KPIT TECHNOLOGIES LIMITED

A project report on Vehicle Availability System

JAVA Track

Submitted by
By
Pragyan Prakhar
Sumit Mandal
Raj Gupta

Under the
Guidance of
Ruchi Gupta

Department of Computer Science / Software Engineering
KPIT Technologies Lt

GLA University, Mathura

Declaration

I hereby declare that the work presented in the project titled “Vehicle Availability System” is an original and authentic record of my own efforts carried out under the supervision of Ruchi Gupta.

This project was undertaken as part of my academic curriculum and in collaboration with KPIT Technologies Ltd.

The contents of this project report, in full or in parts, have not been submitted to any other organization or institution for any academic or professional recognition.

Name of the candidate : Pragyan Prakhar , Sumit Mandal , Raj Gupta

Superset id :

- **Pragyan Prakhar : 5468143**
- **Sumit Mandal : 5387731**
- **Raj Gupta : 5384344**

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to KPIT Technologies Private Limited for providing me with the opportunity to undertake my project and internship within their esteemed organization. The exposure to real-time challenges, professional work culture, and continuous learning at KPIT has been instrumental in enhancing my technical knowledge and practical skills. The support and guidance received during this internship greatly contributed to the successful completion of this project.

I am also sincerely thankful to my university for offering a strong academic framework that laid the foundation for this endeavor. I extend my appreciation to the Department of Computer Engineering and Applications for their consistent support and encouragement throughout my academic journey.

I would like to acknowledge the efforts of the faculty members, staff, and peers who have provided valuable insights, assistance, and motivation throughout the course of my studies and this project.

Lastly, I am profoundly grateful to my family for their unwavering support, constant encouragement, and belief in me during every phase of this journey

Name of Candidate: Pragyan Prakhar , Sumit Mandal, Raj Gupta

ABSTRACT

The rapid evolution of technology has transformed the transportation sector, creating a demand for intelligent, scalable vehicle management systems. The project "Vehicle Availability System" addresses this by building a robust, real-time platform for vehicle tracking, status management, and seamless user interaction across dealership operations.

Developed during my tenure as a Software Engineering Trainee (Java) at KPIT Technologies Ltd, the system leverages Core Java, Hibernate, PostgreSQL, Maven, and a React-based frontend to deliver a secure and user-friendly experience. It includes features such as role-based login/signup, vehicle status tracking, HTTP session management, and real-time vehicle updates.

By integrating a layered architecture of controllers, services, and DAOs, and using RESTful APIs with JSON responses, the project ensures scalability, maintainability, and ease of future integration with microservices. Real-time dashboards empower both users and dealers to monitor availability, edit vehicle details, and manage listings efficiently.

The Vehicle Availability System exemplifies the application of modern software engineering practices to solve real-world industry challenges, highlighting my growing expertise in backend development, database management, and building scalable enterprise-grade applications in the automotive and mobility domain.

CONTENTS

CHAPTER-1
INTRODUCTION

This chapter introduces the Vehicle Availability System project, providing an overview of its purpose, motivation, objectives, and a comparison with similar solutions in the industry. It also outlines the organization of the report.

1.1 OVERVIEW AND MOTIVATION

The transportation and automotive industry is rapidly evolving, demanding real-time, scalable, and reliable systems to manage fleet availability and utilization. The Vehicle Availability System, developed as part of the internship at KPIT Technologies Ltd, addresses these needs by streamlining the tracking and management of vehicles for improved operational efficiency.

The motivation behind this project stems from:

- The growing need for real-time monitoring of vehicle availability, particularly in fleet-based and service-driven industries.
- The challenges in managing and updating vehicle status data manually, which can lead to inefficiencies and miscommunication.
- The importance of automating and simplifying vehicle management tasks to enhance productivity and decision-making for both administrators and users.

1.2 OBJECTIVE

The primary goal of the Vehicle Availability System is to develop a scalable, secure, and user-friendly platform that:

1. Provides real-time vehicle availability information through a centralized system accessible to both users and administrators.
2. Implements role-based access control to ensure that only authorized users (such as

dealers or admins) can perform specific operations like adding, updating, or removing vehicle data.

3. Facilitates efficient vehicle tracking and management by integrating backend technologies such as Core Java, Hibernate ORM, and PostgreSQL for seamless data storage and retrieval.

4. Delivers an intuitive frontend interface using ReactJS to enhance user experience and allow easy interaction with the system.

5. Ensures data integrity and security through proper session handling and validation mechanisms to prevent unauthorized access or data manipulation.

1.3 SUMMARY OF SIMILAR APPLICATION

Several platforms exist for managing vehicle data and availability, such as:

- **CarDekho and Droom**:These are commercial vehicle listing platforms that offer rich user interfaces but primarily cater to end-users without offering backend customization for enterprise use.Blackboard: Offers comprehensive course management features but lacks adaptive learning mechanisms.

- **Fleet Management Systems:**These tools offer comprehensive tracking and logistics but are often expensive and complex to integrate into smaller-scale or internal systems.

- **Custom Dealer Portals**: Many dealerships develop their own tools for vehicle inventory, but these systems often lack scalability, modern interfaces, or real-time updates.

1.4 ORGANIZATION OF PROJECT REPORT

The report is structured to provide a comprehensive overview of the Vehicle Availability System project, covering its motivation, design, implementation, and evaluation strategies. The following is an outline of the chapters included in this report:

Chapter 1: Introduction

This chapter introduces the Vehicle Availability System project, explaining its background, objectives, and relevance within an enterprise environment. It also highlights comparisons with existing vehicle tracking and management systems.

- **1.1 Overview and Motivation**: Describes the importance of efficient vehicle availability tracking in organizations and the role of a well-structured software system.
- **1.2 Objective**: Defines the core goals of the project, including real-time vehicle tracking, role-based access, and integration with modern tech stacks.
- **1.3 Summary of Similar Applications**: Discusses other platforms and highlights the unique aspects of this system tailored for internal use at KPIT.
- **1.4 Organization of the Project Report**: Provides an outline of the subsequent chapters in the report.

Chapter 2: Software Requirement Analysis

This chapter outlines the software and hardware requirements necessary for the development and deployment of the system. It also provides a breakdown of the system's core modules.

- **Requirement Analysis**: Lists the required technologies like Core Java, Hibernate, PostgreSQL, Maven, and React.
- **Feasibility Analysis**: Evaluates the technical, operational, and economic feasibility of the project.
- **Module Description:**Describes key modules including login/signup system, vehicle database integration, and API endpoints.
- **Use Cases:** Includes sample scenarios and interaction flows between users (admin, dealer, general users) and the system.

Chapter 3: Software Design

This chapter presents the architectural and structural design of the system through visual and textual representations.

- **Data Flow Diagrams**: Depicts Level 0 and Level 1 data flows to illustrate the flow of information within the system.

- **UML Diagrams:** Provides Class, Sequence, and Use Case diagrams to represent relationships and interactions.
- **Database Design:** Contains the E-R diagram along with table structures, primary/foreign keys, and relationships managed using PostgreSQL and Hibernate ORM.

Bibliography

Includes all references, online documentation, tools, libraries, and frameworks used during the research and development phases of the project.

Appendices

Additional content such as test cases, sample JSON responses, HTTP request examples, and screenshots from the UI to support understanding of implementation detail

CHAPTER-2

SOFTWARE REQUIREMENT ANALYSIS

This chapter presents a detailed analysis of the software requirements for the Vehicle Availability System. It outlines the essential components needed for the system's functionality, including the technical and functional requirements. Additionally, it evaluates the feasibility of the system's implementation within an enterprise context and provides detailed descriptions of each module. Real-world use cases are also included to demonstrate the practical applicability and workflow of the platform.

The system is developed using Core Java, Hibernate (ORM), PostgreSQL for the backend, and React for the frontend. The application architecture follows an MVC (Model-View-Controller) pattern to ensure scalability, maintainability, and clean separation of concerns. The project includes role-based access control through a login and signup system, and it interacts with the database to manage and display real-time vehicle availability.

2.1 REQUIREMENT ANALYSIS

To successfully develop the Vehicle Availability System, it is essential to understand the functional and non-functional requirements of the system, along with the necessary hardware, software, and technologies. The system is aimed at streamlining the process of checking vehicle status and availability for both customers and dealers, providing a centralized and user-friendly interface with secure access and robust backend processing.

2.1.1 FUNCTIONAL REQUIREMENTS

1. **User Features:**
   - User Registration and Login System:
     - The Vehicle Availability System provides a secure and role-based login mechanism. Users are categorized as either Customers or Dealers. During

    registration, users fill in personal details including first name, last name, email, username, and password. Depending on the selected role (isDealer boolean flag), users are redirected to their respective dashboards after login. HTTP sessions are implemented to track logged-in users and manage authentication without repeated login prompts. These sessions ensure that only authenticated users can access restricted features.

- View Available Vehicles.
    - Once logged in, customers can view a real-time list of all available vehicles. The system displays vehicle details including model, number, type, location, and status. The availability status is dynamically updated by dealers and shown as 'Available' or 'Not Available'. Customers can apply filters based on vehicle type (e.g., Sedan, SUV), location (e.g., city or dealership), and availability. This helps in quickly identifying vehicles that meet their requirements without browsing the entire inventory.

2. **Dealer Features:**
    - Dealer Dashboard:
        - Dealers have access to an exclusive dashboard where they can manage vehicles listed under their inventory. The dashboard presents a summary view of all vehicles owned or managed by the logged-in dealer.
        - Add New Vehicle: Dealers can add new vehicles by providing key information like vehicle model, registration number, type, location, and current availability. The system validates inputs and ensures data consistency before updating the database.
        - Edit Vehicle Details: Dealers can edit existing vehicle records to update location, availability, or other relevant details. This ensures the inventory stays accurate and up-to-date.
        - Delete Vehicles: Dealers can delete vehicles from their inventory. This action marks the vehicle as removed in the database and makes it unavailable to end-users.
3. **System Features:**

- Secure Authentication:
  - Authentication is handled securely with hashed passwords and session management. Passwords are encrypted before storage, ensuring user data safety. HTTP sessions are initiated at login and destroyed at logout, preventing unauthorized access.
- RESTful API Endpoints: The backend of the Vehicle Availability System is driven by RESTful APIs. These endpoints interact with the frontend and ensure smooth data exchange:
  - GET /vehicles: Returns a list of all available vehicles in JSON format.
  - GET /vehicles?id=ID: Fetches details of a specific vehicle by its unique ID.
  - POST /addVehicle: Allows dealers to add a new vehicle by sending vehicle data as a JSON payload.
  - POST /editVehicle: Updates the information of a specific vehicle.
  - POST /login: Authenticates a user with credentials and initiates an HTTP session
  - POST /signup: Registers a new user with role-based details.
  - POST /deleteVehicle: Deletes or deactivates a vehicle based on its ID.
- JSON-based Data Exchange:
  - All data is transferred using JSON format. The org.json library in Java is used to handle API responses and requests. This ensures lightweight and consistent data exchange between the frontend and backend.
- React-based Frontend :
  - Modern Frontend Framework: The system will use React JS for a responsive and modern frontend experience.
  - Responsive & Accessible UI: Designed to work seamlessly across all devices with easy navigation and accessibility standards.
  - Customer Dashboard: Customers will be able to view available vehicles, see the details of each available vehicle in detail but won’t be able to perform any restricted action(edit the vehicle detail,deleting the vehicle).
  - Dealer Dashboard: Dealers will manage vehicle listings, track availability, and update inventory.

- Real-Time Backend Integration: RESTful APIs will allow seamless real-time data syncing with the backend.
- Mobile-Friendly Views: The UI will adapt to mobile screen sizes for on-the-go accessibility.
- Advanced Components (Future): Plans include integrating charts, popups, and multilingual support for global usability.

2.1.2 NON-FUNCTIONAL REQUIREMENTS

1. **Performance**:
   - The Vehicle Availability System is designed to efficiently handle multiple users accessing the application simultaneously. The backend has been optimized to minimize delays and provide a smooth experience even during high load conditions.
   - APIs have been structured using RESTful principles, ensuring quick and reliable data fetching. Queries for vehicle availability and other operations such as login, signup, and vehicle management are designed to deliver responses with minimal latency.
   - The backend, deployed locally via Apache Tomcat server, demonstrates stable performance in a development environment, with scope for scaling in production deployment
2. **Reliability**:
   - The system is built to maintain high availability. Through the use of Java, Hibernate, and PostgreSQL, it ensures reliable backend operations.
   - Database transactions are designed to be atomic and consistent. This means that operations such as adding, editing, or deleting vehicle entries are completed fully or not at all, reducing risks of data corruption.
   - Exception handling mechanisms and transaction rollbacks are in place to ensure system reliability even in the case of unexpected errors or partial failures
3. **Security**:
   - User credentials are stored securely using password hashing techniques. This ensures that even if the database is compromised, the actual passwords are not exposed.

- HTTP sessions are implemented to maintain persistent and secure login states across user interactions. This prevents session hijacking and unauthorized access to protected features.
- Role-based access control has been enforced throughout the application. Only authenticated users can perform operations based on their designated role: Customers can view and filter available vehicles, while Dealers can manage their vehicle inventory.
- Protected routing has been implemented in the frontend using React + Redux. Unauthorized access to pages like dashboards is restricted, improving application security.
- HTTP session management should be enforced for secure and persistent login sessions.

4. **User Experience**:
   - The user interface, developed using React and Redux, offers a responsive and visually clean experience. The design prioritizes ease of navigation for both customers and dealers.
   - All key pages such as login, signup, dashboards, and vehicle lists feature real-time interactivity. Users can view vehicle data dynamically without reloading the page.
   - Although the booking feature is not currently implemented, the frontend design supports future integration for booking modules.
   - The application supports dynamic updates via Redux-managed state, ensuring consistency in data presentation across components.
   - Features like protected routing, status indicators, and future multilingual support modules are part of the frontend roadmap.
   - The local deployment using Tomcat has enabled effective testing of the full stack flow, ensuring all frontend-backend interactions perform seamlessly.

2.1.3 HARDWARE REQUIREMENTS

- Servers:
  - The backend system can be deployed on either local servers or cloud-based infrastructure capable of running a Java Servlet or Spring Boot application.

- Apache Tomcat server is used for local development and deployment, ensuring compatibility with Java EE web applications.
- A PostgreSQL server is required to store structured data including user profiles, vehicle details, and booking records.
- To ensure smooth operation, the server should be equipped with at least 8GB of RAM and a multi-core processor for managing concurrent threads and database operations efficiently.
- For a production-ready environment, cloud services like AWS, Azure, or Heroku can be considered for scalable deployment.

- User Devices:
    - End-users can access the system from any internet-enabled device such as a laptop, desktop, tablet, or smartphone.
    - A stable internet connection is required to load data and interact with the application in real-time.
    - The React-based frontend is compatible with modern browsers including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.
    - Touch responsiveness and adaptive layouts ensure that mobile users have a consistent and smooth browsing experience.
    - Since no specific hardware requirements are imposed on the client side, the application remains accessible to a broad user base without specialized devices.

2.2 FEASIBILITY ANALYSIS

2.2.1 TECHNICAL FEASIBILITY

The technical feasibility of the Vehicle Availability System has been thoroughly analyzed to ensure that the selected tools and technologies are not only appropriate for the system's requirements but also robust, scalable, and maintainable for real-world deployment in an enterprise environment.

- **Backend Infrastructure**:

Core Java is leveraged as the foundation for backend logic, offering several key benefits:

- It provides platform independence, allowing the application to run on any system with a Java Virtual Machine (JVM).
- It supports object-oriented programming principles, enabling better modularity, reusability, and code maintainability.
- It enables structured code organization, promoting a clean and scalable project architecture.
- It ensures efficient memory management, contributing to better runtime performance.
- It offers multi-threading capabilities, making it suitable for handling concurrent user requests and real-time data manipulation effectively.

Hibernate ORM (Object Relational Mapping) is employed to manage database interactions:

- It abstracts away the complexities of writing manual SQL queries by mapping Java classes directly to database tables.
- This approach enhances code readability, allowing developers to work with Java objects rather than raw SQL.
- It also improves maintainability, reducing the need to frequently modify SQL queries when data models evolve.

Maven is used as the build and dependency management tool for the project:

- It ensures that all required libraries, project modules, and plugins are managed in a consistent manner.
- Maven helps maintain uniformity across different development environments, making project setup and collaboration easier.
- The chosen technology stack enables the backend to:

- Efficiently manage vehicle data with robust backend logic.
- Perform all essential CRUD operations (Create, Read, Update, Delete) on the database.
- Maintain consistent session states across users, ensuring a seamless and reliable user experience.

- **Web Technologies:** The backend is implemented using Java Servlets and RESTful APIs, which together form the communication bridge between the frontend and backend components.

  Java Servlets play a crucial role in backend processing:

  - They are responsible for managing HTTP requests, acting as controllers within the server-side logic.
  - They handle session management, ensuring consistent user sessions during interactions.
  - They perform user authentication, verifying credentials and managing secure access.
  - They route incoming requests to the appropriate business logic, facilitating organized request handling.

  RESTful APIs are used to define a standardized interface:

  - They follow standard HTTP methods such as GET, POST, PUT, and DELETE to enable structured communication.
  - These APIs support stateless operations, which enhances scalability and simplifies client-server interactions.
  - They are responsible for exposing vehicle information, allowing the frontend to fetch and display relevant data.

  - They handle user registration and login, ensuring secure user onboarding and access.
  - They manage vehicle records, including creating, updating, retrieving, and deleting vehicle-related data.

  The API endpoints are well-structured and easily extendable:

  - This design approach supports smooth integration with frontend frameworks like React.
  - It ensures that the backend can grow and adapt as project requirements evolve, without disrupting existing functionalities.

- **Frontend Plan**: The frontend is developed using React.js, a modern JavaScript library known for its efficiency and flexibility:

  - It provides a component-based architecture, enabling the creation of dynamic and reusable UI elements.
  - It utilizes a virtual DOM mechanism, ensuring optimal performance by minimizing real DOM updates.
  - It supports the creation of responsive layouts, allowing the interface to adapt across different device screen sizes.

  Customer-facing interface includes features to:

  - View available vehicles, offering clear visibility into listed inventory.

  Dealer-facing interface supports administrative controls such as:

  - Adding new vehicles into the system.
  - Editing existing vehicle entries for updates or corrections.
  - Managing availability status of vehicles for accurate listings.

- Deleting the uploaded vehicle from the system.

Redux is integrated for state management, which:

- Ensures consistent data flow across the application's components.
- Helps maintain uniform UI behavior even as users interact with different parts of the system.

Frontend communicates with the backend via RESTful APIs:

- Enables real-time data exchange between the UI and backend logic.
- Facilitates interaction with server-side features like vehicle management and user operations.

- **Session Management :**HttpSession is implemented to manage user authentication and session state securely:
    - When a user logs in, a new session is created to track their interaction with the system.
    - This session is used to store critical user information, such as:
        - User ID, which uniquely identifies the logged-in user.
        - User role, indicating whether the user is a customer or a dealer.
        - Login timestamp, recording the exact time the session was initiated.

    Session persistence is ensured:

    - The session remains active until:
        - The user logs out manually, or
        - The session times out due to inactivity.
        - This ensures that session-related data remains available for the duration of

valid user activity.

Role-based session validation is enforced to enhance security:

- Users are restricted from accessing unauthorized routes or performing operations outside their permissions.
- This mechanism ensures that:
- Customers cannot access dealer functionalities.
- Dealers cannot perform actions reserved for system-level controls (if any).

This session management approach adds a crucial security layer:

- It safeguards sensitive operations by allowing only authenticated users to perform them.
- It aligns with best practices in secure session handling, promoting a safer and more robust application architecture.

- **Database** : The system utilizes PostgreSQL as its relational database:
    - Known for its stability, making it reliable for production-grade applications.
    - Offers high performance in query execution and data retrieval.
    - Supports advanced query optimization, helping improve the speed and efficiency of complex data operations.

    PostgreSQL is well-suited for:

    - Handling complex queries, making it ideal for systems with intricate data relationships.
    - Efficient indexing, which improves search and retrieval performance.
    - Transaction management, ensuring consistency and reliability in operations

like insert, update, or delete.

The database stores structured information such as:

- User details, including identification and role.
- Vehicle data, including type, status, and ownership.
- Availability status, helping track which vehicles are currently accessible.
- Other related system data required for full functionality.

Relationships between different entities are well-defined:

- These relationships help maintain data consistency and referential integrity.

Hibernate ORM acts as a bridge between Java code and the PostgreSQL schema:

- Provides an abstracted and simplified way to handle database operations.
- Eliminates the need for writing manual SQL queries by mapping Java classes to database tables.
- Enables seamless data handling within the object-oriented Java backend.

2.2.2 ECONOMIC FEASIBILITY

The economic feasibility of the Vehicle Availability System has been carefully analyzed to ensure that the project not only delivers high-quality features and performance but also remains cost-effective and scalable in the long term. Multiple strategies have been adopted in terms of technology selection, resource allocation, and architecture design to minimize expenses while allowing room for future growth.

- **Use of Open-Source Stack**: The system has been built using a variety of open-source tools and frameworks, including:
  - **Core Java** for backend logic and object-oriented programming support.

- PostgreSQL for relational data management.
- Maven for build automation and dependency resolution.
- Hibernate for ORM-based database abstraction.
- React.js for creating dynamic and modular frontend interfaces.

These technologies are:

- Free to use, which helps avoid any upfront or recurring license costs.
- Widely adopted and community-supported, offering rich documentation, regular updates, and stability.

By using open-source software:

- The project incurs zero licensing fees, significantly reducing budget requirements.
- It remains flexible and vendor-independent, allowing modifications and enhancements without proprietary restrictions.
- Developers can leverage existing tools, plugins, and libraries to accelerate development time without incurring extra costs.

Additionally, Maven enhances productivity by:

- Managing project dependencies, builds, and plugins efficiently.
- Saving time and effort during the compilation and testing stages of development.

- **Resource Optimization and In-House Development**: The system was developed entirely in-house during the internship period at KPIT Technologies Ltd., which resulted in :

- A drastic reduction in labor costs, as the work was performed as part of a learning-based internship program.

- Avoidance of external hiring or contractual development expenses.

Hosting and deployment were handled locally using Apache Tomcat, which provided:

- A cost-free solution for running and testing Java-based applications.
- The ability to conduct extensive testing, debugging, and iteration in a safe, offline environment.
- Elimination of the need for paid cloud services or hosting subscriptions during development.

This local setup not only kept development affordable but also allowed developers to:

- Simulate real-time environments, ensuring quality and accuracy.
- Rapidly fix bugs and iterate features without waiting for production-level deployment.

- **Scalable and Modular Architecture :** The system is purposefully designed to be modular, extensible, and scalable, which ensures that:

- It can be expanded in phases, rather than requiring a large upfront investment in infrastructure.
- Additional users, features, or datasets can be supported by making small, incremental enhancements.

Database scalability is addressed through:

- A normalized PostgreSQL schema, which minimizes data redundancy and supports efficient query execution.

- Optimizations that make it capable of handling increased data volumes without structural changes.

Frontend and backend scalability:

- The frontend components built in React are organized for reuse and dynamic rendering.
- Backend services are decoupled and capable of independent scaling, making the system cloud-ready.

Future-readiness:

- If needed, both frontend and backend components can be containerized using Docker.
- Deployment can be transitioned to cloud orchestration platforms like Kubernetes, enabling horizontal scaling based on user load.
- This ensures that the system can grow sustainably, without redesigning the entire architecture.

2.2.3 OPERATIONAL FEASIBILITY

The operational feasibility of the Vehicle Availability System has been thoroughly assessed to ensure that it not only functions effectively across different user roles but also remains easy to use, maintain, and scale in the future. The system's design philosophy emphasizes usability, role clarity, and backend modularity—allowing it to be adaptable for long-term deployment and ongoing development.

- **User Role Segregation:**
  - Clear separation of functionality based on user roles (Admin, Dealer, Customer) simplifies the user experience and enhances usability.
  - Admins are envisioned to have full control over the system:

    - Monitoring system status.
    - Managing global data integrity.
  - Performing administrative actions across users and vehicles.
  - Dealers have access to a dedicated interface that allows them to:
    - Add, update, or remove vehicles from their inventory.
    - Monitor availability statuses.
    - Handle relevant data fields associated with vehicles they manage.
  - Customers have access to:
    - View and explore available vehicles.
    - Check real-time availability, and other essential vehicle details.
    - This role-based interface ensures:
      - Streamlined and context-specific user experience.
      - Reduced risk of unauthorized actions.
      - Improved system reliability and data security by ensuring users only see and modify data relevant to them.

- **Interface Simplicity**: The system's frontend, built with React.js, is crafted with user experience at the forefront:
  - The system's frontend, built with React.js, is crafted with user experience at the forefront.
  - Clean layouts and responsive design ensure compatibility across devices (laptops, tablets, smartphones).
  - Interactive UI elements such as dropdowns, clickable cards, and simplified

forms improve accessibility.

- Key features of the user interface include:
  - Role-specific dashboards that highlight relevant tasks and information.
  - Minimal clutter, with only necessary inputs and actions visible per role.
  - Clear navigation structure, with emphasis on ease of use for non-technical users.
- Even users unfamiliar with enterprise systems can:
  - Navigate seamlessly through the application.
  - Understand the flow of actions.
  - Complete their objectives without a steep learning curve.
- This focus on UI/UX helps improve:
  - Stakeholder satisfaction.
  - User engagement and retention.
  - Overall operational efficiency of the system.

- **Maintainability and Developer-Friendly Architecture :**
  - The backend is built on a well-structured, three-layer architecture**:**
    - Controller Layer – Handles HTTP requests and defines routing logic.
    - Service Layer – Encapsulates business logic and acts as a bridge between the controller and DAO.
    - DAO Layer – Handles interactions with the database using Hibernate ORM.
  - This modular structure promotes:

    - Separation of concerns, making code easier to test and debug.
    - Scalability and ease of updates, since changes in one layer do not impact others directly.
- For example:
    - Changes in the database schema or retrieval logic can be made solely in the DAO.
    - Service logic can evolve independently to incorporate new business rules.
    - The controller can handle new routes without disrupting backend logic.
- The maintainability factor ensures that:
    - The system is not a one-time deployment but a sustainable, long-term solution.
    - Future enhancements, such as new features or integrations, can be added with minimal technical debt.

2.3 MODULE DESCRIPTION

The Vehicle Availability System is structured into clearly defined modules to support distinct functionalities for different user roles. These modules are designed to ensure operational efficiency, real-time data handling, and a smooth user experience. Below is a detailed breakdown of the functional modules by user role:

2.3.1 CUSTOMER MODULE

- **Vehicle Browsing Dashboard:**
    - Customers are provided with a dedicated dashboard where they can conveniently browse the full catalog of vehicles listed on the platform.
    - Each vehicle listing displays key attributes, including:

        - Vehicle model and type (e.g., SUV, Sedan, Hatchback).
        - Availability status, such as:
            - Available
            - Booked
            - Under Maintenance
    - Allows customers to view all available vehicles with details like model, type, availability status, and location.

2.3.2 DEALER MODULE

- **Vehicle Management:**
    - Dealers access a secure and feature-rich dashboard that allows complete management of their vehicle inventory.
    - Through this panel, dealers can:
        - Model name
        - Manufacturing year
        - Color
        - Mileage
        - Vehicle Type
        - Status
    - Edit or update existing vehicle data to reflect:
        - Availability status changes
        - Location modifications
        - Changes in rental terms or pricing
    - Remove vehicles from the listing if no longer available.
    - These functionalities ensure the inventory remains accurate, organized, and up-to-date.
- **Status Updates:**
    - Dealers are empowered to update vehicle status in real time, marking them as:
        - Available
        - Booked
        - Under Maintenance
    - Once a status is updated, the change is instantly reflected on the customer-facing

dashboard.

- The integration between dealer actions and frontend visualization guarantees smooth operational flow, supporting seamless coordination across the system.

2.4 FUNCTIONALITIES OF EACH MODULE

Module	**Functionality**
Customer Module	Browse available vehicles, request bookings, view booking status, manage profile.
Dealer Module	Add/update/delete vehicles, manage availability status, approve booking requests.
Cross-Platform	Role-based secure login (using HTTP Sessions), scalable backend with PostgreSQL & Hibernate, cloud-hosted with support for session management and data persistence.

2.5 USE CASES

Use Case 1: View Available Vehicles

Actors: Customer

Description: Customers log in and access a dashboard displaying the list of available vehicles along with details like model, status, and booking options.

Use Case 2: Update Vehicle Status

Actors: Dealer

Description: Dealers log in to update the status of their vehicles (e.g., Available, Booked, Under Maintenance), add new vehicles, or remove existing ones.

This chapter defines how each type of user interacts with the system and lays the foundation for the Software Design in the next chapter, where diagrams and database design will be explored.

CHAPTER 3

SOFTWARE DESIGN

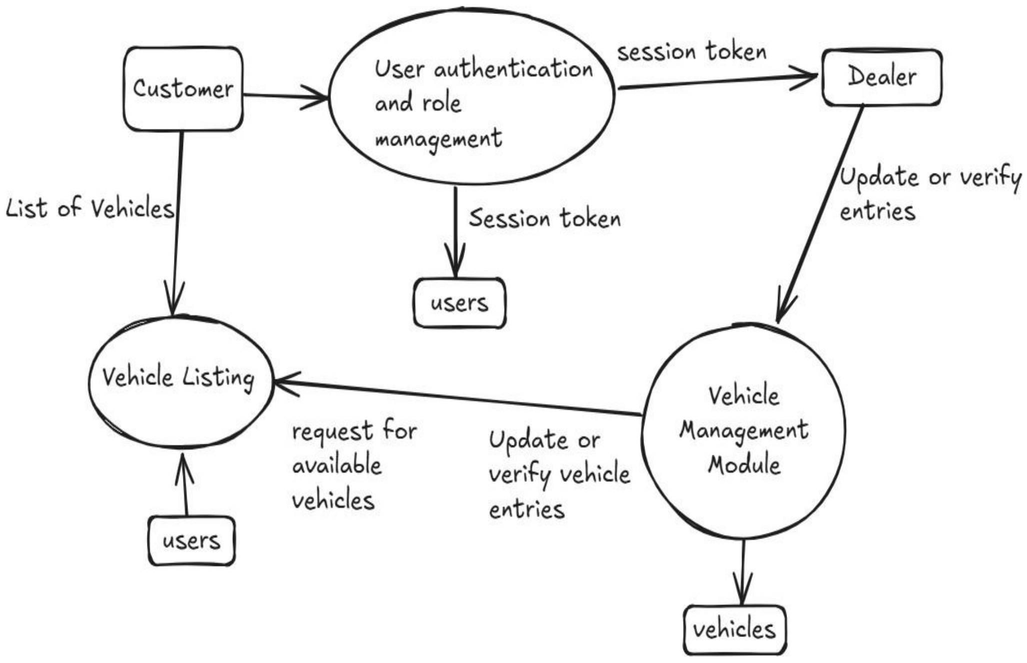
This chapter focuses on the design of the Vehicle Availability System, outlining the architectural structure, key workflows, and database design. It includes Data Flow Diagrams (DFD), UML diagrams, and relational schema for backend development using Java, PostgreSQL, Hibernate, and HTTP Sessions.

3.1 DATA FLOW DIAGRAM (DFD)

Level 0: Context Diagram

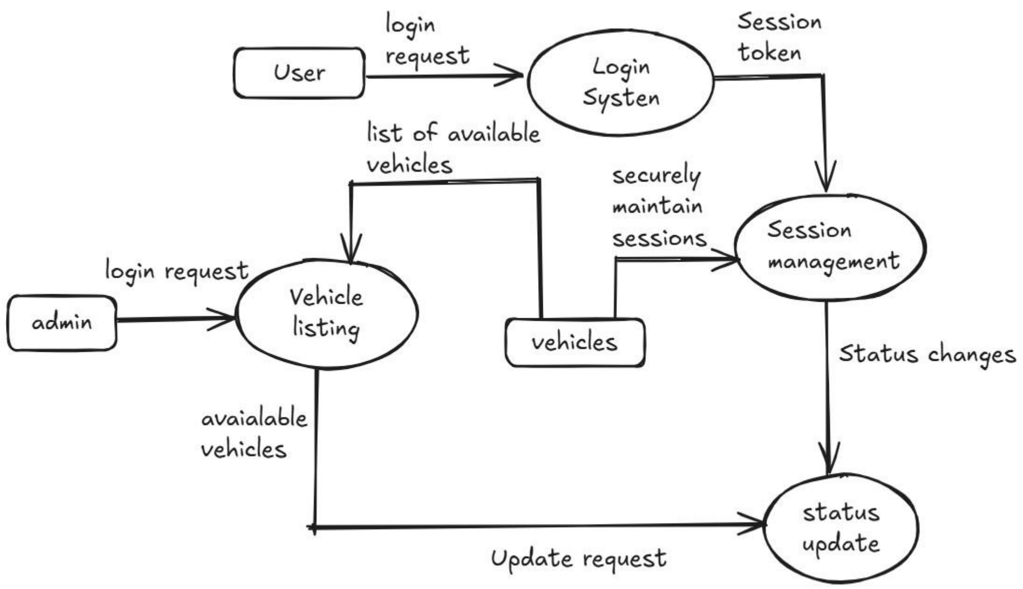
At this level, the system is visualized as a single process interacting with external users and systems.

1. **Entities**:
   - Customers
   - Dealers
   - Admin(optional)
2. **Processes**:
   - Authentication and Session Handling
   - Vehicle Status Management
   - Vehicle Listing
3. **Data Stores**:
   - PostgreSQL Tables: users, vehicles


Customer
User authentication and role management
session token
Dealer
List of Vehicles
Session token
Update or verify entries
users
Vehicle Listing
Vehicle Management Module
request for available vehicles
Update or verify vehicle entries
users
vehicles

Level 1: Detailed Data Flow

- **Login System**: Authenticates users and creates HTTP session objects based on roles (customer/dealer).
- **Vehicle Listing**: Retrieves available vehicles using Hibernate from the vehicles table, filtered by status.
- **Status Update :** Dealers update vehicle availability status via a POST or PUT request.
- **Session Management :** HTTP Sessions track users identity and roles securely across requests.



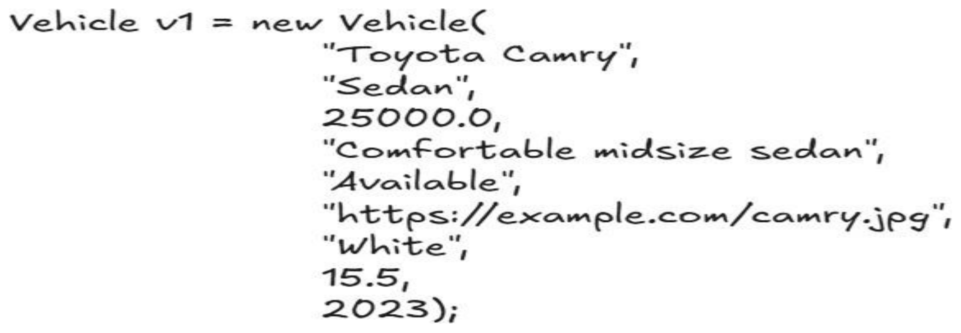
3.2 UML DIAGRAMS

Class Diagram

Class	**Attributes**	**Description**
User	userId, username, email, password, isDealer	Common fields for all users.
Vehicle	vehicleId,model,status,color,mileage,year,description	Represents a vehicle and its availability status

Object Diagram

- An instantiated Vehicle object:



```
Vehicle v1 = new Vehicle(
                    "Toyota Camry",
                    "Sedan",
                    25000.0,
                    "Comfortable midsize sedan",
                    "Available",
                    "https://example.com/camry.jpg",
                    "White",
                    15.5,
                    2023);
```

It's basically initializing a Vehicle object with complete details like model, price, availability, etc., so it can be stored or displayed in the system.

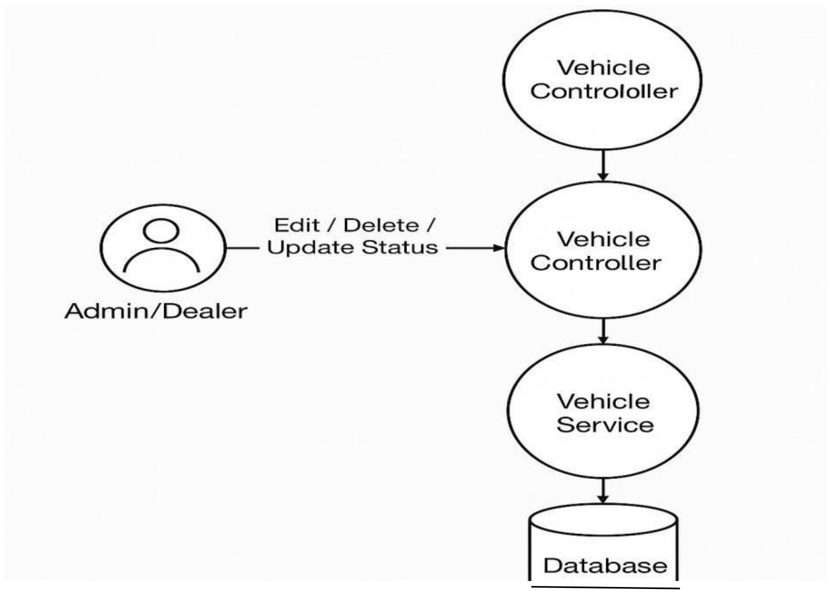
Sequence Diagram

- **Scenario**: A user firsts registers and then logs in and then fetches the list of the vehicles and dealer can add,delete and manage the status of the vehicles.

1. **Step 1**: **Login**
   · User submits credentials
   · Backend authenticates using DAO/service layer
   · HTTP session is created (role is stored in session, e.g., dealer/user).
2. **Step 2**: **Browsing Vehicles**
   · GET /vehicles request
   · Backend calls VehicleService → VehicleDAO → Hibernate →
   · Returns list of vehicles as JSON.
3. **Step 3**: **Edit Vehicle(Only Dealer)**
   · PUT /vehicles/{id} with updated vehicle data.
   · Backend validates and updates record in PostgreSQL using Hibernate.
   · Returns success response.
4. **Step 4**: **Delete Vehicle(Only Dealer)**
   · DELETE /vehicles/{id}
   · Backend deletes the vehicle record from PostgreSQL
   · Sends confirmation response.
5. **Change Vehicle Status (e.g., Available → Booked / Under Maintenance)**
   ·POST/vehicles/{id}/status with new status (e.g., "Booked", "Available", etc.).
   ·Backend updates status in database.
   ·Sends response confirming status change.

Collaboration Diagram

- This Collaboration Diagram shows how Admin/Dealer interacts through UI to edit, delete, or update vehicle status, and how the backend layers (controller → service → DAO) communicate to perform those actions in the PostgreSQL database.



3.3 DATABASE DESIGN

Tables:-

1. **User Table**

Field	**Type**	**Description**
User_id	Serial primary key	Unique identifier for each user
firstName	varchar	First name of the user
lastName	varchar	Last name of the user
email	varchar	email of the user
username	varchar	Unique name of every user.
password	varchar	Hashed password for security
isDealer	boolean	Indicates if the user is dealer or customer

2. **Vehicles Table**

Field	Type	Description
Vehicle_id	Serial primary key	Unique identifier for each vehicle
name	Varchar(255)	Name of the vehicle
type	Varchar(255)	Type of the vehicle(Sedan,SUV)
price	Double precision	Price of the vehicle
Description	TEXT	Description of the vehicle
Status	Varchar(255)	Status of the vehicle(Reserved,Available)
Image_url	Varchar(255)	Url containing the image of the vehicle
Mileage	Double precision	Shows the mileage of the vehicle
Year	Integer	Manufacturing year of the vehicle

Stored Procedure(PostgreSQL functions)

- Update Vehicle Status :-

  A function to update the availability status of a vehicle (e.g.,"Available", "Booked", "Unavailable").
- Delete Vehicle

  A function to safely delete a vehicle by its ID if certain conditions are met.
- Edit Vehicle Details

  A function to allow dealers or admins to update vehicle information like price, description, color, etc.
- Get Available Vehicles

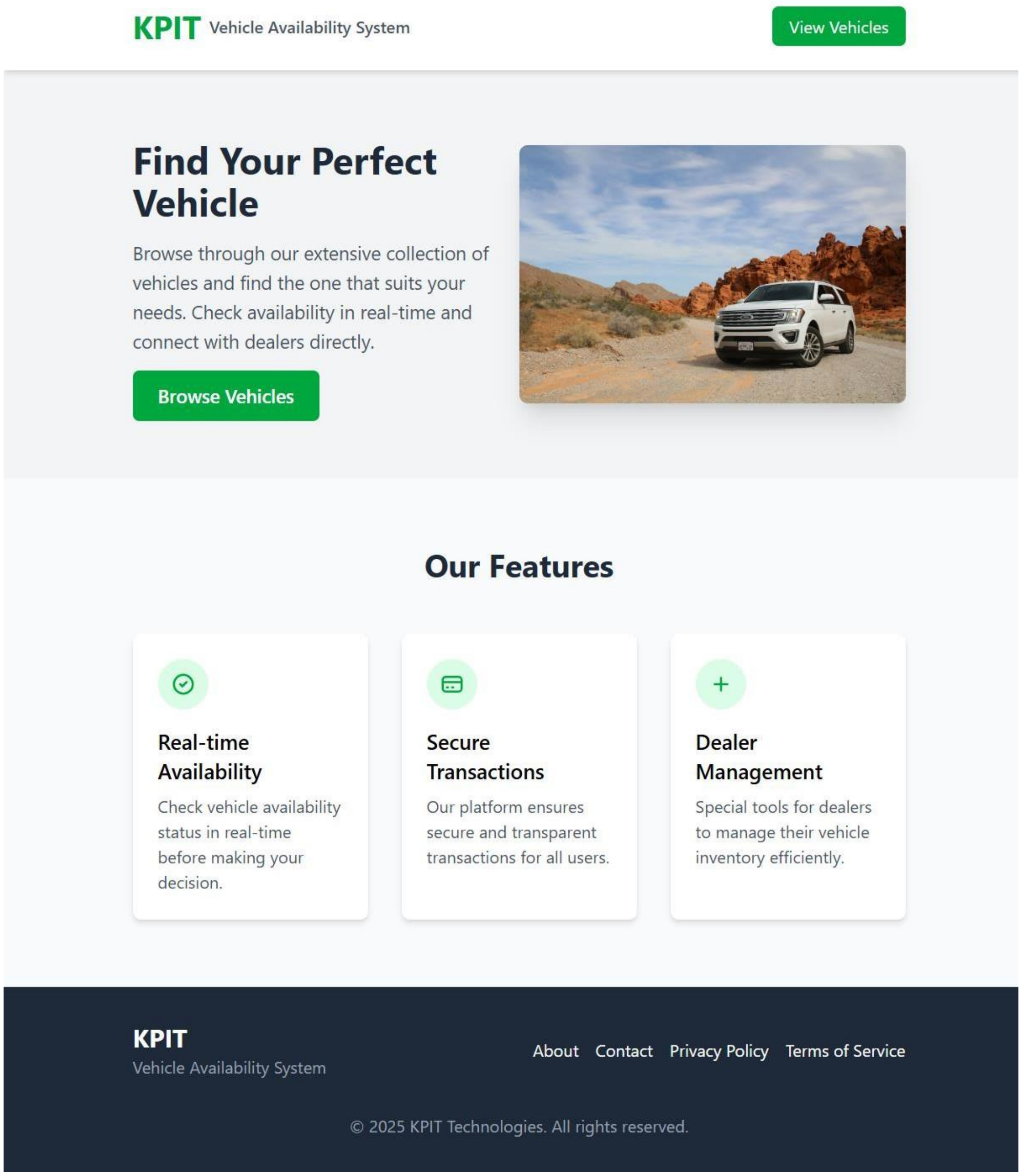
  A function to retrieve all vehicles with the status "Available" for the customers to view.
- Get Vehicles By Id

A function to get the details of the particular vehicle by id.

CHAPTER 4

IMPLEMENTATION AND USER INTERFACE

- **Landing Page**



KPIT Vehicle Availability System

View Vehicles

Find Your Perfect Vehicle

Browse through our extensive collection of vehicles and find the one that suits your needs. Check availability in real-time and connect with dealers directly.

Browse Vehicles

Our Features

Real-time Availability

Check vehicle availability status in real-time before making your decision.

Secure Transactions

Our platform ensures secure and transparent transactions for all users.

Dealer Management

Special tools for dealers to manage their vehicle inventory efficiently.

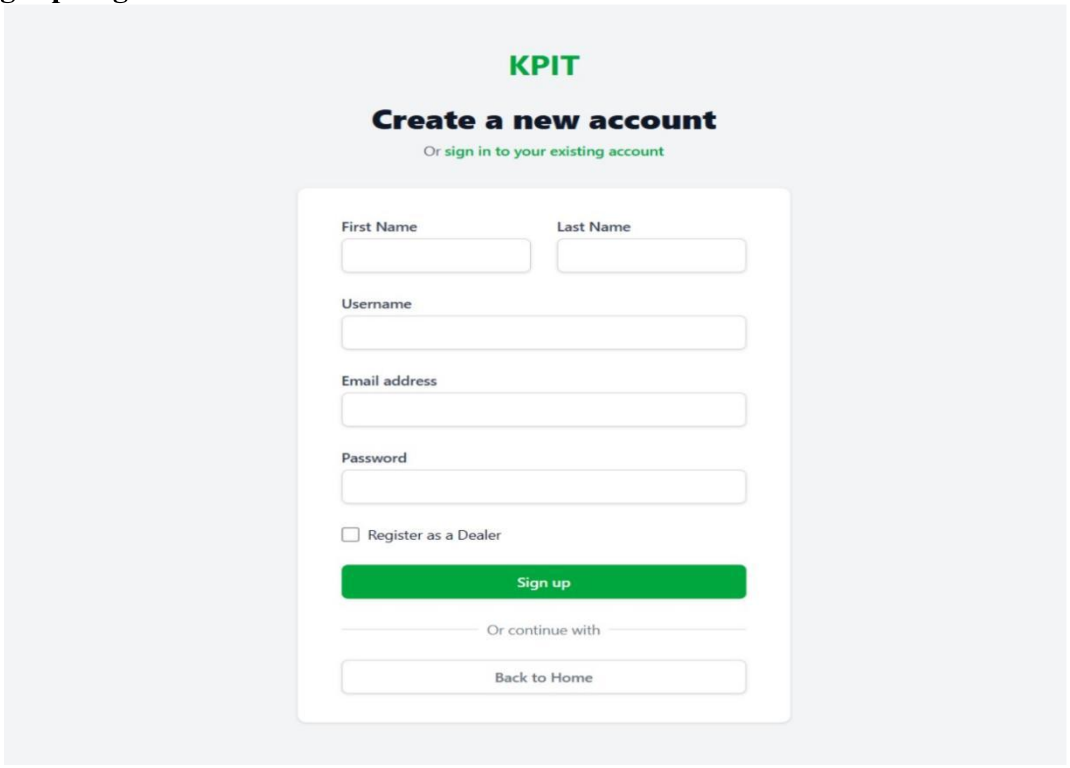
KPIT

Vehicle Availability System

About Contact Privacy Policy Terms of Service

© 2025 KPIT Technologies. All rights reserved.

- **Signup Page:-**



KPIT

Create a new account

Or sign in to your existing account

First Name

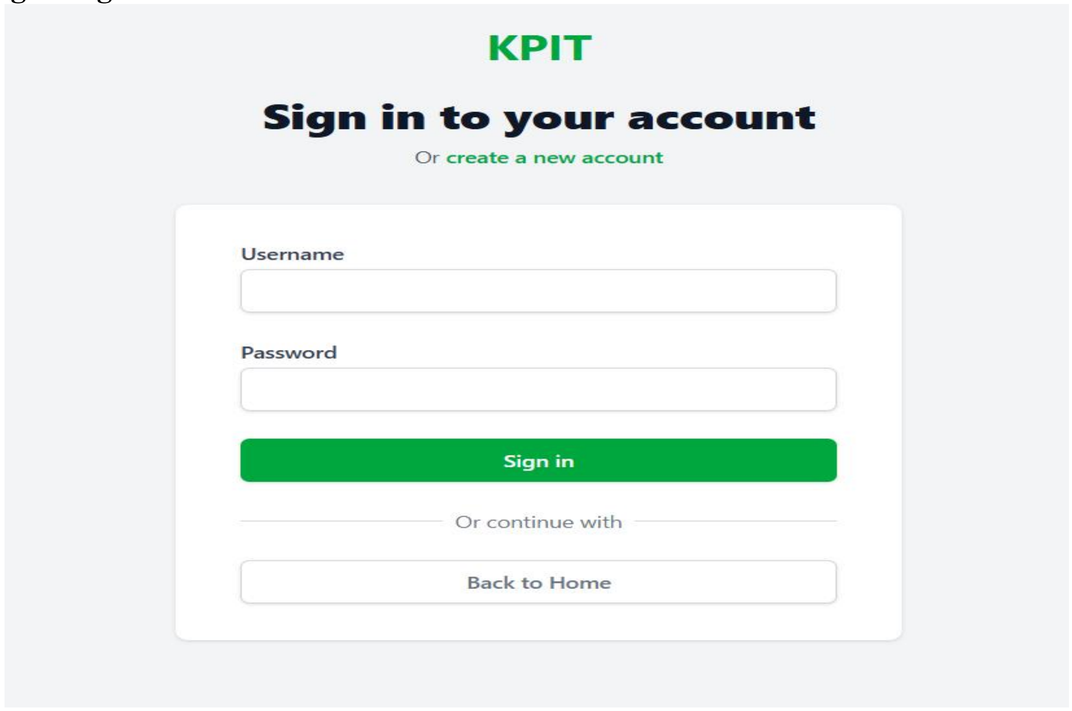
Last Name

Username

Email address

Password

- **Login Page:-**



KPIT

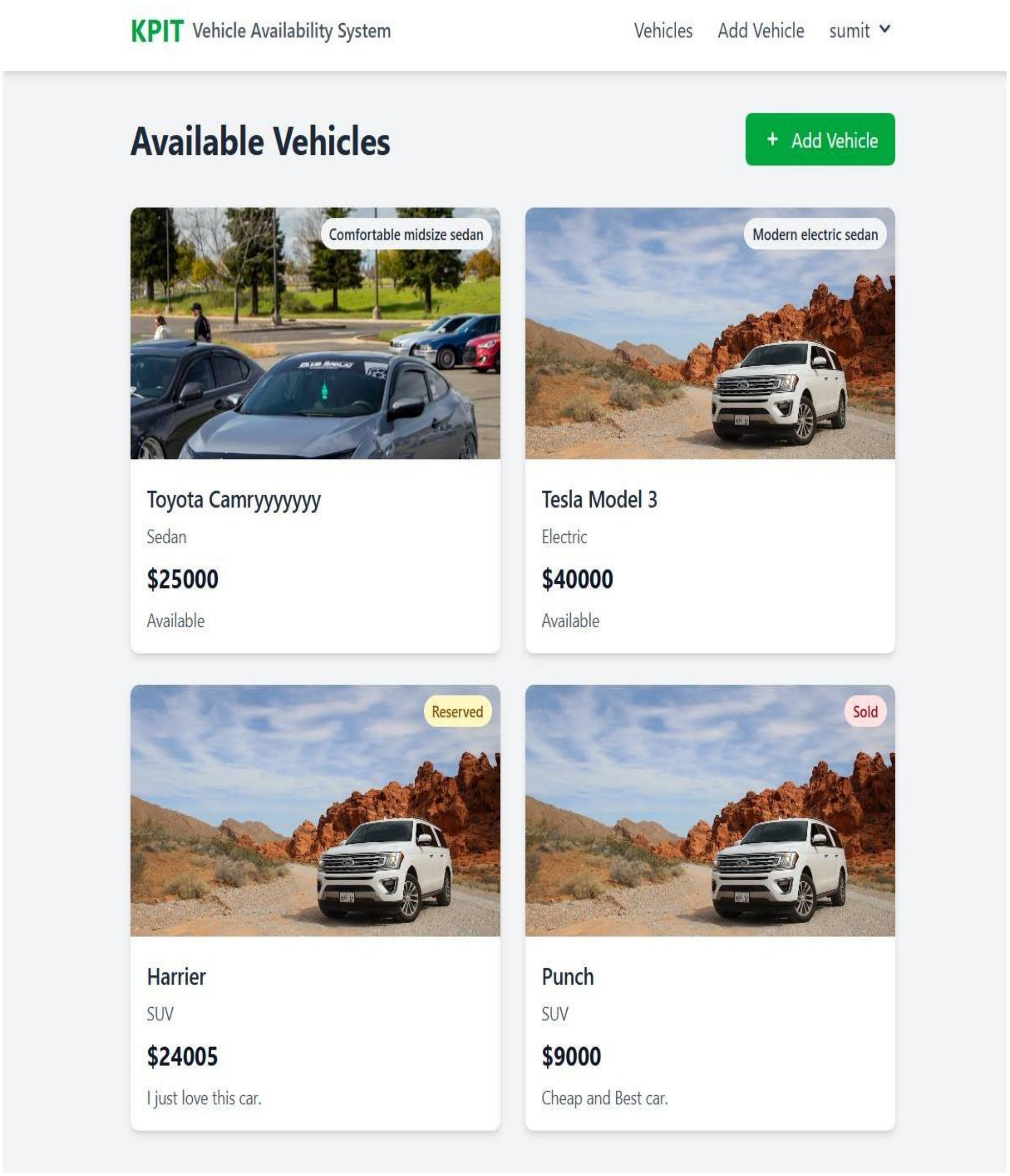
Sign in to your account

Or create a new account

Username

Password

● **Display Page(A page that shows the list of the vehicles)**



Available Vehicles

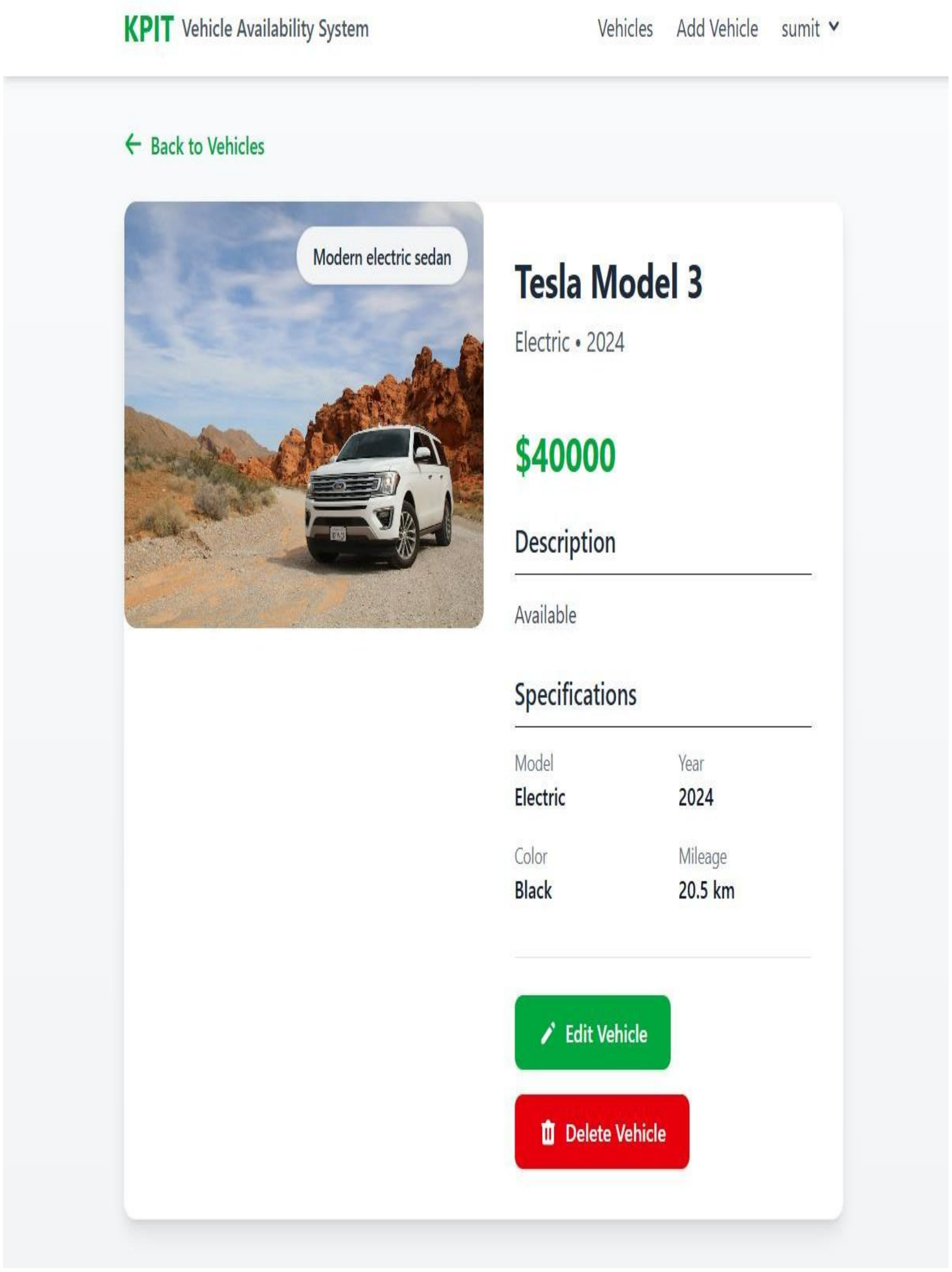
Toyota Camryyyyyyy
Sedan
$25000
Available

Tesla Model 3
Electric
$40000
Available

Harrier
SUV
$24005
I just love this car.

Punch
SUV
$9000
Cheap and Best car.

- **Details Page(A page that shows the details of the page)**



KPIT Vehicle Availability System

Vehicles Add Vehicle sumit

← Back to Vehicles

Modern electric sedan

Tesla Model 3

Electric • 2024

$40000

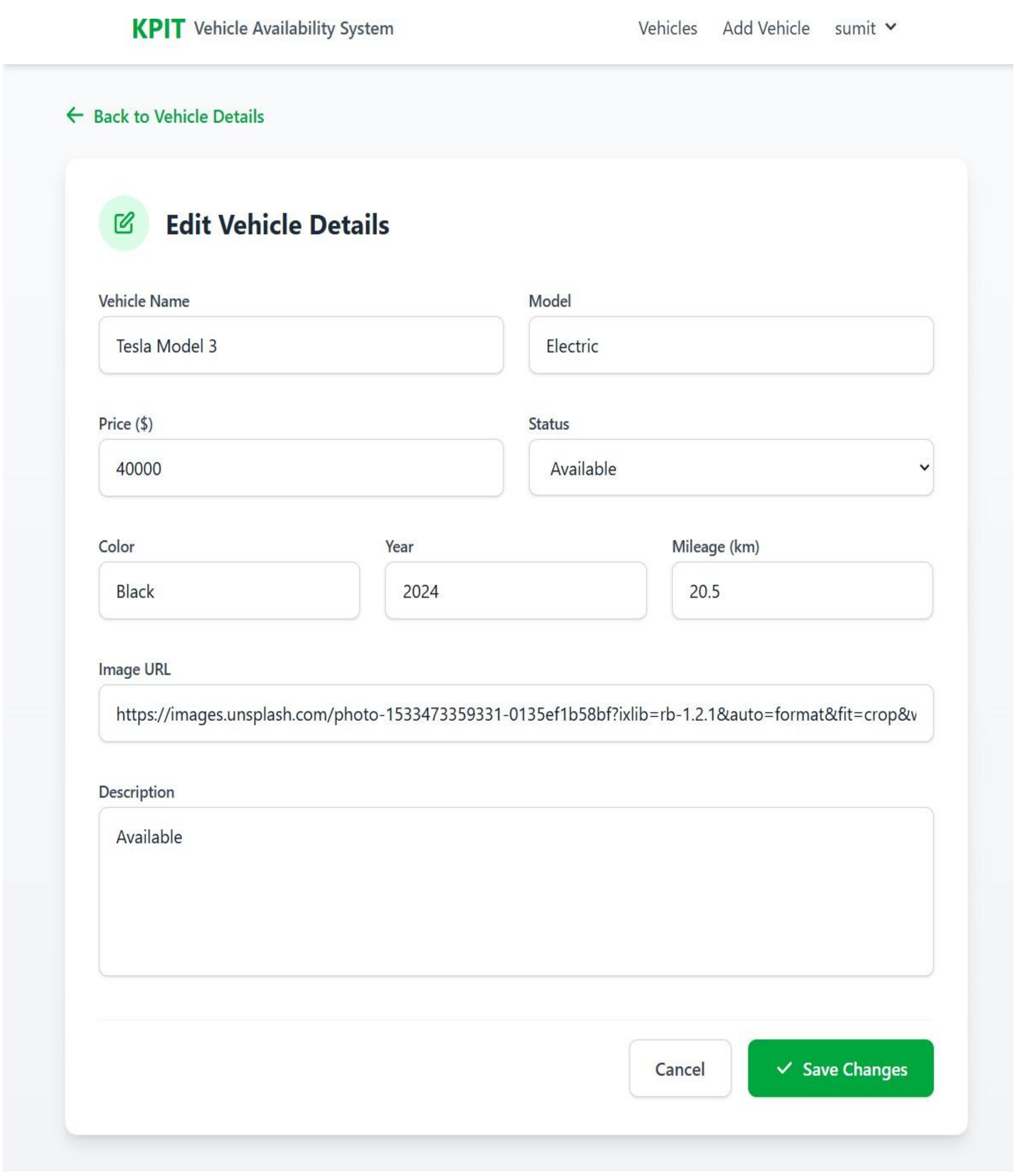
Description

Available

Specifications

Model	Year
Electric	2024
Color	Mileage
Black	20.5 km

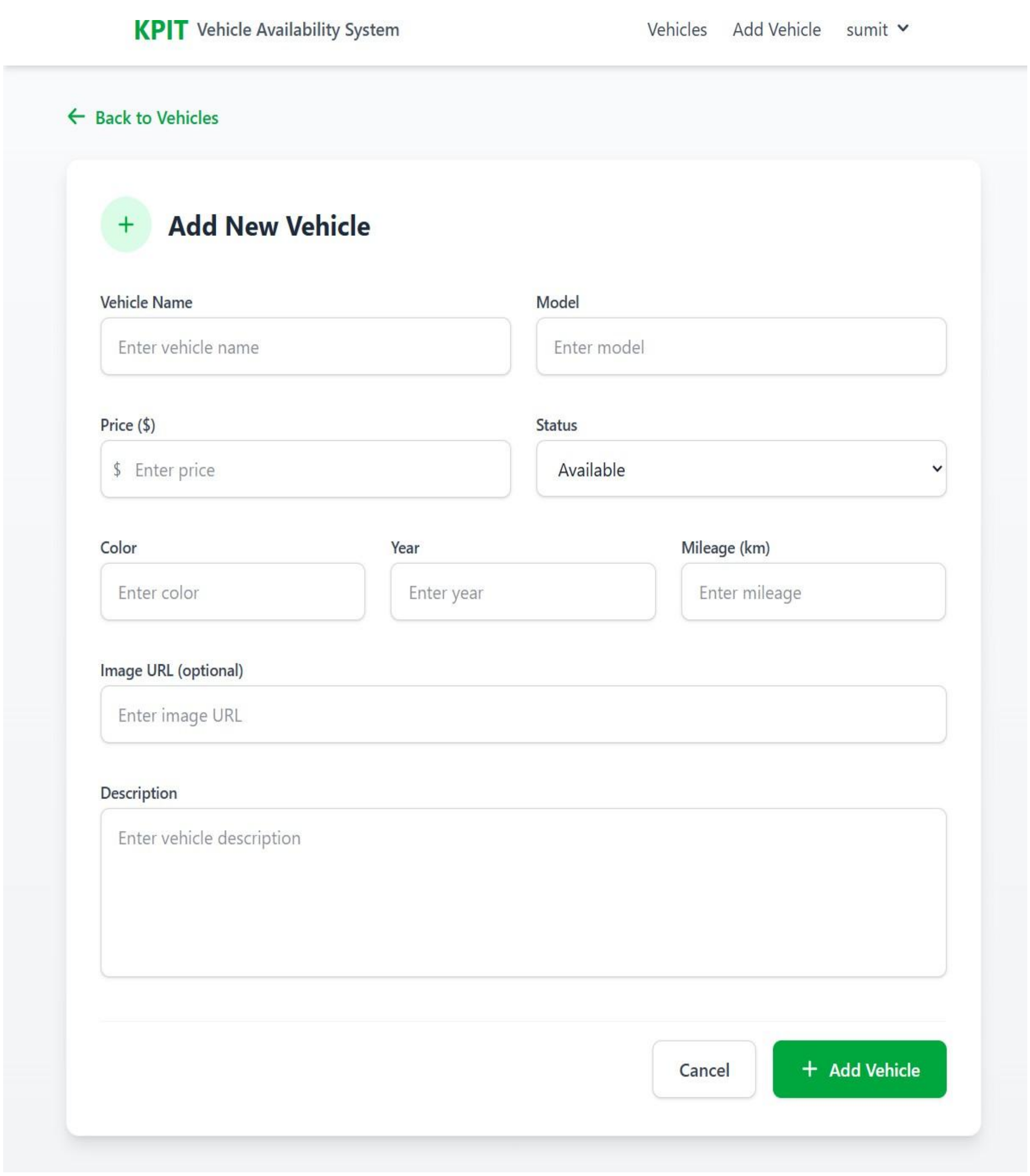
- **Vehicle Edit Page(Only accessible to dealers)**
    - A page which is used to edit the details of the vehicles



Edit Vehicle Details

Vehicle Name	Tesla Model 3
Model	Electric
Price ($)	40000
Status	Available
Color	Black
Year	2024
Mileage (km)	20.5
Image URL	https://images.unsplash.com/photo-1533473359331-0135ef1b58bf?ixlib=rb-1.2.1&auto=format&fit=crop&v
Description	Available

- **Vehicle Add Page(Only accessible to dealers)**
  - A page which is used to add the vehicle by fillling the particular details.


KPIT Vehicle Availability System

Vehicles | Add Vehicle | sumit

← Back to Vehicles

Add New Vehicle

Vehicle Name: Enter vehicle name

Model: Enter model

Price ($): $ Enter price

Status: Available

Color: Enter color

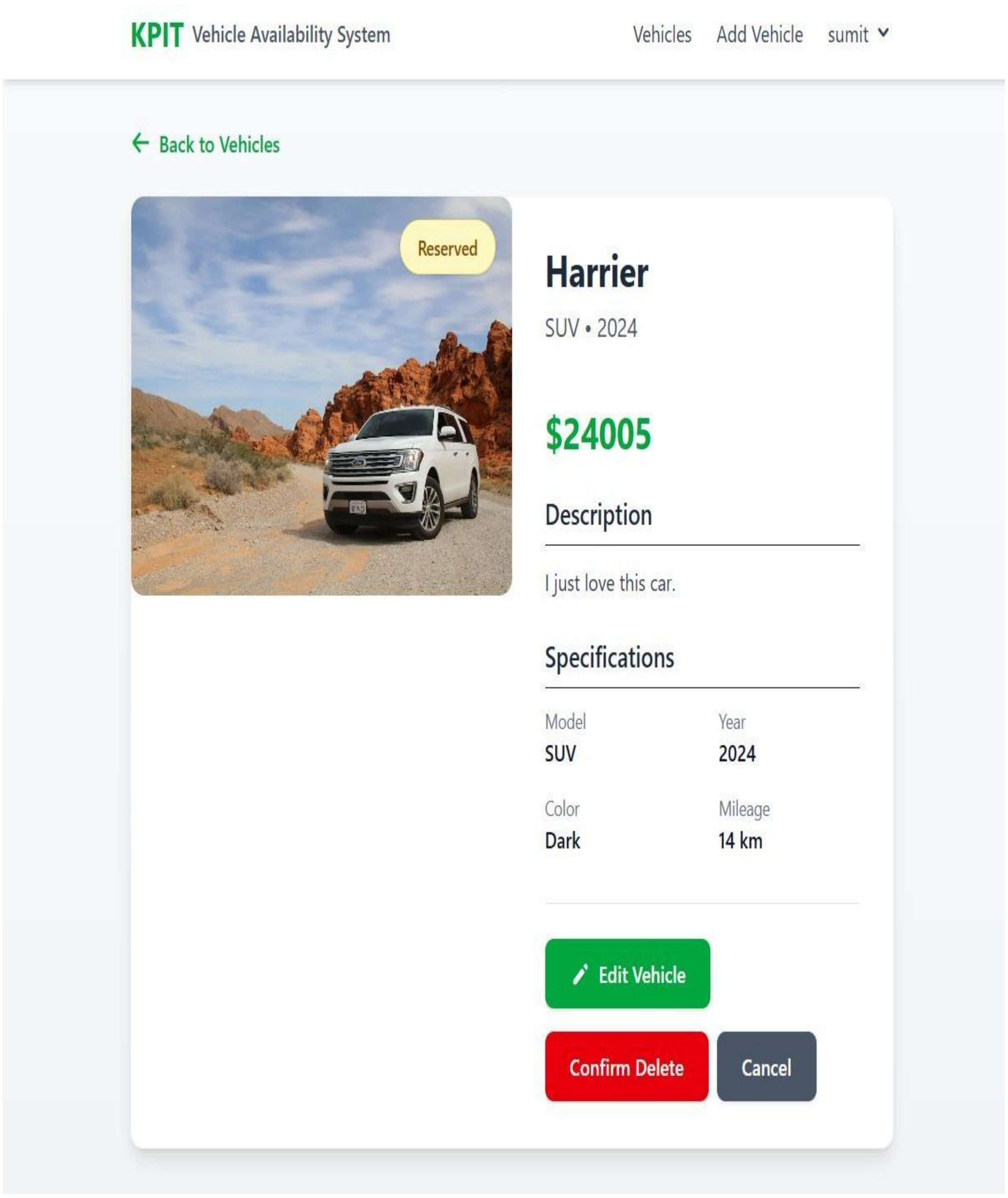
Year: Enter year

Mileage (km): Enter mileage

Image URL (optional): Enter image URL

Description: Enter vehicle description

- **Delete Page(Only accessible to dealers)**
  - ◆ For deleting the particular vehicle there is no any specific page ,simply two buttons will appear there for re-confirming the delete decision.



KPIT Vehicle Availability System

Vehicles Add Vehicle sumit

Back to Vehicles

Reserved

Harrier

SUV • 2024

$24005

Description

I just love this car.

Specifications

Model	Year
SUV	2024
Color	**Mileage**
Dark	14 km

CHAPTER 5
FUTURE SCOPE

The Vehicle Availability System has been developed with a focus on core functionalities such as user authentication, role-based access, and real-time vehicle listing management. While the current version offers a solid foundation, there is significant scope for future enhancements to make the system more comprehensive and practical for real-world use.

One of the most impactful future additions would be the **implementation of a vehicle booking system**. This feature would allow general users to book available vehicles for specific dates and time durations. The booking module would include functionalities such as selecting time slots, automatic vehicle availability updates based on bookings, and confirmation notifications. Dealers could view, manage, or cancel bookings through their dashboard, enabling better planning and resource allocation.

In addition to the booking system, several other improvements can be envisioned:

- **Payment Integration:**Secure online payment gateways could be added for booking confirmations.
- **User Feedback and Ratings:**Allow users to rate vehicles or dealers based on their experience.
- **Advanced Search and Filters:** Help users find vehicles based on location, type, availability dates, or features.
- **Cloud Deployment:**Hosting the application on cloud platforms (e.g., AWS, Azure) would improve scalability and make the system accessible from anywhere.
- **Mobile Application Support**:Developing an Android/iOS version for broader reach and ease of access.

CHAPTER 6
CONCLUSION

The Vehicle Availability System project demonstrates an efficient and organized approach to managing vehicle data using Core Java, Hibernate, and PostgreSQL. Designed with simplicity and functionality in mind, the system allows two types of users—general users and dealers. Dealers have the authority to add new vehicles, edit existing entries, delete records, and update the status of vehicles (such as marking them available or unavailable), ensuring real-time inventory control.

This role-based access control ensures data integrity and security while providing a clear separation of responsibilities. General users can view the list of available vehicles and access specific vehicle details, helping them make informed decisions. The use of HTTP sessions ensures that access is restricted and secure for authenticated users.

The project streamlines vehicle inventory management and serves as a foundational system for businesses or platforms dealing with vehicles. It reduces manual overhead and creates a centralized way to manage availability efficiently.

In the future, this system can be extended by implementing a booking module, integrating user feedback or ratings, enabling advanced search and filter options, or even incorporating cloud deployment for wider scalability. These additions would enhance both functionality and user engagement, making it a more robust and comprehensive solution.

CHAPTER 7 SUMMARY

The project "**Vehicle Availability System**" focuses on developing a user-friendly platform to manage and monitor vehicle availability efficiently. Designed for use by both general users and vehicle dealers, the system leverages Core Java, Hibernate, PostgreSQL, and Maven to deliver a responsive and secure experience. The application includes robust backend integration and role-based access, allowing for organized and controlled interaction with vehicle data.

The system offers a range of features:

- For Users: Ability to view all available vehicles, search by specific criteria, and get detailed information about each vehicle.
- For Dealers: Full control over vehicle inventory including adding new vehicles, editing details, deleting records, and updating availability status (e.g., marking a vehicle as "Available" or "Unavailable").

With structured HTTP session management and clean data flow through MVC architecture, the Vehicle Availability System ensures data security, scalability, and ease of use. The intuitive interface allows dealers to efficiently manage vehicle listings, while users benefit from quick access to accurate information.

Through this platform, the project demonstrates how a well-designed software system can streamline vehicle management operations. It reduces manual workload, improves data accuracy, and enhances the overall experience for both end-users and dealers.

In the future, the system can be extended with advanced functionalities like a vehicle booking module, dealer ratings, location-based filters, and cloud deployment to support broader scalability and real-time analytics.

BIBLIOGRAPHY

The following sources, tools, and references were utilized in the development and documentation of the Vehicle Availability System project:

1. **Java SE Documentation**
   - Official reference for Java language features used in backend development.
   - URL: https://docs.oracle.com/en/java/
2. **Hibernate ORM Documentation**
   - Used for object-relational mapping between Java classes and PostgreSQL database.
   - URL: https://hibernate.org/orm/documentation/
3. **PostgreSQL Documentation**
   - Referenced for database design, query optimization, and stored procedure implementation.
   - URL: https://www.postgresql.org/docs/
4. **Maven Documentation**
   - Used for dependency management and project build automation.
   - URL: https://maven.apache.org/guides/index.html
5. **Spring MVC Concepts (Reference)**
   - Although not implemented with Spring, MVC architectural principles were referred to for controller-service-DAO separation.
   - URL:https://docs.spring.io/springframework/docs/current/reference/html/web.html
6. **HttpSession Management in Java**
   - Used to manage session-based login functionality for secure access control.
   - URL:https://docs.oracle.com/javaee/7/api/javax/servlet/httpHttpSession.html
7. **React.js Documentation**
   - Guidelines for building a responsive and dynamic user interface.

- URL: https://react.dev/

8. **W3Schools and GeeksforGeeks**
   - Referenced for Java, HTML, and frontend/backend integration tutorials.
   - URLs:
     - https://www.w3schools.com/
     - https://www.geeksforgeeks.org/
9. **Additional References**
   - Technical resources and tutorials related to REST API design, JDBC, and role-based access implementation.

APPENDICES

The appendices provide supplementary materials and additional information relevant to the Vehicle Availability System project. This section includes raw data samples, test cases, system diagrams, and configuration details that support a comprehensive understanding of the system

Appendix A: Raw Data Samples

1. **Sample Vehicle Document :**

```
{
  "vehicle_id": "veh123",
  "name": "Toyota Camry",
  "type": "Sedan",
  "price": 25000.0,
  "description": "Comfortable midsize sedan",
  "status": "Available",
  "imageUrl": "https://example.com/camry.jpg",
  "color": "White",
  "mileage": 15.5,
  "modelYear": 2023,
  "dealer_id": "dealer_01"
}
```

2. **User Document:**

```
{
  "user_id": "user_001",
  "name": "Ravi Kumar",
  "email": "ravi@example.com",
  "username": "ravi123",
    "password": "hashed_password",
    "isDealer": true
  }
```

Appendix B: Diagrams

Include the following diagrams as referenced in Chapter 3:

1. **Data Flow Diagrams (Level 0 and Level 1)**
2. **UML Diagrams**:
    - Class Diagram
    - Sequence Diagram
    - Collaboration Diagram

Appendix C: Configuration Details

- Backend: Java, Hibernate, PostgreSQL, Maven, Servlets
- Frontend: React.js
- Tools: Vs Code, Postman, PgAdmin

Appendix D: User Role Summary

- User : View available vehicles, view details
- Dealer: Add, edit, delete vehicles; change status