

Constructure AI - Technical Assignment (36 Hour Challenge)

Welcome to the Constructure AI Applicant Challenge!

This assignment is meant to simulate real product work under time constraints. We are looking for candidates who can ship something useful, robust, and thoughtful in 36 hours.

You will build a mini-AI powered email assistant with:

- Google login
- AI chatbot dashboard
- Email automation (read, respond, delete)

Required Tech Stack

Backend: FastAPI

Frontend: React or Next.js

Deployment: Must be testable through a live Vercel (free tier) deployment

You are free to choose any libraries, patterns, and AI provider within those constraints.

Part 0 - Deployment and Testability (Required)

We must be able to use your app end to end.

Requirements:

- Deploy your frontend to Vercel and include a live URL in the README, for example:
<https://your-app-name.vercel.app>
- Your backend must be reachable by the deployed frontend.
- In your Google Cloud OAuth settings, add **test@gmail.com** as a test user so we can log in during review.
- The app should be usable without any code changes on our side (only environment or config as described in your README).

Part 1 - Google Authentication

Create a login experience using Google OAuth.

Core requirements:

- Implement Google OAuth2 login.
- Request Gmail permissions to:
 - Read emails
 - Send emails
 - Delete emails
- On successful login:
 - Persist an authenticated session.
 - Redirect the user to the chatbot dashboard.
- Handle:
 - Failed logins
 - Revoked permissions
 - Expired sessionswith clear, user friendly messages.

You decide how to structure authentication between the frontend and FastAPI. That is part of what we are evaluating.

Part 2 - Chatbot Dashboard

After login, the user should land on a dashboard containing an AI chatbot interface.

The chatbot should:

- Greet the user using their Google profile information.
- Briefly explain available capabilities or commands on the first load (in natural language).
- Show a conversation thread where:
 - User messages are displayed.
 - System or AI responses and status updates are displayed.
- Accept user input that can trigger email related operations (see Part 3).

The goal is not fancy design, but the UI must be clean, understandable, and usable.

Part 3 - Email Automation

Once a user is authenticated, your system should integrate with Gmail to perform real actions.

3.1 Read Last 5 Emails

On a user command (your choice of wording or format, but it should feel natural):

- Fetch the 5 most recent emails from the user's inbox.
- For each email, show in the chatbot:
 - Sender (name and email if available)
 - Subject
 - A short AI generated summary of the content

This summary must be produced by an AI model, not by simply truncating the body.

3.2 Generate AI Responses

For those same emails (or through a separate command you define):

- Generate proposed replies using an AI model.
- Replies should be:
 - Context aware (based on the original email content)
 - Clear and professional
 - Ready to send

In the chatbot:

- Display each suggested reply clearly tied to its original email.
- Allow the user to confirm sending a reply through Gmail.
- After confirmation, attempt to send the email and report success or failure in the conversation.

3.3 Delete a Specific Email

Implement deletion of a specific email based on user intent.

You decide the UX and parsing approach, but at minimum the user should be able to delete:

- By sender (example: "delete the latest email from ..."), or
- By subject keyword, or
- By referencing one of the previously listed emails (example: "delete email number 2")

Requirements:

- Ask for confirmation before deletion.
- On confirmation, delete the email from Gmail.
- Reflect the result in the chatbot (success or failure).

Try to avoid hard coding specific email IDs or account details. The logic should work for any Gmail account with the right permissions.

Part 4 - Bonus Challenges

You may implement any subset.

Natural Language Command Understanding

Let users type things like:

- "Show me the last few important emails about invoices"
- "Reply to John that I will get back tomorrow"

Map these to your internal actions (read, respond, delete) instead of relying on strict commands.

Smart Inbox Grouping or Categorization

Fetch more than 5 emails (example: last 20) and use AI to group them into categories like:

- Work
- Promotions
- Personal
- Urgent

Show a structured summary of each group in the chatbot.

Daily Digest

A command like:

- "Give me today's email digest"

should produce a single AI generated digest summarizing:

- Key emails for the day

- Suggested actions or follow ups

Observability and Resilience

- Log key events (auth success or failure, Gmail calls, AI calls, send or delete attempts).
- Show some notion of status (example: "contacting Gmail...", "AI generation failed, retrying...").
- Basic retry behaviour for transient errors.

Automated Tests

Add a small but meaningful test suite for:

- Core backend logic (example: parsing Gmail responses, mapping commands to actions)
- Any critical flows you consider important

We do not expect full coverage, but we look for intentional testing.

Submission

Provide a public GitHub repository that includes:

- README with:
 - A brief description of your solution
 - Setup instructions (FastAPI and React or Next)
 - How to configure Google credentials and OAuth
 - Required environment variables
 - Live Vercel URL: <https://your-app-name.vercel.app>
 - Technologies used (libraries, AI provider, etc.)
 - Any assumptions or known limitations

Evaluation Criteria

We will evaluate your work on:

- **Functionality**
Can we log in with testingcheckuser1234@gmail.com and read, reply to, and delete emails?

- **Code Quality**
Structure, clarity, naming, and organization across FastAPI and React or Next.
- **Product Thinking and UX**
Is the flow understandable, and does the chatbot feel like a real assistant rather than a demo?
- **Use of AI**
Are summaries and replies helpful and context aware?
- **Stability and Error Handling**
What happens when APIs fail, tokens expire, or something goes wrong?
- **Ambition**
How much did you attempt within the 36 hour window?
Did you tackle any bonus challenges meaningfully?