InstaGO

Technical Design Document

Table of Contents

Introduction

InstaGO is an innovative platform designed to provide a convenient and eco-friendly transportation solution for university students. The system focuses on enabling students to book electric bikes for short-distance travel near the university campus. Whether for shopping, attending events, or exploring nearby areas, InstaGO offers an efficient alternative to traditional transportation methods.

By integrating modern technologies with a user-friendly interface, InstaGO not only promotes sustainable transportation but also simplifies the process of finding and booking electric bikes. The platform is tailored to the unique needs of university students, ensuring affordability, accessibility, and reliability.

Purpose

The purpose of this document is to provide a comprehensive design blueprint for the development of the **InstaGO** platform. It serves as a reference for all stakeholders, including project developers, designers, and evaluators, to ensure a shared understanding of the project's goals, features, and technical specifications.

Scope

The scope of the **InstaGO** project is to develop a comprehensive and user-friendly platform for booking electric bikes, tailored specifically for university students. The platform will address the transportation challenges faced by students by offering an affordable, accessible, and eco-friendly mobility solution. The system will include essential features for users and administrators, ensuring smooth operations and a seamless user experience.

This project also includes a management interface for administrators to oversee operations, such as monitoring bike usage, managing inventory, and analyzing user patterns for future enhancements.

Features

1. User Features:
    - **Registration and Login:** Secure authentication system for students using their university email.
    - **Real-Time Bike Availability:** View the number of bikes available at different stations.
    - **Fare Calculation:** Display estimated charges based on distance or time.
    - **Bike Booking System:** Book bikes for immediate or scheduled use.
    - **Payment Gateway Integration:** Allow online payment for bookings.

2. Admin Features:
    - Manage bike inventory, including adding or removing bikes.
    - Monitor bike availability and booking history.
    - Analyze user activity and generate reports for better decision-making.

Assumptions

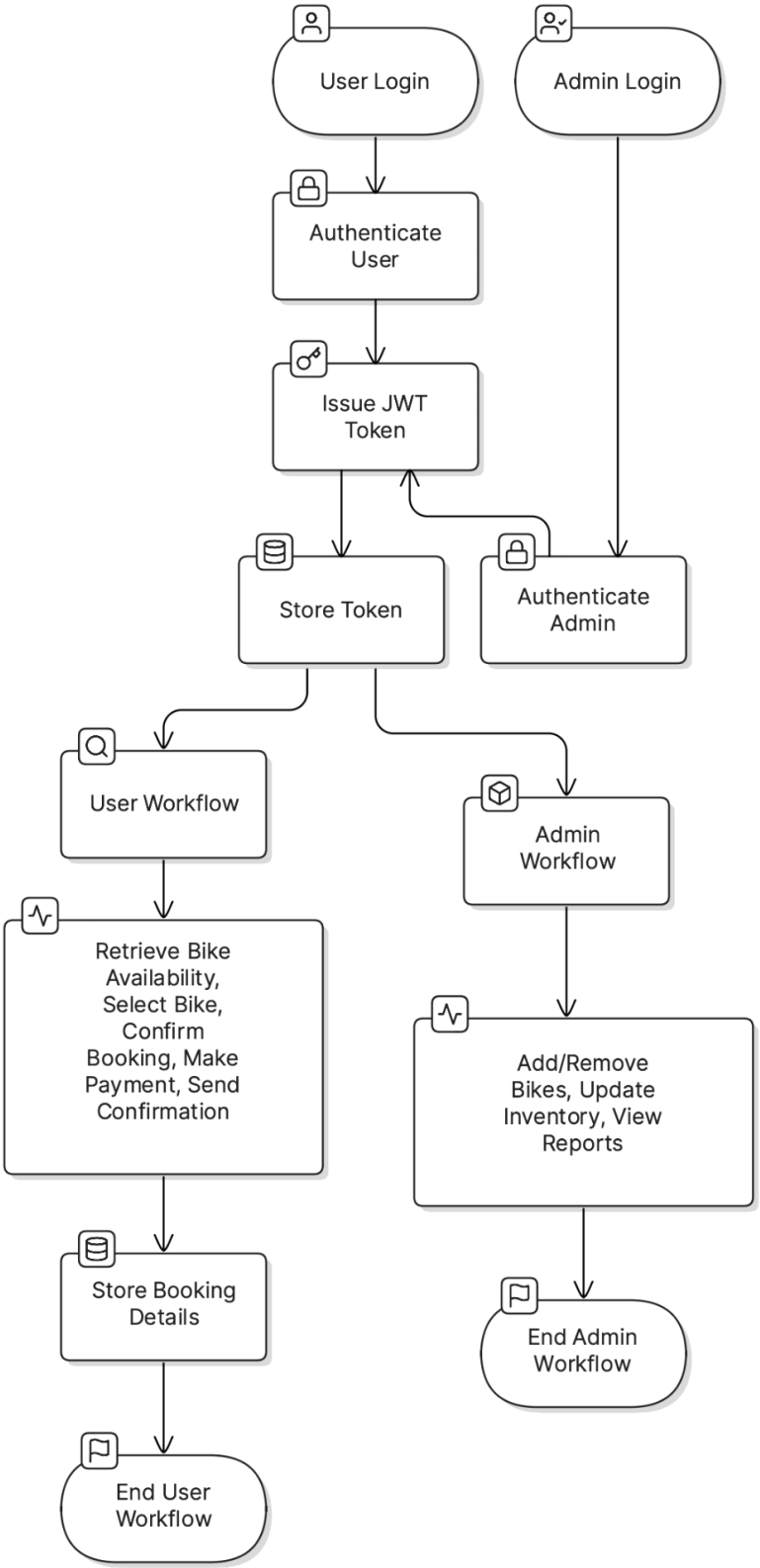
1. The university will provide designated bike parking and charging stations.
2. Internet connectivity will be available for real-time updates and usage of the platform.
3. Users will have smartphones or devices capable of accessing the web application.

System Overview

System Architecture

1. **Frontend:**
    - Built with **React** for dynamic and responsive UI.
    - **react-router-dom** is used for seamless navigation between pages like Login, Dashboard, Booking, and Payment.
    - Axios or Fetch API is used to make secure HTTP requests to the backend.
2. **Backend:**
    - Developed using **Node.js** with **Express.js** as the framework.
    - Provides RESTful APIs for handling user authentication, bike availability, booking, and payments.
    - Implements secure authentication using **JWT (JSON Web Tokens)**.
    - Middleware functions for request validation and error handling.
3. **Database:**
    - Stores user data, bike inventory, and booking records.
    - Database Used: **MongoDB** (NoSQL)
    - Collections/Tables:
        - Users (user_id, name, email, password, role)
        - Bikes (bike_id, station_id, status, last_updated)
        - Bookings (booking_id, user_id, bike_id, start_time, end_time, fare)
4. **Authentication and Authorization:**
    - **JWT Tokens:**
        - Issued upon successful login to authenticate users for API requests.
        - Encodes user information, ensuring secure and stateless authentication.
    - **Role-Based Access Control:**
        - Users (students) can book bikes and view bookings.
        - Admins can manage inventory and view reports.
5. **Payment Gateway Integration:**
    - Integrates with Razorpay for online payments.
    - Backend processes payment requests and validates transactions.

InstaGO System Workflow



System Design

Frontend Design

The front-end is a **Single Page Application (SPA)** built with React.js. It dynamically updates the view without requiring a full page reload, ensuring a fast and smooth user experience. The design follows a **component-based architecture**, where each functionality is encapsulated in reusable components.

Key Pages and Components

1. **Login Page**
    - **Purpose:** Allow users to log in using their credentials.
    - **Design:**
        - Input fields for email and password.
        - A "Login" button that triggers an API call to the backend for authentication.
        - Error messages for incorrect credentials.
    - **Key Features:**
        - Integration with JWT for secure token-based login.
        - Redirects authenticated users to the dashboard.
2. **Registration Page**
    - **Purpose:** Allow new users to register for the platform.
    - **Design:**
        - Input fields for name, email, password, and confirm password.
        - Validation for input fields (e.g., password strength, email format).
    - **Key Features:**
        - Sends a POST request to the backend to create a new user.
3. **Dashboard Page**
    - **Purpose:** Serve as the homepage after login, showing bike availability and quick navigation to other features.
    - **Design:**
        - Real-time bike availability displayed as cards or a table.
        - Fare and charges displayed dynamically.
        - Quick access buttons for "Book a Bike" and "View Bookings."
    - **Key Features:**
        - Fetches data from the backend to display bike availability.
4. **Bike Booking Page**
    - **Purpose:** Allow users to search and book bikes.

- **Design:**
  - Dropdown or map view to select bike stations.
  - Input fields for booking time or duration.
  - "Book Now" button to confirm the booking.
- **Key Features:**
  - Calculates fare dynamically based on user input.
  - Makes an API call to reserve the selected bike.

5. **Payment Page**
   - **Purpose:** Process payments for booked bikes.
   - **Design:**
     - Displays booking details (bike, fare, time).
     - Integration with a payment gateway (Razorpay).
   - **Key Features:**
     - Backend validates and processes the payment.
     - Redirects to confirmation page on successful payment.
6. **Booking History Page**
   - **Purpose:** Display a list of previous and current bookings.
   - **Design:**
     - Table format with columns for booking date, time, bike ID, and fare.
     - Status indicator for active/completed bookings.
   - **Key Features:**
     - Fetches user booking history from the backend.
7. **Admin Page (For Admin Users)**
   - **Purpose:** Allow admins to manage bikes and view system statistics.
   - **Design:**
     - Options to add/remove bikes, update inventory, and view reports.
   - **Key Features:**
     - Restricted access with role-based authorization.
     - Integration with backend APIs for bike management.

Backend Design

The backend architecture follows a **Model-View-Controller (MVC)** design pattern for clear separation of concerns. It is composed of different layers that handle requests, business logic, and database operations. The backend will be structured as a **RESTful API**, which interacts with the frontend via HTTP requests (GET, POST, PUT, DELETE) and responds with JSON data.

Core Components of the Backend Architecture

1. **API Layer (Express.js)**
    - **Express.js:** The core framework for handling routing, middleware, and HTTP request/response management.
    - **RESTful Endpoints:** Define various API routes for user authentication, bike management, booking functionality, and payments.
2. **Authentication and Authorization (JWT)**
    - **JWT Authentication:** Secure token-based authentication mechanism that ensures that only authenticated users can access certain resources.
    - **Role-Based Access Control (RBAC):** Different roles (admin, user) will have specific permissions. Admins can manage the system, while users can book bikes and view their history.
    - **JWT Middleware:** A middleware that verifies JWT tokens for protected routes.
3. **Business Logic Layer (Controllers)**
    - Controllers handle the core business logic and communicate with the database to perform CRUD operations. Each endpoint in Express will route to a specific controller.
4. **Database Layer**
    - **MongoDB:** A NoSQL or relational database to store user, bike, booking, and payment information.
    - **Mongoose:** A library for modeling and querying data in MongoDB.
5. **Payment Gateway Integration**
    - **Payment APIs:** Use third-party payment services like (**Razorpay**) for processing payments.
    - API calls are made to these services to create transactions and verify payments.

API Endpoints

1. **User Registration**
    - **POST /register:**
        - Accepts user data (name, email, password).
        - Password is hashed using **bcrypt** before storing it in the database.
        - Send a response with user details (excluding password).
2. **User Authentication**
    - **POST /login:**
        - Accepts login credentials (email, password).
        - Validates credentials by comparing hashed passwords in the database.
        - If successful, generates a **JWT token** that the user can use to access protected routes.
        - JWT token is sent as a response.
3. **Bike Availability (User)**
    - **GET /bikes:**
        - Fetches bike availability data from the database (or real-time tracking system).
        - Sends a JSON response with bike details (status, location, availability).
4. **Booking a Bike**
    - **POST /book:**
        - Accepts booking details (bike ID, start time, duration).
        - Checks bike availability in the database.
        - Calculates the fare based on time/distance.
        - Books the bike by updating the bike's status in the database.
        - Creates a booking record in the database.
        - Returns the booking details with a success message.
5. **Payment Processing**
    - **POST /payment:**
        - Accepts payment data (amount, user ID, payment method).
        - Integrates with payment gateway (Razorpay) to create a transaction.
        - Verifies the transaction with the payment service.
        - Updates booking status to "paid" upon successful payment.
6. **Booking History (User)**
    - **GET /history:**
        - Fetches and returns all bookings made by the authenticated user.
        - Only accessible with a valid JWT token.
7. **Admin Routes (Bike Management)**
    - **POST /admin/add-bike:**
        - Allows admin to add new bikes to the system.
    - **PUT /admin/update-bike:**
        - Admin can update bike details (availability, location).

Database Schema

Here's a detailed **Database Schema** for your **InstaGO** project, which covers the core entities such as **Users**, **Bikes**, **Bookings**, and **Payments**.

1. User Schema

The `User` schema stores user-related information, including login credentials, role, and booking history. This schema will be used for authentication and authorization purposes.

Fields:

- **_id** (Primary Key): A unique identifier for the user.
- **name**: The full name of the user.
- **email**: The email address used for login (unique).
- **password**: The hashed password for authentication.
- **role**: Role of the user (`user`, `admin`).
- **createdAt**: Date when the user account was created.
- **updatedAt**: Date when the user profile was last updated.

---

2. Booking Schema

The `Booking` schema stores information about bike bookings made by users. It contains the user, bike, booking details (start/end time), fare, and booking status.

Fields:

- **_id** (Primary Key): A unique identifier for the booking.
- **user_id**: Reference to the `User` who made the booking (Foreign Key).
- **bike_id**: Reference to the `Bike` that is being booked (Foreign Key).
- **start_time**: The timestamp when the booking starts.
- **end_time**: The timestamp when the booking ends.
- **fare**: The total fare for the booking.
- **status**: The booking status (`pending`, `completed`, `canceled`).
- **createdAt**: Date when the booking was created.
- **updatedAt**: Date when the booking details were last updated.

3. Payment Schema

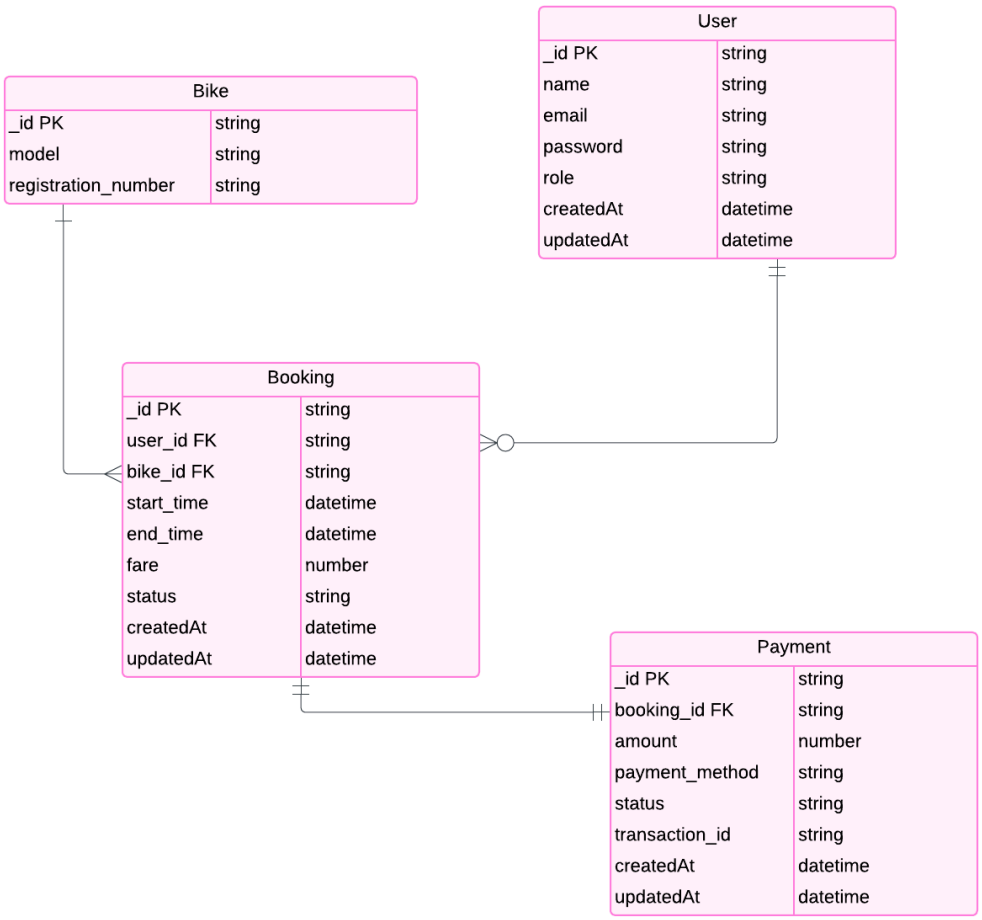
The `Payment` schema stores payment details for each bike booking. It includes information about the payment method, amount, and status.

Fields:

- **_id** (Primary Key): A unique identifier for the payment.
- **booking_id**: Reference to the `Booking` for which the payment was made (Foreign Key).
- **amount**: The total payment amount for the booking.
- **payment_method**: The payment method used (e.g., `credit_card`, `UPI`, `wallet`).
- **status**: The payment status (e.g., `pending`, `successful`, `failed`).
- **transaction_id**: A unique identifier for the payment transaction (provided by the payment gateway).
- **createdAt**: Date when the payment was made.
- **updatedAt**: Date when the payment status was last updated.

Database Relationships

1. **User to Booking**: One-to-many relationship (a user can have multiple bookings).
2. **Bike to Booking**: One-to-many relationship (a bike can be booked multiple times).
3. **Booking to Payment**: One-to-one relationship (each booking can have one payment).



Database Schema Diagram

Deployment

Frontend:

Vercel is used for deploying the React.js frontend. Vercel offers automatic builds, continuous deployment from GitHub, and serverless architecture, making it an easy choice for frontend deployment.

Backend:

AWS EC2 is used to deploy the Node.js backend. You can set up an EC2 instance and configure it to serve the Express.js API. Optionally, use Nginx as a reverse proxy or Docker for containerization.

Database:

MongoDB Atlas is used to manage and deploy the MongoDB database. It offers a cloud-based, fully managed service for MongoDB, which is scalable and includes features like backup, monitoring, and automatic scaling.

Future Enhancement

1. Multi-Campus Support

Expand the platform to support multiple campuses or cities, enabling students from different locations to use InstaGO for electric bike bookings.

2. Real-Time GPS Tracking for Bikes

Integrate real-time GPS tracking for electric bikes, allowing users to see the exact location of available bikes on the map.

3. Bike Health Monitoring

Incorporate IoT sensors on the bikes to monitor their health, battery status, and maintenance needs. This data can be used to notify users if a bike is unavailable due to maintenance or low battery.

4. Enhanced User Profiles and Preferences

Allow users to create detailed profiles where they can save their preferences (e.g., preferred bike type, payment method, ride history).

5. Integration with Campus Services

Integrate InstaGO with other campus services like library bookings, cafeteria orders, or event notifications. For instance, users could book a bike for campus events or trips directly through the app.

Conclusion

In conclusion, InstaGO offers a comprehensive solution for university students to book electric bikes for short trips around the campus and nearby locations, providing convenience, sustainability, and security in a user-friendly environment. The use of modern technologies like React, Node.js, MongoDB, and cloud-based deployments on Vercel and AWS ensures that the platform is reliable, scalable, and capable of handling future growth.