TEST TECHNIQUE

Elaboré par : Omar Mrad

1- Introduction :

La notation polonaise inversée (RPN) est une méthode de notation mathématique qui permet d'effectuer des calculs de manière efficace en utilisant une pile pour stocker les opérandes. Dans le cadre de ce projet, nous avons entrepris de concevoir une calculatrice RPN en mode client/serveur.

2- Objectif du projet :

Le principal objectif de ce projet était de créer une calculatrice RPN fonctionnelle qui répond aux besoins de base des utilisateurs en matière de calculs mathématiques simples. En utilisant une approche client/serveur, nous avons pu offrir une expérience utilisateur fluide et réactive tout en maintenant la logique de calcul du côté serveur.

3- Contenu de projet :

3.1- Technologies Utilisées :

3.1.1- Backend avec FastAPI et Python 3 :

Nous avons choisi **Python 3** comme langage de programmation, avec **FastAPI** comme framework pour le développement de l'API REST. **FastAPI** offre une performance élevée et une documentation automatique, ce qui en fait un choix idéal pour ce projet.

3.1.2- Frontend avec Swagger :

Swagger a été utilisé pour la documentation de l'API et la visualisation des fonctionnalités du frontend.

Cette interface utilisateur interactive permet aux développeurs et aux utilisateurs de tester facilement les différentes fonctionnalités de l'API sans avoir besoin d'outils externes supplémentaires.

3.1.3- Docker pour le déploiement :

Docker a été intégré dans notre processus de développement pour faciliter le déploiement de l'application. En conteneurisant notre application à l'aide de **Docker**, nous avons pu garantir sa portabilité et sa cohérence lors du déploiement sur différentes plateformes et environnements. Cela a également simplifié le processus de mise à l'échelle et de gestion des dépendances de l'application.

3.2- Fonctionnalités Implémentées:

3.2.1 - Afficher les opérateurs:

```python
@app.get("/rpn/op")
async def list_all_the_operand():
    return {"operations": operations}
```

3.2.2 - Appliquer les opérateurs lors du traitement:

```python
@app.post("/rpn/op/{op}/stack/{stack_id}")
async def apply_an_operand_to_a_stack(op: str, stack_id: str):

    ########################## Vérifier si la pile existe
    if stack_id not in stacks:
        raise HTTPException(status_code=404, detail="Stack not found")

    stack = stacks[stack_id]

    ########################## Vérifier si la pile est vide
    if not stack:
        raise HTTPException(status_code=400, detail="Stack is empty")

    ########################## Vérifier si l'opération est valide
    if op not in ['+', '-', '*', 'div']:
        raise HTTPException(status_code=400, detail="Invalid operation")

    ########################## Vérifier si la pile contient au moins deux éléments pour effectuer l'opération
    if len(stack) < 2:
        raise HTTPException(status_code=400, detail="Not enough elements in stack for operation")

    ########################## Effectuer l'opération correspondante
    num2 = stack.pop()
    num1 = stack.pop()

    if op == '+':
        result = num1 + num2
    elif op == '-':
        result = num1 - num2
    elif op == '*':
        result = num1 * num2
    elif op == 'div':
        if num2 == 0:
            raise HTTPException(status_code=400, detail="Division by zero")
        result = num1 / num2

    stack.append(result)

    return {"result": result}
```

3.2.3 - Créer une nouvelle pile :

```python
@app.post("/rpn/stack")
async def create_a_new_stack():
    ############################## Générer un nouvel identifiant de pile
    stack_id = str(len(stacks) + 1)

    ############################## Créer une nouvelle pile vide
    stacks[stack_id] = []

    return {"stack_id": stack_id}
```

3.2.4 - Afficher les piles :

```python
@app.get("/rpn/stack")
def list_the_available_stacks():
    return stacks
```

3.2.5 - Supprimer une pile :

```python
@app.delete("/rpn/stack/{stack_id}")
async def delete_a_stack(stack_id: str):
    ################################### Vérifier si la pile existe
    if stack_id not in stacks:
        raise HTTPException(status_code=404, detail="Stack not found")

    ################################### Supprimer la pile correspondante
    del stacks[stack_id]

    return {"message": "Stack deleted"}
```

3.2.6 - Ajouter une nouvelle valeur à la pile :

```python
@app.post("/rpn/stack/{stack_id}")
async def push_a_new_value_to_a_stack(stack_id: str, value: float):
    # Vérifier si la pile existe
    if stack_id not in stacks:
        raise HTTPException(status_code=404, detail="Stack not found")

    ################################### Ajouter la valeur à la pile correspondante
    stacks[stack_id].append(value)

    return {"message": "Value added to stack"}
```

3.2.7 - Afficher une pile :

```python
@app.get("/rpn/stack/{stack_id}")
async def get_a_stack(stack_id: str):
    ################################### Vérifier si la pile existe
    if stack_id not in stacks:
        raise HTTPException(status_code=404, detail="Stack not found")

    ################################### Renvoyer la pile correspondante
    return {"stack_id": stack_id, "stack": stacks[stack_id]}
```

3.3- Exécution de code:

3.3.1- Sans Docker:

- uvicorn backend:app --reload

```
PS C:\Users\ASUS\Desktop\Test_CIB> uvicorn backend:app --reload
>>
INFO:     Will watch for changes in these directories: ['C:\\Users\\ASUS\\Desktop\\Test_CIB']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [17312] using StatReload
INFO:     Started server process [1828]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

3.3.2- Avec Docker

1) Création d'un fichier Dockerfile :

```
Dockerfile > ...
1   # Utilisez l'image de base Python
2   FROM python:3.9
3
4   # Copiez le code de votre application dans le conteneur
5   COPY . /app
6   WORKDIR /app
7
8   # Installez les dépendances
9   RUN pip install -r requirements.txt
10
11  # Exposez le port sur lequel l'application FastAPI fonctionne
12  EXPOSE 8000
13
14  # Commande pour démarrer l'application FastAPI
15  CMD ["uvicorn", "backend:app", "--host", "0.0.0.0", "--port", "8000"]
16
```

2) Construire l'image Docker :

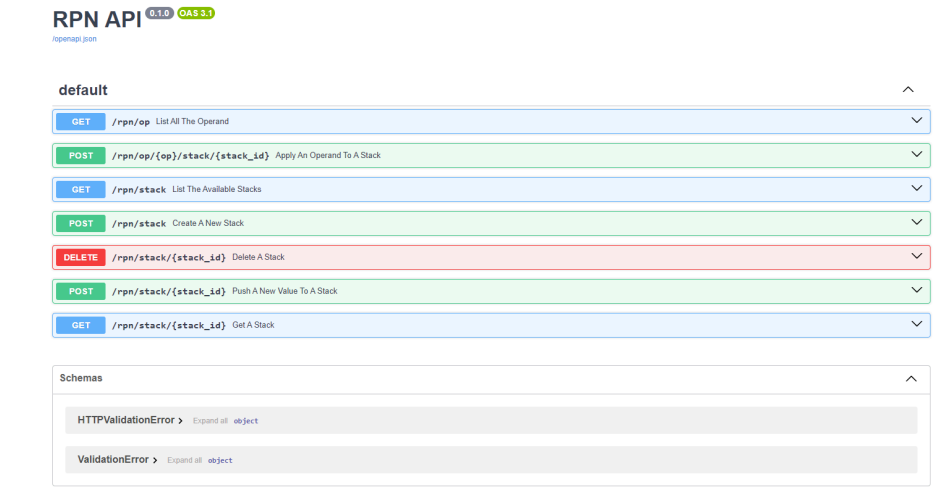
- docker build -t monapp-fastapi .

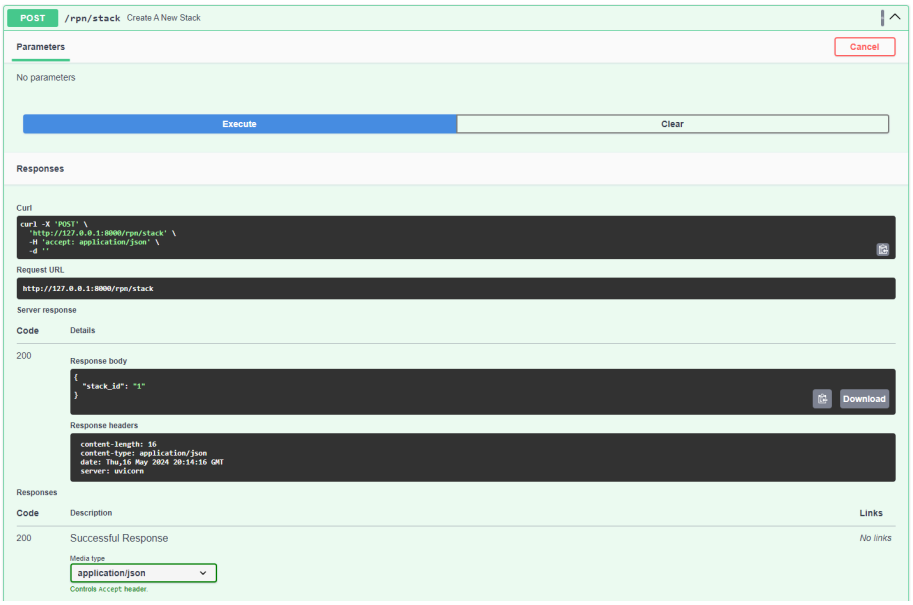
3) Exécuter le conteneur Docker :

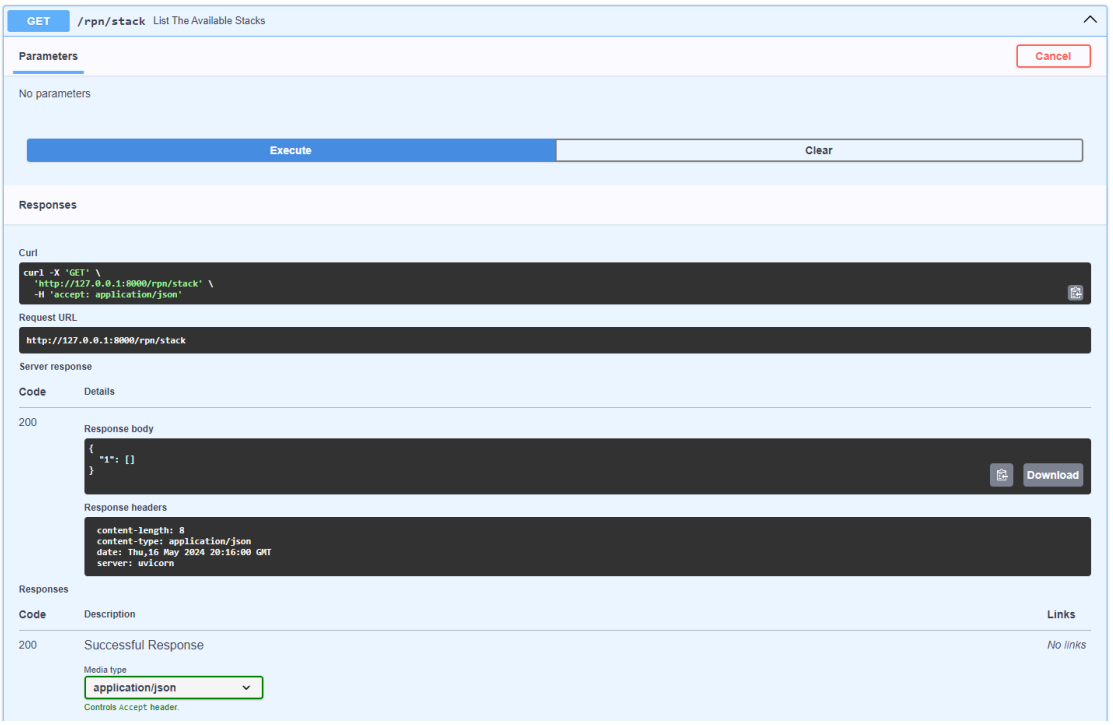
- docker run -d -p 8000:8000 monapp-fastapi

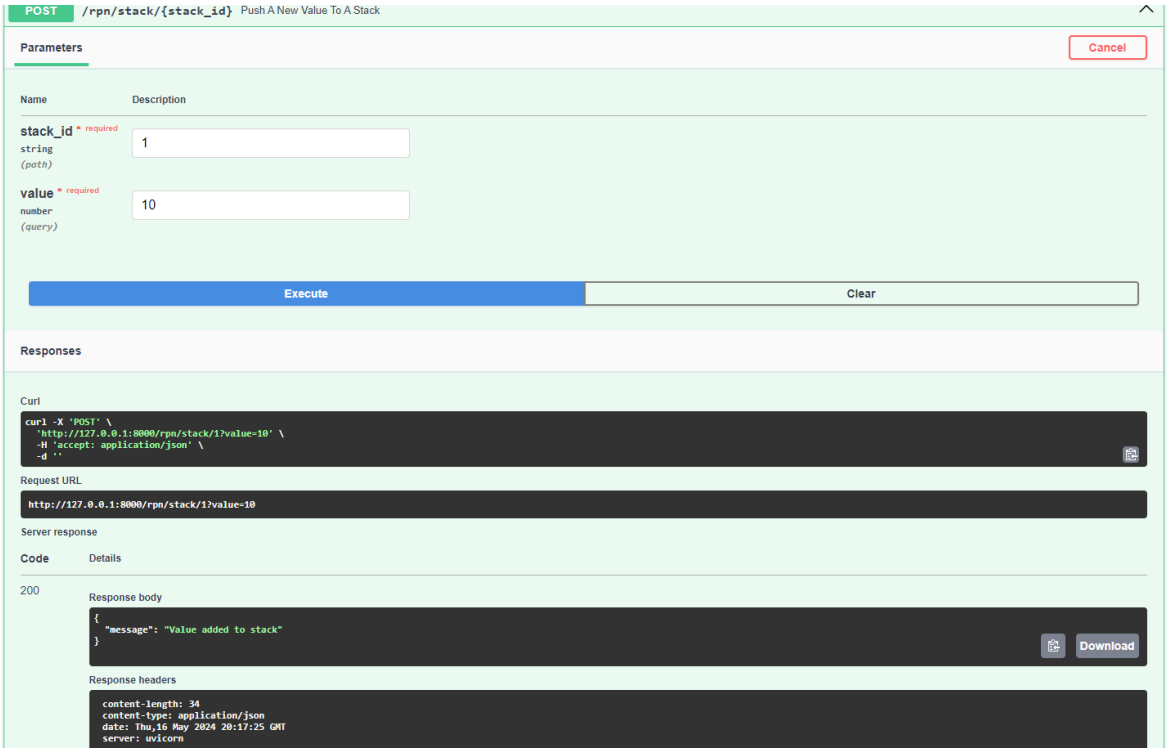
```
PS C:\Users\ASUS\Desktop\Test_CIB> docker build -t monapp-fastapi .
[+] Building 44.2s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 446B
 => [internal] load metadata for docker.io/library/python:3.9
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 3.91kB
 => CACHED [1/4] FROM docker.io/library/python:3.9@sha256:1446afd121c574b13077f4137443114cd8085f1dade5ee63c08305b5870f2b8a
 => [2/4] COPY . /app
 => [3/4] WORKDIR /app
 => [4/4] RUN pip install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:1b5d6bb542a2cb82e138120a9c22768e3f8bac0604845af4034a396b251cf1f3
 => => naming to docker.io/library/monapp-fastapi

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\ASUS\Desktop\Test_CIB> docker run -d -p 8000:8000 monapp-fastapi
```

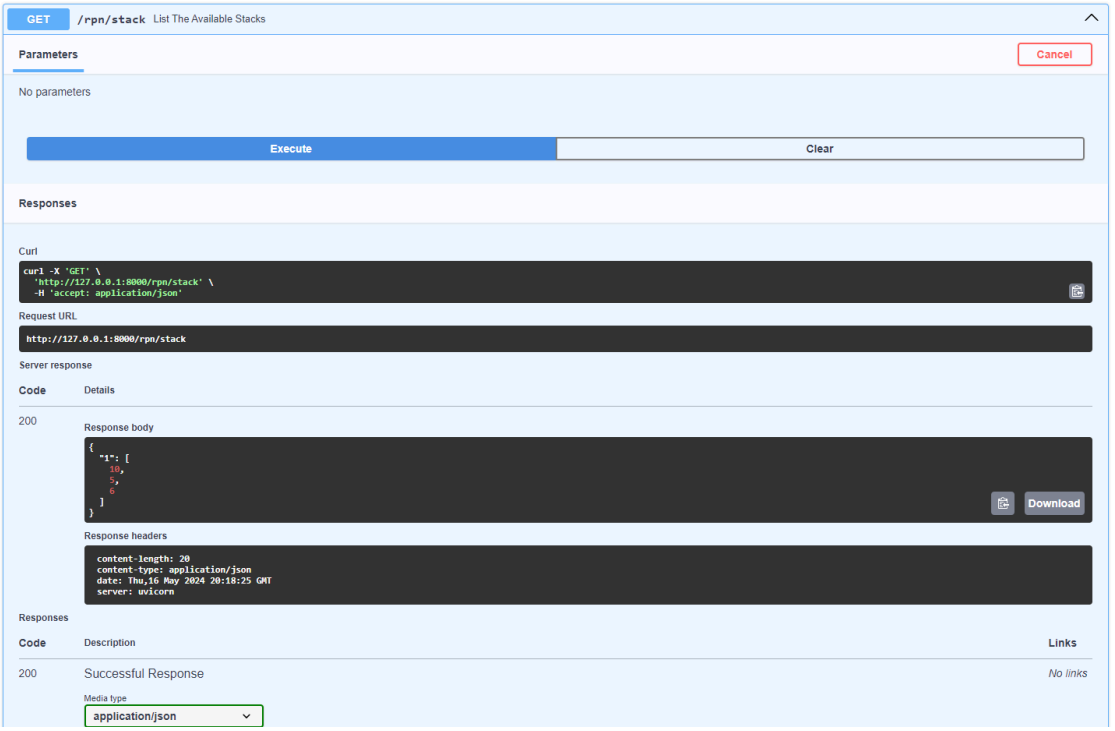
- **Exemple :**

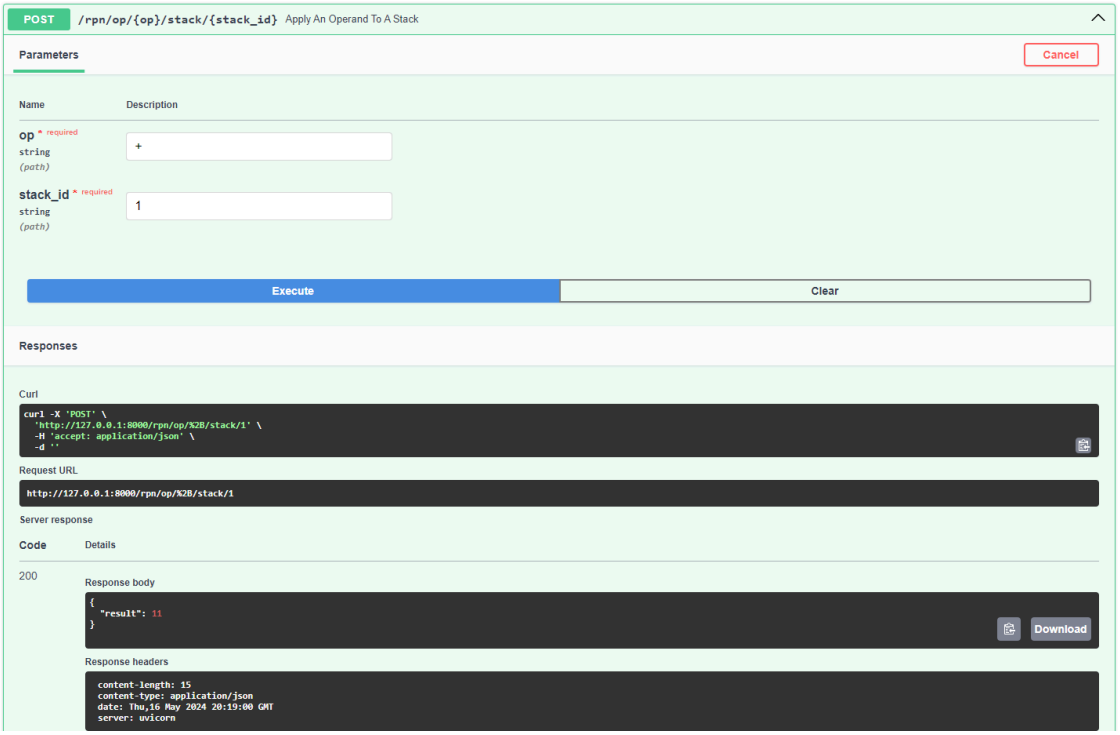


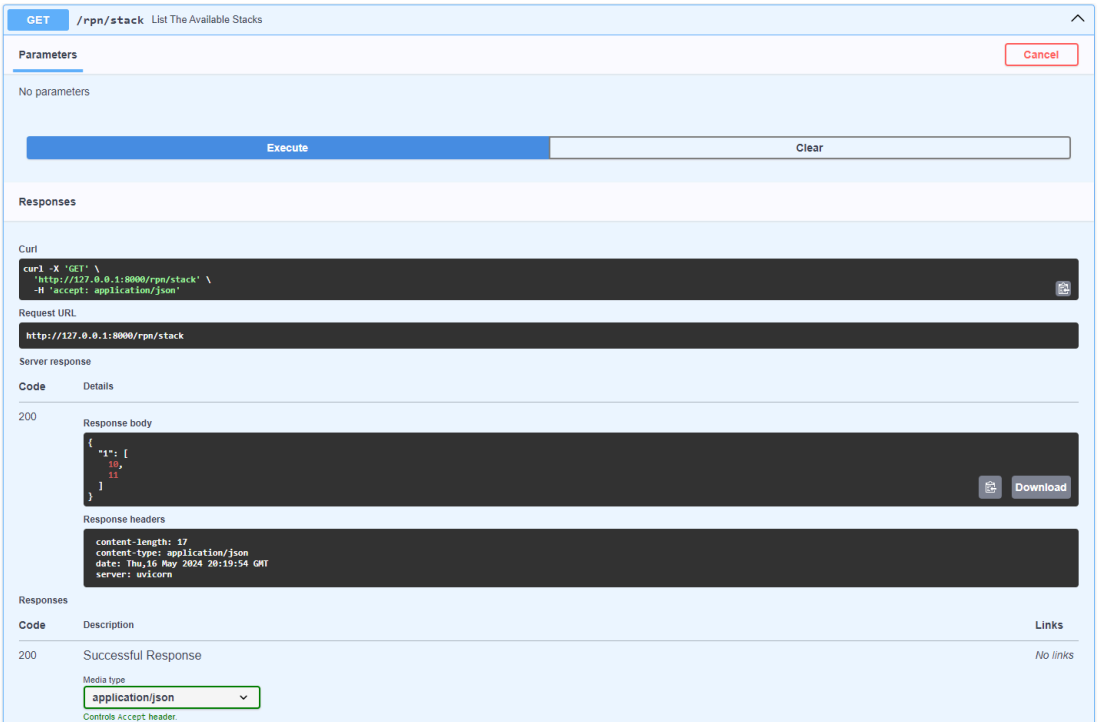



GET
/rpn/stack
List The Available Stacks
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/rpn/stack' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/rpn/stack
Server response
Code
Details
200
Response body
{
"1": []
}
Download
Response headers
content-length: 8
content-type: application/json
date: Thu,16 May 2024 20:16:00 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.


POST
/rpn/stack/{stack_id}
Push A New Value To A Stack
Parameters
Cancel
Name
Description
stack_id * required
string
(path)
1
value * required
number
(query)
10
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/rpn/stack/1?value=10' \
-H 'accept: application/json' \
-d ''
Request URL
http://127.0.0.1:8000/rpn/stack/1?value=10
Server response
Code
Details
200
Response body
{
"message": "Value added to stack"
}
Download
Response headers
content-length: 34
content-type: application/json
date: Thu,16 May 2024 20:17:25 GMT
server: uvicorn


GET
/rpn/stack
List The Available Stacks
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/rpn/stack' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/rpn/stack
Server response
Code
Details
200
Response body
{
"1": [
10,
5,
6
]
}
Download
Response headers
content-length: 20
content-type: application/json
date: Thu,16 May 2024 20:18:25 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json


POST
/rpn/op/{op}/stack/{stack_id}
Apply An Operand To A Stack
Parameters
Cancel
Name
Description
op * required
string
(path)
+
stack_id * required
string
(path)
1
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/rpn/op/%2B/stack/1' \
-H 'accept: application/json' \
-d ''
Request URL
http://127.0.0.1:8000/rpn/op/%2B/stack/1
Server response
Code
Details
200
Response body
{
"result": 11
}
Download
Response headers
content-length: 15
content-type: application/json
date: Thu,16 May 2024 20:19:00 GMT
server: uvicorn


GET
/rpn/stack
List The Available Stacks
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/rpn/stack' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/rpn/stack
Server response
Code
Details
200
Response body
{
"1": [
10,
11
]
}
Download
Response headers
content-length: 17
content-type: application/json
date: Thu,16 May 2024 20:19:54 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.

4- Conclusion :

Ce projet nous a permis de mettre en pratique nos compétences en développement web et en calcul mathématique. La conception d'une calculatrice RPN en mode client/serveur nous a confrontés à des défis techniques intéressants, notamment la gestion des opérations arithmétiques et la manipulation de la pile. Grâce à une collaboration efficace et à une planification minutieuse, nous sommes parvenus à livrer un produit fonctionnel répondant aux exigences du cahier des charges. Toutefois, des améliorations peuvent encore être apportées pour enrichir les fonctionnalités de la calculatrice et améliorer son expérience utilisateur. Le fichier "todo.md" répertorie les améliorations et les raccourcis pris en raison du temps imparti, tandis que le fichier "roadmap.md" propose une première backlog de fonctionnalités à ajouter au projet dans le futur.