Assignment: Development of a Prompt-Driven Email Productivity Agent

Objective

Build an intelligent, prompt-driven Email Productivity Agent capable of processing an inbox (real or mock) and performing automated tasks such as:

- Email categorization
- Action-item extraction
- Auto-drafting replies
- Chat-based inbox interaction

The system will ingest user-defined prompts ("the agent brain") and use them to guide all LLM-powered operations.
A Streamlit or web-based UI (React/HTML/JS allowed) must be included. Backend architecture is flexible.

Submission Requirements

Candidates must provide:

1. Source Code Repository

GitHub/GitLab with full application code.

2. README.md Including

- Setup instructions
- How to run the UI and backend
- How to load the Mock Inbox
- How to configure prompts
- Usage examples

3. Project Assets

Provide or generate:

- A Mock Inbox (JSON or SQLite) with 10–20 sample emails
- Default prompt templates (categorization, action-item extraction, auto-reply, etc.)

4. Demo Video (5–10 mins)

Must demonstrate:

- Loading inbox
- Editing/creating custom prompts
- Email ingestion + automatic categorization & action-item extraction
- Using the Email Agent chat to summarize, reply, and generate emails

Evaluation Criteria

1. Functionality

- Inbox ingestion works
- Emails are categorized and parsed using prompts
- LLM successfully generates summaries, replies, and suggestions
- Drafts are safely stored, not sent

2. Prompt-Driven Architecture

- User can create, edit, and save prompts
- Behavior of the agent must change based on user-defined prompts
- All LLM outputs must use the stored prompts

3. Code Quality

- Clear separation of UI, backend services, state management, and LLM integration
- Readable, modular, commented code

4. User Experience

- Clean prompt configuration panel
- Intuitive inbox viewer
- Smooth “Email Agent” chat interface

5. Safety & Robustness

- Handles LLM errors gracefully
- Defaults to “draft” instead of sending emails

Project Assets

To be provided by the candidate:

1. Mock Inbox (JSON or SQLite)

Should include 10–20 sample emails such as:

- Meeting requests
- Newsletters
- Spam-like messages
- Task requests
- Project updates

2. Default Prompt Templates

Examples:

Categorization Prompt

"Categorize emails into: Important, Newsletter, Spam, To-Do.
To-Do emails must include a direct request requiring user action."

Action Item Extraction Prompt

"Extract tasks from the email. Respond in JSON:
{ "task": "...", "deadline": "..." }."

Auto-Reply Draft Prompt

"If an email is a meeting request, draft a polite reply asking for an agenda."

Functional Requirements

The system consists of three major phases, similar to the QA Agent assignment.

Phase 1: Email Ingestion & Knowledge Base (Prompt Storage)

UI Requirements

The user must be able to:

1. **Load emails**
   - Connect to email service or load Mock Inbox
2. **View list of emails**
   Display:
   - Sender
   - Subject
   - Timestamp
   - Category tags (after processing)
3. **Create and Edit Prompt Configurations**
   A "Prompt Brain" panel with fields for:
   - Categorization Prompt
   - Action Item Prompt
   - Auto-Reply Draft Prompt

Backend Requirements

- Store prompts in a database or JSON file
- Store processed outputs (categories, extracted actions, drafts)
- Provide an ingestion pipeline:
  1. Load emails
  2. Run categorization prompt via LLM
  3. Run action item prompt via LLM
  4. Save results to state
  5. Update UI

Phase 2: Email Processing Agent (RAG Optional)

UI Requirements

Provide an "Email Agent" section where the user can:

- Select an email
- Ask questions like "Summarize this email"

- Ask: “What tasks do I need to do?”
- Ask: “Draft a reply based on my tone”
- Ask general inbox tasks (“Show me all urgent emails”)

Agent Logic

1. The agent receives:
    - User query
    - Email content
    - Stored prompts
2. It then constructs an LLM request combining:
    - The email's text
    - The relevant prompt (e.g., categorization, reply)
    - The user's instruction
3. LLM returns the structured or text output
4. The agent displays results in the UI

Phase 3: Draft Generation Agent

UI Requirements

User must be able to:

- Generate new email drafts
- Ask the agent to write replies
- Edit drafts
- Save drafts

Agent Logic

- When drafting, the agent must:
    - Use the user’s auto-reply prompt
    - Use email thread context (if replying)
    - Never send emails automatically
    - Store drafts for user review

Output Requirements

Drafts should include:

- Subject
- Body
- Optional suggested follow-ups
- JSON metadata for category/action items