**Project Plan**

**1. Define the Scope (Day 1)**

* **Backend (Spring Boot Microservices):**
  + Admin service: CRUD operations for vehicles (bikes and cars).
  + User service: User registration, login, and authentication (JWT).
  + Vehicle service: Listing vehicles, filtering by price, manufacturing year, etc.
  + Booking service: Book a test drive or buy a vehicle.
* **Frontend (React):**
  + Admin panel: Add, update, delete, and list vehicles.
  + User panel: Browse vehicles, filter, book test drives, and buy.
  + Login/register pages.
* **Database:**
  + Use MySQL or PostgreSQL for relational data (users, vehicles, bookings).
  + Use MongoDB (optional) for unstructured data like logs or analytics.
* **API Documentation:**
  + Use Swagger for API documentation.

**2. Set Up the Project (Day 1)**

* **Backend:**
  + Create a multi-module Maven project for microservices.
  + Set up Spring Boot for each microservice (Admin, User, Vehicle, Booking).
  + Configure Eureka Server for service discovery.
  + Use Spring Cloud Gateway for API gateway.
  + Add Spring Security for authentication and authorization.
* **Frontend:**
  + Set up a React project using create-react-app.
  + Install necessary libraries: react-router-dom, axios, redux (optional), and material-ui or bootstrap for styling.
* **Database:**
  + Set up MySQL/PostgreSQL and MongoDB (if needed).
  + Use Flyway or Liquibase for database migrations.

**3. Develop Backend Microservices (Day 2–4)**

* **Admin Service:**
  + CRUD APIs for vehicles (bikes and cars).
  + Use Spring Data JPA for database operations.
* **User Service:**
  + User registration and login APIs.
  + Use JWT for authentication.
* **Vehicle Service:**
  + APIs to list vehicles with filters (price, year, etc.).
  + Use pagination for better performance.
* **Booking Service:**
  + APIs to book a test drive or buy a vehicle.
  + Use transactional management for bookings.
* **Common:**
  + Add exception handling and logging.
  + Use DTOs for data transfer between layers.

**4. Develop Frontend (Day 5–6)**

* **Admin Panel:**
  + Create forms to add, update, and delete vehicles.
  + Display a list of vehicles in a table.
* **User Panel:**
  + Create a homepage to list vehicles with filters.
  + Add a vehicle details page with options to book a test drive or buy.
  + Implement login and registration forms.
* **Integration:**
  + Use axios to call backend APIs.
  + Handle authentication using JWT tokens.

**5. Testing and Bug Fixing (Day 7)**

* **Backend:**
  + Write unit tests using JUnit and Mockito.
  + Test APIs using Postman.
* **Frontend:**
  + Test all features manually.
  + Fix any UI/UX issues.
* **Integration Testing:**
  + Ensure all microservices work together seamlessly.

**6. Hosting (Day 7)**

* **Backend:**
  + Use Docker to containerize each microservice.
  + Deploy on AWS, Heroku, or any cloud platform.
* **Frontend:**
  + Build the React app and deploy it on Netlify, Vercel, or S3.
* **Database:**
  + Use a managed database service like AWS RDS or MongoDB Atlas.

**Tech Stack**

* **Backend:**
  + Java 17 (latest LTS version).
  + Spring Boot 3.x.
  + Spring Cloud (Eureka, Gateway).
  + Spring Security with JWT.
  + Spring Data JPA.
  + MySQL/PostgreSQL.
  + Maven (latest version).
* **Frontend:**
  + React 18.
  + Axios for API calls.
  + React Router for navigation.
  + Material-UI or Bootstrap for styling.
* **Other Tools:**
  + Docker for containerization.
  + Postman for API testing.
  + Git for version control.

**Basic Features to Implement**

1. **Admin Features:**
   * Add, update, delete, and list vehicles.
2. **User Features:**
   * Register, login, and view profile.
   * Browse vehicles with filters (price, year, etc.).
   * Book a test drive or buy a vehicle.
3. **General Features:**
   * Responsive UI.
   * Authentication and authorization.

**Future Features**

1. Integrate payment gateway (Razorpay, Stripe).
2. Book a service at the nearest garage.
3. Add reviews and ratings for vehicles.
4. Implement advanced search and filtering.
5. Add notifications (email/SMS) for bookings.

**Tips for Success**

1. **Prioritize Features:**
   * Focus on the MVP first. Add advanced features later.
2. **Use Ready-Made Libraries:**
   * Use libraries like Material-UI or Bootstrap to speed up frontend development.
3. **Automate Repetitive Tasks:**
   * Use scripts for building and deploying the app.
4. **Collaborate:**
   * If you’re working in a team, use Git for version control and assign tasks using a project management tool like Trello or Jira.

**Day 1: Project Setup**

The goal for Day 1 is to set up the foundation for your project. This includes:

1. Setting up the **backend microservices** using Spring Boot and Maven.
2. Setting up the **frontend** using React and MUI.
3. Configuring the **database** (MySQL/PostgreSQL).
4. Setting up **version control** (Git).

Let’s go step by step.

**1. Backend Setup**

We’ll create a **multi-module Maven project** for the microservices. Each microservice will be a separate module.

**Step 1: Create a Parent Maven Project**

* Open your IDE (IntelliJ IDEA or Eclipse).
* Create a new Maven project named used-vehicle-app.
* In the pom.xml of the parent project, add the following:

<modules>

<module>eureka-server</module>

<module>api-gateway</module>

<module>admin-service</module>

<module>user-service</module>

<module>vehicle-service</module>

<module>booking-service</module>

</modules>

<packaging>pom</packaging>

**Step 2: Create Microservice Modules**

* Create the following modules under the parent project:
  1. **Eureka Server**: For service discovery.
  2. **API Gateway**: For routing requests to microservices.
  3. **Admin Service**: For CRUD operations on vehicles.
  4. **User Service**: For user registration and login.
  5. **Vehicle Service**: For listing and filtering vehicles.
  6. **Booking Service**: For booking test drives or buying vehicles.
* Each module should have its own pom.xml with the necessary dependencies (Spring Boot, Spring Cloud, Spring Data JPA, etc.).

**Step 3: Configure Eureka Server**

* Add the following dependency to the eureka-server module:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

* Add the @EnableEurekaServer annotation to the main class.
* Configure application.yml:
* server:
* port: 8761
* eureka:
* client:
* register-with-eureka: false
* fetch-registry: false

**Step 4: Configure API Gateway**

* Add the following dependency to the api-gateway module:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

* Configure application.yml:
* server:
* port: 8080
* spring:
* cloud:
* gateway:
* routes:
* - id: admin-service
* uri: lb://ADMIN-SERVICE
* predicates:
* - Path=/api/admin/\*\*
* - id: user-service
* uri: lb://USER-SERVICE
* predicates:
* - Path=/api/user/\*\*
* - id: vehicle-service
* uri: lb://VEHICLE-SERVICE
* predicates:
* - Path=/api/vehicle/\*\*
* - id: booking-service
* uri: lb://BOOKING-SERVICE
* predicates:
* - Path=/api/booking/\*\*

**Step 5: Add Common Dependencies**

* Add the following dependencies to each microservice (except Eureka Server):

xml

Copy

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

Run HTML

**2. Frontend Setup**

We’ll use **React** with **MUI** for the frontend.

**Step 1: Create a React App**

* Run the following command to create a React app: npx create-react-app used-vehicle-app-frontend

cd used-vehicle-app-frontend

* Install MUI: npm install @mui/material @emotion/react @emotion/styled

**Step 2: Set Up Routing**

* Install react-router-dom: npm install react-router-dom
* Create a basic routing structure in App.js:

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Home from './pages/Home';

import Admin from './pages/Admin';

import Login from './pages/Login';

function App() {

return (

<Router>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/admin" element={<Admin />} />

<Route path="/login" element={<Login />} />

</Routes>

</Router>

);

}

export default App;

**Step 3: Create Basic Pages**

* Create a pages folder and add the following files:
  1. Home.js: For listing vehicles.
  2. Admin.js: For admin panel.
  3. Login.js: For user login.

**3. Database Setup**

* Install MySQL or PostgreSQL on your machine.
* Create a database named used\_vehicle\_app.
* Configure the database connection in each microservice’s application.yml:

spring:

datasource:

url: jdbc:mysql://localhost:3306/used\_vehicle\_app

username: root

password: your\_password

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

hibernate:

ddl-auto: update

**4. Version Control**

* Initialize a Git repository:

git init

* Create a .gitignore file: gitignore

# Ignore node\_modules

node\_modules/

# Ignore IDE files

.idea/

\*.iml

# Ignore build files

target/

build/

* Commit your initial code:

git add .

git commit -m "Initial project setup"