Documentation API

Table of content

This Project is made with love by Duncan Lhuillery and Thibaut Figueira.

Getting started :

The following steps will help you to get started with your journey towards using our beautiful API:

You can chose to use the api on your local pc without any server or use our docker file who contain two server, a web server and a server for the db.

Run the API on your local Pc

1- **First you need to download "Mysql installer", here the link:**

https://dev.mysql.com/downloads/installer/
On the installer, make sure to install "Mysql serveur" and "Mysql workbench".

2- **Configure Mysql workbench:**

Launch "Mysql workbench" and configure the connection with those information:
Server Name = localhost, User = root, password= azerty.
Then on the management section, select import, Import from Self-contains file, select our database on the database folder in the project (lancement.sql), then click "New" and name it "mj". The db should now be displayable on the schemas section.

3- **Launch the Api:**

Now just type :
dotnet run --project SITE_MJ/API_MJ/
The Api is alive !

4- **Try it on Postman or in hand:**

You can try some request in the search bar of your favorite explorer or the more handy way, do it with our Postman!

Go on Postman website, and create a account.
Now you can access to our Postman:

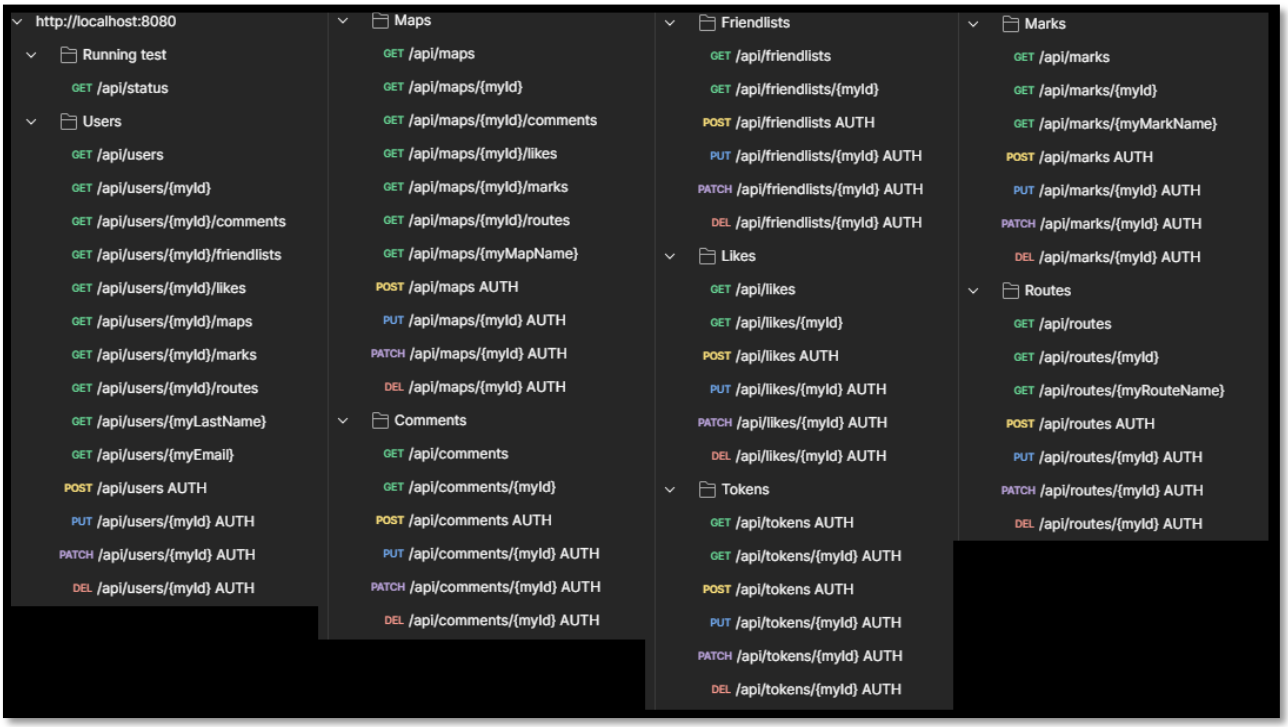
https://app.getpostman.com/join-team?invite_code=9232aefb2b28d3d6fd0eb2592cfee1f9&target_code=43b027181148ccc910e28d68b374eb60

Don'tt forget to download Postman agent:

https://www.postman.com/downloads/postman-agent/

Try out some request!

Here the list of our request (more details in the “references endpoints part, below):



Run the API with our Docker

1. **Setup Docker:**

   First, you will need to download Docker desktop on this link:
   https://www.docker.com/products/docker-desktop/

   Don't forget to open it every time you will need to use our api.

2. **Run Docker:**

   In Docker, be sure to delete every existing container, image and volume related to the project before composing.

   Be sure to not have mysqlworkench open when using the api on docker.

   On the terminal, you just have to type:

   docker-compose up --build

   Everything should be good! You just have to test the api with Postman!

3. **Try it on Postman:**

   You can try some request in the search bar of your favorite explorer or the more handy way, do it with our Postman!

   Go on Postman website, and create a account.
   Now you can access to our Postman:

   https://app.getpostman.com/join-team?invite_code=e33bbf89c4080c120b3fac1a2ec72bf9&target_code=d39e99bc428fa0d95c69a83c7ce353c5

   Dont forget to download Postman agent:

   https://www.postman.com/downloads/postman-agent/

   Try out some request!

Concepts : API calls

Our API is a restful API with different endpoints which return JSON data about users, product, or their shoplist.

Base Url

The base address of our web API is:

http://localhost/api/

Request

Data resources are accessed via standard HTTP requests in UTF-8 format to an API endpoint. The Web API uses the following HTTP verbs:

METHOD	ACTION
GET	Retrieve resources
POST	Creates resources
PUT	Replaces resources or create a new one
PATCH	Partial change of a resources
DELETE	Deletes resources

Responses

Web API normally returns JSON in the response body. Some endpoints don't return JSON but the HTTP status code

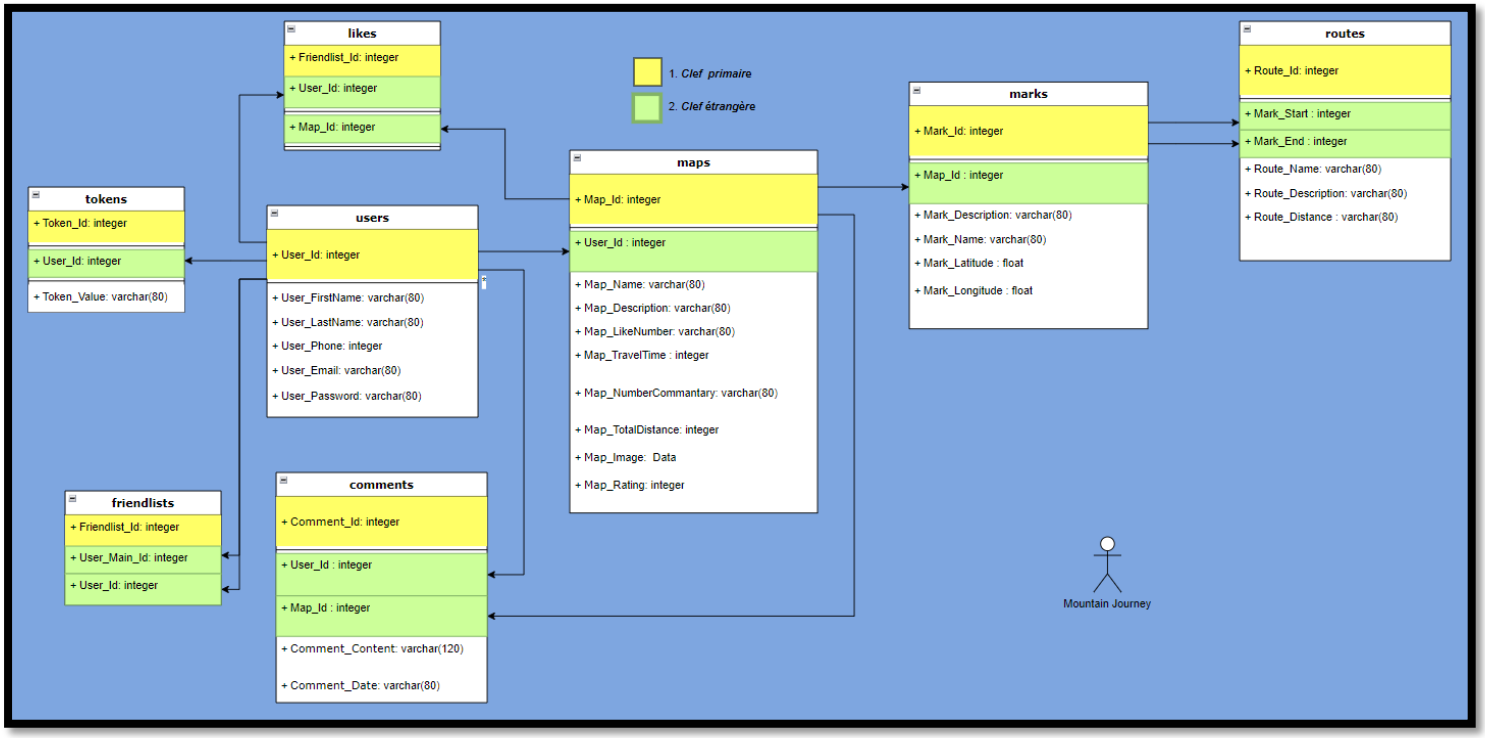
Response status codes

Status code	Description
200	OK
201	Created
202	Accepted
204	No Content
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
429	Too Many Requests
500	Internal Server Error
502	Bad Gateway
503	Service Unavailable

Tokens verification

We use on our api beared token for protect our request, all request under protection have the AUTH word attach to it. Be sure to check the read me or ask to the dev some key for try the api out!

DMC



Here the DMC of our DB, all our api endpoint referred to it.

References endpoint

Running test

Users

Maps

Friendlists

Likes

Marks

Routes

Comments

Tokens

Running test – Status

Test if the API is running; this does not test if the DB connection works properly.

Method: **Get**
Endpoint: `/api/status`
Example: `http://localhost:8080/api/status`

Response:

- **200** – The API is running!
- **503** – Error, could not send a request (the API is offline).

Users – Get all users

Return in a JSON all the users in the DB.

Method: **Get**
Endpoint: `/api/users`
Example: `http://localhost:8080/api/users`

Response:

- **200** – All the JSON users.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get a user by Id

Return in a JSON a user corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/users/{myId}`
myId int Required
Example: `http://localhost:8080/api/users/4`

Response:

- **200** – A JSON of the user.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get the user comments by Id
Return in a JSON the comments made by a user corresponding to the passed Id.

Method: Get
Endpoint: `/api/users/{myId}/comments`
myId int Required
Example: `http://localhost:8080/api/users/4/comments`

Response:

- **200** – A JSON of the user comments.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get the user friendlists by Id
Return in a JSON the friendlists of a user corresponding to the passed Id.

Method: Get
Endpoint: `/api/users/{myId}/friendlists`
myId int Required
Example: `http://localhost:8080/api/users/4/friendlists`

Response:

- **200** – A JSON of the user friendlists.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get the user likes by Id
Return in a JSON the likes of a user corresponding to the passed Id.

Method: Get
Endpoint: `/api/users/{myId}/likes`
myId int Required
Example: `http://localhost:8080/api/users/4/likes`

Response:

- **200** – A JSON of the user likes.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get the user maps by Id
Return in a JSON the maps of a user corresponding to the passed Id.

Method: Get
Endpoint: `/api/users/{myId}/maps`
myId int Required
Example: `http://localhost:8080/api/users/4/maps`

Response:

- **200** – A JSON of the user maps.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get the user marks by Id
Return in a JSON the marks of a user corresponding to the passed Id.

Method: Get
Endpoint: `/api/users/{myId}/marks`
myId int Required
Example: `http://localhost:8080/api/users/4/marks`

Response:

- **200** – A JSON of the user marks.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get the user routes by Id
Return in a JSON the routes of a user corresponding to the passed Id.

Method: Get
Endpoint: `/api/users/{myId}/routes`
myId int Required
Example: `http://localhost:8080/api/users/4/routes`

Response:

- **200** – A JSON of the user routes.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get a user by his lastname

Return in a JSON a user corresponding to the passed lastname.

Method: **Get**
Endpoint: `/api/users/{myLastName}`
myLastName string Required
Example: `http://localhost:8080/api/users/hessel`

Response:

- **200** – A JSON of the user.
- **400** – Invalid name or email, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Get a user by his email

Return in a JSON a user corresponding to the passed email.

Method: **Get**
Endpoint: `/api/users/{myEmail}`
myEmail string Required
Example: `http://localhost:8080/api/users/destin96@blick.com`

Response:

- **200** – A JSON of the user.
- **400** – Invalid name or email, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Users – Post a user

Create a user based on a given body.

Method: **Post**
Endpoint: `/api/users`
Body JSON format required:

```
json

{
    "User_FirstName": "Thibaut",
    "User_LastName": "Figueira",
    "User_Email": "thibaut.ynov@hotmail.com",
    "User_Password": "azerty",
    "User_Phone": "0679034009"
}
```

Example: `http://localhost:8080/api/users`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Users – Put a user

Replace all the data of an existing user or, if the user doesn't exist, create a user based on a given body and Id.

Method: **Put**
Endpoint: `/api/users/{myId}`
myId int Required
Body JSON format required:

```
Json

{
    "User_FirstName": "Duncan",
    "User_LastName": "Chaman",
    "User_Email": "Grand.duncan@hotmail.com",
    "User_Password": "azerty",
    "User_Phone": "0890789887"
}
```

Example: `http://localhost:8080/api/users/6`

Response:

- **200** – It works! User mis à jour!
- **200** – This Id is empty, new user created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Users – Patch a user

Replace some data of an existing user based on a given body and Id.

Method: **Patch**
Endpoint: `/api/users/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "User_Email": "Iliane.smart@hotmail.fr"
}
```

Example: `http://localhost:8080/api/users/7`

Response:

- **200** – Patch success! User updated!
- **200** – This Id is empty, new user created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Users – Delete a user

Delete a user based on a given Id.

Method: **Delete**
Endpoint: `/api/users/{myId}`
myId int Required
Example: `http://localhost:8080/api/users/1`

Response:

- **200** – It works! User supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).

Maps – Get all maps

Return in a JSON all the maps in the DB.

Method: **Get**
Endpoint: `/api/maps`
Example: `http://localhost:8080/api/maps`

Response:

- **200** – All the JSON maps.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Maps – Get a map by Id

Return in a JSON a map corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/maps/{myId}`
myId int Required
Example: `http://localhost:8080/api/maps/4`

Response:

- **200** – A JSON of the map.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Maps – Get comments by map Id
Return in a JSON all comments corresponding to the passed map Id.

Method: Get
Endpoint: `/api/maps/{myId}/comments`
myId int Required
Example: `http://localhost:8080/api/maps/4/comments`

Response:

- **200** – A JSON of the comments.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Maps – Get likes by map Id
Return in a JSON all likes corresponding to the passed map Id.

Method: Get
Endpoint: `/api/maps/{myId}/likes`
myId int Required
Example: `http://localhost:8080/api/maps/4/likes`

Response:

- **200** – A JSON of the likes.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Maps – Get marks by map Id
Return in a JSON all marks corresponding to the passed map Id.

Method: Get
Endpoint: `/api/maps/{myId}/marks`
myId int Required
Example: `http://localhost:8080/api/maps/4/marks`

Response:

- **200** – A JSON of the marks.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Maps – Get routes by map Id
Return in a JSON all routes corresponding to the passed map Id.

Method: Get
Endpoint: `/api/maps/{myId}/routes`
myId int Required
Example: `http://localhost:8080/api/maps/4/routes`

Response:

- **200** – A JSON of the routes.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Maps – Get a map by name

Return in a JSON a map corresponding to the passed name.

Method: **Get**
Endpoint: `/api/maps/{myMapName}`
myMapName string Required
Example: `http://localhost:8080/api/maps/map1`

Response:

- **200** – A JSON of the map.
- **400** – Invalid name, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Maps – Post a map

Create a map based on a given body.

Method: **Post**
Endpoint: `/api/maps`
Body JSON format required:

```
Json

{
    "Map_Name": "map1",
    "Map_Description": "A sample map"
}
```

Example: `http://localhost:8080/api/maps`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Maps – Put a map

Replace all the data of an existing map or, if the map doesn't exist, create a map based on a given body and Id.

Method: **Put**
Endpoint: `/api/maps/{myId}`
myId int Required
Body JSON format required:

```
Json

{
    "Map_Name": "map2",
    "Map_Description": "An updated sample map"
}
```

Example: `http://localhost:8080/api/maps/6`

Response:

- **200** – It works! Map mis à jour!
- **200** – This Id is empty, new map created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Maps – Patch a map

Replace some data of an existing map based on a given body and Id.

Method: **Patch**
Endpoint: `/api/maps/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Map_Description": "Updated description"
}
```

Example: `http://localhost:8080/api/maps/7`

Response:

- **200** – Patch success! Map updated!
- **200** – This Id is empty, new map created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Maps – Delete a map

Delete a map based on a given Id.

Method: **Delete**
Endpoint: `/api/maps/{myId}`
myId int Required
Example: `http://localhost:8080/api/maps/1`

Response:

- **200** – It works! Map supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).

Friendlists – Get all friendlists

Return in a JSON all the friendlists in the DB.

Method: **Get**
Endpoint: `/api/friendlists`
Example: `http://localhost:8080/api/friendlists`

Response:

- **200** – All the JSON friendlists.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Friendlists – Get a friendlist by Id

Return in a JSON a friendlist corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/friendlists/{myId}`
myId int Required
Example: `http://localhost:8080/api/friendlists/4`

Response:

- **200** – A JSON of the friendlist.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Friendlists – Post a friendlist

Create a friendlist based on a given body.

Method: **Post**
Endpoint: `/api/friendlists`
Body JSON format required:

```
json

{
    "User_Id": 7
}
```

Example: `http://localhost:8080/api/friendlists`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Friendlists – Put a friendlist

Replace all the data of an existing friendlist or, if the friendlist doesn't exist, create a friendlist based on a given body and Id.

Method: **Put**
Endpoint: `/api/friendlists/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "User_Id": 9
}
```

Example: `http://localhost:8080/api/friendlists/7`

Response:

- **200** – It works! Friendlist mis à jour!
- **200** – This Id is empty, new friendlist created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Friendlists – Patch a friendlist

Replace some data of an existing friendlist based on a given body and Id.

Method: **Patch**
Endpoint: `/api/friendlists/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "User_Id": 7
}
```

Example: `http://localhost:8080/api/friendlists/8`

Response:

- **200** – Patch success! Friendlist updated!
- **200** – This Id is empty, new friendlist created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Friendlists – Delete a friendlist Delete a friendlist based on a given Id.

Method: **Delete**
Endpoint: `/api/friendlists/{myId}`
myId int Required
Example: `http://localhost:8080/api/friendlists/9`

Response:

- **200** – It works! Friendlist supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).

Likes – Get all likes

Return in a JSON all the likes in the DB.

Method: **Get**
Endpoint: `/api/likes`
Example: `http://localhost:8080/api/likes`

Response:

- **200** – All the JSON likes.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Likes – Get a like by Id

Return in a JSON a like corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/likes/{myId}`
myId int Required
Example: `http://localhost:8080/api/likes/4`

Response:

- **200** – A JSON of the like.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Likes – Post a like

Create a like based on a given body.

Method: **Post**
Endpoint: `/api/likes`
Body JSON format required:

```
json

{
    "User_Id": 7,
    "Map_Id": 5
}
```

Example: `http://localhost:8080/api/likes`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Likes – Put a like

Replace all the data of an existing like or, if the like doesn't exist, create a like based on a given body and Id.

Method: **Put**
Endpoint: `/api/likes/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "User_Id": 9,
    "Map_Id": 7
}
```

Example: `http://localhost:8080/api/likes/7`

Response:

- **200** – It works! Like mis à jour!
- **200** – This Id is empty, new like created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Likes – Patch a like

Replace some data of an existing like based on a given body and Id.

Method: **Patch**
Endpoint: `/api/likes/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Map_Id": 8
}
```

Example: `http://localhost:8080/api/likes/8`

Response:

- **200** – Patch success! Like updated!
- **200** – This Id is empty, new like created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Likes – Delete a like

Delete a like based on a given Id.

Method: **Delete**
Endpoint: `/api/likes/{myId}`
myId int Required
Example: `http://localhost:8080/api/likes/9`

Response:

- **200** – It works! Like supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).

Marks – Get all marks

Return in a JSON all the marks in the DB.

Method: **Get**
Endpoint: `/api/marks`
Example: `http://localhost:8080/api/marks`

Response:

- **200** – All the JSON marks.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Marks – Get a mark by Id

Return in a JSON a mark corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/marks/{myId}`
myId int Required
Example: `http://localhost:8080/api/marks/4`

Response:

- **200** – A JSON of the mark.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Marks – Get routes by mark Id
Return in a JSON all routes corresponding to the passed mark Id.

Method: Get
Endpoint: `/api/marks/{myId}/routes`
myId int Required
Example: `http://localhost:8080/api/marks/4/routes`

Response:

- **200** – A JSON of the routes.
- **400** – Invalid mark Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Marks – Get a mark by name

Return in a JSON a mark corresponding to the passed name.

Method: **Get**
Endpoint: `/api/marks/{myMarkName}`
myMarkName string Required
Example: `http://localhost:8080/api/marks/mark1`

Response:

- **200** – A JSON of the mark.
- **400** – Invalid name, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Marks – Post a mark

Create a mark based on a given body.

Method: **Post**
Endpoint: `/api/marks`
Body JSON format required:

```
json

{
    "Map_Id": 5,
    "Mark_Name": "mark1",
    "Mark_Description": "A sample mark"
}
```

Example: `http://localhost:8080/api/marks`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Marks – Put a mark

Replace all the data of an existing mark or, if the mark doesn't exist, create a mark based on a given body and Id.

Method: **Put**
Endpoint: `/api/marks/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Map_Id": 6,
    "Mark_Name": "mark2",
    "Mark_Description": "An updated sample mark"
}
```

Example: `http://localhost:8080/api/marks/6`

Response:

- **200** – It works! Mark mis à jour!
- **200** – This Id is empty, new mark created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Marks – Patch a mark

Replace some data of an existing mark based on a given body and Id.

Method: **Patch**
Endpoint: `/api/marks/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Mark_Description": "Updated description"
}
```

Example: `http://localhost:8080/api/marks/7`

Response:

- **200** – Patch success! Mark updated!
- **200** – This Id is empty, new mark created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Marks – Delete a mark

Delete a mark based on a given Id.

Method: **Delete**
Endpoint: `/api/marks/{myId}`
myId int Required
Example: `http://localhost:8080/api/marks/1`

Response:

- **200** – It works! Mark supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).

Routes – Get all routes

Return in a JSON all the routes in the DB.

Method: **Get**
Endpoint: `/api/routes`
Example: `http://localhost:8080/api/routes`

Response:

- **200** – All the JSON routes.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Routes – Get a route by Id

Return in a JSON a route corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/routes/{myId}`
myId int Required
Example: `http://localhost:8080/api/routes/4`

Response:

- **200** – A JSON of the route.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Routes – Get a route by name

Return in a JSON a route corresponding to the passed name.

Method: **Get**
Endpoint: `/api/routes/{myRouteName}`
myRouteName string Required
Example: `http://localhost:8080/api/routes/route1`

Response:

- **200** – A JSON of the route.
- **400** – Invalid name, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Routes – Post a route

Create a route based on a given body.

Method: **Post**
Endpoint: `/api/routes`
Body JSON format required:

```
json

{
    "Map_Id": 5,
    "Route_Name": "route1",
    "Route_Description": "A sample route"
}
```

Example: `http://localhost:8080/api/routes`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Routes – Put a route

Replace all the data of an existing route or, if the route doesn't exist, create a route based on a given body and Id.

Method: **Put**
Endpoint: `/api/routes/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Map_Id": 6,
    "Route_Name": "route2",
    "Route_Description": "An updated sample route"
}
```

Example: `http://localhost:8080/api/routes/6`

Response:

- **200** – It works! Route mis à jour!
- **200** – This Id is empty, new route created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Routes – Patch a route

Replace some data of an existing route based on a given body and Id.

Method: **Patch**
Endpoint: `/api/routes/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Route_Description": "Updated description"
}
```

Example: `http://localhost:8080/api/routes/7`

Response:

- **200** – Patch success! Route updated!
- **200** – This Id is empty, new route created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Routes – Delete a route Delete a route based on a given Id.

Method: **Delete**
Endpoint: `/api/routes/{myId}`
myId int Required
Example: `http://localhost:8080/api/routes/1`

Response:

- **200** – It works! Route supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).

Comments – Get all comments

Return in a JSON all the comments in the DB.

Method: **Get**
Endpoint: `/api/comments`
Example: `http://localhost:8080/api/comments`

Response:

- **200** – All the JSON comments.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Comments – Get a comment by Id

Return in a JSON a comment corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/comments/{myId}`
myId int Required
Example: `http://localhost:8080/api/comments/4`

Response:

- **200** – A JSON of the comment.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Comments – Post a comment

Create a comment based on a given body.

Method: **Post**
Endpoint: `/api/comments`
Body JSON format required:

```
json

{
    "User_Id": 7,
    "Map_Id": 5,
    "Comment_Content": "A sample comment"
}
```

Example: `http://localhost:8080/api/comments`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Comments – Put a comment

Replace all the data of an existing comment or, if the comment doesn't exist, create a comment based on a given body and Id.

Method: **Put**
Endpoint: `/api/comments/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "User_Id": 9,
    "Map_Id": 6,
    "Comment_Content": "An updated sample comment"
}
```

Example: `http://localhost:8080/api/comments/6`

Response:

- **200** – It works! Comment mis à jour!
- **200** – This Id is empty, new comment created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Comments – Patch a comment

Replace some data of an existing comment based on a given body and Id.

Method: **Patch**
Endpoint: `/api/comments/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Comment_Content": "Updated comment content"
}
```

Example: `http://localhost:8080/api/comments/7`

Response:

- **200** – Patch success! Comment updated!
- **200** – This Id is empty, new comment created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Comments – Delete a comment Delete a comment based on a given Id.

Method: **Delete**
Endpoint: `/api/comments/{myId}`
myId int Required
Example: `http://localhost:8080/api/comments/1`

Response:

- **200** – It works! Comment supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).

Tokens – Get all tokens

Return in a JSON all the tokens in the DB.

Method: **Get**
Endpoint: `/api/tokens`
Example: `http://localhost:8080/api/tokens`

Response:

- **200** – All the JSON tokens.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Tokens – Get a token by Id

Return in a JSON a token corresponding to the passed Id.

Method: **Get**
Endpoint: `/api/tokens/{myId}`
myId int Required
Example: `http://localhost:8080/api/tokens/4`

Response:

- **200** – A JSON of the token.
- **400** – Invalid Id, Error.
- **404** – Invalid endpoint.
- **503** – Error, could not send a request (the API is offline).

Tokens – Post a token

Create a token based on a given body.

Method: **Post**
Endpoint: `/api/tokens`
Body JSON format required:

```
json

{
    "User_Id": 7,
    "Token_Value": "token_value"
}
```

Example: `http://localhost:8080/api/tokens`

Response:

- **200** – It works! Post effectué!
- **400** – Error during post: {error}.
- **400** – Invalid JSON format.
- **503** – Error, could not send a request (the API is offline).

Tokens – Put a token

Replace all the data of an existing token or, if the token doesn't exist, create a token based on a given body and Id.

Method: **Put**
Endpoint: `/api/tokens/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "User_Id": 9,
    "Token_Value": "new_token_value"
}
```

Example: `http://localhost:8080/api/tokens/7`

Response:

- **200** – It works! Token mis à jour!
- **200** – This Id is empty, new token created.
- **400** – No or bad body sent: {error}.
- **503** – Error, could not send a request (the API is offline).

Tokens – Patch a token

Replace some data of an existing token based on a given body and Id.

Method: **Patch**
Endpoint: `/api/tokens/{myId}`
myId int Required
Body JSON format required:

```
json

{
    "Token_Value": "updated_token_value"
}
```

Example: `http://localhost:8080/api/tokens/8`

Response:

- **200** – Patch success! Token updated!
- **200** – This Id is empty, new token created.
- **400** – Bad body.
- **400** – Invalid Id or no rows affected.
- **503** – Error, could not send a request (the API is offline).

Tokens – Delete a token

Delete a token based on a given Id.

Method: **Delete**
Endpoint: `/api/tokens/{myId}`
myId int Required
Example: `http://localhost:8080/api/tokens/9`

Response:

- **200** – It works! Token supprimé!
- **400** – Invalid Id.
- **503** – Error, could not send a request (the API is offline).