

# **Manuel d'utilisation du projet Web**

## **(Partie I serveur)**

Groupe 3

Pierre ROCHET  
Siyu WANG  
Chen SUN  
Shuai GAO  
Alexandre IVANOV

Master TAL M2  
INALCO

## Sommaire

<b>Lancement de l'application</b>	<b>2</b>
Système Linux	2
Système macOS	2
<b>Requêtes via postman</b>	<b>3</b>
Afficher des données	4
Requête GET	4
Ajouter des données	4
Requête PUT	4
Requête PUT avec erreur	5
Modifier des données	6
Requête POST	6
Requête POST avec erreur	6
Supprimer des données	7
Requête DELETE	7
Requête DELETE avec erreur	8

# 1. Lancement de l'application

Cette application, hébergée sur un server Nginx, permet d'exploiter les données clients et les rendre accessibles sur un Postman (web client).

## 1.1. Système Linux

**Requis :**

- nginx : <http://hg.nginx.org/nginx.org>
- gunicorn : `pip install gunicorn`
- postman : <https://www.postman.com/>

**Etape 0 :** Créer un environnement virtuel python (optionnel).

Commandes : `source venv/bin/activate`

**Etape 1 :** Installer les dépendances.

Commandes : `pip3 install -r requirements.txt`

**Etape 2 :** Mettre le fichier *ceptyconsultant.conf* dans le dossier sites-available de nginx puis créer le chemin symbolique dans le dossier sites-enabled.

Redémarrer ensuite nginx.

Commandes : `sudo service nginx start`

**Etape 3 :** Lancer le fichier *gunicorn\_cmd.sh* qui contient la commande de lancement de l'application par gunicorn.

Commandes : `sh gunicorn_cmd.sh`

**Etape 4 :** Aller sur postman pour faire les requêtes.

## 1.2. Système macOS

**Requis :**

- homebrew pour installer nginx : `brew install nginx`
- gunicorn : `pip3 install gunicorn`
- postman : téléchargeable via le site <https://www.postman.com/downloads/>

**Étape 0 :** Créer un environnement virtuel python : `pipenv shell`

**Étape 1 :** Installer les dépendances : `pip3 install -r requirements.txt`

**Étape 2 :** Mettre le fichier *ceptyconsultant.conf* dans le dossier sites-available de nginx: le chemin par défaut de nginx pour Mac OS est : `usr/local/etc/nginx` et il faut créer manuellement le dossier `sites-available` et `sites-enabled`.

Puis créer le chemin symbolique dans le dossier sites-enabled :

```
ln -s /usr/local/etc/nginx/sites-available/ceptyconsultant.conf  
/usr/local/etc/nginx/sites-enabled/ceptyconsultant.conf
```

**Étape 3 :** Tester les configurations de nginx avant de démarrer : `sudo nginx -t`

Reload après chaque modification de conf : `sudo nginx -s reload`

Redémarrer nginx : `sudo brew services restart nginx`

Pour regarder le status de nginx : `ps aux | grep nginx`

**Étape 4 :** Lancer le fichier `gunicorn_cmd.sh` qui contient la commande de lancement de l'application par gunicorn : `sh gunicorn_cmd.sh`

**Étape 5 :** Démarrer Postman pour faire des requêtes

## 2. Requêtes via postman

Toutes les requêtes s'exécutent à l'adresse <http://www.localhost/ceptyconsultant.local/>.

Pour effectuer une requête, il faut au préalable se connecter avec des identifiants valides.

Des identifiants différents sont disponibles pour chaque collaborateur présent dans le fichier [LISTE\\_COLLABORATEURS.txt](#)

Les démonstrations nous avons faites au plus simple pour créer les comptes des utilisateurs :

→ Le Username est le prénom de l'utilisateur

→ Le Password est le nom de l'utilisateur

mais les identifiants peuvent être modifiés facilement avec des mots de passe beaucoup plus complexes.

Allez dans l'onglet **Authorization** choisissez le type **Basic Auth** puis entrez les identifiants suivant :

→ Username : Thomas

→ Password : Mikolov

On obtient le résultat ci-dessous :

Params **Authorization** Headers (10) Body Pre-request Script Tests Settings

**TYPE**

Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Preview Request

Username: Thomas

Password: Mikolov

☒ Show Password

## 2.1. Afficher des données

### ❏ Requête GET

Il s'agit de la requête la plus simple, elle renvoie simplement le contenu de nos données clients c'est à dire le contenu du fichier `DONNEES_CLIENT.txt`

GET http://www.localhost/ceptyconsultant.local/ Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 112ms Size: 34.52 KB Save Response

Pretty Raw Preview Visualize BETA JSON

```

1 {
2   "contributions": {
3     "data": [
4       {
5         "article_id": "0facf001-cb58-42c5-82b8-cd2dd2099967",
6         "contrib_data": "a_sound_1-2019-08-26T111407.675Z-.wav",
7         "contrib_name": "a",
8         "contrib_path": "https://ntealan.net/soundcontrib/",
9         "contrib_type": "sound",
10        "dico_id": "yb_fr_3031",
11        "last_update": "2019-08-26 11:15:38.811000",
12        "ntealan": true,
13        "public_id": "94ec363b-0537-4929-820e-b0559c57ab4d",
14        "user_id": "5184066e-8d5d-4809-1faa-60e384473935",
15        "user_name": "Bergier",
16        "validate": false
17      },
18    ]
19  }
20 }

```

## 2.2. Ajouter des données

### ❏ Requête PUT

Cette requête permet d'ajouter de nouvelles données saisies par l'utilisateur.

- sélectionner le type de requête **PUT**
- aller dans l'onglet **body**
- choisir le langage json et le format **raw**
- écrire le contenu à ajouter sous format **json**
- lancer la requête

**Remarque** : Seule la clé **article\_id** est obligatoire.

Si tout se passe bien, les nouvelles données seront ajoutées à la liste **data** de nos données clients et les données ajoutées seront renvoyées.

The screenshot shows a REST client interface with a PUT request to `http://www.localhost/ceptyconsultant.local/`. The request body is a JSON object with the following fields: `public_id` (NEW), `dico_id` (NEW), `user_id` (5184066e-8d5d-4809-1faa-60e384473935), `user_name` (Bergier), `article_id` (NEW), `contrib_type` (sound), `contrib_data` (a\_sound\_1-2019-08-26T111407.675Z-.wav), `contrib_path` (https://ntealan.net/soundcontrib/), `contrib_name` (a), `ntealan` (true), `validate` (false), and `last_update` (2019-08-26 11:15:38.811000). The response status is 200 OK, with a time of 129ms and a size of 514 B. The response body is a JSON object with the following fields: `article_id` (NEW), `contrib_data` (a\_sound\_1-2019-08-26T111407.675Z-.wav), `contrib_name` (a), `contrib_path` (https://ntealan.net/soundcontrib/), and `contrib_type` (sound).

## ❏ Requête PUT avec erreur

Si vous tentez d'ajouter du contenu avec un **article\_id** déjà présent dans les données alors une erreur est renvoyée.

The screenshot shows a REST client interface with a PUT request to `http://www.localhost/ceptyconsultant.local/`. The request body is the same JSON object as in the previous screenshot. The response status is 400 BAD REQUEST, with a time of 158ms and a size of 210 B. The response body is a JSON object with the following field: `error` (article\_id already exists).

## 2.3. Modifier des données

### ❑ Requête POST

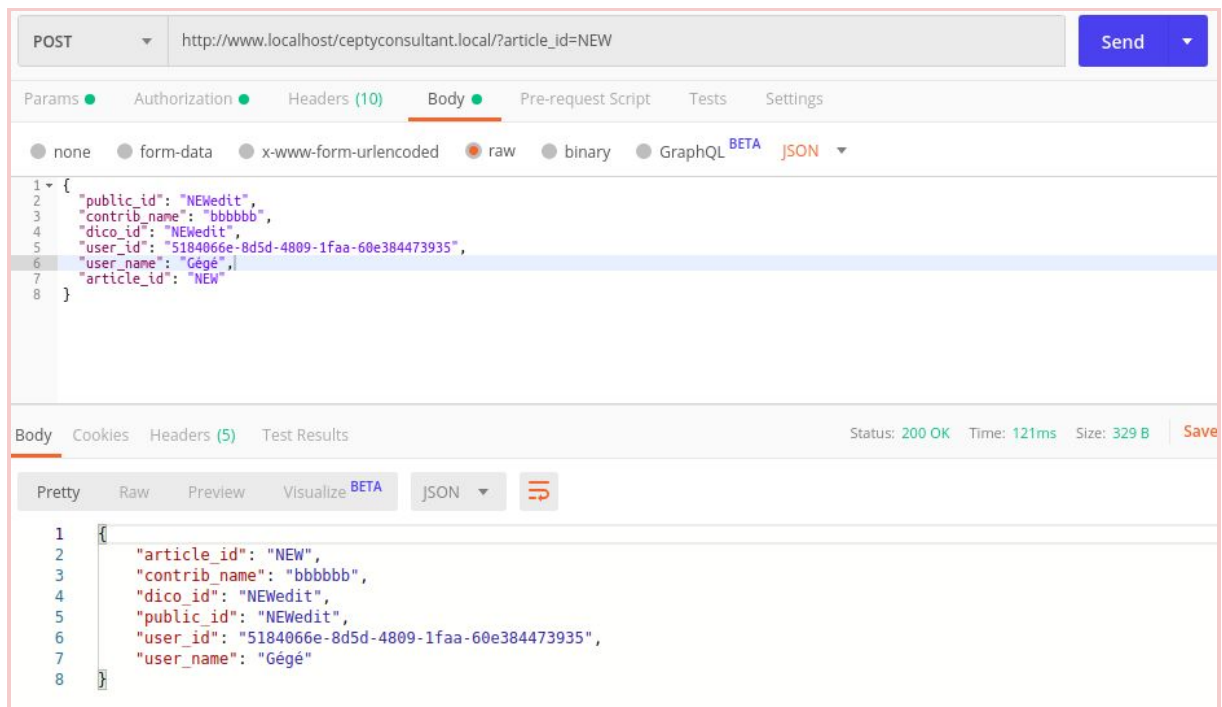
Pour modifier des données:

- sélectionner le type de requête **POST**
- aller dans l'onglet **Params**
- le nom de la clé sera toujours **article\_id**, la valeur est le nom de l'élément à modifier
- ensuite aller dans l'onglet **body**
- écrire les données que vous voulez modifier en format **json**.

Remarque : Si une donnée n'a pas de clé existante alors elle sera créée, si une donnée à une clé existante alors la valeur sera modifiée

- lancer la requête.

Si tout se passe bien, l'élément des données ayant été modifié est renvoyé :



### ❑ Requête POST avec erreur

Si les données ne contiennent pas l'**article\_id** indiqué, une erreur est alors renvoyée.

POST http://www.localhost/ceptyconsultant.local/?article\_id=IDNOEXIST Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL BETA JSON

```

1 {
2   "public_id": "NEWedit",
3   "contrib_name": "bbbbbb",
4   "dico_id": "NEWedit",
5   "user_id": "5184066e-8d5d-4809-1faa-60e384473935",
6   "user_name": "Gégé",
7   "article_id": "NEW"
8 }

```

Body Cookies Headers (5) Test Results Status: 404 NOT FOUND Time: 123ms Size: 203 B Save

Pretty Raw Preview Visualize BETA JSON

```

1 {
2   "error": "article_id not found"
3 }

```

## 2.4. Supprimer des données

### ❏ Requête DELETE

Pour supprimer des données :

- sélectionner le type de requête **DELETE**
- aller à l'onglet **Params**
- la clé sera toujours **article\_id**, la valeur est la nom de l'**article\_id** des données à supprimer
- exécuter la requête

Si rien n'est renvoyé alors la requête a fonctionné et l'élément contenant l'id indiqué est supprimé.

DELETE http://www.localhost/ceptyconsultant.local/?article\_id=NEW Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	article_id	NEW	
	Key	Value	Description

Body Cookies Headers (4) Test Results Status: 204 NO CONTENT Time: 102ms Size: 151 B Save

Pretty Raw Preview Visualize BETA JSON

```

1

```



## ❏ Requête DELETE avec erreur

Si les données ne contiennent pas l'**article\_id** indiqué, une erreur est alors renvoyée.

The screenshot shows a REST client interface with the following components:

- Request Bar:** Method: DELETE, URL: `http://www.localhost/ceptyconsultant.local/?article_id=NEW`, Send button.
- Params Tab:** Contains a table for Query Params.
- Query Params Table:**

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	article_id	NEW	
	Key	Value	Description

- Body Tab:** Status: 404 NOT FOUND, Time: 93ms, Size: 203 B, Save button.
- Response Body:** Pretty, Raw, Preview, Visualize BETA, JSON dropdown, and a JSON response.

```
1 {  
2   "error": "article_id not found"  
3 }
```