Project ID- P10;

Project Name - Vehicle Service Booking

Project Description:
A web app where users can book vehicle servicing appointments and track service history.

Key Concepts Covered:
React, Redux, Routing, REST API, JWT Authentication

Capstone project directory structure

```
/FullStackApp
│── /client (React Frontend)
│   ├── /public
│   ├── /src
│   │   ├── /components
│   │   ├── /pages
│   │   ├── /store (Redux Store)
│   │   ├── /services (API Calls)
│   │   ├── App.js
│   │   ├── index.js
│   └── package.json
│
│── /server (ASP.NET Core API)
│   ├── /Controllers
│   ├── /Models
│   ├── /Data (Database Context)
```

```
│   ├── /Services
│   ├── /Middleware
│   ├── appsettings.json
│   ├── Startup.cs
│   ├── Program.cs
│   └── FullStackApp.sln
│
├── /database
│   ├── setup.sql (DB Schema and Seed Data)
│   ├── migrations/
│   ├── README.md
│
├── README.md
├── .gitignore
├── docker-compose.yml (Optional)
```

Project	User Story	Description	Acceptance Criteria

P10 - Vehicle Service Booking	Book a Service	Schedule vehicle maintenance.	Select date/service type, store bookings.
	Track Service History	View past services.	Fetch service records from MS SQL.
	Get Service Reminders	Receive maintenance reminders.	Calculate/send reminders, store notifications.

Evaluation Rubric for Project Assessment (85 Marks)

Criteria	Excellent (Full Marks)	Needs Improvement (Partial Marks)	Poor (No Marks)
React Frontend (60 Marks) **(Each criterion is 10 marks, 6 sub-criteria)**			
UI/UX Design	Clean, responsive, and user-friendly interface	Some UI issues, minor responsiveness problems	Poor UI, broken responsiveness
Component Reusability	Proper use of modular and reusable components	Some redundant components, minimal reusability	No reusable components, messy code
State Management	Efficient use of React hooks, Context API, or Redux	Some issues in state handling, unnecessary re-renders	Poor state management, frequent bugs
API Integration	API calls handled efficiently with error handling	Basic API calls but lacks proper error handling	API calls fail, no proper state update

Routing & Navigation	Proper use of React Router, seamless navigation	Some broken links or inefficient routing	No routing implemented or broken links
Performance Optimization	Uses memoization, lazy loading, and proper key handling	Some performance lags, missing optimizations	Poor performance, slow load times
.NET Core Web API (15 Marks) **(Each criterion is 5 marks, 3 sub-criteria)**			
API Design & Structure	RESTful API with proper naming conventions, modular structure	Some inconsistent API routes, minor structural issues	Poorly designed API, inconsistent endpoints
Business Logic & Validation	Properly implemented business logic with input validation	Some missing validation checks, minor logical flaws	Major logical errors, missing validation
Security & Error Handling	Implements authentication, authorization, and error handling	Some security flaws, minimal error handling	No security, major vulnerabilities
Database (10 Marks) **(Each criterion is 5 marks, 2 sub-criteria)**			
Schema Design	Well-structured, normalized database schema	Some redundant data, minor normalization issues	Poor schema design, redundant data
Query Performance	Optimized queries, proper indexing	Some unoptimized queries, minor performance issues	Poor query optimization, slow execution

VIVA Evaluation Rubric (15 Marks)

Criteria	Excellent (Full Marks)	Needs Improvement (Partial Marks)	Poor (No Marks)
Documentation, Viva, Presentation, Report **(15Marks)**			
Project Report & Documentation	Well-written documentation with clear API references	Incomplete documentation, missing some details	No documentation, poorly written
Viva & Presentation	Clear understanding of project, good presentation skills	Some difficulty explaining concepts	Poor explanation, lacks clarity