4. Constraints & Best Practices**

* Use **UUIDs** for scalability and distributed systems.
* **Email** and **phone** should be **unique** in the users table.
* Use **foreign keys** for strong relational mapping.
* Implement **soft deletes** with a deleted\_at column if needed.

**🔷 5. Indexes to Add**

* users.email – for login
* cars.brand, cars.model, cars.location – for faster filters
* bookings.user\_id – for dashboard history

**🔷 6. Workflows using Database**

**🔁 Buying a Car Flow:**

1. Buyer filters cars → GET /cars?brand=Honda
2. Clicks on a car → GET /cars/{id}
3. Contacts Seller → POST /contact

**🔁 Booking Service Flow:**

1. User picks a center → GET /service-centers
2. Chooses service → GET /services?center\_id=
3. Books → POST /bookings
4. Optional payment → POST /payments

**Backend**

1. ✅ Backend Tech Stack
2. ✅ Layered Architecture
3. ✅ Core Modules
4. ✅ RESTful API Endpoints (CRUD)
5. ✅ Security and Validation
6. ✅ Backend Workflow Diagrams
7. ✅ Tools/Best Practices

**🔧 1. Tech Stack**

| **Layer** | **Technology** |
| --- | --- |
| Language | Java (with Spring Boot 3+) |
| Framework | Spring MVC, Spring Data JPA, Spring Security |
| DB Connection | Hibernate ORM |
| Database | MySQL / PostgreSQL |
| API Docs | Swagger / Springdoc OpenAPI |
| Build Tool | Maven / Gradle |
| Testing | JUnit + Mockito |

**🏗️ 2. Layered Architecture**

Client → Controller Layer → Service Layer → Repository Layer → Database

↑ ↓

DTO Mapper Validations

* **Controller**: Handles HTTP requests.
* **Service**: Core business logic.
* **Repository (DAO)**: Interacts with the DB using JPA.
* **DTOs**: Used to transfer safe/clean data between client and server.
* **Validation**: Hibernate Validator + Custom validation rules.

**📦 3. Core Modules**

**✅ User Module**

* Register, Login, View Profile
* Role-based access (buyer, seller, service provider)

**✅ Car Module**

* Add, Update, Delete Car Listings
* View Car Listings with Filters

**✅ Service Booking Module**

* View service centers & services
* Book appointments, cancel, update

**✅ Review Module**

* Add/View ratings & reviews for service centers

**✅ Admin Module**

* Approve/Delete Listings
* View all users, bookings

**🔗 4. REST API Endpoints (Sample)**

**🔹 AuthController**

POST /api/auth/register // Register user

POST /api/auth/login // Login, returns JWT

**🔹 UserController**

GET /api/users/profile // Get logged-in user details

PUT /api/users/update // Update profile

**🔹 CarController**

GET /api/cars // Get all car listings

GET /api/cars/{id} // Get single car

POST /api/cars // Add new car

PUT /api/cars/{id} // Update car details

DELETE /api/cars/{id} // Delete car

**🔹 ServiceCenterController**

GET /api/service-centers // List all centers

GET /api/service-centers/{id}/services // Get services

**🔹 BookingController**

POST /api/bookings // Book a service

GET /api/bookings/my // Get user’s bookings

PUT /api/bookings/{id} // Update booking

DELETE /api/bookings/{id} // Cancel

**🔹 ReviewController**

POST /api/reviews // Post a review

GET /api/reviews/service/{id} // Get reviews for service

**🛡️ 5. Security and Validation**

* **Spring Security + JWT** for Auth
* Role-based access:
  + /api/admin/\*\* → ADMIN only
  + /api/cars → SELLER & ADMIN
  + /api/bookings → BUYER only
* **Validation Annotations**:
  + @NotBlank, @Email, @Min, etc.
* **Exception Handling**:
  + @ControllerAdvice with custom exceptions

**📊 6. Backend Workflow Diagram**

[Client Request]

↓

[REST Controller]

↓

[Service Layer]

- Validation

- Business Logic

↓

[Repository Layer]

- CRUD using JPA

↓

[Database]

← Response: DTO via Controller

Example:

* User Books a Service:  
  → Controller validates JWT  
  → Service checks available slots  
  → Booking is saved in DB  
  → Confirmation response returned

**🛠️ 7. Tools & Best Practices**

* DTO pattern with **ModelMapper**
* Global error handling via @ControllerAdvice
* CORS configured for frontend
* Swagger UI for all APIs
* Logs via SLF4J/Logback
* Use of Profiles (dev, prod) for configs

**✅ Backend Plan Summary:**

| **Layer** | **Tool/Lib Used** | **Notes** |
| --- | --- | --- |
| Auth | Spring Security + JWT | Role-based, secure |
| Data Layer | Spring Data JPA | CRUD with repositories |
| Business | Spring Service Layer | Contains validations & logic |
| API Layer | REST + Spring MVC | Exposed to frontend |
| Validation | Hibernate Validator | Bean-level constraints |
| Docs | Swagger / Springdoc | API testing & docs |