C3990-Server API Documentation

Introduction

This document will detail all the endpoints that the server exposes and their inputs and responses. This is an exhaustive document which covers all the endpoints in detail.

/api/user

Provides an interface to get, create and update a user object.

> ENDPOINT: `/api/user`
> REST STATES: `GET, PUT, POST`

Input:

1. `user_id` : found in the JSON component of the request.
2. `update` : found in the JSON component of the request.
3. `google_oauth_token` : found in the JSON component of the request.

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"user_id": <string:user_id>}' http://localhost:5000/api/
```

> onFailure => []
> onSuccess => Document

```
Document
{
  "google_oauth_token":  <string>,
  "id":  <string>,
  "user_id": <string>,
}
```

Example: PUT

```
curl -H "Content-Type: application/json" -X PUT -d '{"user_id": "<string:user_id>", "update": <string:json_ol
```

> NB: "UPDATE" requires a stringified JSON Object which contains the key-value pairs for the fields which you want to update. See the GET Response to see what you can modifiy

onFailure

```
{
  "message": {
    "content": "There is no User with that ID"
    }
}
```

onSuccess => Returns the updated Document, enclosed in an array;

Example: POST

```
curl -H "Content-Type: application/json" -X POST -d '{"google_oauth_token": <string:google_oauth_token>}' ht
```

> onFailure => `[]` It will either throw an error or the ReqParser will catch any bad inputs;
>
> onSuccess => `[{}]` will return the newly created document encased in an array, similar to how HTTP GET request was carried out

/api/user/oauth

Provides a confirmation for oAuth.

> ENDPOINT: /api/user/oauth
> REST STATES: GET

Input:

1. `google_oauth_token` : found in the locations

Example: GET

```
  curl -H "Content-Type: application/json" -X <GET/DELETE> -d '{"google_oauth_token": <string:google_oauth_t
```

> NB This only confirms or disconfirms whether or not the system has the Merchant Registered. It doesn't not do the registration;
>
> onFailure => `{"error": "1"}` this error response is Specific for the Merchant interface. it allows the Merchant interface to perform the actions required to enter the user.
>
> onSuccess => [ArrayOf] Document Example: `{ "user_id": <string>, "id": <string> }`

/api/user/store

> ENDPOINT: /api/user/stores
> REST STATES: GET

Input:

1. `user_id` - found in the JSON component of the request.

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"user_id": <string:user_id>}' http://localhost:5000/api/
```

> onFailure => []
> onSuccess => ArrayOf Document

Document:

```
{
"beacons": [ArrayOf<String>] ,
"id":  <string>,
"lat":  <string>,
"long": <string>,
"name": <string>,
"store_id": <string>,
"store_manager_id":<string>,
}
```

/api/beacon

> ENDPOINT: /api/beacon REST STATES: GET, DELETE, PUT, POST

Input:

1. beacon_id - found in the JSON component of the request.
2. beacon_uuid - found in the JSON component of the request.
3. beacon_major - found in the JSON component of the request.
4. beacon_minor - found in the JSON component of the request.
5. update - found in the JSON component of the request.

Example: GET, DELETE

```
curl -H "Content-Type: application/json" -X <GET/DELETE> -d '{"beacon_id": <string:beacon_id>}' http://loca
```

```
    onSuccess => [
        {
        "beacon_id": <string>,
        "claimed": <string>,
        "id": <string>,
        "major": <string>,
        "minor": <string>,
        "owner": <string>,
        "uuid": <string>
        }
    ]
```

Example: PUT

```
curl -H "Content-Type: application/json" -X PUT -d '{"beacon_id": "<string:beacon_id>", "update": <string:j
```

> NB: "UPDATE" requires a stringified JSON Object which contains the key-value pairs for the fields which you want to update. See the GET Response to see what you can modifiy

```
     onFailure => {
         "message": {
             "content": "There is no Beacon with that ID"
          }
     }
```

```
 `onSuccess => Returns the updated Document, enclosed in an array;`
```

Example: POST

```
curl -H "Content-Type: application/json" -X POST -d '{"beacon_uuid": <string:beacon_uuid>, "beacon_major":
```

/api/user/favourite

provides the user's beacons endpoint.

> ENDPOINT: `/api/user/favourite`
> REST STATES: `GET, POST`

Input:

1. `user_id` - found in the JSON component of the request.
2. `store_id` - found in the JSON component of the request.

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"user_id": <string:user_id>}' http://localhost:5000/api
```

> onSuccess => Document

```
 {
   "favouritestores": ArrayOf<StoreDocument> ,
   "id":  <string>,
   "user_id":  <string>
 }
```

> Store Document:

```
{
    "beacons": ArrayOf<String>,
    "id": <String>,
    "lat": <String>,
    "long": <String>,
    "name": <String>,
    "promotion": {
        "active": <String>,
        "coupon": <String>,
        "expires":<String>,
        "id": <String>,
        "message": <String>,
        "present": <String>,
        "promotionImage":<String>,
        "promotion_id":<String>,
        "store_id":<String>,
        "title":<String>,
    },
    "store_id":<String>,
    "store_manager_id":<String>,
}
```

Example: POST

```
curl -H "Content-Type: application/json" -X POST -d '{"user_id": <string:user_id>, "store_id": <string:store
```

/api/user/interact/beacons

provides the user endpoint;

> ENDPOINT: /api/user/interact/beacons
> REST STATES: GET,POST

Input:

1. `user_id` - found in the JSON component of the request.
2. `update` - found in the JSON component of the request.
3. `google_oauth_token` - found in the JSON component of the request.

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"user_id": <string:user_id>}' http://localhost:5000/api
```

> onFailure => Document

```
{
         "message": {
             "content": "That user does not exist"
         }
     }
```

> onSuccess => Document

```
{
         "interacted":  ArrayOf<DocumentInteractBeacon>,
         "id":  <string>,
         "user_id": <string>,
}
```

> DocumentInteractBeacon

```
{
         "beacon_id": <string>,
         "date": <string>,
         "promotion_id": <string>,
         "store_id": <string>
}
```

Example: POST

```
curl -H "Content-Type: application/json" -X POST -d '{"user_id": <string:user_id>}' http://localhost:5000/ap
```

/api/user/store/promotion

provides the user's store's promotion endpoint;

> ENDPOINT: /api/user/store/promotion
> REST STATES: GET

Input:

1. `user_id` - found in the JSON component of the request.

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"user_id": <string:user_id>}' http://localhost:5000/api/
```

> onSuccess => ArrayOf

Document:

```
{
        "active":  <string>
        "coupon":  <string>
        "expires":  <string>
        "id":  <string>
        "message":  <string>
        "present":  <string>
        "promotionImage":  <string>
        "promotion_id": <string>
        "store_id": <string>
        "title": <string>
}
```

/api/user/store/promotion

provides the user's store's promotion endpoint;

> ENDPOINT: /api/user/store/promotion
> REST STATES: GET

Input:

1. `user_id` - found in the JSON component of the request.

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"user_id": <string:user_id>}' http://localhost:5000/api/
```

onSuccess => ArrayOf Document:

```
   {
       "active":  <string>
       "coupon":  <string>
       "expires":  <string>
       "id":  <string>
       "message":  <string>
       "present":  <string>
       "promotionImage":  <string>
       "promotion_id": <string>
       "store_id": <string>
       "title": <string>
}
```

/api/store

provides the store endpoint;

> ENDPOINT: `/api/store` REST STATES: `GET, DELETE, PUT, POST`

Input:

1. `store_id` - found in the JSON component of the request or in the other fields.
2. `update` - found in the json or other fields.

Example: GET, DELETE

```
curl -H "Content-Type: application/json" -X <GET/DELETE> -d '{"store_id": <string:store_id>}' http://localho
```

> onSuccess => [ArrayOf Document]

```
{
      "beacons": [ArrayOf<String>],
      "id": <string>,
      "lat": <string>,
      "long": <string>,
      "name": <string>,
      "store_id": <string>,
      "store_manager_id": <string>,
      }
```

EXAMPLE DELETE

```
onSuccess => {
             "deleted": <string:store_id>
}
```

Example: PUT

```
curl -H "Content-Type: application/json" -X PUT -d '{"store_id": "<string:store_id>", "update": <string:json
```

> NB: "UPDATE" requires a stringified JSON Object which contains the key-value pairs for the fields which you want to update. See the GET Response to see what you can modifiy

```
onFailure => {
         "message": {
             "content": "There is no Store with that ID"
          }
     }
```

```
onSuccess => Returns the updated Document, enclosed in an array;
```

Example: POST

```
curl -H "Content-Type: application/json" -X POST -d '{"store_manager_id": <string:store_manager_id>}' http:/
```

> NB: This one simply creates an entry, which has to edited via the merchant interface.
>
> onSuccess => `[{}]` will return the newly created document encased in an array, similar to how HTTP GET request was carried out

/api/promotion

provides the promotion endpoint;

> ENDPOINT: `/api/promotion` REST STATES: `GET, DELETE, PUT, POST`

Input:

1. title - found in the JSON component of the request or the default location.
2. message - found in the JSON component of the request or the default location.
3. coupon - found in the JSON component of the request or the default location.
4. present - found in the JSON component of the request or the default location.
5. expires - found in the JSON component of the request or the default location.
6. store_id - found in the JSON component of the request or the default location.
7. beacon_id - found in the JSON component of the request or the default location.
8. active - found in the JSON component of the request or the default location.
9. promotionImage - found in the JSON component of the request or the default location.
10. update - found in the JSON component of the request or the default location.

Example: GET, DELETE

```
curl -H "Content-Type: application/json" -X <GET/DELETE> -d '{"promoton_id": <string:beacon_id>}' http://loca
```

> onSuccess => [{Document}] NB: There will only be one entry in the JSONArray.

```
     {
       "active": <string>,
       "coupon": <string>,
       "expires": <string>,
       "id": <string>,
       "message": <string>,
       "present": <string>,
       "promotionImage": <string>,
       "promotion_id": <string>,
       "store_id": <string>,
       "title": <string>,
     }
```

Example: Delete

```
          onSuccess => {
              "deleted": <string:promotion_id>
          }
```

Example: PUT

```
curl -H "Content-Type: application/json" -X PUT -d '{"promotion_id": "<string:promotion_id>", "update": <str
```

> NB: "UPDATE" requires a stringified JSON Object which contains the key-value pairs for the fields which you want to update. See the GET Response to see what you can modifiy

```
onFailure => {
         "message": {
             "content": "There is no Promotion with that ID"
          }
     }
```

> onSuccess => Returns the updated Document, enclosed in an array;

Example: POST

```
curl -H "Content-Type: application/json" -X POST -d '{"title": <string:title>, "message": <string:message>, '
```

> onFailure => `[]` It will either throw an error or the ReqParser will catch any bad inputs; onSuccess => `[{}]` will return the newly created document encased in an array, similar to how HTTP GET request was carried out

/api/promotion/materials

provides the promotion materials endpoint;

> ENDPOINT: `/api/promotion/materials` REST STATES: `POST`

Example: POST

```
N/A
```

> NB This isn't like the other endpoints. This endpoint accepts raw data via FormData.
> onSuccess => [{}]

```
          {
              "id": <string>,
              "material_id": <string>,
              "path": <string>
          }
```

/api/beacon/store

provides the beaconstore endpoint;

> ENDPOINT: `/api/beacon/store`
> REST STATES: `GET`

Input

1. beacon_id - Flask looks in the default locations

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"beacon_id": <string:beacon_id>}' http://localhost:5000/
```

> onSuccess

```
        [
         {
             "beacons": [ArrayOf<string:beacon_id>],
             "id": <string:id>,
             "name": <string:name>,
             "lat": <string:lat>,
             "long": <string:long>
             "promotion": [ ArrayOf<Document:promotion> ]
                 EXAMPLE DOCUMENT: {
                     "active": <string:active>,
                     "coupon": <string:coupon>,
                     "expires": <string:expires>,
                     "id": <string:id>,
                     "message": <string:message>,
                     "present": <string:present>,
                     "promotionImage": <string:promotionImage>,
                     "promotion_id": <string:promotion_id>,
                     "store_id": <string:store_id>,
                     "title": <string:title>
                 }
             ,
             "store_id": <string:store_id>,
             "store_manager_id": <string:store_manager_id>
         }
      ]
```

/api/stats/interact/store

provides the Stats Data for stores endpoint;

> ENDPOINT: `/api/stats/interact/store`
> REST STATES: `GET`

Input:

1. `store_id` : Flask looks in the default locations, including JSON

Example: GET

```
curl -H "Content-Type: application/json" -X <GET/DELETE> -d '{"store_id": <string:store_id>}' http://localho
```

> onSuccess => [ArrayOf customDocument]

```
CustomDocument Example:  {
         "beacon_id": <string:beacon_id>,
         "date": <string:date>,
         "promotion_id": <string:promotion_id>,
         "store_id": <string:store_id>
      }
```

/api/order

provides place order endpoint;

> ENDPOINT: `/api/order`
> REST STATES: `GET`

Input:

1. `email` : Flask looks in the default locations, including JSON
2. `message` : Flask looks in the default locations, including JSON
3. `beacons` : Flask looks in the default locations, including JSON
4. `user_id` : Flask looks in the default locations, including JSON

Example: GET

```
curl -H "Content-Type: application/json" -X GET -d '{"user_id": <string:user_id>, "message": <string:message
```

> onSuccess

```
         {
             "order_id": <string>
         }
```

/api/materials/

provides the beacon endpoint;

> ENDPOINT: `/api/materials/<string:id>`
> REST STATES: `GET`

Input:

1. `id` - found in the URL of the request.

Example: GET

```
curl -X GET http://localhost:5000/api/materials/<string:id>
```

> NB: This returns an image object on success or a generic error on failure;