**Une image contenant Police, Graphique, logo, graphisme

Description générée automatiquement**

**Architecture Logicielle (A2025)**

**Rapport de Laboratoire**

|  |  |
| --- | --- |
| **Nᵒ du laboratoire** | Laboratoire 1 |
| **Étudiant(s)** | Makesa, Mvuemba Gildor |
| **Cours** | LOG430 |
| **Session** | Aut. 2025 |
| **Groupe** | LOG430-01-02 |
| **Professeur** | Fabio Petrillo |
| **Chargé de Laboratoire** | Gabriel C. Ullmann |
| **Date** | 18 sept. 25 |

# 💡 Question 1

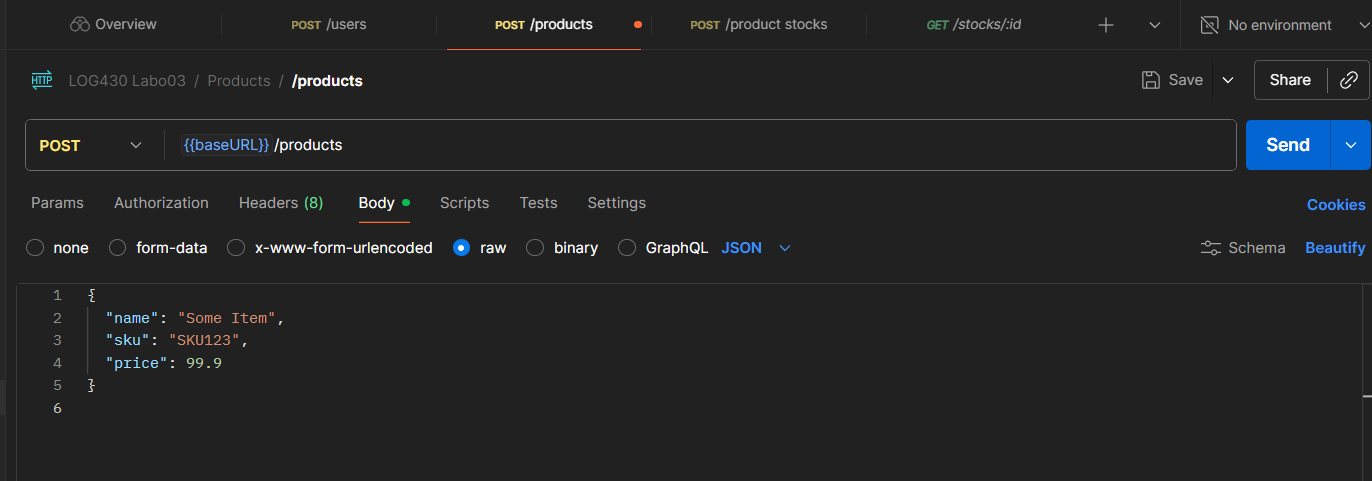
Quel nombre d'unités de stock pour votre article avez-vous obtenu à la fin du test ? Et pour l'article avec id=2 ?

Veuillez inclure la sortie de votre Postman pour illustrer votre réponse

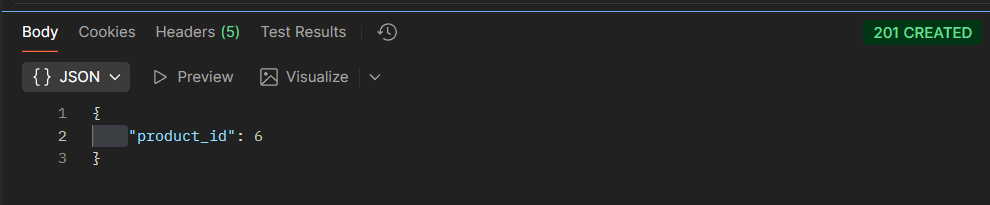
Dans test\_stock\_flow, on :

**A. Créer un produit dans Postman**

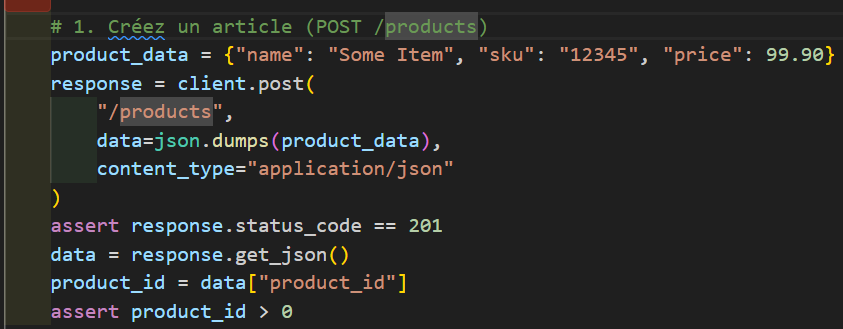
1. Sélectionne la requête **POST {{baseURL}}/products**.
2. Onglet **Headers** → ajoute Content-Type: application/json.
3. Onglet **Body** → **raw** + **JSON** et mets un **produit complet** :



1. Clique **Send**.
2. Réponse attendue (201 Created) :

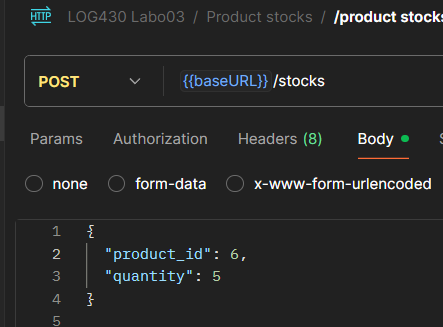


**Tests (onglet “Tests”)** — pour stocker l’ID produit :

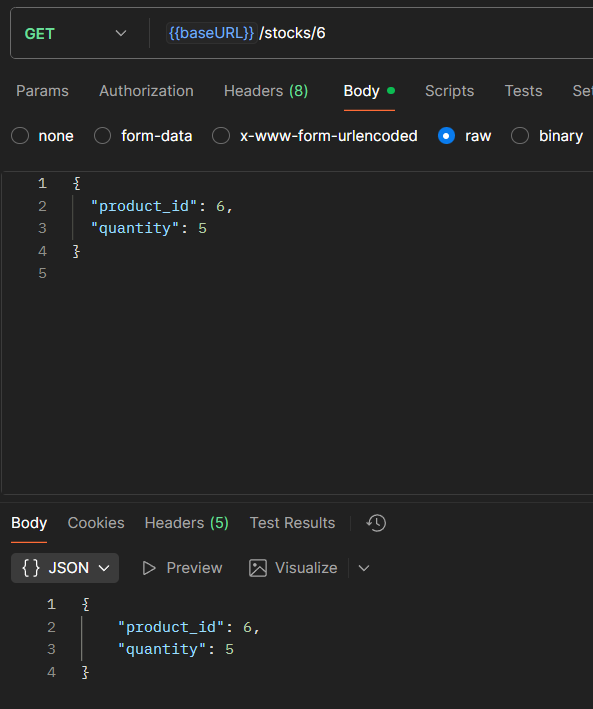


**B. Ajouter du stock pour ce produit**

1. Ensuite, pour augmenter le stock, utilise **POST {{baseURL}}/stocks** avec :



1. Puis vérifie : GET {{baseURL}}/stocks/6

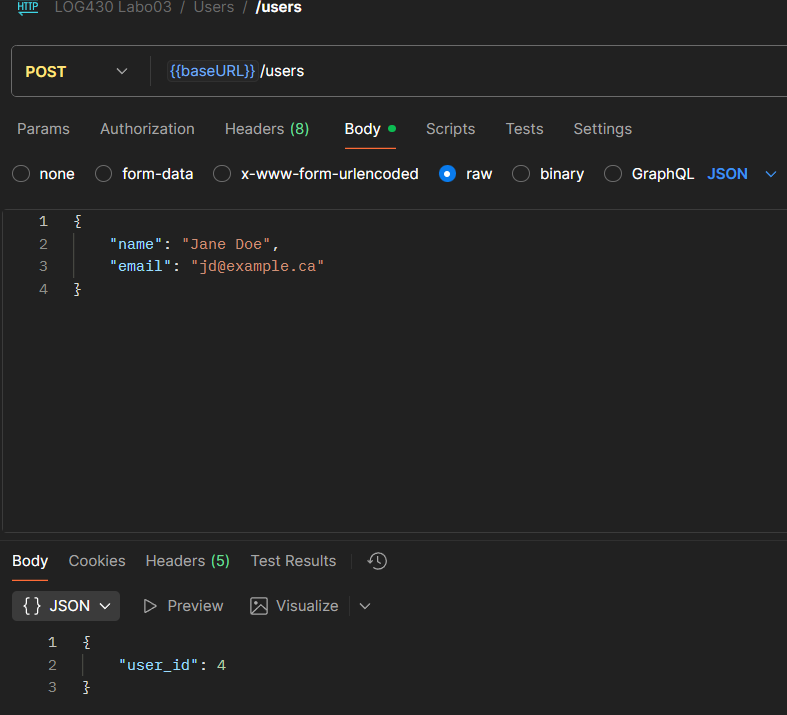


**Tests (onglet “Tests”)** — pour ajouter 5 unité de produit



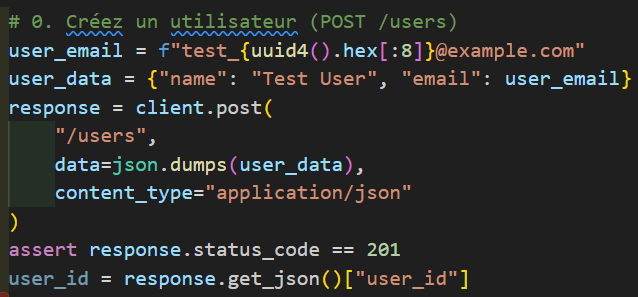
**C.** **Créez une commande de 2 unités (POST /orders)**

1. Tout d’abord, créons un utilisateur en sélectionnant la requête **POST {{baseURL}}/users**.

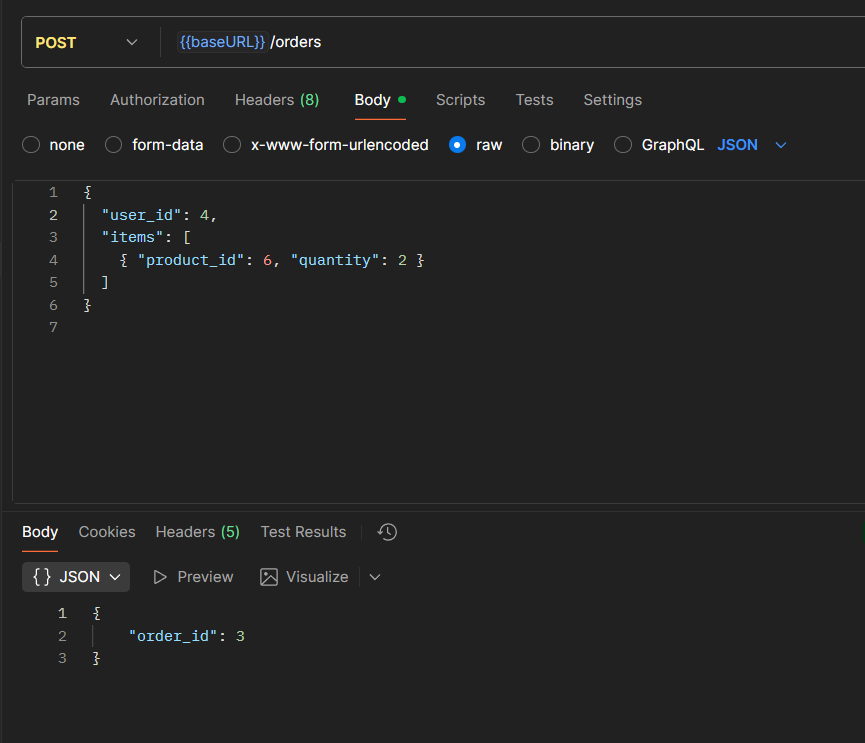


\*Remarque : *Prendre note du user\_id :4*

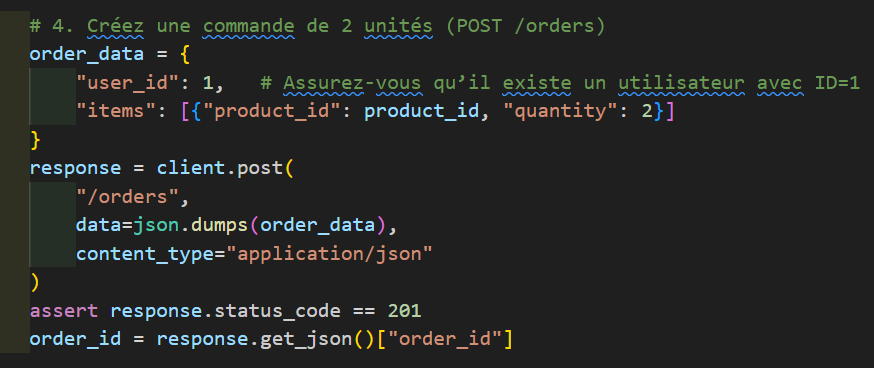
**Tests (onglet “Tests”)** — pour ajouter un user



1. Par la suite, passons une commande de 2 unités dans cet utilisateur :

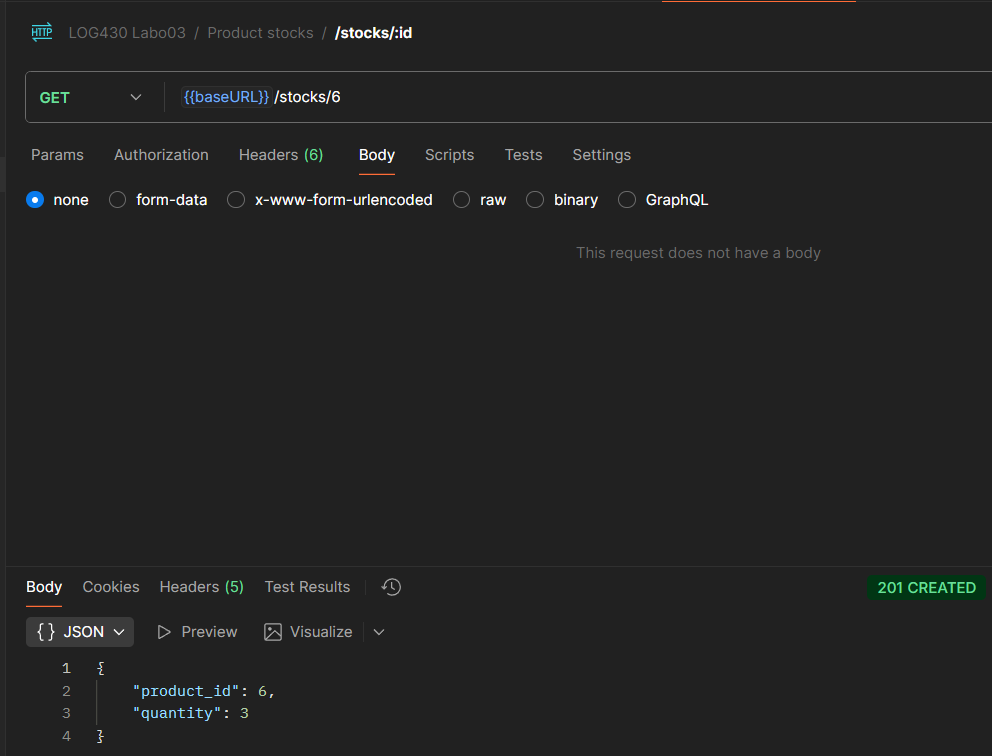


**Tests (onglet “Tests”)** — pour créer une commande de 2 unités (POST /orders)

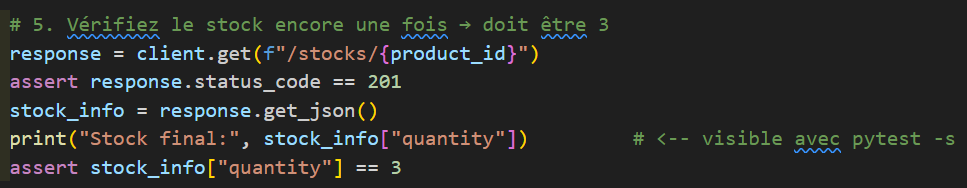


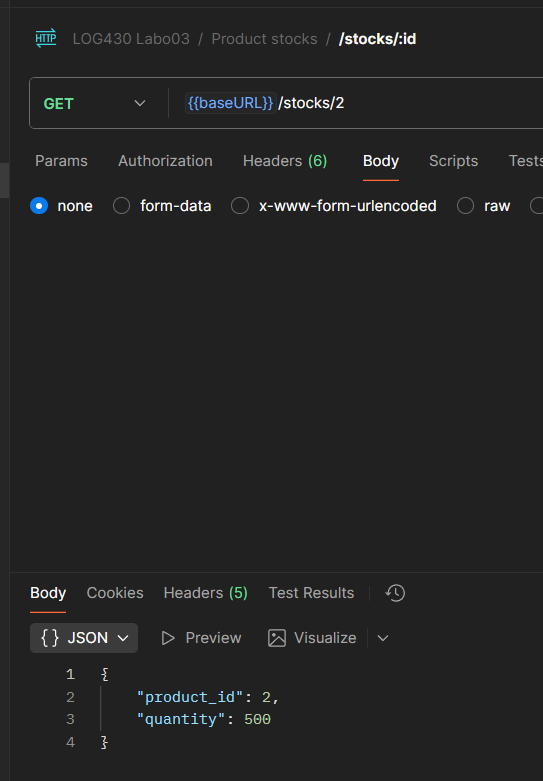
**D.** **Vérifiez le stock encore une fois → doit être 3**

1. Relire le stock **{{baseURL}}/stocks/{{product\_id}}**



**Tests (onglet “Tests”)** — pour Vérifiez le stock encore une fois



Et pour l’article avec le ID = 2 toujours 500 :  


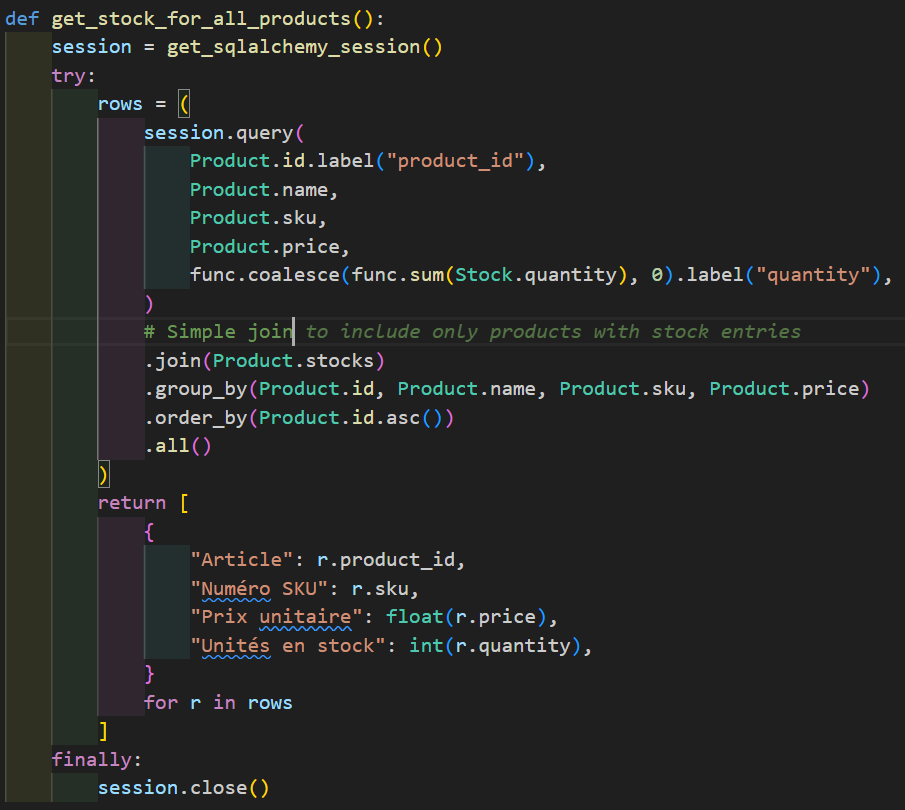
# 💡 Question 2

Décrivez l'utilisation de la méthode join dans ce cas. Utilisez les méthodes telles que décrites à Simple Relationship Joins et Joins to a Target with an ON Clause dans la documentation SQLAlchemy pour ajouter les colonnes demandées dans cette activité. Veuillez inclure le code pour illustrer votre réponse.

L'utilisation de la méthode **join** dans le contexte de la fonction get\_stock\_for\_all\_products() se référe aux mécanismes de jointures de SQLAlchemy. L'objectif est de combiner les données des tables **Product** et **Stock** pour obtenir les informations de base de chaque produit ainsi que la quantité en stock correspondante.

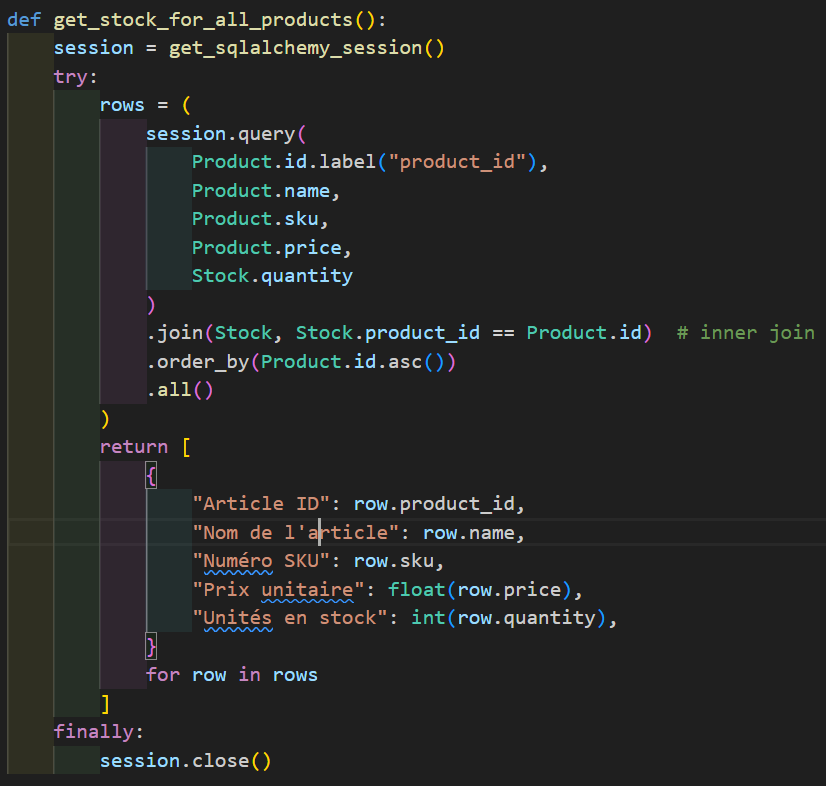
**1. Rôle de la méthode join**

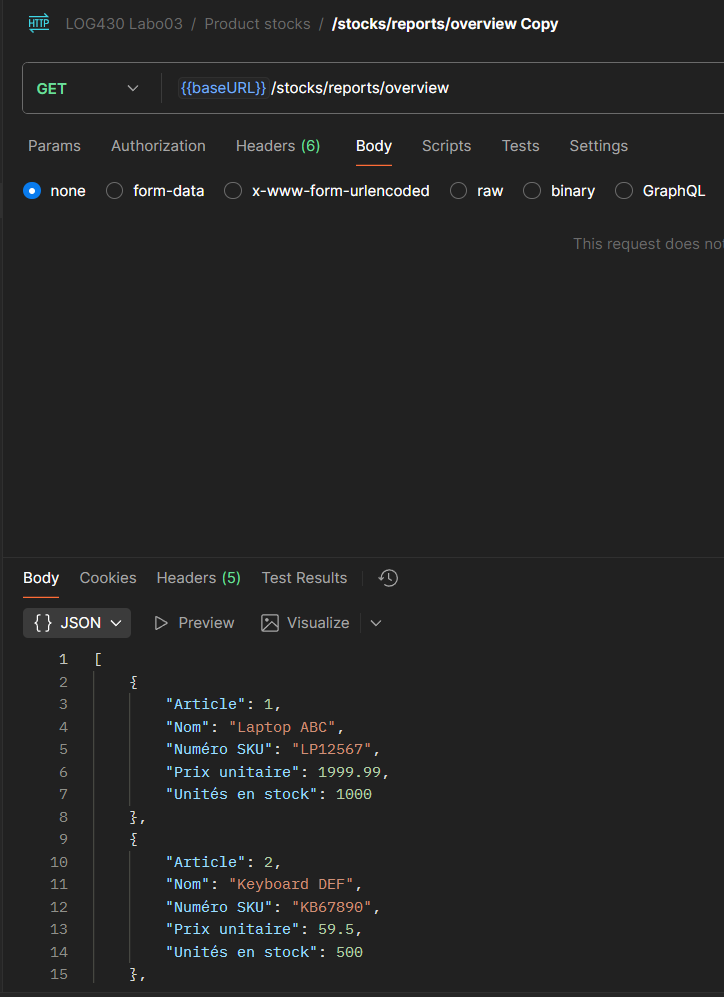
Dans ce cas, la méthode join est utilisée pour effectuer une **jointure interne explicite** entre la table principale **Product** (qui est la source de la requête via session.query(Product...)) et la table **Stock**. Cette opération permet de lier les lignes des deux tables lorsque les valeurs d'une colonne spécifiée correspondent.



**2. Rôle de la méthode join**

L'utilisation de la méthode join pour spécifier la table cible et la condition est une application de la technique **"Joins to a Target with an ON Clause"** de la documentation SQLAlchemy. Elle définit la condition selon laquelle la jointure doit être effectuée. Elle stipule que seules les lignes où la colonne product\_id de la table Stock correspond à la colonne id de la table Product sont combinées et incluses dans le résultat.





# 💡 Question 3

Quels résultats avez-vous obtenus en utilisant l’endpoint POST /stocks/graphql avec la requête suggérée? Veuillez joindre la sortie de votre requête dans Postman afin d’illustrer votre réponse.

En utilisant l'endpoint POST /stocks/graphql avec la requête GraphQL suggérée dans Postman, nous avons obtenu les informations spécifiques du produit ayant l'**ID 1**, en incluant les données de base et la quantité en stock.

La requête, qui ciblait la méthode product(id: 1), a explicitement demandé les champs id, name, sku, et quantity.

**Résultats obtenus**

Les résultats retournés dans le corps de la réponse JSON sont les suivants, confirmant que l'endpoint a réussi à agréger les données du produit et de l'inventaire :

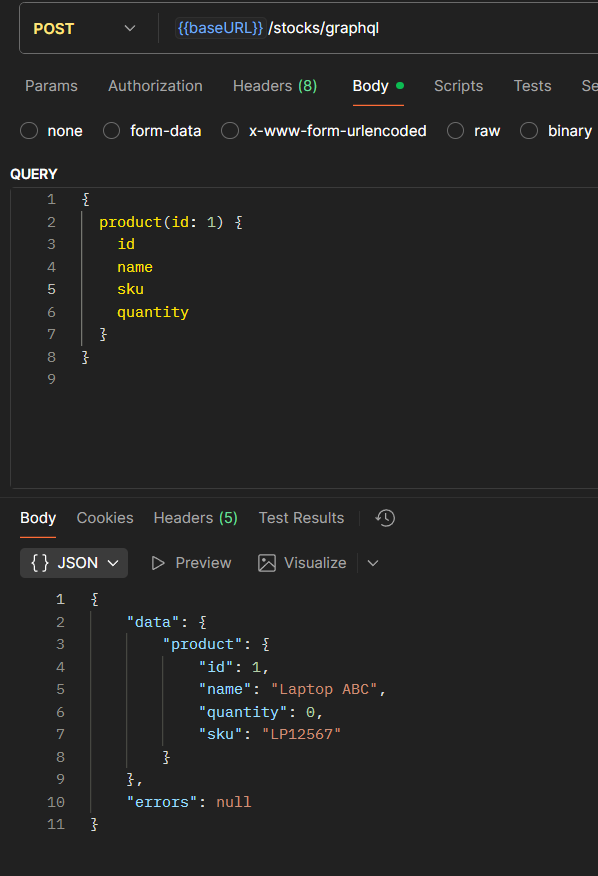
|  |  |  |
| --- | --- | --- |
| **Champs** | **Valeur obtenue** | **Interprétation** |
| ID | 1 | L’identifiant du produit |
| Name | "Laptop ABC" | Le nom du produit. |
| sku | "LP12567" | La référence unique du produit (SKU) |
| Quantity | 0 | La quantité en stock de ce produit (donnée d'inventaire). |

L'objet JSON retourné ne contient **aucune erreur** ("errors": null), ce qui signifie que la requête a été exécutée avec succès et que les champs demandés ont été récupérés.

**llustration de la réponse**

L'image jointe, représentant l'interface Postman, sert d'illustration directe à cette réponse :

1. **Requête** : Le panneau *Query* montre la requête GraphQL envoyée
2. **Réponse** : Le panneau *Body (JSON)* montre le résultat sur mesure



# 💡 Question 4

Quelles lignes avez-vous changé dans update\_stock\_redis? Veuillez joindre du code afin d’illustrer votre réponse.

Les modifications apportées à la fonction update\_stock\_redis visaient à garantir la **cohérence** des données dans **Redis** en gérant deux aspects principaux :

1. La mise à jour de la **quantité** en stock avec un **garde-fou** pour éviter les valeurs négatives.
2. L'enrichissement des **métadonnées** du produit (nom, SKU, prix) dans Redis, en ne les récupérant de MySQL que si elles sont manquantes.

Ainsi,

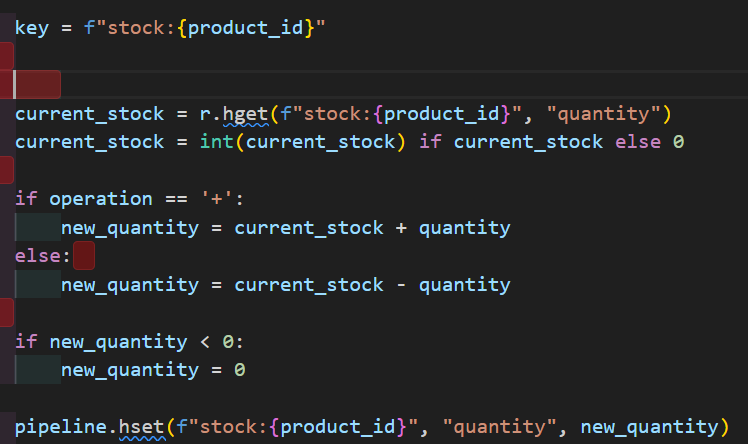
**1. Initialisation de la session SQLAlchemy**

Une session de base de données est ouverte **avant** la boucle de traitement des articles de commande, afin de ne l'ouvrir qu'une seule fois, pour une performance optimale.



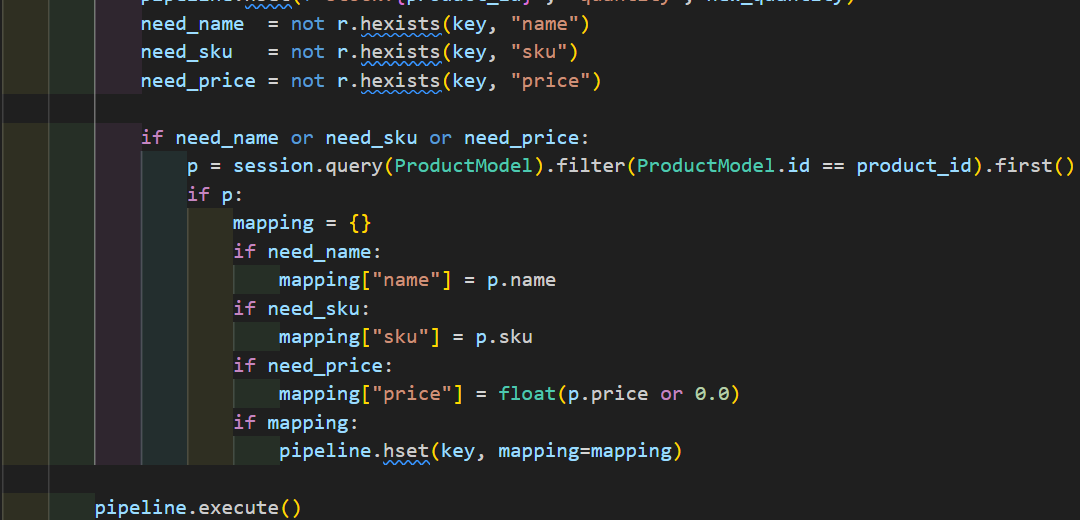
**2. Gestion de la quantité et du garde-fou (Clamp à Zéro)**

Ces lignes définissent la clé Redis et introduisent la logique qui assure que la quantité en stock ne descend jamais en dessous de zéro.



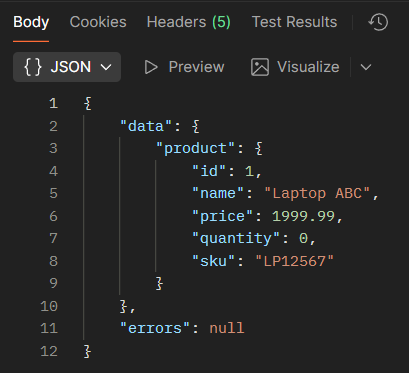
**3. Complétion des métadonnées manquantes (Ajout complet)**

Ce bloc de code est l'ajout le plus significatif. Il vérifie l'existence des champs de métadonnées dans Redis et effectue une requête ciblée vers MySQL seulement si des informations sont manquantes.



# 💡 Question 5

Quels résultats avez-vous obtenus en utilisant l’endpoint POST /stocks/graphql avec les améliorations ? Veuillez joindre la sortie de votre requête dans Postman afin d’illustrer votre réponse.



# 💡 Question 6