Gmail Attachment Exporter Implementation Plan

This implementation plan provides a step-by-step guide for building a tool that exports Gmail attachments to either Google Drive or an AWS S3 bucket.

1. Setup Development Environment

Prerequisites

1. **Python Environment**
   - Install Python 3.9 or later
   - Set up a virtual environment:

     ```
     bash python -m venv gmail-exporter-envsource gmail-exporter-env/bin/activate # Linux/macOS# orgmail-exporter-env\Scripts\activate # Windows
     ```

2. **Required Python Libraries**
   - Install required libraries:

     ```
     bash pip install google-api-python-client google-auth-httplib2 google-auth-oauthlib boto3 flask flask-restful pyjwt
     ```

2. Google Cloud Project Setup

1. **Create a Google Cloud Project**
   - Go to Google Cloud Console
   - Click "New Project" and provide a project name (e.g., "Gmail Attachment Exporter")
   - Click "Create"
   - **Enable Required APIs**
   - In the project dashboard, go to "API & Services" > "Library"
   - Search for and enable the following APIs:
     - Gmail API
     - Google Drive API
   - **Set Up OAuth 2.0 Credentials**
   - Go to "API & Services" > "Credentials"
   - Click "Create Credentials" > "OAuth client ID"
   - Configure the OAuth consent screen:
     - User Type: External (or Internal if using a Google Workspace account)
     - App name: "Gmail Attachment Exporter"
     - User support email: Your email address
     - Developer contact information: Your email address
   - For "Application type," select "Web application" (for API service)
   - Add authorized redirect URIs (e.g., "http://localhost:5000/auth/callback")
   - Give it a name (e.g., "Gmail Attachment Exporter API")
   - Click "Create"
   - Download the credentials JSON file and save it as `credentials.json` in your project directory

3. AWS Configuration (for S3 option)

1. **AWS Account Setup**
   - If you don't have an AWS account, create one at AWS Console
   - **Create IAM User for S3 Access**
   - Go to the AWS Management Console

- Navigate to IAM service
- Create a new user with programmatic access
- Attach the "AmazonS3FullAccess" policy (or create a custom policy with minimal permissions)
- Save the access key ID and secret access key

- **Configure AWS Credentials**

- Install AWS CLI: `pip install awscli`

- Configure credentials:

  ```
  bash aws configure
  ```

- Enter your access key ID and secret access key when prompted

- Set default region (e.g., `us-east-1`)

- **Create an S3 Bucket** (Optional - can also be created by the tool)

- Using the AWS CLI:

  ```
  bash aws s3 mb s3://your-bucket-name
  ```

- Or via the AWS Console: S3 service > Create bucket

4. Create Project Structure

Create the following project structure:

```
gmail-attachment-exporter/ â”œâ”€â”€ app/ â”‚ â”œâ”€â”€ __init__.py # Initialize Flask app â”‚ â”œâ”€â”€ api/
# API routes â”‚ â”‚ â”œâ”€â”€ __init__.py â”‚ â”‚ â”œâ”€â”€ auth.py # Authentication routes â”‚ â”‚ â”œâ”€â”€
gmail.py # Gmail operations â”‚ â”‚ â”œâ”€â”€ drive.py # Google Drive operations â”‚ â”‚ â”‚â”€â”€ s3.py # S3
operations â”‚ â”œâ”€â”€ services/ # Business logic â”‚ â”‚ â”œâ”€â”€ __init__.py â”‚ â”‚ â”œâ”€â”€
auth_service.py # Authentication service â”‚ â”‚ â”œâ”€â”€ gmail_service.py # Gmail service â”‚ â”‚ â”œâ”€â”€
drive_service.py # Drive service â”‚ â”‚ â”œâ”€â”€ s3_service.py # S3 service â”‚ â”œâ”€â”€ utils/ # Utility
functions â”‚ â”‚ â”œâ”€â”€ __init__.py â”‚ â”‚ â”‚â”€â”€ helpers.py # Helper functions â”‚ â”‚â”€â”€
config.py # Configuration â”œâ”€â”€ credentials.json # Google API credentials â”œâ”€â”€ run.py # Application
entry point â”œâ”€â”€ requirements.txt # Dependencies â”‚â”€â”€ README.md # Documentation
```

5. Write the Core Application Code

1. **Authentication Module**

   - Create functions to handle Google API authentication
   - Implement token management (create, refresh, store)
   - Set up proper scopes for Gmail and Drive APIs

   - **Gmail Module**

   - Implement functions to search for emails with attachments

   - Extract attachment metadata (filename, size, type)
   - Download attachments to a temporary location

   - **Google Drive Module**

   - Implement functions to upload files to Google Drive

   - Create or find destination folders
   - Handle file naming conflicts

   - **S3 Module**

   - Implement functions to upload files to S3

   - Create buckets if needed
   - Handle file organization with prefixes

   - **Main Application Logic**

   - Set up Flask application

- Implement API endpoints
- Handle errors and provide logging

6. Implement Main Application Features

1. **Search for Emails with Attachments**

   - By date range
   - By sender
   - By subject
   - Custom queries

   - **Extract and Download Attachments**

   - Handle different attachment types

   - Handle large attachments
   - Manage filename conflicts

   - **Upload to Destination**

   - Support both Google Drive and S3

   - Maintain folder/path structure
   - Provide progress updates

7. API Service Implementation

Authentication and Security

1. **OAuth2 Flow**

   - Implement OAuth2 authorization flow for Gmail and Google Drive
   - Create routes for initiating authorization and handling callbacks
   - Store and manage refresh tokens securely

   - **API Authentication**

   - Implement JWT-based authentication for API clients

   - Create middleware for protecting API routes
   - Set up rate limiting to prevent abuse

   - **Security Headers**

   - Implement secure headers (CORS, Content-Security-Policy, etc.)

   - Set up HTTPS for secure communication

API Routes and Endpoints

1. **Authentication Routes**

   - `/api/auth/login` - Initiate Google OAuth flow
   - `/api/auth/callback` - Handle OAuth callback
   - `/api/auth/refresh` - Refresh access tokens
   - `/api/auth/revoke` - Revoke user access

   - **Gmail Routes**

   - `/api/gmail/search` - Search for emails with attachments

     - Query parameters:
       - `q` - Gmail search query
       - `max_results` - Maximum number of results
       - `start_date` - Start date for search
       - `end_date` - End date for search
   - `/api/gmail/messages/{message_id}` - Get details of a specific email

- `/api/gmail/attachments/{message_id}` - List attachments for an email
- `/api/gmail/attachments/{message_id}/{attachment_id}` - Download specific attachment

- **Export Routes**

- `/api/export/drive` - Export attachments to Google Drive

  - Parameters:
    - `message_ids` - List of message IDs
    - `folder_id` - Target folder in Drive
    - `organize_by` - Organization strategy (date, sender, etc.)
- `/api/export/s3` - Export attachments to S3
  - Parameters:
    - `message_ids` - List of message IDs
    - `bucket` - Target S3 bucket
    - `prefix` - S3 key prefix
    - `organize_by` - Organization strategy (date, sender, etc.)

- **Jobs Routes**

- `/api/jobs` - List all export jobs

- `/api/jobs/{job_id}` - Get status of specific job
- `/api/jobs/{job_id}/cancel` - Cancel a running job

Implementation Example

Example of a route handler for searching emails:

```python @gmail_bp.route('/search', methods=['GET']) @jwt_required def search_emails(): query = request.args.get('q', 'has:attachment') max_results = int(request.args.get('max_results', 100)) start_date = request.args.get('start_date') end_date = request.args.get('end_date')

# Format date range if provided
if start_date and end_date:
    query += f' after:{start_date} before:{end_date}'

# Call Gmail service to perform search
results = gmail_service.search_messages(query, max_results)

return jsonify({
    'success': True,
    'count': len(results),
    'messages': results
})

```

Example of an export to Drive endpoint:

```python @export_bp.route('/drive', methods=['POST']) @jwt_required def export_to_drive(): data = request.get_json() message_ids = data.get('message_ids', []) folder_id = data.get('folder_id') organize_by = data.get('organize_by', 'date')

# Create background job
job_id = str(uuid.uuid4())

# Start background task
export_task.delay(job_id, message_ids, 'drive', folder_id=folder_id, organize_by=organize_by)

return jsonify({
    'success': True,
    'job_id': job_id,
    'message': f'Export job created with ID: {job_id}'
})

```

Background Processing

1. **Job Queue System**

   - Implement a task queue (Celery or similar) for processing exports
   - Store job status and progress in a database

- **Progress Tracking**
- Create endpoints for checking job progress
- Implement websocket connections for real-time updates

API Documentation

1. **OpenAPI Specification**
   - Create OpenAPI (Swagger) documentation for all endpoints
   - Include request/response examples
   - **Interactive Documentation**
   - Implement Swagger UI for testing API endpoints
   - Add clear descriptions and usage examples

8. Advanced Features

1. **Scheduled Export Jobs**
   - Create endpoints for scheduling recurring exports
   - Implement a scheduler for running jobs at specified intervals
   - **Filtering Options**
   - Implement advanced filtering capabilities (file type, size, etc.)
   - Create dedicated endpoints for filter management
   - **Error Handling and Retry Logic**
   - Implement robust error handling
   - Add retry mechanism for network failures
   - Create detailed logs and expose them via API
   - **User Preferences**
   - Store and manage user preferences
   - Create endpoints for updating preferences

9. Testing

1. **Unit Tests**
   - Test individual API endpoints
   - Mock service layer calls
   - **Integration Tests**
   - Test the entire API workflow
   - Test authentication flows
   - **Load Testing**
   - Test API performance under load
   - Identify bottlenecks and optimize

10. API Client Examples

Using cURL

```bash

# Search for emails with PDF attachments

curl -X GET "http://localhost:5000/api/gmail/search?q=has:attachment%20filename:pdf" \ -H "Authorization: Bearer YOUR_JWT_TOKEN"

# Export attachments to Drive

curl -X POST "http://localhost:5000/api/export/drive" \ -H "Content-Type: application/json" \ -H "Authorization: Bearer YOUR_JWT_TOKEN" \ -d '{"message_ids": ["1234abc", "5678def"], "folder_id": "DRIVE_FOLDER_ID"}' ```

### Using Python

```python import requests

API_URL = "http://localhost:5000/api" TOKEN = "YOUR_JWT_TOKEN"

# Search for emails with PDF attachments

response = requests.get( f"{API_URL}/gmail/search", params={"q": "has:attachment filename:pdf"}, headers={"Authorization": f"Bearer {TOKEN}"} ) print(response.json())

# Export attachments to S3

response = requests.post( f"{API_URL}/export/s3", json={ "message_ids": ["1234abc", "5678def"], "bucket": "my-attachments-bucket", "prefix": "emails/" }, headers={"Authorization": f"Bearer {TOKEN}"} ) print(response.json()) ```

## Conclusion

This implementation plan outlines the steps to create a robust Gmail Attachment Exporter API service. The API design provides a flexible interface for searching emails and exporting attachments to either Google Drive or AWS S3. The modular approach allows for easy extension and maintenance, while the comprehensive API endpoints provide powerful attachment management capabilities.