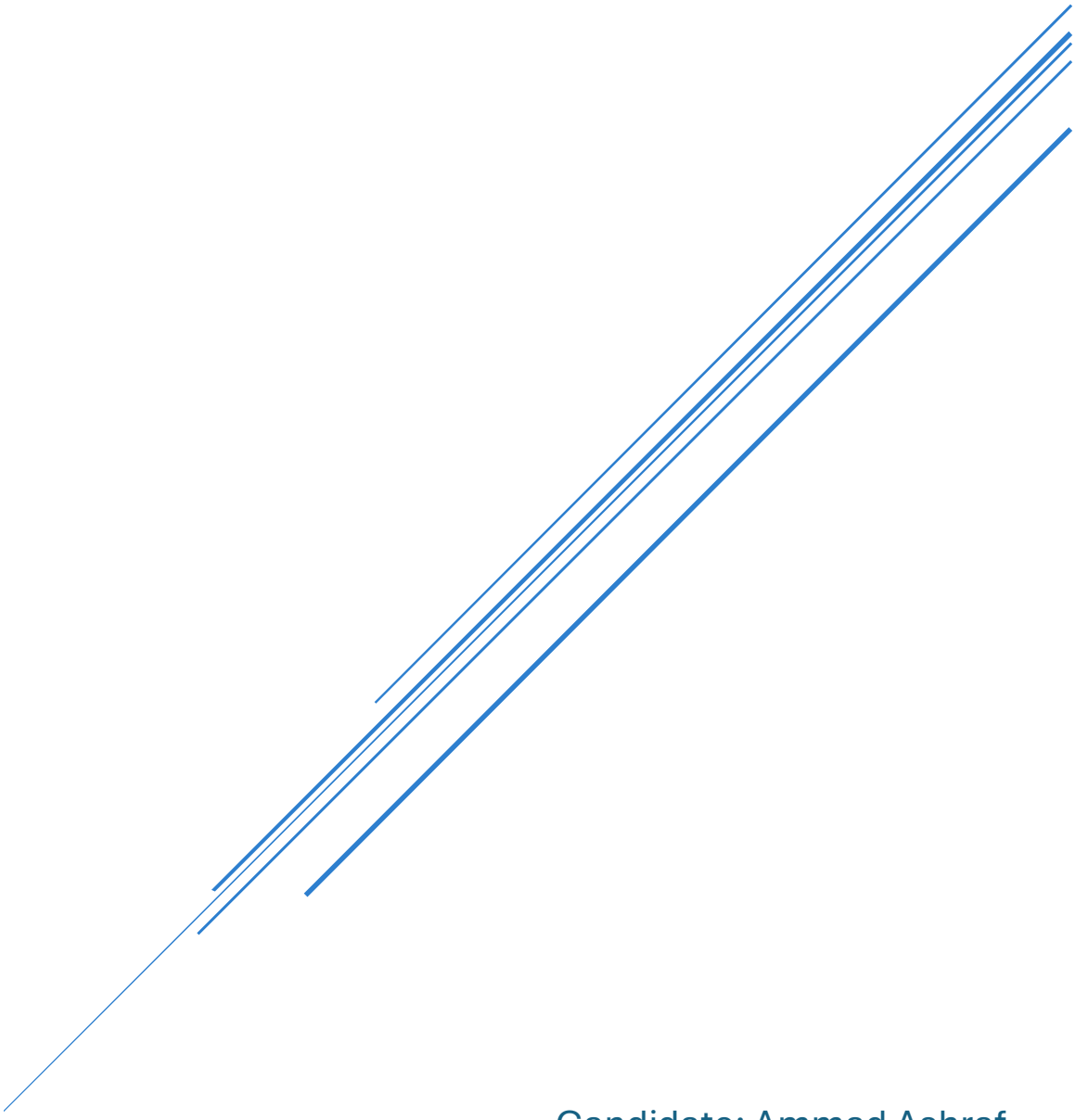
TECHNICAL REPORT

Google Tasks to Notion Sync



Candidate: Ammad Ashraf

Stack: Node.js (Express), Notion API, Google Tasks

Table of Contents

Abstract: 3
1. Learning & Setup Process 4
1.1. Google Tasks API Understanding 4
1.2. Notion API Integration 4
1.3. Sync Logic Development 6
2. Architecture Overview 6
3. Detailed Code Implementation 6
3.1. Google Tasks Integration (google.js) 6
Authentication Flow: 7
Core Functions: 7
Data Structure: 10
3.2. Notion Database Integration (notion.js) 11
Setup and Configuration: 11
Status Mapping: 11
Core Function: 11
3.3. Synchronization Logic (sync.js) 13
Duplicate Prevention System: 13
Core Functions: 14
Sync Process Flow: 15
Error Handling: 15
3.4. Server and Automation (index.js) 15
Manual Sync Endpoint: 15
Automated Sync with Cron: 16
4. Testing & Validation 16
4.1. Manual Testing 16
4.2. Automated Testing 17
Configuration Files for Reference: 17
Credentials.json 17

Token.json 17

. env 18

Abstract:

The system consists of four main code files:

google.js handles Google Tasks API authentication and data fetching.

notion.js manages inserting tasks into Notion database.

sync.js contains the core logic that fetches from Google and pushes to Notion while preventing duplicates.

server.js runs an Express server with scheduled sync every 5 minutes plus a manual sync endpoint.

The application requires OAuth2 credentials from Google Cloud Console stored in ***credentials. json***, a Notion integration token and database ID in ***.env*** file and uses ***synced_tasks. json*** to track already processed tasks.

When running, it authenticates with Google using OAuth2 flow, fetches all tasks from task lists, filters out completed and already-synced ones, then creates corresponding entries in Notion with title, description, status, and due date fields.

GitHub Repo

Demo Video

1. Learning & Setup Process

I started by breaking down the task into small learning modules before jumping into code. Here's the sequence I followed:

1.1. Google Tasks API Understanding

- Created a project in Google Cloud Console
- Enabled the Google Tasks API and set up OAuth2 credentials
- Understood how OAuth2 flow works for installed apps:
    - Redirect to consent screen > Get auth code > Exchange for access/refresh tokens > Store tokens for future use
- Used Google's official Node.js SDK to fetch tasks from all task lists
- Extracted key task data: title, status, due, and later added description

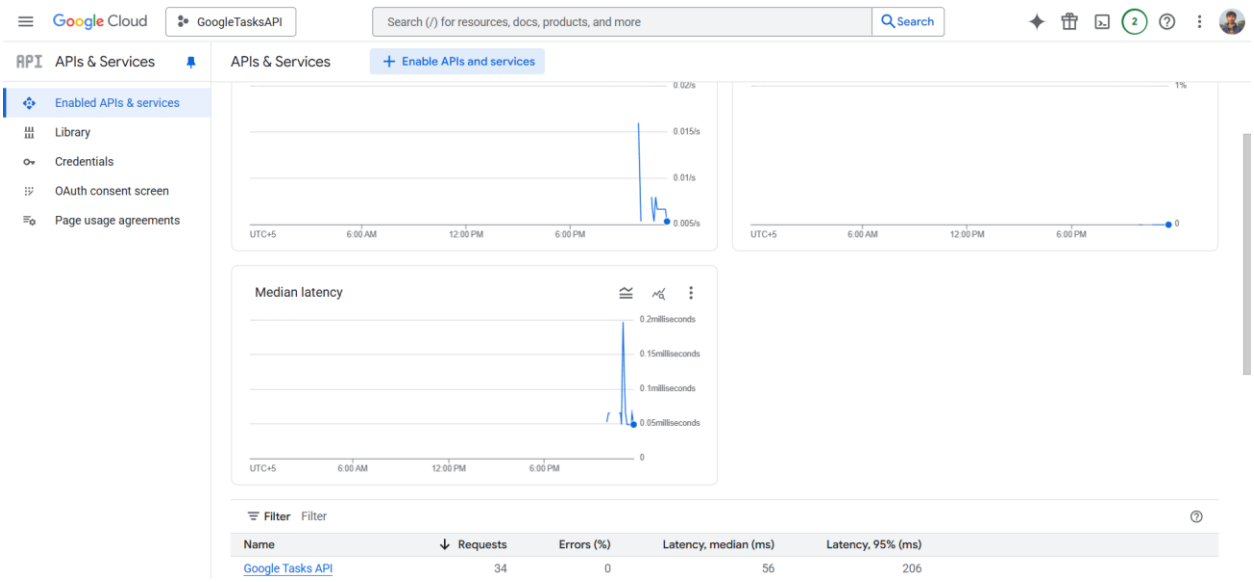


Figure 1| *My Google Cloud Console project with the Google Tasks API enabled.*

1.2. Notion API Integration

- Created a Notion integration, gave it access to a database, and got the internal integration token
- Learned the structure of a Notion database:
    - Properties like Task (Title), Status (Select), Due (Date), and Description (Text)
- Mapped incoming Google Tasks into a format Notion API understands

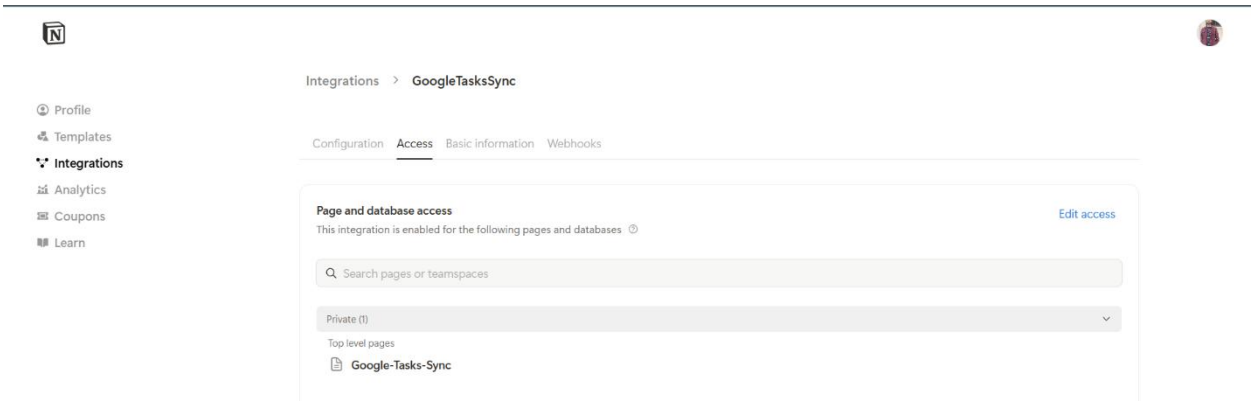


Figure 2 | *Enabled integration with access to my parent page, Google-Tasks-Sync.*

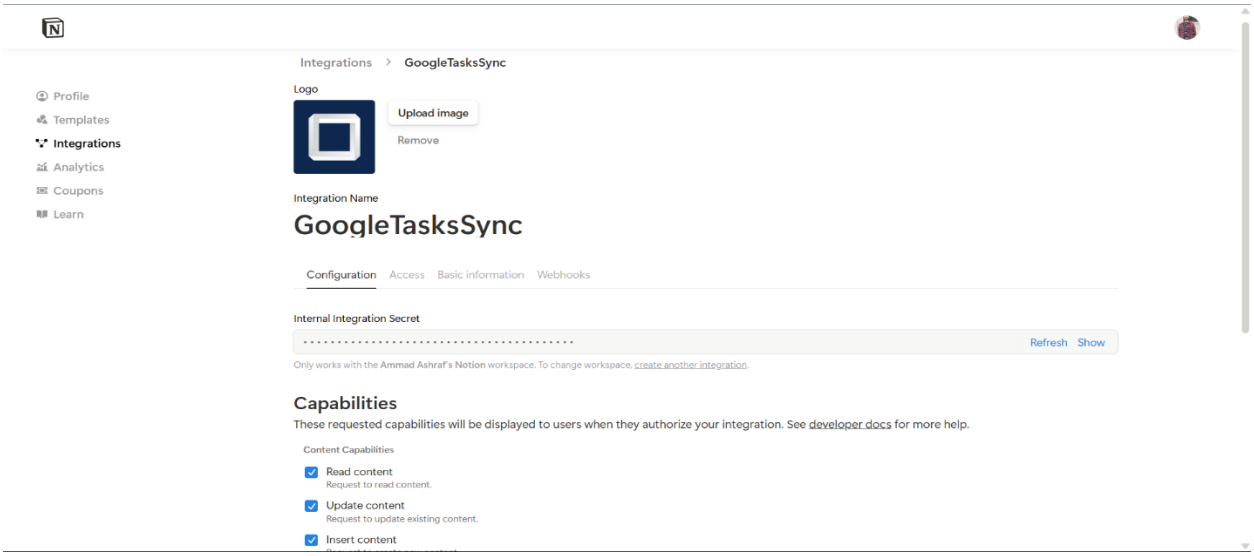


Figure 3 | *Configured a new Notion integration to access the Notion API and OAuth.*

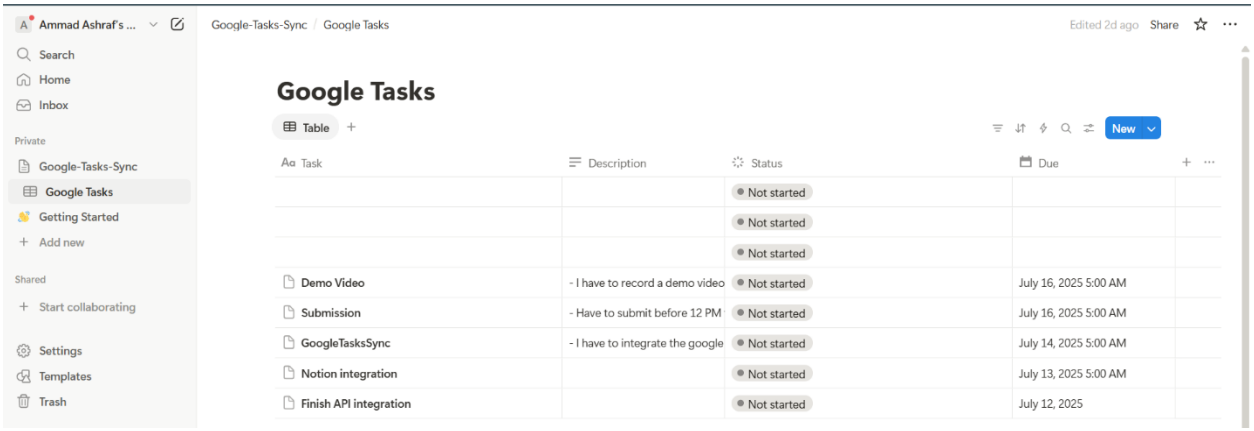


Figure 4 | *Created a new parent page in Notion named Google Tasks, with properties: Task, Description, Status, and Due.*

1.3. Sync Logic Development

- Built a sync.js module that:
    - Fetches all tasks from Google
    - Filters out completed tasks and already-synced ones using a local synced_tasks.json file
    - Inserts new pending tasks into Notion using their API
- Added optional logic to prevent duplication and avoid empty titles or missing statuses

2. Architecture Overview

File	Responsibility
google.js	Handles Google OAuth2 flow and task fetching logic
notion.js	Manages task insertion into Notion database
sync.js	Core sync engine (fetch > filter > insert)
index.js	Express API + Scheduled auto-sync via node-cron
. env	Stores NOTION_TOKEN & NOTION_DATABASE_ID
synced_tasks. json	Local cache of synced Google Task IDs
Credentials. json	Stores OAuth2 credentials downloaded from Google Cloud Console
Token. json	Stores access & refresh token

3. Detailed Code Implementation

3.1. Google Tasks Integration (google.js)

This module handles the complete Google Tasks API integration with OAuth2 authentication flow.

Authentication Flow:

The authentication follows these steps:

1. App starts > Check for token. Json
2. If exists > Load token, authenticate
3. If not exists > Generate auth URL
4. Open browser > User grants permissions
5. User gets code > Paste into terminal
6. Exchange code > Get access token
7. Save token. Json > Future runs skip steps 3-6

Core Functions:

authorize (): Main authentication orchestrator that handles both existing and new token flows.

```
/**
 * Step 1: Authorize and return a ready-to-use authenticated Google client.
 */
async function authorize() {
  const credentials = JSON.parse(fs.readFileSync(CREDENTIALS_PATH));
  const { client_id, client_secret, redirect_uris } = credentials.installed;

  const oAuth2Client = new google.auth.OAuth2(
    client_id,
    client_secret,
    redirect_uris[0]
  );

  // Step 2: Check if token.json exists (already authorized)
  if (fs.existsSync(TOKEN_PATH)) {
    const token = JSON.parse(fs.readFileSync(TOKEN_PATH));
    oAuth2Client.setCredentials(token);
    return oAuth2Client;
  }

  // Step 3-6: If no token, start full OAuth2 flow
  return await getNewToken(oAuth2Client);
```

```
}
```

getNewToken (): Manages the complete OAuth2 flow including browser opening and code input

```
/**
 * Step 3: Redirect user to Google Consent Page
 * Step 4: User logs in and grants access
 * Step 5-6: Exchange code for token, save token.json
 */
function getNewToken(oAuth2Client) {
  return new Promise((resolve, reject) => {
    const authUrl = oAuth2Client.generateAuthUrl({
      access_type: 'offline',
      scope: SCOPES,
    });

    console.log('\n-- Open this URL to authorize access:\n' +
authUrl);

    // Try to auto-open browser using default command
    exec(`start "" "${authUrl}"`);

    // Step 5: Ask user to paste the code
    const rl = readline.createInterface({
      input: process.stdin,
      output: process.stdout,
    });

    rl.question('\n-- Paste the code from the browser here: ', (code)
=> {
      rl.close();

      // Step 6: Exchange code for token
      oAuth2Client.getToken(code, (err, token) => {
        if (err) return reject('-- Error retrieving access token: ' +
err);
        oAuth2Client.setCredentials(token);
```

```
        // Step 6: Save token.json for future runs
        fs.writeFileSync(TOKEN_PATH, JSON.stringify(token));
        console.log('-- Access token saved to', TOKEN_PATH);

        resolve(oAuth2Client);
      });
    });
  });
}
```

getTaskLists (): Retrieves all task lists from user's Google Tasks account

```
/**
 * Step 7: Get all Task Lists (like "My Tasks", "Work", etc.)
 */
async function getTaskLists(auth) {
  const service = google.tasks({ version: 'v1', auth });
  const res = await service.tasklists.list();
  const taskLists = res.data.items || [];

  return taskLists.map((list) => ({
    id: list.id,
    title: list.title,
  }));
}
```

getTasksFromList (): Fetches all tasks from a specific task list

```
/**
 * Step 9: Get all tasks from all task lists
 */
async function getGoogleTasks() {
  const auth = await authorize();
  const taskLists = await getTaskLists(auth);

  let allTasks = [];

  for (const list of taskLists) {
    const tasks = await getTasksFromList(auth, list.id);
```

```
    const tasksWithList = tasks.map((task) => ({
      ...task,
      listTitle: list.title,
    }));

    allTasks = allTasks.concat(tasksWithList);
  }

  return allTasks;
}
```

getGoogleTasks (): Main function that combines all task lists and returns unified task array

```
/**
 * Step 9: Get all tasks from all task lists
 */
async function getGoogleTasks() {
  const auth = await authorize();
  const taskLists = await getTaskLists(auth);

  let allTasks = [];

  for (const list of taskLists) {
    const tasks = await getTasksFromList(auth, list.id);
    const tasksWithList = tasks.map((task) => ({
      ...task,
      listTitle: list.title,
    }));

    allTasks = allTasks.concat(tasksWithList);
  }

  return allTasks;
}
```

Data Structure: *Each task object contains:*

- id: Unique Google Task identifier
- title: Task name
- description: Task notes/details

- status: Current status ('needsAction', 'completed')
- due: Due date in ISO format

3.2. Notion Database Integration (notion.js)

This module handles inserting Google Tasks data into a Notion database using the official Notion SDK.

Setup and Configuration:

```
// STEP 1: Initialize Notion Client with integration token
const notion = new Client({ auth: process.env.NOTION_TOKEN });

// STEP 2: Get database ID from .env
const databaseId = process.env.NOTION_DATABASE_ID;
```

Status Mapping:

```
  const statusMap = {
    needsAction: 'Not Started',
    completed: 'Completed',
  };
```

This maps Google Tasks status values to Notion database status options. The mapping is customizable based on Notion database schema.

Core Function:

insertTaskToNotion ()

This function transforms a Google Task object into Notion's API format:

1. **Title Property**: Creates the main page title in Notion
2. **Description Property**: Adds rich text content (only if description exists)
3. **Status Property**: Maps Google status to Notion status options
4. **Due Property**: Converts ISO date to Notion date format

Error Handling:

- Graceful error handling with detailed logging
- Continues processing other tasks if one fails
- Returns success/failure status for tracking

Notion API Request Structure: The function builds a properly formatted Notion API request with:

- Parent database reference
- Property mappings for each database column
- Conditional property inclusion (only adds properties if data exists)

```
async function insertTaskToNotion(task) {
  const { title, description,status, due } = task;

  // Map Google Task status to Notion
  const statusMap = {
    needsAction: 'Not Started',
    completed: 'Completed',
  };

  try {
    // STEP 4: Send the request to Notion API
    const response = await notion.pages.create({
      parent: { database_id: databaseId },
      properties: {
        Task: {
          title: [
            {
              text: {
                content: title || 'Untitled Task',
              },
            },
          ],
        },
         Description: description
          ? {
              rich_text: [
                {
                  text: {
                    content: description,
```

```
                  },
                },
              ],
            }
          : undefined,
        Status: {
          status: {
            name: statusMap[status] || 'Not Started', // Fallback if
unknown
          },
        },
        Due: due
          ? {
              date: {
                start: due,
              },
            }
          : undefined,
      },
    });

    console.log(`-- Task "${title}" added to Notion.`);
    return response;
  } catch (error) {
    console.error(`-- Failed to insert task "${title}"`, error.body ||
error);
  }
}
```

3.3. Synchronization Logic (sync.js)

This is the core orchestrator that connects Google Tasks and Notion, handling the complete sync process with duplicate prevention.

Duplicate Prevention System:

```
// Path to local file to track synced task IDs
const SYNCED_TASKS_FILE = path.join(__dirname,
'../synced_tasks.json');
```

Uses a local JSON file to track which Google Tasks have already been synced to Notion, preventing duplicate entries.

Core Functions:

1. **loadSyncedTaskIds ()**:
    - Reads previously synced task IDs from JSON file
    - Returns a JavaScript Set for fast O (1) lookup
    - Handles missing file gracefully (returns empty Set)

```
// STEP 1: Read previously synced task IDs
function loadSyncedTaskIds() {
  try {
    const data = fs.readFileSync(SYNCED_TASKS_FILE, 'utf8');
    return new Set(JSON.parse(data));
  } catch (err) {
    return new Set(); // If file doesn't exist, start fresh
  }
}
```

2. **saveSyncedTaskIds ()**:
    - Converts Set to Array for JSON serialization
    - Saves updated list of synced task IDs
    - Uses pretty formatting for readability

```
// STEP 2: Save updated synced task IDs
function saveSyncedTaskIds(taskIdSet) {
  const data = JSON.stringify([...taskIdSet], null, 2);
  fs.writeFileSync(SYNCED_TASKS_FILE, data, 'utf8');
}
```

3. **syncGoogleTasksToNotion ()**: Main sync function that:

- Loads existing synced task IDs
- Fetches all Google Tasks
- Filters tasks based on criteria:
  - Not completed
  - Has a title
  - Not already synced
- Process each task sequentially
- Updates tracking file with new synced IDs

Sync Process Flow:

1. Initialize tracking (load synced IDs)
2. Fetch Google Tasks
3. Filter new/pending tasks
4. Insert each task into Notion
5. Update tracking file
6. Handle errors gracefully

Error Handling:

- Individual task failures don't stop the entire process
- Detailed error logging for debugging
- Tracking file updated even if some tasks fail

3.4. Server and Automation (index.js)

This module creates a web server that provides both manual and automated synchronization capabilities.

Manual Sync Endpoint:

```
app.get('/sync', async (req, res) => {
  try {
    await syncGoogleTasksToNotion(); // Run sync logic
```

```
    res.status(200).send('-- Sync triggered successfully.');
  } catch (error) {
    console.error('-- Sync failed:', error.message);
    res.status(500).send('-- Sync failed. Check server logs.');
  }
});
```

Provides an HTTP endpoint (/sync) that allows manual trigger of synchronization:

- Returns HTTP 200 with success message on completion
- Returns HTTP 500 with error message on failure
- Logs detailed errors for debugging

Automated Sync with Cron:

```
cron.schedule('*/5 * * * *', async () => {
  console.log('-- Running scheduled sync...');
  try {
    await syncGoogleTasksToNotion();
    console.log('-- Scheduled sync complete.');
  } catch (error) {
    console.error('-- Scheduled sync failed:', error.message);
  }
});
```

Implements automatic synchronization every 5 minutes using node-cron:

- Runs in background without user intervention
- Provides console logging for monitoring
- Handles errors gracefully without crashing server

4. Testing & Validation

4.1. Manual Testing

```
# Start server
node index.js
```

```
# Test manual sync
curl http://localhost:3000/sync
# or visit: http://localhost:3000/sync
```

4.2. Automated Testing

- **Cron Job**: Verified automatic sync every 5 minutes
- **Duplicate Prevention**: Tested with repeated syncs
- **Error Handling**: Tested with invalid tokens and network issues
- **Data Integrity**: Verified all task properties sync correctly

Configuration Files for Reference:

Credentials.json


```
{"installed”:
     {"client_id":"6580812886242bk2glfld4dgvpijb1d2g0ifbaf125t0.apps.g
oogleusercontent.com",
     "project_id":"generated-media-465807-q9",
     "auth_uri":"https://accounts.google.com/o/oauth2/auth",
     "token_uri":"https://oauth2.googleapis.com/token",
     "auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/
v1/certs",
     "client_secret":"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",
     "redirect_uris":["http://localhost"]
     }
}
```


Token.json


```
{"access_token":"ya29.a0AS3H6Nyq8no_7eaypbTnWgI_9bDyxGtaGCcOOfSOA1j9QX
xbc0BYsNmFTh0ITYhddsXpSXs0n663ZGahotT2zz7IIOf2D2AFo_jCyJfMOo0Tuz3QIwNc
1CnkuMPfKAAEM0Yd65MvyeUK96oIWwY4i0hvCrYLkxp6OTTB-
Jn3aCgYKAQ4SARUSFQHGX2Mi6D_p92bQ_NCGQREsWDbTJg0175",
```

```
"refresh_token":"1//0gf8COOcU6VyyCgYIARAAGBASNwF-
L9IrkCEiL4Bn3ZMgbqUq0Ezwi0D-
9W4qqFXKw_ZtfBW7EtrSsx7gREDAZw8Szj979iSNTGI",
"scope":"https://www.googleapis.com/auth/tasks.readonly",
"token_type":"Bearer",
"refresh_token_expires_in":604799,
"expiry_date":1752414182831}
```


.env


```
NOTION_TOKEN=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
NOTION_DATABASE_ID=22fbe8510f5880f999c9d47626f6f1f9
```


Thank you!