**UNIVERSITE JOSEPH KI-ZERBO**

**(UJKZ)**

**-=-=-=-=-**

**INSTITUT BURKINABE DES ARTS ET METIERS**

**(IBAM)**

****

**Master informatique**

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

**GROUPE 1**

**EXERCICE 3 :**

**DEVELOPPEMENT D'UNE APPLICATION RESTFUL AVEC FLASK UTILISANT APACHE NIFI POUR GERER LE PROCESSUS ETL**

**‘’SYSTEME DE RESERVATION D'HOTEL’’**

|  |  |
| --- | --- |
| **Membres du groupe** | **Enseignant :** |
| KABORE Mahamady | Monsieur Oumar KY |
| MILLOGO Souleymane |  |
| NIKIEMA W. Jessica |  |
| SAVADOGO Yves |  |
| TAPSOBA Chérif Ramzi Farès |  |

**Année Académique 2023-2024**

[1 ETUDE PREALABLE 5](#_Toc176116244)

[1.1 Contexte et justification 5](#_Toc176116245)

[1.2 Objectifs du Projet 5](#_Toc176116246)

[1.3 Résultats attendus 5](#_Toc176116247)

[1.4 Méthode d’analyse 6](#_Toc176116248)

[1.5 Présentation de groupe de travail 6](#_Toc176116249)

[1.6 Planning des travaux 7](#_Toc176116250)

[1.7 Présentation des technologies utilisées 7](#_Toc176116251)

[1.7.1 Apache NiFi 2.0.0-M4 7](#_Toc176116252)

[1.7.2 Flask 3..0.3 7](#_Toc176116253)

[1.7.3 MySQL community-8.0.39.0 / WampServer 3.3.35 7](#_Toc176116254)

[1.7.4 Postman 11.10.0 7](#_Toc176116255)

[1.7.5 Visual Studio Code 1.92.2 : 7](#_Toc176116256)

[2 Conception et Architecture du Système 7](#_Toc176116257)

[2.1 Les technologies de l’architecture 7](#_Toc176116258)

[2.2 Architecture du système 8](#_Toc176116259)

[3 Configuration et processus ETL avec d’Apache NiFi 9](#_Toc176116260)

[3.1 Présentation d’Apache NiFi et de son Utilisation 9](#_Toc176116261)

[3.2 Installation et configuration d’Apache Nifi 10](#_Toc176116262)

[3.2.1 Installation 10](#_Toc176116263)

[3.2.2 Configuration des Contrôleurs de services (Controller Services) 11](#_Toc176116264)

[3.2.3 Configuration du Flux ETL 13](#_Toc176116265)

[3.2.4 Configuration des processeurs NiFi 14](#_Toc176116266)

[3.2.5 Création de connexions 15](#_Toc176116267)

[3.3 Cas d'utilisation d’extraction des ressources de la base de données 15](#_Toc176116268)

[3.3.1 Décomposition des processeurs du Chemin Vert 18](#_Toc176116269)

[3.3.2 Fonctionnement Global 18](#_Toc176116270)

[3.4 Schéma des Flux NiFi 19](#_Toc176116271)

[4 Développement de l'API RESTful avec Flask 23](#_Toc176116272)

[4.1 Présentation de Flask et de l’API RESTful. 23](#_Toc176116273)

[4.2 Création de l’environnement et installation des dépendances 23](#_Toc176116274)

[4.3 Structure de l’application et organisation des routes 26](#_Toc176116275)

[4.4 Endpoints de l'API 26](#_Toc176116276)

[4.4.1 URL de l’endpoint. 26](#_Toc176116277)

[4.4.2 Méthode HTTP utilisée (GET, POST, PUT, DELETE) et leurs fonctionnalités. 29](#_Toc176116278)

[4.4.3 Méthode http 29](#_Toc176116279)

[4.4.4 Paramètres d'entrée et de sortie. 29](#_Toc176116280)

[4.4.5 Connexion à la base de données 29](#_Toc176116281)

[5 Conception de la Base de Données Mysql 29](#_Toc176116282)

[5.1 Démarrage de la base de données et de l’API RESTful 30](#_Toc176116283)

[6 Tests et documentations avec Postman 31](#_Toc176116284)

[6.1 Présentation de Postman et de son Utilisation 31](#_Toc176116285)

[6.2 Scénarios de Tests Définis 32](#_Toc176116286)

[6.2.1 Requêtes GET pour la récupération des données. 33](#_Toc176116287)

[6.2.2 Requêtes POST pour l’ajout de nouvelles données. 34](#_Toc176116288)

[6.2.3 Requêtes PUT 35](#_Toc176116289)

[6.2.4 Requêtes DELETE 35](#_Toc176116290)

[6.2.5 Validation des erreurs (par exemple, gestion des données manquantes ou incorrectes). 36](#_Toc176116291)

[6.3 Résultats et documentations des Tests 38](#_Toc176116292)

[6.3.1 Captures d’Écran 38](#_Toc176116293)

[6.3.2 Documents des tests 48](#_Toc176116294)

**Introduction**

Dans un monde en constante évolution technologique, la gestion efficace des données est devenue un enjeu majeur pour les entreprises, en particulier dans le secteur de l'hôtellerie. L'accès rapide et précis à l'information est essentiel pour offrir un service de qualité, répondre aux attentes des clients et optimiser les processus internes. C'est dans ce contexte que s'inscrit notre projet : le développement d'une application RESTful avec Flask, utilisant Apache NiFi pour gérer le processus ETL (Extraction, Transformation, Chargement) dans un système de réservation d'hôtel.

Ce projet vise à créer une solution intégrée permettant la gestion fluide des réservations d'hôtel, depuis la collecte des données jusqu'à leur exploitation via une API RESTful. La solution proposée repose sur des technologies modernes et robustes telles qu'Apache NiFi pour l'ETL, Flask pour le développement de l'API, et MySQL pour la gestion de la base de données. Postman sera utilisé pour tester et documenter les différentes fonctionnalités de l'application, garantissant ainsi un produit final de qualité.

Le présent document se structure en plusieurs sections. Après une étude préalable qui contextualise le projet et définit ses objectifs, nous présenterons l'architecture du système et les technologies utilisées. La configuration du processus ETL avec Apache NiFi sera détaillée, suivie du développement de l'API RESTful avec Flask. Enfin, nous conclurons par les tests et la documentation réalisés avec Postman, assurant la validation et la robustesse de notre solution.

# ETUDE PREALABLE

## Contexte et justification

Le secteur de l'hôtellerie est un domaine où la gestion des données joue un rôle crucial. Les systèmes de réservation d'hôtel nécessitent une manipulation efficace des informations client, des disponibilités des chambres, et des transactions financières. La multiplication des sources de données et la nécessité d'une réponse rapide aux demandes des clients rendent indispensable l'automatisation et l'intégration des processus de gestion des données. Le projet que nous proposons répond à ce besoin en développant une solution complète qui intègre l'acquisition, le traitement, et l'exploitation des données via une API RESTful.

## Objectifs du Projet

L’objectif principale du projet est de développer une application RESTful en utilisant Flask qui utilise Apache Nifi pour ETL et tests avec Postman.

De façon spécifique il s’agit de :

* Développer une API RESTful pour la gestion des réservations d'hôtel.
* Implémenter un processus ETL efficace avec Apache NiFi pour assurer une manipulation rapide et précise des données.
* Utiliser MySQL pour la gestion de la base de données de réservation.
* Assurer la robustesse et la fiabilité du système grâce à des tests approfondis avec Postman.

## Résultats attendus

Les résultats attendus incluent la mise en place d'une application capable de :

* Gérer les réservations d'hôtel via des endpoints RESTful.
* Assurer un flux de données efficace entre les sources de données et la base de données.
* Offrir une interface endpoints testée et documentée pour une utilisation aisée.

## Méthode d’analyse

Le processus de développement constitue la démarche fondamentale permettant d’obtenir un produit logiciel dans un délai raisonnable tout en minimisant les ressources.

Dans le cadre de notre étude, nous avons opté pour la méthodologie Agile scrum en raison du temps qui nous ai imparti.

La méthode agile est une approche itérative et incrémentale de gestion de projet, principalement utilisée dans le développement logiciel mais applicable à divers domaines. Elle se caractérise par une flexibilité et une adaptabilité élevée, permettant de répondre efficacement aux changements et aux évolutions des besoins des clients ou des utilisateurs finaux.

## Présentation de groupe de travail

Le groupe de travail représente l’ensemble des personnes nécessaires à la réalisation du projet.

Ainsi, nous avons trois groupes de travail à savoir le groupe projet, le groupe de pilotage et le groupe utilisateur.

* **Groupe de pilotage :** C’est le groupe dirigeant chargé de veiller au bon déroulement du projet. Il est chargé de la planification des dates clés du projet, de l’examen des propositions du groupe de projet et décide des orientations stratégiques.
* **Groupe de projet :** Le groupe de projet est l’ensemble des personnes chargées de réaliser le projet. Il est l’intermédiaire entre le groupe de pilotage et le groupe des utilisateurs et est composé de :
* KABORE Mahamady
* MILLOGO Souleymane
* NIKIEMA W. Jessica
* SAVADOGO Yves
* TAPSOBA Chérif Ramzi Farès
* **Groupe utilisateur :** Ce groupe est constitué de l’ensemble de tous ceux qui vont utiliser le futur système. Il est composé d principalement des employés de la mutuelle de santé.

## Planning des travaux

Pour mener à bien notre étude et respecter les délais, nous avons décomposé le projet en différentes tâches. À cette fin, nous avons employé le diagramme de Gantt, un outil d'ordonnancement et de gestion de projet. Il permet de visualiser, à l'aide d'un graphe, dans le temps, les différentes tâches liées à un projet. Le planning anticipé est synthétisé dans le tableau ci-dessous :

## Présentation des technologies utilisées

## Apache NiFi 2.0.0-M4

Apache NiFi est un outil puissant pour l'automatisation du flux de données entre systèmes. Il sera utilisé pour gérer l'ETL des données de réservation d'hôtel.

## Flask 3..0.3

Flask est un micro-framework Python léger et flexible, idéal pour le développement d'API RESTful.

## MySQL community-8.0.39.0 / WampServer 3.3.35

MySQL est une base de données relationnelle robuste, utilisée pour stocker et gérer les données de réservation.

## Postman 11.10.0

Postman est un outil de test d'API, utilisé pour vérifier la conformité et les performances des Endpoints développés.

## Visual Studio Code 1.92.2 :

Vs Code est un éditeur de code source extrêmement populaire, développé par Microsoft et disponible gratuitement. Il est conçu pour simplifier la vie des développeurs en offrant un environnement de travail flexible et personnalisable.

# Conception et Architecture du Système

## Les technologies de l’architecture

L’architecture du projet repose sur une combinaison de technologies pour créer une solution complète et intégrée pour la gestion des réservations d’hôtel. Elle est structurée autour de trois principaux composants. Elle illustre la manière dont les différents composants interagissent les uns avec les autres. Il inclut :

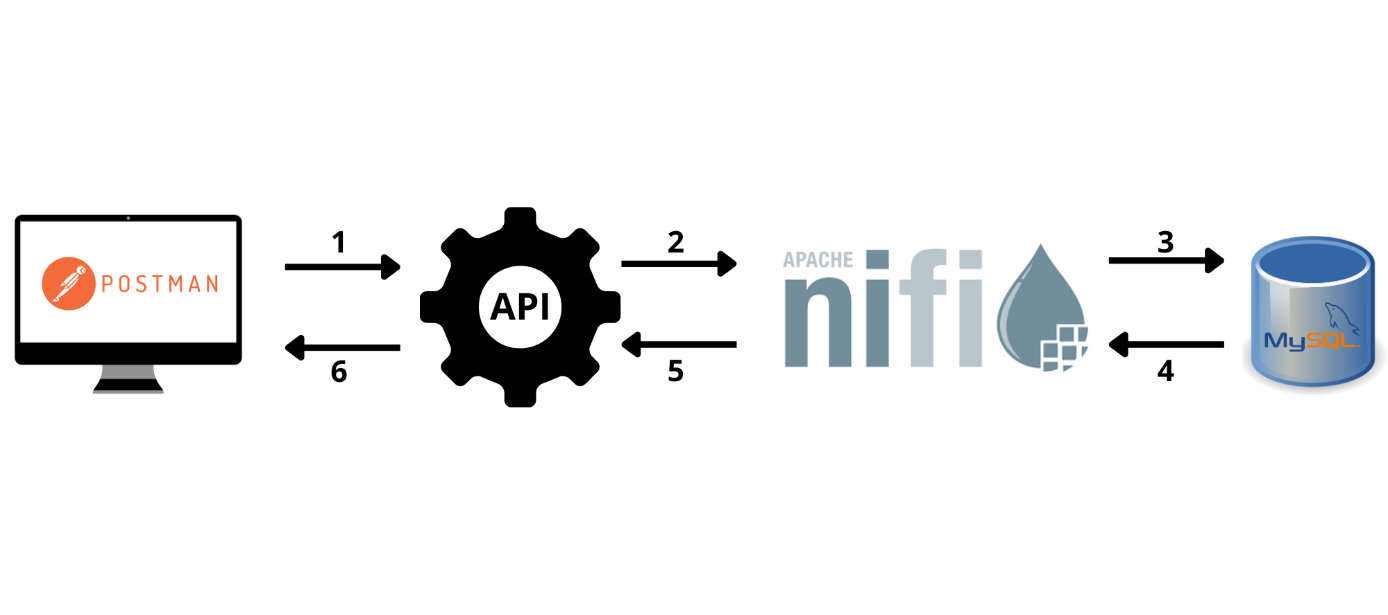
* **Apache NiFi** : utilisé pour le processus ETL (Extraction, Transformation, Chargement) des données de réservation. Il assure la gestion des flux de données en temps réel et facilite leur traitement et leur intégration dans le système. Au cœur du processus ETL, il extrait les données depuis diverses sources, les transforme selon les besoins du système, puis les charge dans la base de données MySQL.
* **Flask** : utilisé pour développer l’API RESTful. Il permet de créer une interface de communication entre le front-end (ou d'autres systèmes) et la base de données, facilitant les opérations de gestion des réservations. **API Flask** repose sur une structure de modèles et de routes, permettant une communication fluide entre l’interface utilisateur ou d’autres systèmes et la base de données.
* **MySQL** : utilisé comme système de gestion de base de données relationnelle, il sert de répertoire centralisé pour toutes les données de réservation, permettant une gestion facile des transactions, des consultations, et des mises à jour des données.
* **Postman** : Utilisé pour tester les endpoints de l’API Flask, en simulant différentes requêtes pour garantir le bon fonctionnement de l'application.

Le système est conçu pour être modulaire et extensible, permettant ainsi l’ajout de nouvelles fonctionnalités ou l’intégration avec d'autres systèmes à l'avenir

## Architecture du système

L’architecture est composée de 3 niveaux a savoir un client qui fournit l’interface Postman, un serveur d’application qui contient la logique métier et un serveur de base de données qui gère la persistance des données.

* Tiers Présentation (client) : Postman
* Tiers logique (serveurs d'applications) Flask et Apache NiFi
* Tiers de données (SGBD) : MySQL



**Figure 1: Architecture du Système**

* Postman envoie une requête HTTP (GET) à l'API Flask pour demander des données
* L'API Flask transmet la requête à Apache NiFi via une requête HTTP
* Apache NiFi exécute une requête SQL sur MySQL pour récupérer les données demandées (ExecuteSQL)
* MySQL renvoie les données extraites à Apache NiFi
* Apache NiFi envoie les données reçues de MySQL à l'API Flask (HandleHttpResponse)
* L'API Flask retourne les données reçues de NiFi à Postman, où elles sont affichées.

# Configuration et processus ETL avec d’Apache NiFi

## Présentation d’Apache NiFi et de son Utilisation

Apache NiFi est une plateforme de gestion de flux de données en temps réel, permettant de déplacer, transformer et orchestrer des données entre différents systèmes. Il offre une interface utilisateur conviviale pour la création et la gestion de flux de données, avec une large gamme de processeurs pour l'ingestion, la transformation, et le routage des données.

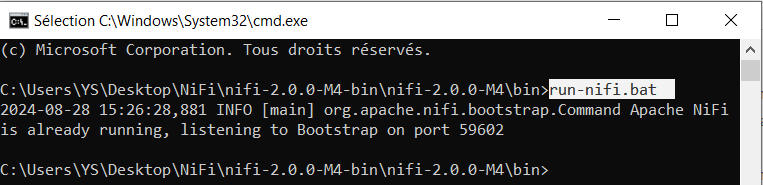
Dans ce projet, NiFi est utilisé pour extraire les données de diverses sources (comme des fichiers CSV, des bases de données externes, ou des API), les transformer pour les adapter aux besoins du système, puis les charger dans la base de données MySQL.

## Installation et configuration d’Apache Nifi

#### Procédure d’installation

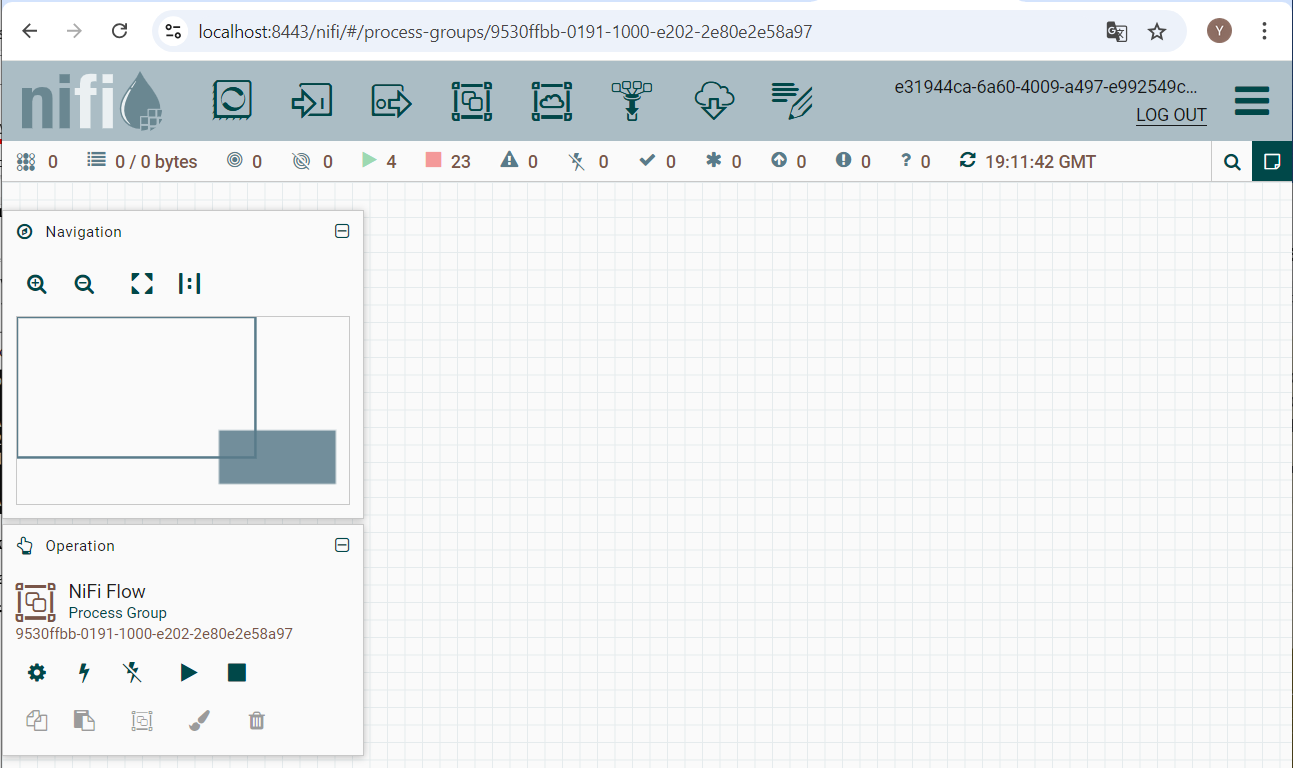
* L'installation d'Apache NiFi se fait en téléchargeant le package correspondant depuis le site officiel et en suivant les instructions d'installation pour Windows. Dans le cadre de notre projet, nous avons téléchargé et installé la version **nifi-2.0.0-M4-bin.** Une fois installé,

Sur Windows, NiFi peut être démarré via la ligne de commande en lançant la commande **run-nifi.bat** après avoir inséré le chemin du répertoire bin de Nifi dans le CMD



**Figure 2: Commande de démarrage d’Apache Nifi**

* Accédez à l'interface utilisateur via un navigateur par l'adresse : **http://localhost:8443/nifi**
* Renseigner les identifiants (User et mot de passe ) dans la fenêtre puis accéder à la fenêtre principale de Nifi



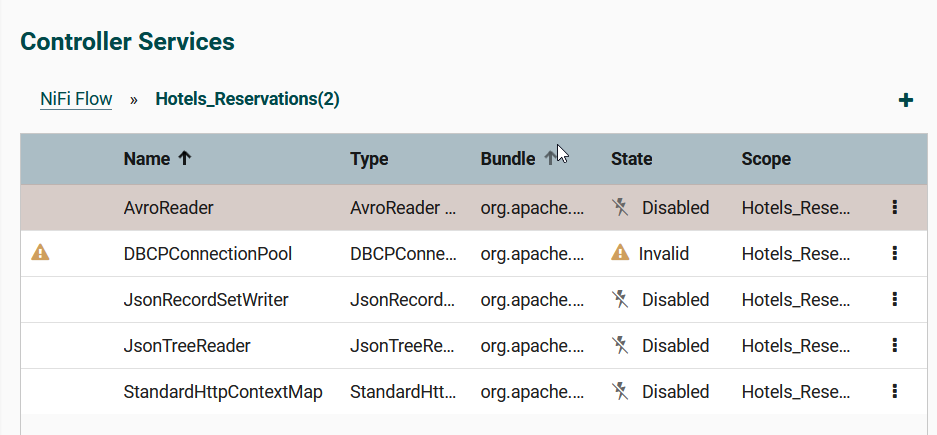
**Figure 3: Interface principale d'Apache Nifi**

Les étapes de configuration incluent :

### Configuration des Contrôleurs de services (Controller Services)

Ces contrôleurs de service sont essentiels pour la création de flux de données robustes dans Apache NiFi, permettant la gestion efficace des données structurées, des connexions à des bases de données, et des interactions HTTP. chaque contrôleur de service dans Apache NiFi est généralement utilisé en conjonction avec un ou plusieurs processeurs spécifiques pour accomplir des tâches dans un flux de données

* **AvroReader:** est un contrôleur de service utilisé pour lire des données en format **Avro**. Il convertit les données Avro en un ensemble de records que NiFi peut traiter dans un flux de données. (Exemple de processeurs utilisés: ConvertRecord, QueryRecord, PartitionRecord )
* **DBCPCConnectionPool :** gère un pool de connexions vers une base de données relationnelle. Il permet aux processeurs NiFi d'accéder à une base de données en partageant un ensemble de connexions ouvertes, ce qui améliore l'efficacité et les performances des opérations de base de données. Utilisé par des processeurs comme PutDatabaseRecord, QueryDatabaseTable, ou ExecuteSQL
* **JsonRecordSetWriter :** utilisé pour écrire un ensemble de records (records set) dans un format **JSON**. Il transforme les données internes NiFi (comme des records) en JSON, qui est ensuite écrit dans un flux de sortie.( Exemple de processeurs utilisés : PutFile, PutKafkaRecord, PutDatabaseRecord)
* **JsonTreeReader :** contrôleur de service qui permet de lire des données structurées au format **JSON** et de les convertir en records. Il traite les fichiers JSON complexes en interprétant leur structure sous forme d'arbre. (Exemple de processeurs utilisés : (Exemple de processeurs utilisés : ConvertRecord,QueryRecord, PartitionRecord)
* **StandardHttpContextMap :** contrôleur de service utilisé pour gérer le contexte HTTP partagé entre différents processeurs NiFi. Il permet de conserver des données contextuelles lors de la gestion de requêtes HTTP, comme les sessions ou les en-têtes. (Exemple de processeurs utilisés : HandleHttpResponse, HandleHttpRequest, InvokeHTTP)



**Figure 4: Les contrôleurs de service en mode désactivé**

Ces services sont par défaut désactivés, ils doivent être démarrés et pour ce faire nous devons les activer à travers les trois boutons verticaux à leur gauche.

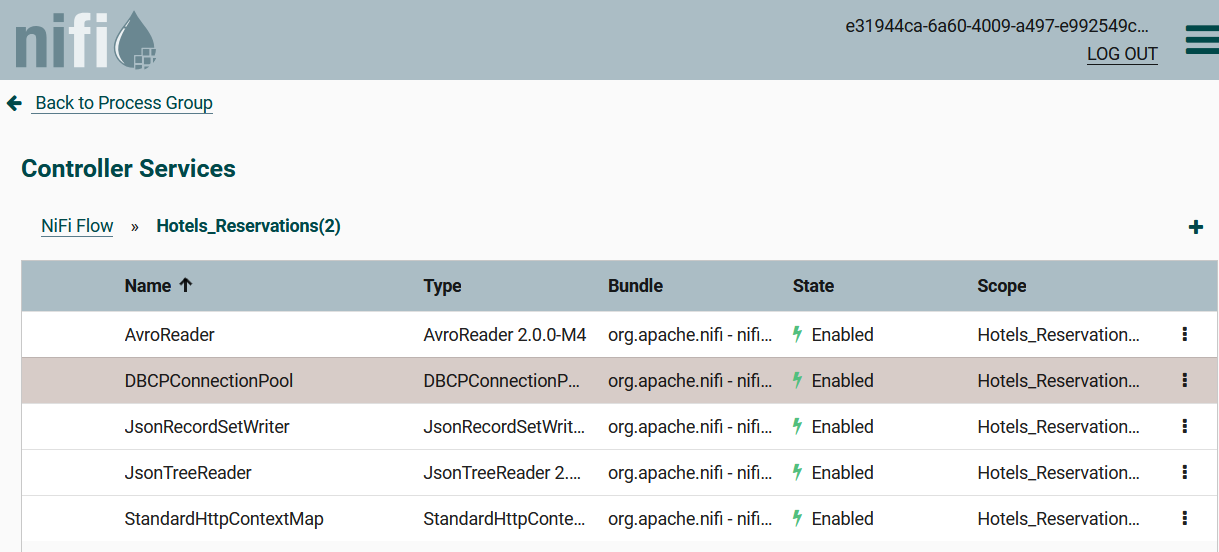
La configuration du contrôleur DBCPCConnectionPool  nécessite le demarrage et de la manière suivante :

* **Database Connection URL:** jdbc:mysql://localhost:3306/dbrestful
* **Database Driver Class Name:** com.mysql.cj.jdbc.Driver
* **Database Driver Location(s) :** C:\Users\YS\Desktop\NiFi\nifi-2.0.0-M4-bin\nifi-2.0.0-M4\lib\mysql-connector-j-9.0.0.jar

(mysql-connector-j-9.0.0.jar a été téléchargé et placé dans le sous dossier lib de dossier nifi telque décrit dans le chemin)

* **Database**: User: root
* **Password ;** (vide)

Après avoir démarrés les controlleurs de services en les activant, on obtient ceci :



**Figure 5: : Les contrôleurs de service en mode activé**

### Configuration du Flux ETL

Le flux ETL (Extraction, Transformation, Chargement) est un processus central dans l'architecture de notre projet, permettant de traiter efficacement les données de réservation. Voici comment ce flux est configuré dans Apache NiFi :

#### Extraction

Les données sont extraites depuis plusieurs sources, telles que des fichiers CSV contenant des informations de réservation ou des bases de données tierces via des connecteurs spécifiques (par exemple, GetFile ou QueryDatabaseTable)

**Exemple :**

* **Processeur GetFile**: Ce processeur permet de lire les données à partir de fichiers situés sur le système de fichiers local ou distant.
* **Processeur InvokeHTTP:** Si les données sont disponibles via une API REST, ce processeur permet de les récupérer.
* **Processeur QueryDatabase:** Pour extraire des données à partir d'une base de données.

#### Transformation

Une fois les données extraites, elles sont transformées pour s'adapter à la structure requise par l'API Flask et la base de données MySQL. Des processeurs comme ConvertRecord ou UpdateRecord sont utilisés pour normaliser les données, gérer les formats de date, ou encore supprimer les doublons.

**Exemple :**

* **Processeur UpdateAttribute**: Ce processeur permet de modifier les attributs des flux de données.
* **Processeur ConvertCharacterSet :** Pour convertir l'encodage des caractères.
* **Processeur ExecuteScript**: Pour effectuer des transformations plus complexes à l'aide de scripts (Jython, Python).
* **Processeur SplitText:** Pour diviser un flux de texte en plusieurs flux.

#### Chargement

Les données transformées sont ensuite chargées dans la base de données MySQL. Ce processus est géré par le processeur PutDatabaseRecord, qui insère les données dans les tables appropriées, garantissant que la base de données est à jour et cohérente avec les informations disponibles.

**Exemple :**

* **Processeur PutDatabaseRecord :** Ce processeur permet d'insérer des données dans une base de données.
* **Processeur PutFile :** Pour écrire les données dans un fichier.

### Configuration des processeurs NiFi

Un **processeur** dans Apache NiFi est une composante fondamentale qui représente une unité de traitement dans un flux de données. Il agit comme un nœud dans un graphe, recevant des données en entrée, les transformant selon sa logique interne, puis les envoyant vers d'autres processeurs.

Les propriétés de chaque processeur sont configurées pour définir son comportement. Il s’agit d’ajuster les paramètres tels que les ports utilisés, et les chemins des répertoires.

* **Glissez-déposez:** Dans la palette des processeurs, trouvez le processeur dont vous avez besoin (par exemple, GetFile pour lire un fichier) et glissez-le sur la toile.
* **Configuration:** Double-cliquez sur le processeur pour ouvrir sa fenêtre de configuration. Configurez les propriétés selon vos besoins (par exemple, le chemin du fichier à lire pour GetFile).
* **Fonctionnement d'un Processeur :** Entrée et Sortie, les processeurs reçoivent des données sous forme de FlowFiles (les unités de données dans NiFi) par le biais de connexions provenant d'autres processeurs. Ils effectuent des opérations sur ces FlowFiles et les envoient ensuite à d'autres processeurs pour un traitement supplémentaire. **La configuration** de chaque processeur se fait en fonction du type de processeur et avec des propriétés spécifiques via son interface utilisateur. Ces propriétés peuvent inclure des chemins de fichiers, des expressions régulières, des paramètres de connexion à des bases de données, etc

### Création de connexions

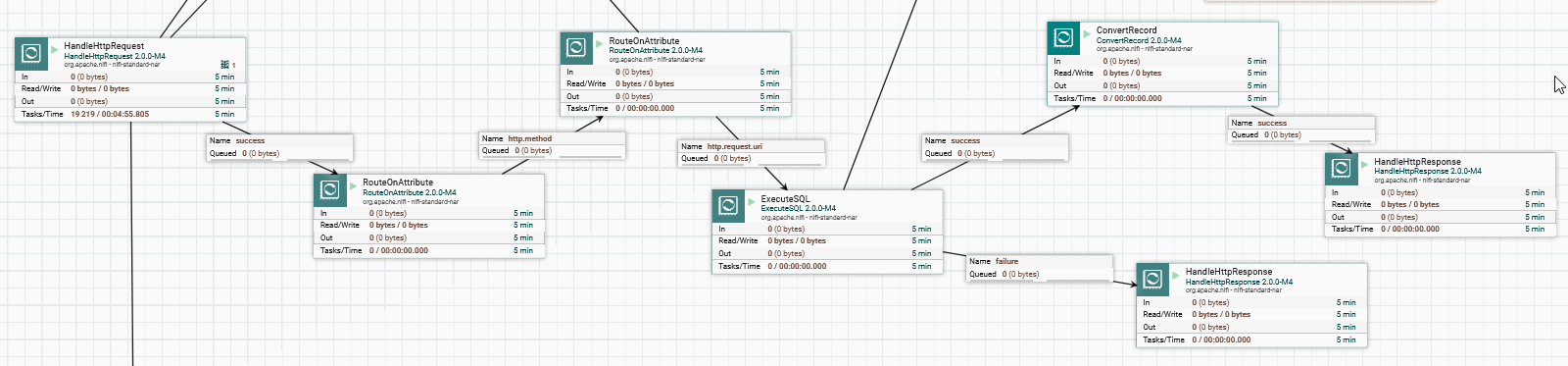
* **Dessinez une ligne:** Cliquez sur la sortie du premier processeur et faites glisser votre souris vers l'entrée du processeur suivant. Une ligne de connexion apparaîtra.
* **Configuration (optionnel):** Double-cliquez sur la connexion pour configurer des propriétés spécifiques, comme la relation entre les données entrantes et sortantes.

#### Création des processus relatives aux réservations

## Cas d'utilisation d’extraction des ressources de la base de données

Notre cas d’utilisation consistera à configurer des processeurs pour gérer des requêtes http GET et récupérer des données de réservation d’hôtel à partir de l’identifiant de réservation de notre base de données MySQL. Le chemin correspondant à ce flux de données fait intervenir les processeurs suivants :

Analyse du Flux NiFi : HandleFttpRequest => RouteOnAttribute => RouteOnAttribute => ExecuteSQL(Success) => ConvertRecord(Success) => HandleHttpResponse comme décrit en couleur verte dans l’image ci-dessous :



**Figure 6:Shema du flux de récupération d'une réservation par ID**

### Décomposition des processeurs du Chemin Vert

* **HandleFttpRequest:** Ce processeur est configuré pour écouter sur le port 8080, indiquant qu'il reçoit des requêtes HTTP.
* **RouteOnAttribute:** Ce processeur filtre les requêtes et achemine le flux en fonction de la méthode HTTP. La configuration http.method=$ http. Methode :equals(GET) garantit que seules les requêtes GET procèdent.
* **RouteOnAttribute:** Ce processeur est un routage secondaire qui filtre davantage les requêtes en fonction du chemin d'URL. La configuration http.request.uri=${http.request.uri:startsWith('/api/reservations/')} garantit que seules les requêtes commençant par /api/reservations/ procèdent.
* **ExecuteSQL:** Ce processeur exécute la requête SQL SELECT \* FROM réservations WHERE id = ${http.request.uri:substringAfter('/api/reservations/')}. Autrement dit, il exécute une requête SQL pour récupérer des données de réservation en utilisant la base de données via DBCPControPool. La requête extrait les données de réservation en fonction de l'ID extrait de l'URL de la requête.
* **ConvertRecord:** Ce processeur convertit le jeu de résultats de la requête SQL en format JSON. Les configurations Record Reader=Avroreader et Record writer=JsonRecordSetWriter suggèrent que les données sont initialement au format Avro et sont converties en JSON pour la réponse HTTP.
* **HandleHttpResponse:** Ce processeur est configuré pour définir le code de statut HTTP à 200 (OK) et le type de contenu à "application/json". Il renvoie ensuite les données formatées en JSON en réponse à la requête HTTP originale.

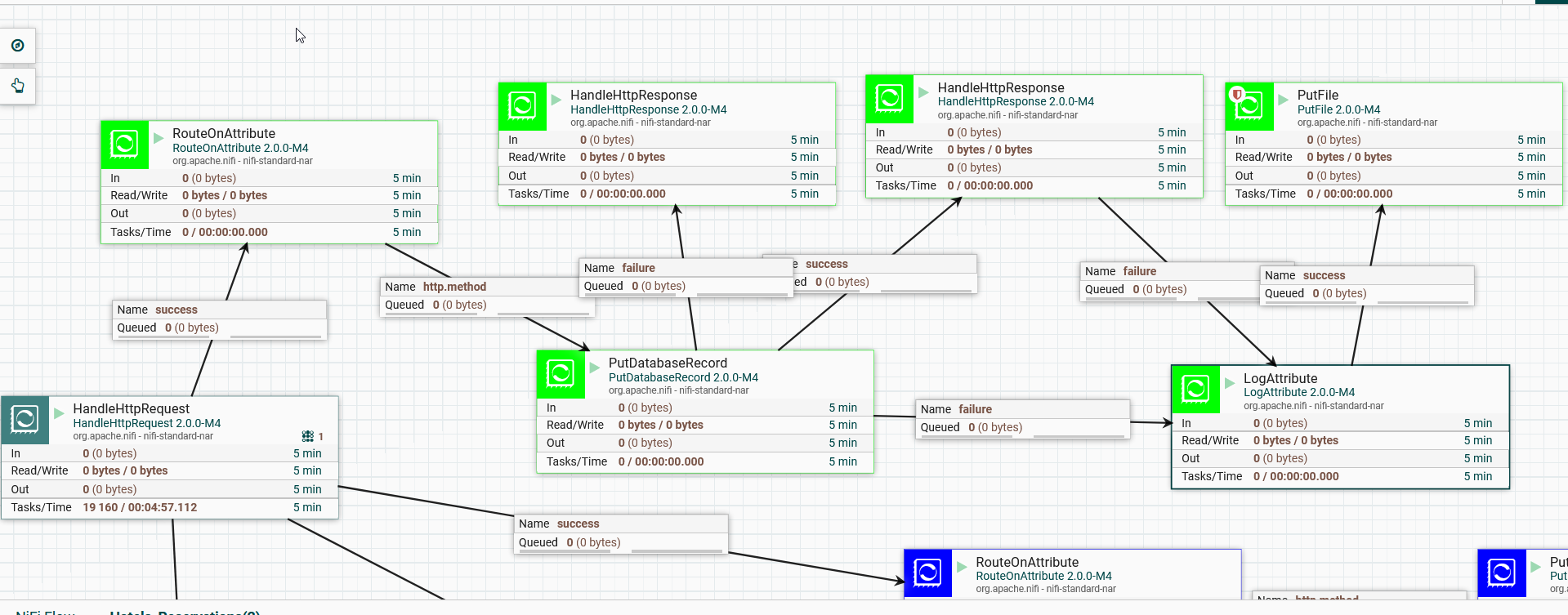
### Fonctionnement Global

* Le flux écoute les requêtes HTTP sur le port 8080.
* Les requêtes GET qui commencent par /api/reservations/ sont acceptées.
* L'ID de la réservation est extrait de l'URI de la requête.
* Une requête SQL est exécutée pour récupérer les données de la réservation.
* Les données sont converties en format JSON.
* Une réponse HTTP est envoyée avec le code 200 et les données JSON.

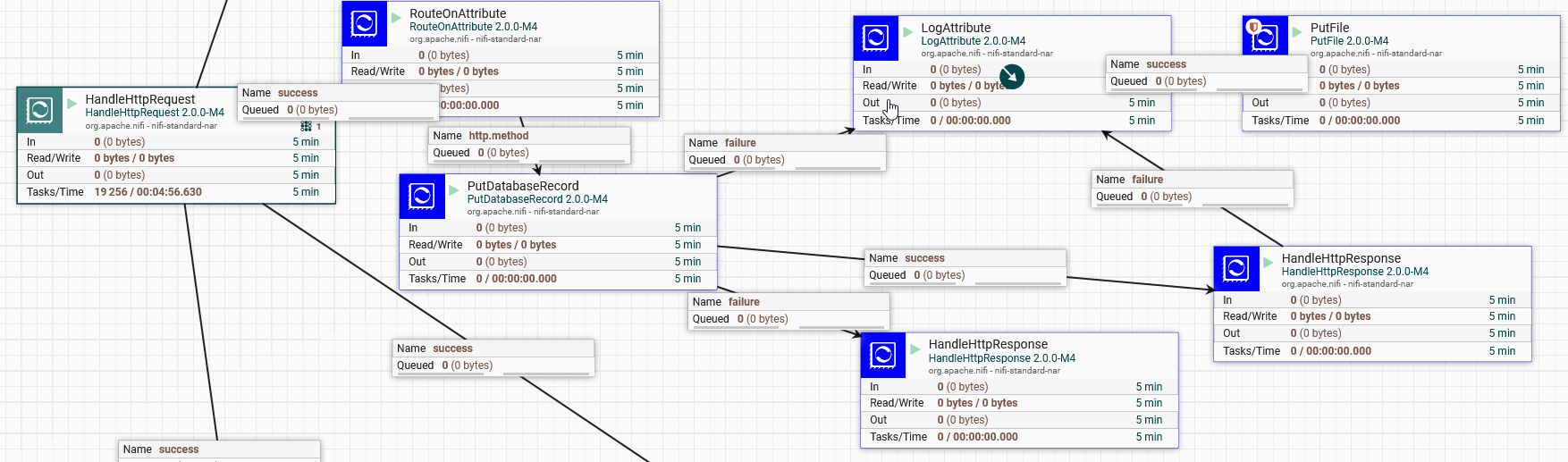
## Schéma des Flux NiFi

Le schéma du flux NiFi représente graphiquement les étapes d'extraction, transformation, et chargement des données. Il montre les différents processeurs utilisés, les connexions entre eux, et les routes de données suivies à travers le flux.

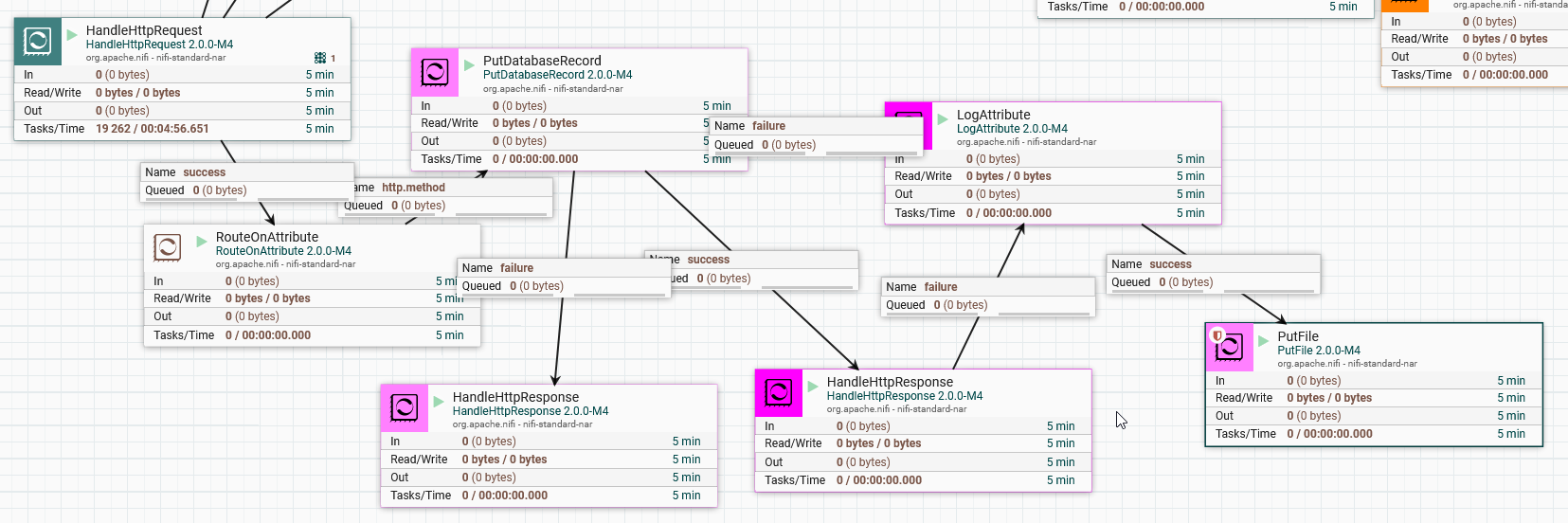
Chaque étape du flux est détaillée, montrant comment les données passent de l'extraction à partir des sources, à leur transformation, puis à leur insertion dans la base de données MySQL. Ce schéma est essentiel pour comprendre l'ensemble du processus ETL et pour identifier les points critiques ou les goulots d'étranglement potentiels.

****

**Figure 7 : Schéma du flux de transformation de réservation**

****

**Figure 8: Schéma du flux de chargement de données**

****

**Figure 9 : Schéma du flux de chargement de données**

# Développement de l'API RESTful avec Flask

## Présentation de Flask et de l’API RESTful.

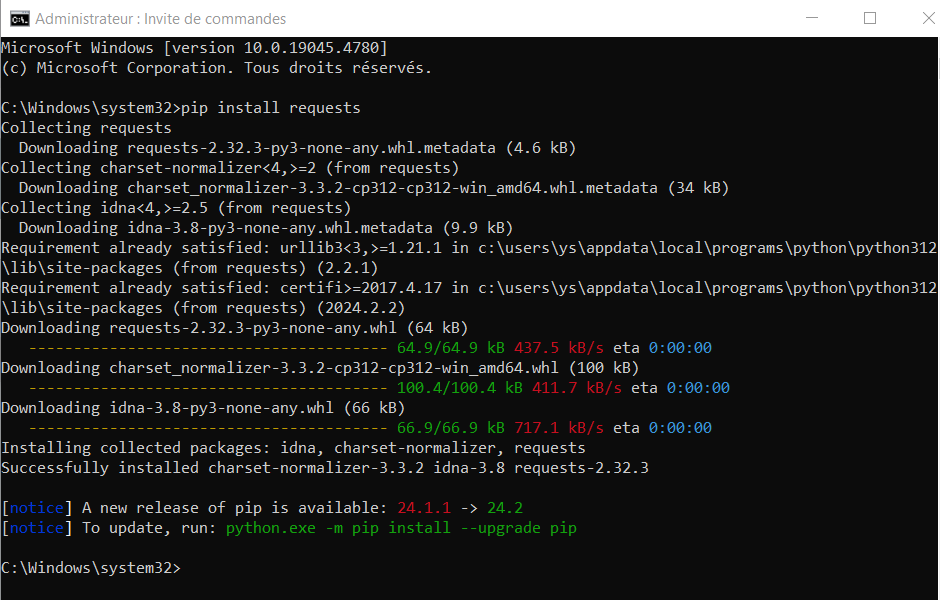
* **Flask** est un microframework Python léger et flexible, idéal pour développer des applications web, notamment des API RESTful. Il permet de créer des routes (endpoints) qui correspondent à des URL spécifiques et à des méthodes HTTP (GET, POST, PUT, DELETE).Dans le cadre de notre projet, nous avons installé la version **Flask 3.0.0**
* **API RESTful** : Une API RESTful suit les principes de l'architecture REST, offrant une interface simple et uniforme pour accéder et manipuler des données. Chaque ressource (ici, une réservation) est identifiée par une URL unique et les interactions avec ces ressources se font à l'aide de verbes HTTP (GET pour récupérer, POST pour créer, PUT pour mettre à jour, DELETE pour supprimer).

## Création de l’environnement et installation des dépendances

* **Procédure d’installation des dépendances**

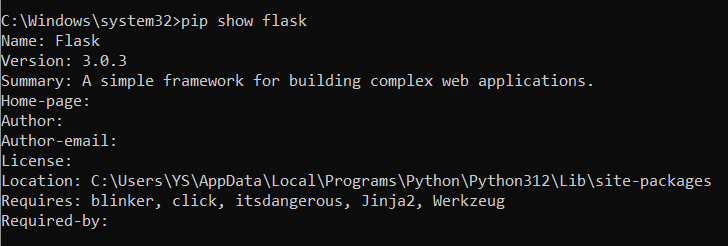
En ligne de commande, en tant qu’administrateur, nous allons tapez la commande **pip install Flask requests** pour l’installation des dépendances.

Les dépendances principales incluent Flask pour le développement web et Requests pour les requêtes HTTP vers Apache NiFi.



**Figure 10 : Installation de dépendance de Flask**

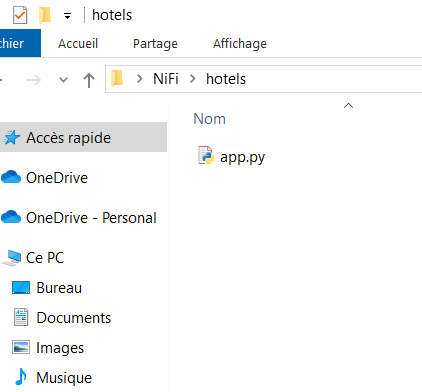
Vérifions qu’il est bien installé : **pip show flask**



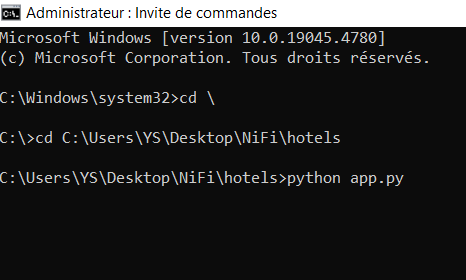
**Figure 11 : Vérification de l'installation de flask**

* **Démarrage de l’API**

L’emplacement du fichier app.py qui servira de lancement en ligne de commande

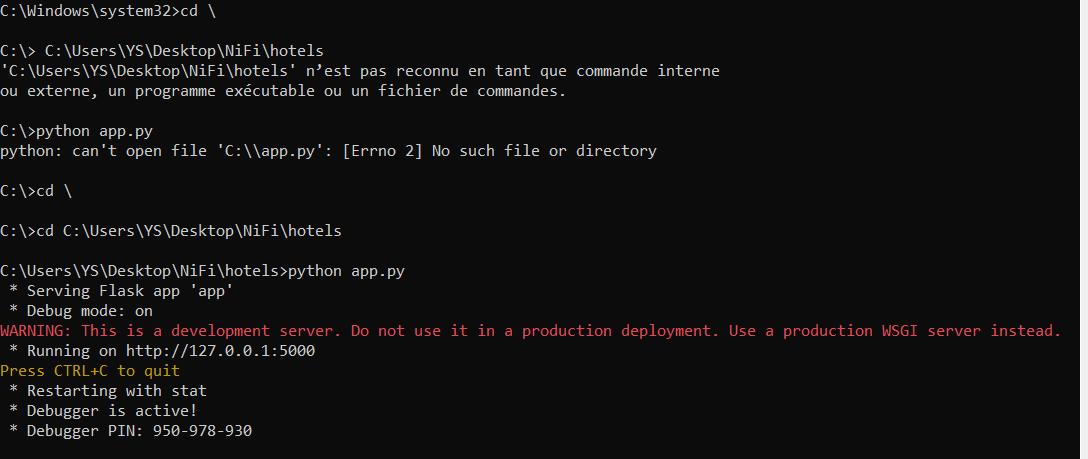


En ligne de commande en tant qu’administrateur, taper les commandes comme suit :



**Figure 12 : Procédure de démarrage de l'API**

Le chemin ‘**’C:\Users\YS\Desktop\NiFi\hotels’’** est l’emplacement du fichier app.py



**Figure 13: Démarrage de l'API**

## Structure de l’application et organisation des routes

L'application est organisée autour de différentes routes, chacune correspondant à une fonctionnalité de l'API RESTful. Les routes sont définies dans le fichier principal de l'application (app.py), où chaque route est associée à une méthode HTTP spécifique pour gérer les opérations CRUD sur les réservations.

Voici la structure des routes :

* /api/reservations : Gestion des réservations (GET pour lister toutes les réservations, POST pour créer une nouvelle réservation).
* /api/reservations/<int:id> : Gestion d'une réservation spécifique par ID (GET, PUT, DELETE).

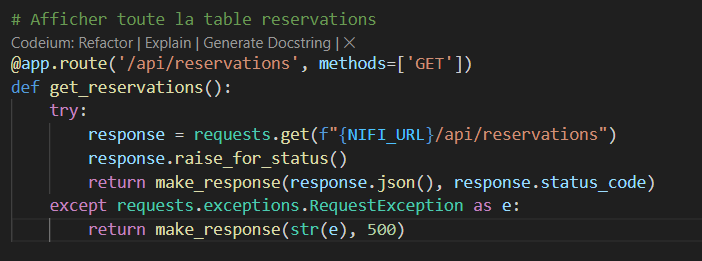
Les modèles de données sont implicitement gérés via les requêtes envoyées à Apache NiFi, qui traite les données selon le modèle de la base de données définie dans votre système.

## Endpoints de l'API

### URL de l’endpoint.

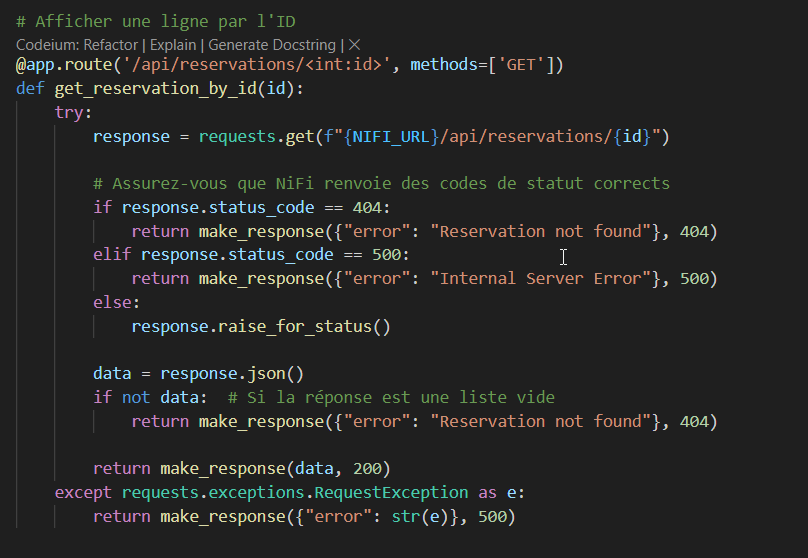
Les endpoints de l'API sont définis comme suit :

* GET /api/reservations : Récupère toutes les réservations.



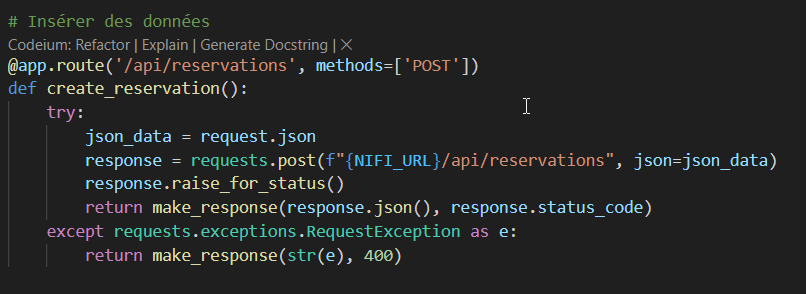
**Figure 14: Script d'affichage de toute la table reservation**

* GET /api/reservations/<int:id> : Récupère une réservation spécifique par son ID.



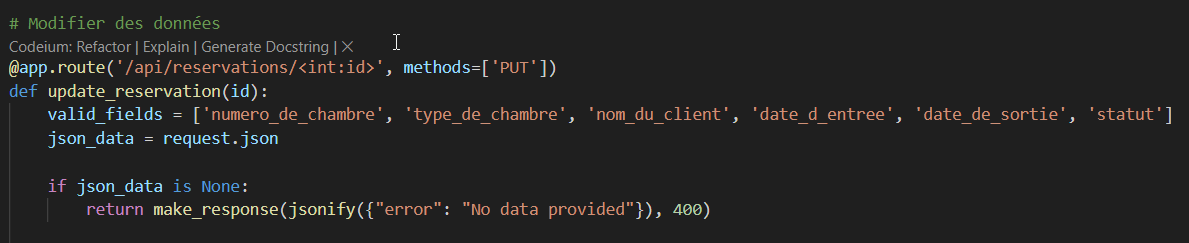
**Figure 15: Script permettant d'afficher une ligne par identifiant**

* POST /api/reservations : Crée une nouvelle réservation.



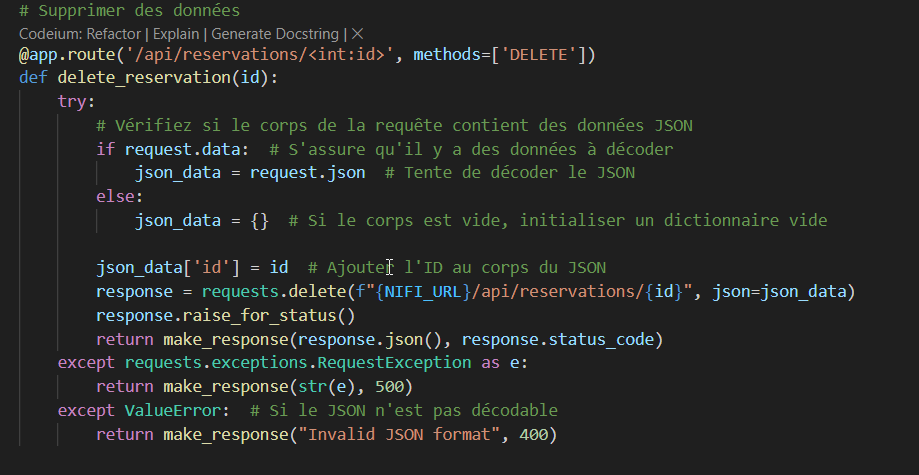
**Figure 16: Script de création d'une nouvelle réservation**

* PUT /api/reservations/<int:id> : Met à jour une réservation existante.



**Figure 17: Script de mise à jour d'une réservation existante**

* DELETE /api/reservations/<int:id> : Supprime une réservation.



**Figure 18: Script de suppression d'une réservation**

### Méthode HTTP utilisée (GET, POST, PUT, DELETE) et leurs fonctionnalités.

### Méthode http

* **GET:** Pour récupérer des données (toutes les réservations ou une réservation spécifique).
* **POST:** Pour créer une nouvelle réservation.
* **PUT:** Pour mettre à jour une réservation existante.
* **DELETE:** Pour supprimer une réservation.

### Paramètres d'entrée et de sortie.

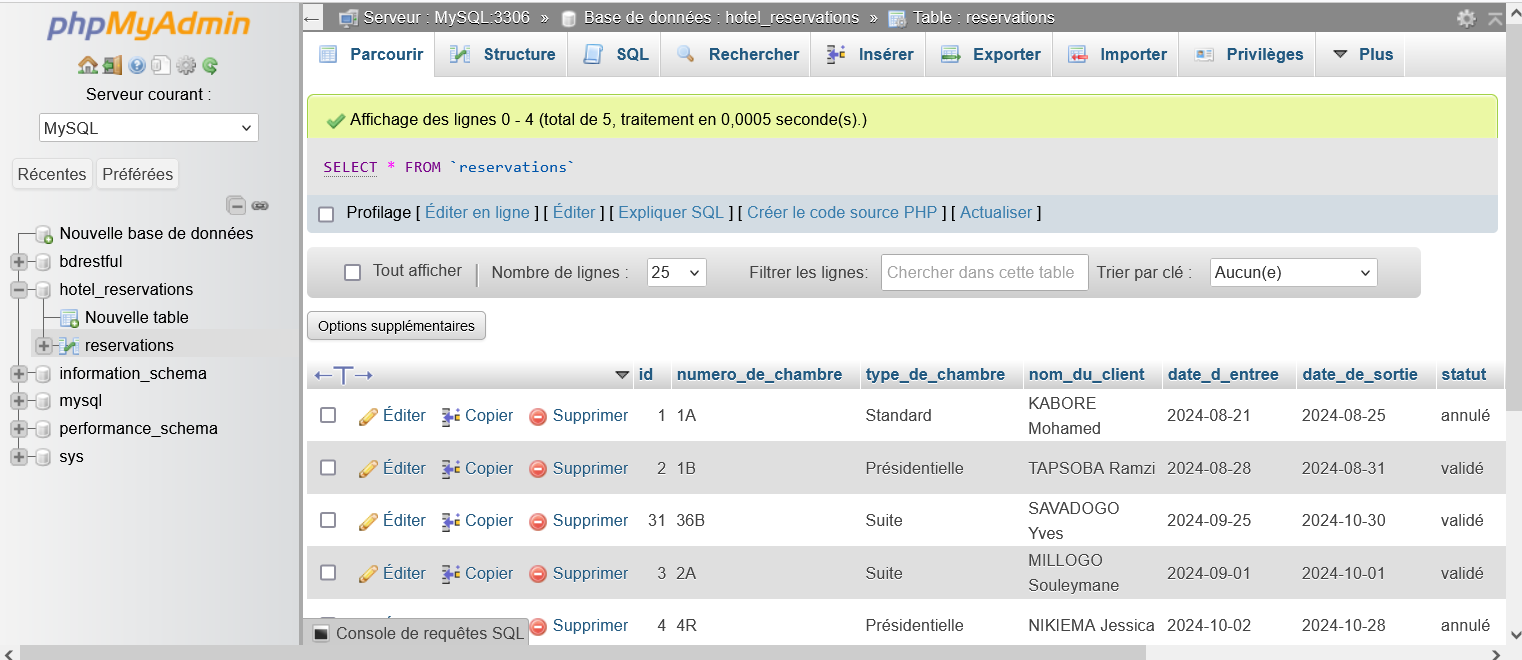
* **GET /api/reservations** : Aucune entrée nécessaire. Renvoie une liste de toutes les réservations.
* **GET /api/reservations/int:id** : ID de la réservation en entrée. Renvoie les détails de la réservation correspondante.
* **POST /api/reservations** : Un JSON contenant les détails de la réservation en entrée. Renvoie la réservation créée avec son ID.
* **PUT /api/reservations/int:id** : Un JSON contenant les détails mis à jour de la réservation en entrée. Renvoie la réservation mise à jour.
* **DELETE /api/reservations/int:id** : ID de la réservation en entrée. Renvoie un message confirmant la suppression.

### Connexion à la base de données

La connexion à la base de données est gérée via Apache NiFi, qui interagit directement avec la base de données MySQL. L'API Flask envoie des requêtes HTTP à NiFi, qui traite ces requêtes et interagit avec la base de données pour exécuter les opérations demandées.

# Conception de la Base de Données Mysql

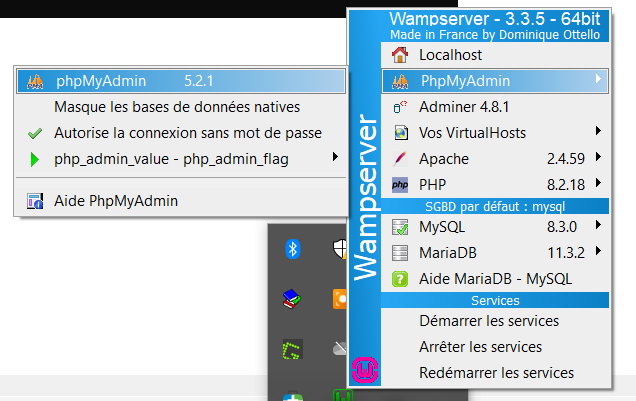
La base de données nommée **hotel\_reservations** sous-jacente pour cette API est modélisée de manière à représenter les réservations d'hôtel. Pour le besoins de notre projet, la base de données comporte une seule table nommée **reservations**. Chaque réservation comprend des informations telles que l'identifiant de réservation, le numéro de chambre, le type de chambre, le nom du client, la date d’entrée, la date de sortie et d'autres détails pertinents. Pour le moment, les détails de la base de données sont gérés via Apache NiFi, qui interagit directement avec les requêtes SQL.



**Figure 19 : la table réservation de la base de données Mysql**

## Démarrage de la base de données et de l’API RESTful

Lancer Wamp64 pour démarrer le serveur Mysql

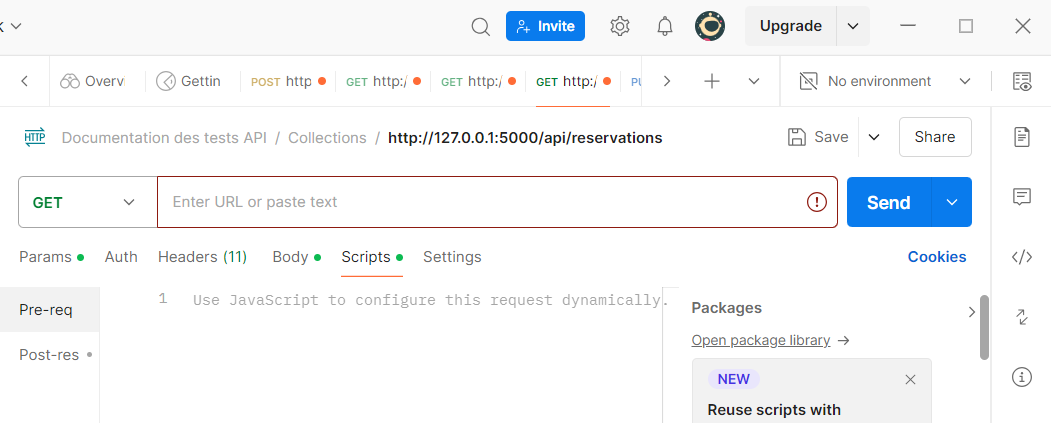


**Figure 20: Démarrage de la base de données**

# Tests et documentations avec Postman

## Présentation de Postman et de son Utilisation

Postman est un outil puissant et populaire pour le développement et le test d'API. La version 11.10.0 de Postman a été utilisé dans le cadre de notre projet permettant ainsi de construire des requêtes HTTP, de les envoyer et d'en examiner les réponses. Pour notre API de réservation, Postman sera utilisé pour :

* **Envoyer des requêtes HTTP** aux différents endpoints de notre API (GET, POST, PUT, DELETE).
* **Vérifier les réponses** de l'API pour s'assurer qu'elles sont correctes et conformes aux attentes.
* **Documenter les tests** pour faciliter la vérification des résultats et la maintenance

**Figure 21 : La barre d'adresse URL des requêtes**

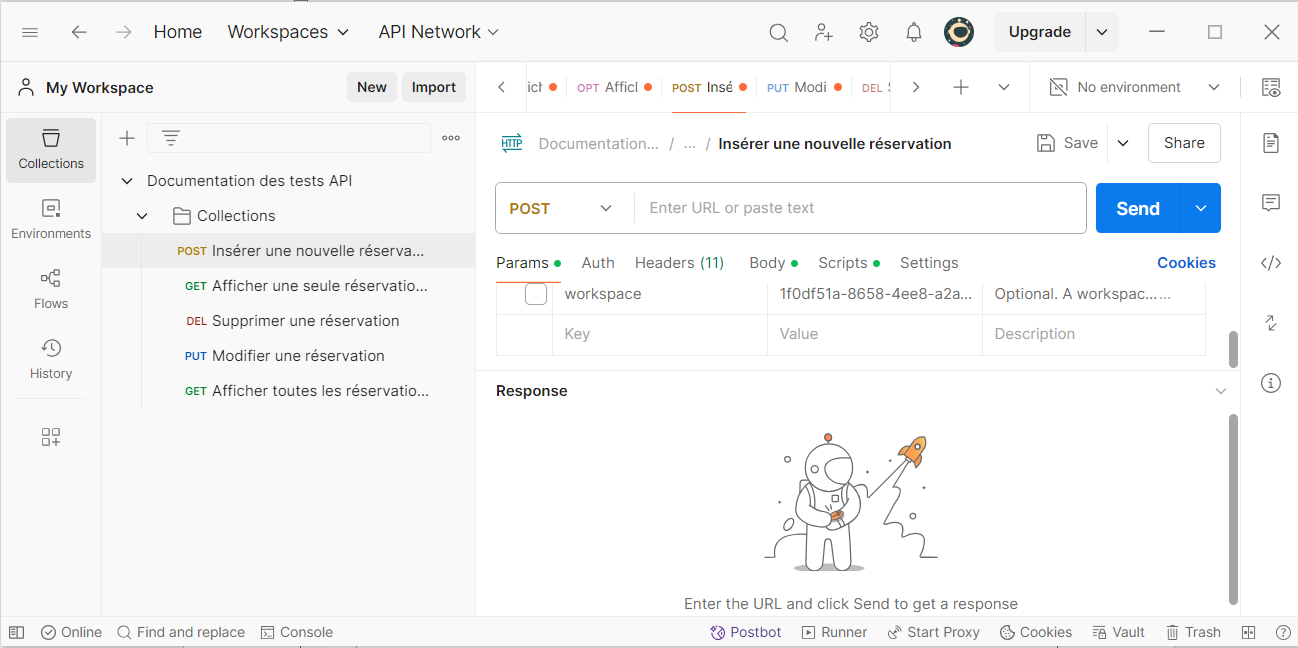
Avant d’utiliser Postman, il faut s’assurer que l’APPi RESTful est démarré et que le serveur Appache nifi est aussi démarré avec tous les processus ;

Dans notre cas, le démarrage des flux de données du schéma réalisé dans Apache nifi est fonction de la méthode (GET, POST, PUT, DELETE…) et de l’URL lancés dans POSTMAN.

Autrement dit, pour démarrer par exemple le processus de récupération d’une réservation par ID comme illustré dans les cas précèdent, il faut sélectionner la methode **GET** puis l’url **’http://127.0.0.1:5000/api/reservations/1’’** dans POSTMAN puis le lancer.

## Procédure d’installation

Après avoir téléchargé Postman à partir du site officiel, lancer le fichier d’installation en suivant la procédure par défaut :



**Figure 22: Fenêtre principale de Postman**

## Scénarios de Tests Définis

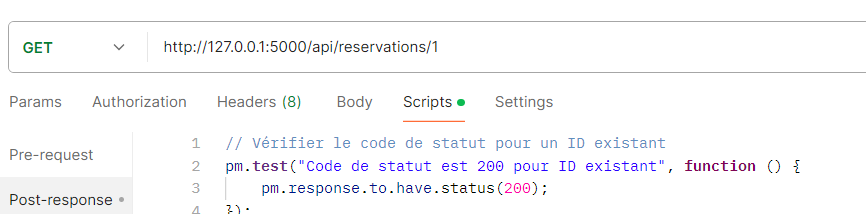
Les tests ont été définis pour couvrir toutes les fonctionnalités de l'API, en simulant des requêtes HTTP typiques envoyées par un client.

### Requêtes GET pour la récupération des données.

**Objectif :** Vérifier que les données de réservation peuvent être récupérées avec succès.

* **Test 1 :** Récupérer toutes les réservations.
  + **URL :** GET /api/reservations **(‘’http://127.0.0.1:5000/api/reservations’’)**
  + **Exécution :** Envoyer une requête GET pour obtenir la liste de toutes les réservations.
  + **Résultat attendu :** La réponse doit contenir un tableau JSON avec toutes les réservations stockées.

**Figure 23 : Requête GET pour la récupération de la table réservations**

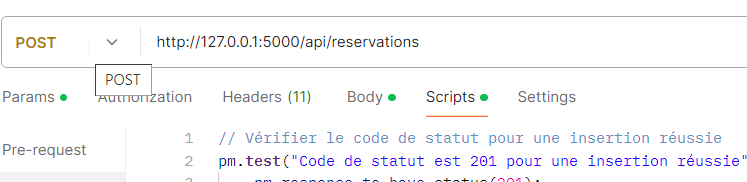
* **Test 2 :** Récupérer une réservation spécifique par ID
  + **URL :** GET /api/reservations/1 **(‘’http://127.0.0.1:5000/api/reservations/1’’)**
  + **Exécution :** Envoyer une requête GET avec un ID spécifique pour obtenir les détails d'une réservation.
  + **Résultat attendu :** La réponse doit contenir un objet JSON avec les détails de la réservation spécifiée par l'ID.

**Figure 24: Requête GET pour la récupération d'une réservation par son identifiant**

### Requêtes POST pour l’ajout de nouvelles données.

**Objectif :** Vérifier que de nouvelles réservations peuvent être créées avec succès.

* **Test 3 :** Créer une nouvelle réservation.
  + **URL :** POST /api/reservations **(‘’http://127.0.0.1:5000/api/reservations’’)**
  + **Exécution :** Envoyer une requête POST avec les données de la nouvelle réservation.
  + **Résultat attendu :** La réponse doit contenir un objet JSON avec les détails de la réservation créée, y compris un ID unique.



**Figure 25 : Requête POST pour l'ajout d'une réservation**

### Requêtes PUT

**Objectif :** Vérifier que les informations d'une réservation existante peuvent être mises à jour.

* **Test 4 :** Mettre à jour une réservation existante.
  + **URL :** PUT /api/reservations/1 (‘’**http://127.0.0.1:5000/api/reservations/2’’)**
  + **Exécution :** Envoyer une requête PUT avec les données mises à jour.
  + **Résultat attendu :** La réponse doit contenir un objet JSON avec les nouvelles informations de la réservation mise à jour.



**Figure 26 : Requête PUT pour la modification d'une réservation existante**

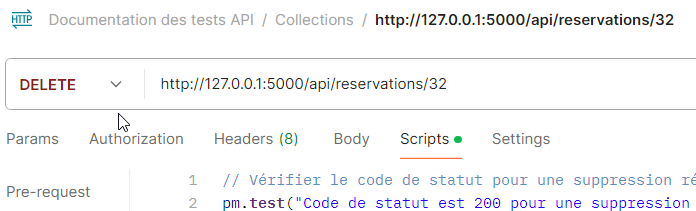
### Requêtes DELETE

**Objectif :** Vérifier que les réservations peuvent être supprimées avec succès.

* **Test 5 :** Supprimer une réservation spécifique.
  + **URL :** DELETE/api/reservations/1 **(‘’http://127.0.0.1 :5000/api/reservation**

**/api/reservation/32’’)**

* + **Exécution :** Envoyer une requête DELETE avec l'ID de la réservation à supprimer.
  + **Résultat attendu :** La réponse doit confirmer que la réservation a été supprimée avec succès.

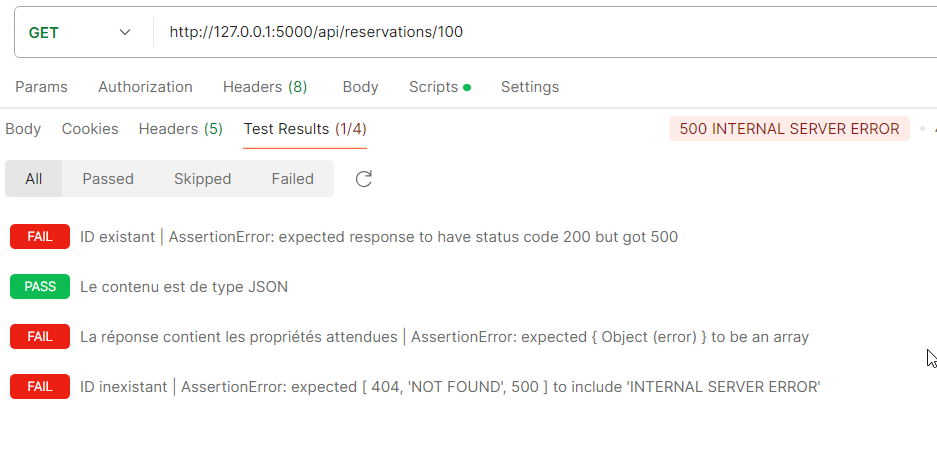


**Figure 27: Requête DELETE pour la suppression de la réservation 32**

### Validation des erreurs (par exemple, gestion des données manquantes ou incorrectes).

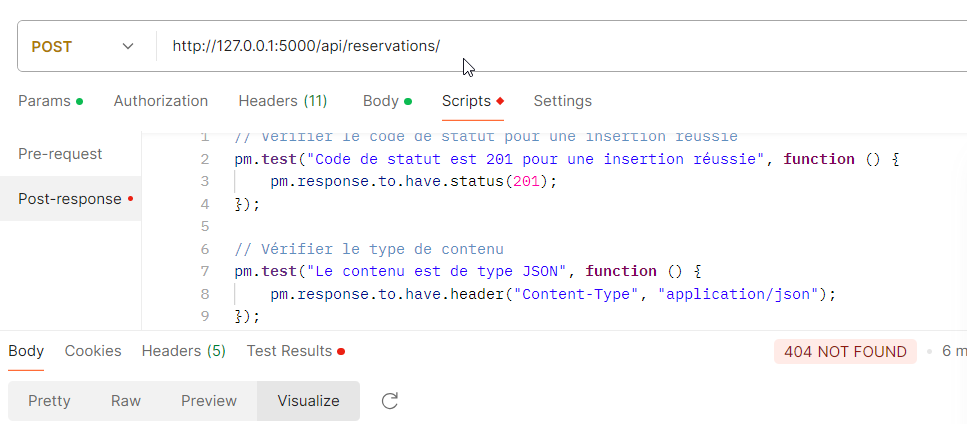
**Objectif :** Vérifier que l'API gère correctement les erreurs, telles que les requêtes mal formées ou les ressources non trouvées.

* **Test 6 :** Requête GET pour une réservation inexistante.
  + **URL :** GET /api/reservations/100
  + **Exécution :** Envoyer une requête GET avec un ID qui n'existe pas dans la base de données.
  + **Résultat attendu :** La réponse doit retourner un code de statut 500 avec un message d'erreur indiquant que la réservation n'a pas été trouvée.



**Figure 28: Validation d'erreur pour un identifiant inexistant**

* **Test 7 :** Requête POST avec des données manquantes.
  + **URL :** POST /api/reservations
  + **Exécution :** Envoyer une requête POST avec des champs obligatoires manquants.
  + **Résultat attendu :** La réponse doit retourner un code de statut 404 avec un message d'erreur indiquant les champs manquants ou invalides.



**Figure 29 : Validation d'erreur pour des données manquantes dans la requête POST**

**Figure 30 : Script de la requête POST**

## Résultats et documentations des Tests

Les résultats des tests ont été documentés pour vérifier le bon fonctionnement de chaque fonctionnalité de l'API. Ces documents permettent de garantir que l'API répond correctement à toutes les situations prévues.

### Captures d’Écran

Des captures d'écran des résultats des tests ont été prises pour chaque scénario, illustrant les requêtes envoyées, les réponses reçues, et les éventuelles erreurs rencontrées. Ces captures sont incluses dans le rapport pour démontrer visuellement le bon fonctionnement de l'API.

* **Requêtes GET pour la récupération des données.**

**Test 1 :** Récupérer toutes les réservations.

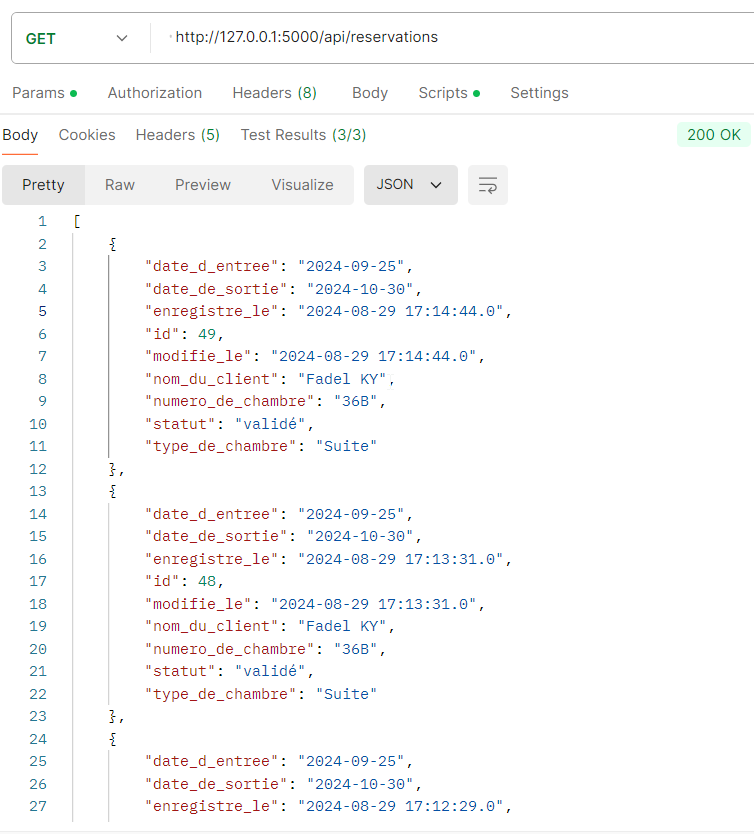
**La requête (URL) :** GET /api/réservations **(‘’http://127.0.0.1:5000/api/reservations’’)**

**Le script de la requête**

****

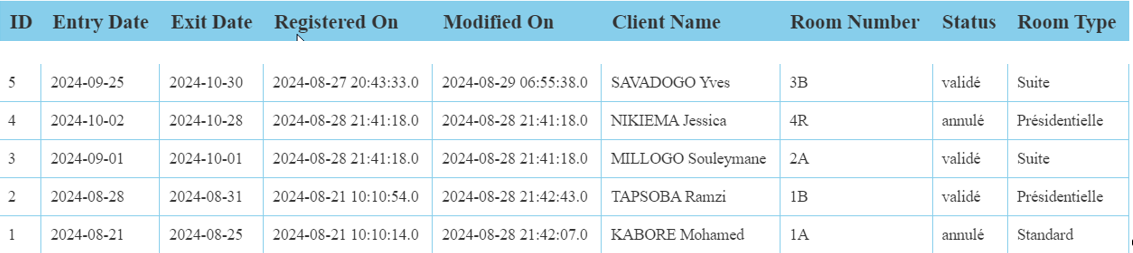
**Figure 31 : Script de la requête GET**

**Le corps de la réponse :**

****

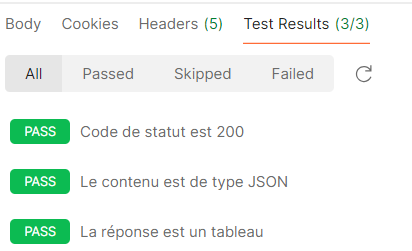
**Figure 32 : Extrait du corps de la réponse**

**Réponse en mode visuel :**



**Figure 33 : Résultat de la requête GET en mode visuel**

**Etat de tous les résultats**

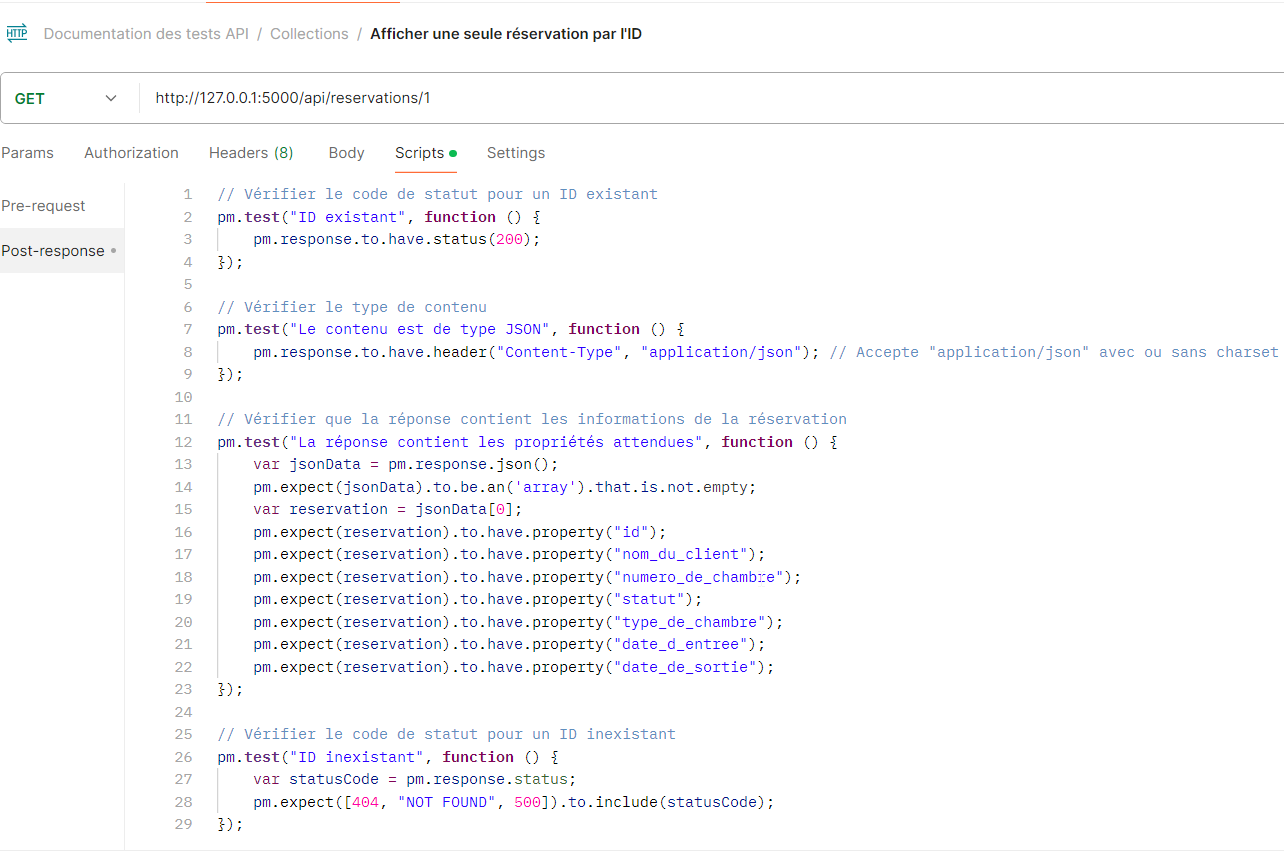


**Figure 34: Résultat de test de la requête GET**

**Test 2 :** Récupérer une réservation spécifique par ID

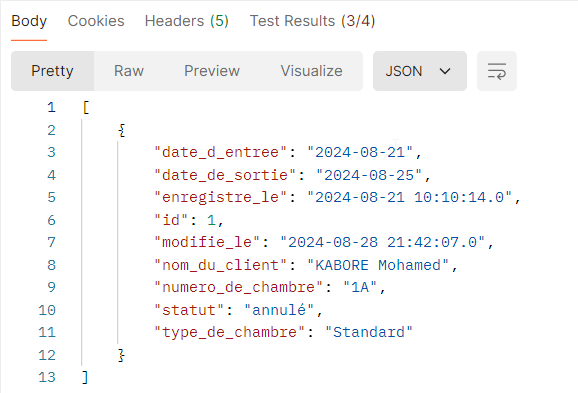
**La requête (URL) :** GET /api/reservations/1 **(‘’http://127.0.0.1:5000/api/reservations/1’’)**

**Le script de la requête**



**Figure 35: Script de la requête GET http://127.0.0.1:5000/api/reservations/1**

**Le corps de la réponse :**

****

**Figure 36: Extrait du corps de la réponse**

**Réponse en mode visuel :**

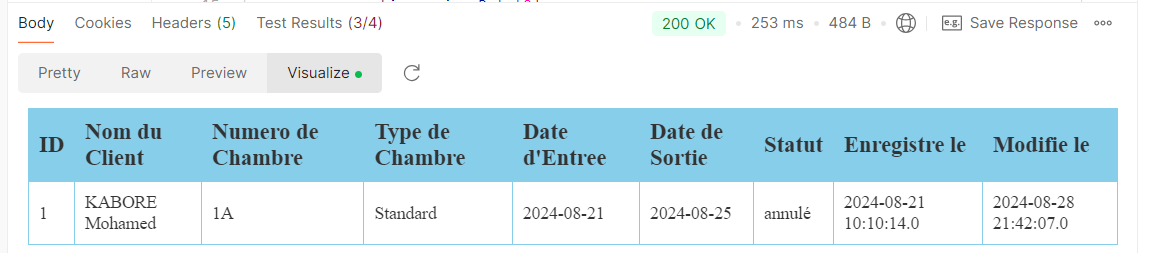
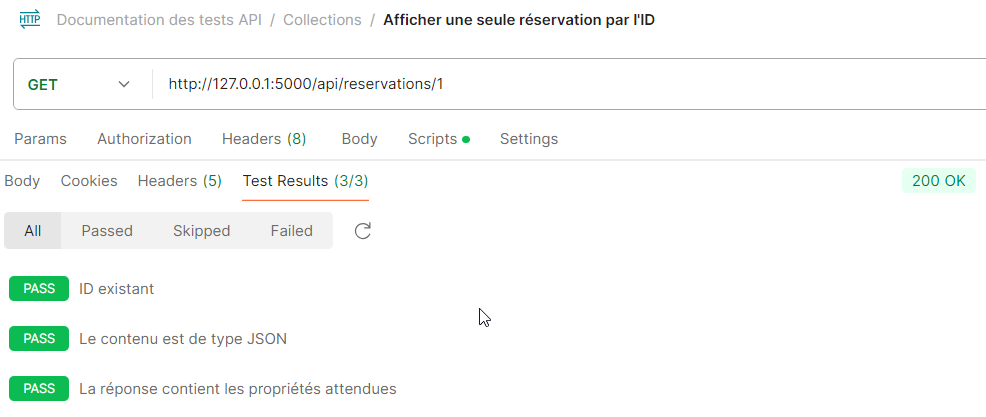
****

Figure 37: Résultat de la requête GET en mode visuel

**Etat de tous les résultats**

****

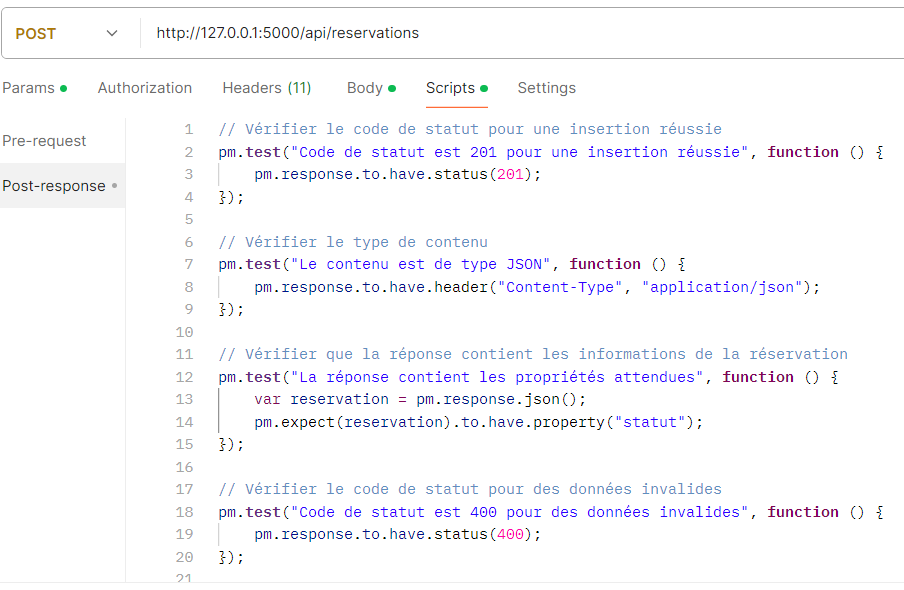
**Figure 38: Résultat de test de la requête GET**

* **Requêtes POST pour l’ajout de nouvelles données**

**Test 3 :** Créer une nouvelle réservation.

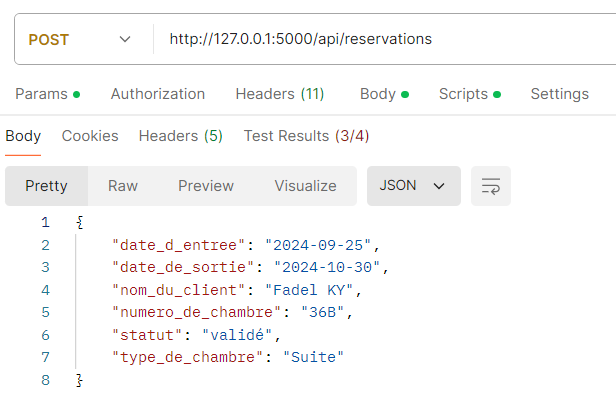
**Requête (URL) :** POST /api/reservations **(‘’http://127.0.0.1:5000/api/reservations’’)**

**Le script de la requête**

****

**Figure 39 : Script de la requête POST**

**Le corps de la réponse :**

****

**Figure 40 : Extrait du corps de la réponse**

**Réponse en mode visuel :**

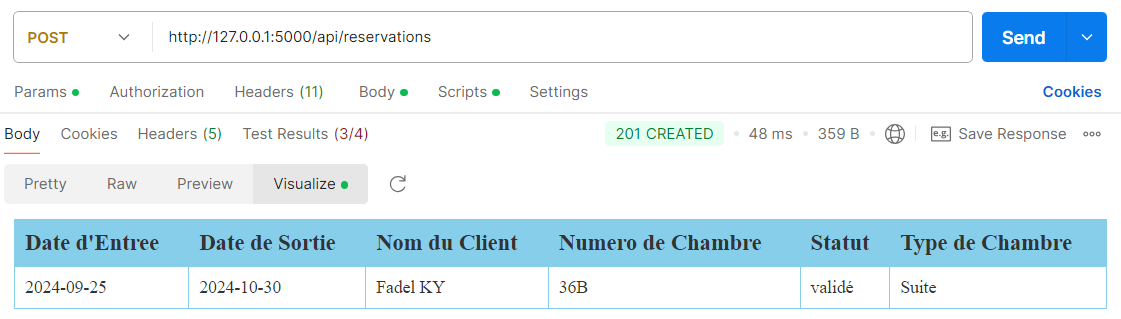
****

Figure 41: Résultat de la requête POST en mode visuel

**Etat de tous les résultats**

****

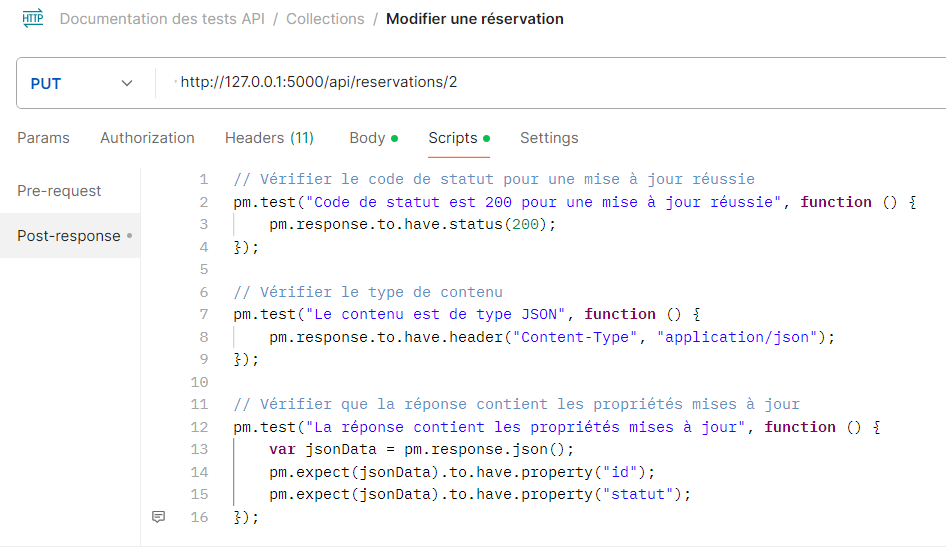
**Figure 42: Résultat de test de la requête POST**

* **Requêtes PUT**

**Test 4 :** Mettre à jour une réservation existante.

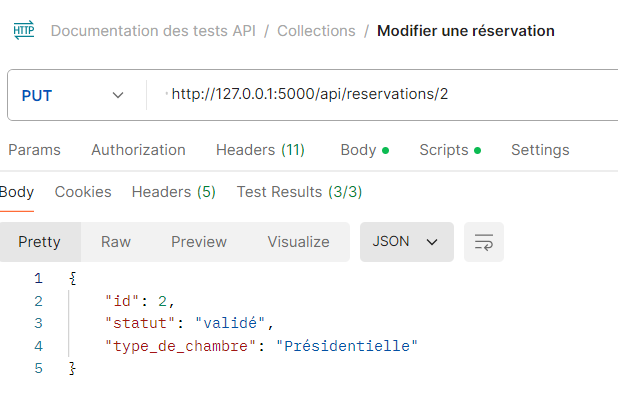
**Requête (URL) :** PUT /api/reservations/1 (‘’**http://127.0.0.1:5000/api/reservations/2’’)**

**Le script de la requête**

****

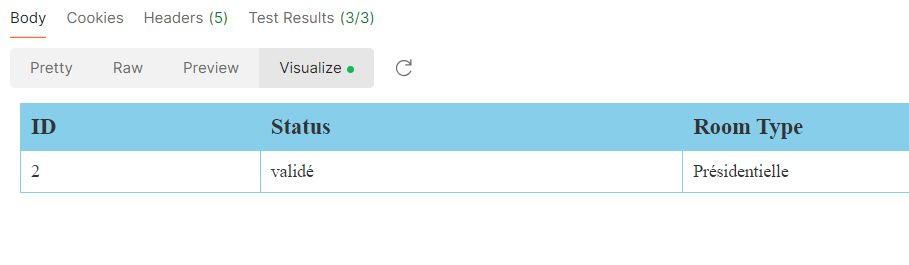
**Figure 43 : Script de la requête PUT**

**Le corps de la réponse :**

****

**Figure 44: Extrait du corps de la réponse**

**Réponse en mode visuel :**

****

**Figure 45: Résultat de la requête PUT en mode visuel**

**Etat de tous les résultats**

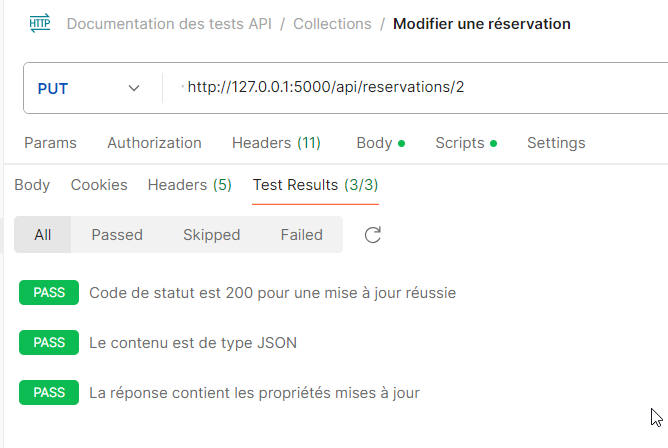
****

Figure 46: Résultat de test de la requête PUT

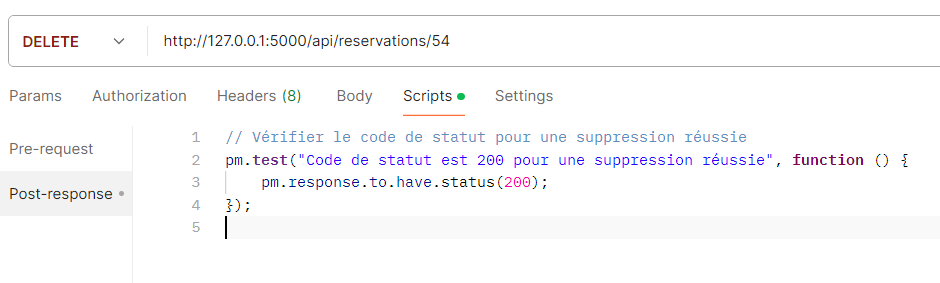
* **Requêtes DELETE**

**Test 5 :** Supprimer une réservation spécifique.

**Réquête (URL :** DELETE /api/reservations/1 **(‘’ http://127.0.0.1:5000/api/reservations/54’’)**

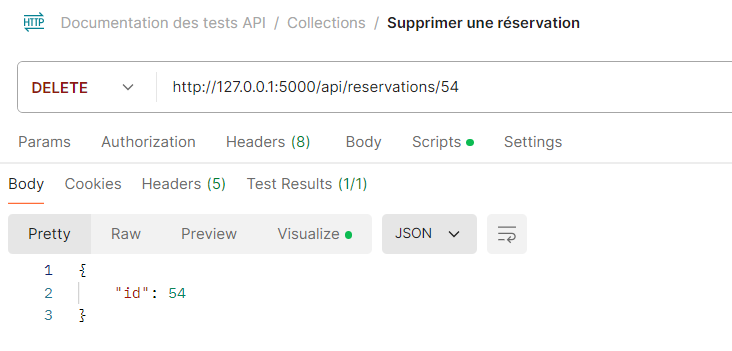
Dans notre cas, nous lançons la requête pour le test de suppression de la réservation 54

**Le script de la requête**

****

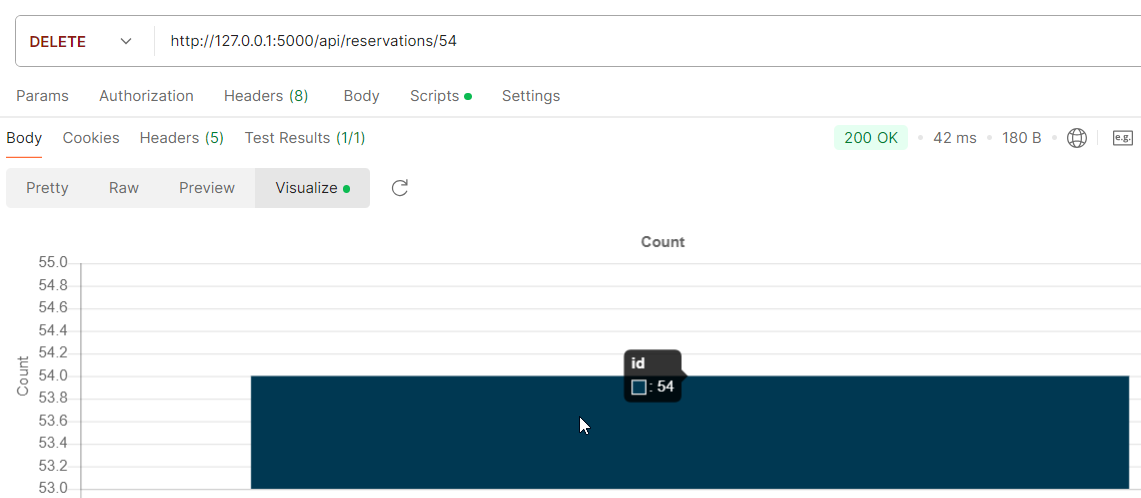
**Figure 47: Script de la requête DELETE**

**Le corps de la réponse :**

****

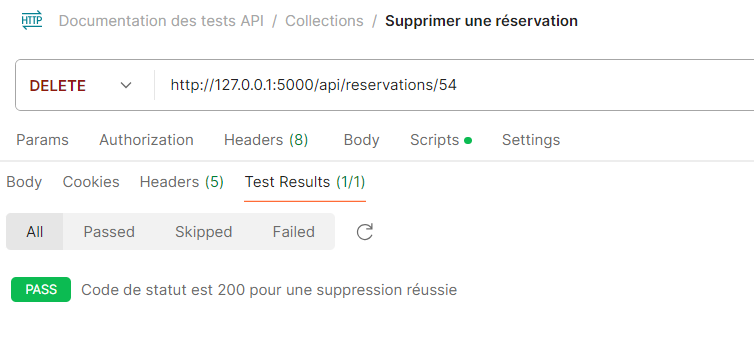
**Figure 48: Extrait du corps de la reponse**

**Réponse en mode visuel :**

****

**Figure 49: Requête DELETE d'une réservation en mode visuel**

**Etat des résultats du test**

****

**Figure 50: Résulte de test de la requête DELETE**

### Documents des tests

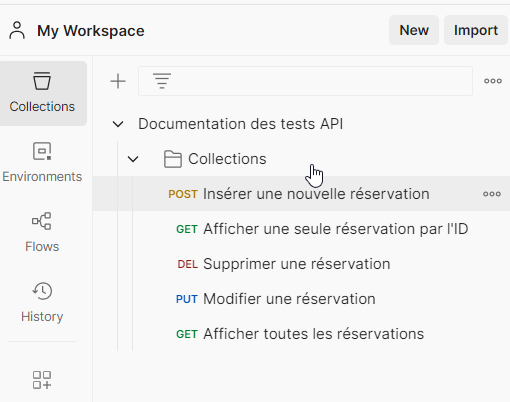
Les résultats des tests ont été documentés pour vérifier le bon fonctionnement de chaque fonctionnalité de l'API. Ces documents permettent de garantir que l'API répond correctement à toutes les situations prévues.

Nous allons générer le document de résultat des tests regroupant l’ensemble des tests (GET, POST, PUT et DELETE) effectués.

Pour se faire, dans le volet gauche de la fenêtre de Postman cliquez sur les trois boutons disposés verticalement à droite **Document des tests API** puis sélectionner **run collections.**

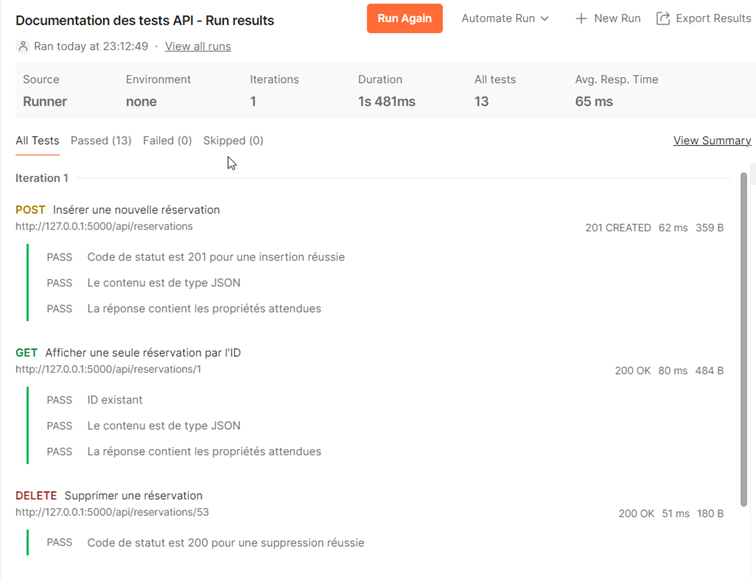
Rassurez vous que dans le volet central **run order,** toutes les methodesconcernées par le test sont cochées, puis lancer à partir de **Run Documentation des tests API**

* **Document de rapport général des tests**

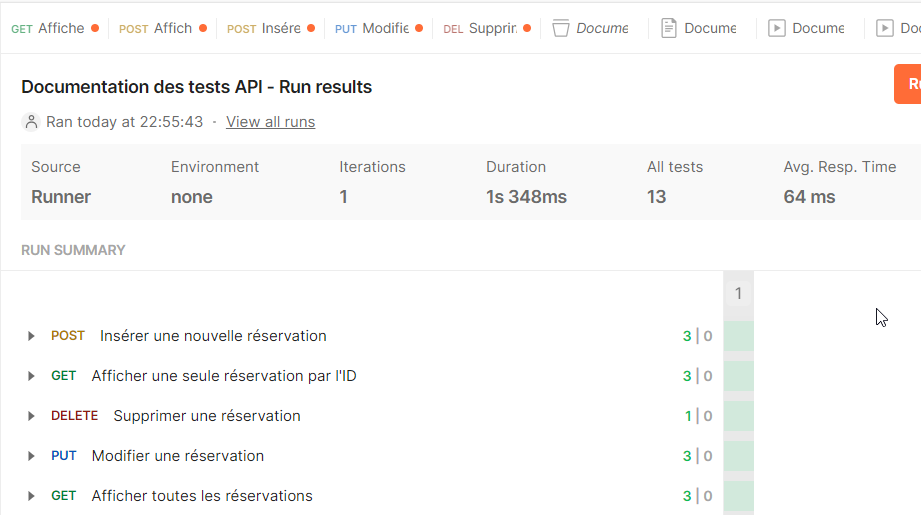
****

**Figure 51: L'ensemble requêtes des tests effectués**

**Figure 52: Document des tests de toutes les requêtes 1/2**



* **Le résumé de document des tests effectués**

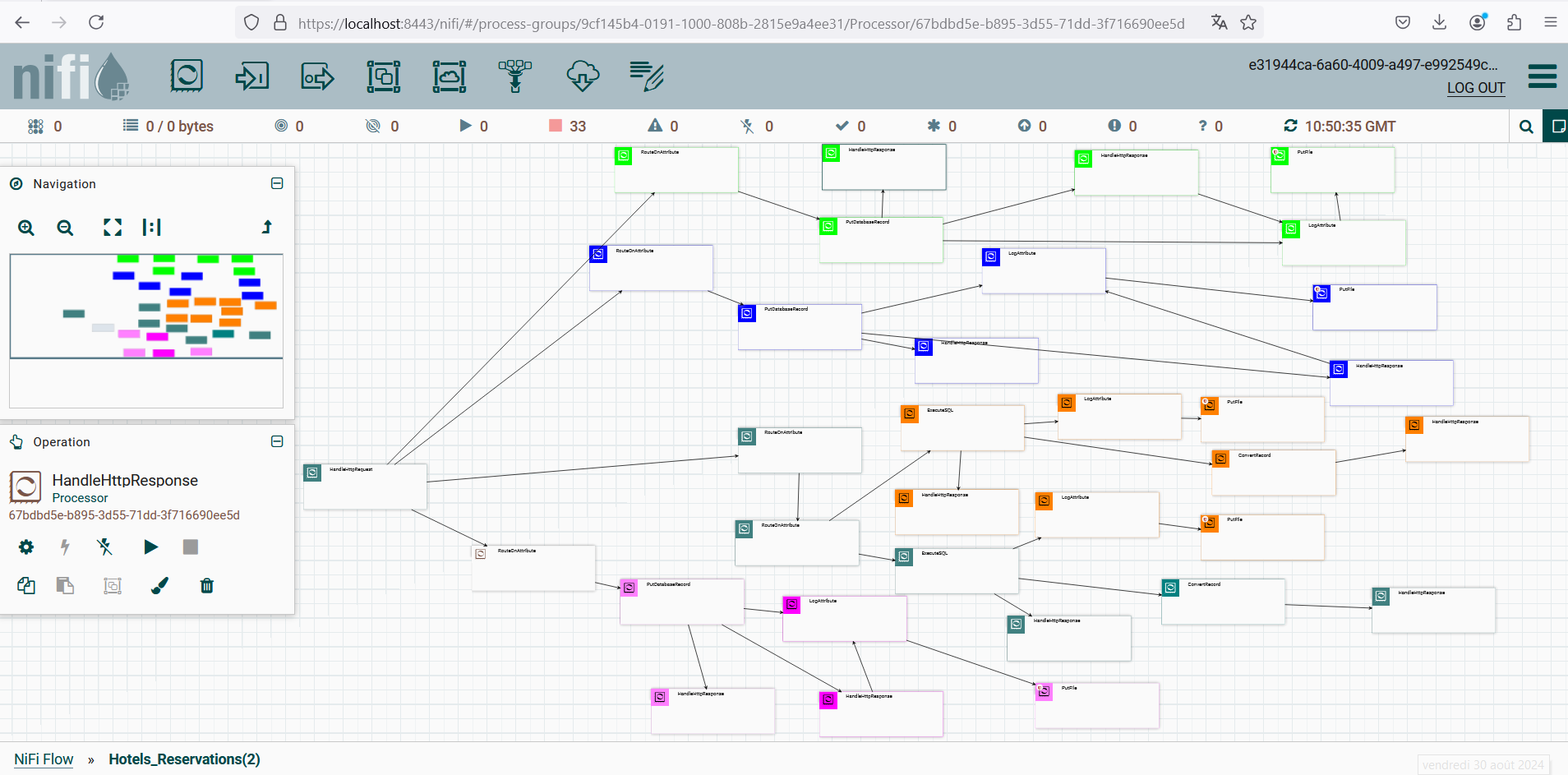


**Figure 53: Document récapitulatif des tests de toutes les requêtes**

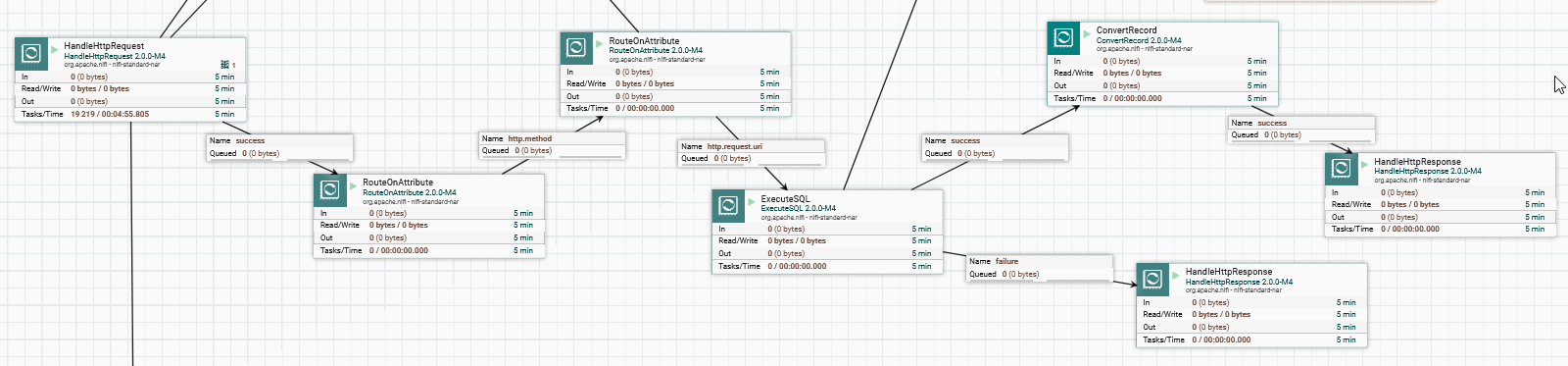
**Conclusion**

Ce projet a abouti au développement d'une solution intégrée pour la gestion des réservations d'hôtel, utilisant des technologies modernes comme Flask pour l'API RESTful et Apache NiFi pour l'ETL. MySQL a assuré une gestion robuste des données, et Postman a validé les fonctionnalités via des tests rigoureux. L'application répond aux besoins du secteur hôtelier, tout en offrant une architecture évolutive et adaptable à d'autres contextes, démontrant la flexibilité des technologies employées

**Annexes**

* Schéma global des processeurs utilisés dans Apache Nifi.
* 

**Figure 54: Schéma global des flux de données**



**Figure 55: Schéma du flux de récupération d'une réservation**