Common application API proposal

Definition

Server: The Application runs an HTTP(S) web server that accepts API requests as defined below.

Client: A Backend (AppStore service) acts as a client and interacts with the server to create tasks, upload images and fetch results.

Process flow

1. A client creates a new task; server responds by requesting attributes and providing taskId.
2. The client starts uploading images using the **Upload assets** endpoint. The client must respect `assets_per_batch` the option specified by the server.
3. Once all images are downloaded, server starts processing the task; - status changed to PROCESSING
4. The client checks for task status.
5. Once the status is changed to COMPLETED, the client can download the response.
6. Optionally, the client can instruct to clean up all task data by calling **Delete task** endpoint.

Authentication

All endpoints should secured using OIDC authentication. The only exception is **Status endpoint**, which should be publicly accessible.

An example of how to use OpenID authentication with Flask web server can be found here:

`https://gist.github.com/infohash/bfb58ffa0fc125c323b81539cab51498`

Server endpoints

Create new task `POST /api/v1/task`

Instructs the server to create a new task, and returns a response specifying required image attributes and image count per batch.

Response data JSON:

```
{
  "task_id": uuid,
  "attributes": [], // Attribute names
  "assets_per_batch": int, // How many assets (images) should be sent per a batch
}
```

Upload assets `POST /api/v1/assets`

The client uploads asset data to the server.
The client only uploads Asset Download URLs, so the server is authoritative over which assets will be downloaded. Additionally, this reduces the bandwidth requirements as the client doesn't necessarily need to download assets, and it gives the server ability to convert data to whatever formats it needs.

Body:

```
{
  "task_id": uuid // A Guid/UUID of the task
  "data": AssetData[] // A list of AssetData objects
}
```

Response: The server returns a response object, with no extra data.

Task status endpoint `GET /api/v1/task?id={taskId}`

The purpose of this endpoint is to provide information about the task's status.

Response body:

```
{
  "task_id": uuid, // An ID of a task
  "status": Status, // A Status enum serialized as uppercase string
  "assets_downloaded": int, // A number of assets downloaded in %
  "result_id": string|null, // Null or a URL to download result if the status is COMPLETED;
The result URL should most likely use Result Endpoint
  "result_type": ResultType|null // A null or ResultType object describing the type of data
the result is
}
```

Result endpoint `GET /api/v1/result?id={taskId}`

This endpoint is meant to return the result of the task. Based on the tasks `ResultType` the result can be:

- **STREAM:** In the format of `octet-stream` response with `attachment` an attribute specifying the file name.
- **JSON:** A JSON body response.
- **WEBSITE:** A permanent URL pointing to a page where there results can be found.

In the case of the task producing multiple files, the response type should be STREAM of a ZIP archive, that the client will download.

In case of task is **not found, 404 should be returned.**

Delete task `DELETE /api/v1/task?id={taskId}`

This endpoint should delete all task data (including results) from the server. *Implementation of this endpoint is optional.*

Response: The server returns a response object, with no extra data.

Status endpoint `GET /api/v1/status`

Implementation of this endpoint is optional. But would be useful for monitoring information from all applications once deployed on a larger scale to the cloud.

This is a metrics-like endpoint that provides information about the server. The format should be compatible with the Prometheus metrics endpoint standard, but including any extra metrics is optional.

At least these metrics are required: *example of body*

```
pending_tasks_total <decimal> // A number of currently not finished tasks
```

More about Prometheus metrics, and a Python example can be found here: https://prometheus.github.io/client_python/getting-started/three-step-demo/

Objects

Response

```
{
  "success": true|false,
  "message": string|null,
  "data": JSON
}
```

AssetData

```
{
  "file_name": string // A name of the asset file
  "attributes": { // A map of attribute names to values
    "longitude": object
    // ...
  },
  "download_url": string // A URL from where the asset can be downloaded from
}
```

Status [Enum]

```
CREATED // A task was created, but no assets were downloaded yet
DOWNLOADING // A server is downloading assets
WAITING // A server is no longer downloading assets, is waiting for more assets or waiting
for processing to start
PROCESSING // A processing of data is happening
COMPLETED // A task has been completed, and client can download result
```

ResultType [Enum]

```
JSON,
STREAM,
WEBSITE
```