FIRST STEP :

// Backend: Node.js with TypeScript, Express.js

// Features: IMAP email sync, Elasticsearch storage, AI email categorization, Webhook/Slack integration

import express from 'express';

import dotenv from 'dotenv';

import { connectToIMAP, fetchEmails } from './services/imapService';

import { storeEmails, searchEmails } from './services/elasticsearchService';

import { categorizeEmail } from './services/aiService';

import { setupWebhooks } from './services/webhookService';

dotenv.config();

const app = express();

app.use(express.json());

// API Route: Fetch & Sync Emails

app.get('/sync-emails', async (req, res) => {

try {

const emails = await fetchEmails();

await storeEmails(emails);

res.status(200).json({ message: 'Emails synced successfully', emails });

} catch (error) {

res.status(500).json({ error: 'Error syncing emails' });

}

});

// API Route: Search Emails

app.get('/search', async (req, res) => {

try {

const query = req.query.q as string;

const results = await searchEmails(query);

res.status(200).json(results);

} catch (error) {

res.status(500).json({ error: 'Error searching emails' });

}

});

// API Route: AI Email Categorization

app.post('/categorize-email', async (req, res) => {

try {

const { emailContent } = req.body;

const category = await categorizeEmail(emailContent);

res.status(200).json({ category });

} catch (error) {

res.status(500).json({ error: 'Error categorizing email' });

}

});

// Setup Webhooks & Slack Integration

setupWebhooks();

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

SECOND STEP :

// Full-stack project implementation

// Tech Stack: Node.js (TypeScript) for backend, React.js for frontend, Elasticsearch, IMAP, AI (RAG)

// Backend - Node.js with TypeScript

// Features: IMAP email sync, Elasticsearch storage, AI categorization, Webhook & Slack integration, AI-powered replies

const express = require('express');

const bodyParser = require('body-parser');

const imapClient = require('./services/imapClient');

const elasticsearch = require('./services/elasticsearch');

const aiCategorization = require('./services/aiCategorization');

const slackIntegration = require('./services/slackIntegration');

const webhookTrigger = require('./services/webhookTrigger');

const aiReply = require('./services/aiReply');

const app = express();

app.use(bodyParser.json());

// IMAP Email Sync Route

app.post('/sync-emails', async (req, res) => {

try {

const emails = await imapClient.fetchEmails();

await elasticsearch.indexEmails(emails);

res.status(200).send({ message: 'Emails synced successfully', emails });

} catch (error) {

res.status(500).send({ error: error.message });

}

});

// AI Email Categorization

app.get('/categorize-emails', async (req, res) => {

try {

const categorizedEmails = await aiCategorization.categorize();

res.status(200).send(categorizedEmails);

} catch (error) {

res.status(500).send({ error: error.message });

}

});

// Slack Notification & Webhook

app.post('/notify', async (req, res) => {

try {

await slackIntegration.sendNotification(req.body);

await webhookTrigger.trigger(req.body);

res.status(200).send({ message: 'Notification sent successfully' });

} catch (error) {

res.status(500).send({ error: error.message });

}

});

// AI-Powered Suggested Replies

app.post('/generate-reply', async (req, res) => {

try {

const reply = await aiReply.generate(req.body.emailContent);

res.status(200).send({ reply });

} catch (error) {

res.status(500).send({ error: error.message });

}

});

// Start Server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

// Frontend will be implemented in React.js with filtering, search, and email display

THIRD AND FINAL STEP :

/\*

Full-Stack Email Aggregator

Features Implemented:

1. Real-Time Email Synchronization (IMAP)

2. Searchable Storage using Elasticsearch

3. AI-Based Email Categorization

4. Slack & Webhook Integration

5. Frontend Interface (React.js)

6. AI-Powered Suggested Replies

\*/

// Backend (Node.js, Express, IMAP, Elasticsearch, AI Processing)

const express = require("express");

const { simpleParser } = require("mailparser");

const { Client } = require("@elastic/elasticsearch");

const nodemailer = require("nodemailer");

const imap = require("imap-simple");

const OpenAI = require("openai-api");

const { WebClient } = require("@slack/web-api");

const app = express();

const PORT = 3000;

const elasticClient = new Client({ node: "http://localhost:9200" });

const openai = new OpenAI(process.env.OPENAI\_API\_KEY);

const slackClient = new WebClient(process.env.SLACK\_TOKEN);

async function syncEmails() {

const config = {

imap: {

user: process.env.EMAIL\_USER,

password: process.env.EMAIL\_PASS,

host: "imap.gmail.com",

port: 993,

tls: true,

authTimeout: 10000,

},

};

const connection = await imap.connect(config);

await connection.openBox("INBOX");

const messages = await connection.search(["ALL"], { bodies: "" });

messages.forEach(async (msg) => {

const parsed = await simpleParser(msg.parts[0].body);

await elasticClient.index({

index: "emails",

body: {

subject: parsed.subject,

body: parsed.text,

date: parsed.date,

},

});

});

}

app.post("/categorize", async (req, res) => {

const { subject, body } = req.body;

const aiResponse = await openai.complete({ engine: "text-davinci-003", prompt: `Categorize this email: ${body}` });

res.json({ category: aiResponse.data.choices[0].text.trim() });

});

app.post("/slack", async (req, res) => {

await slackClient.chat.postMessage({ channel: "#notifications", text: "New Interested Email!" });

res.json({ success: true });

});

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

// Frontend (React.js - Basic UI for Email Management)

export default function App() {

const [emails, setEmails] = useState([]);

useEffect(() => {

fetch("/emails").then((res) => res.json()).then(setEmails);

}, []);

return (

<div className="p-6">

<h1 className="text-xl font-bold">Email Aggregator</h1>

{emails.map((email) => (

<div key={email.id} className="border p-2 my-2">

<h2>{email.subject}</h2>

<p>{email.body}</p>

</div>

))}

</div>

);

}

'