Backend Assignment

Objective:

Develop a scalable backend system to track the live location (GPS) of users in real-time. The system should handle location pings sent every 4 seconds and include endpoints for optional user registration, login, and an admin interface to monitor and manage user data.

Core Requirements:

1. **User Features:**
    a. **Registration & Login**: Users should be able to register and log in.
    b. **Location Tracking**: After logging in, track the user's GPS location and send a ping to the server every 4 seconds.
    c. **Scalability**: The backend must handle at least **500 live users** tracking their locations.
2. **Admin Features:**
    a. **User Monitoring**: Provide an admin interface to view all registered users.
    b. **Location Logs**: Allow the admin to view detailed location logs for individual users.
3. **Technical Stack:**
    a. **Backend**: Use **Node.js** with any modern framework (e.g., **NestJS**, **Express**).
    b. **Databases**: Use either only SQL, NoSQL or a combination of both. For e.g.-
        i. Relational Database (**PostgreSQL**) for structured data (e.g., user profiles).
        ii. NoSQL Database (**MongoDB**) for flexible storage of location data.
    c. **Optimization**: Consider using tools like **Redis** or similar for caching and scaling, feel free to use any other technology for optimizations.
    d. **Frontend (Minimal UI)**: Build a basic frontend using a JS/TS-based framework (if possible or Postman client would also work)

Important Notes:

- This assignment is for a backend-focused role. A minimal UI is acceptable.
- Emphasis should be on backend design, scalability, and efficient data handling.
- **No penalties for not completing bonus tasks**. They are entirely optional and meant for showcasing additional skills if you wish.

Bonus (Optional):

1. **Redis Integration**:
   a. Use Redis for caching frequently queried data or optimizing real-time data handling.
2. **Relational Database Optimization**:
   a. Implement advanced features of relational databases (e.g., indexing, partitions) for better performance.
3. **Code Architecture**:
   a. Use clean, modular, and scalable code architecture with clear separation of concerns.
4. **Enhanced Admin Panel**:
   a. Build a more advanced admin panel with filters, user search, and exportable logs.
5. **React JS App**:
   a. Create a web app for users to register, log in, and share their location. (Minimal frontend would be suffice)

Evaluation Criteria:

- **Core Features**: Completion of the user and admin functionality.
- **Scalability**: Ability to handle >=500 live/concurrent users.
- **Database Design**: Efficient use of relational and/or NoSQL databases.
- **Code Quality**: Clean, maintainable, and well-documented code.
- **Creativity**: Any unique approach or optimizations beyond the basic requirements.