# Backend Engineer Coding Exercise

## Background

Casetext offers several products that allow customers to upload their own content for a one-of-a-kind research experience.

Some customers are interested in these products, but have too much content to upload themselves. Fortunately, their content is often stored in a third party document management system (DMS) that they're happy to give us access to so we can upload their content on their behalf.

In that scenario, we tend to split our effort into two phases. First, we'll do a oneoff backfill of their content into our system. Then we'll deploy an ongoing synchronization tool that keeps our copy of their content up to date moving forward. This approach gets them up and running quickly.

## Assignment

Our goal is to implement a portion of that ongoing DMS -> Casetext synchronization.

Assume we have the following:

- a customer with a large amount of data
- an internal research service that indexes arbitrary content
- access to this customer's DMS, which we've already used to perform an initial backfill into our research service

We've got their data in our system, but with each day our copy of it grows staler. We need something that'll keep it up to date moving forward.

### DMS

The customer's DMS exposes an API with an endpoint that lists the files in a data store.

For this exercise, we'll use mock responses. dms-responses.zip is a bunch of JSONL files, each the DMS' response to a data store listing request at a different point in time. All files correspond to the same data store.

Each JSON object in each JSONL file represents a file in the data store at that point in time. Each data store file has a unique id, a name, and a metadata blob.

## Research Service

Our research service exposes an API to manage its copies of files. Its relevant operations are

- createFile. This takes in a file ID, name, and metadata.
- updateFileName. This takes in an existing file ID and new name for the file.
- updateFileMeta. This takes in an existing file ID and new metadata blob.
- deleteFile. This takes in a file ID.

## Deliverable

Develop an HTTP endpoint that identifies the research service operations necessary to sync the DMS data store to our research service across two given points in time.

The endpoint should have two parameters, each a date. Its output should be JSON that includes a list of operations. It should identify the operations for any two dates using the mock DMS responses.

For example, let's say our mock DMS responses include 2023-05-01.jsonl with these files:

```
{"id": "67e5e9ed-baab-49f7-8290-1e9885ba8fa0", "name": "msj-a",
"meta": {"matter": "uber"}}
{"id": "2b89256a-ae91-420a-ac0b-35b36e45fa4f", "name": "mtd", "meta":
{"cat": "old"}}
{"id": "3b6b7a0e-a16d-4d2c-a2ae-670efcf444a3", "name": "depo",
"meta": {}}
```

and 2023-05-02.jsonl with these files:

```
{"id": "67e5e9ed-baab-49f7-8290-1e9885ba8fa0", "name": "msj-a FINAL",
"meta": {"matter": "uber"}}
{"id": "2b89256a-ae91-420a-ac0b-35b36e45fa4f", "name": "mtd", "meta":
{"cat": "old", "deleteSoon": true}}
{"id": "8e1fe401-f031-46be-9437-f00273baca1c", "name": "msj-b",
"meta": {}}
```

Then when the endpoint is given the dates 2023-05-01 and 2023-05-02, its output should include the following operations:

```
createFile {"id": "8e1fe401-f031-46be-9437-f00273baca1c", "name":
"msj-b", "meta": {}}
deleteFile {"id": "3b6b7a0e-a16d-4d2c-a2ae-670efcf444a3"}
updateFileName {"id": "67e5e9ed-baab-49f7-8290-1e9885ba8fa0", "name":
"msj-a FINAL"}
```

```
updateFileMeta {"id": "2b89256a-ae91-420a-ac0b-35b36e45fa4f", "meta": {"cat": "old", "deleteSoon": true}}
```

Your endpoint should be implemented in Python, and have at least one unit test.

## Tips

- A JSONL file is just a bunch of JSON objects separated by newlines. It's also known as NDJSON.
- Use any web framework you like! flask and fastAPI are a couple options.
- The endpoint needs to accept two dates and respond with the operations, but its request and response formats are totally up to you.
- The project shouldn't take longer than about 2-6 hours.
- Feel free to ask questions if you get stuck anywhere.

## Evaluation

We'll evaluate the code you submit for correctness and extensibility. We'll go over some test cases by asking you to perform HTTP requests to your endpoint, possibly involving a new set of mocked DMS responses.