anrt
agence nationale de réglementation des télécommunications
الوكالة الوطنية لتقنين المواصلات
ⵜⴰⵡⵉⵙⴰⵖⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵎⵣⵡⵓⵔⵉ ⵏ ⵜⵙⵡⵉⵍⵜ

INPT
المعهد الوطني للبريد والمواصلات
ⴰⵙⴳⵎⵓ ⴰⵏⴰⵎⵓⵔ ⵏ ⵜⵎⵣⵡⵓⵔⵉ ⴷ ⵜⵙⵡⵉⵍⵜ
Institut National des Postes et Télécommunications

2022 /2023

DATA EXTRACION USING GRAPHQL

PROJET FIN D'ANNEE

RÉALISÉE PAR
ABDELLAH GOURANE

ENCADRÉ PAR
M IKRAM EL ASRI

INTRODUCTION :

Remerciement :

Avant de commencer, je tiens à profiter de cette occasion pour remercier sincèrement mon encadrante, Mme Ikram El Asri. Ses efforts inlassables et ses précieux conseils ont été essentiels tout au long de mon projet de PFA. C'est grâce à sa poursuite constante de l'excellence et à son encouragement constant que j'ai pu évoluer et atteindre le stade actuel de mon travail.

Je suis reconnaissant(e) d'avoir eu la chance de bénéficier de sa guidance et de sa bienveillance tout au long de ce projet. Sa disponibilité et son soutien constant ont été un véritable moteur pour moi, me poussant à donner le meilleur de moi-même.

Je tiens à remercier chaleureusement Mme Ikram El Asri pour son engagement sans faille et son rôle déterminant dans la réussite de mon projet de PFA.

PRESENTATION DU PROJET :

De nos jours, l'accès à une quantité massive de données est devenu crucial dans de nombreux domaines, y compris le développement logiciel. Dans le cadre de mon projet de PFA, j'ai choisi de me concentrer sur l'extraction de données à partir de GitHub en utilisant l'API GraphQL.

L'objectif principal de mon projet est de développer un programme capable de prendre en entrée le nom d'un projet GitHub spécifique et d'extraire des informations précieuses à son sujet. Plus précisément, le programme récupérera des informations sur un certain nombre de référentiels liés au projet.

L'une des principales fonctionnalités de mon programme est d'identifier le langage de programmation le plus couramment utilisé dans ce projet. Mon programme déterminera le langage de programmation dominant et fournira en sortie le pourcentage d'utilisation de ce langage dans le projet.

Table de matières

I. **Introduction----------1**

1. Remerciement----------2
2. Présentation du projet----------3
3. Table de matières----------4

II. **Ch1 : présentation de la technologie de travail**----------5

1. C'est quoi une API ----------6
2. REST to GraphQL----------7
3. Les avantages de GraphQl----------8
4. Envoye d'une requête GraphQL ----------9
5. Notion de bases sur GraphQL----------12
6. Personal acsses token----------16

III. **Ch2 : présentation des résultats du projet**----------17

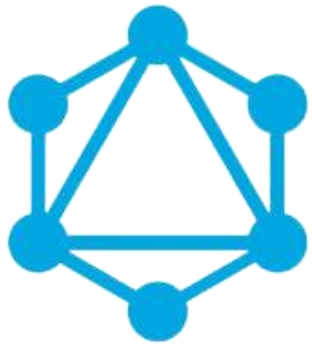
1. Python-GraphQL----------18
2. Langage d'un projet----------22

IV. **CONCLUSION :** ----------25

1. Synthèse----------26
2. Perspectives du projet----------27

CHAPITRE 1:

présentation de la technologie de travail



GraphQL

C’est quoi une API !

Une API (Application Programming Interface) est un ensemble de règles et de protocoles qui permettent à différentes applications informatiques de communiquer et d'interagir entre elles. Elle définit les méthodes, les formats de données et les conventions à suivre pour l'échange d'informations entre les différentes parties d'un logiciel ou entre différents logiciels.

En d'autres termes, une API est une interface qui permet à un programme d'accéder et d'interagir avec les fonctionnalités d'un autre programme, d'un service web ou d'une plateforme. Elle facilite l'intégration de différentes applications et permet de tirer parti des fonctionnalités offertes par d'autres systèmes sans avoir à développer tout le code nécessaire à partir de zéro.

Les API sont couramment utilisées dans de nombreux domaines, tels que les services web, les réseaux sociaux, les systèmes d'exploitation, les bases de données, les frameworks de développement, etc. Elles permettent aux développeurs d'utiliser les fonctionnalités d'une application ou d'un service tiers de manière structurée et normalisée, en évitant ainsi de devoir connaître tous les détails internes de mise en œuvre.

En résumé, une API est un moyen efficace et standardisé pour permettre à différents logiciels de communiquer et de coopérer entre eux de manière transparente, ouvrant ainsi la porte à des intégrations et à des collaborations plus faciles entre les applications.

REST to GraphQL :

GitHub, en tant que plateforme de gestion de code source populaire, a traditionnellement utilisé une API basée sur REST (Representational State Transfer) pour permettre aux développeurs d'interagir avec les fonctionnalités de la plateforme. Cependant, afin d'améliorer l'expérience des développeurs et de répondre à leurs besoins plus spécifiques, GitHub a décidé de migrer vers GraphQL.

GraphQL est un langage de requête et une spécification ouverte développée par Facebook. Contrairement à REST, où les clients doivent effectuer plusieurs appels d'API pour récupérer différentes ressources, GraphQL permet aux clients de formuler des requêtes précises pour spécifier exactement les données dont ils ont besoin. Cela permet une récupération plus efficace des données et évite les problèmes de surchargement de l'API avec des données superflues.

Les avantages de GraphQl :

La migration de GitHub vers GraphQL présente plusieurs avantages. Tout d'abord, elle permet aux développeurs de récupérer des données de manière plus efficace, en évitant les problèmes de surcharge réseau et en réduisant la quantité de données inutiles transférées. De plus, GraphQL offre une flexibilité accrue en permettant aux clients de spécifier les champs et les relations exacts dont ils ont besoin dans une seule requête, ce qui facilite le développement d'applications plus performantes et réactives.

En outre, GraphQL offre également des fonctionnalités telles que les mutations, qui permettent aux clients d'effectuer des opérations de création, de mise à jour et de suppression de manière cohérente et simplifiée. Cela simplifie le processus de modification des données sur la plateforme GitHub.

Envoye d'une requête GraphQL :

J'ai identifié trois méthodes d'exécution de GraphQL avec GitHub :

1- Interroger GraphQL dans une commande curl : j'effectue une requête POST avec une charge utile JSON. La charge utile doit contenir une chaîne nommée query voilà un exemple :

```
curl -H "Authorization: bearer TOKEN" -X POST -d " \
{ \ \"query\": \"query { viewer { login }}\" \ } \ "
https://api.github.com/graphql
```

- La commande curl est utilisée pour envoyer une requête HTTP à un serveur.
- L'option **-H** est utilisée pour inclure l'en-tête HTTP Autorisation, qui **contient un jeton d'accès d'authentification valide.**
- L'option **-X** spécifie le type de requête HTTP à envoyer, qui est POST dans ce cas.
- L'option **-d** est utilisée pour inclure la charge utile JSON de la requête GraphQL.
- La chaîne nommée "query" dans la charge utile contient la requête GraphQL à envoyer. Dans cet exemple, la requête demande le nom d'utilisateur GitHub du "viewer" actuel.

On peut exécuter directement cette commande dans le CMD !!!

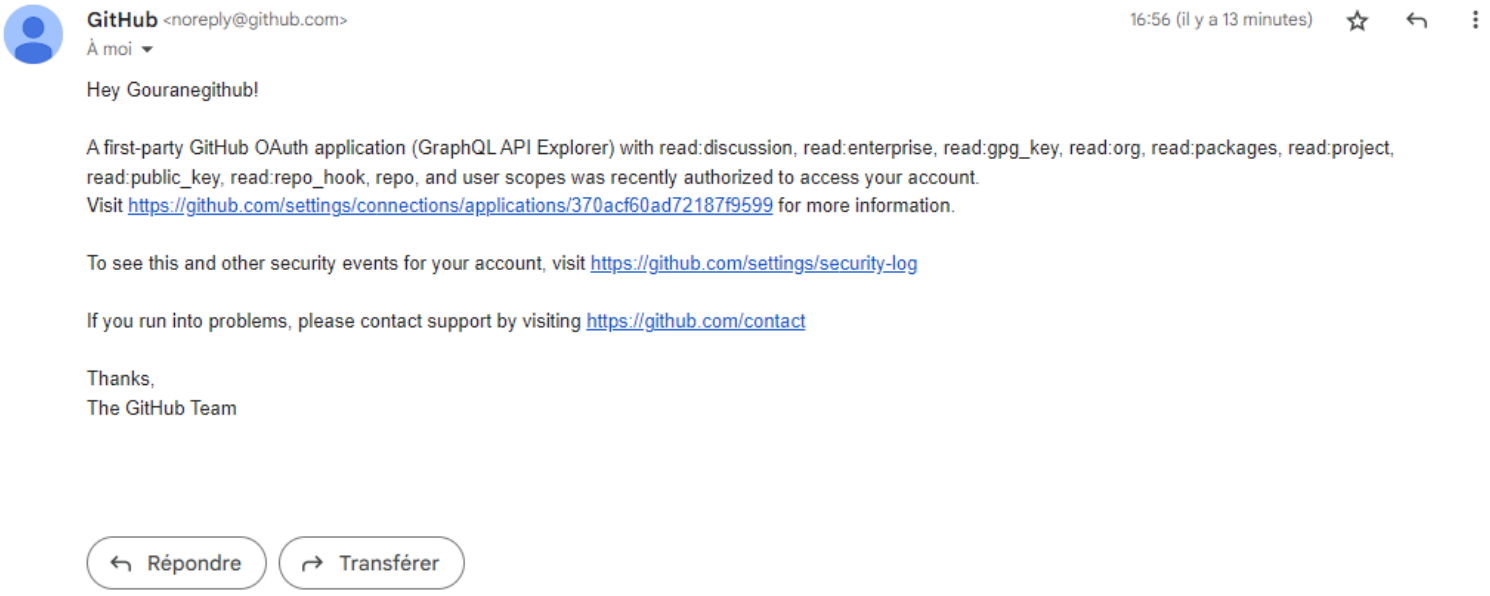
2- Exécuter une requête graphql Dans un code python (Js…) : il y'a des bibliothèques que j'ai déjà installées dans mon ordinateur permettant d'envoyer des requête graphql au serveur graphql

```
C:\Windows\system32>pip install graphene
Collecting graphene
  Downloading graphene-3.2.2-py2.py3-none-any.whl (125 kB)
     ---------------------------------------- 125.6/125.6 kB 923.2 kB/s eta 0:00:00
Collecting graphql-core<3.3,>=3.1
  Downloading graphql_core-3.2.3-py3-none-any.whl (202 kB)
     ---------------------------------------- 202.9/202.9 kB 1.5 MB/s eta 0:00:00
:\Windows\system32>pip install starlette-graphql
ollecting starlette-graphql
 Downloading starlette_graphql-0.2.1-py3-none-any.whl (7.2 kB)
ollecting python-gql>=0.2
 Downloading python_gql-0.2.4-py3-none-any.whl (24 kB)
equirement already satisfied: starlette>0.13 in c:\users\abdellah\appdata\local\programs\pyt
ges (from starlette-graphql) (0.26.1)
equirement already satisfied: graphql-core<4,>=3 in c:\users\abdellah\appdata\local\programs
ackages (from python-gql>=0.2->starlette-graphql) (3.2.3)
equirement already satisfied: anyio<5,>=3.4.0 in c:\users\abdellah\appdata\local\programs\py
ages (from starlette>0.13->starlette-graphql) (3.6.2)
equirement already satisfied: idna>=2.8 in c:\users\abdellah\appdata\local\programs\python\p
from anyio<5,>=3.4.0->starlette>0.13->starlette-graphql) (3.4)
equirement already satisfied: sniffio>=1.1 in c:\users\abdellah\appdata\local\programs\pytho
s (from anyio<5,>=3.4.0->starlette>0.13->starlette-graphql) (1.3.0)
nstalling collected packages: python-gql, starlette-graphql
uccessfully installed python-gql-0.2.4 starlette-graphql-0.2.1

notice] A new release of pip available: 22.3.1 -> 23.1
notice] To update, run: python.exe -m pip install --upgrade pip
```

```
C:\Windows\system32>pip install fastapi
Collecting fastapi
  Downloading fastapi-0.95.1-py3-none-any.whl (56 kB)
     ---------------------------------------- 57.0/57.0 kB 599.1 kB/s eta 0:00:00
Collecting pydantic!=1.7,!=1.7.1,!=1.7.2,!=1.7.3,!=1.8,!=1.8.1,<2.0.0,>=1.6.2
  Downloading pydantic-1.10.7-cp311-cp311-win_amd64.whl (2.1 MB)
     ---------------------------------------- 2.1/2.1 MB 1.3 MB/s eta 0:00:00
:\Windows\system32>pip install starlette
ollecting starlette
  Downloading starlette-0.26.1-py3-none-any.whl (66 kB)
     ---------------------------------------- 66.9/66.9 kB 726.6 kB/s eta 0:00:00
ollecting anyio<5,>=3.4.0
  Downloading anyio-3.6.2-py3-none-any.whl (80 kB)
     ---------------------------------------- 80.6/80.6 kB 902.3 kB/s eta 0:00:00
ollecting idna>=2.8
  Downloading idna-3.4-py3-none-any.whl (61 kB)
     ---------------------------------------- 61.5/61.5 kB 1.7 MB/s eta 0:00:00
```

N.B : Pour avoir l'accès aux données du serveur graphql Il faut avoir une propre jeton d'accès d'authentification valide « personal acces token » j'ai déjà demander une :



GitHub <noreply@github.com> 16:56 (il y a 13 minutes)

À moi

Hey Gouranegithub!

A first-party GitHub OAuth application (GraphQL API Explorer) with read:discussion, read:enterprise, read:gpg_key, read:org, read:packages, read:project, read:public_key, read:repo_hook, repo, and user scopes was recently authorized to access your account.
Visit https://github.com/settings/connections/applications/370acf60ad72187f9599 for more information.

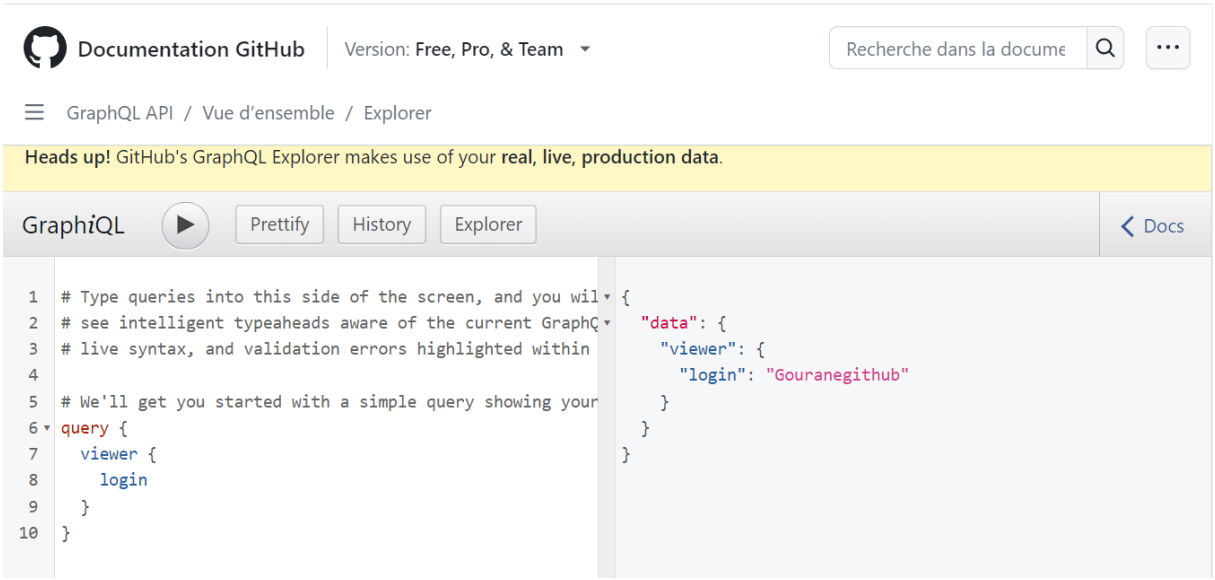
To see this and other security events for your account, visit https://github.com/settings/security-log

If you run into problems, please contact support by visiting https://github.com/contact

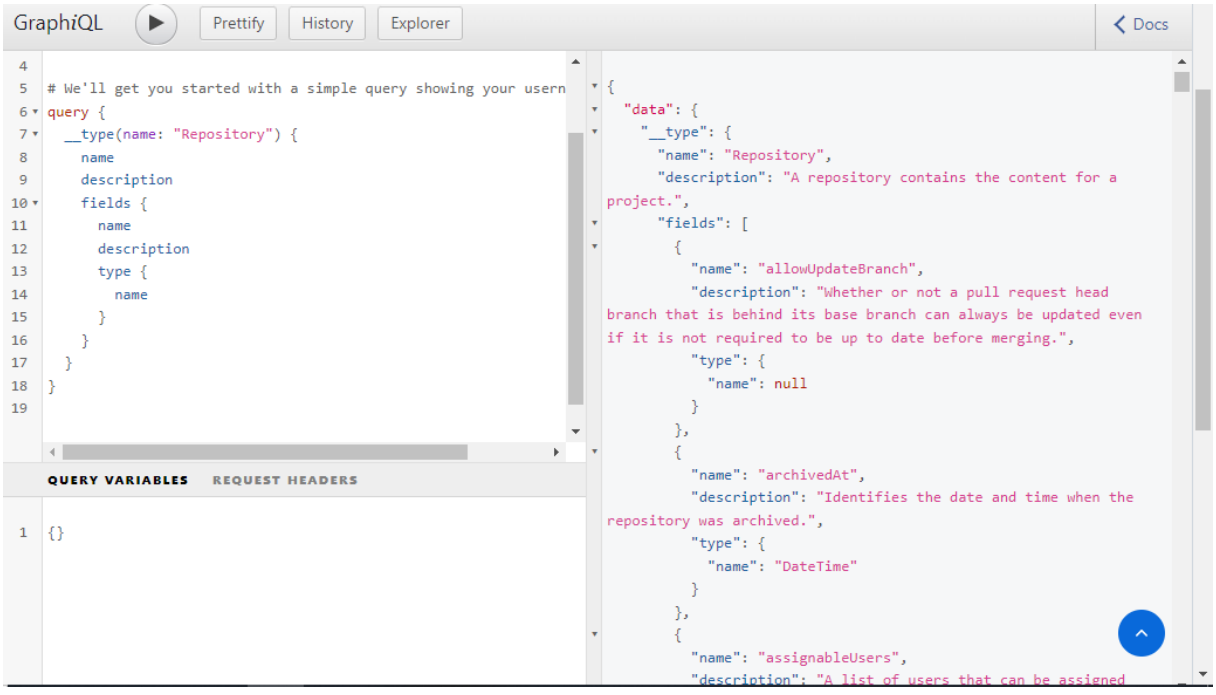
Thanks,
The GitHub Team

Répondre Transférer

3- On peut directement écrire des requête graphql dans github graphql explorer après avoie connecter avec mon compte :



Je vais utiliser la troisième méthode pour me familiariser avec graphql . Donc pour pouvoir manipuler les donné il faut d'abord avoir le schéma . et pour connaitre ce schéma on peut écrire une requête graphql pour le demander :



GraphQL :

- Il’y a pas juste les requetés il y’a aussi les mutation :

Les requêtes : sont utilisées pour récupérer des données à partir d'un schéma GraphQL

Vocabulaire : "Edges" est un terme utilisé dans le contexte des relations entre les données dans GraphQL. Les "edges" sont les liens entre les nœuds (ou "nodes") dans un graphe. Par exemple, si vous avez un graphe représentant une base de données de films, les "edges" seraient les liens entre les films, les acteurs, les réalisateurs, etc.

"Node" désigne simplement un objet dans un graphe. Dans l'exemple de la base de données de films, un "node" pourrait représenter un film spécifique, avec toutes ses informations, comme le titre, la date de sortie, les acteurs, etc.

Un "repository" est un terme utilisé en informatique pour désigner un emplacement de stockage de données, de code source ou de fichiers binaires. Il est également souvent appelé "repo" pour faire court

Le terme "viewer" fait référence à l'utilisateur qui effectue la requête GraphQL. Dans le contexte de GraphQL, le "viewer" est considéré comme le nœud racine de l'arbre de requête.

Les mutations : sont utilisées pour modifier les données dans un schéma GraphQL.

Pour former une mutation, vous devez spécifier trois éléments :

1. Nom de la mutation. Type de modification que vous souhaitez effectuer.
2. Objet d’entrée. Données que vous souhaitez envoyer au serveur, composées de champs d’entrée. Passez-les comme argument au nom de la mutation.

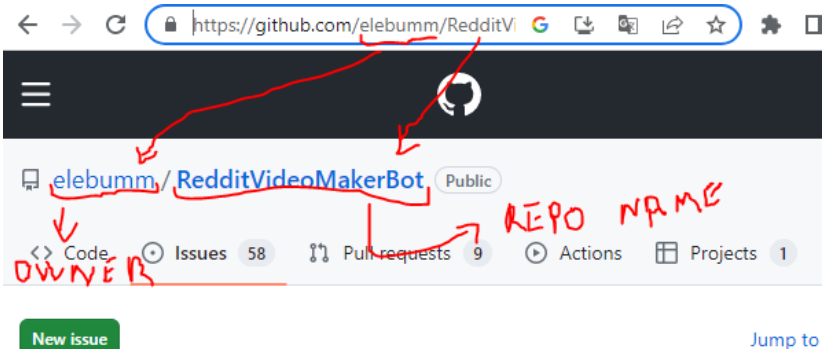
3. Objet de charge utile. Données que vous souhaitez retourner à partir du serveur, composées de champs de retour. Passez-les comme corps du nom de la mutation.

Voici quelques notions de base sur GraphQL :

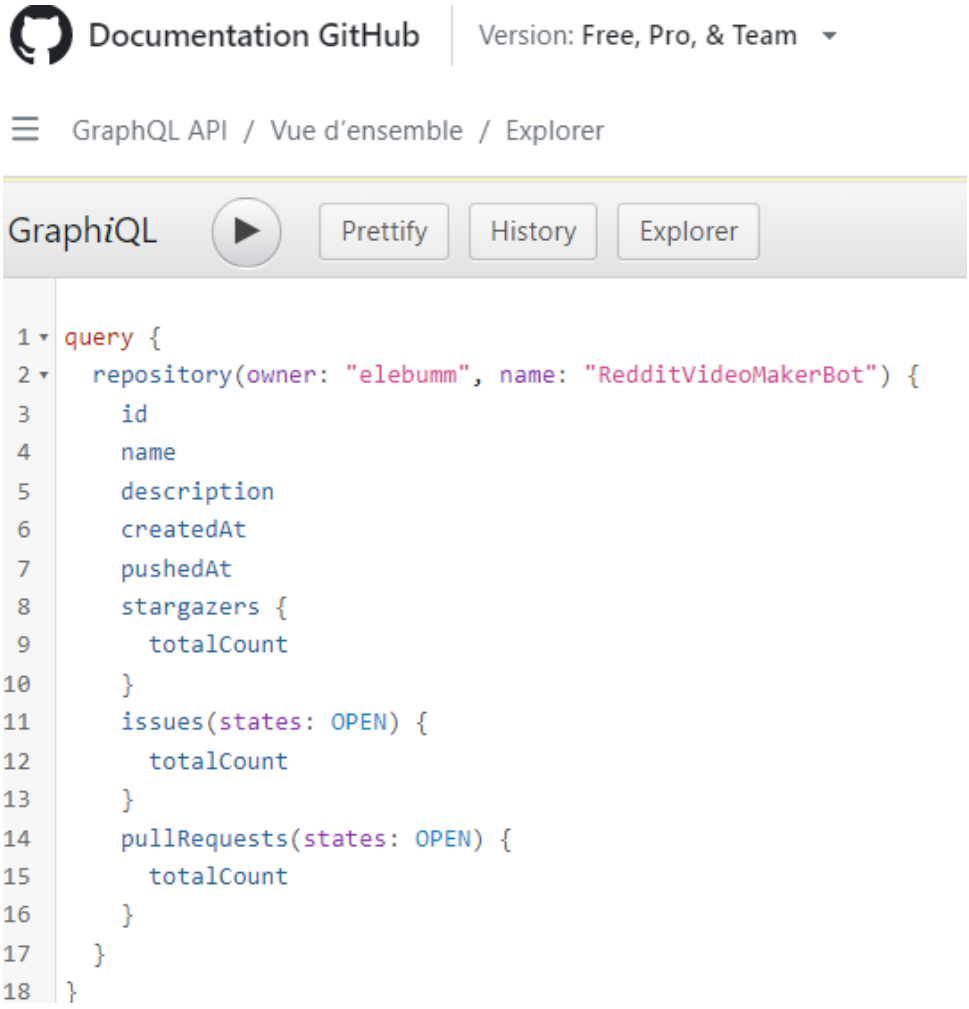
- ✓ Schéma : Le schéma GraphQL définit les types d'objets disponibles et les relations entre ces objets. Il agit comme une sorte de contrat entre le client et le serveur, définissant les opérations et les données pouvant être demandées.
- ✓ Types : GraphQL propose des types de base tels que String, Int, Boolean, etc., ainsi que la possibilité de définir des types personnalisés. Les types personnalisés permettent de modéliser les données spécifiques à votre domaine.
- ✓ Query : Une requête GraphQL est utilisée pour demander des données spécifiques au serveur. Elle suit une structure de type arborescente, où vous spécifiez les champs que vous souhaitez récupérer pour chaque objet demandé.
- ✓ Mutation : Une mutation GraphQL est utilisée pour effectuer des modifications sur le serveur, telles que la création, la mise à jour ou la suppression de données. Les mutations sont définies dans le schéma et peuvent avoir des arguments pour fournir les données nécessaires.
- ✓ Résolveurs : Les résolveurs sont des fonctions qui sont responsables de récupérer les données demandées dans une requête GraphQL. Chaque champ dans le schéma a un résolveur correspondant qui indique comment obtenir la valeur de ce champ.
- ✓ Variables : GraphQL prend en charge l'utilisation de variables dans les requêtes. Les variables permettent de paramétrer dynamiquement les requêtes et de les rendre réutilisables.
- ✓ Introspection : GraphQL offre la possibilité d'effectuer une introspection sur le schéma, ce qui signifie que vous pouvez interroger le serveur pour obtenir des informations sur les types disponibles, les champs et les relations.
- ✓ Subscriptions : GraphQL prend également en charge les abonnements, qui permettent aux clients de s'abonner à des événements spécifiques sur le serveur et de recevoir des mises à jour en temps réel.

Extraction !

avand d'extraire des donner je dois avoir des informations sur le dépôt « repository » qu'on va aborder . On a besoin Just du nom du repo et le propriétaire « owner » . On les trouve en haut du repo. Exp :



On ajout ces deux info à notre requête « query » :



Voilà un exemple dans lequel je demande des info sur le repo comme l'id, nam , description, pullrequests etc…

On remarque que les services GraphQL répondent généralement en utilisant JSON, cependant la spécification GraphQL ne l'exige pas. JSON peut sembler un choix étrange pour une couche d'API promettant de meilleures performances réseau

la réponse devrait être renvoyée dans le corps de la requête au format JSON. Comme mentionné dans la spécification, une requête peut retourner des données ainsi que des erreurs, et celles-ci doivent être renvoyées dans un objet JSON de la forme suivante : { "data": { ... }, "errors": [...] } Si aucune erreur n'est renvoyée, le champ "errors" ne doit pas être présent dans la réponse. Si aucune donnée n'est renvoyée, selon la spécification GraphQL, le champ "data" ne doit être inclus que s'il n'y a pas eu d'erreurs lors de l'exécution.

Personal acsses token :

Un jeton d'accès personnel (Personal Access Token en anglais) est un élément de sécurité utilisé pour authentifier et autoriser les requêtes API effectuées vers des services tels que GitHub. Il agit comme un substitut de votre mot de passe et vous permet d'effectuer des requêtes API authentifiées au nom de votre compte utilisateur.

Pour générer un jeton d'accès personnel sur GitHub il faut :

1. Connectez-vous à votre compte GitHub.
2. Cliquez sur votre photo de profil en haut à droite, puis sélectionnez "Paramètres" dans le menu déroulant.
3. Dans la barre latérale gauche, cliquez sur "Paramètres développeur".
4. Sélectionnez "Jetons d'accès personnel" dans le menu.
5. Cliquez sur le bouton "Générer un nouveau jeton".
6. Donnez un nom significatif à votre jeton et sélectionnez les autorisations ou les scopes que vous souhaitez accorder au jeton.
7. Une fois que vous avez configuré les paramètres du jeton, cliquez sur le bouton "Générer le jeton".
8. GitHub générera votre jeton d'accès personnel. Assurez-vous de le copier et de le stocker en toute sécurité dans un endroit sûr, car il ne sera plus affiché.

CHAPITRE 2:

présentation des résultats du projet

Python-GraphQL :

Pour commencer il faut importer d'abord deux bibliothèque, requires pour envoyer la requet et JSON pour pouvour lire et manipuler la répence json retourné

```
import requests
import json
query = '''

   query {
  repository(owner: "mchelali", name: "Analyse-et-traitement-d-Image") {

    defaultBranchRef {
      target {
        ... on Commit {
            oid
            committedDate
            author{
              name
              email
            }
            message

        }
      }
    }

    pullRequests(first :3) {
       edges {
        node {
          number
          title
          createdAt
          updatedAt
          author {
            login
          }
          reviews(first: 3){
            edges{
              node{
```

```
                author{
                login
                }
                body
              }
            }
          }
        }
      }

    }

   primaryLanguage { name }

    collaborators(affiliation: ALL, first: 10) {

        edges {
          node {
            login
          }
        }
      }

  }
}


'''

headers = {'Authorization':  'Bearer
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', 'Accept':
'application/vnd.github.v3+json'}
url = 'https://api.github.com/graphql'
response = requests.post(url, json={'query': query}, headers= headers)


dataa = json.loads(response.text)
data = dataa['data']
print(f"the Data returned is: {data}")

#the commit informations
commit_oid = data['repository']['defaultBranchRef']['target']['oid']
commit_date =
data['repository']['defaultBranchRef']['target']['committedDate']
commit_author_name =
data['repository']['defaultBranchRef']['target']['author']['name']
commit_author_email =
data['repository']['defaultBranchRef']['target']['author']['email']
commit_message = data['repository']['defaultBranchRef']['target']['message']
```

```
#language utilisé
commit_oid = data['repository']['primaryLanguage']['name']



print('the oid of the commit is:'+commit_oid+"\n")
print('the date of the commit is:'+commit_date+"\n")
print('the author of the commit is:'+commit_author_name +'and his email
is'+commit_author_email+"\n")
print('the message in the commit is:'+commit_message+"\n")
```

D'abord j'affect la requête que j'ai écrit dans une variable dans ce cas j'ai nommé la variable query

Pour envoyer la requête en utilisons requetés je serai besoin de trois paramètres l'url de l'api dans lequel je vais envoyer la requête, et finalement un paramètre qui s'appelle headers, il contient l'Authentification (the Personal acsses token) etc…

J'itualise Json.load (response.text) pour lire la réponse json je la stock dans une variable data et après j'accède tous simplement aux informations retournées

Le résulta :

```
the Data returned is: {'repository': {'defaultBranchRef': {'target': {'oid':
'a4c4e572222dd4d6b0105b3a12348f02e1c4f3f7', 'committedDate': '2021-09-
29T09:42:40Z', 'author': {'name': 'CHELALI Mohamed', 'email':
'mouha.chelali@gmail.com'}, 'message': 'maj'}}, 'pullRequests': {'edges': []},
'primaryLanguage': {'name': 'Java'}, 'collaborators': None}}

the oid of the commit is:Java

the date of the commit is:2021-09-29T09:42:40Z

the author of the commit is:CHELALI Mohamedand his email is
mouha.chelali@gmail.com

the message in the commit is:maj
```

Langage d’un projet :

```
import requests

import json
from collections import Counter

query = '''
query($searchQuery: String!, $number: Int!) {
  search(query: $searchQuery, type: REPOSITORY, first: $number) {
    edges {
      node {
        ... on Repository {
          name
          description
          primaryLanguage { name }
        }
      }
    }
  }
}
'''

# Define the variable values
variables = {
  'searchQuery': input('Enter your search query: '),
  'number': int(input('how many project u need to analyse: '))
}

payload = {
  'query': query,
  'variables': variables
}



headers = {'Authorization':  'Bearer
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', 'Accept':
'application/vnd.github.v3+json'}
url = 'https://api.github.com/graphql'
response = requests.post(url, json=payload, headers= headers)

dataa = json.loads(response.text)
data = dataa['data']


#print(f"the Data returned is: {data}")
Liste_lang = []
```

```
nuber_of_results=len(data['search']['edges'])

j=variables['number']
for i in range(0,min(j,nuber_of_results),1):
   repo_name = data['search']['edges'][i]['node']['name']
   repo_description = data['search']['edges'][i]['node']['description']
   primary_Language = data['search']['edges'][i]['node']['primaryLanguage']
   repo_language = primary_Language['name'] if primary_Language else None

   Liste_lang.append(repo_language)

   print('THE REPO NAME: '+repo_name)
   print('DESCRIPTION: '+repo_description)
   print('THE LANGUAGE USED: '+repo_language)

#print(Liste_lang)


counter = Counter(Liste_lang)
most_common = counter.most_common(1)[0]
most_common_element = most_common[0]
count = most_common[1]

print(f"The most repeated language is {most_common_element} (repeated {count}
times) wich means {count*100/len(Liste_lang)}% of the projects use
{most_common_element}  ")
```

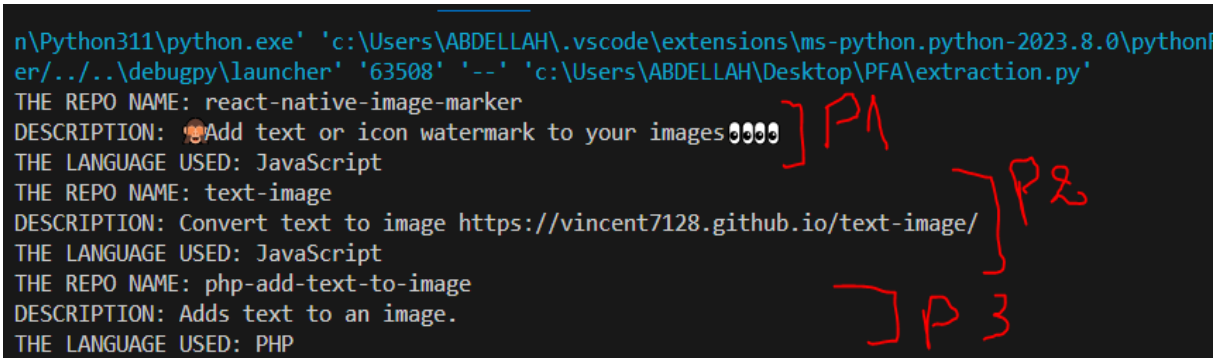
le programme demande d'entre un nom d'un projet :

```
PS C:\Users\ABDELLAH\Desktop\OCR projec
\Python311\python.exe' 'c:\Users\ABDELL
auncher' '61175' '--' 'c:\Users\ABDELLA
Enter your search query: OCR
```

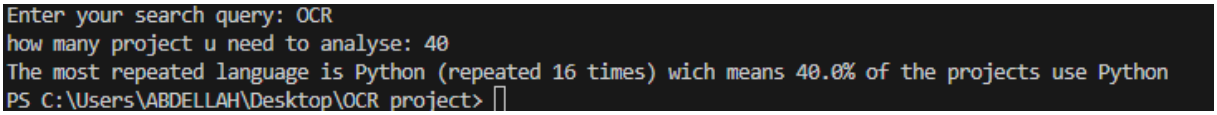
Le nombre de repo que vous voulez analyser

```
auncher' '61175' '--' 'c:\Users\ABDELLAH\Des
Enter your search query: OCR
how many project u need to analyse: 40
```

Il sélection dans ce cas par exemple les 40 premiers repo et retourne des infos sur chaque repo



Et finalemen il donne le longuage de programation le plus utilisé dans ce type de projets



CONCLUSION

Synthèse

En résumé, le projet vise à développer un programme permettant d'extraire des données à partir de GitHub en utilisant l'API GraphQL. Il implique l'utilisation de méthodes d'exécution GraphQL avec GitHub, l'authentification avec un jeton d'accès personnel, et la manipulation des réponses au format JSON. Ce projet offre une opportunité d'explorer les fonctionnalités de GraphQL, d'interagir avec une plateforme populaire comme GitHub et de développer des compétences en matière de récupération et d'analyse de données.

Perspectives du projet

le projet de récupération de données à partir de GitHub en utilisant l'API GraphQL offre de nombreuses perspectives d'amélioration, d'extension et de valorisation. Il est important d'évaluer les besoins des utilisateurs, de suivre les évolutions technologiques et de continuer à développer et à affiner le projet pour le rendre encore plus utile et performant. Donc Généralement L'extraction de données à partir de GitHub offre de nombreuses possibilités d'utilisation et d'exploration. Voici quelques exemples des différentes actions que vous pouvez entreprendre avec les données extraites :

- Analyse de tendances : Utiliser les données
- Évaluation de la qualité du code : Utiliser les données extraites pour évaluer la qualité du code des projets sur GitHub, en identifiant les bonnes pratiques, les erreurs courantes, etc.
- Analyse de la communauté : Utiliser les données entifiant les contributeurs les plus actifs, les collaborations fructueuses, etc.
- Aide à la prise de décision : Utiliser les données extraites pour aider les développeurs à prendre des décisions éclairées sur les langages, les bibliothèques, les frameworks, etc. à utiliser dans leurs propres projets.
- Génération de rapports automatisés : Utiliser les données extraites pour générer automatiquement des rapports sur l'état et les performances des projets sur GitHub, en fournissant des informations utiles aux parties prenantes.