React Native Technical Task – Real Estate App (Test Project)

Objective:
Build a simple **Real Estate mobile app** using **React Native** with a **Node.js backend**.
The app should demonstrate your ability to work with UI, maps, and basic backend integration.

Frontend (React Native):

- Create a simple **UI** with the following sections:
    1. **Projects**
    2. **Units**
    3. **Developers**
    4. **Zones**
- Each **Unit** should display:
    - Unit name / image / price
    - Developer name
    - Zone name
    - **Map view** showing the **unit's location (latitude & longitude)** using any map library (e.g., Google Maps API, react-native-maps).
- Add **Login & Signup screens** that connect with the backend (basic authentication).

Backend (Node.js + Express):

- Create a simple backend API with the following features:
    - **User Authentication** (Login / Signup using JWT)
    - **CRUD endpoints** for:
        - Projects
        - Units
        - Developers
        - Zones
- Each **Unit** record should include a **location field** (latitude, longitude).

Bonus (optional):

- Simple **search or filter** by zone or developer.
- Store images using static folder or links.

Submission:

- Share:
    1. The **GitHub repository** (with both frontend & backend folders).
    2. Short **README file** explaining how to run both apps.
    3. Screenshots or short video (optional).

Advanced Backend Requirements (Mandatory)

1. Advanced Geographic Search (Geo-Fencing/Proximity Search)

Requirements:

- Implement a dedicated API endpoint for searching real estate units within a specified radius (R kilometers) from the user's current location
- **Input Parameters:**
    - userLat (User's latitude - required)
    - userLng (User's longitude - required)
    - radius (Search radius in kilometers - optional, default: 10km)

Technical Implementation:

1. **Database Optimization:**
    - Create a **2dsphere index** on the location field in your MongoDB Unit collection
    - Use MongoDB's native geospatial queries ($geoWithin with $centerSphere) for maximum performance

2. **Performance Requirements:**
    - The search must execute in **under 100ms** even with millions of property records
    - Implement proper query optimization and indexing strategies

3. **API Response:**
    - Return units sorted by proximity (nearest first)
    - Include calculated distance for each property in the response
    - Support pagination for large result sets