**IBAM – M1 Informatique**

**Promotion 2023- 2024**

**TP : Projet de DEVELOPPEMENT A BASE DE COMPOSANTS ET SERVICES WEB**

**Exercice 3 : Application RESTful avec Flask ou django**

**Objectif : Développer une application RESTful en utilisant Flask qui utilise Apache Nifi pour ETL et tests avec Postman.**

**Rapport des travaux effectués**

**Groupe de travail : Groupe 10**

|  |  |  |  |
| --- | --- | --- | --- |
| **N° ordre** | **Nom** | **Prénom(s)** | **Rôle** |
|  | DOMBOUE | M. Victorien | Chef de projet |
|  | ILBOUDO | RAMANE | Développeur, |
|  | TAPSOBA | Mohamed Awal | Développeur |

[INTRODUCTION 3](#_Toc176727682)

[I. PRESENTATION DU PROJET 4](#_Toc176727683)

[I.1. Objectifs et consistance du projet 4](#_Toc176727684)

[I.2. Tâches planifiées du projet 4](#_Toc176727685)

[I.3. Approche méthodologique/démarche globale adoptée : étapes clés 4](#_Toc176727686)

[II. DEVELOPPEMENT DE L'API RESTFUL AVEC FLASK 5](#_Toc176727687)

[II.1. Présentation de Flask 5](#_Toc176727688)

[**II.1.1.** **Définition** 5](#_Toc176727689)

[**II.1.2.** **Cas d'utilisation de Flask :** 5](#_Toc176727690)

[**II.1.3.** **Avantages de Flask :** 5](#_Toc176727691)

[II.2. Présentation des fichiers de développement de l’application Flask 6](#_Toc176727692)

[**II.2.1.** **Présentation de l’arborescence de l’application : conception de l'architecture** 6](#_Toc176727693)

[**II.2.2.** **Présentation du fichier models.py de l’application : modélisation des données** 7](#_Toc176727694)

[**II.2.3.** **Présentation du fichier app.py de l’application** 9](#_Toc176727695)

[**II.2.4.** **Présentation du fichier config.py de l’application** 10](#_Toc176727696)

[**II.2.5.** **Présentation du fichier init.py de l’application** 11](#_Toc176727697)

[**II.2.6.** **Présentation du fichier routes.py de l’application** 12](#_Toc176727698)

[II.3. Lancement de l’application de l’application Flask 16](#_Toc176727699)

[**II.3.1.** **Exemple d’exposition du contenu de la table CHAMBRES via l’API RESTful** 16](#_Toc176727700)

[**II.3.2.** **Exemple d’exposition du contenu de la table CLIENTS via l’API RESTful** 16](#_Toc176727701)

[**II.3.3.** **Exemple d’exposition du contenu de la table RESERVATIONS via l’API RESTful** 17](#_Toc176727702)

[III. CONFIGURATION NIFI 17](#_Toc176727703)

[III.1. Définition et contexte d’utilisation de NIFI 17](#_Toc176727704)

[III.2. Installation de Apache NiFi. et configuration des connexions à la base de données : 20](#_Toc176727705)

[III.3 Conception du flux ETL. 21](#_Toc176727706)

[**III.3.1 Extraction des Données (à l’aide d’un composant ExecuteSQL ou QueryDatabaseTable :** 21](#_Toc176727707)

[**III.3.2.** **Transformation : à l’aide des processors comme UpdateRecord ou ConvertRecord.** 21](#_Toc176727708)

[**III.3.3.** **Chargement : à l’aide de PutDatabaseRecord ou PutFile.** 21](#_Toc176727709)

[IV. VALIDATION AVEC POSTMAN 22](#_Toc176727710)

[IV.1. Définition et contexte d’utilisation de Postman 22](#_Toc176727711)

[IV.2. Scénarios de Tests Définis 24](#_Toc176727712)

[**IV.2.1.** **Avantages de l'utilisation de Postman pour les tests :** 24](#_Toc176727713)

[**IV.2.2.Test de création d'une réservation (POST) avec Postman** 25](#_Toc176727714)

[CONCLUSION 27](#_Toc176727715)

# **INTRODUCTION**

Chaque année universitaire, à la fin du cours de Développement à base de composants et services web, les étudiants en première année de Master Informatique de l’IBAM ont l’obligation de présenter un projet concret.

Pour satisfaire à une telle exigence, le groupe 10 de la promotion M1 informatique 2023-2024, dont nous sommes membres, a fait le choix de s’intéresser au projet dont l’objectif est de *«****Développer une application RESTful en utilisant Flask qui utilise Apache Nifi pour ETL et tests avec Postman.».****,*

Au terme de notre étude et suite à des échanges nourris entre les membres de l’équipe, le Rapport des travaux effectués présente les livrables suivants : (i) Présentation générale du projet ; (ii) développement de l'API RESTful avec Flask ; (iii) Configuration et processus ETL avec Apache NiFi ; (iv) Utilisation de Postman pour tester les endpoints de l'API et vérifier leur fonctionnalité.

# **I. PRESENTATION DU PROJET**

## **I.1. Objectifs et consistance du projet**

Le projet consiste à développer une application RESTful en utilisant Flask qui utilise Apache Nifi pour ETL et tests avec Postman.

## **I.2. Tâches planifiées du projet**

Les tâches à exécuter pour mener à bien le projet se résument ainsi qu’il suit :

* Utiliser Apache Nifi pour extraire et transformer des données de réservations d'hôtel dans MySQL/PostgreSql.
* Développer une application Flask pour exposer ces données via une API RESTful.
* Utiliser Postman pour tester les endpoints de l'API et vérifier leur fonctionnalité.
* Documenter les tests Postman et assurez-vous que l'API fonctionne correctement.

## **I.3. Approche méthodologique/démarche globale adoptée : étapes clés**

1. **Planification et Conception** : Définir les objectifs, concevoir l'architecture, et modéliser les données.
2. **Développement Flask** : Configurer l'environnement, définir les modèles et routes, et développer l'application.
3. **Configuration NiFi** : Installer NiFi, concevoir les flux ETL, et configurer les connexions.
4. **Tests avec Postman** : Créer des tests pour chaque endpoint, vérifier les réponses, et automatiser les tests.
5. **Déploiement et Surveillance** : Déployer l'application, configurer la surveillance, et gérer les alertes.
6. **Documentation et Maintenance** : Documenter l'API et maintenir l'application.

Cette méthodologie nous guidera à travers les étapes clés pour développer, tester, et déployer une application RESTful intégrée avec Apache NiFi et testée avec Postman.

# **II. DEVELOPPEMENT DE L'API RESTFUL AVEC FLASK**

## **II.1. Présentation de Flask**

### **II.1.1. Définition**

**Flask** est un framework web léger et minimaliste écrit en Python, qui permet de créer des applications web rapidement et avec flexibilité. Il est conçu pour être simple à utiliser et pour donner aux développeurs un grand contrôle sur les composants qu'ils souhaitent inclure, tout en étant extensible pour des projets plus complexes.

### **II.1.2. Cas d'utilisation de Flask :**

* **API RESTful** : Flask est fréquemment utilisé pour développer des API RESTful, grâce à sa gestion simple des routes et des requêtes HTTP.
* **Prototypes rapides** : Sa simplicité et flexibilité en font un excellent choix pour créer des prototypes ou des projets expérimentaux.
* **Applications web complètes** : Avec les bonnes extensions (comme Flask-SQLAlchemy pour la gestion de base de données), Flask peut être utilisé pour développer des applications web complexes.

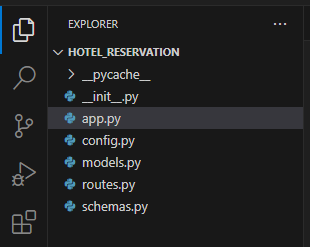
### **II.1.3. Avantages de Flask :**

* **Simple et léger** : Idéal pour les petits projets ou pour apprendre.
* **Personnalisable** : Vous n'êtes pas limité par une structure imposée, ce qui vous permet d'ajouter uniquement les composants nécessaires.
* **Communauté et extensions** : Une large communauté et une grande variété d'extensions sont disponibles.
* **Facilité de prise en main** :
  + Flask a une syntaxe simple et intuitive, ce qui le rend facile à apprendre, particulièrement pour ceux qui commencent avec Python ou le développement web.
* **Extensibilité** :
  + Bien que minimaliste, Flask est extrêmement extensible. Il existe de nombreuses extensions tierces pour ajouter des fonctionnalités comme la gestion des bases de données avec SQLAlchemy, la gestion des sessions, la sécurité, ou la génération de formulaires.
* **Flexibilité** : Flask ne force pas une architecture particulière, ce qui permet aux développeurs de structurer leurs applications comme ils le souhaitent. Il est facile de l'adapter à des projets de petite, moyenne ou grande envergure.
* **Modularité** : Vous pouvez organiser votre application en "blueprints", ce qui facilite la gestion de grandes applications en les divisant en modules plus petits et plus gérables

## **II.2. Présentation des fichiers de développement de l’application Flask**

### **II.2.1. Présentation de l’arborescence de l’application : conception de l'architecture**

L’arborescence d’une application Flask est la structure de répertoires et de fichiers qui organise le projet. Elle permet de gérer efficacement les composants d'une application comme les fichiers de code, les templates HTML, les fichiers statiques (CSS, JavaScript, images), et la configuration. Dans le cas de notre application, l’arborescence se présente comme suit :



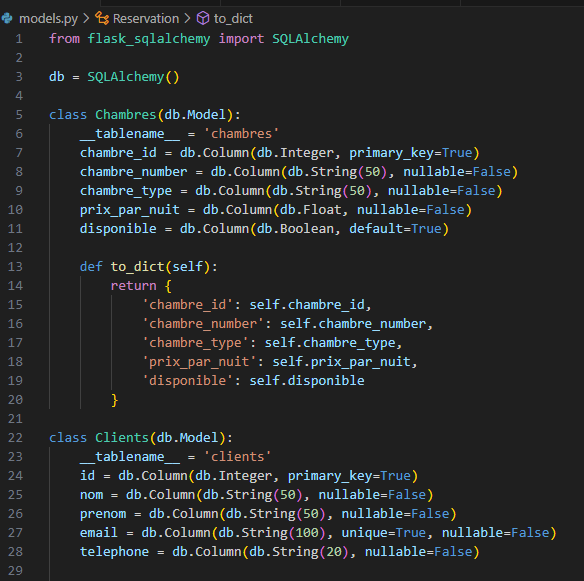
### **II.2.2. Présentation du fichier models.py de l’application : modélisation des données**

Le fichier **models.py** dans une application Flask est généralement utilisé pour définir les **modèles de données** qui représentent les tables dans une base de données relationnelle. Ces modèles sont des classes Python qui décrivent la structure des tables, les colonnes, les types de données, et les relations entre les tables. En utilisant un ORM (Object-Relational Mapping) comme **SQLAlchemy**, ces classes sont ensuite mappées automatiquement sur les tables de la base de données.

**Contenu typique de models.py :**

1. **Configuration de SQLAlchemy** :
   * L'instance de la base de données (souvent appelée db) est configurée ici et est utilisée pour interagir avec la base de données.
2. **Définition des modèles** :
   * Les modèles sont des classes Python qui héritent de db.Model. Chaque modèle correspond à une table dans la base de données.
3. **Colonnes et types de données** :
   * Les attributs des classes représentent les colonnes des tables. Chaque colonne est définie avec un type de données spécifique (par exemple, Integer, String, Float).
4. **Relations entre les tables** :
   * Les relations entre les tables (comme les relations un-à-plusieurs, plusieurs-à-plusieurs) peuvent également être définies ici.

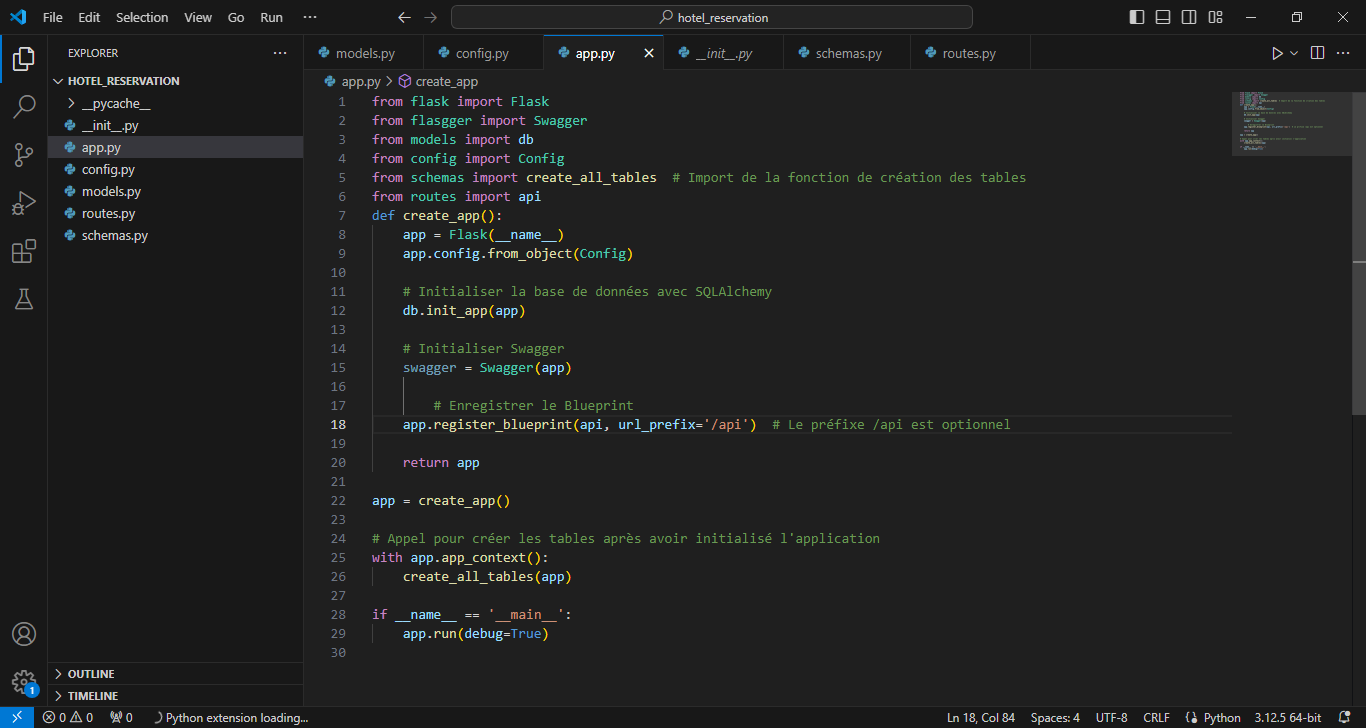
Le fichier models.py de notre projet se présente comme suit :





### **II.2.3. Présentation du fichier app.py de l’application**

Le fichier app.py est un script Python qui sert généralement de point d'entrée pour les applications web développées avec le framework **Flask**. Il définit l'application Flask, configure les routes (endpoints de l'API), initialise les bases de données et d'autres composants nécessaires à l'exécution de l'application. Dans le cadre de notre application, le fichier app.py se présente comme suit :



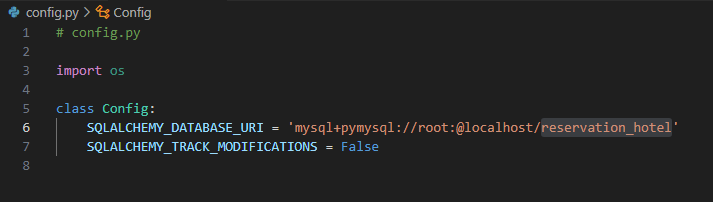
En résumé, app.py est le fichier central où tout est orchestré dans notre projet Flask.

### **II.2.4. Présentation du fichier config.py de l’application**

Le fichier **config.py** dans une application Flask est un fichier dédié à la gestion des configurations globales de l'application. Il permet de centraliser toutes les configurations importantes dans un seul endroit, facilitant la gestion, la sécurité et la flexibilité du projet, plus particulièrement :

* **Centralisation des configurations** : Toutes les variables et options de configuration sont stockées dans un seul fichier, ce qui rend le projet plus maintenable.
* **Facilité de changement** : Si vous devez changer des paramètres (comme passer d'une base de données de développement à une base de données de production), il vous suffit de modifier ce fichier.
* **Séparation des environnements** : Vous pouvez définir des configurations spécifiques pour différents environnements (développement, test, production).
* **Sécurité** : Les informations sensibles (comme les clés API ou les mots de passe) peuvent être gérées de manière plus sécurisée.

Pour notre application, ce fichier config.py se présente ainsi qu’il suit :



En résumé, le fichier config.py est une manière ordonnée et flexible de gérer les configurations de l'application Flask, permettant de facilement changer de configuration selon l'environnement (développement, production, test).

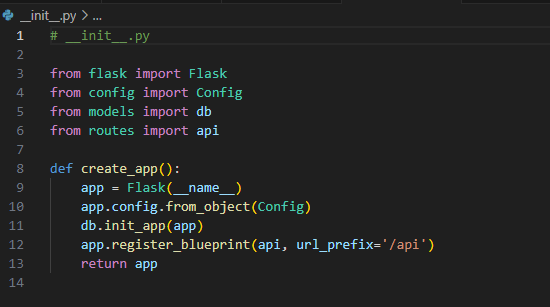
### **II.2.5. Présentation du fichier init.py de l’application**

Le fichier \_\_init\_\_.py est un fichier spécial dans les projets Python qui indique à Python que le répertoire où il se trouve doit être traité comme un **package**. Ce fichier est utilisé pour initialiser un module ou un package, ce qui permet d'organiser le code en plusieurs fichiers et répertoires tout en permettant de les importer facilement dans d'autres parties du projet.

Dans le contexte d'une application **Flask**, le fichier \_\_init\_\_.py sert généralement à **initialiser l'application Flask**, à configurer les extensions, et à enregistrer les routes ou "blueprints" (si l'application est modulée).

1. **Importation des dépendances** :
   * On importe les bibliothèques nécessaires, comme Flask, SQLAlchemy (ou d'autres extensions) et la configuration à partir de config.py.
2. **Création de l'application Flask (create\_app)** :
   * La fonction create\_app() est utilisée pour créer et configurer l'instance Flask. Cette fonction permet de générer une application Flask de manière flexible, particulièrement utile si vous avez plusieurs environnements (développement, production, tests).
3. **Chargement de la configuration** :
   * La configuration est chargée à partir du fichier config.py via app.config.from\_object(Config).
4. **Initialisation des extensions** :
   * Les extensions comme SQLAlchemy (ou Flask-Migrate, Flask-Login, etc.) sont initialisées avec db.init\_app(app) pour qu'elles soient prêtes à être utilisées avec l'application.
5. **Enregistrement des routes ou des blueprints** :
   * Si vous avez des routes définies dans un fichier séparé (comme routes.py), elles sont importées ici et enregistrées dans l'application à l'aide de app.register\_blueprint().

Dans le cadre de notre application, ce fichier se présente comme suit :



**En résumé, l**e fichier \_\_init\_\_.py dans une application Flask sert à **initialiser l'application**, **charger la configuration**, **initialiser les extensions** et **enregistrer les routes ou blueprints**. Il joue un rôle central dans l'organisation du code, surtout pour des applications Flask de taille moyenne à grande.

### **II.2.6. Présentation du fichier routes.py de l’application**

Le fichier **routes.py** dans une application Flask est l'endroit où sont définies les **routes** ou **endpoints** de l'application. Les routes sont les URL ou chemins que les utilisateurs peuvent visiter pour interagir avec l'application. Chaque route est associée à une fonction Python qui est exécutée lorsque cette URL est demandée. Ces fonctions peuvent retourner des pages HTML, des fichiers JSON (dans le cas d'une API), ou rediriger vers d'autres pages.

**Fonctionnement des routes dans Flask**

Flask associe des **routes** à des fonctions à l'aide du **décorateur** @app.route(). Ce décorateur mappe une URL à une fonction spécifique qui gère la requête HTTP reçue pour cette URL.

**Méthodes HTTP (GET, POST, PUT, DELETE)** :

Par défaut, les routes acceptent les requêtes GET, mais Flask permet d'utiliser d'autres méthodes HTTP comme POST, PUT, DELETE, etc., notamment pour gérer des APIs ou des formulaires.

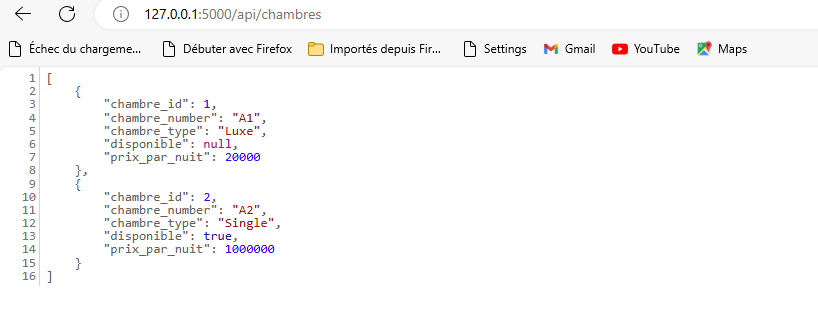
Voici un exemple d’utilisation de route associée à une méthode GET pour récupérer des ressources de la table CHAMBRES de notre base de données :



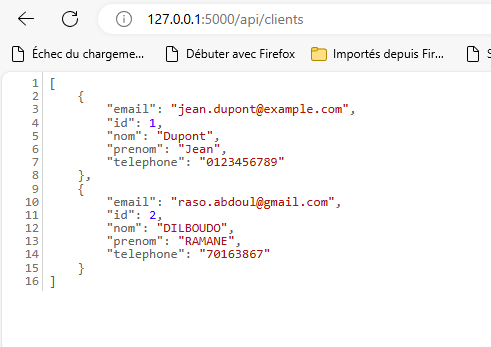


## **II.3. Lancement de l’application de l’application Flask**

### **II.3.1. Exemple d’exposition du contenu de la table CHAMBRES via l’API RESTful**



### **II.3.2. Exemple d’exposition du contenu de la table CLIENTS via l’API RESTful**



### **II.3.3. Exemple d’exposition du contenu de la table RESERVATIONS via l’API RESTful**



# **III. CONFIGURATION NIFI**

## **III.1. Définition et contexte d’utilisation de NIFI**

**Apache NiFi** est un outil open-source pour l'automatisation du flux de données entre systèmes logiciels. Il a été conçu pour gérer le flux de données, y compris leur ingestion, leur transformation, leur routage, et leur stockage. NiFi est particulièrement apprécié pour sa facilité de configuration via une interface graphique et sa capacité à traiter des données en temps réel.

**Principales caractéristiques d'Apache NiFi :**

* **Interface Graphique** : Interface utilisateur web pour concevoir, configurer et gérer les flux de données visuellement.
* **Débit de Données** : Capacité à traiter un grand volume de données à haute vitesse.
* **Flexibilité** : Supporte divers protocoles et formats de données, comme HTTP, FTP, SFTP, Kafka, JSON, XML, CSV, etc.
* **Facilité d'Intégration** : Connexion facile avec des systèmes de stockage de données, bases de données, systèmes de messagerie, et plus.
* **Gestion des Flux** : Gestion et routage des flux de données en temps réel, avec des possibilités de transformation et de filtrage.
* **Résilience** : Fonctionnalités de gestion des erreurs, reprise après sinistre et stockage des données intermédiaires.
* **Sécurité** : Support pour l'authentification, l'autorisation, et le chiffrement des données.

**Contexte d'Utilisation d'Apache NiFi**

1. **Intégration et Orchestration des Données**

* **Scénario** : Une entreprise souhaite intégrer des données provenant de plusieurs sources hétérogènes (bases de données, fichiers, API, etc.) dans un entrepôt de données ou une plateforme de big data.
* **Utilisation** : NiFi est utilisé pour créer des flux de données qui extrairont, transformeront et chargeront les données (ETL) entre les différentes sources et destinations.

2. **Gestion des Flux de Données en Temps Réel**

* **Scénario** : Une application de surveillance nécessite le traitement et l'analyse en temps réel des données générées par des capteurs ou des logs.
* **Utilisation** : NiFi ingère les données en continu, applique des transformations et envoie les résultats à des systèmes de surveillance ou de stockage pour une analyse en temps réel.

3. **Transformation des Données**

* **Scénario** : Les données brutes extraites d'une base de données doivent être nettoyées, enrichies et formatées avant d'être chargées dans un système cible.
* **Utilisation** : NiFi applique des transformations telles que le filtrage, l'enrichissement, ou la conversion de formats (ex. CSV à JSON) sur les données extraites.

4. **Migration de Données**

* **Scénario** : Migration de données d'une ancienne base de données vers une nouvelle base de données ou un nouveau format.
* **Utilisation** : NiFi peut extraire les données de la base source, appliquer des transformations nécessaires, et les charger dans la base cible.

5. **Support des Données Big Data**

* **Scénario** : Une entreprise utilise des plateformes de big data comme Hadoop ou Spark et souhaite alimenter ces plateformes avec des données provenant de diverses sources.
* **Utilisation** : NiFi intègre et prépare les données avant de les envoyer vers les systèmes de big data pour le traitement et l'analyse.

6. **Automatisation des Processus**

* **Scénario** : Les processus manuels d'intégration et de traitement des données doivent être automatisés pour améliorer l'efficacité et réduire les erreurs humaines.
* **Utilisation** : NiFi orchestre les flux de données automatiquement en utilisant des processus définis, ce qui réduit la nécessité d'intervention manuelle.

**Exemples d'Utilisation Pratiques :**

* **Retail** : Intégrer des données de ventes, de stock et de CRM pour fournir des analyses et des rapports en temps réel.
* **Finance** : Traiter les transactions financières en temps réel pour la détection de fraudes et le reporting.
* **Santé** : Intégrer et transformer des données de capteurs médicaux, de dossiers patients et de systèmes de laboratoire.

**En résumé :**

**Apache NiFi** est un outil puissant pour la gestion et l'intégration des flux de données. Sa flexibilité, sa facilité de configuration, et sa capacité à traiter les données en temps réel en font un choix idéal pour divers scénarios d'ETL, de transformation de données, et d'automatisation des processus. Il est utilisé dans des environnements variés pour améliorer l'efficacité des opérations de données et fournir des informations précieuses à partir des données traitées.

## **III.2. Installation de Apache NiFi. et configuration des connexions à la base de données :**

**url d’accès à la connexion NIFI :**[**https://127.0.0.1:8443/nifi/**](https://127.0.0.1:8443/nifi/)

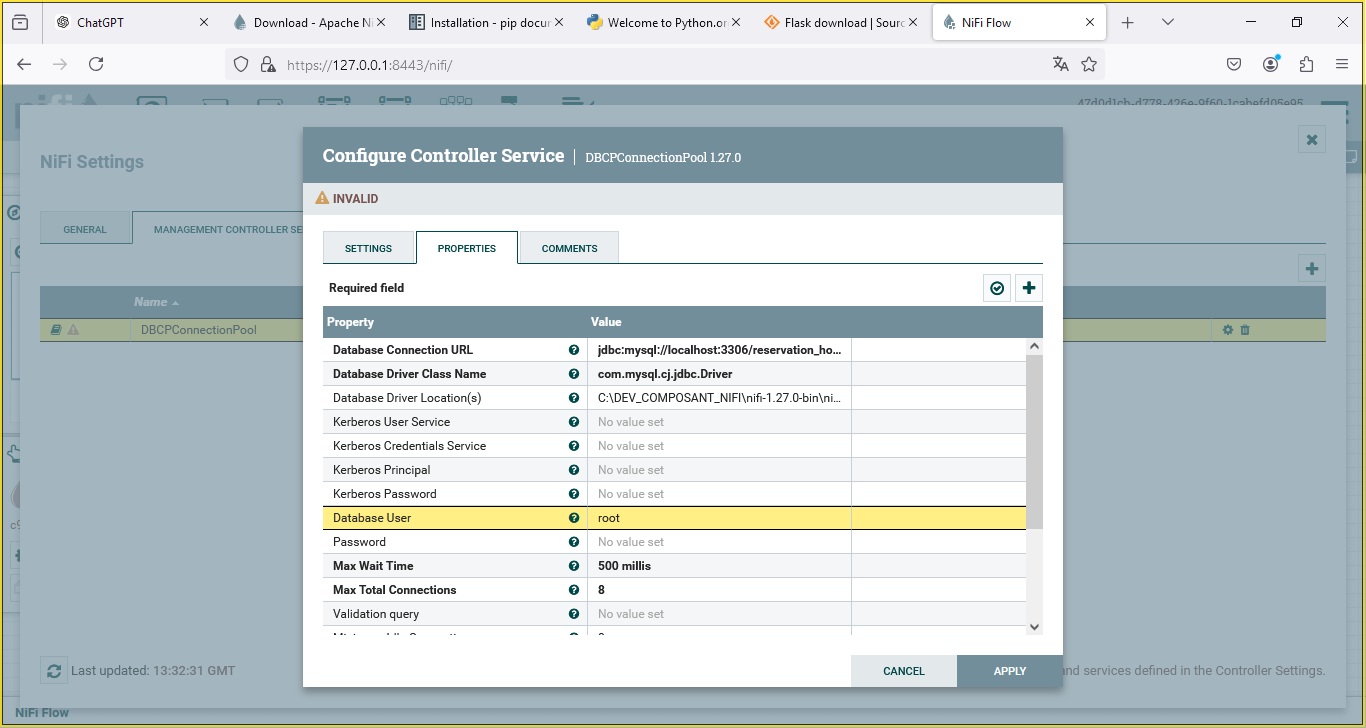
Configuration pour MySQL

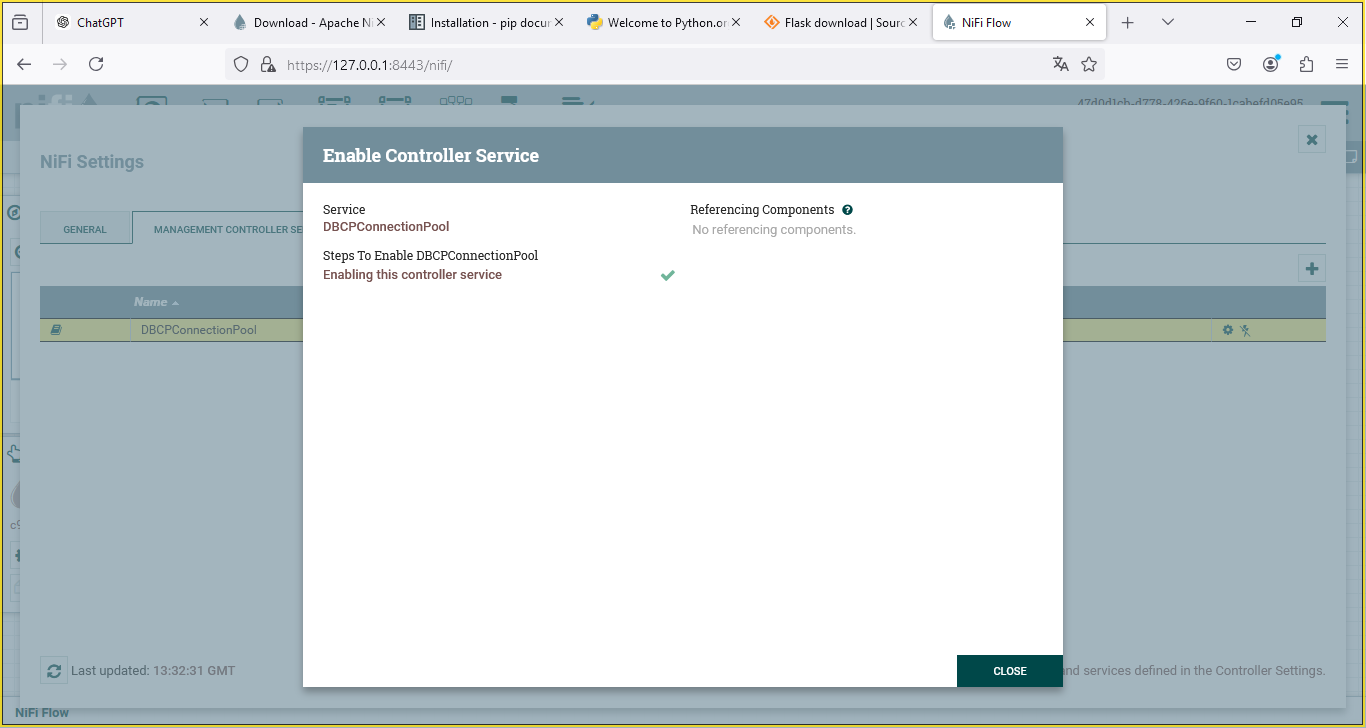
**Database Connection URL: jdbc:mysql://localhost:3306/reservation\_hotel**

**Database Driver Class Name: com.mysql.cj.jdbc.Driver**

**Database User: root**

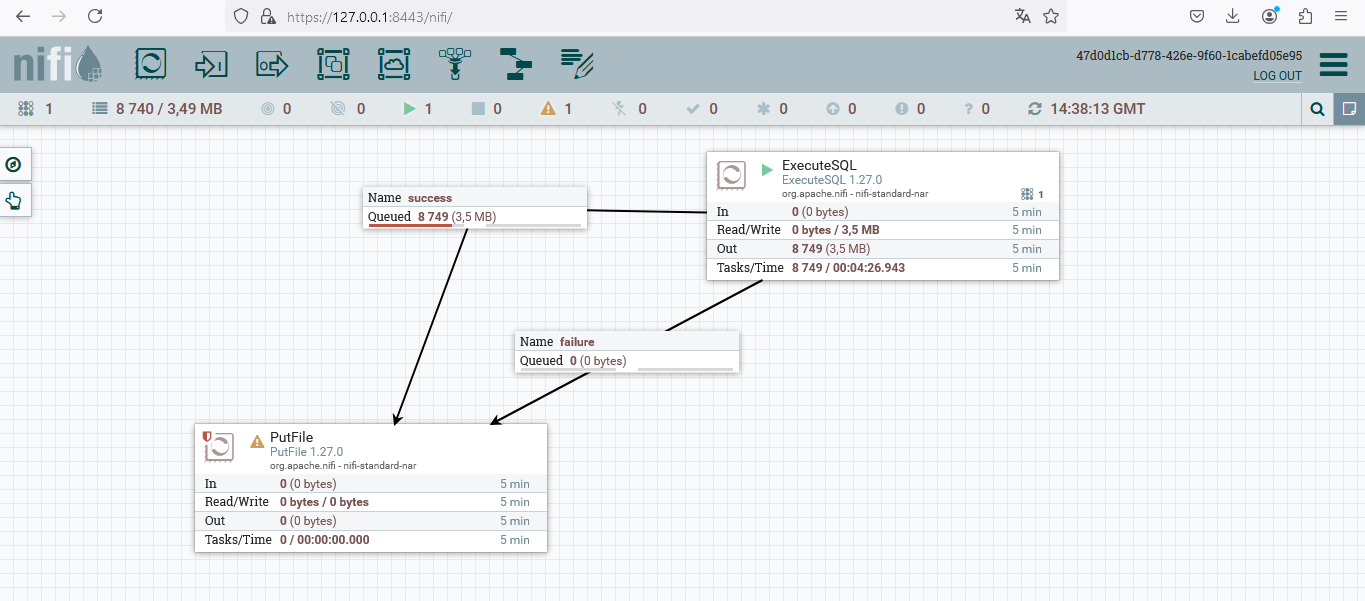
**Password: mon\_mot\_de\_passe**





## **III.3 Conception du flux ETL.**

### **III.3.1 Extraction des Données (à l’aide d’un composant ExecuteSQL ou QueryDatabaseTable :**



### **III.3.2. Transformation : à l’aide des processors comme UpdateRecord ou ConvertRecord.**

### **III.3.3. Chargement : à l’aide de PutDatabaseRecord ou PutFile.**

# **IV. VALIDATION AVEC POSTMAN**

## **IV.1. Définition et contexte d’utilisation de Postman**

**Postman** est un outil populaire pour le développement, le test, et la gestion des API (Application Programming Interfaces). Il fournit une interface utilisateur conviviale pour envoyer des requêtes HTTP/HTTPS et inspecter les réponses, facilitant ainsi le processus de test et de débogage des API. Postman est utilisé par les développeurs et les équipes de test pour vérifier que les API fonctionnent correctement et répondent aux spécifications.

**Principales caractéristiques de Postman :**

* **Interface Graphique** : Permet de construire et d'envoyer des requêtes API via une interface utilisateur intuitive.
* **Gestion des Collections** : Organise les requêtes en collections et dossiers pour une gestion structurée.
* **Tests Automatisés** : Permet d'écrire des scripts de test pour valider les réponses des API et automatiser les tests.
* **Environnements** : Supporte la gestion des environnements pour tester les API dans différents contextes (développement, staging, production).
* **Documentation** : Génère automatiquement de la documentation pour les API à partir des requêtes et des réponses.
* **Collaboration** : Permet de partager des collections et des environnements avec d'autres membres de l'équipe.
* **Mock Servers** : Crée des serveurs simulés pour tester les API avant leur mise en œuvre réelle.

**Contexte d'Utilisation de Postman**

1. **Développement d'API**

* **Scénario** : Un développeur crée une nouvelle API et souhaite tester ses endpoints pendant le développement.
* **Utilisation** : Postman permet de construire des requêtes pour chaque endpoint de l'API, de vérifier que les réponses sont correctes, et de déboguer les problèmes éventuels.

2. **Test des API**

* **Scénario** : Une équipe QA (Quality Assurance) doit tester les fonctionnalités d'une API pour s'assurer qu'elle respecte les spécifications.
* **Utilisation** : Les testeurs utilisent Postman pour envoyer des requêtes API, vérifier les réponses, et exécuter des tests automatisés à l'aide de scripts intégrés.

3. **Validation des Endpoints**

* **Scénario** : Avant de déployer une API en production, il est nécessaire de valider que tous les endpoints fonctionnent comme prévu.
* **Utilisation** : Postman est utilisé pour envoyer des requêtes aux endpoints de l'API et comparer les réponses aux spécifications définies.

4. **Documentation des API**

* **Scénario** : Une équipe développe une API et souhaite fournir une documentation détaillée aux utilisateurs de l'API.
* **Utilisation** : Postman génère automatiquement la documentation à partir des collections de requêtes, fournissant des informations sur les endpoints, les paramètres, et les réponses attendues.

5. **Simulation et Mocking**

* **Scénario** : Les développeurs ont besoin de tester des intégrations avec une API avant que l'API réelle ne soit disponible.
* **Utilisation** : Postman crée des serveurs mock (serveur simulé) pour simuler les réponses de l'API, permettant aux développeurs de tester les intégrations sans avoir besoin de l'API réelle.

6. **Collaboration et Partage**

* **Scénario** : Une équipe de développement travaille sur une API et souhaite partager les requêtes et les tests avec d'autres membres de l'équipe.
* **Utilisation** : Postman permet de partager des collections et des environnements, facilitant la collaboration entre les membres de l'équipe.

7. **Automatisation des Tests**

* **Scénario** : Une API doit être testée régulièrement pour s'assurer qu'elle reste fonctionnelle après des changements ou des mises à jour.
* **Utilisation** : Postman permet d'écrire des scripts de test automatisés pour vérifier les fonctionnalités de l'API et intégrer les tests dans des pipelines d'intégration continue (CI).

**Exemples d'Utilisation Pratiques :**

* **Développement de Microservices** : Tester les interactions entre microservices via des requêtes API pour s'assurer qu'ils fonctionnent correctement ensemble.
* **Intégration de Services** : Valider l'intégration de services tiers en envoyant des requêtes et en analysant les réponses.
* **API RESTful et SOAP** : Tester des API RESTful en envoyant des requêtes GET, POST, PUT, DELETE, et tester des services SOAP avec des requêtes XML.

**En résumé :**

**Postman** est un outil essentiel pour les développeurs et les équipes de test d'API. Il simplifie le développement, le test, la documentation, et la gestion des API grâce à son interface graphique intuitive, ses fonctionnalités de test automatisé, et ses capacités de collaboration. Postman est utilisé dans divers contextes pour garantir que les API sont fonctionnelles, conformes aux spécifications, et bien documentées.

## **IV.2. Scénarios de Tests Définis**

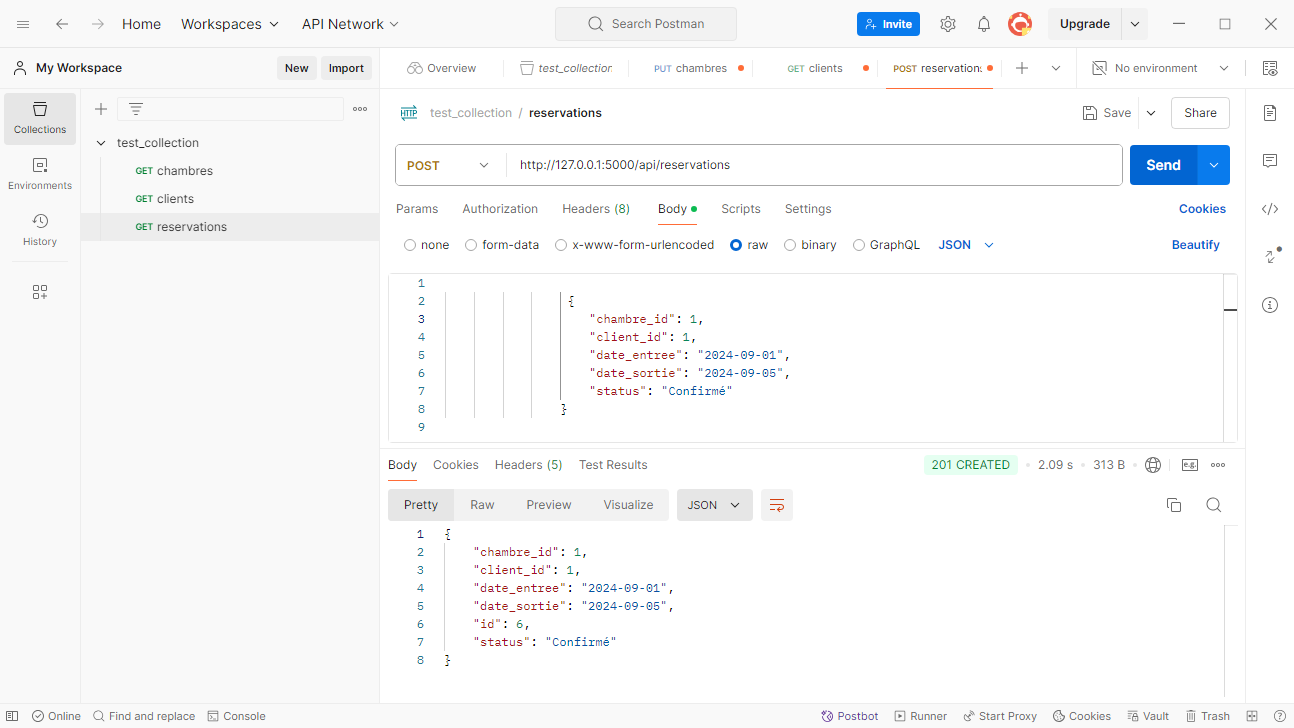
### **IV.2.1. Avantages de l'utilisation de Postman pour les tests :**

1. **Facilité d'utilisation** : Interface graphique simple pour créer des requêtes HTTP et analyser les réponses.
2. **Scripts de tests** : Vous pouvez ajouter des scripts avant ou après la requête pour vérifier les résultats.
3. **Automatisation** : Postman permet d'automatiser les scénarios de tests et de les exécuter régulièrement.
4. **Gestion des environnements** : Postman gère différents environnements (développement, production) avec des variables globales pour les URLs ou les jetons.
5. **Collaboration** : Postman permet de partager des collections de tests entre les membres de l'équipe.

### **IV.2.2.Test de création d'une réservation (POST) avec Postman**

**Scénario : Créer une nouvelle réservation pour un client**

* **But** : Vérifier que l'API accepte et crée correctement une nouvelle réservation.
* **Requête** :
  + Méthode : POST
  + URL : http://127.0.0.1:5000/api/reservations
  + Corps (JSON) **{**
  + **"chambre\_id": 1,**
  + **"client\_id": 1,**
  + **"date\_entree": "2024-09-01",**
  + **"date\_sortie": "2024-09-05",**
  + **"status": "Confirmé"**
  + **}**



# **CONCLUSION**

Les travaux présentés dans ce RAPPORT s’inscrivent dans le cadre de la réalisation concrète d’un projet dans le domaine de Développement à base de composants et services web, plus particulièrement le projet de réalisation d’une Application RESTful avec Flask.

Le sujet concret qui a retenu notre attention traite du ***développement d’une application RESTful en utilisant Flask qui utilise Apache Nifi pour ETL et tests avec Postman.».****,*.***,***

Ce domaine de recherche nous a permis, en tant qu’étudiants de Master 1 de concevoir, de développer et de déployer une application RESTful en utilisant Flask qui utilise Apache Nifi pour ETL et tests avec Postman.

Enfin, ces recherches, tout en inculquant les bases de la programmation avec le framework Flask et de la manipulation d’outils tels que Apache NIFI et Postman, nous donnent aussi beaucoup d’espoir dans la réalisation de futurs projets dans ce domaine…