Project Proposal: PG/Flat Booking Platform

Student Name: Himanshu Gupta

Student UID: 23BCS10889

Section: 23BCS_KRG_2-B

Teacher: Prof. Juned Alam (E18479)

1. Problem Statement

Finding suitable paying guest (PG) accommodations or rental flats can be a significant challenge for students and working professionals. The current process often involves:

- **Lack of Centralized Information:** Dispersed listings across various websites, local brokers, or word-of-mouth make it difficult to compare options efficiently.
- **Time-Consuming Search:** Usethers spend considerable time visiting multiple locations, often to find accommodations that don't meet their criteria.
- **Verification Issues:** Difficulty in verifying the authenticity of listings, amenities, and landlord credibility.
- **Inefficient Communication:** Challenges in direct and timely communication between prospective tenants and property owners.
- **Limited Transparency:** Lack of clear pricing, facility details, and review mechanisms.

Similarly, property owners face difficulties in reaching a wide audience of potential tenants, managing inquiries, and showcasing their properties effectively.

2. High-Level Design

The PG/Flat Booking Platform will be a web-based application designed to connect prospective tenants with property owners. It will follow a typical client-server architecture.

- **User Interface (Frontend):** This will be the interactive part of the application that users (tenants and owners) interact with. It will handle displaying listings, user input, and navigation.
- **Backend API (Server):** This will be the core logic of the application, handling data processing, business rules, user authentication, and communication with the database. It will expose RESTful APIs for the frontend.
- **Database:** This will store all the application data, including user profiles, property listings, bookings, reviews, and messages.

3. Solution

Our proposed PG/Flat Booking Platform aims to solve the identified problems by providing a comprehensive, user-friendly, and transparent solution:

- **Centralized Listing Platform:** A single platform where property owners can list their PGs/flats with detailed information, and tenants can browse and filter options.
- **Advanced Search & Filtering:** Users can search based on location, price range, amenities (e.g., Wi-Fi, AC, food), room type, and gender preference.
- **Detailed Property Profiles:** Each listing will include high-quality images, detailed descriptions, amenity lists, pricing structure, and house rules.
- **User Authentication & Profiles:** Secure login for both tenants and owners, allowing them to manage their profiles, listings, and inquiries.
- **Direct Communication:** An in-app messaging system to facilitate direct communication between interested tenants and property owners.
- **Reviews and Ratings:** A system for tenants to leave reviews and ratings for properties, enhancing transparency and trust.
- **Booking/Inquiry Management:** Functionality for tenants to send booking inquiries and for owners to manage these inquiries (accept/reject).

4. Low-Level Design

Technologies:

The project will follow a **MERN stack** architecture, leveraging its full-stack JavaScript capabilities for seamless development.

- **Frontend:**
  - **React.js:** For building a dynamic and responsive Single Page Application (SPA) user interface.
  - **Tailwind CSS:** For efficient and utility-first styling, ensuring a modern and responsive design.
  - **JavaScript (ES6+):** For frontend logic and interactivity.
- **Backend:**
  - **Node.js with Express.js:** For building a robust, scalable, and RESTful API.
  - **MongoDB (NoSQL Database):** For flexible data storage, especially suitable for varied property details and user profiles.
  - **JWT (JSON Web Tokens):** For secure, stateless user authentication and authorization for all REST API endpoints.
  - **OAuth 2.0:** For enabling third-party login options (e.g., Google, Facebook), enhancing user convenience and security.

- **Deployment (Consideration):** Heroku, Vercel, or similar cloud platforms for easy deployment of both frontend and backend components.

Database Schema (Simplified - MongoDB Collections):

- **users Collection:**
  - _id (ObjectId)
  - email (String, unique)
  - password (String, hashed - *optional if only OAuth is used*)
  - googleId (String, *for OAuth*)
  - facebookId (String, *for OAuth*)
  - role (String: "tenant", "owner")
  - name (String)
  - phone (String)
  - profilePicture (String, URL)
  - createdAt (Date)
  - updatedAt (Date)
- **properties Collection:**
  - _id (ObjectId)
  - ownerId (ObjectId, ref to users collection)
  - title (String)
  - description (String)
  - address (Object: street, city, state, pincode, landmark)
  - type (String: "PG", "Flat")
  - rent (Number)
  - deposit (Number)
  - amenities (Array of Strings: "Wi-Fi", "AC", "Food", "Laundry", etc.)
  - images (Array of Strings, URLs)
  - genderPreference (String: "Male", "Female", "Unisex")
  - occupancyType (String: "Single", "Double", "Triple", etc.)
  - isAvailable (Boolean, default: true)
  - createdAt (Date)
  - updatedAt (Date)
- **bookings (or inquiries) Collection:**
  - _id (ObjectId)
  - tenantId (ObjectId, ref to users collection)
  - propertyId (ObjectId, ref to properties collection)
  - inquiryDate (Date)
  - checkInDate (Date, proposed)
  - status (String: "Pending", "Accepted", "Rejected", "Completed")

    - message (String, optional)
    - createdAt (Date)
    - updatedAt (Date)
- **reviews Collection:**
    - _id (ObjectId)
    - tenantId (ObjectId, ref to users collection)
    - propertyId (ObjectId, ref to properties collection)
    - rating (Number, 1-5)
    - comment (String)
    - createdAt (Date)

Key Functionalities (Frontend & Backend Interaction):

1. **User Registration & Login (with JWT & OAuth):**
    - **Email/Password:** Frontend sends email/password to backend. Backend hashes password, saves user, generates JWT, sends token to frontend. Frontend stores JWT for subsequent authenticated requests.
    - **OAuth (e.g., Google Login):** Frontend initiates OAuth flow (e.g., redirects to Google login). After successful authentication with the third-party provider, the provider sends a token/code back to the frontend. Frontend sends this to the backend. Backend verifies the token with the OAuth provider, retrieves user details, creates/updates user in DB, generates its own JWT, and sends it to the frontend.
2. **Property Listing (Owner):**
    - Owner fills a form with property details and images.
    - Frontend sends data to backend API (/api/properties) with the JWT in the authorization header.
    - Backend validates data and JWT, stores in properties collection, associates with ownerId.
3. **Property Search & Filter (Tenant):**
    - Tenant uses a search bar and filters on the front end.
    - Frontend sends query parameters to backend API (/api/properties?location=...&rent_max=...).
    - Backend queries properties collection, returns filtered results.
4. **Property Details View:**
    - Tenant clicks on a property.
    - Frontend requests specific property details from backend (/api/properties/:id).
    - Backend retrieves property and associated owner/review data.
5. **Booking Inquiry:**
    - Tenant sends an inquiry from the property details page.

    - Frontend sends inquiry details to the backend API (/api/bookings) with the JWT.
    - Backend validates JWT, creates a new entry in bookings collection.
6. **Messaging System:**
    - Frontend provides a chat interface.
    - Backend API handles message storage and retrieval (could use WebSockets for real-time, or simple polling for MVP), ensuring messages are authorized via JWT.
7. **Review Submission:**
    - Tenant submits a review after a booking is completed.
    - Frontend sends review data to backend API (/api/reviews) with the JWT.
    - Backend validates JWT, saves review, updates property's average rating.