

# Rapport Générale de l'Api

Présenté par :  
Berenger BENAM

Professeur :  
Latyr Ndiaye

## 1. Programmer des API avec Python et Flask

### Objectifs

- Les API (Application Programming Interfaces) Web sont des outils permettant de rendre de l'information et des fonctionnalités accessibles via internet.

### Qu'est-ce qu'une API ?

- Une API web permet à de l'information et à des fonctionnalités d'être manipulées par des programmes informatiques via internet.
- Par exemple, avec l'API web Twitter, on peut écrire un programme dans un langage comme Python ou Javascript qui collecte des métadonnées sur les tweets.

### Qu'est-ce qu'une API ?

- Plus généralement, en programmation, le terme API (abréviation de Application Programming Interface) réfère à une partie d'un programme informatique conçue pour être utilisée ou manipulée par un autre programme, contrairement aux interfaces qui sont conçues pour être utilisées ou manipulées par des humains.
- Les programmes informatiques ont souvent besoin de communiquer entre eux ou avec le système d'exploitation sous-jacent et les API sont une façon de le faire.
- Dans la suite, toutefois, on se limitera aux API Web.

### Quand créer une API ?

- Si on dispose de données qu'on souhaite partager avec le monde entier, créer une API est une façon de le faire.
- Toutefois, les API ne sont pas toujours la meilleure façon de partager des données avec des utilisateurs.
- Si le volume des données qu'on souhaite partager est relativement faible, il vaut mieux proposer un « data dump » sous la forme d'un fichier JSON, XML, CSV ou Sqlite. Selon les ressources dont on dispose, cette approche peut être viable jusqu'à un volume de quelques Gigaoctets. Notons qu'on peut fournir à la fois un data dump et une API; à charge des utilisateurs de choisir ce qui leur convient le mieux.

### Terminologie des API

- Lorsqu'on construit ou qu'on utilise une API, on rencontre fréquemment les termes suivants :
- HTTP (HyperText Transfert Protocol) : c'est le principal moyen de communiquer de l'information sur Internet. HTTP implémente un certain nombre de « méthodes » qui disent dans quelle direction les données doivent se déplacer et ce qui doit en être fait. Les deux plus fréquentes sont GET, qui récupère les données à partir d'un serveur, et POST, qui envoie de nouvelles données vers un serveur.
- URL (Uniform Ressource Locator) : Adresse d'une ressource sur le web, comme <http://www-facultesciences.univ-ubs.fr/fr>. Une URL consiste en un protocole (<http://>),

un domaine ([www-facultesciences.univ-ubs.fr](http://www-facultesciences.univ-ubs.fr)), un chemin optionnel (/fr). Elle décrit l'emplacement d'une ressource spécifique, comme une page Web. Dans le domaine des API, les termes URL, request, URI, endpoint désignent des idées similaires. Dans la suite, on utilisera uniquement les termes URL et request, pour être plus clair. Pour effectuer une requête GET ou suivre un lien, il suffit de disposer d'un navigateur Web. JSON (JavaScript Object Notation) : c'est un format texte de stockage des données conçu pour être lisible à la fois par les êtres humains et les machines. JSON est le format le plus usuel des données récupérées par API, le deuxième plus usuel étant XML.

- **REST** (REpresentational State Transfer) : c'est une méthodologie qui regroupe les meilleures pratiques en termes de conception et d'implémentation d'APIs. Les API conçues selon les principes de la méthodologie REST sont appelées des API REST (REST APIs). Il y a toutefois de nombreuses polémiques autour de la signification exacte du terme. Dans la suite, on parlera plus simplement d'API Web ou d'API HTTP.

## Implémenter une API

- Dans la suite, on va voir comment créer une API en utilisant Python et le cadre de travail Flask.
- Notre exemple d'API portera sur un catalogue de livres allant au delà de l'information bibliographique standard.
- Plus précisément, en plus du titre et de la date de publication, notre API fournira la première phrase de chaque livre.

## Flask

- Flash est un cadre de travail (framework) Web pour Python. Ainsi, il fournit des fonctionnalités permettant de construire des applications Web, ce qui inclut la gestion des requêtes HTTP et des canevas de présentation.
- Nous allons créer une application Flask très simple, à partir de laquelle nous construirons notre API.

## Pourquoi Flask ?

- Python dispose de plusieurs cadre de développement permettant de produire des pages Web et des API.
- Le plus connu est Django, qui est très riche.
- Django peut toutefois être écrasant pour les utilisateurs non expérimentés.
- Les applications Flask sont construites à partir de canevas très simples et sont donc plus adaptées au prototypage d'APIs.

On commence par créer un nouveau répertoire sur notre ordinateur, qui servira de répertoire de projet et qu'on nommera projects.

- Les fichiers de notre projet seront stockés dans un sous-répertoire de projects, nommé api.

```
root@berenger:~# mkdir projects
root@berenger:~# cd projects/
root@berenger:~/projects# mkdir api
root@berenger:~/projects# ls
api
root@berenger:~/projects#
```

On se place dans le dossier api puis créer le fichier

```
root@berenger:~/projects/api# touch api.py
root@berenger:~/projects/api#
```

On lance ensuite **api.py** et on saisit le code suivant :

```
import flask
app = flask.Flask(__name__)
app.config["DEBUG"] = True
@app.route('/', methods=['GET'])
def home():
    return "<h1>Bienvenue dans la 1ere API developper par Berenger&Deranger ah oui oui !.</p>"
app.run()
```

On sauvegarde ensuite le programme sous le nom **api.py** dans le répertoire api précédemment créé.  
Pour l'exécuter, on exécute le programme

```
root@berenger:~/projects/api# python3 api.py
 * Serving Flask app 'api' (lazy loading)
 * Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Debug mode: on
 * Running on [http://127.0.0.1:5000/] (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 518-181-142
```

Il suffit de saisir le lien précédent dans un navigateur Web pour accéder à l'application.

- On a ainsi créé une application Web fonctionnelle.



### Comment fonctionne Flask ?

- Flask envoie des requêtes HTTP à des fonctions Python.
- Dans notre cas, nous avons appliqué un chemin d'URL ('/') sur une fonction : home. Flask exécute le code de la fonction et affiche le résultat dans le navigateur.
- Dans notre cas, le résultat est un code HTML de bienvenue sur le site hébergeant notre API.
- Le processus consistant à appliquer des URL sur des fonctions est appelé routage (routing).
- L'instruction :  
`@app.route('/', methods=['GET'])` apparaissant dans le programme indique à Flask que la fonction home correspond au chemin /.
- La liste methods (methods=['GET']) est un argument mot-clé qui indique à Flask le type de requêtes HTTP autorisées.
- On utilisera uniquement des requêtes GET dans la suite, mais de nombreuses applications Web utilisent à la fois des requêtes GET (pour envoyer des données de l'application aux utilisateurs) et POST (pour recevoir les données des utilisateurs).
- import Flask : Cette instruction permet d'importer la bibliothèque Flask, qui est disponible par défaut sous Anaconda.
- app = flask.Flask(\_\_name\_\_) : Crée l'objet application Flask, qui contient les données de l'application et les méthodes correspondant aux actions susceptibles d'être effectuées sur l'objet. La dernière instruction app.run() est un exemple d'utilisation de méthode.
- app.config[« DEBUG » = True] : lance le débogueur, ce qui permet d'afficher un message autre que « Bad Gateway » si il y a une erreur dans l'application.
- app.run() : permet d'exécuter l'application.

### Création de l'API

- Afin de créer l'API, on va spécifier nos données sous la forme d'une liste de dictionnaires Python.
- À titre d'exemple, on va fournir des données sur trois romans de Science-Fiction. Chaque dictionnaire contiendra un numéro d'identification, le titre, l'auteur, la première phrase et l'année de publication d'un livre.
- On introduira également une nouvelle fonction : une route permettant aux visiteurs d'accéder à nos données.
- créer un nouveau fichier **api1.py** par le code suivant :

- On exécute le code et on met à jour la fenêtre du navigateur, ce qui donne :

```
root@berenger:~/projects/api# python3 api1.py
 * Serving Flask app 'api1' (lazy loading)
 * Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Debug mode: on
 * Running on [http://127.0.0.1:5000/] (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 518-181-142
```

The screenshot shows a web browser window with the URL `127.0.0.1:5000` in the address bar. The page title is "L'avenir de EC2LT". Below the title, there is a small note: "Ecole Centrale des Logiciels Libres et de Télécommunications(EC2LT)." The main content of the page is the Python code for a Flask application. The code imports `flask`, creates a `Flask` app, sets `DEBUG` to `True`, and defines a list of books. It includes routes for the home page and a route to return all books. The code ends with `app.run()`.

```
import flask
from flask import request, jsonify
app = flask.Flask(__name__)
app.config["DEBUG"] = True
# Create some test data for our catalog in the form of a list of dictionaries.
books = [
    {'id': 0,
     'title': 'Le sommeil froid lui-même était sans rêve',
     'author': 'Berenger BENAM',
     'first_sentence': 'Le coldsleep lui-même était sans rêve.',
     'year_published': '2022'},
    {'id': 1,
     'title': 'la vie',
     'author': 'Morez Wan',
     'first_sentence': 'que le meilleur gagne.',
     'published': '1973'},
    {'id': 2,
     'title': 'Dhalgren',
     'author': 'Samuel OUYA',
     'first_sentence': 'seul travail qui libère .',
     'published': '1975'}
]
@app.route('/', methods=['GET'])
def home():
    return '''<h1>Distant Reading Archive</h1>
<p>A prototype API for distant reading of science fiction novels.</p>'''
# A route to return all of the available entries in our catalog.
@app.route('/api/v1/resources/books/all', methods=['GET'])
def api_all():
    return jsonify(books)
app.run()
```

Pour accéder à l'ensemble des données, il suffit de saisir dans le navigateur l'adresse : <http://127.0.0.1:5000/api/v1/resources/books/all>

The screenshot shows a terminal window with the command `python3 api1.py` being run. The output shows the Flask application starting and serving requests. It includes a warning about using it in production and logs several GET requests for the books endpoint.

```
root@berenger:~/projects/api# python3 api1.py
* Serving Flask app 'api1' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 518-181-142
127.0.0.1 - - [26/Apr/2022 21:52:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [26/Apr/2022 21:52:48] "GET /api/v1/resources/books/all HTTP/1.1" 200 -
127.0.0.1 - - [26/Apr/2022 21:52:49] "GET /api/v1/resources/books/all HTTP/1.1" 200 -
127.0.0.1 - - [26/Apr/2022 21:52:50] "GET /api/v1/resources/books/all HTTP/1.1" 200 -
```

- On a utilisé la fonction jsonify de Flask. Celle-ci permet de convertir les listes et les dictionnaires au format JSON.
- Via la route qu'on a créé, nos données sur les livres sont converties d'une liste de dictionnaires vers le format JSON, avant d'être fournies à l'utilisateur.
- A ce stade, nous avons créé une API fonctionnelle, bien que limitée.
- Dans la suite, nous verrons comment permettre aux utilisateurs d'effectuer des recherches spécifiques, par exemple à partir de l'identifiant d'un livre.

## Accéder à des ressources spécifiques

- En l'état actuel de notre API, les utilisateurs ne peuvent accéder qu'à l'intégralité de nos données; il ne peuvent spécifier de filtre pour trouver des ressources spécifiques.
- Bien que cela ne pose pas problème sur nos données de test, peu nombreuses, cela devient problématique au fur et à mesure qu'on rajoute des données.
- Dans la suite, on va introduire une fonction permettant aux utilisateurs de filtrer les résultats renvoyés à l'aide de requêtes plus spécifiques.

## Conception d'une API REST

- Action : Create, Verbe HTTP : POST, Chemin d'URL : /api/people, Description : définit une URL unique pour créer une nouvelle personne
- Action : Read, Verbe HTTP : GET, Chemin d'URL : /api/people, Description : définit une URL unique pour lire le catalogue de personnes
- Action : Read, Verbe HTTP : GET, Chemin d'URL : /api/people/Farrell, Description : définit une URL unique pour lire les données d'une personne particulière du catalogue
- Action : Update, Verbe HTTP : PUT, Chemin d'URL : /api/people/Farrell, Description : définit une URL unique pour mettre à jour les données d'une personne du catalogue
- Action : Delete, Verbe HTTP : DELETE, Chemin d'URL : /api/people/Farrell, Description : définit une URL unique pour supprimer les données d'une personne du catalogue

## Premier pas

- On va commencer par créer un serveur web très simple à l'aide du micro-cadre de développement Flask.
  - On crée un répertoire api, un sous-répertoire version\_1, puis un sous-répertoire templates où on sauvegarde les programmes suivants : Python (server.py)
- voici le contenu du code

```
from flask import (
    Flask,
    render_template
)
# Create the application instance
app = Flask(__name__, template_folder="templates")
# Create a URL route in our application for "/"
@app.route('/')
def home():
    """
    This function just responds to the browser ULR
    localhost:5000/
    :return:
    the rendered template 'home.html'
    """
    return render_template('home.html')
# If we're running in stand alone mode, run the application
if __name__ == '__main__':
    app.run(debug=True)
```

on crée le dossier **templates**

```
root@berenger:~/projects/api# mkdir templates
```

on édite le fichier home.html dans templates

```
root@berenger:~/projects/api# vim templates/home.html
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Application Api/HTML</title>
  </head>
<body>
  <font color="#B22222" size="3"><h1>Demonstration Api/HTML</h1></font>

  <h2>
    Bienvenue la famille Api
  </h2>
</body>
</html>
```

```
root@berenger:~/projects/api# python3 server.py
 * Serving Flask app 'server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 518-181-142
127.0.0.1 - - [26/Apr/2022 22:33:23] "GET / HTTP/1.1" 200 -
```



## Demonstration Api/HTML

Bienvenue la famille Api

Le programme HTML est nommé `home.html` plutôt que `index.html` comme il est d'usage, car `index.html` pose problème lorsqu'on importe le module Connexion, ce qu'on fera dans la suite.

- On obtient le résultat suivant en saisissant `http://127.0.0.1:5000` dans la barre du navigateur :

## CURL :

**Curl** est un langage de balisage comme le HTML, c'est-à-dire que le texte brut s'affiche en tant que texte ; il inclut également un langage de **programmation** orientée objet qui permet l'héritage multiple.

## cURL sous Linux: ce qu'il faut savoir pour débuter

Le logiciel libre cURL est l'un des projets open source les plus anciens et les plus appréciés. Écrit en C, ce programme sert à envoyer des données dans les réseaux d'ordinateurs. Le nom cURL renvoie à « client URL ». La licence ouverte accorde un droit d'utilisation à n'importe quelle fin. À l'heure actuelle, cURL est utilisé dans de très nombreux appareils.

## Qu'est-ce que cURL ?

Le logiciel cURL dispose de deux composantes. **libcurl**, la **bibliothèque du programme**, est l'épine dorsale du transfert de données et supporte les protocoles suivants :

DICT

- FILE
- FTP
- FTPS
- GOPHER

- HTTP
- HTTPS
- IMAP
- IMAPS
- LDAP
- LDAPS
- POP3
- POP3S
- RTMP
- RTSP
- SCP
- SFTP
- SMB
- SMBS
- SMTP
- SMTPS
- TELNET
- TFTP

Le **programme d'invite de commande cURL** agit quant à lui comme une interface basée sur du texte et interagit avec libcurl via l'invite de commande.

Ce programme est un **outil essentiel pour le développement Web** car il permet aux développeurs de communiquer directement avec les serveurs plutôt que de devoir passer par un navigateur. Les scripts basés sur les commandes cURL (également appelées « curl commands » en anglais) sont utilisés pour automatiser les processus ainsi que pour procéder à des tests et au débogage.

## Comment fonctionne cURL ?

Les deux composantes de cURL fonctionnent différemment.

Comment libcurl est-il utilisé ?

La bibliothèque du programme libcurl met à disposition des fonctionnalités permettant d'envoyer des données dans des réseaux d'ordinateurs. Il existe des « **bindings de langage** » (que l'on peut traduire par « liaisons de langage ») pour des douzaines de langages de programmation populaires. Ces bindings permettent à un grand nombre de logiciels communiquant avec des serveurs d'utiliser les fonctionnalités de libcurl.

## Comment cURL est-il utilisé ?

Le programme d'invite de commande cURL est utilisé dans le développement Web. La méthode la plus simple consiste à saisir les commandes cURL dans l'invite de commande. Avec le savoir-faire correspondant, vous pourrez ainsi **tester et déboguer** des serveurs et des [API](#).

Plutôt que de saisir manuellement les commandes dans l'interface de commande, vous pouvez également les regrouper dans des [scripts](#). Ceci permet de **standardiser et d'automatiser des opérations laborieuses**, qu'il s'agisse du chargement/du téléchargement de données, du remplissage automatique de formulaires ou de la mise en miroir de sites internet entiers.

La structure d'une commande cURL se présente généralement comme suit :

### - INSTALLATION :

on installe le paquet : **apt-get install curl**

```
root@berenger:/var/www/html/API# apt-get install curl
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
curl est déjà la version la plus récente (7.58.0-2ubuntu3.17).
Vous pouvez lancer « apt --fix-broken install » pour corriger ces problèmes.
Les paquets suivants contiennent des dépendances non satisfaites :
 openjdk-17-jdk : Dépend: openjdk-17-jdk-headless (= 17.0.3+7-0ubuntu0.18.04.1) mais 17.0.2+8-1~18.04 devra
 être installé
```

on peut vérifier la version installé par la commande :

```
root@berenger:/var/www/html/API# curl --version
curl 7.58.0 (x86_64-pc-linux-gnu) libcurl/7.58.0 OpenSSL/1.1.1 zlib/1.2.11 libidn2/2.0.4 libpsl/0.19.1 (+li
bidn2/2.0.4) nghttp2/1.30.0 librtmp/2.3
Release-Date: 2018-01-24
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp smb smbs smtp sm
tsp telnet tftp
Features: AsyncDNS IDN IPv6 Largefile GSS-API Kerberos SPNEGO NTLM NTLM_WB SSL libz TLS-SRP HTTP2 UnixSoc
ets HTTPS-proxy PSL
root@berenger:/var/www/html/API#
```

on crée un fichier :

```
root@berenger:/var/www/html/API# touch fichier
```

voici son contenu

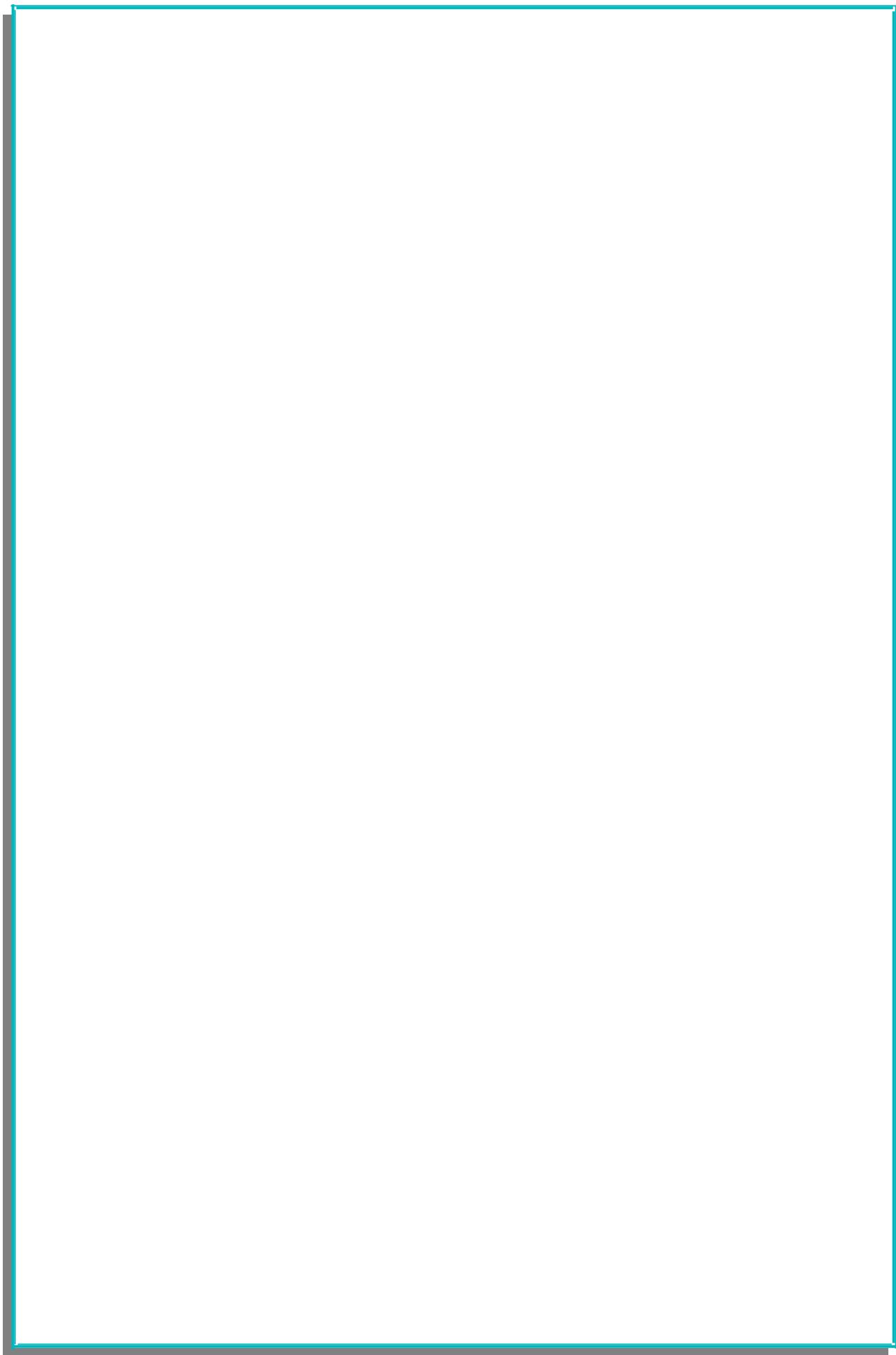
```
GNU nano 2.9.3                                     fichier

{
    "id": 3,
    "nom": "b",
    "prenom": "bnb",
    "email": "bnb@bg.sn",
    "password": "passer123/"}
```

on crée un compte au format simple pour pouvoir exécuter avec le client curl

```
root@berenger:/var/www/html/API# touch db.json
```

on crée aussi un fichier json



```
GNU nano 2.9.3                                     db.json

{
  "users": [
    {
      "id": 1,
      "nom": "Benam",
      "prenom": "Berenger",
      "email": "berengerbenam@gmail.com",
      "password": "passer"
    }
  ],
  "comptes": [
    {
      "id": 1,
      "nom": "Benam",
      "prenom": "Berenger",
      "email": "berengerbenam@gmail.com",
      "password": "passer",
      "solde": "20000"
    },
    {
      "id": 3,
      "nom": "Ben",
      "prenom": "neymar",
      "email": "psg@gmail.com",
      "password": "passer"
    },
    {
      "prenom": "Pirlo",
      "nom": "Maxime",
      "email": "pirlo@gmail.com",
      "password": "passer123",
      "solde": "20000",
      "id": 4
    }
  ]
}
```

#### TEST :

on lance deux terminaux un pour le **json** et l'autre pour exécuter la commande **CURL**

```
root@berenger:/var/www/html/API# json-server -p 5000 db.json
```

```
\{^_^\}/ hi!
```

```
Loading db.json
Done
```

#### Resources

```
http://localhost:5000/users
http://localhost:5000/comptes
```

#### Home

```
http://localhost:5000
```

```
Type s + enter at any time to create a snapshot of the database
```

on lance le curl

```

root@berenger:/var/www/html/API# curl -i http://localhost:5000/users
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Origin, Accept-Encoding
Access-Control-Allow-Credentials: true
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
X-Content-Type-Options: nosniff
Content-Type: application/json; charset=utf-8
Content-Length: 135
ETag: W/"87-L8FVlnP4MZ8CaK5pvC4UiA3j28"
Date: Mon, 02 May 2022 10:31:16 GMT
Connection: keep-alive
Keep-Alive: timeout=5

[
  {
    "id": 1,
    "nom": "Benam",
    "prenom": "Berenger",
    "email": "berengerbenam@gmail.com",
    "password": "passer"
  }
]root@berenger:/var/www/html/API#

```

on voit avec curl il affiche le compte json  
pour voir le 1er compte dans notre fichier db.json il suffit de copier cette url dans un navigateur

```
root@berenger:/var/www/html/API# json-server -p 5000 db.json
```

```

\{^_^\}/ hi!

Loading db.json
Done

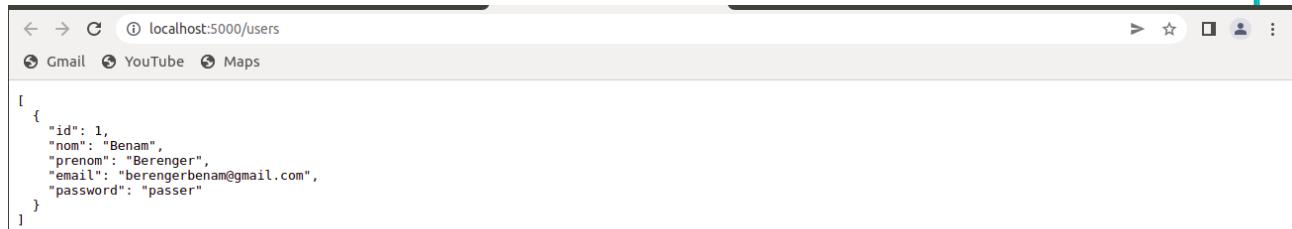
Resources
http://localhost:5000/users
http://localhost:5000/comptes

Home
http://localhost:5000

Type s + enter at any time to create a snapshot of the database
GET /users 200 6.744 ms - 135

```

## résultat



The screenshot shows a browser window with the URL `localhost:5000/users`. The page displays a single JSON object representing a user:

```
{
  {
    "id": 1,
    "nom": "Benam",
    "prenom": "Berenger",
    "email": "berengerbenam@gmail.com",
    "password": "passer"
  }
}
```

il affiche juste le 1<sup>er</sup> compte

-si on veut afficher tous les comptes il suffit de copier cette url

```
root@berenger:/var/www/html/API# json-server -p 5000 db.json
```

```
\{^_^\}/ hi!
```

```
Loading db.json  
Done
```

```
Resources
```

```
http://localhost:5000/users  
http://localhost:5000/comptes
```

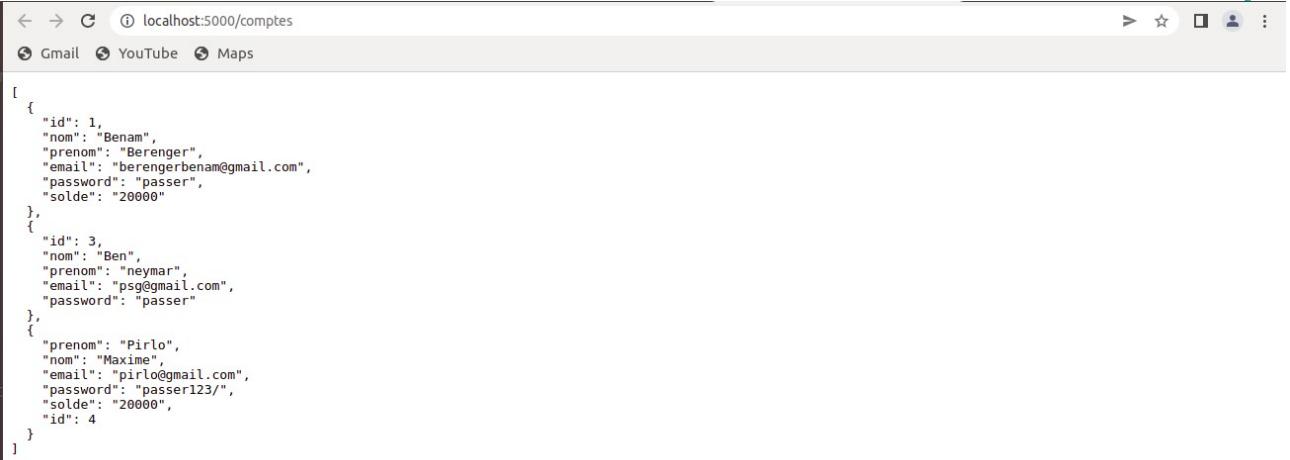
```
Home
```

```
http://localhost:5000
```

```
Type s + enter at any time to create a snapshot of the database
```

```
GET /users 200 6.744 ms - 135  
GET /users 200 4.562 ms - 135  
GET /comptes 304 3.170 ms - -
```

coté navigateur



```
[{"id": 1, "nom": "Benam", "prenom": "Berenger", "email": "berengerbenam@gmail.com", "password": "passer", "solde": "20000"}, {"id": 3, "nom": "Ben", "prenom": "neymar", "email": "psg@gmail.com", "password": "passer"}, {"id": 4, "prenom": "Pirlo", "nom": "Maxime", "email": "pirlo@gmail.com", "password": "passer123", "solde": "20000", "id": 4}]
```

on visualise tous les comptes

et on démarre le json puis lancer un autre terminal pour utiliser la méthode POST dans le curl

```
root@berenger:/var/www/html/API# curl -i -H "Content-Type:application/json" -X POST -d @fichier http://localhost:5000/users/
HTTP/1.1 201 Created
X-Powered-By: Express
Vary: Origin, X-HTTP-Method-Override, Accept-Encoding
Access-Control-Allow-Credentials: true
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Access-Control-Expose-Headers: Location
Location: http://localhost:5000/users//3
X-Content-Type-Options: nosniff
Content-Type: application/json; charset=utf-8
Content-Length: 98
ETag: W/"62-crYIxdrTI3ox5lXSRQHyWz1tX8"
Date: Mon, 02 May 2022 10:45:59 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "id": 3,
  "nom": "b",
  "prenom": "bnb",
  "email": "bnb@bg.sn",
  "password": "passer123/"
root@berenger:/var/www/html/API#
```

quand on lance le curl il va créer le compte qui affiche sur le terminal dans db.json

```
GNU nano 2.9.3                               db.json
[{"users": [
    {"id": 3,
     "nom": "b",
     "prenom": "bnb",
     "email": "bnb@bg.sn",
     "password": "passer123/"},
    ],
 "comptes": [
    {
        "id": 1,
        "nom": "Benam",
        "prenom": "Berenger",
        "email": "berengerbenam@gmail.com",
        "password": "passer",
        "solde": "20000"
    },
    ]
},                                              Lecture de 36 lignes 1
```

on voit le compte a été ajouté

- on essaye avec la méthode PUT on édite le fichier et changer le compte

```
GNU nano 2.9.3                               fichier                         Modifié
{
    "id": 3,
    "nom": "onelove",
    "prenom": "love",
    "email": "onelove@gmail.com",
    "password": "passer"
}
```

on crée le compte **onelove** et on laisse l'id 3

```
root@berenger:/var/www/html/API# curl -i -H "Content-Type:application/json" -X PUT -d @fichier http://localhost:5000/users/3
HTTP/1.1 200 OK
X-Powered-By: Express
Vary: Origin, Accept-Encoding
Access-Control-Allow-Credentials: true
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
X-Content-Type-Options: nosniff
Content-Type: application/json; charset=utf-8
Content-Length: 109
ETag: W/"6d-9JjBN5ihVoH2Z8+v4XCJH03r1A"
Date: Mon, 02 May 2022 11:44:28 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "id": 3,
    "nom": "onelove",
    "prenom": "love",
    "email": "onelove@gmail.com",
    "password": "passer"
}root@berenger:/var/www/html/API#
```

on peut voir ce que la méthode **PUT** fait sur l'autre terminal

```
root@berenger:/var/www/html/API# json-server -p 5000 db.json
\{^_^\} hi!

Loading db.json
Done

Resources
http://localhost:5000/users
http://localhost:5000/comptes

Home
http://localhost:5000

Type s + enter at any time to create a snapshot of the database
PUT /users/3 200 23.526 ms - 109
```

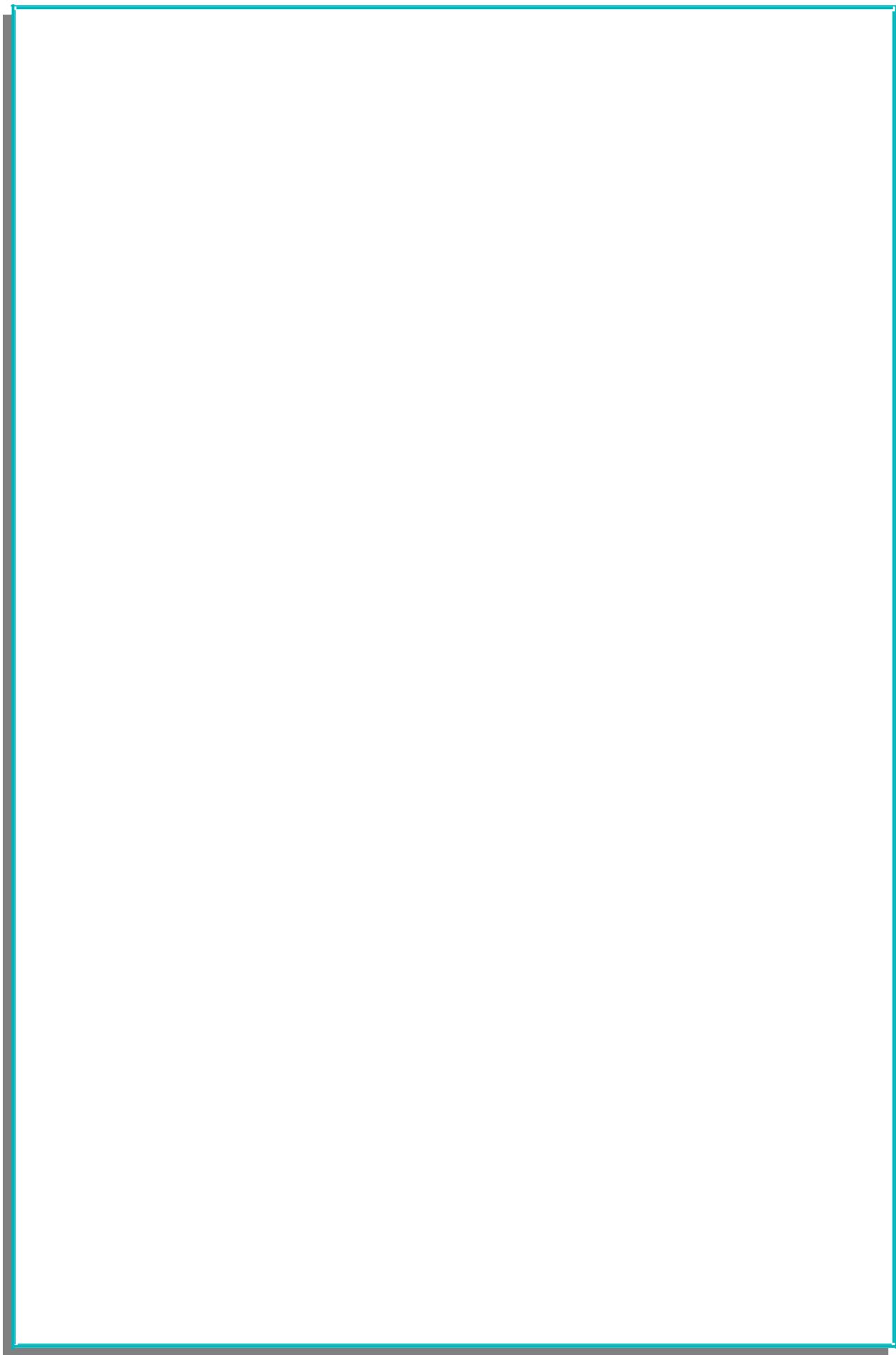
on test aussi la méthode GET

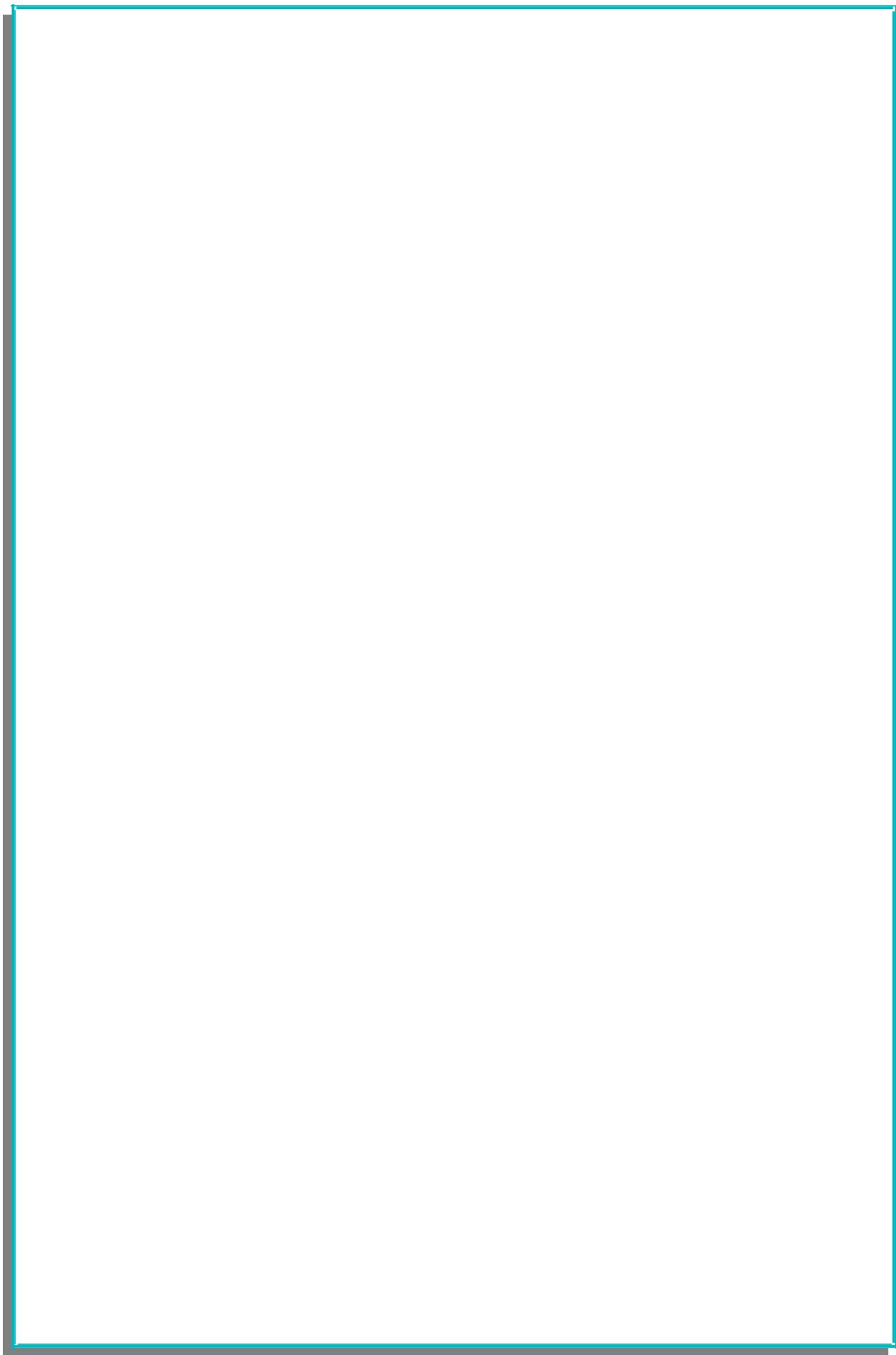
si on fait **cat db.son**

```
root@berenger:/var/www/html/API# cat db.json
{
  "users": [
    {
      "id": 3,
      "nom": "onelove",
      "prenom": "love",
      "email": "onelove@gmail.com",
      "password": "passer"
    }
  ],
  "comptes": [
    {
      "id": 1,
      "nom": "Benam",
      "prenom": "Berenger",
      "email": "berengerbenam@gmail.com",
      "password": "passer",
      "solde": "20000"
    },
    {
      "id": 3,
      "nom": "Ben",
      "prenom": "neymar",
      "email": "psg@gmail.com",
      "password": "passer"
    },
    {
      "prenom": "Pirlo",
      "nom": "Maxime",
      "email": "pirlo@gmail.com",
      "password": "passer123",
      "solde": "20000",
      "id": 4
    }
  ]
}
```

il a fait une mise à jours et le compte **onelove** est la

- on utilise la méthode **DELETE** pour supprimer le compte **onelove** qui a l'id 3





```
root@berenger:/var/www/html/API# curl -i -H "Content-Type:application/json" -X DELETE -d @fichier http://localhost:5000/users/3
HTTP/1.1 200 OK
X-Powered-By: Express
Var root@berenger:/var/www/html/API# json-server -p 5000 db.json
Acc
Cac
Pra
Exp
X-C Loading db.json
Con Done
Con
Con
ETa
Dat
Con
Kee Resources
http://localhost:5000/users
http://localhost:5000/comptes
{} Home
http://localhost:5000

Type s + enter at any time to create a snapshot of the database
DELETE /users/3 200 22.034 ms - 2
```

il supprime le compte **onlove** dont l'id égale 3

```
root@berenger:/var/www/html/API# cat db.json
{
  "users": [],
  "comptes": [
    {
      "id": 1,
      "nom": "Benam",
      "prenom": "Berenger",
      "email": "berengerbenam@gmail.com",
      "password": "passer",
      "solde": "20000"
    },
    {
      "id": 3,
      "nom": "Ben",
      "prenom": "neymar",
      "email": "psg@gmail.com",
      "password": "passer"
    },
    {
      "prenom": "Pirlo",
      "nom": "Maxime",
      "email": "pirlo@gmail.com",
      "password": "passer123",
      "solde": "20000",
      "id": 4
    }
  ]
}
root@berenger:/var/www/html/API#
```

- Dans cette partie on va créer un dossier dans API pour pouvoir gérer des codes html et php avec le fichier json

```
root@berenger:/var/www/html/API# mkdir doc_api_all
```

on se place dans le dossier **doc\_api\_all** puis créer un fichier

```
root@berenger:/var/www/html/API/doc_api_all# touch ajoutjson.html
```

voici le contenu du fichier **ajoutjson.html**

```
GNU nano 2.9.3                               ajoutjson.html                                Modifié

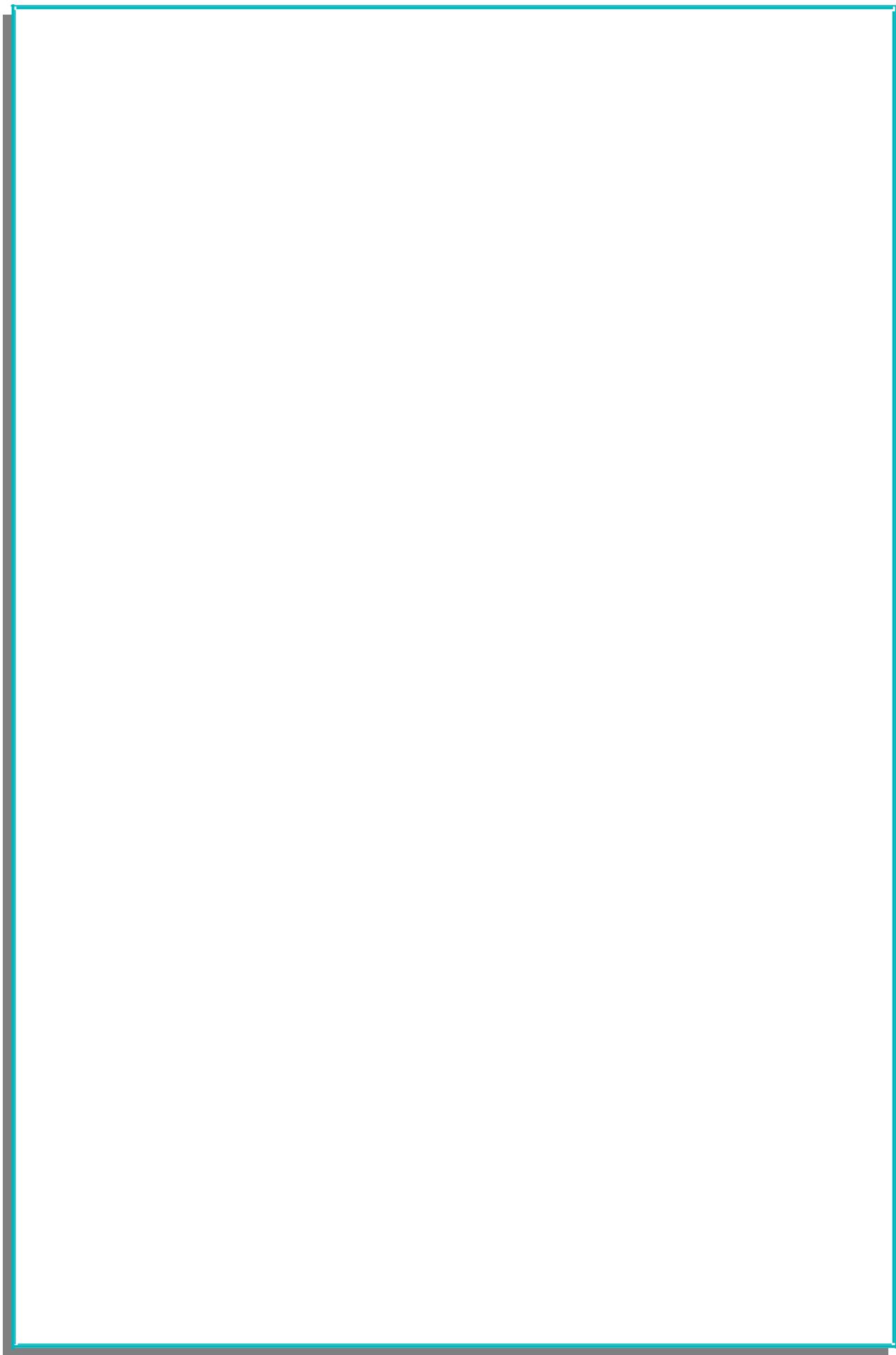
<html>
  <head>
<title>Formulaire ajout compte</title>
  </head>
<body>

</body>
<font color="#B22222" size="3"><h1>Bienvenue dans mon formulaire html</h1></font>

<form method="POST" action="ajoutjson.php">
  prenom: <input type="text" name="prenom"/><br>
  nom: <input type="text" name="nom" required="required"/><br>
  email: <input type="text" name="email"/><br>
  password: <input type="text" name="password"/><br>
  <input type="submit" value="valider"/><br><br>
</form>

</html>
```

puis créer le fichier **ajoutjson.php**



```

GNU nano 2.9.3                               ajoutjson.php

?php
$prenom = $_POST["prenom"];
$nom = $_POST["nom"];
$email = $_POST["email"];
$password = $_POST["password"];
$url = "url_to_post";
$data = array("prenom"=>$prenom,"nom"=>$nom,"email"=>$email,"password"=>$password);
$donnee = json_encode($data);
$curl = curl_init($url);

curl_setopt($curl,CURLOPT_POST,1);
curl_setopt($curl,CURLOPT_POSTFIELDS,$donnee);
curl_setopt($curl,CURLOPT_RETURNTRANSFER,1);
curl_setopt($curl,CURLOPT_HTTPHEADER,array("Content-Type:application/json"));
//curl_setopt($curl,CURLOPT_URL,"http://api.ec2lt.sn/compte");
curl_setopt($curl,CURLOPT_URL,"http://localhost:5000/users");

$result = curl_exec($curl);
curl_close($curl);
if(result==True){
echo "Insertion réussie";
} else{
echo "insertion echoué";
}
echo $donnee;
?>

```

## TEST :

Bienvenue dans mon formulaire html

prenom: Benoit16  
nom: Mandja  
email: wandet.morez@gmail.com  
password: passer123/

**cliquer sur valider**

```

localhost/API/doc_api_all/ajoutjson.php
insertion réussie {"prenom":"Benoit16","nom":"Mandja","email":"wandet.morez@gmail.com","password":"passer123v"}

```

on peut voir l'insertion réussie

```

root@berenger:/var/www/html/API# json-server -p 5000 db.json
\{^_^\}/ hi!
Loading db.json
Done

Resources
http://localhost:5000/users
http://localhost:5000/comptes

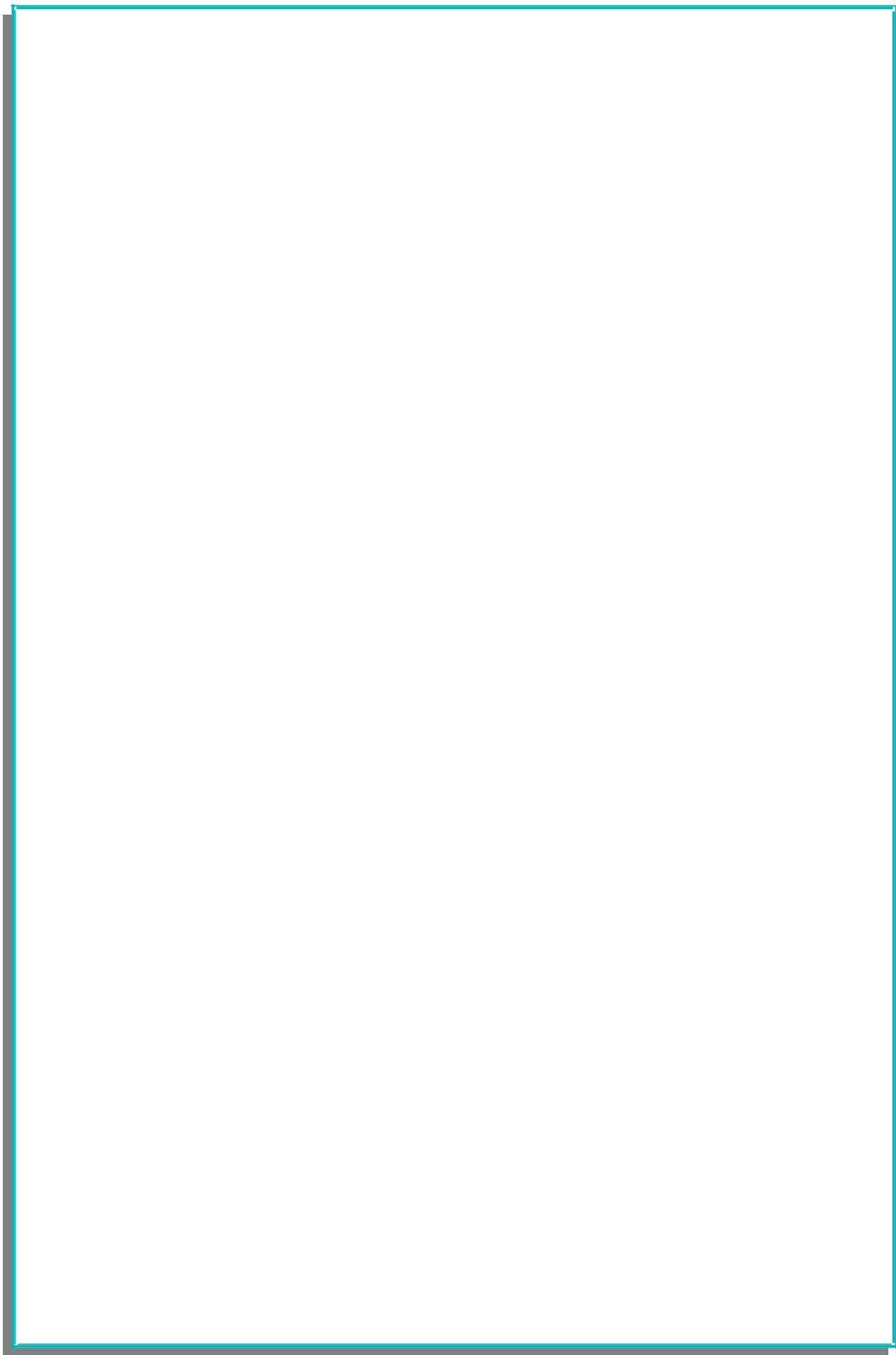
Home
http://localhost:5000

Type s + enter at any time to create a snapshot of the database
POST /users 201 50.983 ms - 121

```

sur cette capture on voit le code **201** qui signifie on vient de faire une insertion en utilisant la méthode **POST**

si on fait **cat db.json**



```
{  
  "users": [  
    {  
      "prenom": "Benoit16",  
      "nom": "Mandja",  
      "email": "wandel.morez@gmail.com",  
      "password": "passer123/",  
      "id": 1  
    }  
  ],  
  "comptes": [  
    {  
      "id": 1,  
      "nom": "Benam",  
      "prenom": "Berenger",  
      "email": "berengerbenam@gmail.com",  
      "password": "passer",  
      "solde": "20000"  
    },  
    {  
      "id": 3,  
      "nom": "Ben",  
      "prenom": "neymar",  
      "email": "psg@gmail.com",  
      "password": "passer"  
    },  
    {  
      "prenom": "Pirlo",  
      "nom": "Maxime",  
      "email": "pirlo@gmail.com",  
      "password": "passer123/",  
      "solde": "20000",  
      "id": 4  
    }  
  ]  
}  
root@berenger:/var/www/html/API#
```

on voit le compte **Benoit16**

on ajoute le compte toto

localhost/API/doc\_api\_all/ajoutjson.html

### Bienvenue dans mon formulaire html

prenom:   
nom:   
email:   
password:

valider

localhost/API/doc\_api\_all/ajoutjson.php

```
insertion reussie("prenom":"toto","nom":"nama","email":"toto.nam@gmail.com","password":"passer123V")
```

coté json

```
root@berenger:/var/www/html/API# json-server -p 5000 db.json
\{^_^\}/ hi!

Loading db.json
Done

Resources
http://localhost:5000/users
http://localhost:5000/comptes

Home
http://localhost:5000

Type s + enter at any time to create a snapshot of the database
POST /users 201 17.636 ms - 111
```

il ajoute le compte toto

```
root@berenger:/var/www/html/API# cat db.json
{
  "users": [
    {
      "prenom": "Benoit16",
      "nom": "Mandja",
      "email": "wandel.morez@gmail.com",
      "password": "passer123/",
      "id": 1
    },
    {
      "prenom": "toto",
      "nom": "nama",
      "email": "toto.nam@gmail.com",
      "password": "passer123/",
      "id": 2
    }
  ],
  "comptes": [
    {
      "id": 1,
      "nom": "Benam",
      "prenom": "Berenger",
      "email": "berengerbenam@gmail.com",
      "password": "passer"
    }
  ]
}
```

- Dans cette partie on va créer notre api

```
root@berenger:/var/www/html/API# touch apitrans.py
```

voici le contenu du fichier **apitrans.py**

```
from flask import Flask
from flask import jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_restx import Api, fields, Resource, reqparse
from flask_jwt_extended import JWTManager
from flask_jwt_extended import create_access_token, create_refresh_token, jwt_required, get_jwt_identity

# c est l etape de declaration de notre application (api)
app = Flask(__name__)
api = Api(app)

jwt = JWTManager(app)

# c est la partie ou on declare les parametre des decoration qui vont decorer nos methodes(nos ressources)
parser = reqparse.RequestParser()
parser.add_argument('prenom', help = 'le prenom du user', required = True)
parser.add_argument('nom', help = 'nom', required = True)
parser.add_argument('email', help = 'email', required = True)
parser.add_argument('password', help = 'mot de passe', required = True)

# parser compte
parser_compte = reqparse.RequestParser()
parser_compte.add_argument('prenom', help = 'le prenom', required = True)
parser_compte.add_argument('nom', help = 'nom', required = True)
parser_compte.add_argument('email', help = 'email', required = True)
parser_compte.add_argument('numero', help = 'numero', required = True)
parser_compte.add_argument('password', help = 'not de passe', required = True)
parser_compte.add_argument('typecompte', help = 'typecompte', required = True)
parser_compte.add_argument('solde', help = 'solde', required = True)

# parser transferer

parser_transfert = reqparse.RequestParser()
parser_transfert.add_argument('numero_fournisseur', help = 'numero_fournisseur', required = True)
parser_transfert.add_argument('numero_client', help = 'numero_client', required = True)
parser_transfert.add_argument('credit', help = 'credit', required = True)

# parser vente
parser_vente = reqparse.RequestParser()
parser_vente.add_argument('numero_fournisseur', help = 'numero_fournisseur', required = True)
parser_vente.add_argument('nom', help = 'nom', required = True)
parser_vente.add_argument('prenom', help = 'prenom', required = True)
parser_vente.add_argument('email', help = 'email', required = True)
parser_vente.add_argument('password', help = 'password', required = True)
parser_vente.add_argument('numero_revendeur', help = 'numero_revendeur', required = True)
parser_vente.add_argument('credit', help = 'credit', required = True)

# parser ajoutsolde
parser_rechargecompte = reqparse.RequestParser()
parser_rechargecompte.add_argument('numero_fournisseur', help = 'numero_fournisseur', required = True)
parser_rechargecompte.add_argument('credit', help = 'credit', required = True)

app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://berenger:Passer123@localhost/transaction'

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

def save_to_db(self):
    db.session.add(self)
    db.session.commit()

@classmethod
def find_by_email(cls, email):
    return cls.query.filter_by(email = email).first()

@classmethod
def return_all(cls):
    def to_json(x):
        return {
            'id': x.id,
            'prenom': x.prenom,
            'nom': x.nom,
            'email': x.email,
            'numero': x.numero,
            'typecompte': x.typecompte,
            'solde': x.solde
        }
    return {'comptes': list(map(lambda x: to_json(x), Compte.query.all()))}

@api.route('/Utilisateurs')
class UserRegistration(Resource):
    @api.doc(parser=parser)
    def post(self):
        data = parser.parse_args()
        if Users.find_by_email(data['email']):
            try:
                new_user.save_to_db()
                access_token = create_access_token(identity = data['email'])
                return {
                    'message': 'User {} was created'.format(data['prenom']),
                    'access_token': access_token
                }
            except:
                return {'message': 'something went wrong'}
```

```

class ConsulterSolde(Resource):
    def get(self,numero):
        return return_one(numero)

    #pour supprimer un compte
    @api.route('/SupprimerCompte/<int:id>')
    class SupprimerSolde(Resource):
        def delete(self,id):
            return delete_one(id)

    #pour afficher les comptes
    @api.route('/AfficheComptes')
    class Allcompte(Resource):
        def get(self):
            return Compte.return_all()

    #pour la methode transferer
    def transfert(numero_fournisseur,numero_client,credit):
        compte_fournisseur=Compte.query.filter_by(numero=numero_fournisseur).first()
        if compte_fournisseur == None:
            return {'message':'le numero fournisseur n existe pas'}
        elif (float(compte_fournisseur.solde) > float(credit)):
            solde_init = compte_fournisseur.solde
            nouveau_solde_fournisseur = float(solde_init) - float(credit)
            compte_client = Compte.query.filter_by(numero=numero_fournisseur).first()
            if compte_fournisseur == None:
                return {'message': 'le numero du client n existe pas'}
            else:
                solde_init_client=compte_client.solde
                nouveau_solde_client = float(solde_init_client) + float(credit)
                compte_fournisseur.solde=nouveau_solde_fournisseur

                compte_client.solde=nouveau_solde_client
                db.session.commit()
                return {'message':'votre transfere est effectué avec succès donc votre nouveau solde est maintenant :{} FCFA'.format(compte_fournisseur.solde)}
        else:
            return {'message':'votre credit est inferieur'}

    #pour la ressource transfert_client
    @api.route('/Transfert_Client')
    class Transfert(Resource):
        @api.doc(parser=parser_transfert)
        def put(self):
            data = parser_transfert.parse_args()
            return transfert(data['numero_fournisseur'],data['numero_client'],data['credit'])

    #pour la methode ventes
    def vente(nom,prenom,email,password,numero_fournisseur,numero_revendeur,credit):
        compte_fournisseur=Compte.query.filter_by(numero=numero_fournisseur).first()
        if compte_fournisseur == None:
            return {'message':'le Fournisseur n existe pas'}
        elif (float(compte_fournisseur.solde) > float(credit)):
            solde_init = compte_fournisseur.solde
            nouveau_solde_fournisseur = float(solde_init) - float(credit)
            compte_revendeur = Compte.query.filter_by(numero=numero_revendeur).first()
            if compte_revendeur == None:
                return {'message': 'le client n existe pas'}
            else:
                solde_init_revendeur=compte_revendeur.solde
                nouveau_solde_revendeur = float(solde_init_revendeur) + float(credit)
                compte_fournisseur.solde=nouveau_solde_fournisseur
                compte_revendeur.solde=nouveau_solde_revendeur
                db.session.commit()

            return {'message':'vous avez effectué un achat donc il vous reste maintenant :{} FCFA'.format(compte_fournisseur.solde)}
        else:
            return {'message':'votre credit inferieur'}

    #pour la ressource ventes
    @api.route('/Vente_Credit')
    class Vente(Resource):
        @api.doc(parser=parser_vente)
        def put(self):
            data = parser_vente.parse_args()
            return vente(data['nom'],data['prenom'],data['email'],data['password'],data['numero_fournisseur'],data['numero_revendeur'],data['credit'])

    #pour la methode ajout de credit pour l administrateur
    def rechargecompte(numero_fournisseur,credit):
        compte_fournisseur=Compte.query.filter_by(numero=numero_fournisseur).first()
        if (numero_fournisseur == "1000"):
            solde_init_fournisseur=compte_fournisseur.solde
            nouveau_solde_fournisseur = float(solde_init_fournisseur) + float(credit)
            compte_fournisseur.solde=nouveau_solde_fournisseur
            db.session.commit()
            return {'message':'vous avez recharger votre compte et le nouveau solde est maintenant :{} FCFA'.format(compte_fournisseur.solde)}

        else:
            return {'message': 'tu n es pas fournisseur'}

    @api.route('/RechargeCompte')
    class Rechargecompte(Resource):
        @api.doc(parser=parser_rechargecompte)
        def put(self):
            data = parser_rechargecompte.parse_args()
            return rechargecompte(data['numero_fournisseur'],data['credit'])

if __name__ == '__main__':
    app.run(port= 8080, debug = True, host='0.0.0.0')

```

on importe l'api (parce qu'elle contient tous les tables)

```
root@berenger:/var/www/html/API# python3
Python 3.6.9 (default, Mar 15 2022, 13:55:28)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from apitrans import db
>>> db.create_all()
>>> 
```

on va créer un utilisateur et lui attribue tous les priviléges

```
MariaDB [(none)]> create database transaction;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> use transaction;
Database changed
MariaDB [transaction]> grant all privileges on transaction.* to berenger identified by 'Passer123/';
Query OK, 0 rows affected (0.00 sec)

MariaDB [transaction]> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

on peut faire show tables pour visualiser les tables

```
MariaDB [transaction]> show tables;
+-----+
| Tables_in_transaction |
+-----+
| compte
| users
+-----+
2 rows in set (0.00 sec)
```

on constate sur l'api elle crée deux tables dans la base de donnée transaction

```
MariaDB [transaction]> desc compte;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra
+-----+-----+-----+-----+-----+
| id | int(11) | NO | PRI | NULL | auto_increment |
| prenom | varchar(50) | YES | | NULL |
| nom | varchar(50) | YES | | NULL |
| email | varchar(50) | YES | | NULL |
| numero | varchar(50) | YES | | NULL |
| password | varchar(50) | YES | | NULL |
| typecompte | varchar(50) | YES | | NULL |
| solde | varchar(50) | YES | | NULL |
+-----+-----+-----+-----+-----+
8 rows in set (0.01 sec)
```

voici la description de la table **compte** et idem pour **users**

```
MariaDB [transaction]> desc users;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra
+-----+-----+-----+-----+-----+
| id | int(11) | NO | PRI | NULL | auto_increment |
| prenom | varchar(50) | YES | | NULL |
| nom | varchar(50) | YES | | NULL |
| email | varchar(50) | YES | | NULL |
| password | varchar(50) | YES | | NULL |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

TEST :

sur le terminal on lance le programme

```
root@berenger:/var/www/html/API# python3 apitrans.py
 * Serving Flask app 'apitrans' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://192.168.1.14:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 769-411-174
```

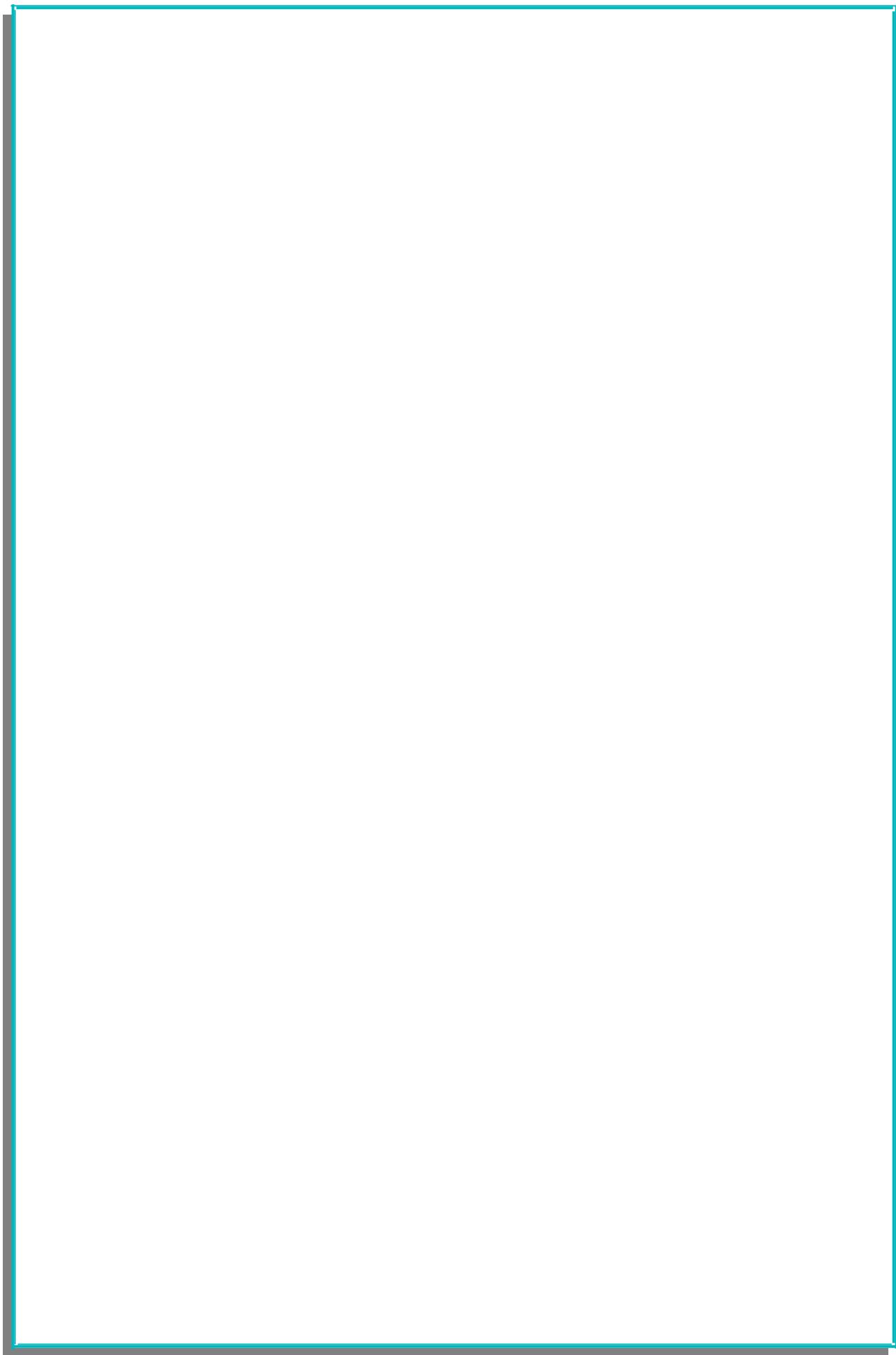
puis copier l'url

The screenshot shows a web browser window with the URL `192.168.1.6:8080` in the address bar. The page title is "API 1.0". Below the title, there is a note "[ Base URL: / ]" and a link to "/swagger.json". The main content area is titled "default Default namespace". It lists several API endpoints:

- GET /AfficheComptes
- POST /Compte ressource pour creer un compte
- GET /ConsulterSolde/{numero}
- PUT /RechargeCompte
- DELETE /SupprimerCompte/{id}
- PUT /Transfert\_Client
- POST /Utilisateurs
- PUT /Vente\_Credit

voici la page d'api pour le Fournisseur et il a tous les taches a effectués comme création de compte,ajouter un user etc...

- Le fournisseur va créer trois utilisateur Berenger,Nasry et Morez voir la capture ci-dessous



POST /compte ressource pour creer un compte

Cancel

Parameters

| Name                                       | Description                      |
|--|----------------------------------|
| prenom * required<br>string<br>(query)     | le prenom<br>Berenger            |
| nom * required<br>string<br>(query)        | nom<br>BENAM                     |
| email * required<br>string<br>(query)      | email<br>berengerbenam@gmail.com |
| numero * required<br>string<br>(query)     | numero<br>1000                   |
| password * required<br>string<br>(query)   | mot de passe<br>Passer123/       |
| typecompte * required<br>string<br>(query) | typecompte<br>Client             |

Non sécurisé | 192.168.1.252:8080

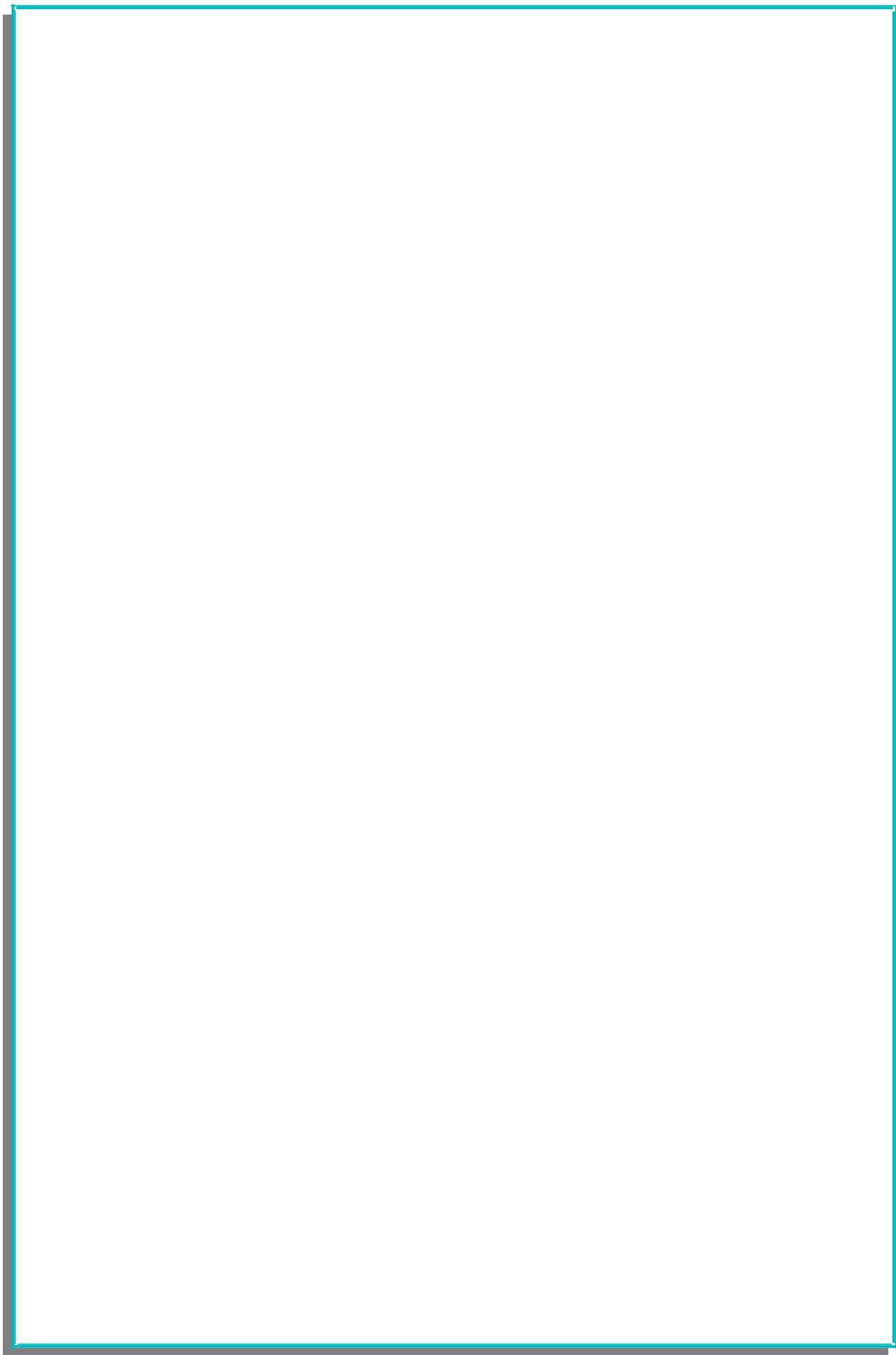
Gmail YouTube Maps

POST /compte ressource pour creer un compte

Cancel

Parameters

| Name                                       | Description                |
|--|----------------------------|
| prenom * required<br>string<br>(query)     | le prenom<br>Nasy          |
| nom * required<br>string<br>(query)        | nom<br>AHAMADI             |
| email * required<br>string<br>(query)      | email<br>nasy@gmail.com    |
| numero * required<br>string<br>(query)     | numero<br>1001             |
| password * required<br>string<br>(query)   | mot de passe<br>Passer123/ |
| typecompte * required<br>string<br>(query) | typecompte<br>Client       |



Non sécurisé | 192.168.1.252:8080

G Gmail YouTube Maps

POST /compte ressource pour créer un compte

Cancel

| Name                                       | Description                |
|--|----------------------------|
| prenom * required<br>string<br>(query)     | le prenom<br>Morez         |
| nom * required<br>string<br>(query)        | nom<br>WANDET              |
| email * required<br>string<br>(query)      | email<br>morez@gmail.com   |
| numero * required<br>string<br>(query)     | numero<br>1002             |
| password * required<br>string<br>(query)   | mot de passe<br>Passer123/ |
| typecompte * required<br>string<br>(query) | typecompte<br>Revendeur    |

selon le fournisseur chaque utilisateur a une tâche bien précise .  
 maintenant on fait un feed-back dans la base de donnée pour voir les users créés

```
MariaDB [transaction]> select * from compte;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | prenom | nom | email | numero | password | typecompte | solde |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Berenger | BENAM | berengerbenam@gmail.com | 1000 | Passer123/ | Fournisseur | 60000 |
| 2 | Nasry | AHAMADI | nasry@gmail.com | 1001 | Passer123/ | Client | 20000 |
| 3 | Morez | WANDET | morez@gmail.com | 1002 | Passer123/ | Revendeur | 40000 |
+----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

MariaDB [transaction]>
```

oui tout marche bien  
 Le Fournisseur va transférer un montant sur le compte de revendeur (1002)

← → ⌂ Non sécurisé | 192.168.1.6:8080

Gmail YouTube Maps

POST /Utilisateurs

PUT /Vente\_Credit

Parameters

| Name                          | Description       |
|-------------------------------|-------------------|
| numero_fournisseur * required | string<br>(query) |
| nom * required                | string<br>(query) |
| prenom * required             | string<br>(query) |
| email * required              | string<br>(query) |
| password * required           | string<br>(query) |
| numero_revendeur * required   | string<br>(query) |
| credit * required             | string<br>(query) |

Cancel

Execute

Responses

Code Description

200 Success

Response content type application/json

il remplit le champ

PUT /Vente\_Credit

Parameters

| Name                          | Description       |
|-------------------------------|-------------------|
| numero_fournisseur * required | string<br>(query) |
| nom * required                | string<br>(query) |
| prenom * required             | string<br>(query) |
| email * required              | string<br>(query) |
| password * required           | string<br>(query) |
| numero_revendeur * required   | string<br>(query) |
| credit * required             | string<br>(query) |

Cancel

1000

WANDET

Morez

morez@gmail.com

Passer123/

1002

10000

Execute

Responses

Code Description

200 Success

Response content type application/json

il clique sur Execute

Curl  
curl -X 'PUT' \  
'http://192.168.1.6:8080/Vente\_Credit?numero\_fournisseur=1000&nom=WANDET&prenom=Morez&email=morez%40gmail.com&password=Passer123%2F&numero\_revendeur=1002&credit=10000' \  
-H 'accept: application/json'

Request URL  
http://192.168.1.6:8080/Vente\_Credit?numero\_fournisseur=1000&nom=WANDET&prenom=Morez&email=morez%40gmail.com&password=Passer123%2F&numero\_revendeur=1002&credit=10000

Server response

| Code | Details   |
|------|---|
| 200  | Response body<br>{<br>"message": "vous avez effectué un achat donc il vous reste maintenant :50000 FCFA"<br>}<br>Response headers<br>content-length: 96<br>content-type: application/json<br>date: Sat, 23 Apr 2022 23:47:32 GMT<br>server: Werkzeug/2.0.2 Python/3.6.9 |

Responses

| Code | Description |
|------|-------------|
| 200  | Success     |

il peut voir son solde sur l'interface y compris le montant envoyé et idem sur la base de donnée on peut voir tous ses informations.

```
MariaDB [transaction]> select *from compte;
+----+-----+-----+-----+-----+-----+-----+
| id | prenom | nom   | email        | numero | password | typecompte | solde |
+----+-----+-----+-----+-----+-----+-----+
| 1  | Berenger | BENAM | berengerbenam@gmail.com | 1000  | Passer123/ | Fournisseur | 50000 |
| 2  | Nasry    | AHAMADI | nasry@gmail.com | 1001  | Passer123/ | Client      | 20000 |
| 3  | Morez    | WANDET | morez@gmail.com | 1002  | Passer123/ | Revendeur   | 50000 |
+----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

MariaDB [transaction]>
```

On constate que le solde de Fournisseur a été diminué avant il a 60000 sur son compte et son transfert vers le client il a seulement 50000

maintenant le Fournisseur peut recharger son compte il suffit de définir une fonction qui va gérer cette transaction et dans mon cas la fonction c'est L'Ajout qui permet à fournisseur de recharger son compte

PUT /RechargeCompte

Parameters

| Name   | Description                                     |
|--|---|
| numero_fournisseur <small>* required</small> | numero_fournisseur                              |
| string<br>(query)                            | <input type="text" value="humero_fournisseur"/> |
| credit <small>* required</small>             | credit  |
| string<br>(query)                            | <input type="text" value="credit"/>             |

Execute

Responses

Response content type: application/json

sur cette capture le Fournisseur recharger son compte

PUT /RechargeCompte

Parameters

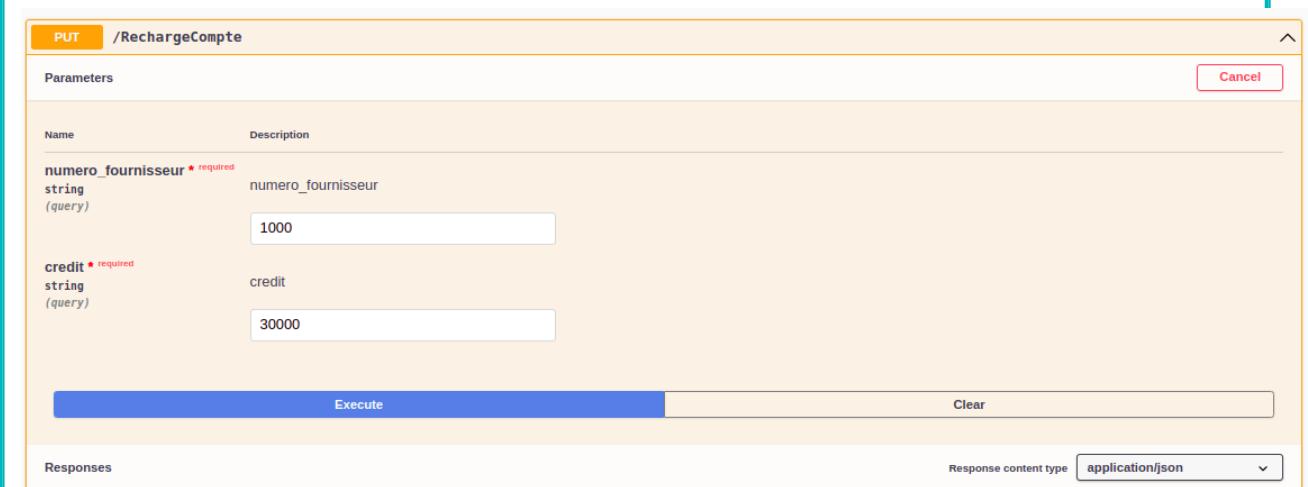
Name Description

numero\_fournisseur \* required  
string  
(query)  
1000

credit \* required  
string  
(query)  
30000

Execute Clear

Responses Response content type application/json



on voit bel et bien son solde est recharge

```
MariaDB [transaction]> select *from compte;
+-----+-----+-----+-----+-----+-----+-----+
| id | prenom | nom | email | numero | password | typecompte | solde |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | Berenger | BENAM | berengerbenam@gmail.com | 1000 | Passer123/ | Fournisseur | 80000 |
| 2 | Nasry | AHAMADI | nasry@gmail.com | 1001 | Passer123/ | Client | 20000 |
| 3 | Morez | WANDET | morez@gmail.com | 1002 | Passer123/ | Revendeur | 50000 |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

MariaDB [transaction]>
```

donc le fournisseur on le considère comme l'Administrateur il a le droit à tout.

## ● Hébergement d'Api(WSGI) avec Apache2

### Étape 1 : (hébergement de l' apitrans.py)

**Installation de paquet libapache2-mod-wsgi-py3** et il faut désinstaller la version **libapache2-mod-wsgi** souvent ça bloque Apache2 et le fichier virtuel donc c'est important de purger ce paquet pour installer la version **libapache2-mod-wsgi-py3**

### ● les paquets à installés

-il faut installer nodejs version 14

- install snap,snap install postman,npm install g jsonser

copie le code **apitrans.py** à **\_\_init\_\_.py**

```
root@berenger:/var/www/html/API# cp apitrans.py __init__.py
```

on donne le droit d'exécution à notre fichier **\_\_init\_\_.py**

```
root@berenger:/var/www/html/API# chmod +x __init__.py
```

on va créer le fichier virtuel de notre site dans **/etc/apache2/sites-available**  
on créer le fichier **apitrans.conf**

on édite le fichier **apitrans.conf**

```
GNU nano 2.9.3                               /etc/apache2/sites-available/apitrans.conf                         Modifié
<VirtualHost *:80>
    ServerName api.apitrans.sn
    WSGIScriptAlias / /var/www/html/API/apitrans.wsgi
</VirtualHost>
```

on sauvegarde puis on active le site virtuel et démarrer apache2

```
root@berenger:/etc/apache2/sites-available# a2ensite apitrans.conf
```

puis

```
root@berenger:/etc/apache2/sites-available# service apache2 restart
root@berenger:/etc/apache2/sites-available#
```

création du script wsgi, on va créer le fichier **apitrans.wsgi** dans **/var/www/html/API** et voici son contenu

```
import sys
#sys.path.insert(0 '/var/www/html/API')
sys.path.append('/var/www/html/API')
sys.stdout= sys.stderr
from api import app as application
```

vue que on a pas le nom de domaine on va dans **/etc/hosts** pour faire la correspondance qu'on a définit tout a l'heure dans notre fichier virtuel qui se trouve dans **/etc/apache2/sites-available**

|  |            |         |
|--|------------|---------|
| GNU nano 2.9.3   | /etc/hosts | Modifié |
| 127.0.0.1 localhost api.apitrans.sn                        |            |         |
| 127.0.1.1 berenger   |            |         |
| # The following lines are desirable for IPv6 capable hosts |            |         |

si tout marche bien on prend un navigateur et taper juste le nom **api.apitrans.sn** pour afficher la page d'Api

## Test :

The screenshot shows a web browser displaying the API documentation for version 1.0. The URL is `api.apitrans.sn`. The page title is "API 1.0". It lists several endpoints under the "default" namespace:

- GET /AfficheComptes
- POST /Compte (resource pour creer un compte)
- GET /ConsulterSolde/{numero}
- PUT /RechargeCompte
- DELETE /SupprimerCompte/{id}
- PUT /Transfert\_Client
- POST /Utilisateurs
- PUT /Vente\_Credit

on clique sur **AfficheComptes** :

```

200
Response body
{
  "comptes": [
    {
      "id": 1,
      "prenom": "Berenger",
      "nom": "BENAM",
      "email": "berengerbenam@gmail.com",
      "numero": "1000",
      "typecompte": "Fournisseur",
      "solde": "132000"
    },
    {
      "id": 2,
      "prenom": "Nasry",
      "nom": "AHAMADI",
      "email": "nasry@gmail.com",
      "numero": "1001",
      "typecompte": "Client",
      "solde": "20000"
    },
    {
      "id": 3,
      "prenom": "Morez",
      "nom": "WANDETT",
      "email": "morez@gmail.com",
      "numero": "1002",
      "typecompte": "Revendeur",
      "solde": "50000"
    }
  ]
}

Response headers
connection: Keep-Alive
content-length: 439
content-type: application/json
date: Tue, 03 May 2022 16:32:06 GMT
keep-alive: timeout=5,max=100
server: Apache/2.4.29 (Ubuntu)

```

on voit les trois comptes qui ont été créés avant l'hébergement.

résultat on arrive à héberger l'api nickel .

## ➤ Interaction des formulaires via l'api

le but est de créer un formulaire et faire des actions comme ajout,insertion etc... via l'api

on crée le fichier **formulaireAjoutCompte.html**

```
root@berenger:/var/www/html/API# touch formulaireAjoutCompte.html
```

voici le code :

```

root@berenger:/var/www/html/API                               formulaireAjoutCompte.html
GNU nano 2.9.3

html>
  <head>
    <title>Formulaire ajout compte</title>
  </head>
<body>
</body>
<font color="#B22222" size="5"><h1>Bienvenue dans mon formulaire html</h1></font>

<form method="POST" action="formulaire.php">
  prenom: <input type="text" name="prenom" required="required"/><br>
  nom: <input type="text" name="nom" required="required"/><br>
  email: <input type="text" name="email" required="required"/><br>
  numero: <input type="text" name="numero" required="required"/><br>
  password: <input type="password" name="password" required="required"/><br>
  typecompte: <input type="text" name="typecompte" required="required"/><br>
  solde: <input type="text" name="solde" required="required"/><br>
  <input type="submit" value="valider"/>
</form>

</html>

```

voici le code html et on respect les champs de notre api

on crée le deuxième fichier **php** pour interagir à notre fichier formulaire : **formulaire.php**

```
root@berenger:/var/www/html/API# touch formulaire.php
```

code

```
GNU nano 2.9.3                               formulaire.php

?php
$prenom = $_POST["prenom"];
$nom = $_POST["nom"];
$email = $_POST["email"];
$numéro = $_POST["numero"];
$password = $_POST["password"];
$typecompte = $_POST["typecompte"];
$solde = $_POST["solde"];
$url = "url_to_post";
$data = array("prenom"=>$prenom, "nom"=>$nom, "email"=>$email, "numero"=>$numéro, "password"=>$password, "typecompte"=>$typecompte, "solde"=>$solde);
$donnee = json_encode($data);
$curl = curl_init($url);

curl_setopt($curl,CURLOPT_POST,1);
curl_setopt($curl,CURLOPT_POSTFIELDS,$donnee);
curl_setopt($curl,CURLOPT_RETURNTRANSFER,1);
curl_setopt($curl,CURLOPT_HTTPHEADER,array("Content-Type:application/json"));
curl_setopt($curl,CURLOPT_URL,"http://api.apitrans.sn/Compte");
//curl_setopt($curl,CURLOPT_CUSTOMREQUEST,"GET");
$result = curl_exec($curl);
curl_close($curl);

echo $donnee;
?>
```

**TEST :**

on tape cette url : **localhost/API/formulaireAjoutCompte.html**



## Bienvenue dans mon formulaire html

prenom:

nom:

email:

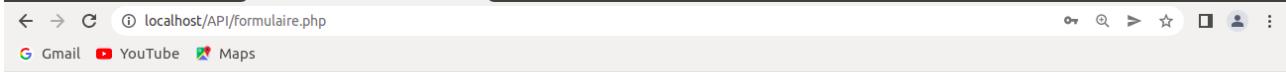
numero:

password:

typecompte:

solde:

puis cliquer sur **valider**



```
{"prenom":"Latyr","nom":"Ndiaye","email":"latyr.ndiaye@gmail.com","numero":"1003","password":"Passer123V","typecompte":"client","solde":"40000"}
```

on voit l'insertion marche bien maintenant on va au niveau de L'API pour voir s'il a crée l'utilisateur **Latyr**

Non sécurisé | api.apitrans.sn

Gmail YouTube Maps

## API 1.0

[ Base URL: / ]  
[/swagger.json](#)

**default** Default namespace

- GET** /AfficheComptes
- POST** /Compte ressource pour creer un compte
- GET** /ConsulterSolde/{numero}
- PUT** /RechargeCompte
- DELETE** /SupprimerCompte/{id}
- PUT** /Transfert\_Client
- POST** /Utilisateurs

on clique sur AfficheComptes

Non sécurisé | api.apitrans.sn

Gmail YouTube Maps

```
-H accept: application/json
```

Request URL  
<http://api.apitrans.sn/AfficheComptes>

Server response

| Code | Details   |
|------|---|
| 200  | Response body   |
|      | <pre> {   "id": 2,   "prenom": "Nasry",   "nom": "AHAMADI",   "email": "nasry@gmail.com",   "numero": "1001",   "typecompte": "Client",   "solde": "20000" }, {   "id": 3,   "prenom": "Morez",   "nom": "WANDET",   "email": "morez@gmail.com",   "numero": "1002",   "typecompte": "Revendeur",   "solde": "50000" }, {   "id": 5,   "prenom": "Latyr",   "nom": "Ndiaye",   "email": "latyr.ndiaye@gmail.com",   "numero": "1003",   "typecompte": "client",   "solde": "40000" } ] </pre> |

[Copy](#) [Download](#)

on voit le compte Latyr idem dans la base de donnée

```
Database changed
MariaDB [transaction]> select *from compte;
+----+-----+-----+-----+-----+-----+-----+
| id | prenom | nom | email | numero | password | typecompte | solde |
+----+-----+-----+-----+-----+-----+-----+
| 1 | Berenger | BENAM | berengerbenam@gmail.com | 1000 | Passer123/ | Fournisseur | 132000 |
| 2 | Nasry | AHAMADI | nasry@gmail.com | 1001 | Passer123/ | Client | 20000 |
| 3 | Morez | WANDET | morez@gmail.com | 1002 | Passer123/ | Revendeur | 50000 |
| 5 | Latyr | Ndiaye | latyr.ndiaye@gmail.com | 1003 | Passer123/ | client | 40000 |
+----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

MariaDB [transaction]>
```

super

on crée le fichier : **formulaireModifier.html** pour faire l'ajout sur le compte de **Fournisseur**

```
GNU nano 2.9.3                               formulaireModifier.html

<html>
    <head>
<title>Formulaire mise à jour RechargeCompte</title>
</head>
<body>

<font color="#B22222" size="5"><h1 class="modal-title">Bienvenue sur la page Recharge Compte</h1></font>
    <form method="POST" action="formulaireModifier.php">
        numero_fournisseur: <input type="text" name="numero_fournisseur" required="required"/><br>
        credit: <input type="text" name="credit" required="required"/><br>
        <input type="submit" value="valider"/>
    </form>
</body>
</html>
```

idem pour le fichier : **formulaireModifier.php**

```
GNU nano 2.9.3                               formulaireModifier.php

<?php
$numero_fournisseur = $_POST["numero_fournisseur"];
$credit = $_POST["credit"];
$url = 'url_to_post';
$data = array("numero_fournisseur"=>$numero_fournisseur, "credit"=>$credit);
$donnee = json_encode($data);

$curl = curl_init($url);

curl_setopt($curl,CURLOPT_POSTFIELDS,$donnee);
curl_setopt($curl,CURLOPT_RETURNTRANSFER,1);
curl_setopt($curl,CURLOPT_HTTPHEADER,array("Content-Type:application/json"));
curl_setopt($curl,CURLOPT_CUSTOMREQUEST,'PUT');
curl_setopt($curl,CURLOPT_URL,"http://api.apitrans.sn/RechargeCompte");
//curl_setopt($curl,CURLOPT_URL, "http://localhost:5000/comptes/
$result = curl_exec($curl);
curl_close($curl);

echo $donnee;
?>
```

voici le compte de Fournisseur

A screenshot of a browser window displaying a JSON response. The URL in the address bar is `http://api.apitrans.sn/AfficheComptes`. The response code is 200. The response body contains an array of three objects representing accounts:

```
{ "comptes": [ { "id": 1, "prenom": "Berenger", "nom": "BENAM", "email": "berengerbenam@gmail.com", "numero": "1000", "typecompte": "Fournisseur", "solde": "50000" }, { "id": 2, "prenom": "Nasry", "nom": "AHAMAD", "email": "nasry@gmail.com", "numero": "1001", "typecompte": "client", "solde": "20000" }, { "id": 3, "prenom": "Morez", "nom": "WANDET", "email": "morez@gmail.com", "numero": "1002", "typecompte": "Revendeur", "solde": "50000" } ] }
```

There are buttons for 'Download' and 'Copy' on the right side of the JSON block.

on voit sur cette capture le Fournisseur a 50000 sur son compte maintenant on lance le **formulaireModifier.html**

**TEST :**

A screenshot of a browser window showing the title "Bienvenue sur la page Recharge Compte". Below the title are three input fields: "numero\_fournisseur:", "credit:", and a "valider" button.

on remplit les champs

A screenshot of a browser window showing the title "Bienvenue sur la page Recharge Compte". The "numero\_fournisseur" field contains "1000", the "credit" field contains "10000", and the "valider" button is visible below the fields.

cliquer sur valider

A screenshot of a browser window showing the title "Bienvenue sur la page Recharge Compte". The input field contains the JSON object: {"numero\_fournisseur": "1000", "credit": "10000"}.

tout marche bien on va lancer l'api pour voir le changement sur le compte **Fournisseur**

The screenshot shows a browser window with the URL <http://api.apitrans.sn/AfficheComptes>. The page title is "Non sécurisé | api.apitrans.sn". Below the address bar, there are links for Gmail, YouTube, and Maps. The main content area is titled "Server response" and shows a "Code" section with "200" and a "Details" section. Under "Response body", a JSON object is displayed:

```
{ "comptes": [ { "id": 1, "prenom": "Berenger", "nom": "BENAM", "email": "berengerbenam@gmail.com", "numero": "1000", "typecompte": "Fournisseur", "solde": "60000" }, { "id": 2, "prenom": "Nasry", "nom": "AHAMADI", "email": "nasry@gmail.com", "numero": "1001", "typecompte": "client", "solde": "20000" }, { "id": 3, "prenom": "Morez", "nom": "WANDEL", "email": "morez@gmail.com", "numero": "1002", "typecompte": "Revendeur", "solde": "50000" } ] }
```

At the bottom right of the JSON block, there are "Copy" and "Download" buttons.

on voit avant il a 50000 maintenant c'est 60000 donc le **formulaireModifier** marche sans problème.

- On créer un formulaire qui permet de supprimer un compte user  
voici le code : **formulaireDELETE.html**

```
GNU nano 2.9.3                               formulaireDELETE.html                         Modifié

<html>
    <head>
<title>Formulaire suppression compte</title>
    </head>
<body>
</body>
<font color="green" size="5"><h1>Formulaire DELETE qui permet de supprimer un compte a travers son ID </h1></font>
    <form method="POST" action="formulaireDELETE.php">
        id: <input type="text" name="id"/>br>
        <input type="submit" value="valider"/>
    </form>
</html>
```

et le code : **formulaireDELETE.php**

```

GNU nano 2.9.3                               formulaireModifier.php

<?php
$numero_fournisseur = $_POST["numero_fournisseur"];
$credit = $_POST["credit"];
$url = 'url_to_post';
$data = array("numero_fournisseur"=>$numero_fournisseur,"credit"=>$credit);
$donnee = json_encode($data);

$curl = curl_init($url);

curl_setopt($curl,CURLOPT_POSTFIELDS,$donnee);
curl_setopt($curl,CURLOPT_RETURNTRANSFER,1);
curl_setopt($curl,CURLOPT_HTTPHEADER,array("Content-Type:application/json"));
curl_setopt($curl,CURLOPT_CUSTOMREQUEST,'PUT');
curl_setopt($curl,CURLOPT_URL,"http://api.apitrans.sn/RechargeCompte");
//curl_setopt($curl,CURLOPT_URL,"http://localhost:5000/comptes/"
$result = curl_exec($curl);
curl_close($curl);

echo $donnee;

?>

```

maintenant on va essayer de supprimer le compte **Latyr** donc on va faire un feedback dans la base de donnée pour connaître son id.

```

Database changed
MariaDB [transaction]> select *from compte;
+----+-----+-----+-----+-----+-----+-----+
| id | prenom | nom | email | numero | password | typecompte | solde |
+----+-----+-----+-----+-----+-----+-----+
| 1 | Berenger | BENAM | berengerbenam@gmail.com | 1000 | Passer123/ | Fournisseur | 132000 |
| 2 | Nasry | AHAMADI | nasry@gmail.com | 1001 | Passer123/ | Client | 20000 |
| 3 | Morez | WANDET | morez@gmail.com | 1002 | Passer123/ | Revendeur | 50000 |
| 5 | Latyr | Ndiaye | latyr.ndiaye@gmail.com | 1003 | Passer123/ | client | 40000 |
+----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

on sait que le compte **Latyr** a comme ID 5

**TEST :** on tape **localhost/API/formulaireDELETE.html**

localhost/API/formulaireDELETE.html

Gmail YouTube Maps

## Formulaire DELETE

id:

on clique sur **valider**

localhost/API/formulaireDELETE.php

Gmail YouTube Maps

```
{"message": "Compte 5"}
```

nickel on lance L'API pour vérifier le compte **Latyr** qui a comme id 5

#### Server response

Code Details

200

Response body

```
"comptes": [
    {
        "id": 1,
        "prenom": "Berenger",
        "nom": "BENAM",
        "email": "berengerbenam@gmail.com",
        "numero": "1000",
        "typecompte": "Fournisseur",
        "solde": "60000"
    },
    {
        "id": 2,
        "prenom": "Nasry",
        "nom": "AHAMADI",
        "email": "nasry@gmail.com",
        "numero": "1001",
        "typecompte": "Client",
        "solde": "20000"
    },
    {
        "id": 3,
        "prenom": "Morez",
        "nom": "WANDET",
        "email": "morez@gmail.com",
        "numero": "1002",
        "typecompte": "Revendeur",
        "solde": "50000"
    }
]
```



Download

le compte **Latyr** a été supprimé idem dans la base de donnée

**Database changed**

```
MariaDB [transaction]> select *from compte;
```

| id | prenom   | nom     | email                   | numero | password   | typecompte  | solde |
|----|----------|---------|-------------------------|--------|------------|-------------|-------|
| 1  | Berenger | BENAM   | berengerbenam@gmail.com | 1000   | Passer123/ | Fournisseur | 60000 |
| 2  | Nasry    | AHAMADI | nasry@gmail.com         | 1001   | Passer123/ | Client      | 20000 |
| 3  | Morez    | WANDET  | morez@gmail.com         | 1002   | Passer123/ | Revendeur   | 50000 |

```
3 rows in set (0.01 sec)
```

```
MariaDB [transaction]> █
```

## Conclusion :

Les développeurs travaillent avec des API pour créer des logiciels et des applications. Il est rare que vous, l'utilisateur final, interagissiez directement avec une API. Les API fonctionnent comme une porte d'entrée, permettant aux entreprises de partager des informations sélectionnées, mais aussi de garder les demandes non désirées.

