# Constructure AI - Applied LLM Engineer Technical Assignment (36-Hour Challenge)

You have 36 hours to submit this.

## Role and Tech Constraints

This assignment is for an Applied LLM / AI Engineer role.

You will be evaluated on how you:

- Design retrieval and generation workflows
- Handle messy, semi-structured construction data
- Build something testable, not just a notebook demo

**Required tech stack:**

- Backend: FastAPI
- Frontend: React or Next.js
- AI: Any LLM provider (for example: OpenAI, Anthropic, local model, etc.)

You are free to structure the system however you want within these constraints.

## High-Level Goal

Build a mini "Project Brain" for a single construction project:

- Ingest a small set of project documents (PDFs and schedules we will provide).
- Expose a chat interface where a user can:
  - Ask natural language questions about the project.
  - Request structured information (for example: "give me a door schedule").
  - Return source-backed answers and structured outputs using an LLM powered pipeline.

This should feel like a tiny, opinionated version of Constructure AI.

**Part 0 - Deployment and Testability (Required)**

We must be able to use what you build.

**Requirements:**

- Deploy the frontend to Vercel (free tier) and include a live URL in your README, such as:
  https://your-app-name.vercel.app
- The frontend should talk to your FastAPI backend (cloud, tunnel, or Vercel-compatible API).
- We should be able to:
  - Open the URL
  - Interact with the chat
  - Run queries against the sample project documents

Your README should clearly explain:

- How to run the backend locally
- What environment variables are needed (keys, model names, etc.)

No need to over-engineer infrastructure, but this should feel like a real app, not just a script.

**Part 1 - Document Ingestion and Indexing**

You will be given a small set of construction project files (for example: specs, a drawing export, and a schedule PDF).

Your system should:

- Ingest multiple documents for a single project.
- Split them into searchable units (pages, sections, or chunks) using heuristics of your choice.
- Store them in some type of searchable index, including:
  - Text content
  - File name
  - Page or section reference
  - Any other metadata you find useful

You can use any combination of:

- Vector search
- Keyword search
- Hybrid retrieval

We will not tell you how to chunk, embed, or store data — that is part of what we are evaluating.

## Part 2 - Project Q and A Chat (RAG)

Build a simple chat interface on your React or Next frontend that talks to your FastAPI backend.

The chat should support natural language questions such as:

- “What is the fire rating for corridor partitions?”
- “What is the specified flooring material in the lobby?”
- “Are there any accessibility requirements for doors?”

For each answer:

- Use your retrieval pipeline to fetch relevant text from the ingested documents.
- Use an LLM to generate a concise answer.
- Return citations: file name plus page or section for the sources used.
- Show the full conversation on the frontend (user and AI messages).

We are looking for:

- Reasonable retrieval quality
- Sensible prompting and answer formatting
- Clear links back to sources (no hallucinations without caveats)

## Part 3 - Structured Extraction Task (Door or Element Schedule)

This is where it becomes closer to Constructure.

Implement at least one structured extraction flow, for example:

- Door schedule extraction
- Room summary extraction
- MEP equipment list extraction

(depends on the provided documents)

**Requirements:**

The user should be able to trigger this through the chat, for example:

- “Generate a door schedule”
- “List all rooms with their area and floor finish”

Your backend should:

- Retrieve relevant parts of the documents.
- Use an LLM (and any logic you design) to extract a structured JSON result.

**Example shape (for a door schedule):**

```
[
  {
    "mark": "D-101",
    "location": "Level 1 Corridor",
    "width_mm": 900,
    "height_mm": 2100,
    "fire_rating": "1 HR",
    "material": "Hollow Metal"
  }
]
```

Your frontend should:

- Display the structured result (for example: a table).
- Still, show which pages or sections the data came from.

You are free to design the exact schema, but it should be:

- Consistent
- Reasonably complete given the data
- Not obviously overfitted to one example

**Part 4 - Minimal Evaluation and Introspection**

We want to see how you think about quality, not only the system's basic functionality.

Implement a lightweight evaluation or introspection step, for example:

- A small set of hard-coded test queries (5 to 10) with expected rough answers.
- A simple script or endpoint that:
  - Runs these queries through your pipeline
  - Logs the model answers and their sources
  - Outputs something like "looks correct / partially correct / wrong" based on simple heuristics or manual labels

We are not expecting a full evaluation system just enough to show that you are thinking about:

- Failure modes
- Retrieval errors
- Model hallucinations

**Bonus (Totally Optional but Very Impressive)**

If you have time within the 36 hours, any of these will stand out:

**Multi-strategy Retrieval**

- Combine vector and keyword search and rerank results before sending to the LLM.

**Per-Tool Behavior**

Different "modes" or "tools" in the chat:

- Q and A mode
- Schedule extraction mode
- "Show me sources only" mode

**Caching**

- Cache retrieval results or LLM responses to speed up repeated queries.

**Basic Analytics**

- Log the types of questions asked and the documents most frequently used.

**Red-Flag Detection**

Add a command such as:

- “Are there any conflicting specs about door fire ratings?”

Try to detect inconsistencies using retrieval plus LLM.

These are intentionally non-trivial only build them if your core flows are solid.

**Submission**

Your public GitHub repo should include:

**README with:**

- Short description of your solution
- How to run the FastAPI backend locally
- How to run the React or Next frontend locally
- Required environment variables (for example: OPENAI_API_KEY)
- Deployed Vercel URL
- Brief notes on:
    - How you chunk and index the documents
    - How your RAG pipeline works
    - What you chose for the structured extraction task
    - Any simple scripts or tools used for ingestion or evaluation

**How We Evaluate**

We will look at:

**Core functionality**

- Can we log in with testingcheckuser1234@gmail.com
- Can we load your app, ask real questions about the project documents, and get usable answers with sources?
- Does your structured extraction produce sensible structured data?

**AI / RAG design**

- Chunking, retrieval, prompting, error handling
- How you handle noisy or partial documents

**Code quality**

- FastAPI and React or Next organization
- Clear separation between data, retrieval, and LLM logic

**Product sense**

- Does this feel like a plausible small slice of Constructure AI, not just a toy?

**Depth**

- How far you pushed within the 36-hour window
- Whether you attempted any introspection or bonus features meaningfully