**PROJET DE DEVELOPPEMENT DE BASE DE COMPOSANT ET SERVICES WEB**

**IBAM 2024**

**PROFESSEUR : M. KY F Omar**

**GROUPE N°9**

* TRAORE Bafima Téli Martin Daouda
* KONATE Salim Adam
* KONATE Karim
* BAZIE Alex Damien
* NIKIEMA Guy Arthur

Table des matières

[I. Étape 1 : Configuration de l'Environnement 4](#_Toc175755061)

[A. Création d’un dépôt Git public 4](#_Toc175755062)

[B. Présentation, Installation et Configuration d'Apache NiFi pour ETL 5](#_Toc175755063)

[II. Étape 2 : Développement de l'Application Flask 17](#_Toc175755064)

[A. Création la structure du projet 17](#_Toc175755065)

[B. Connexion Flask à la base de données 17](#_Toc175755066)

[C. Création des Modèles et Routes 18](#_Toc175755067)

[D. Démarrez votre application 20](#_Toc175755068)

[III. Étape 3 : Documentation et Test de l’API avec Postman 21](#_Toc175755069)

[A. Documenter le Processus 21](#_Toc175755070)

[B. Création des tests Postman 22](#_Toc175755071)

[WEBOGRAPHIE 30](#_Toc175755072)

Listes de figures

[Figure 1: Fenêtre de création d’un dépôt Git/ Source Git public 4](#_Toc175754837)

[Figure 2: Fenêtre d'ajout des membres/collaborateurs / Source Git public 4](#_Toc175754838)

[Figure 3:Figure 2: Fenêtre du lien d'accès du dépôt Git / Source Git public 5](#_Toc175754839)

[Figure 4: Lien de téléchargement d’Apache NiFi/ Source : Apache NIFI 6](#_Toc175754840)

[Figure 5 Démarrage de Apache NiFi/ Source : CMD Windows 6](#_Toc175754841)

[Figure 6: Interface de connexion d’Apache NiFi/ Source : Apache NiFi 7](#_Toc175754842)

[Figure 7: Fenêtre identification et d'authentification d’Apache NiFi/ Source : Apache NiFi 7](#_Toc175754843)

[Figure 8: Fenêtre principale d’Apache NiFi/ Source : Apache NiFi 8](#_Toc175754844)

[Figure 9: Fenêtre Ajout d’un processeur d’Apache NiFi/ Source : Apache NiFi 9](#_Toc175754845)

[Figure 10: Fenêtre de configuration de ma BD à Apache NIFI/ Source : Apache NiFi 9](#_Toc175754846)

[Figure 11: Fenêtre de configuration du processus ExecuteSQL\_Reservat/ Source : Apache NiFi 10](#_Toc175754847)

[Figure 12: Fenêtre de configuration du processus ExecuteSQL\_chambre/ Source : Apache NiFi 10](#_Toc175754848)

[Figure 13: Fenêtre de configuration du processus ConvertRecord\_Reserv/ Source : Apache NiFi 11](#_Toc175754849)

[Figure 14: Fenêtre de configuration du processus ConvertRecord\_chamb/ Source : Apache NiFi 12](#_Toc175754850)

[Figure 15: Fenêtre de configuration du processus PutFile\_chambres/ Source : Apache NiFi 13](#_Toc175754851)

[Figure 16: Fenêtre de configuration du processus PutFile\_Reservations/ Source : Apache NiFi 14](#_Toc175754852)

[Figure 17: Fenêtre de configuration du processus Invoke\_chambres/ Source : Apache NiFi 14](#_Toc175754853)

[Figure 18: : Fenêtre de configuration du proc LogAttribute\_chambres/ Source : Apache NiFi 15](#_Toc175754854)

[Figure 19:: Fenêtre de configura globale des pr utilisés dans notre projet/ Source : Apache NiFi 16](#_Toc175754855)

[Figure 20: Fenêtre affichant la structure du notre projet/ Source : VS code 17](#_Toc175754856)

[Figure 21: Fenêtre de configuration de ma BD à l’API RES full / Source : VS code 17](#_Toc175754857)

[Figure 22: Fenêtre de indiquant mes classe/ Source : VS code 18](#_Toc175754858)

[Figure 23: Figure 22: Fenêtre de indiquant le démarrage de mon API RESTfull/ Source : VS code 20](#_Toc175754859)

[Figure 24: Fenêtre de Test GET /réservations 23](#_Toc175754860)

[Figure 25: Fenêtre de Test GET /chambres\_types 24](#_Toc175754861)

[Figure 26: Fenêtre de Test GET /chambres 25](#_Toc175754862)

[Figure 27: Fenêtre de Test POST /reservations 26](#_Toc175754863)

[Figure 28:Fenêtre de Test POST /chambres\_types 27](#_Toc175754864)

[Figure 29: Fenêtre de Test POST /chambres 28](#_Toc175754865)

[Figure 30: Figure 27: Fenêtre de Test GET /chambres\_stats 29](#_Toc175754866)

Listes des Tableaux

[Tableau 1: Quelques commandes utiles pour Apache NIFI 8](#_Toc175722163)

# **Étape 1 : Configuration de l'Environnement**

## **Création d’un dépôt Git public**

* **Création d’un dépôt Git public sur une plateforme GitHub.**

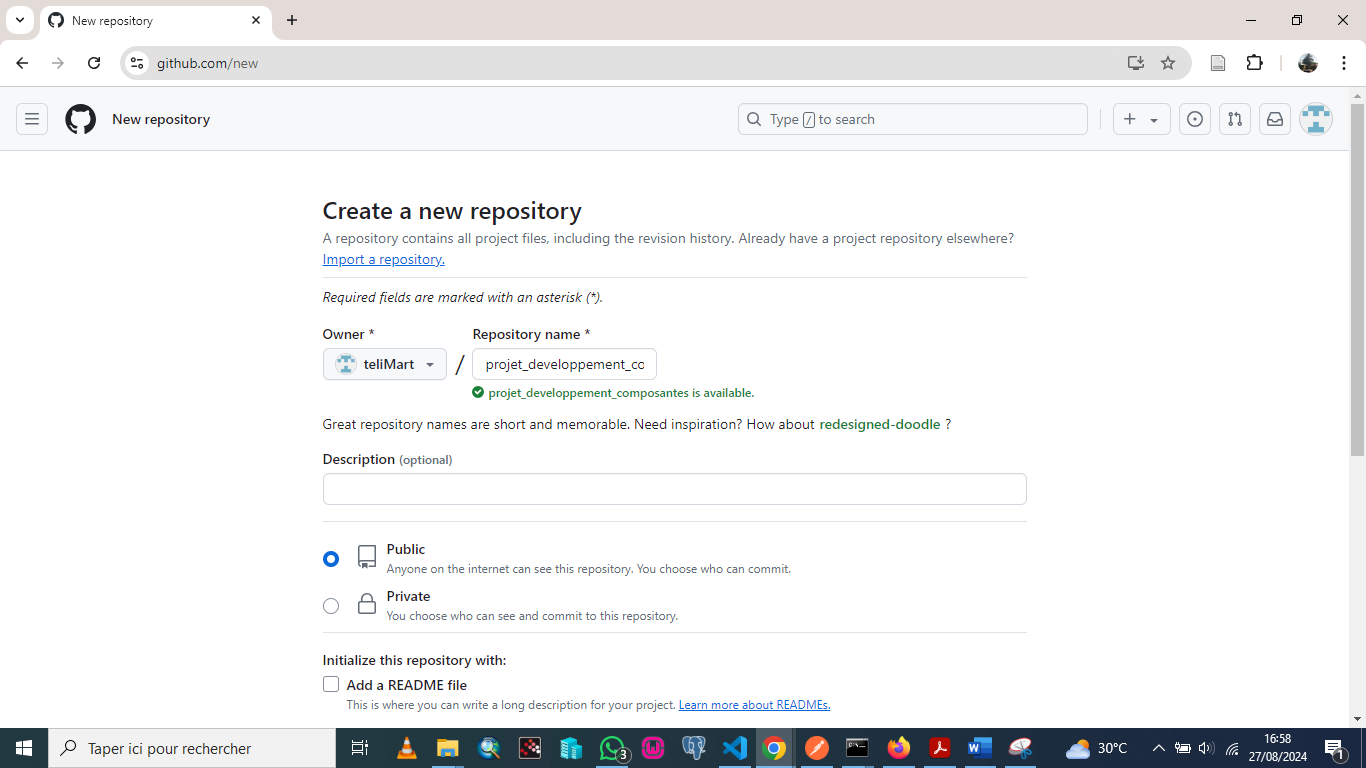


Figure 1: Fenêtre de création d’un dépôt Git/ Source Git public

* **Ajouter les membres de votre groupe comme collaborateurs**

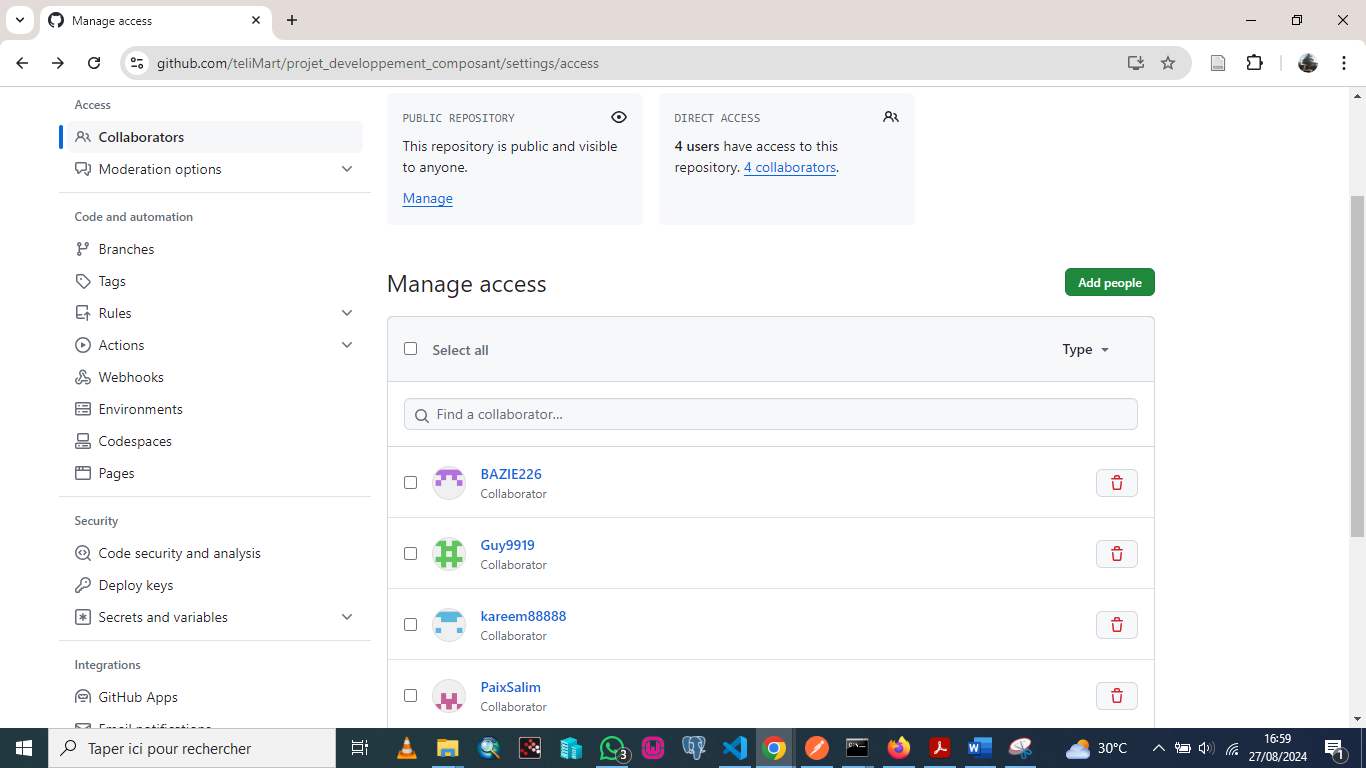


Figure 2: Fenêtre d'ajout des membres/collaborateurs / Source Git public

* **Fourniture du lien du dépôt pour évaluation**

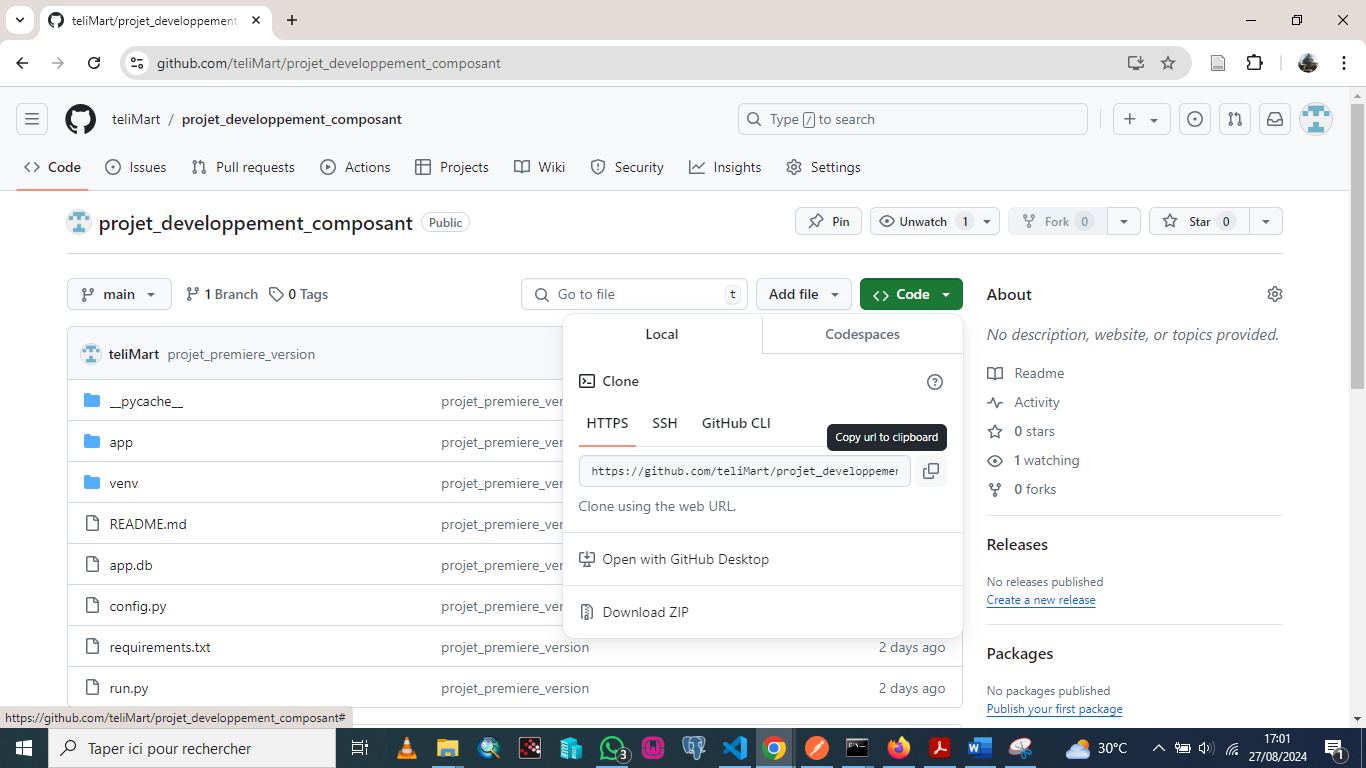


Figure 3:Figure 2 : Fenêtre du lien d'accès du dépôt Git / Source Git public

## **Présentation, Installation et Configuration d'Apache NiFi pour ETL**

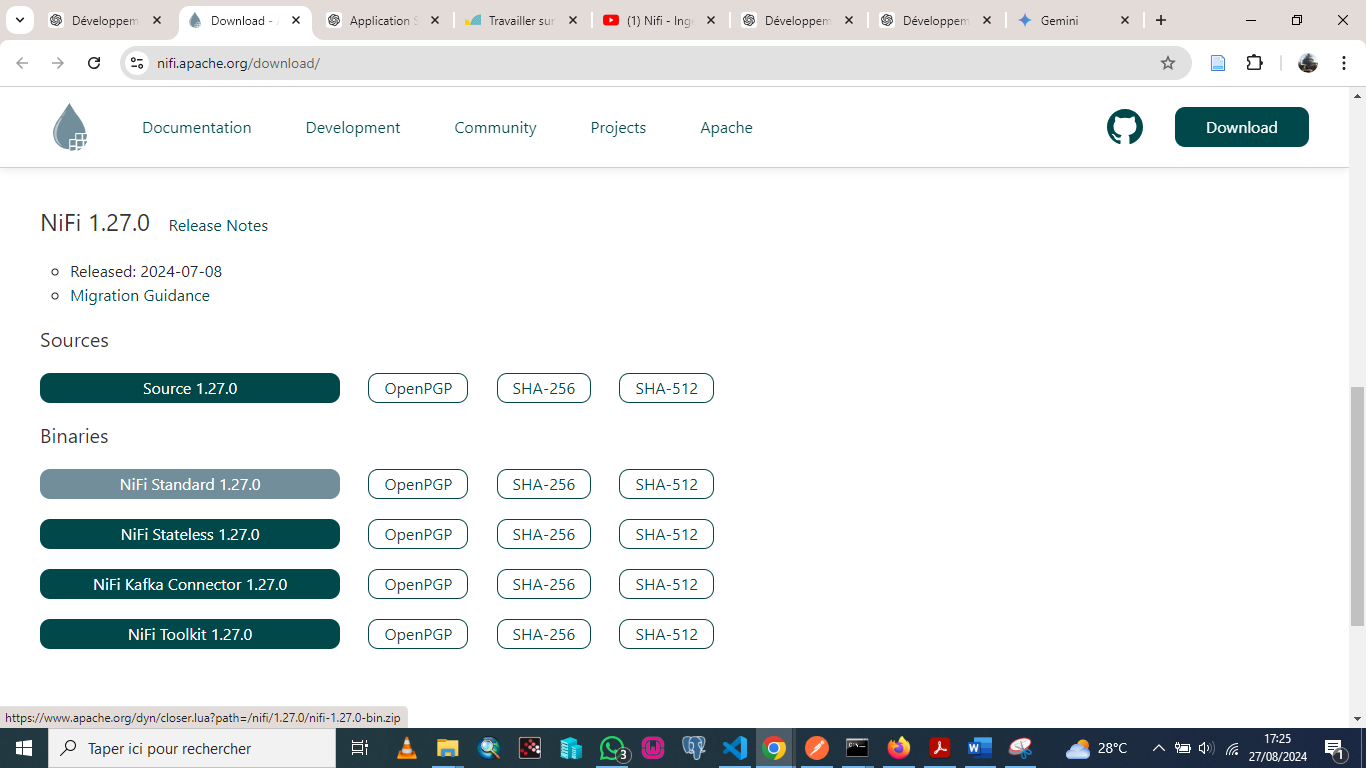
### **Présentation de Apache NiFi**

Apache NiFi est un outil puissant utilisé pour automatiser le flux de données c’est un logiciel open source de gestion de données ETL (Extract Transform and Load), qui vise à collecter, organiser, contrôler, sécuriser et analyser des flux de données entre différents systèmes sources, de façon automatisée et en temps réel. La plateforme dispose d’un processus de flux de données personnalisable et modifiable dans le cloud à partir d’une interface web, ou dans une architecture informatique distribuée.

### **Installation de Apache NiFi**

* **Téléchargez la dernière version d'Apache NiFi depuis le site officiel.**

Pour pouvoir utiliser et tester Apache NiFi, il suffit de **télécharger le fichier source ou binaire**sur le site Apache ou hortonworks (<https://nifi.apache.org/download.html>). Choisissez la version souhaitée (la dernière version stable est recommandée) et téléchargez le fichier compressé correspondant à votre système d'exploitation (par exemple, .zip pour Windows). Dans notre cas il s’agit de **“ nifi-1.27.0 ”.**



**1**

**3**

**2**

Figure 4: Lien de téléchargement d’Apache NiFi/ Source : Apache NIFI

**Lançons ensuite le fichier** « ***run.cmd******start »*** qui se trouve dans le répertoire C:\Users\HP\Desktop\DevComposante\nifi-1.27.0-bin\nifi-1.27.0\bin.

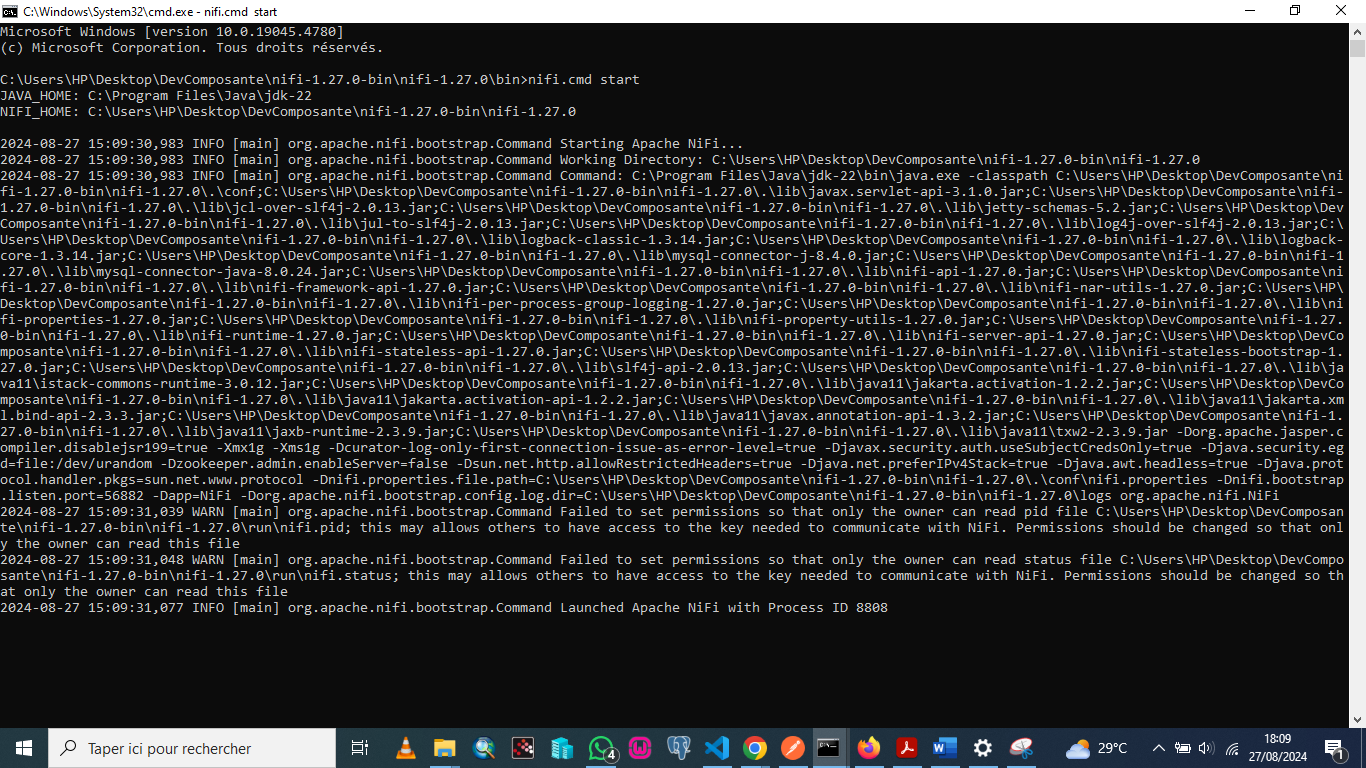


Figure 5 Démarrage de Apache NiFi/ Source : CMD Windows

D'après le journal d'exécution dans la capture d'écran, Apache NiFi a été démarré avec succès. La ligne suivante confirme que NiFi a été lancé :

Cela signifie que NiFi est en cours d'exécution avec l'ID de processus 19996.

**Accédez à l'interface utilisateur de NiFi** en utilisant votre navigateur web à l'adresse *https://localhost:8443/nifi*

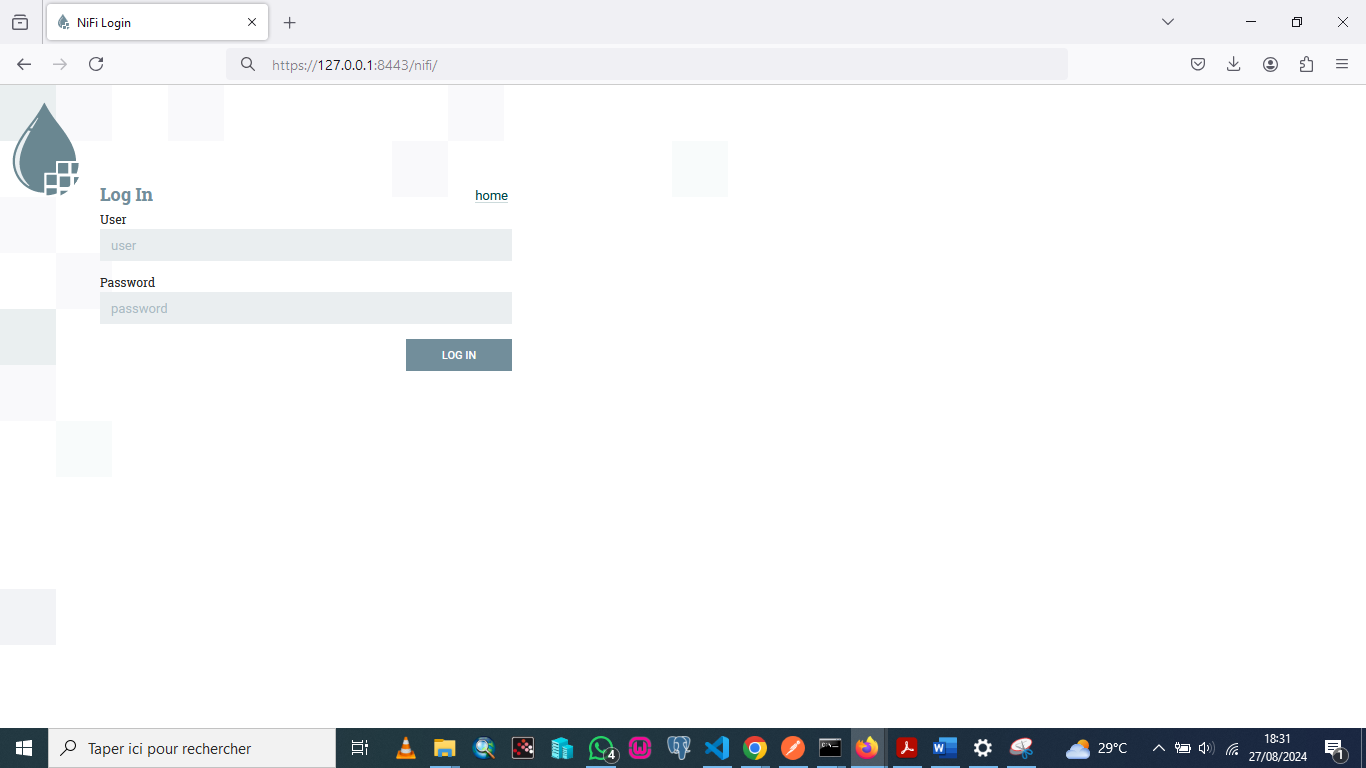


Figure 6: Interface de connexion d’Apache NiFi/ Source : Apache NiFi

A cet emplacement *C:\Users\HP\Desktop\DevComposante\nifi-1.27.0-bin\nifi-1.27.0\logs* copier et coller les accès d’authentification après avoir effectué une recherche sure « Username » du fichier *nifi-app*.

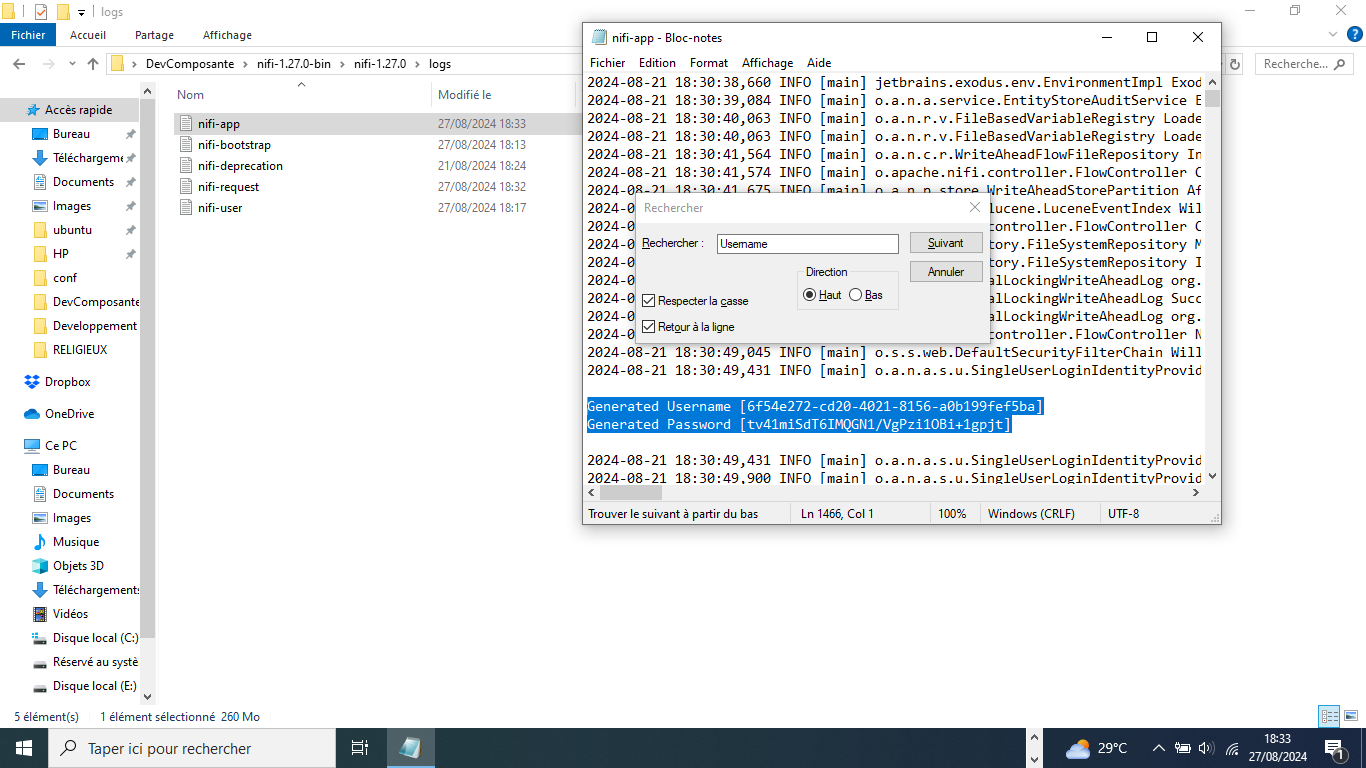


Figure 7: Fenêtre identification et d'authentification d’Apache NiFi/ Source : Apache NiFi

Nous obtenons alors cette fenêtre qui est la fenêtre principale de Apache nifi :

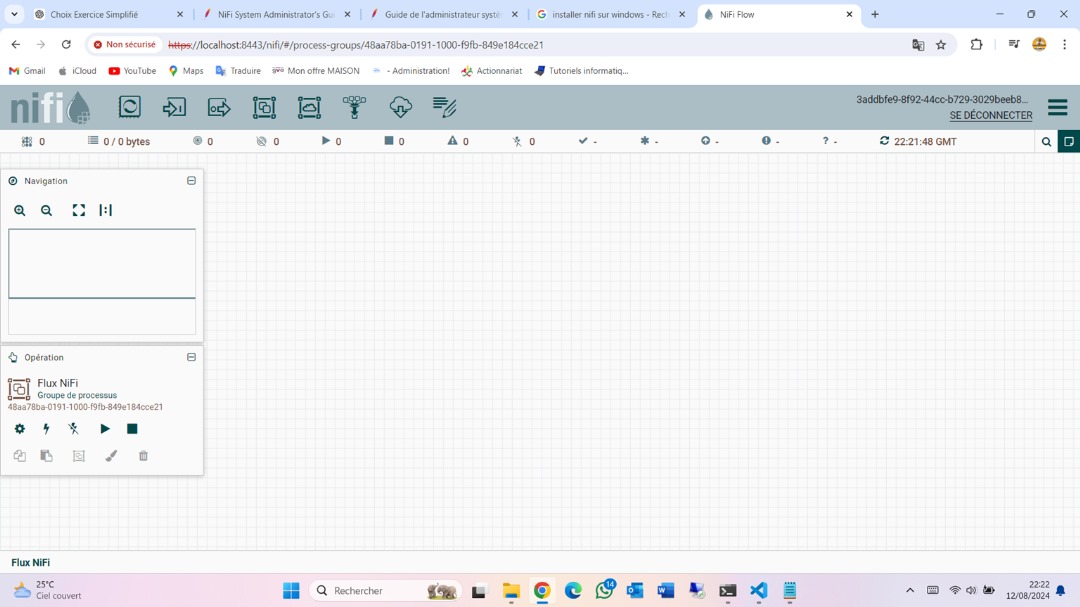


Figure 8: Fenêtre principale d’Apache NiFi/ Source : Apache NiFi

Apache NiFi est désormais **installé et lancé** sur notre poste de travail.

* **Commandes Utiles pour Apache NiFi**

Voici quelques commandes que vous pouvez utiliser dans votre terminal :

Tableau 1: Quelques commandes utiles pour Apache NIFI

|  |  |
| --- | --- |
| COMMANDES | FONCTIONNALITÉS |
| nifi.cmd start | Démarre le service NiFi. |
| nifi.cmd stop | Arrête le service NiFi. |
| nifi.cmd restart | Redémarre le service NiFi. |
| nifi.cmd status | Affiche l'état actuel de NiFi (en cours d'exécution, arrêté, etc.). |
| nifi.cmd diagnostics | Génère un rapport de diagnostic détaillé sur l'état de NiFi. |

### **Configuration Apache NiFi pour extraire les données**

* **Recherche / Ajout d’un processeur**

NiFi possède **plus de 200 connecteurs ou processeurs** qui permettent de **collecter en temps réel des données** issues de plus de 80 sources : bases de données, messages, fichiers, flux Twitter, etc.

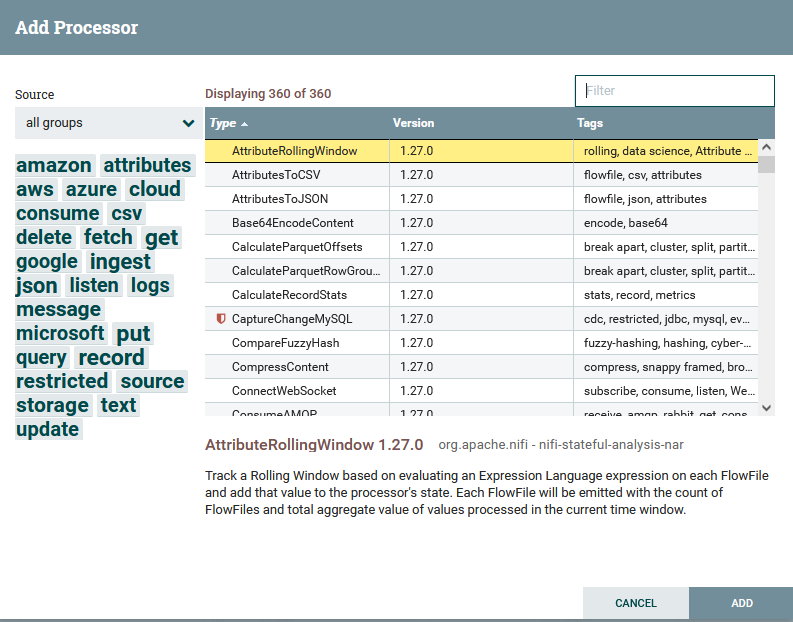


Figure 9: Fenêtre Ajout d’un processeur d’Apache NiFi/ Source : Apache NiFi

* **Configuration de la Connexion à la Base de Données**

Avant tout il faut d’abord créer un Data Source dans NiFi pour nous connecter à notre base de données MySQL « db\_hotel ».

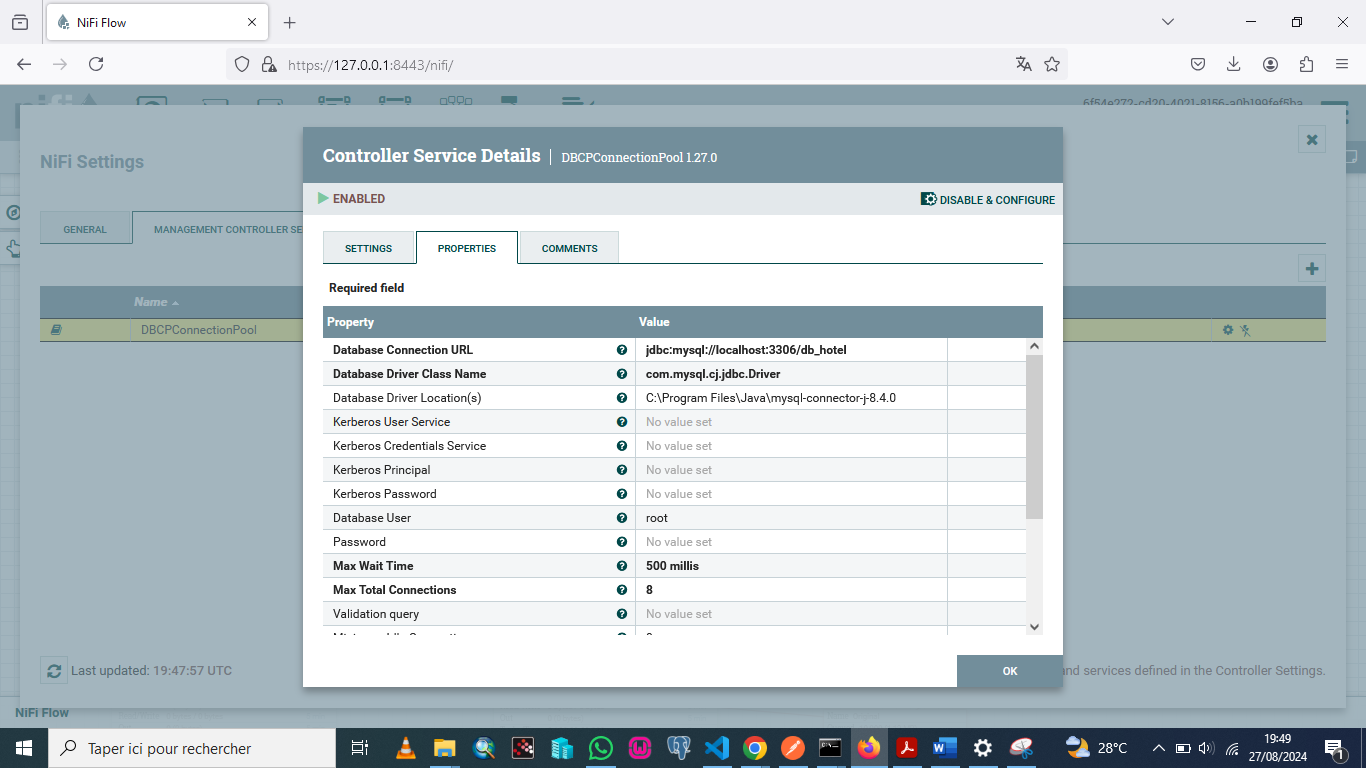


Figure 10: Fenêtre de configuration de ma BD à Apache NIFI/ Source : Apache NiFi

* **Utilisation du Processeur ExecuteSQL**

Ces processeurs exécutent des requêtes SQL sur la base de données pour extraire des informations sur les chambres (chambre) et les réservations (Reservation). Le résultat de ces requêtes est envoyé aux processeurs suivants via les connexions success.

* Configurez le processeur ExecuteSQL pour extraire les données de réservations d'hôtel.

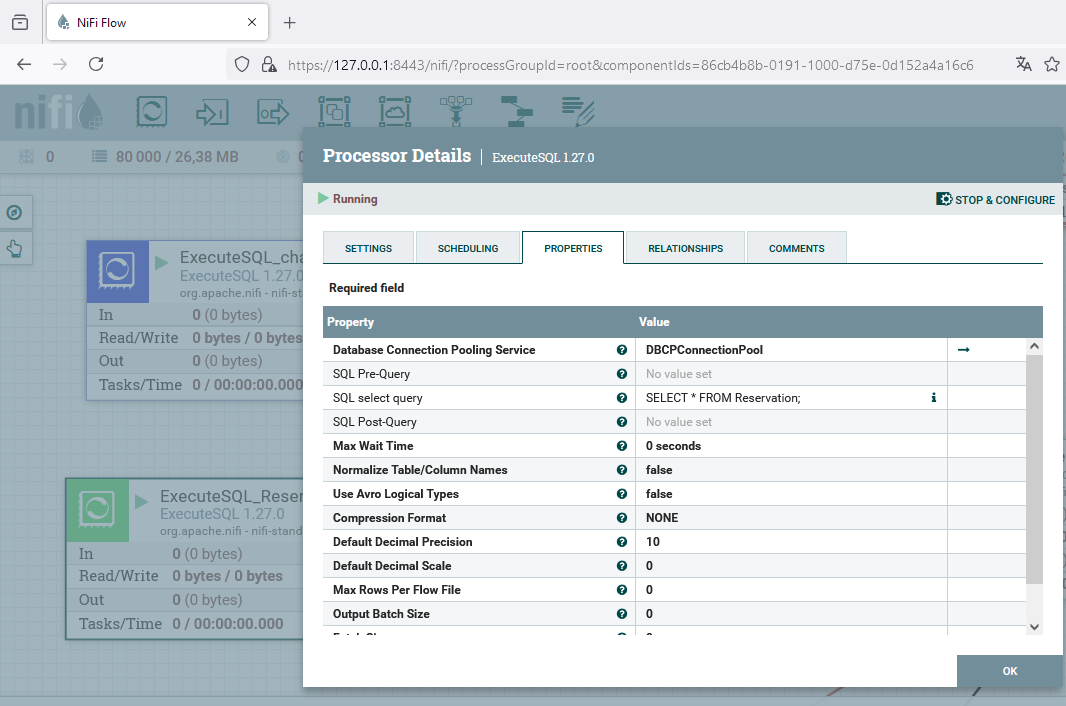


Figure 11: Fenêtre de configuration du processus ExecuteSQL\_Reservations/ Source : Apache NiFi

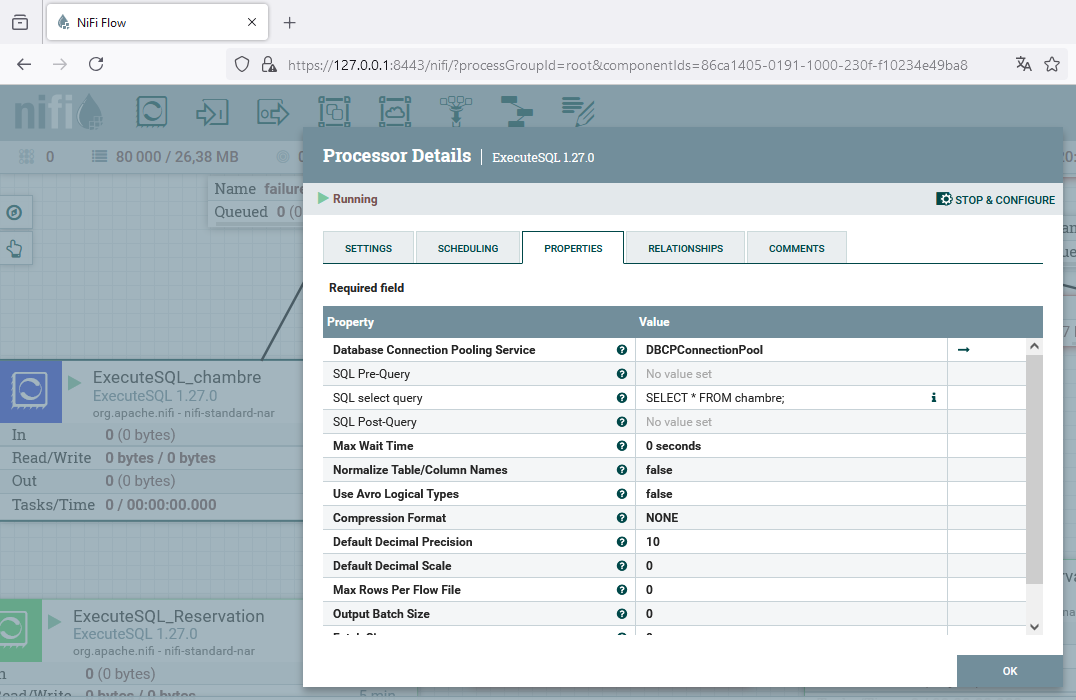
* Configurez le processeur ExecuteSQL pour extraire les données de chambres d'hôtel.

Figure 12: Fenêtre de configuration du processus ExecuteSQL\_chambre/ Source : Apache NiFi

### **Transformation des données**

* **Utilisation de ConvertRecord :**

Configurez ConvertRecord pour transformer les données extraites en un format CSV ou Avro compatible avec votre application Flask. Ces processeurs convertissent les résultats SQL extraits par ExecuteSQL en un format spécifique (comme JSON, Avro, etc.). Cela facilite le traitement ultérieur des données. Les connexions success envoient les données converties vers les processeurs suivants pour un traitement supplémentaire.

* Configurez le processeur ConvertRecord pour extraire les données de réservations d'hôtel.

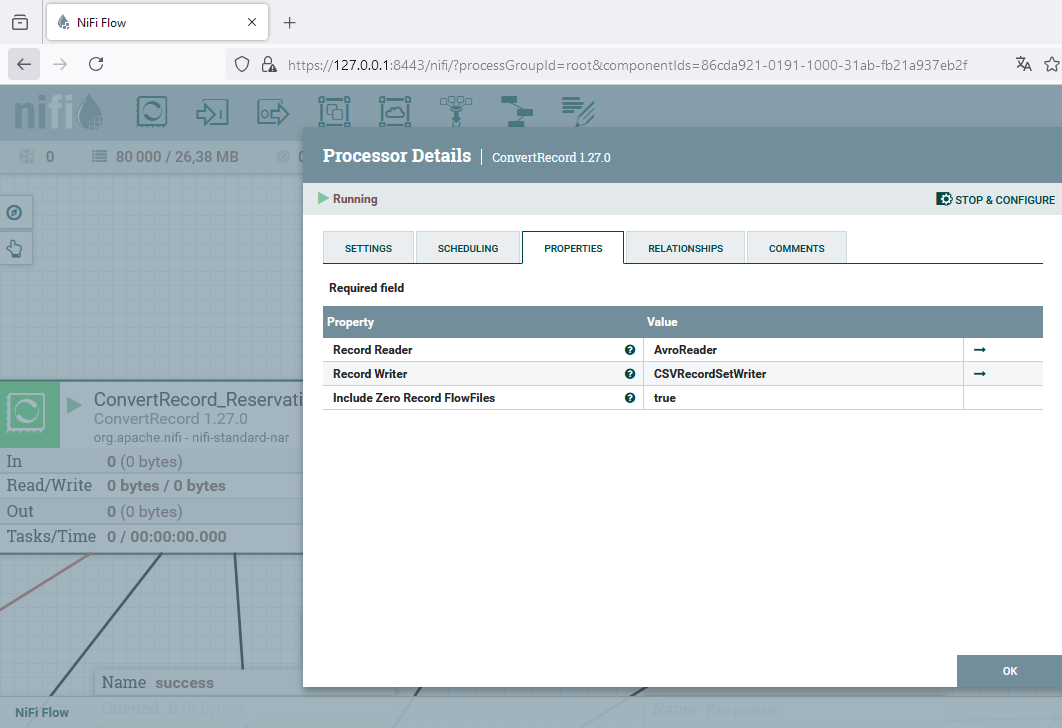


Figure 13: Fenêtre de configuration du processus ConvertRecord\_Reservations/ Source : Apache NiFi

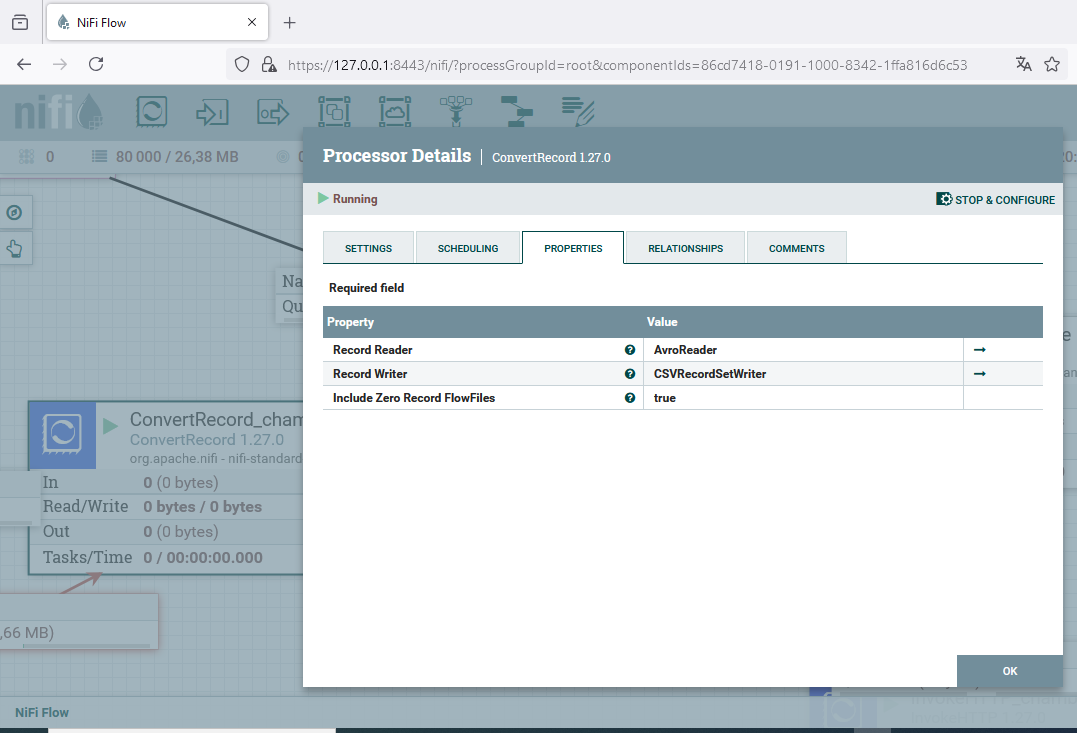
* Configurez le processeur ConvertRecord pour extraire les données de chambres d'hôtel.

Figure 14: Fenêtre de configuration du processus ConvertRecord\_chambres/ Source : Apache NiFi

* **Ajouter des Logiques de Transformation :**

Ajoutez des règles de transformation ou de validation des données selon les besoins.

* Si vous devez transformer les données extraites, vous pouvez ajouter un processus ConvertRecord après ExecuteSQL.
* Configurez ConvertRecord pour convertir les données du format SQL à un autre format (par exemple, JSON, Avro).
* Pour appliquer des règles spécifiques de transformation ou validation des données, utilisez des processus comme UpdateRecord ou ScriptedTransformRecord.

### **Chargement des données transformées**

* **Configuration du Processeur PutFile :**

Configurez PutFile pour enregistrer les données transformées dans un fichier ou les transférer via HTTP. Ces processeurs sauvegardent les données transformées dans un fichier sur le système de fichiers local. Cela permet d'archiver ou de stocker les données pour une analyse future.

* Configurez le processeur PutFile pour extraire les données de réservations d'hôtel.



Figure 15: Fenêtre de configuration du processus PutFile\_chambres/ Source : Apache NiFi

* Configurez le processeur PutFile pour extraire les données de chambres d'hôtel

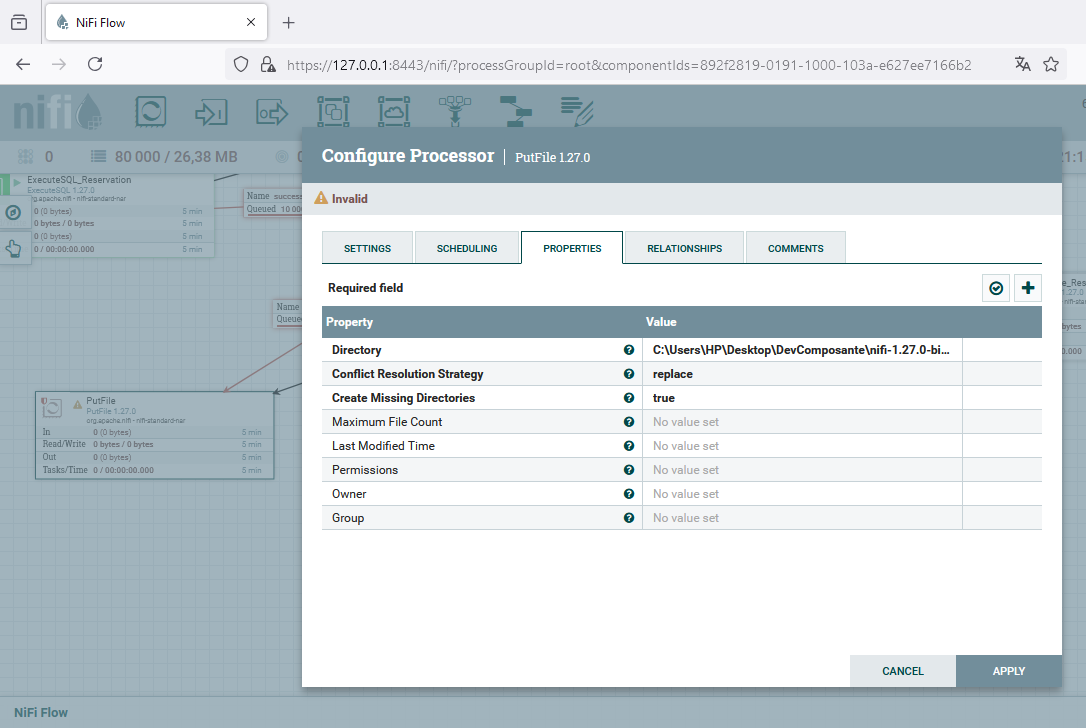


Figure 16: Fenêtre de configuration du processus PutFile\_Reservations/ Source : Apache NiFi

* **Utilisation de InvokeHTTP pour Intégration :**

Si les données doivent être envoyées à l'application Flask, configurez InvokeHTTP pour appeler une route spécifique. Ces processeurs envoient les données transformées via des requêtes HTTP vers une autre application ou un service web pour un traitement supplémentaire ou un stockage. Ils ont plusieurs connexions pour gérer les résultats (success, failure, Retry, etc.), indiquant différents scénarios de traitement.

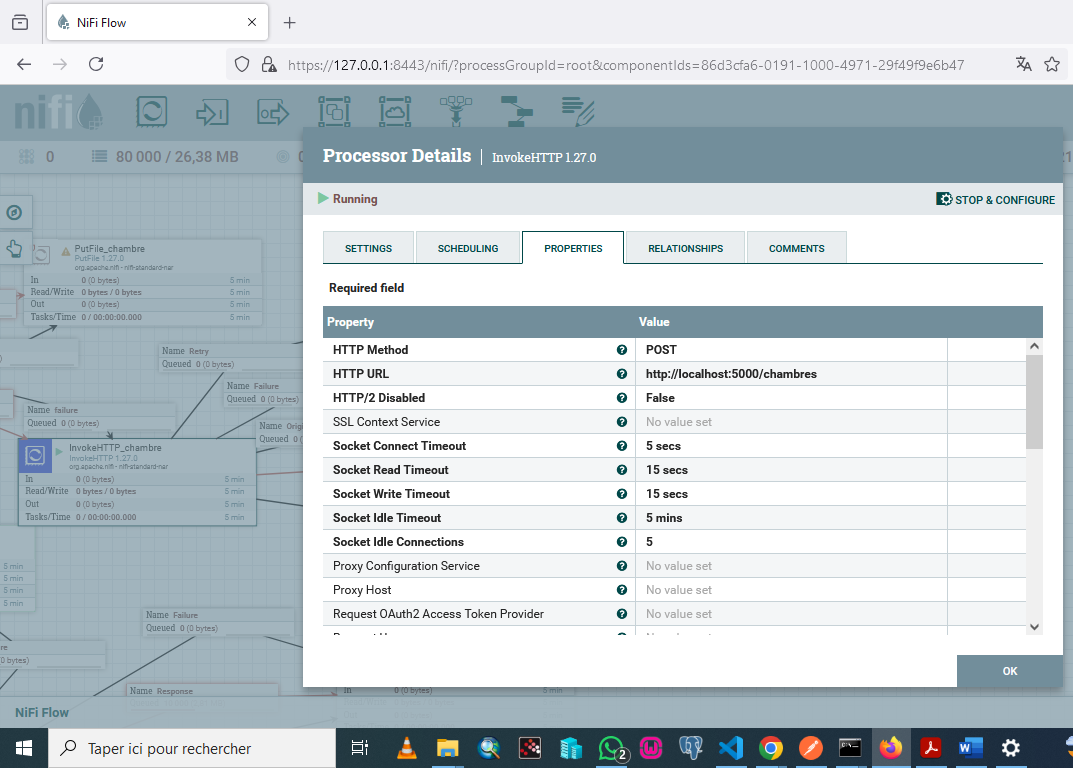
* Configurez le processeur InvokeHTTP pour extraire les données de chambres d'hôtel

Figure 17: Fenêtre de configuration du processus Invoke\_chambres/ Source : Apache NiFi

### **Gestion des erreurs et surveillance des flux de données**

Ces processeurs enregistrent les attributs des flux de données pour le débogage ou le suivi. Ils sont souvent utilisés pour surveiller le flux de données et pour diagnostiquer des problèmes potentiels.

* Configurez le processeur LogAttribute pour enregistrer les logs de chambres d'hôtel.

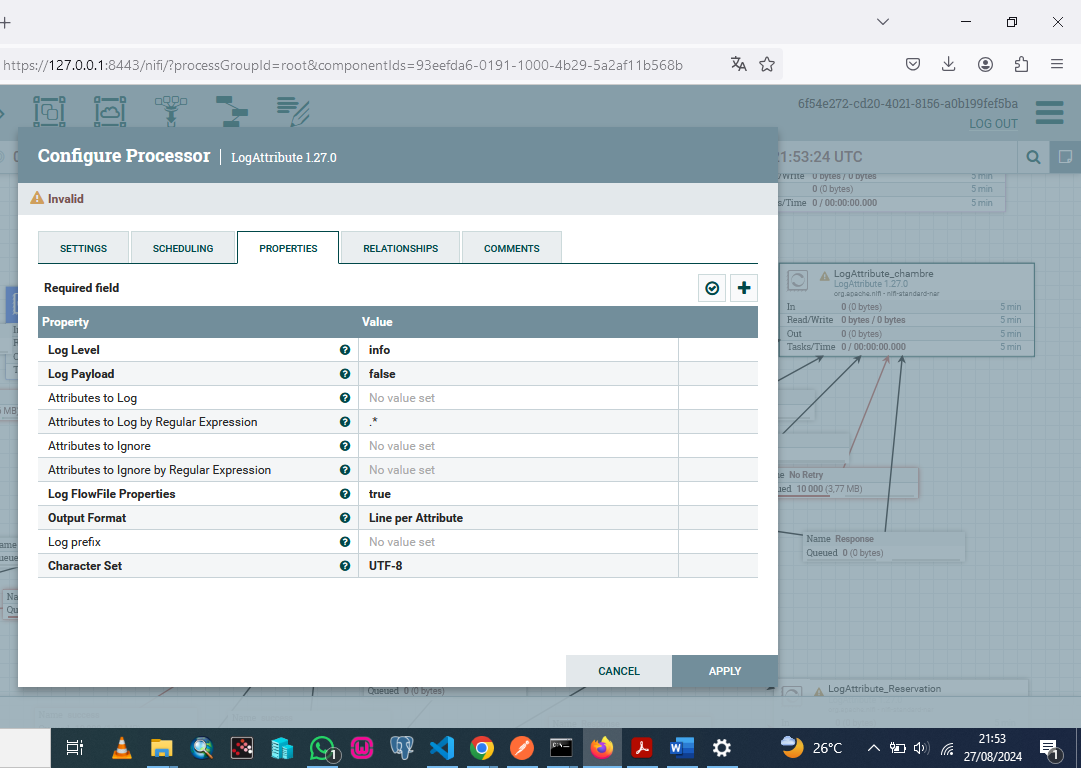


Figure 18: : Fenêtre de configuration du processus LogAttribute\_chambres/ Source : Apache NiFi

* **En résumé des processeurs utilisés dans le cadre de notre projet**

Ce flux NiFi est conçu pour extraire des données sur les chambres et les réservations, les transformer en un format approprié, puis les envoyer à un service web via HTTP ou les sauvegarder dans des fichiers. Les connexions gèrent les différents scénarios de succès ou d'échec à chaque étape du processus, et les données peuvent être suivies à chaque étape grâce aux processeurs LogAttribute.

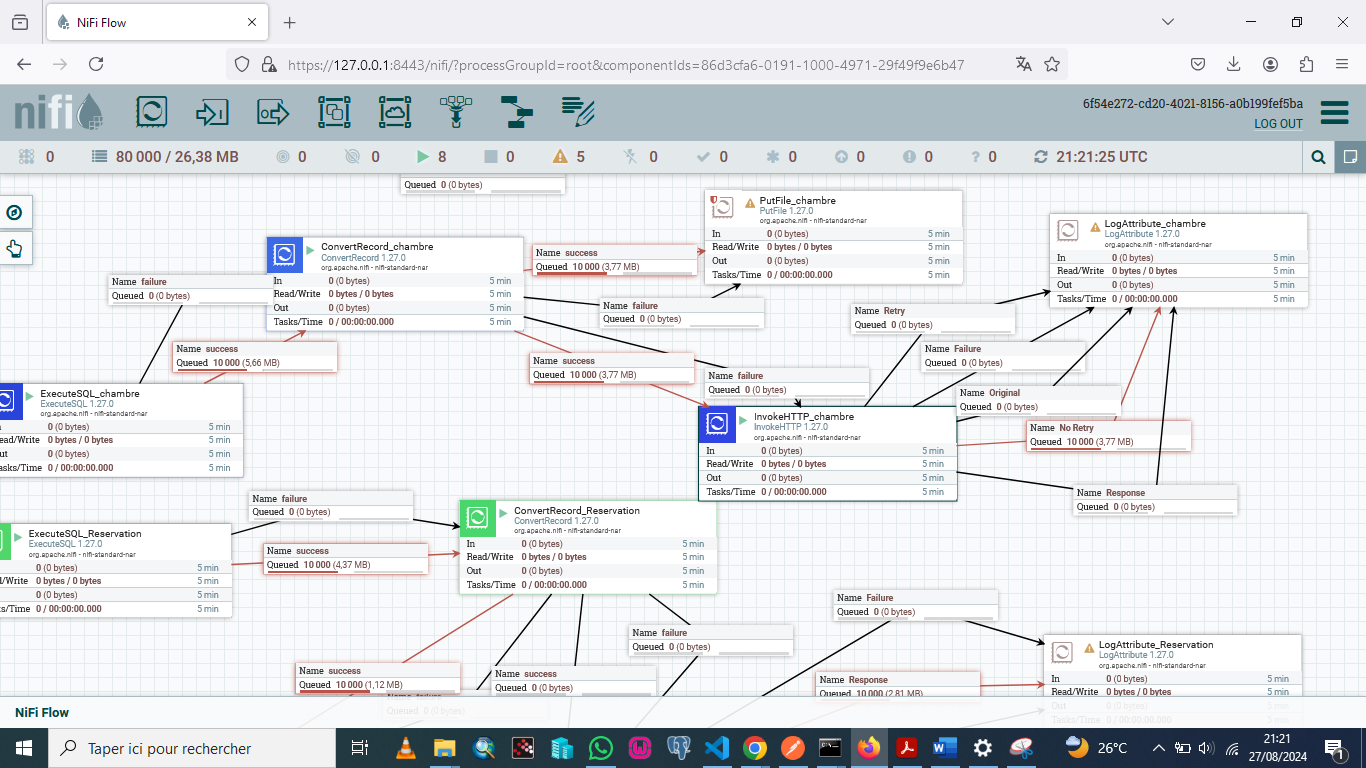


Figure 19:: Fenêtre de configuration globale des processus utilisés dans notre projet/ Source : Apache NiFi

# **Étape 2 : Développement de l'Application Flask**

## **Création la structure du projet**

Cette structure montre la structuration de notre **Application RESTful avec Flask** avec un environnement virtuel pour isoler les dépendances, des tests pour assurer la qualité du code, une base de données locale, et des fichiers de configuration pour contrôler le comportement de l'application.

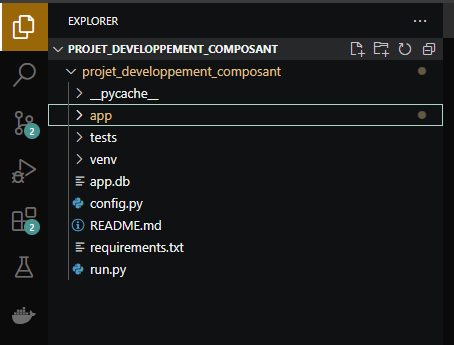


Figure 20: Fenêtre affichant la structure du notre projet/ Source : VS code

## **Connexion Flask à la base de données**

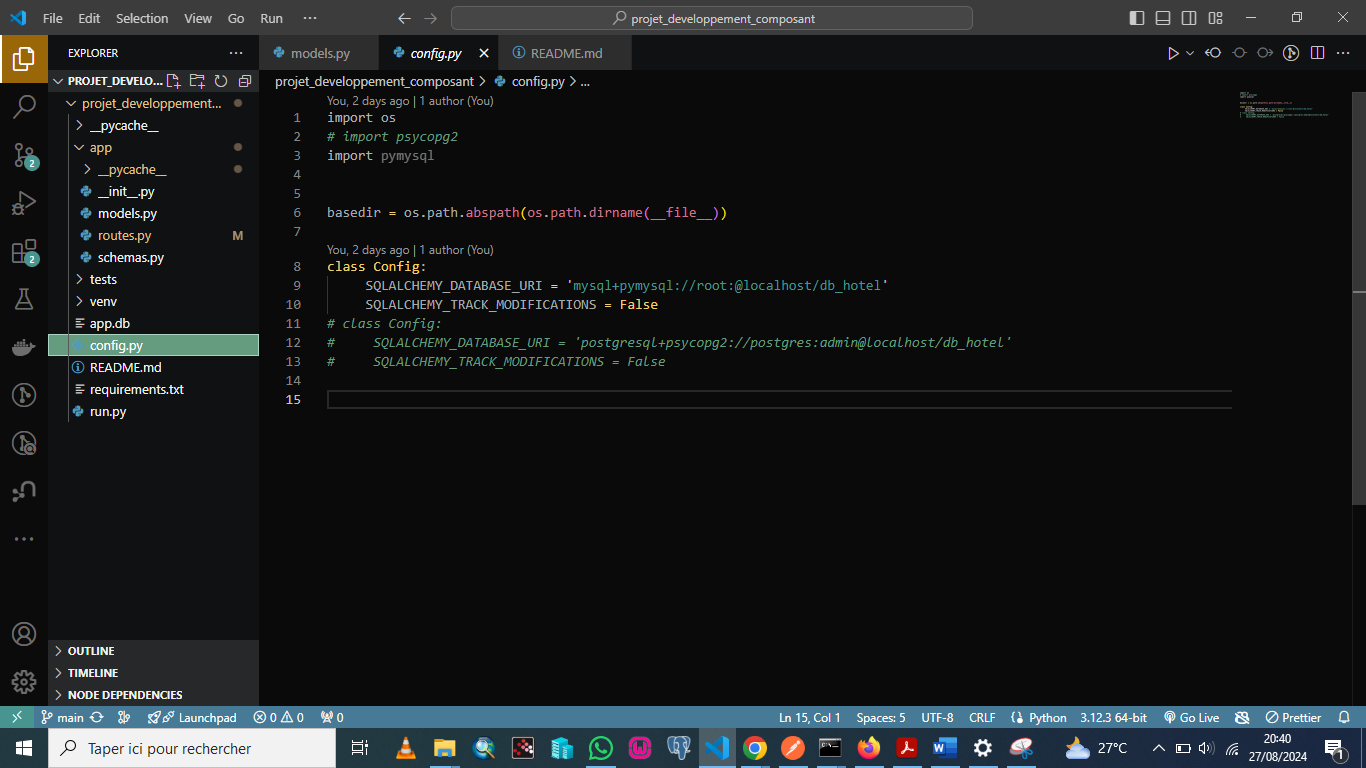


Figure 21: Fenêtre de configuration de ma BD à l’API RES full / Source : VS code

## **Création des Modèles et Routes**

* **Création des Modèles**

Ce code Flask gère un système de réservation d'hôtel en fournissant plusieurs fonctionnalités. Il permet de créer, lister, modifier, et supprimer des réservations et des chambres. Les chambres sont associées à des types, et chaque type a un coût et un nombre de chambres disponibles. Lorsqu'une réservation est effectuée, la disponibilité de la chambre est vérifiée, et elle est marquée comme louée. Les chambres peuvent être libérées une fois la réservation terminée. Des statistiques sur les chambres, telles que le nombre de chambres louées et disponibles pour chaque type, sont également fournies. Le code comprend aussi une fonctionnalité pour générer automatiquement des numéros de chambres uniques.

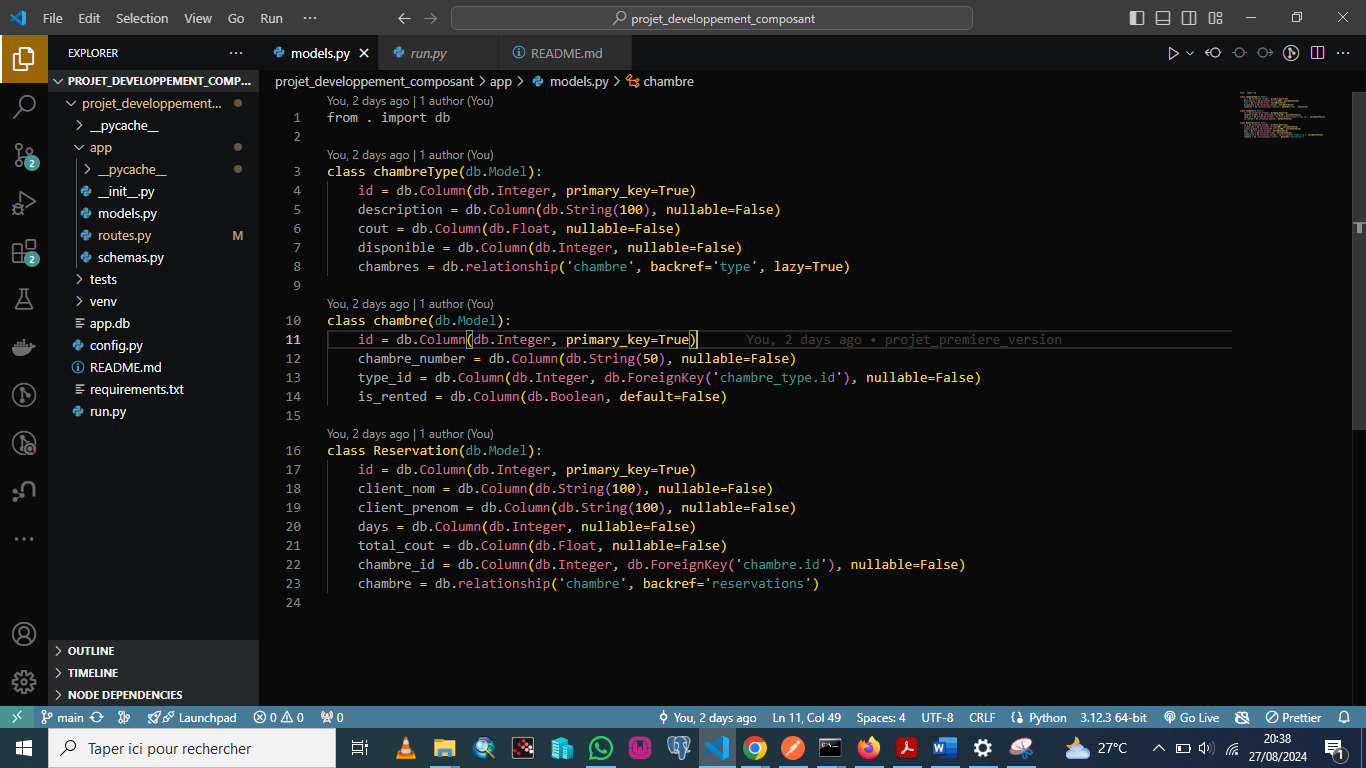


Figure 22: Fenêtre de indiquant mes classe/ Source : VS code

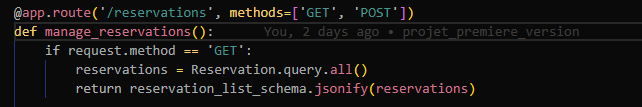
* **Création des Routes**

Ces captures définissent une série de routes pour une application Flask destinée à gérer un système de réservation d'hôtel.

Voici un résumé des principales fonctionnalités :

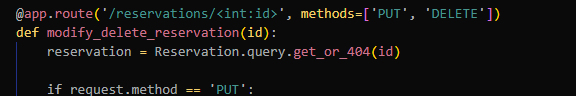
1. **Gestion des réservations** (/reservations) :

Permet de lister toutes les réservations (GET) ou d'ajouter une nouvelle réservation (POST). Lors de l'ajout, la disponibilité de la chambre est vérifiée, et si la chambre est libre, une réservation est créée et la chambre est marquée comme louée.



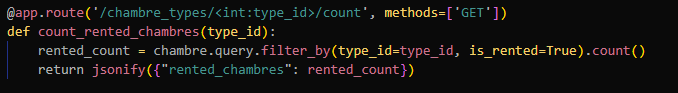
1. **Modification ou suppression des réservations** (/reservations/<int:id>):

Permet de mettre à jour (PUT) ou de supprimer (DELETE) une réservation existante. Lors de la suppression, la chambre est marquée comme disponible.



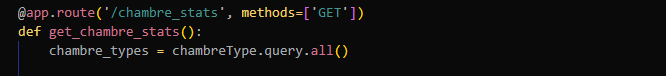
1. **Compter les chambres louées par type** (/chambre\_types/<int:type\_id>/count):

Compte et retourne le nombre de chambres louées pour un type spécifique de chambre.



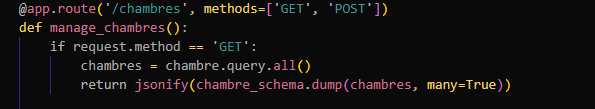
1. **Statistiques des chambres** (/chambre\_stats) :

Retourne des statistiques sur les chambres, y compris le nombre de chambres louées, libres, et le total par type de chambre.



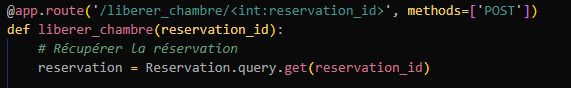
1. **Gestion des chambres** (/chambres) :

Permet de lister toutes les chambres (GET) ou d'ajouter une nouvelle chambre (POST).



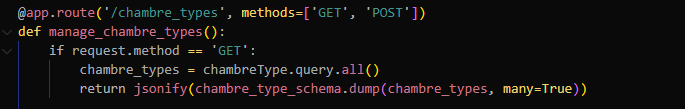
1. **Libération de chambre** (/liberer\_chambre/<int:reservation\_id>):

Libère une chambre associée à une réservation en la marquant comme disponible.



1. **Gestion des types de chambres** (/chambre\_types) :

Permet de lister tous les types de chambres (GET) ou d'ajouter un nouveau type de chambre (POST). Lors de l'ajout, un certain nombre de chambres sont automatiquement créées en fonction du nombre spécifié comme disponible.



Ces routes couvrent la plupart des opérations CRUD (Create, Read, Update, Delete) pour la gestion des réservations, des chambres et des types de chambres dans un hôtel.

## **Démarrez votre application**

Après avoir effectué toutes les vérifications, redémarrons notre application Flask à travers la commande “Python run.py” via une invite de commande en se plaçant sur le répertoire indiqué *C:\Users\HP\Desktop\DevComposante\projet\_developpement\_composant\projet\_developpement\_composant.* Le fichier run.py est le point d'entrée de notre application Flask. Il contient le code pour démarrer l'application Flask, comme la création de l'instance de l'application et l'appel à app.run() pour lancer le serveur.

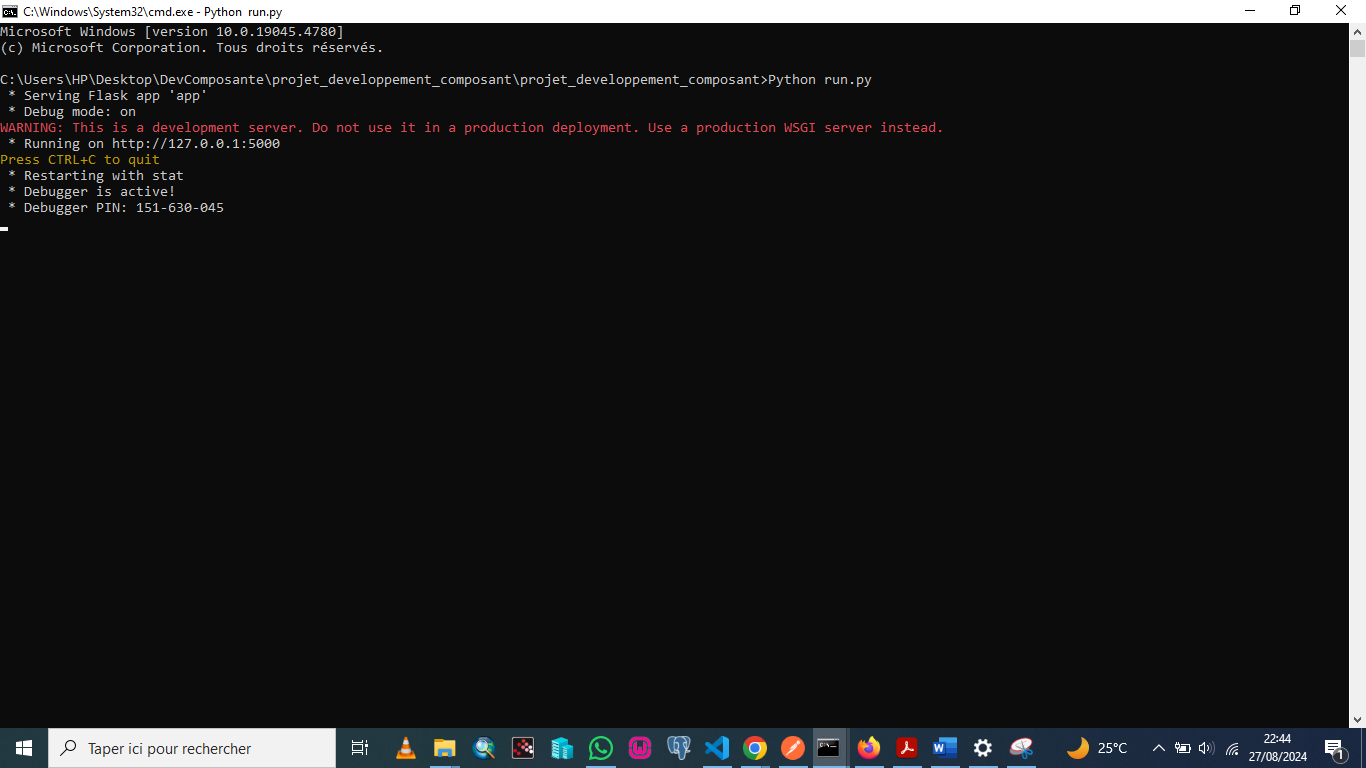


Figure 23: Figure 22: Fenêtre de indiquant le démarrage de mon API RESTfull/ Source : VS code

# **Étape 3 : Documentation et Test de l’API avec Postman**

## **Documenter le Processus**

Dans cette partie nous allons créer une documentation claire et compréhensible des tests que nous allons exécuter avec Postman, incluant les étapes à suivre, les entrées à fournir, et les résultats attendus.

**Documentation des APIs Flask avec Flask Swagger UI**

La documentation des APIs RESTful est une étape cruciale dans le développement d'applications web. Elle permet aux développeurs d'avoir une vue d'ensemble des endpoints disponibles, des paramètres requis, des réponses possibles, et des erreurs attendues. Pour automatiser cette documentation et la rendre facilement accessible, l'utilisation d'outils comme Swagger UI est devenue une pratique courante.

**Flask Swagger UI**

Flask Swagger UI est une extension de Flask qui permet de générer et de visualiser la documentation des APIs directement à partir d'un fichier de configuration Swagger (au format JSON ou YAML). Cette documentation est accessible via une interface utilisateur graphique, ce qui rend la consultation des différents endpoints de l'API très intuitive.

**Fonctionnalités de Flask Swagger UI**

* **Génération automatique de la documentation** : À partir d'un fichier swagger.json ou swagger.yaml, Flask Swagger UI génère automatiquement une documentation complète de l'API.
* **Interface interactive** : Les utilisateurs peuvent tester les différents endpoints directement depuis l'interface Swagger UI, en envoyant des requêtes HTTP et en visualisant les réponses en temps réel.
* **Support des schémas** : Swagger UI prend en charge les schémas JSON, permettant de décrire précisément les formats de données attendus par l'API, tant pour les requêtes que pour les réponses.

**Mise en œuvre**

Pour intégrer Flask Swagger UI dans une application Flask, voici les étapes principales :

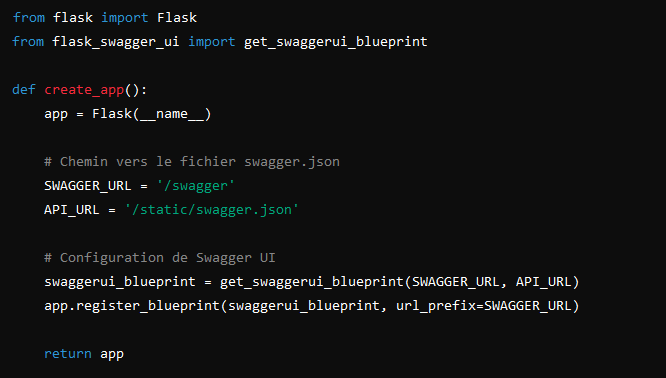
1. **Installation de la bibliothèque Flask Swagger UI**



1. **Création du fichier swagger.json**

Ce fichier contient la définition de l'API, y compris les endpoints, les méthodes HTTP supportées, les paramètres, et les schémas de réponse.

1. **Configuration de l'application Flask**



1. **Accès à la documentation**

Une fois l'application Flask démarrée, la documentation de l'API est accessible via l'URL spécifiée (/swagger).

**Flask Swagger UI** est un outil puissant pour la documentation des APIs Flask. Il permet de produire une documentation claire, interactive, et facile à maintenir, ce qui est essentiel pour le développement d'APIs robustes et bien documentées.

1. **Gestion des Réservations**

* Endpoint /reservations (GET, POST) :
  + GET : Récupère et renvoie toutes les réservations existantes dans la base de données sous forme de JSON.
  + POST : Crée une nouvelle réservation en vérifiant d'abord si la chambre demandée est disponible. Si elle est disponible, elle est marquée comme louée, et la réservation est créée avec le coût total calculé en fonction du nombre de jours.
* Endpoint /reservations/<int:id> (PUT, DELETE):
  + PUT : Modifie une réservation existante en mettant à jour les informations du client, le nombre de jours, et le coût total.
  + DELETE : Supprime une réservation et marque la chambre associée comme libre (non louée).

1. **Gestion des Types de Chambres**

* Endpoint /chambre\_types/<int:type\_id>/count (GET):
  + Compte et renvoie le nombre de chambres louées pour un type de chambre spécifique.
* Endpoint /chambre\_stats (GET) :
  + Renvoie des statistiques sur les chambres pour chaque type de chambre, y compris le nombre de chambres louées, libres et le total.
* Endpoint /chambre\_types (GET, POST) :
  + GET : Récupère et renvoie tous les types de chambres disponibles.
  + POST : Crée un nouveau type de chambre et génère automatiquement un ensemble de chambres associées à ce type, en fonction du nombre de chambres disponibles spécifié.

1. **Gestion des Chambres**

* Endpoint /chambres (GET, POST) :
  + GET : Récupère et renvoie toutes les chambres existantes.
  + POST : Crée une nouvelle chambre avec un numéro unique et l'associe à un type de chambre donné.
* Endpoint /liberer\_chambre/<int:reservation\_id> (POST):
  + Libère une chambre en marquant une réservation comme terminée et en rendant la chambre disponible à nouveau.

## **Création des tests Postman**

Pour tester les opérations GET et POST pour les modèles chambreType, chambre, et Réservation via Postman, voici comment nous avons procédé :

1. **Tester les Opérations GET**

* **Tester GET /reservations**
* **Ouvrir Postman** :
  + Lancez Postman et créez une nouvelle requête.
* **Configurer la Requête** :
  + **Méthode** : GET
  + **URL** : http://localhost:5000/reservations
* **Envoyer la Requête** :
  + Cliquez sur le bouton Send.
* **Vérifier la Réponse** :

Cet endpoint récupère une liste de réservations.

**Requête**  
Ceci est une requête GET à l’Endpoint http://localhost:5000/reservations.

**Réponse**  
La réponse est au format JSON ou en visuel avec un code de statut 200. Elle retourne un tableau d'objets, où chaque objet représente une réservation. Les propriétés de chaque objet de réservation incluent comme le montre la figure 24 :

* **chambre\_id** (nombre) : L'ID de la chambre pour la réservation.
* **client\_nom** (chaîne de caractères) : Le nom de famille du client faisant la réservation.
* **client\_prenom** (chaîne de caractères) : Le prénom du client faisant la réservation.
* **days** (nombre) : Le nombre de jours pour la réservation.
* **id** (nombre) : L'identifiant unique pour la réservation.
* **total\_cout** (nombre) : Le coût total de la réservation.

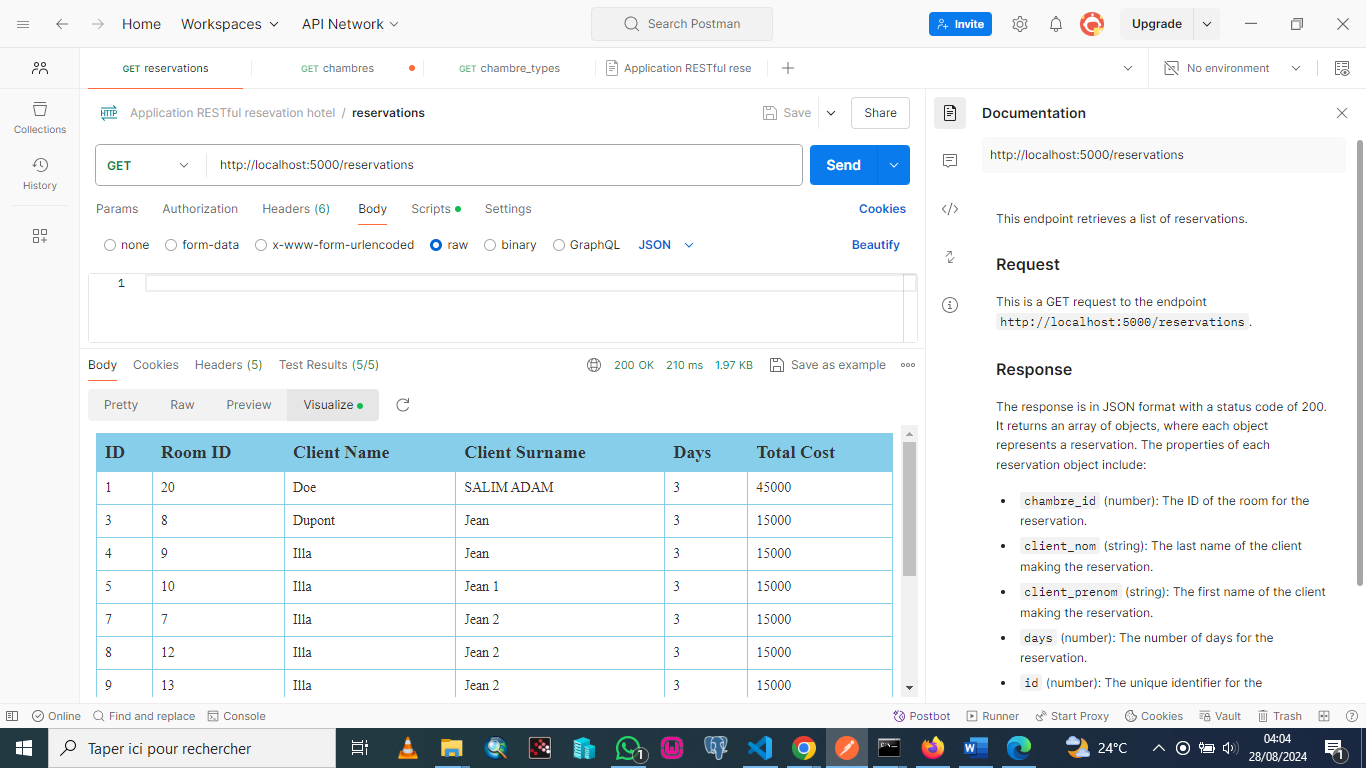


Figure 24: Fenêtre de Test GET /réservations

* **Tester GET /chambre\_types**
* **Configurer la Requête** :
  + **Méthode** : GET
  + **URL:** http://localhost:5000/chambre\_types.
* **Envoyer la Requête** :
  + Cliquez sur le bouton Send.
* **Vérifier la Réponse** :

**Cet endpoint effectue une requête HTTP GET pour récupérer une liste de types de chambres depuis le serveur.**

* **Requête**  
  Méthode : GET  
  URL : http://localhost:5000/chambre\_types
* **Réponse**  
  Statut : 200  
  Content-Type : application/json

La réponse contient un tableau de types de chambres, où chaque type inclut les attributs suivants comme le montre la **figure suivante** :

* **chambres** (tableau) : La liste des IDs des chambres associées au type.
* **cout** (nombre) : Le coût du type de chambre.
* **description** (chaîne de caractères) : Une description du type de chambre.
* **disponible** (nombre) : Le statut de disponibilité du type de chambre.
* **id** (nombre) : L'identifiant unique du type de chambre.

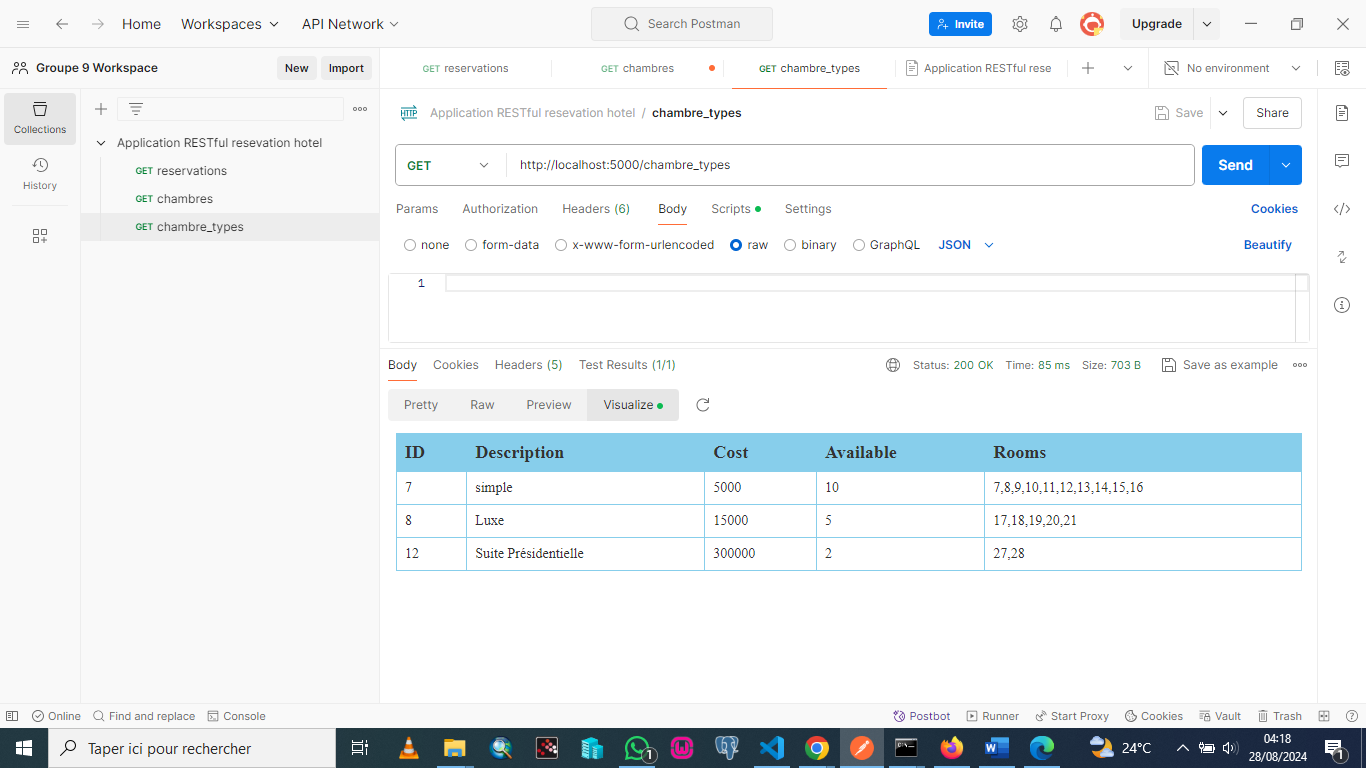


Figure 25: Fenêtre de Test GET /chambres\_types

* **Tester GET /chambres**
* **Configurer la Requête** :
  + **Méthode** : GET
  + **URL** : http://localhost:5000/chambres.
* **Envoyer la Requête** :
  + Cliquez sur le bouton Send.
* **Vérifier la Réponse** :

**Obtenir les Chambres**  
Cet endpoint récupère une liste de chambres.

**Corps de la Requête**  
Cette requête ne nécessite pas de corps de requête.

* **Réponse**  
  Statut : 200  
  Content-Type : application/json

La réponse contient un tableau de chambres, où chaque objet chambre inclut le numéro de la chambre, l'ID, le statut de location, et l'ID du type de chambre comme le montre la figure 26.

* **chambre\_number** (chaîne de caractères) : Le numéro de la chambre.
* **id** (entier) : L'identifiant unique de la chambre.
* **is\_rented** (booléen) : Indique si la chambre est actuellement louée.
* **type\_id** (entier) : L'identifiant du type de chambre.

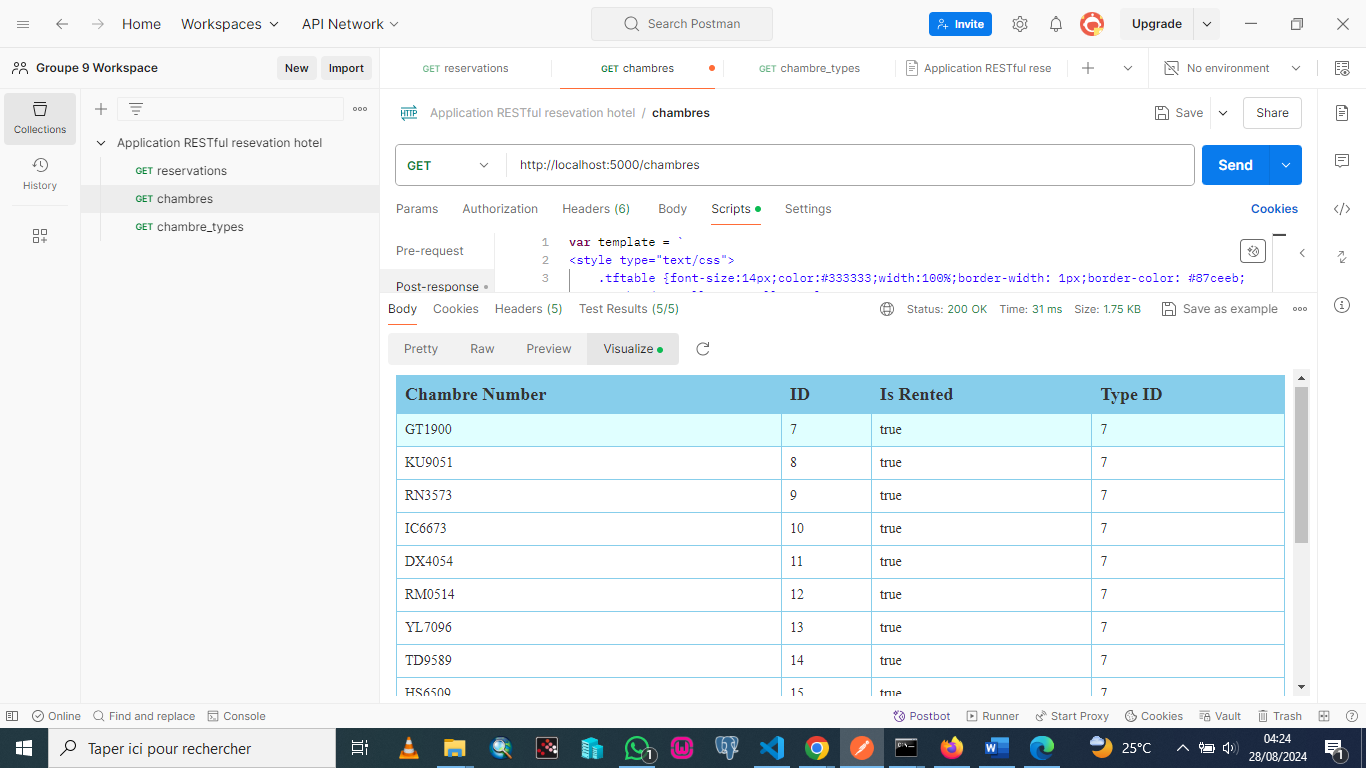


Figure 26: Fenêtre de Test GET /chambres

1. **Tester les Opérations POST**

* **Tester POST /reservations**
* **Ouvrir Postman** :
  + Lancez une nouvelle requête.
* **Configurer la Requête** :
  + **Méthode** : POST
  + **URL** : http://localhost:5000/reservations
  + **Headers** : Ajoutez un header Content-Type avec la valeur application/json.
  + **Body** : Sélectionnez raw et entrez un JSON avec les détails de la réservation
* **Corps de la requête**
* client\_nom (chaîne de caractères, requis) : Le nom de famille du client.
* client\_prenom (chaîne de caractères, requis) : Le prénom du client.
* chambre\_id (entier, requis) : L'ID de la chambre pour la réservation.
* days (entier, requis) : Le nombre de jours pour la réservation.

**Réponse**

Lors de la création réussie, le point de terminaison renvoie un objet JSON avec les champs suivants comme le montre la figure 27.

* id (entier) : L'ID de la réservation.
* client\_nom (chaîne de caractères) : Le nom de famille du client.
* client\_prenom (chaîne de caractères) : Le prénom du client.
* chambre\_id (entier) : L'ID de la chambre pour la réservation.
* days (entier) : Le nombre de jours pour la réservation.

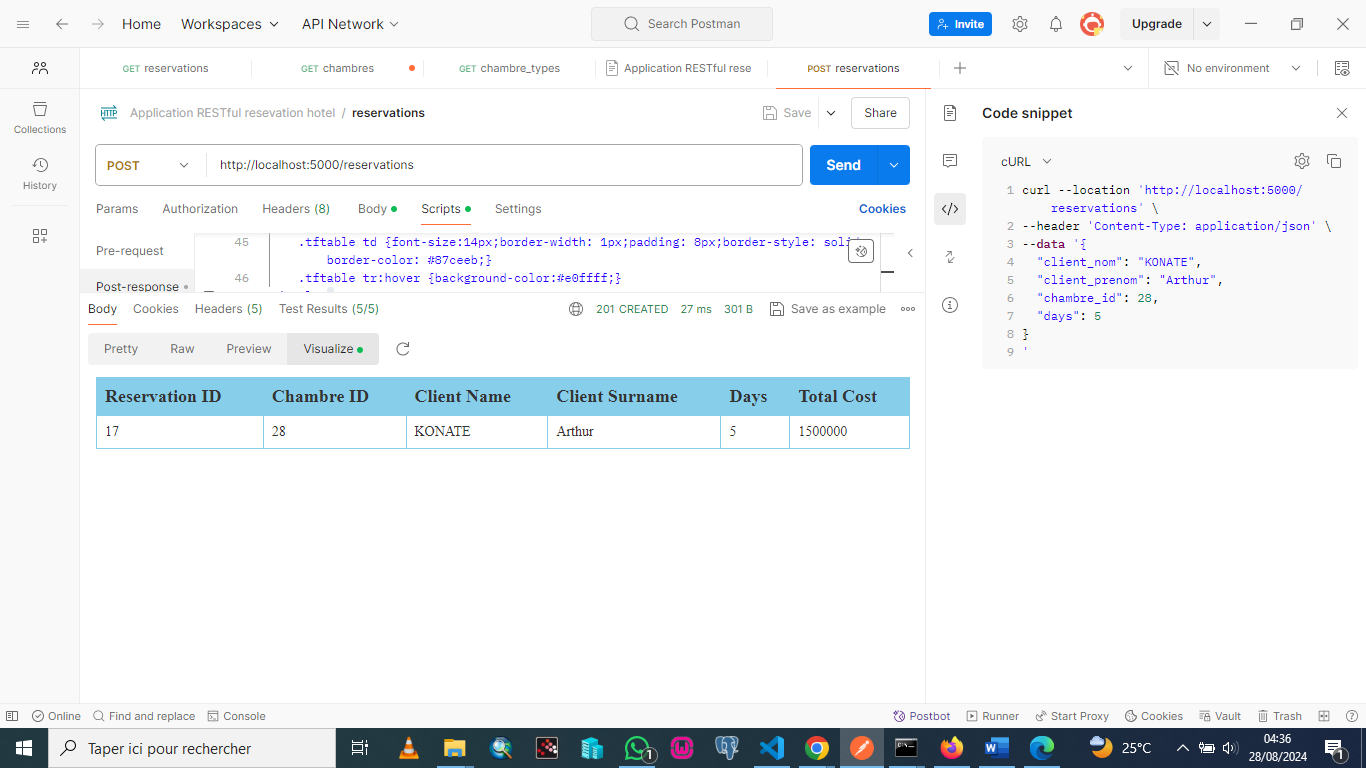


Figure 27: Fenêtre de Test POST /reservations

* **Tester POST /chambres\_types**

Cet endpoint est utilisé pour créer un nouveau type de chambre comme le montre la figure 28.

**Corps de la Requête**

* **description** (chaîne de caractères, requis) : Une brève description du type de chambre.
* **Cout** (chaîne de caractères, requis) : Le coût du type de chambre.
* **Disponible** (nombre, requis) : Indique la disponibilité du type de chambre.

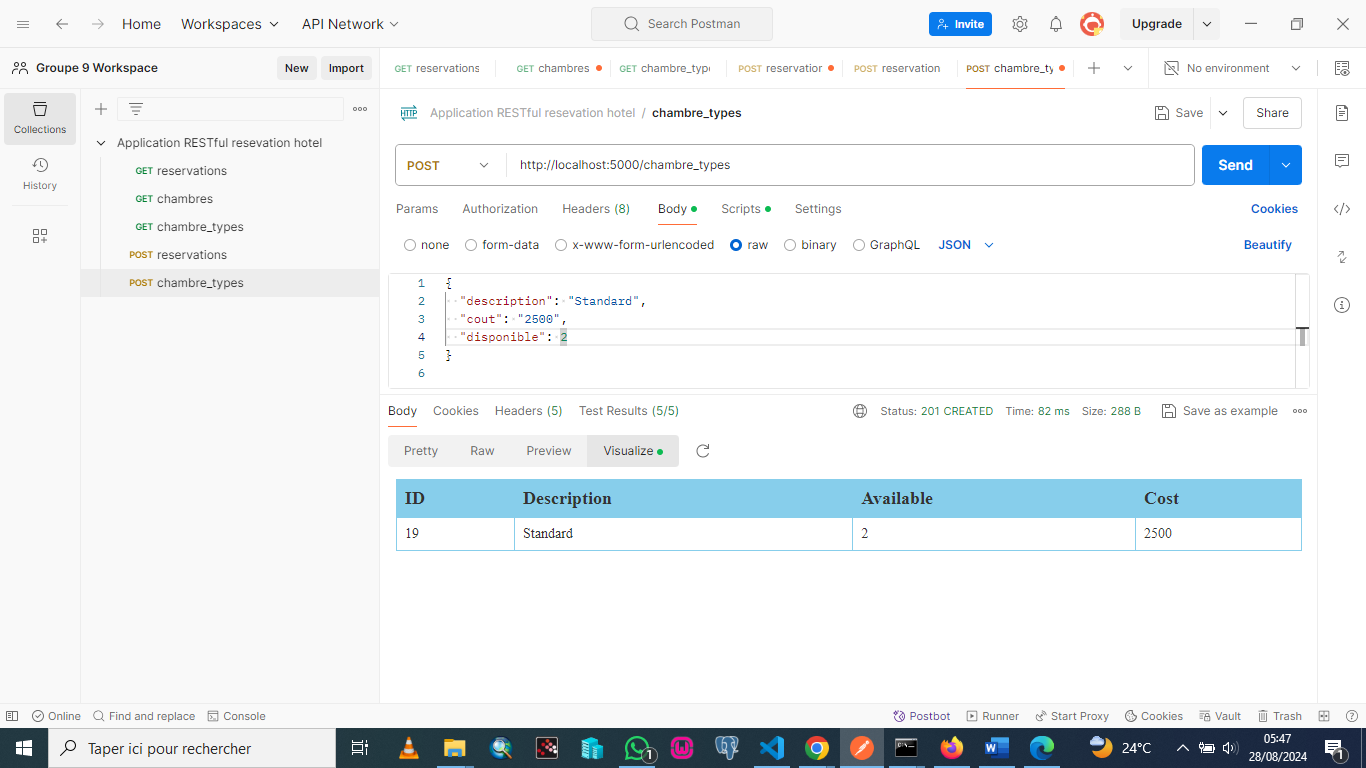


Figure 28:Fenêtre de Test POST /chambres\_types

* **Tester POST /chambres**

**Créer une Nouvelle Chambre**

Cet endpoint permet de créer une nouvelle chambre le montre la figure 29.

**Corps de la Requête :**

* **chambre\_number** (string, requis) : Le numéro de la chambre.
* **type\_id** (integer, requis) : L'ID du type de chambre.
* **is\_rented** (integer, requis) : Indique si la chambre est louée ou non.

**Réponse :**

* **Statut :** 201 Created
* **Content-Type :** application/json

**Corps de la Réponse :**

* **chambre\_number** (string) : Le numéro de la chambre.
* **id** (integer) : L'ID de la chambre créée.
* **is\_rented** (boolean) : Indique si la chambre est louée ou non.
* **type\_id** (integer) : L'ID du type de chambre.

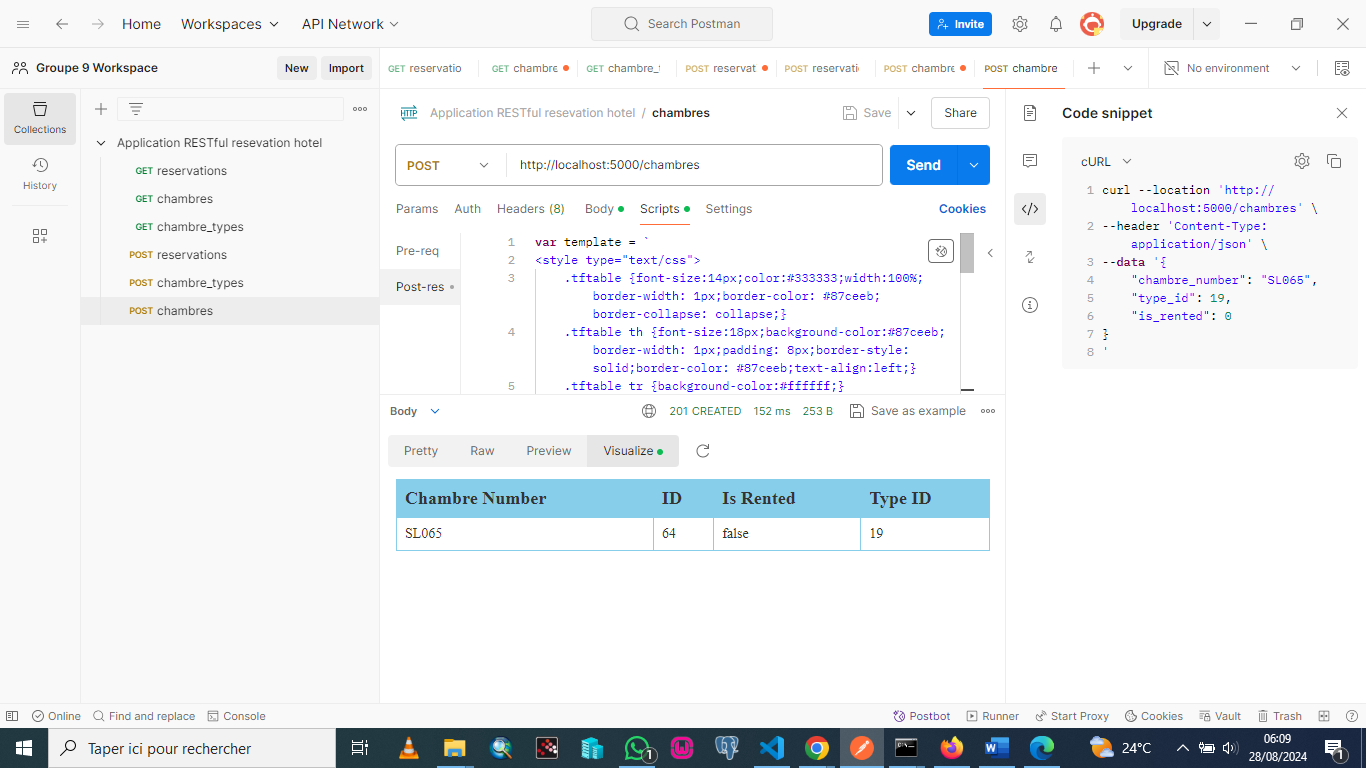


Figure 29: Fenêtre de Test POST /chambres

* **Tester GET / chambre\_stats**

**GET /chambre\_stats**

**Description de l'Endpoint :**

Cet endpoint récupère les statistiques liées aux chambres disponibles le montre la figure 30.

**Requête :**

* **Méthode :** GET
* **URL :** http://localhost:5000/chambre\_stats

**Réponse :**

* **Statut :** 200 OK
* **Content-Type :** application/json

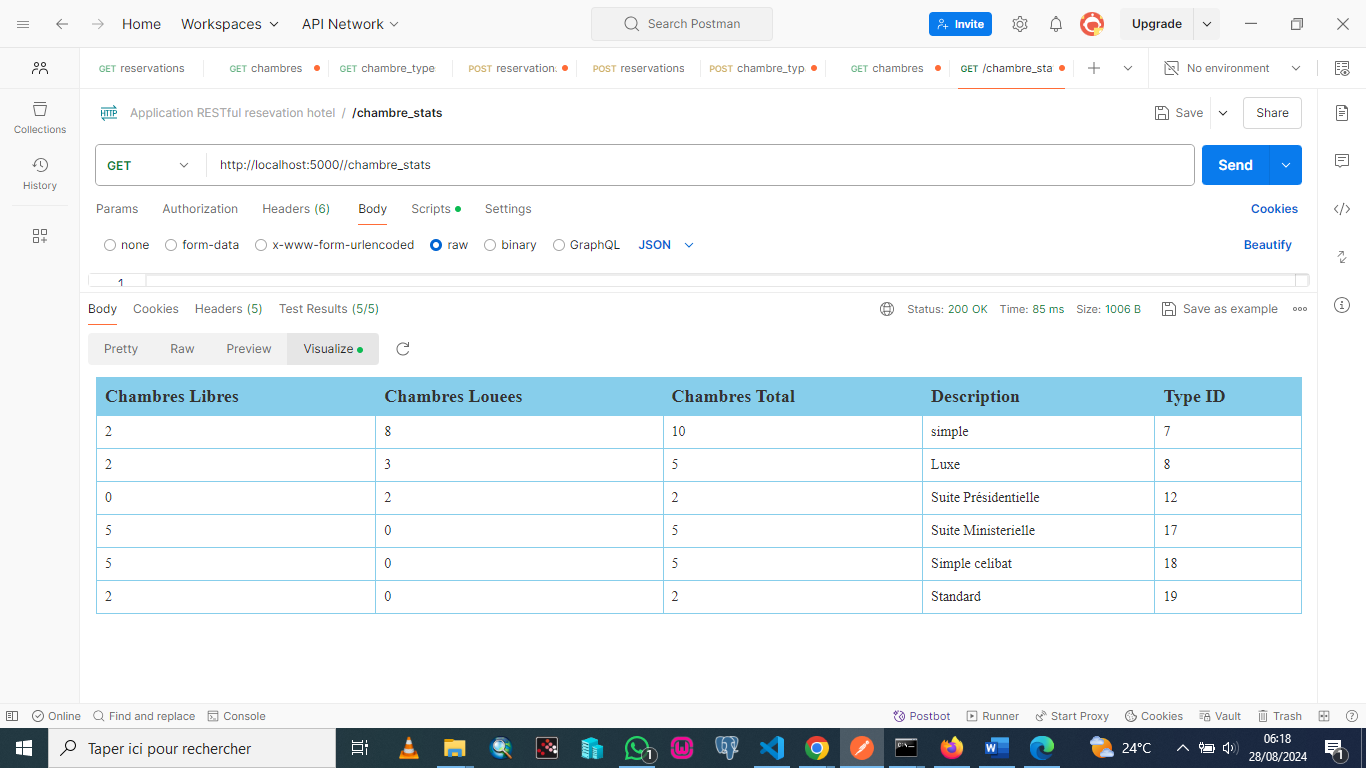


Figure 30:Fenêtre de Test GET /chambres\_stats

# **WEBOGRAPHIE**

<https://www.blogdumoderateur.com/tools/apache-nifi/>

<https://meritis.fr/blog/travailler-donnee-apache-nifi>

<https://www.youtube.com/watch?v=8KxcAiNdqvw>

<https://www.youtube.com/watch?v=0YROsMuqpFo>

<https://www.postman.com/>