Technical Assessment

MERN Stack Developer

Concept

PropTrack – Real Estate Listings & Client Management Platform

At PropTrack, we are building a real estate web platform to help agents manage property listings, client inquiries, and scheduled viewings. The platform is designed to streamline the property management workflow while providing users with a smooth and intuitive browsing experience for real estate listings.

This usually involves creating and managing detailed property listings, receiving client inquiries, and scheduling viewings — all while ensuring high performance at scale and clean separation of backend and frontend responsibilities.

Before starting this assessment, consider PropTrack as both an internal agent dashboard (CMS) and a **public-facing listing portal**. The core of the system revolves around three key entities:

- **Property**: Describes a unit available for sale or rent. Each property includes details such as title, price, type (rent/sale), location, amenities, and images.
- **Client**: Represents a user who has inquired about a property. A client can have one or more scheduled viewings.
- **Viewing**: A scheduled time where an agent meets a client for a property showing. Viewings are tracked with date, time, and status.

User Stories and Functional Requirements

Your task is to build a small web application that allows users (agents and public users) to do the following:

For the Public Interface:

- View a list of all **active properties** with pagination
- Filter and search the property listings by:
    - Price range
    - Location
    - Property type (sale/rent)
    - Bedrooms, bathrooms, area and amenities

- Click a property to see full details and an image gallery

- Submit an inquiry form (which should be stored as a client lead in the system)

For the Agent Dashboard:

- Add a new property listing with full details (title, description, price, location, type, images, etc.)
- Edit, archive, or delete existing properties
- Filter/search property listings on the dashboard
- View a list of client inquiries (automatically generated from the public inquiry form)
- Schedule a viewing for a specific property and client
- Mark a viewing as completed or a no-show, and add internal notes

Non-functional Requirements

- Your code should be written using the MERN stack (MongoDB, Express, React, Node.js)
- All property data must be persisted in MongoDB. Apply indexing to ensure fast filtering and lookups for 10,000+ listings
    - Make use of MongoDB's aggregation pipeline where applicable (e.g, filtering, searching)
- You must use a shared/central state (implemented by a framework like Redux, Mobx, Zustand, etc). Your app cannot rely only on the use of useState or useContext. This may seem unnecessary for a small application like this, but it's important to demonstrate your ability to work with state management frameworks.
    - The state management should be designed in a way that allows the app to expand and grow to include more complex features, without becoming laggy or buggy
- Your work should be developed in a manner consistent with industry standards and the lifecycle of software development. You shouldn't just have a single commit with all the code in it. Your commits should be made just like you would if you're building this in the real world, with commits having well-defined boundaries and scope, meaningful commit messages, and so on
    - We will not have a code review process like the real world, but imagine that each commit should be like a PR that one of your colleagues would have reviewed if you were working in the real world

Non-requirements

You do not need to implement the following for this task:

- Authentication or user session management (login, signup, role permissions)
- Third-party email or chat integrations
- Real SMS or calendar integrations for viewings

- Production-grade file/image hosting (you may use placeholder URLs or simple base64 images)

Bonus Features

- Show off your UI skills by building beautiful interfaces
- Add support for internationalization
- Real-time chat between client and agent
- Add map view using Google Maps API or Leaflet.js
- Anything else you'd like to show off
    - This is an opportunity to show your strengths, so if there is a feature or a functionality you want to build to demonstrate your skills, please do so.

Submission Instructions

Upload your work to a public **GitHub** repo. Your repo should have a README.md file that includes the following:

- Instructions on how to run the app from scratch
- Screenshots of the views you created. Video demo is a bonus
- Any assumptions you made to be able to complete this task
- A summary of the technical choices you made in building the app and an explanation for why you did them
- Future plans or things you would do differently or add if you had more time
- Any stretch goals or bonus features you attempted
- Any shortcuts you took or compromises you made
- The total amount of time spent on this assessment
- Any resources, tools, frameworks, or technologies you utilized in your solution
- Anything else you'd like to share with us

Commit your code to a public repo and provide the link by **replying** to the same email thread

Evaluation Criteria

The purpose of this assessment is to simulate the real world as much as possible. That's why we are trying to only ask for things that are relevant in the real-world implementation of the applications and services we are building.

Overall, we will look for the following things to assess submissions:

- Code quality and cleanliness
- Utilization of good software design principles

- Completeness of features/requirements
- Incremental changes and meaningful commit history
- Quality of requested documentation (via README.md file)
- Stretch goals or bonus features attempted

Final thoughts

In real-life projects, requirements are often evolving or incomplete, and you will need to make judgment calls based on context, user experience, and technical constraints.

For this assessment, we encourage you to decide on any missing details yourself. Please document all such decisions, assumptions, or interpretations clearly in your README.md. This helps us understand your thought process and how you approach problem-solving in uncertain environments.

To maintain fairness and consistency, we ask that you do **not** reach out to us with clarification questions during the assessment. Instead, make informed decisions or creative assumptions where needed, and document them accordingly.

There are many directions in which you could take this solution. We invite you to bring your own perspective, structure, and UX touch to the solution. This is your opportunity to showcase not just how you code, but how you think like a product developer.

If this assessment feels broad, that's intentional. Treat the requirements as a superset. You are not expected to implement everything. Use your judgment to define an MVP version of the system and focus on building a functional, clean, and extensible solution. We only expect you to spend the equivalent of **a single productive working day**—prioritize accordingly and let your solution reflect how you would approach a tight, real-world sprint.