Simulations API Documentation - Ortho (public)

Overview

This document describes how Simulations API's public Ortho feature works.

To execute a ortho simulation the client will need to do a single request `POST /public-api/simulations/ortho` sending the photo binary as `multipart/form-data`. The response will be in JSON format as detailed along this document.

This request is meant to be done directly by the client (via web browser or mobile devices), using a server as middleman won't work since we have ip rate limiting to prevent malicious usage.

Request Structure

- POST /public-api/simulations/ortho
- Headers
  - `Authorization: Bearer $SIGNATURE`
  - `Content-Type: multipart/form-data`
  - `Origin: https://client-host.com` (for CORS)
- Body *(format: multipart/form-data)*
  - `img_photo: $PHOTO_BLOB`

Illustrating with cURL

```
curl -XPOST \
  -H "Authorization: Bearer $SIGNATURE" \
  -H "Content-Type: multipart/form-data" \
  -H "Origin: https://client-host.com" \
    -F "img_photo=@$PHOTO_PATH" \
    "https://api.e91efc7.dentrino.ai/public-api/simulations/ortho"
```

Response Structure

Success

```
// Status Code: 2xx
{
  "success": true,
  "beforeUrl": "https://.../before.jpg?...",
  "resultUrl": "https://.../result.jpg?..."
}
```

Error

```
// Status Code: 4xx / 5xx
{
  "success": false,
  "error": {
    "id": "PUBLIC ID",
    "message": "PUBLIC MESSAGE",
    "debug": {
      "__ALERT__": "THIS DEBUG OBJECT WILL NOT EXIST IN PRODUCTION",
      "debugId": "INTERNAL ID FOR DEBUGGING",
      "message": "PUBLIC MESSAGE FOR DEBBUGGING",
      "details": {...},
      "tags": {...}
    }
  }
}
```

Signature

Overview

The signature informs which client is using the API and prevents data from being tampered. It needs to be generated before each request and sent in the `Authorization` header.

To generate a signature the client will need a `CLIENT_ID` and `CLIENT_SECRET` that will be generated by TastyTech to each client. The `Client Secret` is used to hash the claims preventing them from being tampered.

The signature is composed by two parts joined by an `:`.

```
// Illustrative pseudo-code
CLAIMS_JSON = "{...}"
PART1 = base64($CLAIMS_JSON)
PART2 = hmac.sha256($CLAIMS_JSON, $CLIENT_SECRET)
SIGNATURE = "$PART1:$PART2"
```

Signature Claims

The claims are represented as a JSON and have the following information:

```
{
  "client_id": "...",
  "recaptcha_token": "...",
  "params_hashed": { ... }
}

// Recaptcha Token isn't mandatory for mobile clients.
```

Claim: `client_id`

The plain `CLIENT_ID`.

Claim: `recaptcha_token`

The token responded by Recaptcha v3. That should be used on web browsers, see Recaptcha v3 Google Docs for more information.

Claim: `params_hashed`

All the parameters sent in the body as `multipart/form-data` must be hashed as MD5 and added to the `claims.params_hashed` JSON.

```
// Sample of "params_hashed when sending "img_photo: $PHOTO_BINARY" in the body
{
  ...
  "params_hashed": {
    "img_photo": md5($PHOTO_BINARY)
  }
}
```

Demo

Install Dependencies

To run serve the demo webapp first install the dependencies by running:

```
# Install Dependencies
pip install -r requirements.txt
```

Credentials

By default it'll use the following credentials (for testing purposes only):

```
CLIENT_ID=ODMzNDQwMDMzNzU4OFdLIztKNFJd_testext_front
CLIENT_SECRET=2dba220dc7b1ffbbf96104bd4cdce7fabf2ec00af8083ccbb8fa51ba12c2924e
```

Sending requests via webapp

This folder can be served as a small webapp that send requests to the api. To do that run:

```
httpserver

# Starting server on ('127.0.0.1', 8080)
```

After starting the server access `http://localhost:8080` to send photos to the api. You can also read the files `index.html` and `js/*` to understand better how to make those requests.