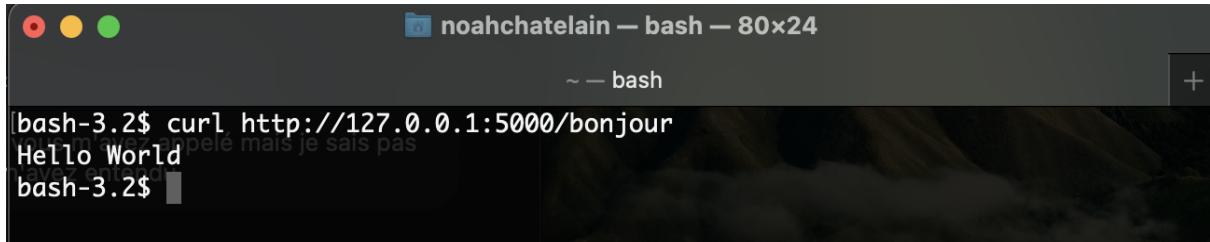


TP1 WebServers - Noah Chatelain & Ambroise Gyre

⚠ Pour vos tests, le mot de passe par défaut de tous les étudiants est : **123456**
Il apparaît dans la base de donné hashé en SHA256

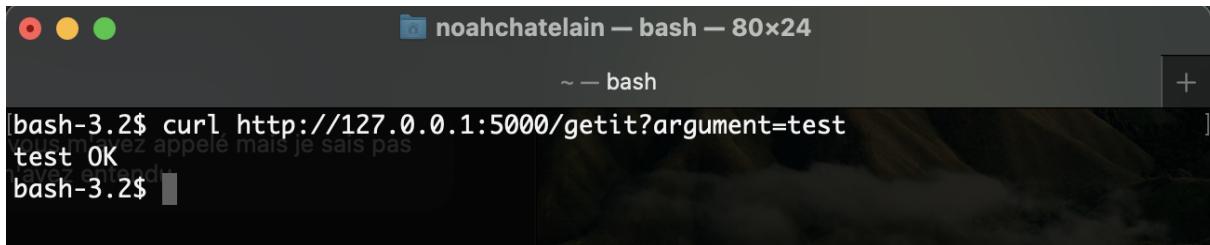
```
MBP-de-Noah:TP Webserver noahchatelain$ python3 app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

http://127.0.0.1/5000/bonjour



```
noahchatelain — bash — 80x24
~ — bash
[bash-3.2$ curl http://127.0.0.1:5000/bonjour
Hello World
bash-3.2$ ]
```

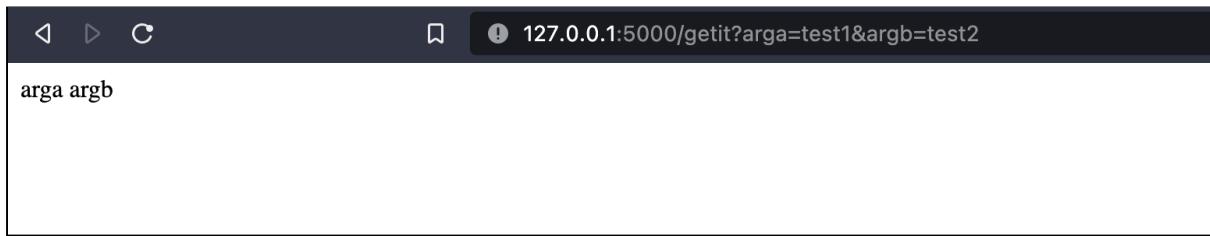
http://127.0.0.1/5000/getit?argument=test



```
noahchatelain — bash — 80x24
~ — bash
[bash-3.2$ curl http://127.0.0.1:5000/getit?argument=test
test OK
bash-3.2$ ]
```

curl 127.0.0.1:5000/getit?arga=test1&argb=test2

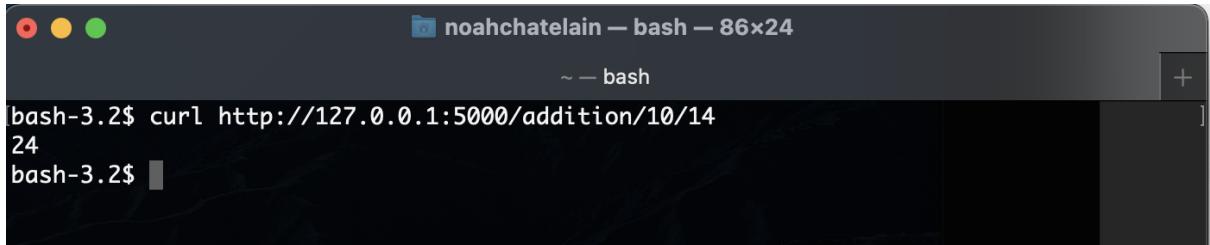
(sans modification permettant l'addition de deux paramètres)



```
127.0.0.1:5000/getit?arga=test1&argb=test2
arga argb
```

curl 127.0.0.1:5000/addition/10/14

(avec modification permettant l'addition de deux paramètres)



```
noahchatelain — bash — 86x24
~ — bash
[bash-3.2$ curl http://127.0.0.1:5000/addition/10/14
24
bash-3.2$ ]
```

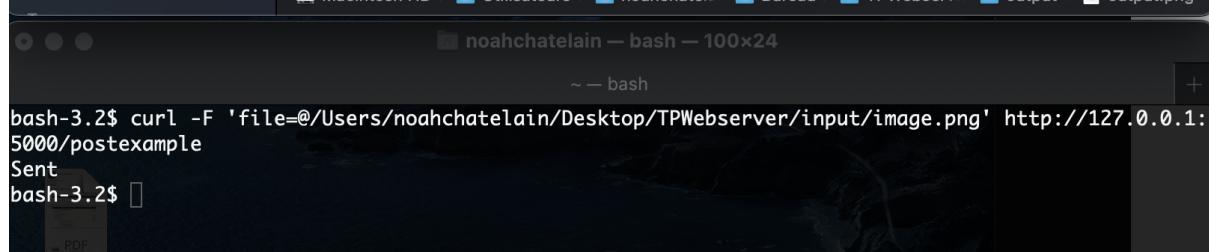
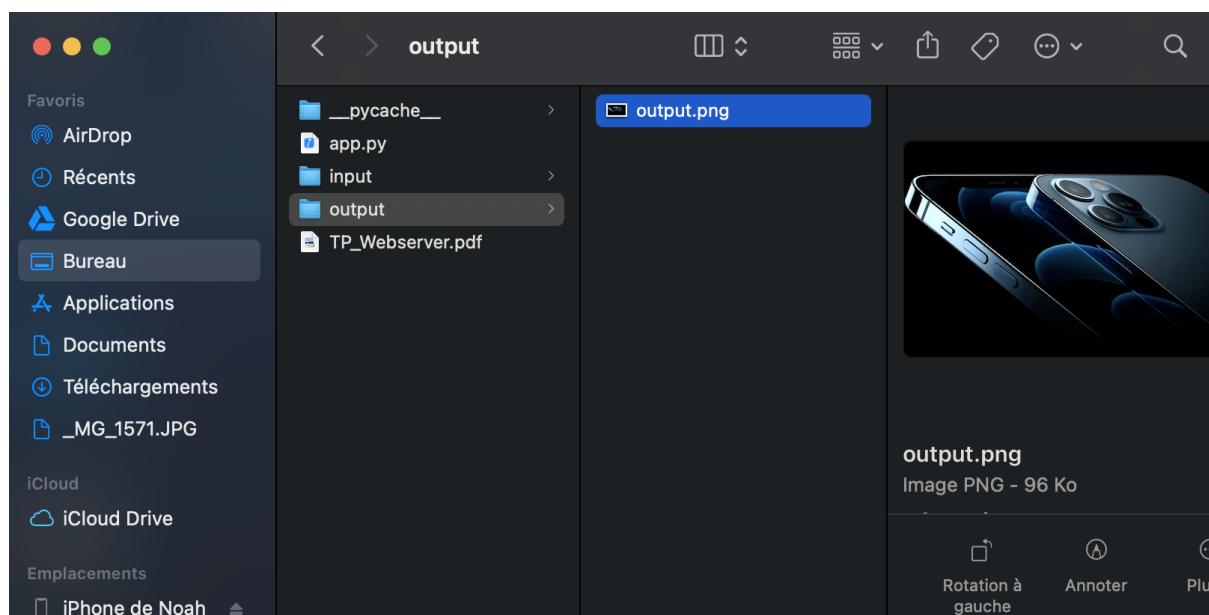
http://127.0.0.1:5000/compare/test1/test2

(ajout de la route permettant la comparaison de deux chaînes de caractères)



```
[bash-3.2$ curl http://127.0.0.1:5000/compare/test1/test2
False
[bash-3.2$ curl http://127.0.0.1:5000/compare/test1/test1
True
bash-3.2$ ]
```

Réception d'un fichier et requête POST :



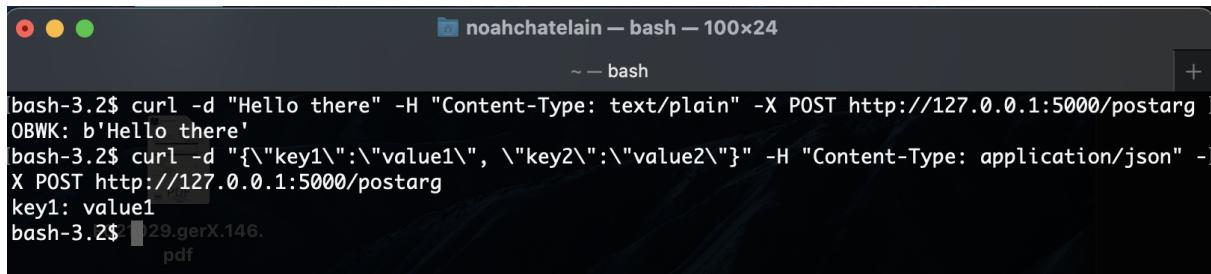
```
bash-3.2$ curl -F 'file=@/Users/noahchatelain/Desktop/TPWebserver/input/image.png' http://127.0.0.1:5000/postexample
Sent
bash-3.2$ ]
```

```
@app.route('/postexample', methods=['POST'])
def postexample():
    if request.method == 'POST':
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
    return "Sent\n"
```

Traitement différencié selon le type de contenu (texte pur ou json).

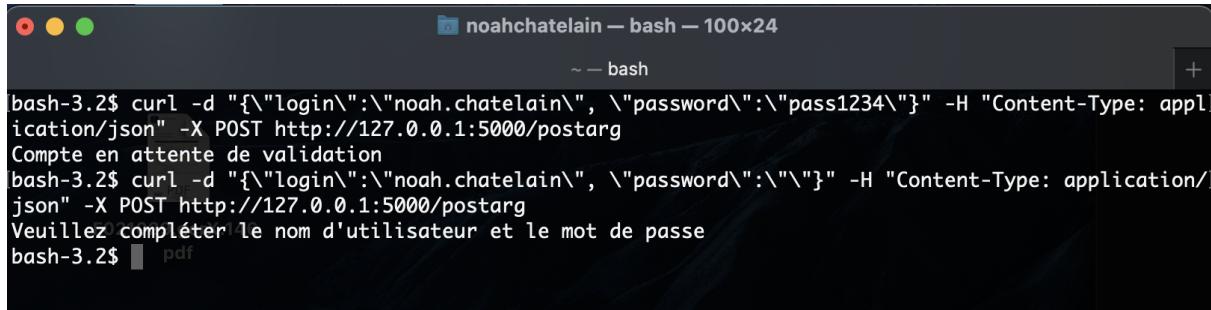
Explication demandée :

Les 2 requêtes *curl* du TP permettent de passer dans le Header de la requête POST le Content-Type de ce que l'on va envoyer. Ainsi, dans notre API, nous pourrons faire des conditions sur ce Content-Type et effectuer des actions différentes.



```
noahchatelain — bash — 100x24
~ — bash
bash-3.2$ curl -d "Hello there" -H "Content-Type: text/plain" -X POST http://127.0.0.1:5000/postarg
OBWk: b'Hello there'
bash-3.2$ curl -d "{\"key1\":\"value1\", \"key2\":\"value2\"}" -H "Content-Type: application/json" -X POST http://127.0.0.1:5000/postarg
key1: value1
bash-3.2$ pdf
```

Accepter des informations de login et mot de passe dans un json, pour l'instant toujours répondre « Compte en attente de validation » si un json avec des informations de login ET de mot de passe non vides sont présentes.



```
noahchatelain — bash — 100x24
~ — bash
bash-3.2$ curl -d "{\"login\":\"noah.chatelain\", \"password\":\"pass1234\"}" -H "Content-Type: application/json" -X POST http://127.0.0.1:5000/postarg
Compte en attente de validation
bash-3.2$ curl -d "{\"login\":\"noah.chatelain\", \"password\":\"\"}" -H "Content-Type: application/json" -X POST http://127.0.0.1:5000/postarg
Veuillez compléter le nom d'utilisateur et le mot de passe
bash-3.2$ pdf
```

```
@app.route('/postarg', methods=['POST'])
def postarg():
    if request.headers['Content-Type'] == 'text/plain':
        return "OBWk: " + str(request.data) + "\n"
    elif request.headers['Content-Type'] == 'application/json':
        if request.json["login"] != "" and request.json["password"] != "":
            return "Compte en attente de validation\n"
        else:
            return "Veuillez compléter le nom d'utilisateur et le mot de passe\n"
    return "NOK"
```

Création d'une nouvelle route qui permettra la création d'une base de données :

```
@app.route('/createmanu')
def createmanu():
    conn = sqlite3.connect('database.db')
    print ("Opened database successfully")
    #type : https://www.sqlite.org/datatype3.html
    conn.execute('CREATE TABLE students (name TEXT, addr TEXT, city TEXT, pin TEXT)')
    print("Table created successfully")
    conn.close()
    return 'Your table is created.'
```

```
MBP-de-Noah:TPWebserver noahchatelain$ python3 app.py
 * Serving Flask app "app" (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: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Opened database successfully
Table created successfully
127.0.0.1 - - [23/Sep/2021 09:16:02] "GET /createmanu HTTP/1.1" 200 -
```



Your table is created.

A screenshot of the DB Browser for SQLite application. The title bar says "DB Browser for SQLite - /". Below the title bar are two buttons: "Nouvelle Base de Données" and "Ouvrir une Base de Données". A navigation bar at the top has tabs: "Structure de la Base de ...", "Parcourir les ...", "Éditer les ...", and "Exécut...". The main area shows a table structure. The columns are "Nom" and "Type". Under "Tables (1)", there is a single entry "students" which contains four columns: "name" (TEXT), "addr" (TEXT), "city" (TEXT), and "pin" (TEXT). There are also sections for "Index (0)", "Vues (0)", and "Déclencheurs (0)".

| Nom | Type | | | | | | | | | | |
|------------------|--|--|--|------|------|------|------|------|------|-----|------|
| Tables (1) | | | | | | | | | | | |
| students | <table border="1"><thead><tr><th></th><th></th></tr></thead><tbody><tr><td>name</td><td>TEXT</td></tr><tr><td>addr</td><td>TEXT</td></tr><tr><td>city</td><td>TEXT</td></tr><tr><td>pin</td><td>TEXT</td></tr></tbody></table> | | | name | TEXT | addr | TEXT | city | TEXT | pin | TEXT |
| | | | | | | | | | | | |
| name | TEXT | | | | | | | | | | |
| addr | TEXT | | | | | | | | | | |
| city | TEXT | | | | | | | | | | |
| pin | TEXT | | | | | | | | | | |
| Index (0) | | | | | | | | | | | |
| Vues (0) | | | | | | | | | | | |
| Déclencheurs (0) | | | | | | | | | | | |

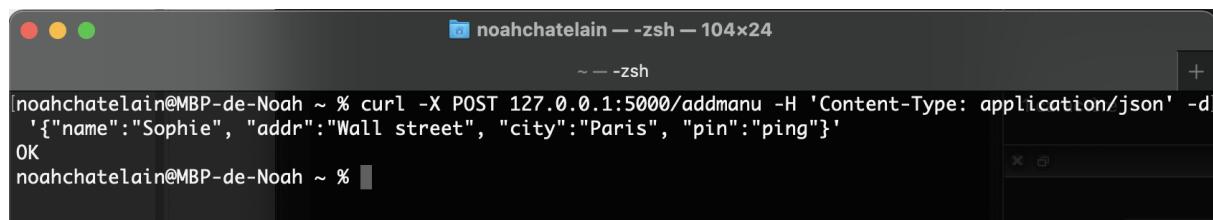
Résultat visuel de la création de la table dans DB Browser for SQLite

/addmanu

```
@app.route('/addmanu')
def addmanu():
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    oneRecord = ['Sophie', 'Boulevard Concorde', 'Paris', 'Xae-A12']
    cursor.execute('INSERT INTO students VALUES (?,?,?,?)', oneRecord)
    conn.commit()
    conn.close()
    return 'OK'
```

Modifier la méthode en requête POST :

```
@app.route('/addmanu', methods=['POST'])
def addmanu():
    if request.headers['content-type'] == 'application/json':
        name = request.json["name"]
        addr = request.json["addr"]
        city = request.json["city"]
        pin = request.json["pin"]
        conn = sqlite3.connect('database.db')
        cursor = conn.cursor()
        oneRecord = [name, addr, city, pin]
        cursor.execute('INSERT INTO students VALUES (?,?,?,?)', oneRecord)
        conn.commit()
        conn.close()
        return 'OK\n'
    return "NOK"
```



A screenshot of a terminal window titled "noahchatelain -- zsh -- 104x24". The window shows a command being run in the background and its output in the foreground. The command is:

```
[noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/addmanu -H 'Content-Type: application/json' -d '{"name":"Sophie", "addr":"Wall street", "city":"Paris", "pin":"ping"}'
```

The output shows the command being run and the response "OK" returned by the server.

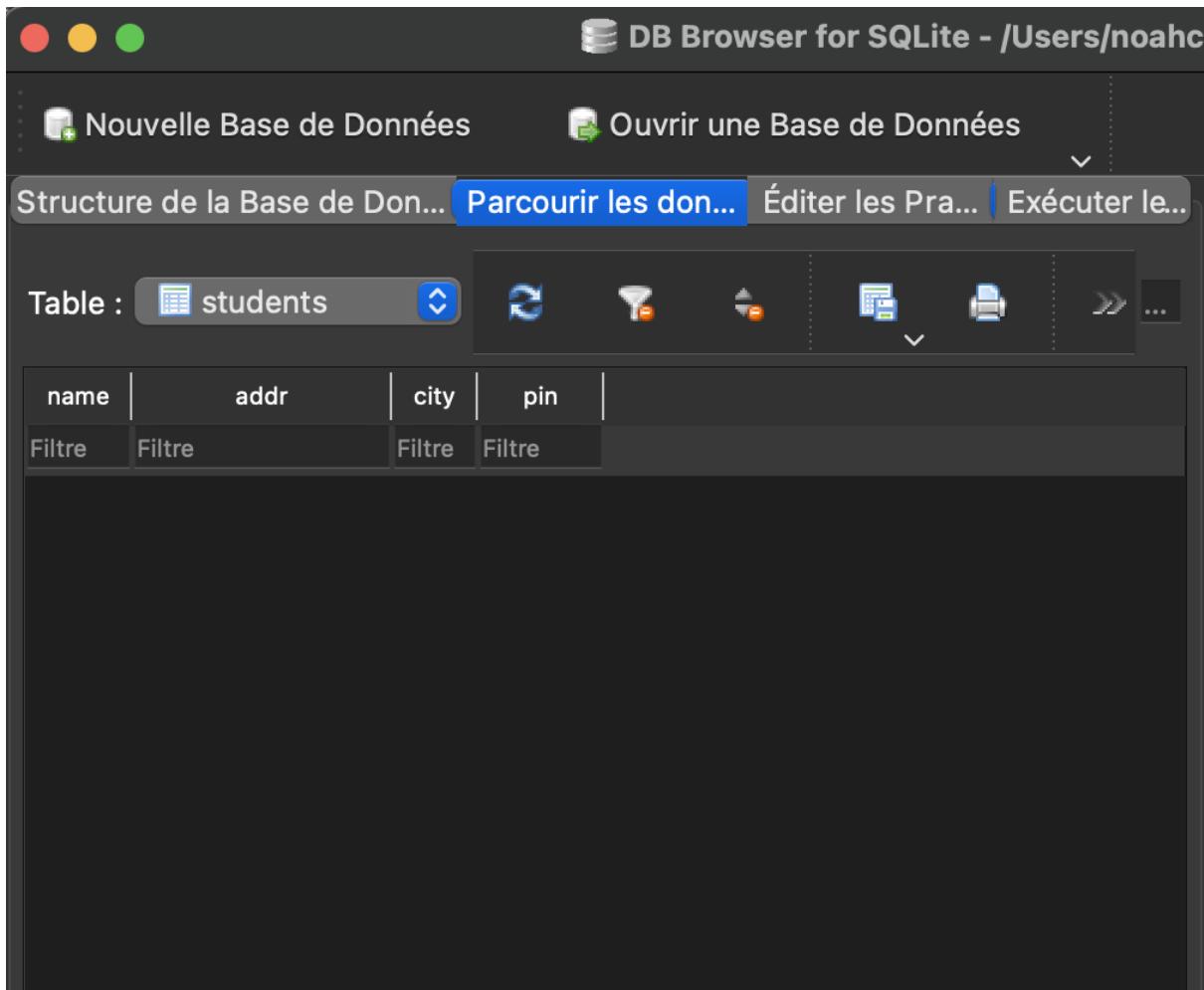
The screenshot shows a dark-themed interface of DB Browser for SQLite. At the top, there are buttons for 'Nouvelle Base de Données' and 'Ouvrir une Base de Données'. Below that is a menu bar with 'Structure de la Base de Données', 'Parcourir les don...', 'Éditer les Pra...', and 'Exécuter le...'. The main area shows a table named 'students' with four columns: 'name', 'addr', 'city', and 'pin'. A single row is displayed with the values: name = Sophie, addr = Wall street, city = Paris, pin = ping. There are filter buttons for each column.

/deletemanu

```
@app.route('/deletemanu', methods=['POST'])
def deletemanu():
    if request.headers['content-type'] == 'application/json':
        if request.json["name"] != "":
            name = request.json["name"]
            if checkIfNameExistInDb(name):
                conn = sqlite3.connect('database.db')
                cursor = conn.cursor()
                oneRecord = [name]
                cursor.execute('DELETE FROM students WHERE name = ?', oneRecord)
                conn.commit()
                conn.close()
                return 'OK\n'
            else:
                return str("There are no results for the name : " + name + "\n")
        return "NOK"
```

The terminal window title is 'noahchatelain -- -zsh -- 104x24'. It shows the user running two curl commands. The first command attempts to delete a student named 'Sophiee', but since it doesn't exist, it returns 'There are no results for the name : Sophiee'. The second command attempts to delete a student named 'TestIMTezrgt', which also fails because it doesn't exist. Both commands return 'OK' at the end.

```
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/deletemanu -H 'Content-Type: application/json' -d '{"name":"Sophiee"}'
There are no results for the name : Sophiee
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/deletemanu -H 'Content-Type: application/json' -d '{"name":"TestIMTezrgt"}'
OK
noahchatelain@MBP-de-Noah ~ %
```



Récupération des informations d'un étudiant : `/getmanu/<student_name>`

```
noahchatelain@MBP-de-Noah ~ % curl localhost:5000/getmanu/Test
There are no results for the name : Test
noahchatelain@MBP-de-Noah ~ % curl localhost:5000/getmanu/Sophie
Sophie | Wall street | Paris | ping
Sophie | Anywhere | Lille | pong
noahchatelain@MBP-de-Noah ~ %
```

Modification de la première route : Vérifier s'il n'y a pas de doublons à l'ajout

```
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/addmanu -H 'Content-Type: application/json' -d '{"name":"Noah", "addr":"Wall street", "city":"Paris", "pin":"ping"}'
OK
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/addmanu -H 'Content-Type: application/json' -d '{"name":"Noah", "addr":"Wall street", "city":"Paris", "pin":"ping"}'
The name Noah is already taken
noahchatelain@MBP-de-Noah ~ %
```

Création de la fonction `nameInTheDataBase()` pour vérifier si `name` existe dans la BDD

```
def nameInTheDataBase(name):
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    oneRecord = [name]
    cursor.execute("SELECT * FROM students WHERE name = ?", oneRecord)
    rows = cursor.fetchall()
    return len(rows) >= 1
```

Ajout de la mise à jour d'un étudiant avec la méthode HTTP PUT :

```
@app.route('/updatemanu', methods=['PUT'])
def updatemanu():
    if request.headers['content-type'] == 'application/json':
        name = request.json["name"]
        addr = request.json["addr"]
        city = request.json["city"]
        pin = request.json["pin"]
        if nameInTheDataBase(name):
            conn = sqlite3.connect('database.db')
            cursor = conn.cursor()
            oneRecord = [addr, city, pin, name]
            cursor.execute('UPDATE students SET addr = ?, city = ?, pin = ? WHERE name = ?', oneRecord)
            conn.commit()
            conn.close()
            return 'OK\n'
        return str("The name " + name + " is already taken\n")
```



The screenshot shows a terminal window titled "noahchatelain -- zsh -- 104x24". The command entered is:

```
curl -X PUT 127.0.0.1:5000/updatemanu -H 'Content-Type: application/json' -d '{"name": "Noah", "addr": "Wall streetz", "city": "Pariz", "pin": "pingz"}'
```

The response is "OK". Below the terminal, a table is displayed:

| | 10 | 11 | Noah | Wall streetz | Pariz | pingz |
|-----|----|----|------|--------------|-------|-------|
| pdf | | | | | | |

Ajout de la table *notes* qui relie l'ID de la table à la valeur *pin* de la table *students*.

Grâce à ça, on peut désormais retrouver toutes les notes d'un étudiant avec une requête POST avec en body le JSON du *name* et du *password* :

```
curl -X POST 127.0.0.1:5000/getNote -H 'Content-Type: application/json' -d '{"name": "Sophie", "password": "123456"}'
```

Avec les messages d'erreurs demandés, tels que "*Incorrect password*" ou encore "*There are no results for the name xxxx*"

```

@app.route('/getNote', methods=['POST'])
def getNote():
    if request.headers['content-type'] == 'application/json':
        if request.json["name"] != "" and request.json["password"] != "":
            name = request.json["name"]
            password = request.json["password"]
            if nameInTheDataBase(name):
                if nameAndPasswordExist(name, password):
                    conn = sqlite3.connect('database.db')
                    cursor = conn.cursor()
                    oneRecord = [name, password]
                    cursor.execute('SELECT name, note FROM students INNER JOIN notes ON students.pin = notes.id WHERE students.name = ? and students.password = ?', oneRecord)
                    rows = cursor.fetchall()
                    result = ""
                    for row in rows:
                        result += str(row[0]) + " | " + str(row[1]) + "\n"
                        print(row)
                    return str(result)
                else:
                    return str("Incorrect password\n")
            else:
                return str("There are no results for the name : " + name + "\n")
        else:
            return str("There are no results for the name : " + name + "\n")
    return "NOK"

```

```

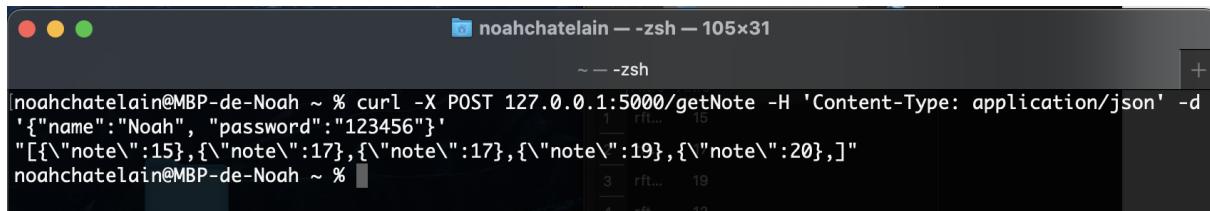
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/getNote -H 'Content-Type: application/json' -d '{"name":"Jean-Paul", "password":"123456"}'
There are no results for the name : Jean-Paul
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/getNote -H 'Content-Type: application/json' -d '{"name":"Sophie", "password":"123457"}'
Incorrect password
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/getNote -H 'Content-Type: application/json' -d '{"name":"Sophie", "password":"123456"}'
Sophie | 12
Sophie | 14
Sophie | 15
Sophie | 17
Sophie | 19
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/getNote -H 'Content-Type: application/json' -d '{"name":"Louise", "password":"123456"}'
Louise | 13
Louise | 14
Louise | 16
Louise | 18
noahchatelain@MBP-de-Noah ~ % curl -X POST 127.0.0.1:5000/getNote -H 'Content-Type: application/json' -d '{"name":"Noah", "password":"123456"}'
Noah | 15
Noah | 17
Noah | 17
Noah | 19
Noah | 20
noahchatelain@MBP-de-Noah ~ %

```

Résultat des erreurs de *name* inexistant et de mot de passe incorrects, et des notes

En revoyant le JSON avec uniquement les notes de l'étudiant :

```
@app.route('/getNote', methods=['POST'])
def getNote():
    if request.headers['content-type'] == 'application/json':
        if request.json["name"] != "" and request.json["password"] != "":
            name = request.json["name"]
            password = request.json["password"]
            if nameInTheDataBase(name):
                if nameAndPasswordExist(name, password):
                    conn = sqlite3.connect('database.db')
                    cursor = conn.cursor()
                    oneRecord = [name, password]
                    cursor.execute('SELECT name, note FROM students INNER JOIN notes ON students.pin = notes.id WHERE students.name = ? and students.password = ?', oneRecord)
                    rows = cursor.fetchall()
                    result = ""
                    for row in rows:
                        result += '{"note":' + str(row[1]) + '},'
                    print(result)
                    result = "[" + result + "]"
                    return jsonify(result)
                else:
                    return str("Incorrect password\n")
            else:
                return str("There are no results for the name : " + name + "\n")
        return "NOK"
```



A terminal window titled "noahchatelain -- -zsh -- 105x31" showing the output of a curl POST request to 127.0.0.1:5000/getNote. The command sent was:

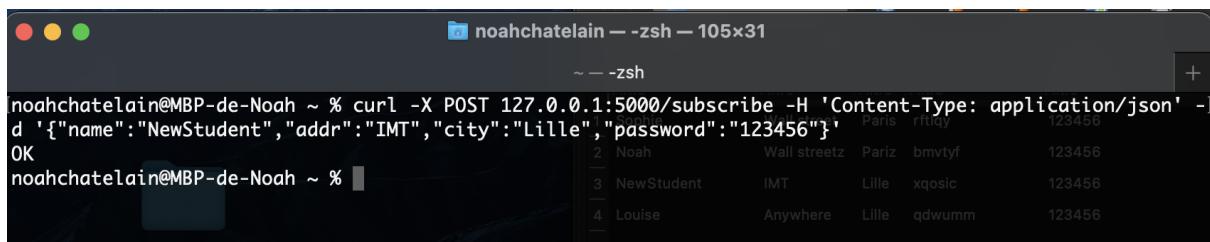
```
curl -X POST 127.0.0.1:5000/getNote -H 'Content-Type: application/json' -d '{"name":"Noah", "password":"123456"}'
```

The response received was:

```
[{"note":15}, {"note":17}, {"note":17}, {"note":19}, {"note":20}]
```

L'utilisateur dispose d'un lien d'inscription /subscribe qui lui permet de compléter ses informations avec un mot de passe, et son identifiant unique qui le relie à la table *note* est généré aléatoirement via une chaîne de caractères de taille 6.

```
@app.route('/subscribe', methods=['POST'])
def subscribe():
    if request.headers['content-type'] == 'application/json':
        name = request.json["name"]
        addr = request.json["addr"]
        city = request.json["city"]
        password = request.json["password"]
        if not nameInTheDataBase(name):
            conn = sqlite3.connect('database.db')
            cursor = conn.cursor()
            oneRecord = [name, addr, city, createRandomString(), password]
            cursor.execute('INSERT INTO students VALUES (?,?,?,?,?)', oneRecord)
            conn.commit()
            conn.close()
            return 'OK\n'
        return str("The name " + name + " is already taken\n")
```



A terminal window titled "noahchatelain -- -zsh -- 105x31" showing the output of a curl POST request to 127.0.0.1:5000/subscribe. The command sent was:

```
curl -X POST 127.0.0.1:5000/subscribe -H 'Content-Type: application/json' -d '{"name":"NewStudent", "addr":"IMT", "city":"Lille", "password":"123456"}'
```

The response received was:

```
OK
```

The terminal then lists the contents of the database:

| | name | addr | city | password | |
|---|------------|--------------|-------|----------|--------|
| 2 | Noah | Wall streetz | Pariz | brmtyf | 123456 |
| 3 | NewStudent | IMT | Lille | xqosic | 123456 |
| 4 | Louise | Anywhere | Lille | qdwumm | 123456 |

Création d'une fonction `stringToSha256()` afin de n'avoir aucun mot de passe en clair dans la base de données.

```
def stringToSha256(string):
    return hashlib.sha256(string.encode()).hexdigest()
```

```
@app.route('/getNote', methods=['POST'])
def getNote():
    if request.headers['content-type'] == 'application/json':
        if request.json["name"] != "" and request.json["password"] != "":
            name = request.json["name"]
            password = stringToSha256(request.json["password"])
            if nameInTheDataBase(name):
                if nameAndPasswordExist(name, password):
                    conn = sqlite3.connect('database.db')
                    cursor = conn.cursor()
                    oneRecord = [name, password]
                    cursor.execute('SELECT name, note FROM students INNER JOIN notes ON students.pin = notes.id WHERE students.name = ? and students.password = ?', oneRecord)
                    rows = cursor.fetchall()
                    result = ""
                    for row in rows:
                        result += '{"note":' + str(row[1]) + '},'
                    result = "[" + result + "]"
                    return jsonify(result)
                else:
                    return str("Incorrect password\n")
            else:
                return str("There are no results for the name : " + name + "\n")
        else:
            return str("NOK")
```

Ajout de la possibilité de mettre à jour la photo d'un étudiant, la nouvelle photo envoyée est stockée dans le dossier `students_photo` :

```
curl -X POST -i -F "name=Noah" -F "password=123456" -F "file=@/path/to/image.png"
127.0.0.1:5000/updatePhoto
```

The screenshot shows a terminal window titled "noahchatelain — zsh — 148x24". It displays two separate curl POST requests:

- The first request is for a student named "Noah". The command is:


```
curl -X POST -i -F "name=Noah" -F "password=123456" -F "file=@/Users/noahchatelain/Desktop/TP1_CHATELAIN_GYRE-Webserveur/input/image2.png" 127.0.0.1:5000/updatePhoto
```

 Response headers include:


```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 3
Server: Werkzeug/1.0.1 Python/3.9.6 on Darwin
Date: Fri, 24 Sep 2021 20:03:19 GMT
```
- The second request is for a student named "Louise". The command is:


```
curl -X POST -i -F "name=Louise" -F "password=123456" -F "file=@/Users/noahchatelain/Desktop/TP1_CHATELAIN_GYRE-Webserveur/input/image1.png" 127.0.0.1:5000/updatePhoto
```

 Response headers include:


```
HTTP/1.1 100 Continue
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 3
Server: Werkzeug/1.0.1 Python/3.9.6 on Darwin
Date: Fri, 24 Sep 2021 20:03:34 GMT
```