Multiple File Upload API Integration Guide

Base URL: http://127.0.0.1:8080/vector/upload

1. Create a Batch (Register files to upload)

- Method: POST
- Endpoint: /batches
- Headers: Content-Type: application/json
- Body (JSON):

```
{
  "user_id": "user123",
  "files": [
    {"name": "example1.pdf", "order": 1},
    {"name": "example2.jpg", "order": 2}
  ]
}
```

- Response:

```
{
  "batch_id": "batchIdHere",
  "files": [
    {
      "file_id": "fileId1",
      "name": "example1.pdf",
      "order": 1,
      "upload_url": "/upload/upload-file/fileId1"
    },
    {
      "file_id": "fileId2",
      "name": "example2.jpg",
      "order": 2,
      "upload_url": "/upload/upload-file/fileId2"
    }
  ]
}
```

2. Upload Each File (One by one)

- Method: PUT
- Endpoint: /upload-file/{file_id} (Use file_id from batch response)
- Headers: Content-Type: multipart/form-data (handled automatically)
- Body: form-data key=file value=actual file selected
- Curl Example:

```
curl -X PUT "http://127.0.0.1:8080/vector/upload/upload-file/fileId1" -F "file=@/path/to/example1.pdf"
```

- Response:

```
{
  "file_id": "fileId1",
  "filename": "example1.pdf",
  "status": "uploaded",
  "file_hash": "hashstring"
}
```

3. Finalize Batch (After all files uploaded)

- Method: POST
- Endpoint: /batches/{batch_id}/finalize
- Response:

```
{
  "batch_id": "batchIdHere",
  "status": "completed"
}
```

- Note: If not all files uploaded, error mentions pending files.

4. Get Batch Status and Files

- Method: GET
- Endpoint: /batches/{batch_id}
- Response:

```
{
  "batch_id": "batchIdHere",
  "status": "completed",
  "files": [
    {
      "file_id": "fileId1",
      "filename": "example1.pdf",
      "status": "uploaded",
      "file_hash": "hashstring"
    },
    {
      "file_id": "fileId2",
      "filename": "example2.jpg",
      "status": "uploaded",
      "file_hash": "hashstring"
    }
  ]
}
```

5. Get Count of Files in Batch

- Method: GET
- Endpoint: /batches/{batch_id}/count
- Response:

```
{
  "batch_id": "batchIdHere",
  "file_count": 2
}
```

6. Download a File

- Method: GET
- Endpoint: /files/{file_id}/download
- Response: file binary stream (to save on frontend)

7. Cancel/Delete a Batch

- Method: DELETE
- Endpoint: /batches/{batch_id}
- Response:

```
{
  "status": "deleted",
  "batch_id": "batchIdHere"
}
```

Additional Notes for Frontend:

- Allowed file types:
  Images: image/jpeg, image/png, image/jpg
  Others: application/pdf, application/txt, application/docx,
          application/vnd.openxmlformats-officedocument.wordprocessingml.document,
          application/octet-stream, audio/mpeg, audio/mp3, audio/wav, audio/x-wav
- Max image size: 5MB
- Max file size: 40MB

Workflow:

1. Create batch with file metadata
2. Upload each file to its file_id
3. Finalize batch after all uploads
4. Check batch status or count as needed
5. Download files or cancel batch as needed

Error handling:

- 400 Bad Request: invalid or missing data
- 404 Not Found: batch or file missing
- 409 Conflict: batch completed or duplicate file
- 413 Payload Too Large: file size exceeds limit

Headers: No auth headers by default; add if needed.