Algorithms integration

Definitions

CC.AI - CancerCenter.ai

TPA - Third-Party Algorithm

Protocol

Communication is done by HTTP requests. For one algorithm run there are following steps:

1. `CC.AI` starts by sending by sending `Initialization` request to TPA URL. `TPA` should respond with status code `200` and run algorithm on image using data from `Initialization` request.
2. The algorithm is in progress. `TPA` sends `Status update` requests that inform `CC.AI` about current progress of processing. The time between subsequent `Status update` requests (and `Initialization` and first `Status update`) should not exceed one minute - otherwise `CC.AI` will consider the algorithm run as crashed.
3. The algorithm returns the result. It should send the `Status update` request with status `completed` or `error`, indicating the result. After that, all `Status update` requests will be ignored.

Test server and client

Test server and client are provided to help with implementing protocol on `TPA` side - `test-server` imitates `CC.AI` and test-client is an example implementation of `TPA`. They are both written using Python. In order to run the server and the client user should install dependencies provided in `requirements.txt` .

Test server

Running test server

`test-server` is activated by running command `python main.py` inside `\server` directory. It supports following command-line arguments:

Argument	Mandatory	Description	Default value
`--image_path`	Yes	Path to the image to process. It will be accessible on `tiles_url` from `Initialization` request	--
`--tpa_url`	No	URL to which `test-server` will send `Initialization` request	`http://127.0.0.1:12346/run_algorithm`
`--auth`	No	If specified, its value will be set as `Authorization` header value while sending `Initialization request`	--
`--port`	No	Port number which `test-server` will use	12345
`--host_url`	No	This URL will be used to create `return_url` field in `Initialization` request	`http://127.0.0.1`

Example activation: `python main.py --image_path path/to/data/1.svs --port 54321`

Behavior

Upon activation it will send `Initialization request` to `--tpa_url` and start listening for `Status update` requests. For each of those requests it will print on command-line information that it received. When `Status update` with `status` field set to `completed` or `error` is received, `test-server` will print the result and finish. The server will also provide tiles from image specified by `image_path` argument by responding on endpoint specified in `tiles_url` field.

Test client

Running client server

`test-client` is activated by running command `python main.py` inside `\client` directory. It supports following command-line arguments:

Argument	Mandatory	Description	Default value
`--port`	No	Port number which `test-client` use	12346

Example activation: `python main.py --port 54321`

Behavior

When ran, client will listen for `Initialization` requests, upon receiving one it will start processing it. It will download tiles from image, while sending `Status update` requests with progress updates in meantime. After that it will either randomly send `Status update` with status `error` or notify `CC.AI` about further progress and end processing with `Status update` request that has status `completed`, and sample result set.

Requests

Initialization

`Initialization` is a `POST` request with following body:

```
{
  "image": {
    "levels": int,
    "width": int,
    "height": int,
    "tile_size": int,
    "tiles_url": string,
    "objective_magnification": float,
    "microns_per_pixel": float
  },
  "return_url": string,
  "id": string
}
```

Fields description

Field	Description
id	Unique ID assigned to single algorithm run
return_url	URL to which `TPA` should send `Status update` requests
image	Object that contains information about image on which the algorithm should be ran. The image is available to download as a tile pyramid. The file structure of the pyramid is described below, but for better understanding, one can search for information about Deep Zoom Image format (which we actually use under the hood).
image.levels	Number of zoom levels of image. Level `image.levels - 1` corresponds to original image, with size `image.height` x `image.width`. `image.levels - 2` corresponds to image downscaled 2 times, with size `image.height / 2` x `image.width / 2`, level `image.levels - 3` to image downscaled 4 times, with size `image.height / 4` x `image.width / 4` and so on. Level `0` corresponds to image downscaled enough times to have size 1x1
image.width	Width of the image
image.height	Height of the image
image.tile_size	Width and height of the tiles returned by sending requests to `image.tiles_url`. Last tiles of either dimension can be smaller than this value
image.tiles_url	URL of form `https://example.com/path/{level}/{x}_{y}.jpeg`, that can be used to download image tiles. `TPA` should substitute desired values for `{level}`, `{x}` and `{y}`. `x` and `y` are tile coordinates meaning that call with values `level = image.levels - 2`, `x = 3` and `y = 4` will return tile starting at `point(x = 3 * image.tile_size, y = 4 * image.tile_size)` from image downscaled 2 times
image.objective_magnification	Objective magnification that was used to scan image (typically 10, 20 or 40)
image.microns_per pixel	Microns per pixel of the scan

Status update

`Status update` is a `POST` request with following body:

```
{
  "status": str
  "progress": int
  "error": str
  "result": object
}
```

Fields description

Field	When mandatory	Description
status	Always	Status update that `TPA` wants to inform about. Possible values are: `in_progress`, `error` and `completed`
progress	When status is `in_progress`	Progress of algorithm run. Number in range 0 - 100
error	When status is `error`	Reason of failed processing
result	When status is `completed`	Object containing result of algorithm

Example request bodies

Initialization

```
{
    "image": {
        "levels": 17,
        "width": 41381,
        "height": 52329,
        "tile_size": 512,
        "tiles_url": "http://127.0.0.1:54321/slide_files/{level}/{x}_{y}.jpeg",
        "objective_magnification": 40,
        "microns_per_pixel": 0.2491
    },
    "return_url": "http://127.0.0.1:54321/integrations/algorithm/5fb22a48-2e55-4e3f-8d91-1d5c32b67f38/status/",
    "id": "5fb22a48-2e55-4e3f-8d91-1d5c32b67f38"
}
```

Status update - `error`

```
{
    "status": "error",
    "error": "Random error occured"
}
```

Status update - `in_progress`

```json
{
    "status": "in_progress",
    "progress": 30
}
```

Status update - `completed`

```json
{
    "status": "completed",
    "result": {
        "regions_of_interest": [
            {
                "x": 20690,
                "y": 26164,
                "w": 100,
                "h": 100,
                "label": "Label 1",
                "score": 0.97
            }
        ],
        "other": "anything"
    }
}
```

Request for tile

`http://127.0.0.1:12345/slide_files/9/0_0.jpeg`