Backend Code - Email Deliverability Tool

Generated: 2025-10-16T16:47:55.627574 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 ---

```
PORT=4000 BASE_URL=http://localhost:3000
MONGODB_URI=mongodb+srv://<user>:<pw>@cluster0.mongodb.net/emailtool?retryWrites=true&w=majority
GMAIL_CLIENT_ID=your-google-client-id GMAIL_CLIENT_SECRET=your-google-client-secret
GMAIL_REFRESH_TOKEN_INBOX1=... GMAIL_REFRESH_TOKEN_INBOX2=... OUTLOOK_CLIENT_ID=your-azure-client-id
OUTLOOK_CLIENT_SECRET=your-azure-client-secret OUTLOOK_REFRESH_TOKEN_INBOX1=... SMTP_HOST=smtp.sendgrid.net
SMTP_PORT=587 SMTP_USER=apikey SMTP_PASS=your_sendgrid_api_key FROM_EMAIL=no-reply@yourdomain.com
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(); 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');
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'}); }}); 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'}); }}); 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(); }
```

--- 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'); 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); 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(); 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 ---

```
const {google}=require('googleapis'); 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 ---

```
const fetch=require('node-fetch'); const TENANT='common'; const
TOKEN_ENDPOINT=`https://login.microsoftonline.com/${TENANT}/oauth2/v2.0/token`; const INBOX_TOKEN_MAP={
'test3@outlook.com':process.env.OUTLOOK_REFRESH_TOKEN_INBOX1 }; async function getAccessToken(refreshToken){
const params=new URLSearchParams(); params.append('client_id',process.env.OUTLOOK_CLIENT_ID);
params.append('client_secret',process.env.OUTLOOK_CLIENT_SECRET); params.append('grant_type','refresh_token');
params.append('refresh_token',refreshToken); params.append('scope','https://graph.microsoft.com/.default
offline_access openid profile'); const res=await fetch(TOKEN_ENDPOINT,{method:'POST',body:params}); const
data=await res.json(); if(!data.access_token) throw new Error('No access token from outlook token endpoint');
return data.access_token; } async function searchMessages(inboxAddress,testCode){ const
refreshToken=INBOX_TOKEN_MAP[inboxAddress]; if(!refreshToken) return {found:false}; try{ const
accessToken=await getAccessToken(refreshToken); const
url=`https://graph.microsoft.com/v1.0/me/messages?$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];
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};
```