

# Validation d'API avec Postman



# POSTMAN

## Table des matières

Validation d'API avec Postman .....	1
Introduction .....	2
Présentation de Postman.....	2
Objectifs des tests.....	2
Configuration Initiale .....	2
Procédure de test.....	3
1°/ Démarrer le projet .....	3
2°/ Importer les données.....	3
3°/ Se placer dans le répertoire de l'exécutable .....	3
Analyse des données : .....	5
Résultat .....	5
Descriptions des tests : .....	6

# Introduction

Cette documentation décrit comment utiliser la collection Postman pour tester le chemin critique de l'application MedHead, de la création de compte à la réservation d'un lit hospitalier. Postman est utilisé ici pour envoyer des requêtes HTTP et visualiser les données JSON renvoyées par les microservices.

## Présentation de Postman

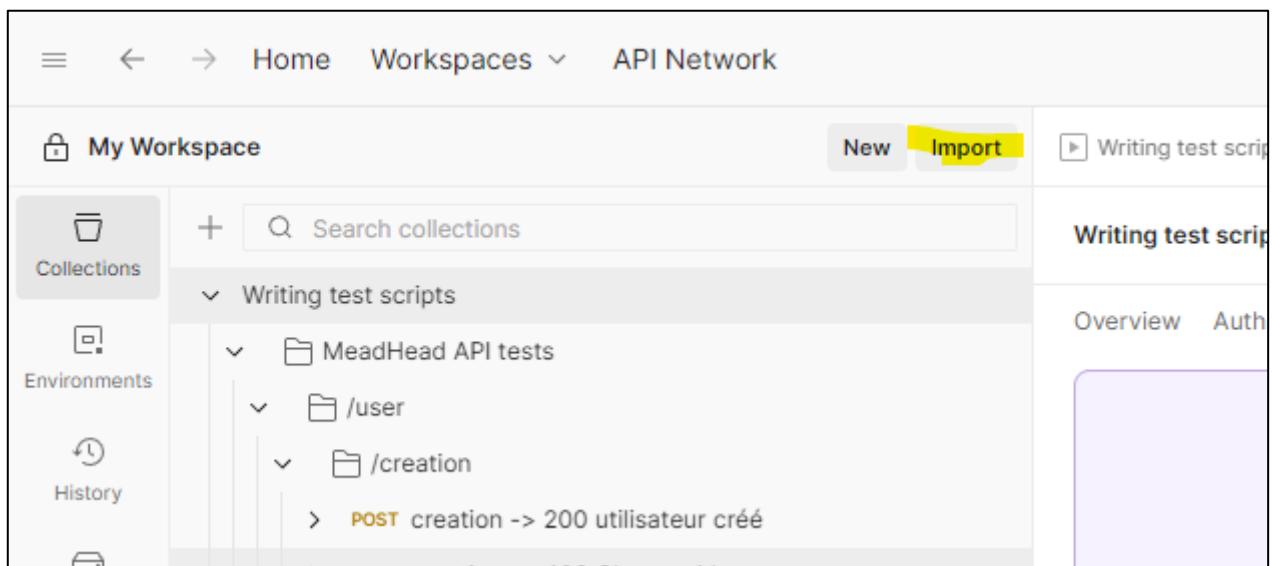
Postman permet d'envoyer des requêtes HTTP (GET, POST, PUT, DELETE, etc.) à un serveur et de visualiser la réponse immédiatement. C'est un peu comme un navigateur web, mais au lieu d'afficher une page graphique, il affiche les données brutes (souvent en format JSON) renvoyées par une base de données ou un service.

Postman permet aussi de créer des collections, c'est-à-dire plusieurs types de requêtes avec des données différentes et des scripts pour vérifier automatiquement les résultats de nos requêtes.

## Objectifs des tests

L'objectif principal des tests pour le projet MedHead est de valider le chemin critique d'un utilisateur au sein de l'application. Cela permet de s'assurer que les prochaines modifications apportées ne changent pas le comportement des Endpoint.

## Configuration Initiale



Puis importé la collection « tests\_POC\_11.postman\_collection » .

# Procédure de test

## 1°/ Démarrer le projet

Voir Initialisation du projet.

## 2°/ Importer les données

Se connecter à la bdd

`http://localhost:8081/?server=db&username=root&db=poc_db&select=specialisation`

Aller sur [Requête SQL](#), puis exécuter les requêtés du « data\_set.sql »

## 3°/ Se placer dans le répertoire de l'exécutable

The screenshot shows the Postman application interface. On the left, there's a sidebar with various collections like 'MeadHead API tests', 'Writing test scripts', and 'Writing test scripts Copy'. The 'Writing test scripts' collection is currently selected. The main area displays the 'Runs' tab under the 'Overview' section. It shows two recent runs:

Start time	Source	Duration	All tests	Passed	Failed	Skipped	Avg. Resp. Time	Comments
Dec 30, 2025 12:53:13	Runner	5s 824ms	117	117	0	0	64 ms	-
Dec 30, 2025 12:51:44	Runner	8s 72ms	117	116	1	0	115 ms	-

Cliquer sur la collection, puis Runs

Run Sequence

Deselect All | Select All | Reset

1  > > > > **POST** creation -> 200 utilisateur crée

2  > > > > **POST** creation -> 400 Champ vide

3  > > > > **POST** creation -> 400 email ko

4  > > > > > **POST** creation-> 400 email vide

5  > > > > > **POST** creation -> 400 pwd < 8

6  > > > > > **POST** creation -> 400 pwd weak

7  > > > > > **POST** creation -> 409 utilisateur existant

8  > > > > > **POST** conection -> 200 sucess

9  > > > > > **POST** conection -> 400 Pas de compte

10  > > > > > **POST** conection -> 400 Mauvais champ

11  > > > > > **POST** conection -> 401 Mauvais mdp

12  > > > > > **GET** specilites -> 200 sucess

13  > > > > > **GET** trajet -> 200 avec adresse

14  > > > > > **GET** trajet -> 200 avec position gps

15  > > > > > **GET** trajet -> 400 DTO Trop d'informations

16  > > > > > **GET** trajet -> 400 DTO Données manquantes

17  > > > > > **GET** trajet -> 400 DTO Latitude borne sup

18  > > > > > **GET** trajet -> 400 DTO Latitude borne inf

19  > > > > > **GET** trajet -> 400 DTO Longitude borne sup

20  > > > > > **GET** trajet -> 400 DTO Longitude borne inf

21  > > > > > **GET** trajet -> 400 DTO Données adresse ko

22  > > > > > **GET** trajet -> 400 DTO Données adresse ko espaces

23  > > > > > **GET** trajet -> 400 Mismath arg

24  > > > > > **GET** trajet -> 403 Forbiden (mauvais token)

25  > > > > > **GET** trajet -> 404 Pas de lit disponible

Functional Performance

Choose how to run your collection

Run manually  
Run this collection in the Collection Runner.

Schedule runs  
Periodically run collection at a specified time on the Postman Cloud.

Automate runs via CLI  
Configure CLI command to run on your build pipeline.

Run configuration

Iterations

Delay  ms

Test data file Only JSON and CSV files are accepted.

> Advanced settings

# Analyse des données :

## Résultat

Writing test scripts - Run results

Ran today at 12:53:13 PM · [View all runs](#)

Source	Environment	Iterations	Duration	All tests	Errors	Avg. Resp. Time
Runner	none	1	5s 824ms	117	0	64 ms

[All Tests](#) Passed (117) Failed (0) Skipped (0) Errors (0) Console Log [Generate Tests](#) [View Summary](#)

Iteration 1

**POST** MeadHead API tests / /user / /creation / creation -> 200 utilisateur créé  
http://localhost:8080/api/user/creation

PASS L'utilisateur est créé avec succès (200 OK)

PASS Temps de réponse inférieur à 800ms

PASS Le corps de la réponse est vide

**POST** MeadHead API tests / /user / /creation / creation -> 400 Champ vide  
http://localhost:8080/api/user/creation

PASS Statut attendu : 400 Bad Request

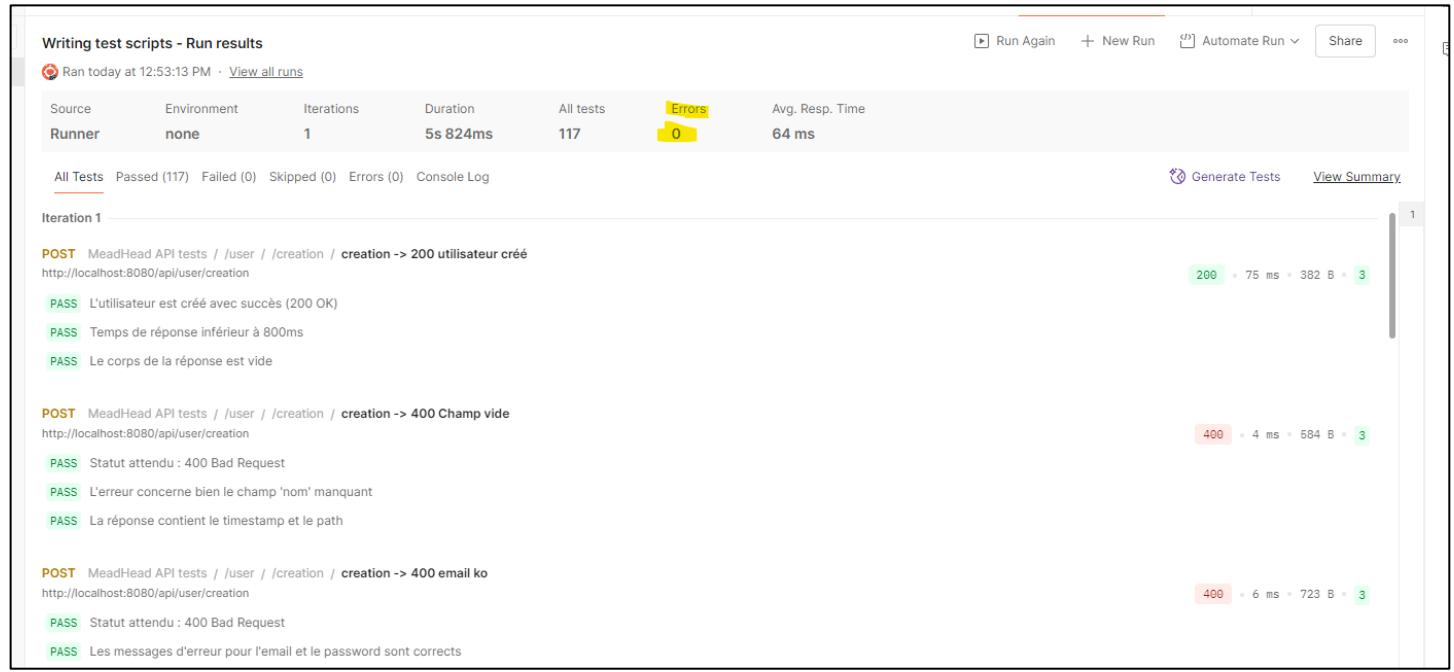
PASS L'erreur concerne bien le champ 'nom' manquant

PASS La réponse contient le timestamp et le path

**POST** MeadHead API tests / /user / /creation / creation -> 400 email ko  
http://localhost:8080/api/user/creation

PASS Statut attendu : 400 Bad Request

PASS Les messages d'erreur pour l'email et le password sont corrects



Les tests devraient passer sans erreurs.

# Descriptions des tests :

```
Writing test scripts
└── MeadHead API tests
    ├── /user
    │   ├── /creation
    │   │   > POST creation -> 200 utilisateur créé
    │   │   > POST creation -> 400 Champ vide
    │   │   > POST creation -> 400 email ko
    │   │   > POST creation-> 400 email vide
    │   │   > POST creation -> 400 pwd < 8
    │   │   > POST creation -> 400 pwd weak
    │   │   > POST creation -> 409 utilisateur existant
    │   └── /connection
    │       > POST conection -> 200 sucess
    │       > POST conection -> 400 Pas de compte
    │       > POST conection -> 400 Mauvais champ
    │       > POST conection -> 401 Mauvais mdp
    └── /specialitees
    └── /unitesoins
    └── /reservation
    └── /unitesoins

    GET 404 end point pas trouvé
    GET 404 end point pas trouvé
```

Chacun des tests est classé par end point, triées par code retour croissant.

HTTP Writing test scripts / MeadHead API tests / /user / /connection / connection -> 200 sucess

POST

{{base\_url}} /user/connection

Docs Params Authorization Headers (10) Body Scripts Settings

Pre-request

Post-response

```
1 // 1. Vérifier le code de statut
2 pm.test("Connexion réussie : 200 OK", function () {
3 |   pm.response.to.have.status(200);
4 });
5
6 // 2. Vérifier la présence et le format du Token
7 pm.test("Le token Bearer est présent et bien formé", function () {
8 |   var jsonData = pm.response.json();
9 |
10 // CORRECTION : On définit la variable 'token' avant de l'utiliser
11 // On extrait la valeur depuis 'jsonData.Bearer'
12 var tokenValue = jsonData.Bearer;
13
14 // Vérifie que la propriété existe bien
15 pm.expect(jsonData).to.have.property("Bearer");
16
17 // Vérifie le format
18 pm.expect(tokenValue).to.be.a('string');
19 pm.expect(tokenValue.length).to.be.above(10);
20
21 // SAUVEGARDE : Utilise le nom de la variable définie au-dessus
22 pm.collectionVariables.set("token_jwt", tokenValue);
23
24 console.log("Token JWT sauvégarde avec succès !");
25 });
26
27 // 3. Vérifier le temps de réponse
28 pm.test("Réponse de connexion < 2s", function () {
29 |   pm.expect(pm.response.responseTime).to.be.below(2000);
30 });


```

Ils tests au moins le code retour et le format de la donnée. Pour le token de connexion (celui qui permet de tester les Endpoint nécessitant une connexion, il est récupéré sur le test de la connexion).