Google API Integration Research for Dynamic Scheduler Agent

This document provides comprehensive information on how to programmatically interact with Google Calendar, Google Tasks, and Gmail APIs for building a Dynamic Scheduler Agent.

Table of Contents

Authentication Requirements and Methods

All Google APIs use OAuth 2.0 for authentication and authorization. Here's how to set up authentication for your Dynamic Scheduler Agent:

OAuth 2.0 Authentication

1. **Create a Google Cloud Project**:
    - Go to the Google Cloud Console
    - Create a new project or select an existing one
    - Enable the Google Calendar API, Google Tasks API, and Gmail API
2. **Configure OAuth Consent Screen**:
    - In the Google Cloud Console, navigate to "APIs & Services" > "OAuth consent screen"
    - Select the appropriate user type (Internal or External)
    - Fill in required app information (name, user support email, etc.)
    - Add the necessary scopes for each API (see below)
    - Add test users if using External user type
3. **Create OAuth 2.0 Credentials**:
    - In the Google Cloud Console, navigate to "APIs & Services" > "Credentials"
    - Click "Create Credentials" and select "OAuth client ID"

- Choose the appropriate application type (Web application, Desktop app, etc.)
- Configure the redirect URIs where your application will receive the authorization code

Required Scopes

For the Dynamic Scheduler Agent, you'll need the following scopes:

Google Calendar API: - `https://www.googleapis.com/auth/calendar` - Full access to manage calendars - `https://www.googleapis.com/auth/calendar.events` - Manage events only - `https://www.googleapis.com/auth/calendar.readonly` - Read-only access (if you only need to view events)

Google Tasks API: - `https://www.googleapis.com/auth/tasks` - Full access to manage tasks - `https://www.googleapis.com/auth/tasks.readonly` - Read-only access

Gmail API: - `https://www.googleapis.com/auth/gmail.send` - Send emails only - `https://www.googleapis.com/auth/gmail.compose` - Create and send emails - `https://www.googleapis.com/auth/gmail.readonly` - Read-only access to emails

Service Accounts vs. User Authentication

For a Dynamic Scheduler Agent, you have two main authentication options:

1. **User Authentication (Recommended for personal schedulers)**:
   - The user grants permission to your application
   - The application acts on behalf of the user
   - Requires user interaction for initial authorization
   - Tokens can be refreshed without user interaction
2. **Service Account (For organizational deployments)**:
   - Acts as its own identity
   - Can access user data through domain-wide delegation in Google Workspace domains
   - No user interaction required
   - Not suitable for standard Gmail users (only works with Google Workspace)

Retrieving Data

Retrieving Calendar Events

The Google Calendar API allows you to retrieve events from a user's calendar using the `events.list` method.

Key Parameters: - `calendarId`: The ID of the calendar to retrieve events from (use 'primary' for the user's primary calendar) - `timeMin`: The start time of the interval to retrieve events (as an RFC3339 timestamp) - `timeMax`: The end time of the interval - `maxResults`: Maximum number of events to return - `singleEvents`: Whether to expand recurring events into instances - `orderBy`: Order of the events returned ("startTime" is recommended)

Retrieving Tasks

The Google Tasks API allows you to retrieve tasks from a user's task lists using the `tasks.list` method.

Key Parameters: - `tasklist`: The ID of the task list to retrieve tasks from - `maxResults`: Maximum number of tasks to return - `dueMin`: Lower bound for a task's due date (as an RFC3339 timestamp) - `dueMax`: Upper bound for a task's due date - `showCompleted`: Whether to include completed tasks - `showDeleted`: Whether to include deleted tasks - `showHidden`: Whether to include hidden tasks

Retrieving Emails

The Gmail API allows you to retrieve emails from a user's inbox using the `messages.list` method.

Key Parameters: - `userId`: The user's email address (use 'me' for the authenticated user) - `q`: Query for filtering messages (similar to Gmail search syntax) - `maxResults`: Maximum number of messages to return - `labelIds`: Only return messages with these labels

Creating, Updating, and Deleting Calendar Events

Creating Calendar Events

To create a calendar event, use the `events.insert` method with an Event resource.

Required Fields: - `summary`: Title of the event - `start`: Start time (dateTime or date) - `end`: End time (dateTime or date)

Optional Fields: - `location`: Where the event takes place - `description`: Description of the event - `attendees`: List of attendees (email addresses) - `reminders`: Notification settings - `recurrence`: For recurring events

Updating Calendar Events

To update an existing event, use the `events.update` or `events.patch` method.

- `events.update`: Requires the full event resource
- `events.patch`: Allows partial updates (only include fields to be changed)

Deleting Calendar Events

To delete an event, use the `events.delete` method with the event ID and calendar ID.

Sending Emails Programmatically

The Gmail API allows you to send emails programmatically using the `messages.send` method.

Process Overview

1. Create a MIME message (including headers, body, and any attachments)
2. Encode the MIME message as a base64url string
3. Create a Message resource with the encoded string in the `raw` property
4. Call `messages.send` to send the email

Alternative Approach

You can also create a draft first using `drafts.create` and then send it using `drafts.send`.

Rate Limits and Restrictions

Understanding rate limits is crucial for a reliable Dynamic Scheduler Agent. Here are the limits for each API:

Google Calendar API Quotas

- **Per minute per project**: Limits the number of requests made by your Google Cloud project
- **Per minute per project per user**: Limits the number of requests made by any one user in your project

If either quota is exceeded, you'll receive a `403 usageLimits` or `429 usageLimits` status code.

Best practices to avoid hitting limits: - Use exponential backoff for retries - Randomize traffic patterns - Use push notifications instead of polling - Properly account for service account usage with the `quotaUser` parameter

Google Tasks API Quotas

- **Default courtesy limit**: 50,000 queries per day
- Service account calls are treated as originating from a single account

Gmail API Quotas

- **Daily sending limit**: Varies by account type (typically 2,000 messages per day for regular Gmail accounts)
- **Rate limits**: Not explicitly documented, but generally follows Google API standard practices

Requesting Quota Increases

For all APIs, you can request quota increases through the Google Cloud Console: 1. Ensure you have a billing account for your project 2. Visit the "Quotas" page in the API Console 3. Select the API and quota you want to increase 4. Submit a request for an increase

Code Examples

Authentication Code Examples

Python OAuth 2.0 Authentication

```
import os
import pickle
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request

# Define the scopes your application needs
SCOPES = [
    'https://www.googleapis.com/auth/calendar',
    'https://www.googleapis.com/auth/tasks',
    'https://www.googleapis.com/auth/gmail.send'
]

def get_credentials():
    """Get and refresh user credentials from OAuth 2.0 flow."""
    creds = None
    # The file token.pickle stores the user's access and refresh tokens
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)

    # If there are no valid credentials, let the user log in
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)

        # Save the credentials for the next run
        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)

    return creds
```

Calendar Operations Code Examples

Retrieving Calendar Events (Python)

```
from googleapiclient.discovery import build
from datetime import datetime, timedelta
import pytz

def get_upcoming_events(credentials, days=7, max_results=10):
    """Retrieve upcoming calendar events."""
    # Build the service
    service = build('calendar', 'v3', credentials=credentials)

    # Calculate time boundaries
    now = datetime.utcnow().replace(tzinfo=pytz.UTC)
    time_min = now.isoformat()
    time_max = (now + timedelta(days=days)).isoformat()

    # Call the Calendar API
    events_result = service.events().list(
        calendarId='primary',
        timeMin=time_min,
        timeMax=time_max,
        maxResults=max_results,
        singleEvents=True,
        orderBy='startTime'
    ).execute()

    events = events_result.get('items', [])
    return events
```

Creating a Calendar Event (Python)

```
def create_calendar_event(credentials, summary, location, description, start_time, end_time
    """Create a new calendar event."""
    service = build('calendar', 'v3', credentials=credentials)

    # Create event body
    event = {
        'summary': summary,
        'location': location,
        'description': description,
        'start': {
            'dateTime': start_time,
            'timeZone': 'America/Los_Angeles',
        },
        'end': {
            'dateTime': end_time,
            'timeZone': 'America/Los_Angeles',
        },
        'reminders': {
            'useDefault': False,
            'overrides': [
                {'method': 'email', 'minutes': 24 * 60},
                {'method': 'popup', 'minutes': 30},
            ],
        },
    }

    # Add attendees if provided
    if attendees:
        event['attendees'] = [{'email': email} for email in attendees]

    # Call the Calendar API
    event = service.events().insert(calendarId='primary', body=event).execute()
    return event
```

Tasks Operations Code Examples

Retrieving Tasks (Python)

```
def get_tasks(credentials, tasklist_id='@default', max_results=100):
    """Retrieve tasks from a specified task list."""
    service = build('tasks', 'v1', credentials=credentials)

    # Call the Tasks API
    results = service.tasks().list(
        tasklist=tasklist_id,
        maxResults=max_results,
        showCompleted=True
    ).execute()

    tasks = results.get('items', [])
    return tasks
```

Creating a Task (Python)

```
def create_task(credentials, title, notes=None, due=None, tasklist_id='@default'):
    """Create a new task in the specified task list."""
    service = build('tasks', 'v1', credentials=credentials)

    # Create task body
    task = {
        'title': title
    }

    if notes:
        task['notes'] = notes

    if due:
        task['due'] = due  # RFC 3339 timestamp format

    # Call the Tasks API
    result = service.tasks().insert(tasklist=tasklist_id, body=task).execute()
    return result
```

Gmail Operations Code Examples

Sending an Email (Python)

```
import base64
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

def send_email(credentials, to, subject, message_text, from_email=None):
    """Send an email using the Gmail API."""
    service = build('gmail', 'v1', credentials=credentials)

    # Create a MIME message
    message = MIMEMultipart()
    message['to'] = to
    message['subject'] = subject

    if from_email:
        message['from'] = from_email

    # Add text part
    msg = MIMEText(message_text)
    message.attach(msg)

    # Encode the message
    raw_message = base64.urlsafe_b64encode(message.as_bytes()).decode()
    # Create the message object
    message_object = {
        'raw': raw_message
    }

    # Send the message
    sent_message = service.users().messages().send(
        userId='me',
        body=message_object
    ).execute()

    return sent_message
```

Conclusion

This document provides a comprehensive overview of how to programmatically interact with Google Calendar, Google Tasks, and Gmail APIs for building a Dynamic Scheduler Agent. By following the authentication methods, understanding the data retrieval and manipulation capabilities, and respecting the rate limits, you can create a robust scheduling system that maximizes high-leverage outputs while protecting time for deep thinking, leadership, and personal wellbeing.

References

1. Google Calendar API Documentation
2. Google Tasks API Documentation
3. Gmail API Documentation
4. Google OAuth 2.0 Documentation
5. Google API Client Libraries