

# Présentation et Explication du Code Python GLPI API

# Introduction

Ce code permet d'interagir avec l'API GLPI pour gérer des tickets et des utilisateurs de manière automatisée. Il est conçu pour :

- ✓ Créer une session avec l'API GLPI
- ✓ Créer un utilisateur
- ✓ Créer un ticket pour cet utilisateur

# 1. Pourquoi ce projet ?

Le but de ce projet est d'automatiser la gestion des tickets dans GLPI en interagissant avec l'API REST.

Plutôt que de faire ces tâches manuellement dans l'interface web de GLPI, on utilise un script Python qui :

- Établit une connexion sécurisée à l'API
- Crée des utilisateurs dynamiquement
- Génère des tickets automatiquement

# 2.Dépendances utilisées et justification

Nous utilisons plusieurs bibliothèques Python pour rendre le code plus modulaire, sécurisé et facile à maintenir.

- La bibliothèque **requests** est utilisée pour effectuer des requêtes HTTP vers l'API GLPI. Elle permet d'envoyer des requêtes GET, POST, PUT, etc., et de traiter les réponses du serveur.
- La bibliothèque **json** est essentielle pour formater et lire les réponses JSON de l'API. GLPI retourne ses réponses au format JSON, il est donc important de pouvoir les manipuler facilement en Python.
- La bibliothèque **os** est utilisée pour récupérer des variables d'environnement. Cela permet d'accéder à des informations comme l'URL de l'API, les identifiants de connexion et les tokens d'authentification sans les inclure directement dans le code.

La bibliothèque **dotenv** permet de stocker les informations sensibles dans un fichier **.env** au lieu de les coder en dur. Cela améliore la sécurité en évitant que des données comme le mot de passe ou le token d'authentification ne soient visibles directement dans le script.

## Pourquoi utiliser dotenv ?

Plutôt que de mettre les informations sensibles (**URL, login, tokens**) directement dans le code, on les stocke dans un fichier **.env**. Cela permet de :

- Sécuriser les informations sensibles ✓
- Faciliter la maintenance ✓
- Pouvoir changer facilement les paramètres (ex: URL, token) sans modifier le code ✓

## ❖ 2.5. Téléchargement de GLPI et configuration de l'App Token pour la sécurité

### **Explication :**

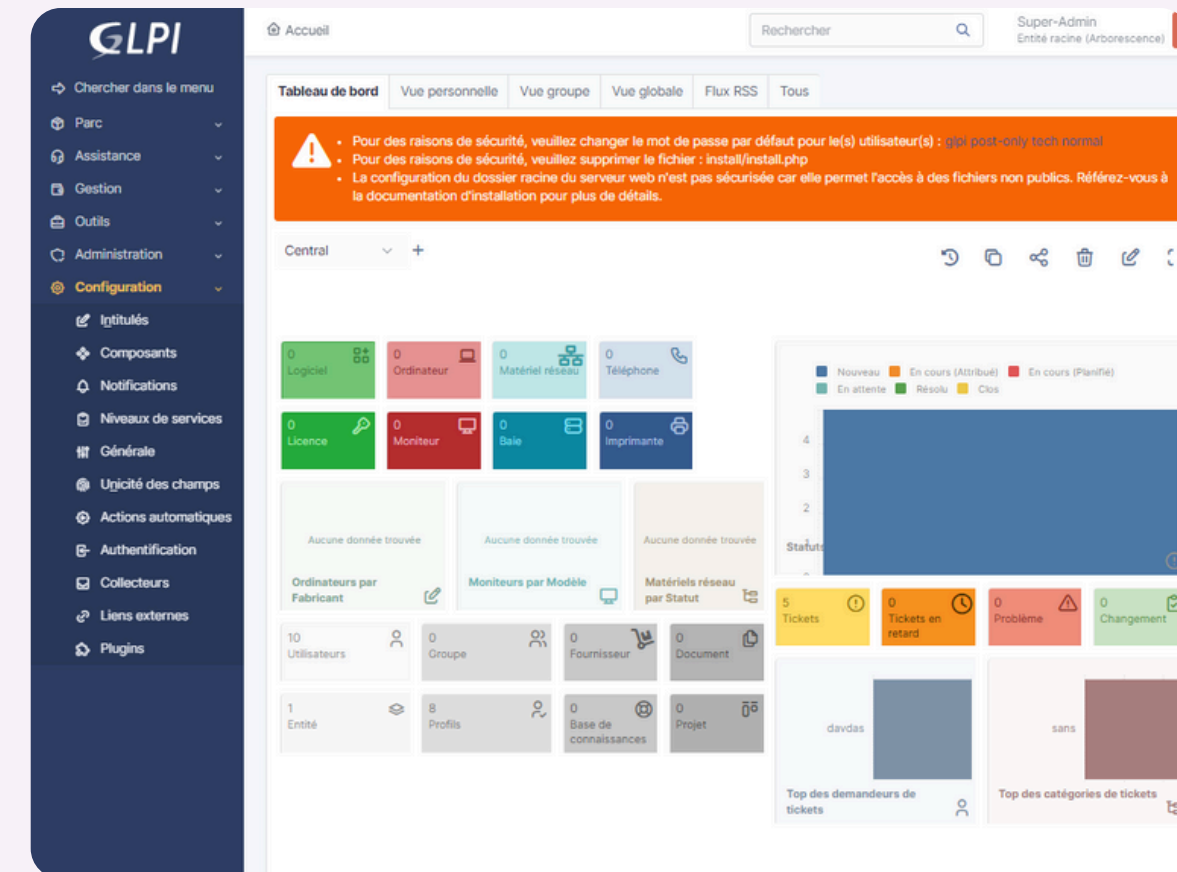
- Téléchargez Glpi sur <https://glpi-project.org/fr/telecharger-glpi/>
- Ajoutez-le sur votre serveur local, par exemple dans le dossier htdocs de XAMPP.
- Décompressez l'archive GLPI et lancez-le sur votre serveur local. Vérifiez que toutes les dépendances sont à jour.
- Connectez-vous avec les identifiants par défaut :
- Identifiant : glpi
- Mot de passe : glpi
- Dans le menu de gauche, accédez à Configuration → Générale → API.



Presentation 1



Presentation 2

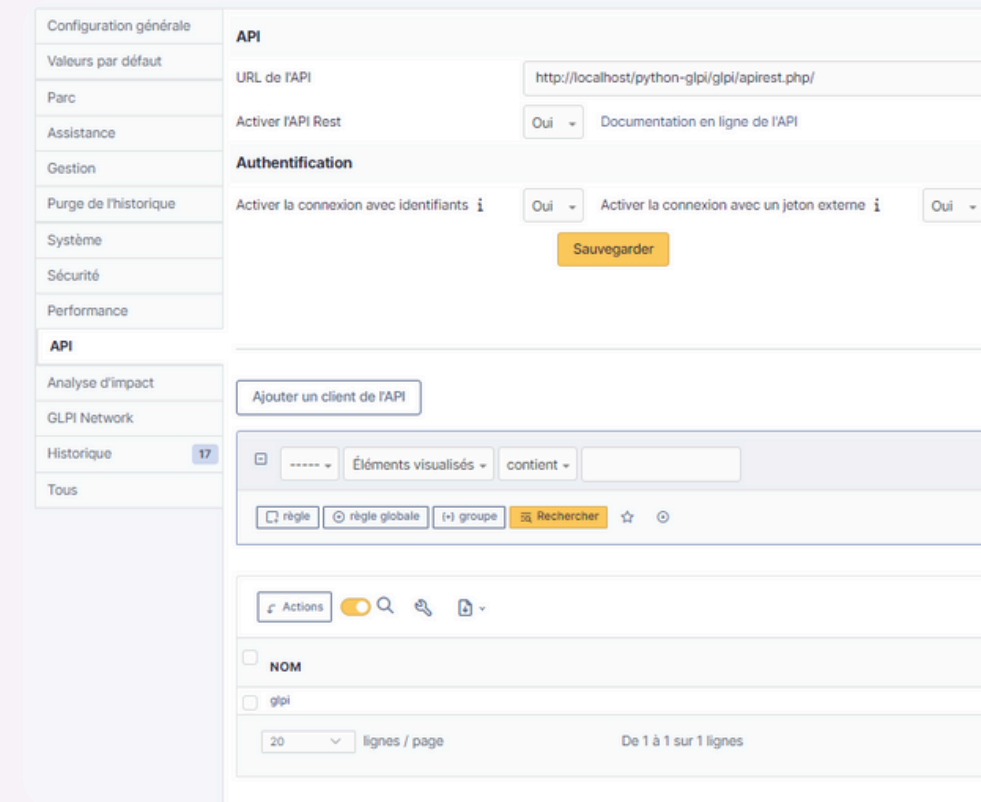


Presentation 3

### 3. Telechargement de Glpi et App\_token pour l'aspect sécurité

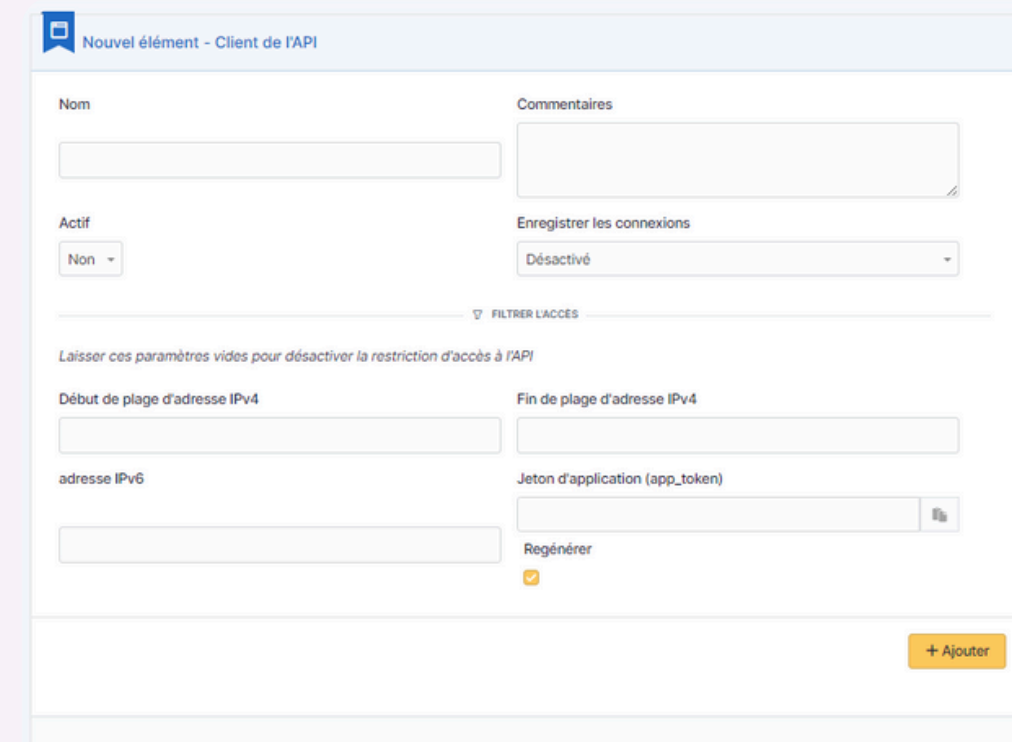
#### **Explication :**

- Vérifiez que l'URL de l'API est correcte, puis activez les autres options.
- Cliquez sur "Ajouter un client de l'API", puis remplissez le formulaire selon vos préférences.
- Votre nom s'affichera en bas de la section "Ajouter un client de l'API".
- En enregistrant, vous obtiendrez votre App Token, qui vous permettra d'effectuer des manipulations via l'API.



The screenshot shows the 'API' configuration page in GLPI. On the left is a sidebar menu with options like 'Configuration générale', 'Valeurs par défaut', 'Parc', 'Assistance', 'Gestion', 'Purge de l'historique', 'Système', 'Sécurité', 'Performance', 'API', 'Analyse d'impact', 'GLPI Network', 'Historique', and 'Tous'. The main content area is titled 'API' and contains two sections: 'API' and 'Authentification'. In the 'API' section, there is a text input for 'URL de l'API' with the value 'http://localhost/python-glpi/glpi/apirest.php/'. In the 'Authentification' section, there are two toggle switches: 'Activer l'API Rest' (set to 'Oui') and 'Activer la connexion avec un jeton externe' (set to 'Oui'). A 'Sauvegarder' button is located below these toggles. Below the configuration section, there is a section for 'Ajouter un client de l'API' which includes a search bar, a list of API clients (currently showing 'glpi'), and a 'Rechercher' button.

Presentation 4



The screenshot shows the 'Nouvel élément - Client de l'API' form. It has a light blue header with the title 'Nouvel élément - Client de l'API'. The form contains several fields: 'Nom' (text input), 'Commentaires' (text area), 'Actif' (radio button set to 'Non'), 'Enregistrer les connexions' (dropdown menu set to 'Désactivé'), 'Début de plage d'adresse IPv4' (text input), 'Fin de plage d'adresse IPv4' (text input), 'adresse IPv6' (text input), and 'Jeton d'application (app\_token)' (text input with a 'Regénérer' button). There is also a 'FILTRE L'ACCÈS' section with a note: 'Laisser ces paramètres vides pour désactiver la restriction d'accès à l'API'. At the bottom right, there is a '+ Ajouter' button.

Presentation 5

## 3.1 Explication du code par section

Le code est organisé en plusieurs fonctions pour le rendre modulaire et réutilisable.

### **Explication :**

- `load_dotenv()` charge les variables depuis le fichier `.env`.
- `os.getenv()` permet d'accéder aux valeurs sans les écrire directement dans le code.

```
import os
from dotenv import load_dotenv

# Charger les variables d'environnement depuis le fichier .env
load_dotenv()

# Récupérer les valeurs des variables d'environnement
glpi_url = os.getenv("GLPI_URL")
username = os.getenv("GLPI_USERNAME")
password = os.getenv("GLPI_PASSWORD")
app_token = os.getenv("GLPI_APP_TOKEN")
```

## 3.2. Se connecter à l'API et récupérer un Token de session

### **Explication :**

- Cette fonction envoie une requête POST à GLPI pour obtenir un token de session.
- Le session\_write=true permet de créer des objets (utilisateurs, tickets).
- Si la réponse est 200 OK, on extrait et retourne le session token.

```
def get_session_token():  
    url = f"{glpi_url}/initSession?session_write=true"  
    payload = {"login": username, "password": password}  
    response = requests.post(url, headers=headers, json=payload)  
  
    if response.status_code == 200:  
        session_token = response.json().get('session_token')  
        print(f"Session-Token: {session_token}")  
        return session_token  
    else:  
        print(f"Error: {response.status_code} - {response.text}")  
        return None
```



### 3.3. Créer un utilisateur

#### **Explication :**

- Pourquoi créer dynamiquement un utilisateur ?
  - Permet d'automatiser la gestion des utilisateurs dans GLPI sans passer par l'interface.
- On envoie une requête POST avec les informations du nouvel utilisateur.
- Si l'utilisateur est créé avec succès (201 Created), l'ID est retourné.

```
def create_user(session_token, username, firstname, lastname, email):  
    url = f"{glpi_url}/User"  
    headers.update({"Session-Token": session_token})  
  
    user_data = {  
        "input": {  
            "name": username,  
            "firstname": firstname,  
            "lastname": lastname,  
            "email": email,  
            "authtoken": "unique-auth-token",  
            "status": 1  
        }  
    }  
  
    response = requests.post(url, headers=headers, json=user_data)  
  
    if response.status_code == 201:  
        user_id = response.json().get("id")  
        print(f"User created successfully! ID: {user_id}")  
        return user_id  
    else:  
        print(f"Error creating user: {response.status_code} - {response.text}")  
        return None
```

## 3.4. Créer un ticket

### **Explication :**

- Cette fonction crée un ticket pour un utilisateur donné.
- On associe l'ID de l'utilisateur (requester: user\_id) au ticket.
- urgency: 3 définit une priorité haute.

```
def create_ticket(session_token, user_id, title, description):
    url = f"{glpi_url}/Ticket"
    headers.update({"Session-Token": session_token})

    ticket_data = {
        "input": {
            "name": title,
            "content": description,
            "status": 1,
            "requester": user_id,
            "urgency": 3,
        }
    }


    response = requests.post(url, headers=headers, json=ticket_data)


    if response.status_code == 201:
        ticket_id = response.json().get("id")
        print(f"Ticket created successfully! ID: {ticket_id}")
        return ticket_id
    else:
        print(f"Error creating ticket: {response.status_code} - {response.text}")
        return None
```

## 3.5. Exécuter le script python

### **Explication :**

1. On récupère le token de session
2. On crée un utilisateur (si le token est valide)
3. On crée un ticket pour cet utilisateur

🔥 **Avantages :**  Modularité : Chaque fonctionnalité est une fonction indépendante.

 **Réutilisable :** On peut facilement appeler `create_user()` ou `create_ticket()` séparément.

 **Maintenance facilitée :** Facile à modifier ou améliorer.

```
def main():
    session_token = get_session_token()

    if session_token:
        print("Session token obtained successfully!")

        # Create a user
        username = "daved"
        firstname = "daved"
        lastname = "daveddass"
        email = "daved@example.com"

        user_id = create_user(session_token, username, firstname, lastname, email)

        if user_id:
            # Create a ticket for the user
            ticket_title = "Computer issue"
            ticket_description = "The computer won't start."
            create_ticket(session_token, user_id, ticket_title, ticket_description)
        else:
            print("Failed to obtain session token.")

if __name__ == "__main__":
    main()
```



## 4. Présentation des résultats

### 1. Affichage des résultats dans la console

Après l'exécution du script, les résultats sont affichés directement dans la console. Chaque étape est confirmée par un message pour assurer le bon déroulement des opérations.

### 2. Création de l'utilisateur

Si l'utilisateur est bien créé, un message affiche son ID unique attribué par GLPI. Cela permet de vérifier que l'ajout s'est bien déroulé.

### 3. Création du ticket

Une fois l'utilisateur ajouté, le script génère un ticket d'incident associé à cet utilisateur. Le système retourne un ID de ticket, confirmant son enregistrement dans GLPI.

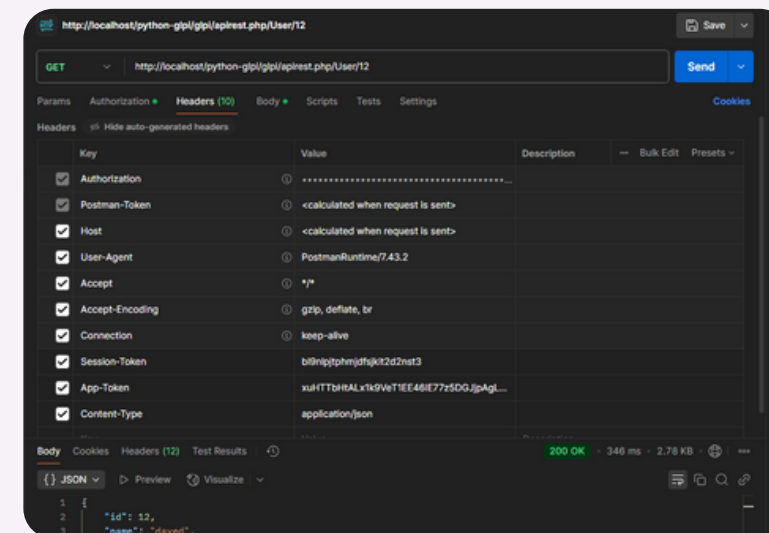
### 4. Validation dans l'interface GLPI

En parallèle, on peut vérifier les nouveaux utilisateurs et tickets directement dans l'interface web de GLPI. Cela permet de confirmer que les données ont bien été enregistrées et sont accessibles via l'outil de gestion

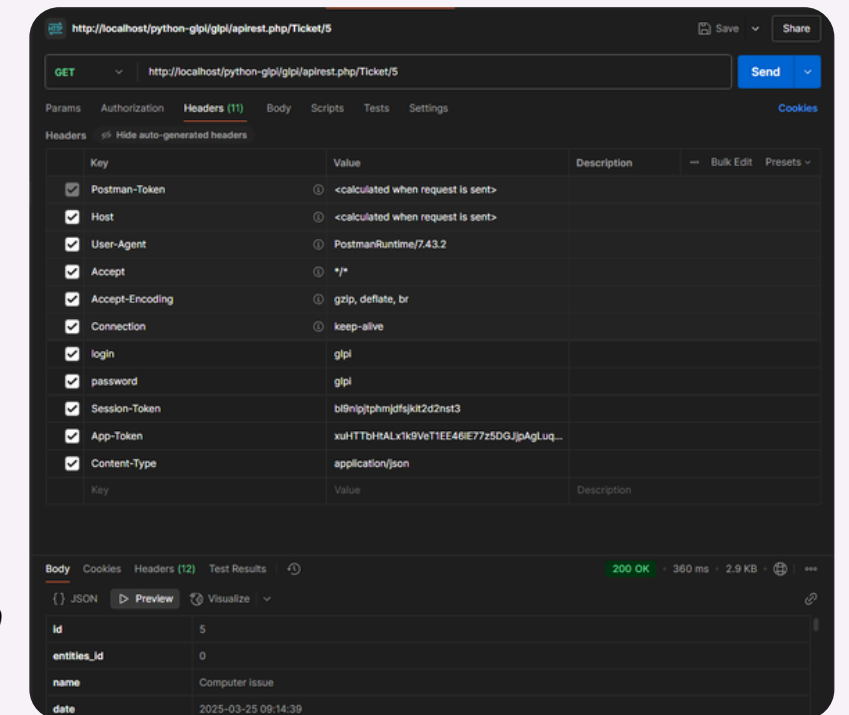
### Exemple de sortie console

Lorsque tout fonctionne correctement, la console affiche des messages comme :

```
rancharaz@rancharaz MINGW64 /c/Users/rancharaz/OneDrive/Desktop/PYTHON TEST/python-test 2
$ py App.py
Session-Token: ugb2os1t5o6k6ooqpbrmr1l9pb
Session token obtained successfully!
User created successfully! ID: 12
Ticket created successfully! ID: 5
(very) (very)
```



Test sur postman pour User response: 200



Test sur postman pour Ticket response: 200

Ce projet montre comment automatiser la création d'utilisateurs et de tickets via l'API GLPI en utilisant Python. L'intégration de bibliothèques comme requests et dotenv a permis de sécuriser les informations sensibles et de rendre le code modulaire et maintenable.

N'hésitez pas si vous avez des questions.  
Merci pour votre attention !