lAdd Google Docs integration to create and maintain living scope documents for each job.
CONTEXT:
We now have thread tracking working - emails are properly grouped by conversation. When a new job is created, we need to generate a Google Doc that serves as the living scope document. As the email conversation continues, we'll update this document with synthesized information.
GOOGLE CLOUD SETUP (Already Complete):
- Google Docs API enabled
- Google Drive API enabled
- Service account: gmail-api-server@lendtautobot.iam.gserviceaccount.com
- Service account has access to Drive folder: 1_ETKqozy_yae7VjoMGdwJlsdInbCYzMY
- Service account JSON key file exists (same one used for Gmail)
REQUIREMENTS:
1. Install Dependencies:
npm install googleapis
2. Add Environment Variables:
In .env or Replit Secrets:
- GOOGLE_SERVICE_ACCOUNT_PATH=./service-account-key.json (or path to existing Gmail service account key)
- GOOGLE_DRIVE_FOLDER_ID=1_ETKqozy_yae7VjoMGdwJlsdInbCYzMY
3. Create Google Docs Service Module (server/services/google-docs.ts):
Create a service that:
- Initializes Google Docs and Drive APIs using the service account
- Creates new Google Docs in the specified folder
- Updates existing Google Docs with new content
- Formats documents with proper structure
Key functions needed:
- async createJobDocument(jobId: string, jobData: any): Promise<{docId: string, docUrl: string}>
- async updateJobDocument(docId: string, content: any): Promise<void>
- async getDocumentContent(docId: string): Promise<string>
4. Update Database Schema:
Add to jobs table in shared/schema.ts:
- google_doc_id VARCHAR(255)
- google_doc_url TEXT
Generate migration if using Drizzle migrations.
5. Update Job Creation Logic (server/routes.ts):
When creating a NEW job (not updating existing), after the job is created in PostgreSQL:

```
// Create Google Doc for new job
const docResult = await googleDocsService.createJobDocument(job.id, {
  clientEmail: job.clientEmail,
  subject: job.subject,
  location: job.location,
  scheduledDate: job.scheduledDate,
  scheduledTime: job.scheduledTime,
  jobType: job.jobType,
  techsNeeded: job.techsNeeded,
  bodyPlain: emailData["body-plain"]
});
```

```
// Store doc info in job record
await storage.updateJob(job.id, {
  google_doc_id: docResult.docId,
  google_doc_url: docResult.docUrl
});
// Include doc URL in response
return res.json({
  status: "success",
  message: "Job created with scope document",
  jobId: job.id,
  documentUrl: docResult.docUrl,
  timestamp: new Date().toISOString()
});
```

6. Document Structure:
When creating a new job document, use this structure:

```
Title: "Job Scope - [Client Name] - [Date]
==========
JOB SCOPE DOCUMENT
==========
Job ID: [job.id]
Created: [timestamp]
Last Updated: [timestamp]
Status: [job.status]
----------
CLIENT INFORMATION
----------
Company: [extracted from email or "TBD"]
Contact: [contact name from email or "TBD"]
Email: [clientEmail]
----------
CONFIRMED SCOPE
----------
Location: [location or "TBD"]
Date: [scheduledDate or "TBD"]
Time: [scheduledTime or "TBD"]
Job Type: [jobType or "TBD"]
Technicians Required: [techsNeeded or "TBD"]
Duration: [if mentioned, or "TBD"]
Specific Requirements:
• [Bullet points of key requirements extracted from emails]
• [e.g., "UT Level II certification required"]
• [e.g., "Current Chevron site clearance needed"]
----------
OPEN QUESTIONS
----------
• [What's still unclear or pending client response]
• [e.g., "Confirm exact unit number"]
• [e.g., "Need clarification on access requirements"]
```

QUOTE / PRICING

Rate: [if discussed, or "Not yet provided"]
Estimated Total: [if calculated, or "TBD"]
Payment Terms: [if discussed, or "TBD"]

ASSIGNED TECHNICIANS

Lead Tech: [name or "TBD"]
Additional Techs: [names or "TBD"]

ACTION ITEMS

Our side:
• [What you need to do]
• [e.g., "Send insurance certificates by EOD"]
Client side:
• [What client needs to provide]
• [e.g., "Provide site access badge info"]

NOTES

[Any additional context that doesn't fit above categories]
[Changes to scope, special considerations, etc.]

CONVERSATION SUMMARY

[HIGH-LEVEL summary only - NOT full transcript]
[e.g., "Initial request received Oct 7. Quote provided same day. Client confirmed Oct 8."]
Update Job Update Logic:
When updating an EXISTING job (thread matched), fetch the Google Doc and add a communication entry:

```
if (existingJob.google_doc_id) {
  // For now, just log that we would update the doc
  logger.info("Would update Google Doc", {
    docId: existingJob.google_doc_id,
    direction: direction
  });

  // Actual doc updating will be implemented in next phase with Claude synthesis
}
```

8. Error Handling:
- If Google Docs API fails, still create the job but log the error
- Set google_doc_id and google_doc_url to null if doc creation fails
- Don't let doc creation failure prevent job creation
9. Logging:
Log clearly:

- When document is created: logger.info("Google Doc created", { jobId, docId, docUrl })
- When document creation fails: logger.error("Failed to create Google Doc", { jobId, error })
- When document would be updated: logger.info("Document update triggered", { docId, direction })

10. Security:
- Service account JSON key should be in .gitignore
- Never expose service account credentials in logs or API responses
- Only return publicly shareable doc URLs to users

TESTING CHECKLIST:
1. Send new job email → Check that Google Doc is created in Drive folder
2. Verify doc URL is stored in PostgreSQL jobs table
3. Click doc URL → Verify document opens and contains job details
4. Send reply email → Verify Core recognizes it as update (doc update logging appears)
5. Check that doc has proper sharing permissions (anyone with link can view)

CRITICAL NOTES:
- Use the SAME service account key that's working for Gmail API
- Documents should be created in folder ID: 1_ETKqozy_yae7VjoMGdwJlsdInbCYzMY
- Set document permissions to "anyone with link can view" so owner can access easily
- Document formatting should be clean and readable (use proper spacing and sections)
- For now, we're just creating and storing docs - AI synthesis comes in next phase

FUTURE PHASES (Not implemented yet):
- Phase 2: Claude reads all job communications and synthesizes scope updates
- Phase 3: Auto-update doc as conversation evolves
- Phase 4: Generate tech assignment recommendations in doc