SUMMARY

Problem Understanding

The two-wheeler industry is one of the fastest-growing sectors, especially in India and other emerging markets. However, buyers often face challenges while exploring and comparing different two-wheelers such as bikes, scooters, and EVs. Information is scattered across multiple websites, making it time-consuming to research specifications, compare models, calculate costs, and manage dealer interactions. Additionally, there is no unified digital platform that connects buyers, sellers, and dealers in a seamless way. This leads to poor customer experience and inefficiency in the purchase cycle.

Proposed Prototype Solution

We propose **Wheelora – Smart Two-Wheeler Marketplace**, a web-based platform designed to centralize all two-wheeler-related needs. The platform allows users to browse, filter, and compare two-wheelers, calculate financial costs such as EMI and fuel expenditure, book test rides, and explore upcoming vehicle launches. It also includes a used bike marketplace where sellers can list verified bikes. Dealers get access to their own dashboards for inventory and customer management. The system integrates advanced AI features, such as a chatbot that recommends models based on user budget and preferences, along with machine learning–based personalized recommendations.

Uniqueness and Impact

What makes *Wheelora* unique is its ability to offer an end-to-end experience that goes beyond a sales listing website. Unlike existing platforms, it integrates new bikes, used bikes, and EVs in one ecosystem, combined with powerful decision-making tools such as EMI and fuel calculators. The inclusion of dealer dashboards ensures a two-sided marketplace that benefits both customers and businesses.

Future-ready AI features provide personalization, making the buyer journey smarter and faster. The impact of *Wheelora* lies in its ability to simplify decision-making for users, improve dealer visibility, and accelerate EV adoption, ultimately creating a unified digital ecosystem for two-wheelers.

Problem Statement Chosen:

Users face challenges in browsing, comparing, and managing the buying process for two-wheelers due to fragmented information and lack of an integrated platform.

Reason to Choose the Problem Statement

Two-wheelers are the most widely used vehicles in India and many other countries. With the rise of EVs, consumers need more clarity when comparing options across fuel types, price ranges, and specifications. Existing platforms are fragmented, focusing primarily on sales without providing financial tools, model comparison, or test ride management. Addressing this problem ensures that millions of potential buyers and thousands of dealers can interact on a single, efficient platform.

Proposed Approach

Our approach is to develop a web application using **React (frontend)**, **Django REST Framework (backend)**, and **PostgreSQL/Supabase (database)**. The frontend is deployed on Vercel, the backend on Render, and the database on Supabase, ensuring scalability and cost efficiency. To optimize performance, we use caching with Redis, database indexing, and lazy loading on the client side. The system is modular, consisting of browsing, comparison, financial tools, test ride management, and dealer dashboards. Advanced features like AI chatbots and ML-based recommendations enhance personalization and user engagement.

Key Features / Modules

1. **Browse & Filter**

   Users can filter vehicles by brand, price, mileage, fuel type, and more to find relevant options quickly.

2. **AI Features**

   Suggests bikes or scooters based on user preferences, budget, past searches, and common usage patterns.

3. **Interactive Product Pages**

   Detailed specs, side-by-side comparisons, 3D/AR views, and community reviews help users make confident decisions.

4. **Finance & Affordability Tools**

   EMI calculator, fuel cost estimator, and resale value predictor ensure transparent decision-making.

5. **Showrooms & Test Rides**

   Dealer dashboards and booking management.

6. **Dealer Dashboard**

Dealers can manage inventory, connect with buyers, and showcase verified listings effectively.

7. **Upcoming Launches**

   Notifications and alerts for upcoming vehicles.

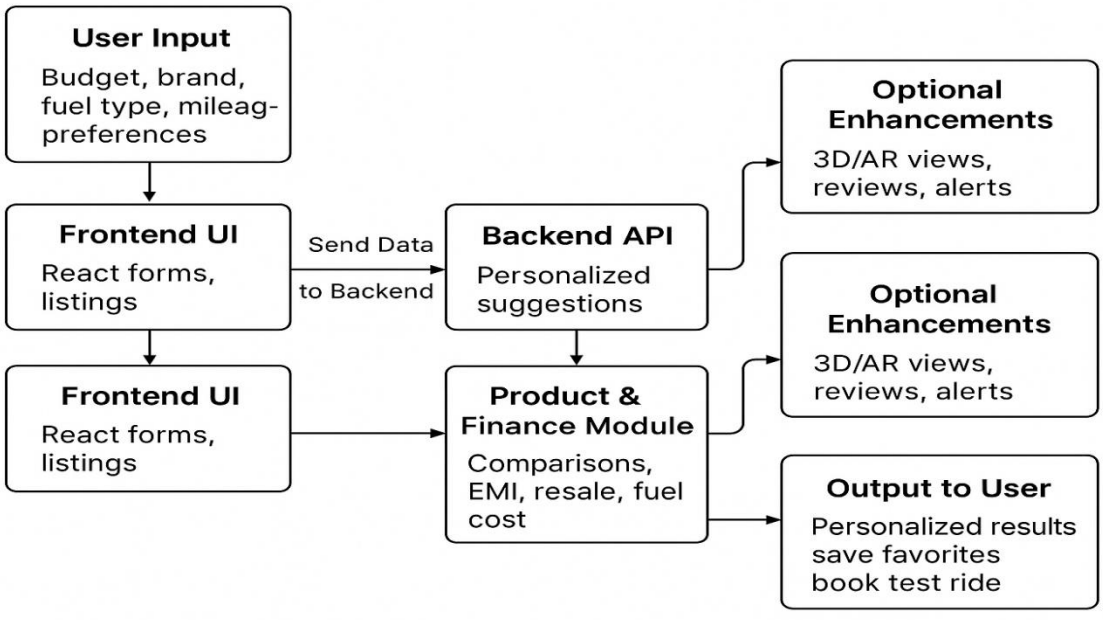
8. **Optional Enhancements**

- Favorites & Alerts: Save preferred vehicles and get price or launch notifications.
- Upcoming Launches: Explore new models before they hit showrooms.
- Booking & Test Rides: Book test rides directly through the platform.

Overview

The system starts when a user opens the app and enters their preferences or budget. The frontend captures the input and sends it to the backend, where the AI recommendation engine analyzes the requirements and generates personalized vehicle suggestions. The system then displays relevant bikes, scooters, or EVs with detailed specs, side-by-side comparisons, and finance tools. If the user wants to book a test ride or contact a dealer, the platform directs them seamlessly, ensuring a smooth and interactive buying journey.

Block Diagram



Step-by-Step Flow

1. **User Input:** The user enters preferences like brand, budget, mileage, or fuel type (or uses filters/search bar).
2. **Frontend (React UI):** The app captures input and shows available bikes/scooters/EVs dynamically.
3. **Send to Backend API:** User choices are sent to the backend (Node.js/Django) for processing.
4. **Database Fetch:** Vehicle details, specs, images, and dealer info are pulled from the database (PostgreSQL/MongoDB).
5. **Comparison & Tools:** Users can compare models side by side, and use EMI/fuel cost calculators.
6. **Dealer & Booking:** Verified dealer dashboard updates inventory; users can book test rides or sell used bikes.
7. **Output to User:** Final personalized bike list, comparisons, calculators, and booking options are shown in a clear UI.

1. Technology Stack

Frontend:

- **React.js** for dynamic, responsive UI.
- **Tailwind CSS** for clean styling.

Backend:

- **Django (Python) / Node.js (Express)** for handling APIs and business logic.
- **REST APIs** for communication between frontend and backend.

Database:

- **PostgreSQL** for structured vehicle listings, dealer data, and user accounts.
- **MongoDB** (optional) for flexible storage like reviews, alerts, and logs.

APIs / Libraries:

- **Google Maps API** for locating nearby showrooms.
- **JWT Authentication** for secure login.

- **Email / SMS Alerts API** for price drop or launch notifications.

Optional Enhancements (Automation Layer):

- **Chatbot Integration** for instant Q&A.

2. Preprocessing Methods

- **Input Normalization**: Clean and standardize user search inputs (brand names, price ranges, fuel types) for accurate filtering.
- **Query Parsing**: Extracts key elements (e.g., "EV bikes under 1L" → *type: EV, price < 1L*).
- **Dynamic Data Handling**: Continuously updates listings with live dealer data and upcoming launches.
- **User Context Memory**: Stores user preferences (budget, favorites, filters) for personalized recommendations.

3. Storage / Pipeline Setup

Databases

- **PostgreSQL**: Stores structured bike listings, dealer inventory, and user accounts.
- **MongoDB** (optional): Stores unstructured data like reviews, price alerts, and activity logs.

Pipeline Flow

1. **User Input → Frontend (React UI)**: User enters filters/search or favorites.
2. **Frontend → Backend (Node.js/Django API)**: Request sent for processing.
3. **Backend → Recommendation Engine**: AI suggests best bikes/models based on preferences.
4. **Backend → Database Fetch**: Specs, dealer info, offers, EMI data retrieved.
5. **Processed Output → Frontend**: User sees listings, comparisons, calculators, and booking options.

Implementation Plan

1. Initial Setup & Environment

- Set up **React frontend** with TailwindCSS.
- Configure **Django/Node.js backend** with REST APIs.
- Connect **PostgreSQL database** for structured data.
- Prepare **basic datasets**: bike models, specs, dealers.

2. Core Module Development

- **Frontend Module**: Dynamic UI with search, filters, comparison, and calculators.
- **Backend Module**: APIs for listings, dealers, EMI, reviews, test ride bookings.
- **Dealer Dashboard**: Inventory management, offers, and buyer leads.

3. Integration & Testing

- Connect frontend ↔ backend ↔ database.
- Test real queries: "Show EV scooters under 80k with 100+ km range."
- Validate EMI calculator, booking system, and comparison module.
- Handle edge cases: no results, invalid input, dealer not verified.

4. Final Deployment-ready Build

- Optimize database queries for speed.
- Ensure frontend responsiveness and mobile support.
- Deploy app on **Render/Vercel**.
- Final testing: end-to-end flow (search → compare → calculate EMI → book test ride).