Full Backend Code - Email Deliverability Tool

Generated: 2025-10-16T16:42:58.028450 UTC

--- package.json ---

```
{   "name": "email-deliverability-backend",   "version": "1.0.0",   "main": "server.js",   "scripts": {
"start": "node server.js",     "dev": "nodemon server.js"   },   "dependencies": {     "axios": "^1.4.0",
"body-parser": "^1.20.2",     "express": "^4.18.2",     "mongoose": "^7.0.0",     "nodemailer": "^6.9.3",
"uuid": "^9.0.0",     "dotenv": "^16.0.0",     "googleapis": "^126.0.0",     "@azure/msal-node": "^2.0.0",
"node-fetch": "^2.6.7"   },   "devDependencies": {     "nodemon": "^2.0.0"   } }
```

--- .env.example ---

```
# Server PORT=4000 BASE_URL=http://localhost:3000  # MongoDB
MONGODB_URI=mongodb+srv://<user>:<pw>@cluster0.mongodb.net/emailtool?retryWrites=true&w=majority  # Gmail
(create OAuth2 credentials and obtain refresh tokens for each inbox) GMAIL_CLIENT_ID=your-google-client-id
GMAIL_CLIENT_SECRET=your-google-client-secret # One refresh token per Gmail inbox you control:
GMAIL_REFRESH_TOKEN_INBOX1=refresh_token_for_inbox1 GMAIL_REFRESH_TOKEN_INBOX2=refresh_token_for_inbox2  #
Outlook (Azure app + refresh tokens) OUTLOOK_CLIENT_ID=your-azure-client-id OUTLOOK_CLIENT_SECRET=your-azure-
client-secret OUTLOOK_REFRESH_TOKEN_INBOX1=refresh_token_for_outlook_inbox1  # SMTP (for sending report
emails) SMTP_HOST=smtp.sendgrid.net SMTP_PORT=587 SMTP_USER=apikey SMTP_PASS=your_sendgrid_api_key
FROM_EMAIL=no-reply@yourdomain.com  # Poller settings POLL_INTERVAL_MS=5000 POLL_TIMEOUT_MS=300000
```

--- server.js ---

```
require('dotenv').config(); const app = require('./src/app'); const mongoose = require('mongoose');  const
PORT = process.env.PORT || 4000;  async function start() {   if (!process.env.MONGODB_URI) {
console.error('MONGODB_URI not set');     process.exit(1);   }   await
mongoose.connect(process.env.MONGODB_URI);   console.log('Connected to MongoDB');    app.listen(PORT, () =>
console.log(`Server listening on ${PORT}`)); } start();
```

--- src/app.js ---

```
const express = require('express'); const bodyParser = require('body-parser'); const cors = require('cors');
const inboxesRouter = require('./routes/inboxes'); const testsRouter = require('./routes/tests');  const app =
express(); app.use(cors()); app.use(bodyParser.json());  app.use('/api/inboxes', inboxesRouter);
app.use('/api/tests', testsRouter);  app.get('/', (req, res) => res.send('Email Deliverability Tool API'));
module.exports = app;
```

--- src/models/Test.js ---

```
const mongoose = require('mongoose');  const InboxResult = new mongoose.Schema({   provider: String,
address: String,   received: { type: Boolean, default: false },   folder: { type: String, default: null },
messageId: { type: String, default: null },   checkedAt: { type: Date, default: null } });  const TestSchema =
new mongoose.Schema({   testCode: String,   userEmail: String,   status: { type: String, enum:
['pending','completed','failed'], default: 'pending' },   createdAt: { type: Date, default: Date.now },
inboxes: [InboxResult],   reportUrl: String,   score: Number });  module.exports = mongoose.model('Test',
TestSchema);
```

--- src/routes/inboxes.js ---

```
const express = require('express'); const router = express.Router();  // Replace these addresses with inboxes
you control and have OAuth access for const TEST_INBOXES = [   { provider: 'gmail', address:
'test1+inbox@gmail.com' },   { provider: 'gmail', address: 'test2+inbox@gmail.com' },   { provider: 'outlook',
address: 'test3@outlook.com' },   { provider: 'outlook', address: 'test4@outlook.com' },   { provider:
'custom', address: 'test5@yourdomain.com' } ];  router.get('/', (req, res) => {   res.json(TEST_INBOXES); });
module.exports = router;
```

--- src/routes/tests.js ---

```
const express = require('express'); const router = express.Router(); const Test = require('../models/Test');
const generateCode = require('../utils/generateCode'); const poller = require('../services/poller'); const
emailSender = require('../utils/emailSender');  // Create a new test (generate code) router.post('/', async
(req, res) => {   try {     const { userEmail } = req.body;     if (!userEmail) return res.status(400).json({
error: 'userEmail required' });      const testCode = generateCode();      const inboxes = [       { provider:
```

'gmail', address: 'test1+inbox@gmail.com' }, { provider: 'gmail', address: 'test2+inbox@gmail.com' }, { provider: 'outlook', address: 'test3@outlook.com' }, { provider: 'outlook', address: 'test4@outlook.com' }, { provider: 'custom', address: 'test5@yourdomain.com' }]; const test = new Test({ testCode, userEmail, inboxes }); await test.save(); res.json({ testId: test._id, testCode, inboxes }); } catch (err) { console.error('Create test error:', err); res.status(500).json({ error: 'server error' }); } }); // Start test (polling) router.post('/:testId/start', async (req, res) => { try { const { testId } = req.params; const test = await Test.findById(testId); if (!test) return res.status(404).json({ error: 'Test not found' }); poller.startPolling(test._id.toString()); res.json({ message: 'Polling started' }); } catch (err) { console.error('Start test error:', err); res.status(500).json({ error: 'server error' }); } }); // Get test status/result router.get('/:testId', async (req, res) => { try { const test = await Test.findById(req.params.testId); if (!test) return res.status(404).json({ error: 'Test not found' }); res.json(test); } catch (err) { console.error('Get test error:', err); res.status(500).json({ error: 'server error' }); } }); module.exports = router;

--- src/utils/generateCode.js ---

const { v4: uuidv4 } = require('uuid'); module.exports = function generateCode() { return uuidv4().split('-')[0].toUpperCase(); // short code like 'A1B2C3' }

--- src/utils/emailSender.js ---

const nodemailer = require('nodemailer'); const transporter = nodemailer.createTransport({ host: process.env.SMTP_HOST, port: parseInt(process.env.SMTP_PORT || '587'), auth: { user: process.env.SMTP_USER, pass: process.env.SMTP_PASS } }); async function sendReportEmail(to, subject, html) { const info = await transporter.sendMail({ from: process.env.FROM_EMAIL, to, subject, html }); return info; } module.exports = { sendReportEmail };

--- src/services/poller.js ---

const Test = require('../models/Test'); const gmailClient = require('./mailClients/gmailClient'); const outlookClient = require('./mailClients/outlookClient'); const emailSender = require('../utils/emailSender'); const POLL_INTERVAL_MS = parseInt(process.env.POLL_INTERVAL_MS || '5000'); const POLL_TIMEOUT_MS = parseInt(process.env.POLL_TIMEOUT_MS || '300000'); // 5min const activePolls = new Map(); async function checkInboxForCode(inbox, testCode) { const provider = inbox.provider; try { if (provider === 'gmail') { return await gmailClient.searchMessages(inbox.address, testCode); } if (provider === 'outlook') { return await outlookClient.searchMessages(inbox.address, testCode); } // custom or other providers - not implemented return { found: false }; } catch (err) { console.error('checkInboxForCode error', err); return { found: false }; } } async function pollOnce(testId) { const testDoc = await Test.findById(testId); if (!testDoc) return; let changed = false; for (let i = 0; i < testDoc.inboxes.length; i++) { const inbox = testDoc.inboxes[i]; if (inbox.received) continue; const res = await checkInboxForCode(inbox, testDoc.testCode); if (res && res.found) { inbox.received = true; inbox.folder = res.folder || 'Unknown'; inbox.messageId = res.id || null; inbox.checkedAt = new Date(); changed = true; } } if (changed) await testDoc.save(); const allChecked = testDoc.inboxes.every(i => i.received === true); if (allChecked) { testDoc.status = 'completed'; testDoc.score = Math.round((testDoc.inboxes.filter(i => i.folder === 'Inbox').length / testDoc.inboxes.length) * 100); testDoc.reportUrl = `${process.env.BASE_URL || 'http://localhost:3000'}/report/${testDoc._id}`; await testDoc.save(); // send report email to user const html = `<p>Your deliverability test is complete. <a href="${testDoc.reportUrl}">View report</a></p>`; try { await emailSender.sendReportEmail(testDoc.userEmail, 'Deliverability Test Report', html); } catch (err) { console.error('Error sending report email', err); } return 'done'; } return 'continue'; } async function startPolling(testId) { if (activePolls.has(testId)) return; const startAt = Date.now(); const interval = setInterval(async () => { try { const result = await pollOnce(testId); if (result === 'done') { clearInterval(interval); activePolls.delete(testId); } else if (Date.now() - startAt > POLL_TIMEOUT_MS) { const testDoc = await Test.findById(testId); if (testDoc) { testDoc.status = 'failed'; await testDoc.save(); } clearInterval(interval); activePolls.delete(testId); } } catch (err) { console.error('polling error', err); } }, POLL_INTERVAL_MS); activePolls.set(testId, interval); } module.exports = { startPolling };

--- src/services/mailClients/gmailClient.js ---

// Gmail client using googleapis OAuth2 and Gmail REST API const { google } = require('googleapis'); // Map inbox address to refresh token env vars const INBOX_TOKEN_MAP = { 'test1+inbox@gmail.com': process.env.GMAIL_REFRESH_TOKEN_INBOX1, 'test2+inbox@gmail.com': process.env.GMAIL_REFRESH_TOKEN_INBOX2 }; function getOauth2ClientForInbox(inboxAddress) { const refreshToken = INBOX_TOKEN_MAP[inboxAddress]; if

```
(!refreshToken) throw new Error('No refresh token configured for ' + inboxAddress);   const oAuth2Client = new google.auth.OAuth2(process.env.GMAIL_CLIENT_ID, process.env.GMAIL_CLIENT_SECRET);
oAuth2Client.setCredentials({ refresh_token: refreshToken });   return oAuth2Client; }  async function searchMessages(inboxAddress, testCode) {   const client = getOauth2ClientForInbox(inboxAddress);   const gmail = google.gmail({ version: 'v1', auth: client });   const q = `"${testCode}"`;    try {     const res = await gmail.users.messages.list({ userId: 'me', q, maxResults: 5 });     if (!res.data.messages || res.data.messages.length === 0) return { found: false };     const msg = res.data.messages[0];     const msgDetail = await gmail.users.messages.get({ userId: 'me', id: msg.id, format: 'metadata' });     const labels = msgDetail.data.labelIds || [];     let folder = 'Inbox';     if (labels.includes('SPAM')) folder = 'Spam';     if (labels.includes('CATEGORY_PROMOTIONS')) folder = 'Promotions';     return { found: true, folder, id: msg.id };   } catch (err) {     console.error('gmail search error', err);     return { found: false };   } }
module.exports = { searchMessages };
```

--- src/services/mailClients/outlookClient.js ---

```
// Outlook (Microsoft Graph) minimal client using refresh token // This uses node-fetch to exchange refresh token for access token, then calls Microsoft Graph /messages const fetch = require('node-fetch');  const TENANT = 'common'; // or your tenant id const TOKEN_ENDPOINT = `https://login.microsoftonline.com/${TENANT}/oauth2/v2.0/token`;  // Map inbox to refresh tokens const INBOX_TOKEN_MAP = {   'test3@outlook.com': process.env.OUTLOOK_REFRESH_TOKEN_INBOX1 };  async function getAccessToken(refreshToken) {   const params = new URLSearchParams();   params.append('client_id', process.env.OUTLOOK_CLIENT_ID);   params.append('client_secret', process.env.OUTLOOK_CLIENT_SECRET);
params.append('grant_type', 'refresh_token');   params.append('refresh_token', refreshToken);
params.append('scope', 'https://graph.microsoft.com/.default offline_access openid profile');   const res = await fetch(TOKEN_ENDPOINT, { method: 'POST', body: params });   const data = await res.json();   if (!data.access_token) throw new Error('No access token from outlook token endpoint');   return data.access_token; }  async function searchMessages(inboxAddress, testCode) {   const refreshToken = INBOX_TOKEN_MAP[inboxAddress];   if (!refreshToken) return { found: false };   try {     const accessToken = await getAccessToken(refreshToken);     // Use Microsoft Graph to list messages that contain the testCode in subject/body     const query = encodeURIComponent(`contains(subject,'${testCode}') or contains(body,'${testCode}')`);     const url = `https://graph.microsoft.com/v1.0/me/messages?$search="${testCode}"`;     const res = await fetch(url, { headers: { Authorization: `Bearer ${accessToken}`, 'Prefer': 'outlook.body-content-type="text"' }     });
const json = await res.json();     if (!json.value || json.value.length === 0) return { found: false };
const msg = json.value[0];     // Determine folder - Graph returns parentFolderId; we could resolve folder name by calling /me/mailFolders/{id}     let folder = msg.parentFolderId || 'Inbox';     return { found: true, folder, id: msg.id };   } catch (err) {     console.error('outlook search error', err);     return { found: false };   } }  module.exports = { searchMessages };
```

--- README.md ---

```
# Backend - Email Deliverability Tool (Full Backend)  ## Overview This backend provides endpoints to create deliverability tests, poll configured mailboxes (Gmail and Outlook) for a unique test code, and generate a report. It uses MongoDB to store test records and nodemailer to send the final report to the user.  ## Files included - package.json - .env.example - server.js - src/app.js - src/models/Test.js - src/routes/inboxes.js - src/routes/tests.js - src/services/poller.js - src/services/mailClients/gmailClient.js - src/services/mailClients/outlookClient.js - src/utils/generateCode.js - src/utils/emailSender.js  ## Important setup steps 1. Create 5 inbox accounts you control (Gmail/Outlook) and register apps to obtain OAuth credentials. 2. For Gmail: create Google Cloud OAuth client, obtain refresh tokens for each inbox with `https://www.googleapis.com/auth/gmail.readonly` scope. 3. For Outlook: register an Azure app, obtain refresh token with `Mail.Read` scope. 4. Populate `.env` with MONGODB_URI, client ids/secrets and refresh tokens, SMTP details. 5. `npm install` then `npm run dev`  ## Notes - This starter implements polling; for production, consider push/webhook for immediate detection. - Store tokens securely (vault) in production. - The Outlook client included is minimal; you may need to adjust Graph queries depending on mailbox type (personal vs work).
```