Take-Home Assignment — Full Stack (Developer (8 hours

Objective

Build a **Location Manager** application where users can add, edit, view, and delete “locations of interest” on an interactive map, enriched with address/coordinate lookups from an external .API

Requirements

(Backend (Nest.js + MongoDB

- .Implement REST API endpoints for managing `Location` entities
-

```ts
Location {
  id: string
  name: string
  category: "office" | "store" | "landmark"
  coordinates: {
    lon: number
    lat: number
  }
  address?: string
  notes?: string
  createdAt: Date
  updatedAt: Date
}
```

- :Features
  - .List with pagination, search, and category filter
  - .Detail, create, update, delete
  - .Input validation with Zod
  - .Use MongoDB indexes
  - Error handling
- **:(External API integration (backend**
  - :Use **OpenStreetMap Nominatim** (or similar) to resolve
    - .Coordinates → Address
    - .Address → Coordinates
  - Cache results in memory for 10–30 min.
    .Handle errors, timeouts, and rate limits gracefully
- :Tests
  - .Unit tests for service logic
  - .Controller tests for at least one happy path and one validation error

---

(Frontend (React + TypeScript

- :Map & List view
  - .Show all locations on an OpenLayers map with markers
  - .Clicking a marker highlights the list item and vice versa
- :Form for create/edit
  - .Built with React Hook Form + Zod resolver
  - .Set coordinates by clicking/dragging a marker **or** entering manually
  - .Validate inputs and show clear error messages

(Form Fields (Create/Edit Location

- **Name**
  - Required
  - Text field
  - Minimum length: 2 characters
  - Maximum length: 60 characters
- **Category**
  - Required
  - Select field
  - Options: `office`, `store`, `landmark`
- **Coordinates**
  - Required
  - Two numeric fields: `longitude`, `latitude`
  - Longitude range: -180 to 180
  - Latitude range: -90 to 90
- **Address**
  - Optional
  - Text field
  - Maximum length: 120 characters
- **Notes**
  - Optional
  - Multiline text field
  - Maximum length: 500 characters

- Data layer:
  - Use React Query for all fetches and mutations.
  - Implement optimistic update for create/delete.
  - Show loading and error states with retries.
- UI:
  - Use MUI components.
  - Responsive layout (desktop + mobile).

Bonus (optional): Real-time updates without WebSockets

Goal

Keep all clients in sync when locations are created/updated/deleted — **without** WebSockets.

Constraints

- Do **not** use WebSockets or socket libraries.
- Use plain HTTP approaches: **Server-Sent Events (SSE)** or **HTTP caching + smart polling**.
- Must gracefully degrade if the stream/polling is unavailable.

Non-Functional Requirements

- Code must be typed (TypeScript) and linted (ESLint + Prettier).
- Expose API docs via Swagger at `/docs`
- Secure API with basic middlewares (CORS, Helmet).

Deliverables

- :`README.md` including
  - Setup and run instructions
  - Test instructions
  - Time spent and cut corners
  - Screenshots or GIFs of map, list, and form
- Optional: Postman collection or `.http` file for API calls

Evaluation Criteria

- .**Correctness & completeness (30%)** – meets core requirements
- .**Code quality & architecture (20%)** – clear, modular, maintainable
- .**State & data flow (15%)** – proper React Query + form validation
- .**External API integration (15%)** – robustness, caching, error handling
- .**Testing (10%)** – meaningful unit/controller tests
- .**UX polish & accessibility (10%)** – responsive, usable, and clear

---

Time expectation: about **8 hours**. ✅
. If you run out of time, please document trade-offs and assumptions in the README