Full-Stack Email Aggregator Project

1. Real-Time Email Synchronization

- - Sync multiple IMAP accounts in real-time (minimum 2).
- - Fetch at least the last 30 days of emails.
- - Use persistent IMAP connections (IDLE mode) for real-time updates (No cron jobs!).

2. Searchable Storage using Elasticsearch

- - Store emails in a locally hosted Elasticsearch instance (use Docker).
- - Implement indexing to make emails searchable.
- - Support filtering by folder & account.

3. AI-Based Email Categorization

- - Implement an AI model to categorize emails into the following labels:
- - Interested
- - Meeting Booked
- - Not Interested
- - Spam
- - Out of Office

4. Slack & Webhook Integration

- - Send Slack notifications for every new Interested email.
- - Trigger webhooks (use webhook.site as the webhook URL) for external automation when an email is ma

5. Frontend Interface

- - Build a simple UI to display emails, filter by folder/account, and show AI categorization.
- - Basic email search functionality powered by Elasticsearch.

6. AI-Powered Suggested Replies

- - Store the product and outreach agenda in a vector database.
- - Use RAG (Retrieval-Augmented Generation) with an LLM to suggest replies.
- - Example AI Reply Suggestion:
- 'Thank you for shortlisting my profile! I'm available for a technical interview. You can book a slot here: ht

Technology Stack

- - Backend: Node.js, TypeScript, Express.js
- - IMAP Email Handling: IMAP.js
- - Search: Elasticsearch
- - AI Categorization: TensorFlow.js / OpenAI API
- - Vector Database: Pinecone / FAISS
- - Frontend: React.js, Tailwind CSS
- - Notifications: Slack API, Webhooks
- - Deployment: Docker, GitHub Actions

How to Run the Project

- 1. Clone the repository.
- 2. Set up environment variables for IMAP, Elasticsearch, and AI APIs.
- 3. Run `docker-compose up` to start Elasticsearch.
- 4. Start the backend server with `npm run start`.
- 5. Start the frontend with `npm run dev`.
- 6. Open the browser to access the email UI.