

Formation

# Tests webservices avec Postman

Décembre 2022

**onepoint.**

# Programme

1. Introduction aux tests API
2. Installation et configuration Postman
3. Création des requêtes simples
4. Ajout des vérifications
5. Enchainement de plusieurs requêtes
6. Utilisation des variables
7. Utilisation des données CSV / JSON
8. Création d'une boucle
9. API Mock
10. Documentation des APIs (swagger)
11. Annexes  
(Exercices complémentaires, Exécution en ligne de commande, Intégration Jenkins, Exercices complémentaires avec taiga.io, Présentation de l'application de démo, Liens utiles)

1

---

# Introduction aux tests API

# Rappel ISTQB : niveaux de test

Le syllabus de l'ISTQB définit 4 niveaux de test principaux :

1. Les tests composants
2. Les tests d'intégration
3. Les tests système
4. Les tests d'acceptation

Conseil :

Penser à les différencier par les vérifications qui sont faites dans chaque niveau, et les risques couverts.

# Rappel ISTQB : techniques de test

- Boîte noire
  - 1. Partitions d'équivalence
  - 2. Analyse des valeurs limites
  - 3. Test de tables de décision
  - 4. Test des transitions d'état
  - 5. Test des cas d'utilisation
- Boîte blanche
  - 1. Test et couverture des instructions
  - 2. Test et couverture des décisions

Conseil :

Pour les tests API, on utilisera le plus souvent les tests en boîte grise.

# Rappel ISTQB : Définition d'un cas de test

Cas de test : un ensemble de valeurs d'entrée, de **préconditions** d'exécution, de **résultats attendus** et de post conditions d'exécution, développées pour un objectif ou une condition de tests particulier, tel qu'exécuter un chemin particulier d'un programme ou vérifier le respect d'une **exigence spécifique**

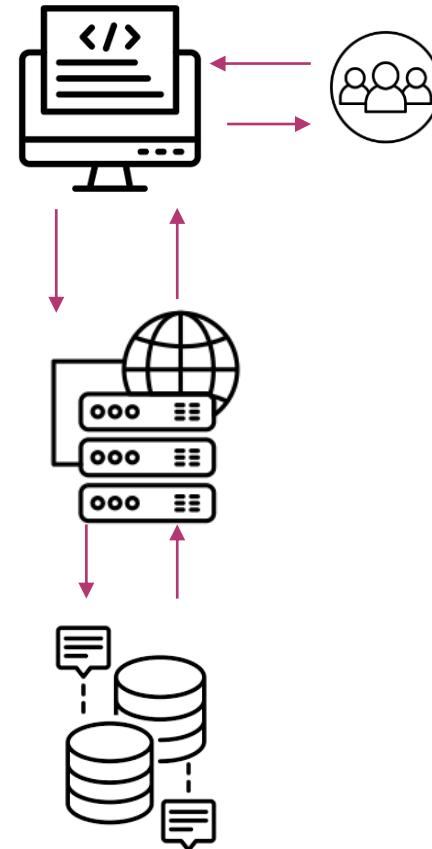
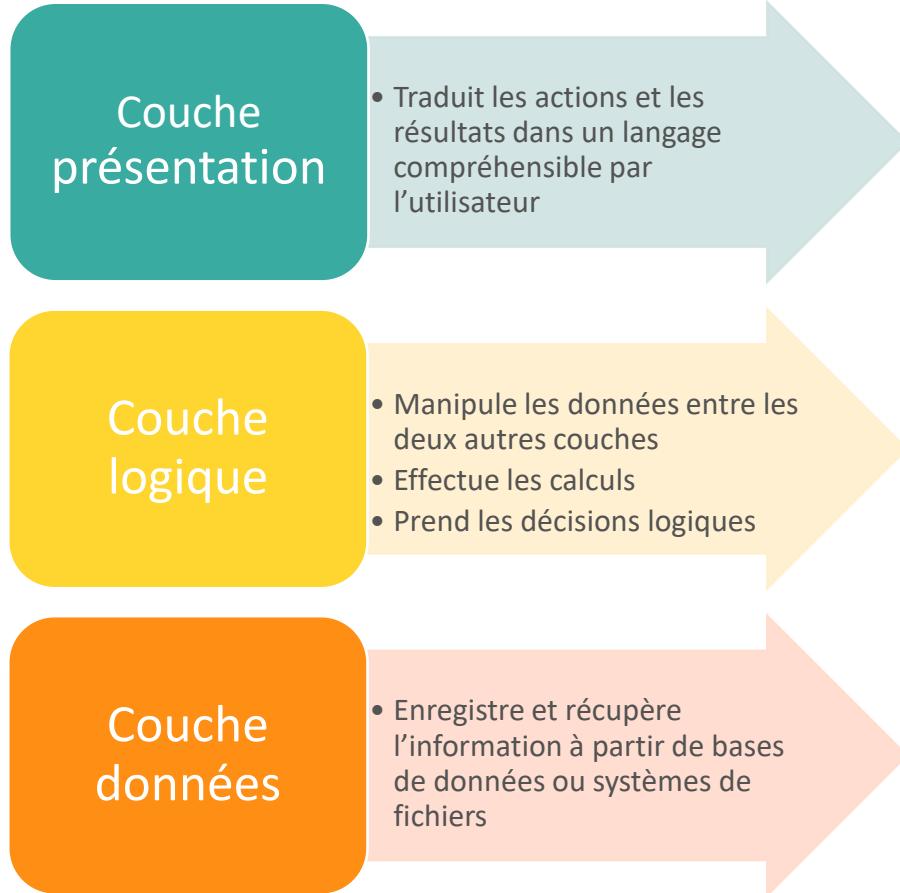
[d'après IEEE 610]

# Architecture et intégration de l'application

- Différentes stratégies d'intégration existent et peuvent être basées sur :
  - l'architecture de l'application ;
  - les tâches fonctionnelles ;
  - les séquences d'exécution de transactions...
- Stratégie Top-Down
  - Approche incrémentale où les composants en haut de la hiérarchie sont intégrés et testés d'abord, avec les composants de niveau inférieur simulés par des bouchons.
- Stratégie Bottom-Up
  - Approche incrémentale où le niveau le plus bas des composants est intégré et testé d'abord, et ensuite utilisé pour faciliter les tests des composants de plus haut niveau.
- **Stratégie Big Bang (déconseillé)**
  - Un type d'intégration dans lequel les éléments logiciels, matériel ou les deux sont combinés en une seule fois.
  - Peut poser des problèmes pour identifier les cas de tests et les raisons des erreurs.

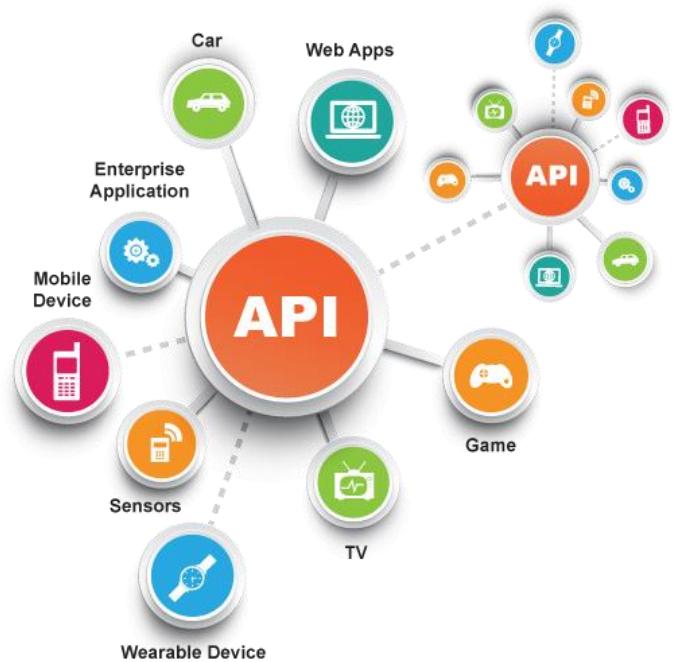


# Architecture n-tiers



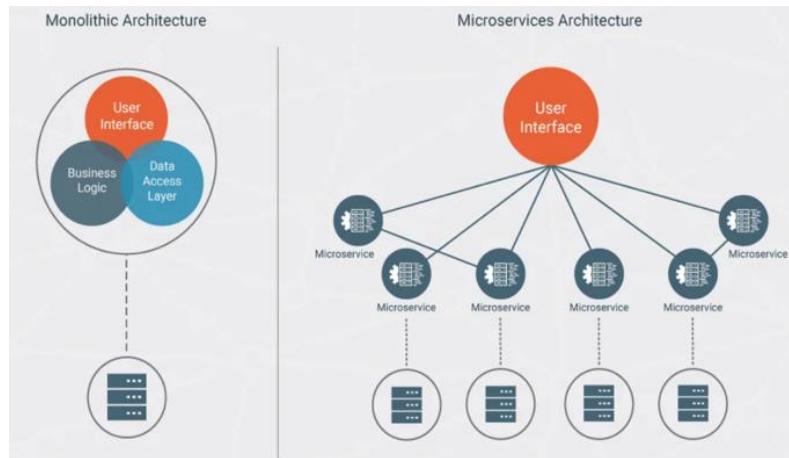
# Les API – définition

- Les API (Application Programmable Interface) sont des ensembles de règles et standards qui étendent les fonctionnalités d'un composant ou d'une application.
- Ce sont des interfaces standardisées qui permettent à plusieurs applications d'échanger des informations.
- Une API définit en général :
  - Les types de messages et de données que le composant peut accepter en entrée
  - Les types et les contenus des réponses
  - Les composants peuvent appartenir à la même application ou être fournis par des services tiers



# Pourquoi les API ?

- Les applications développées doivent être flexibles et pouvoir s'adapter à différents environnements
- Elles doivent souvent communiquer avec d'autres applications ou services du SI et doivent s'interfacer facilement
- Elles devront en outre pouvoir intégrer des changements dans un minimum de temps
- Cela s'appuie nécessairement sur une architecture applicative construite sur des composants suffisamment petits et réutilisables
- Les assemblages de composants peuvent varier en fonction des besoins
- Les interfaces entre ces composants sont donc essentielles au bon fonctionnement du système complet

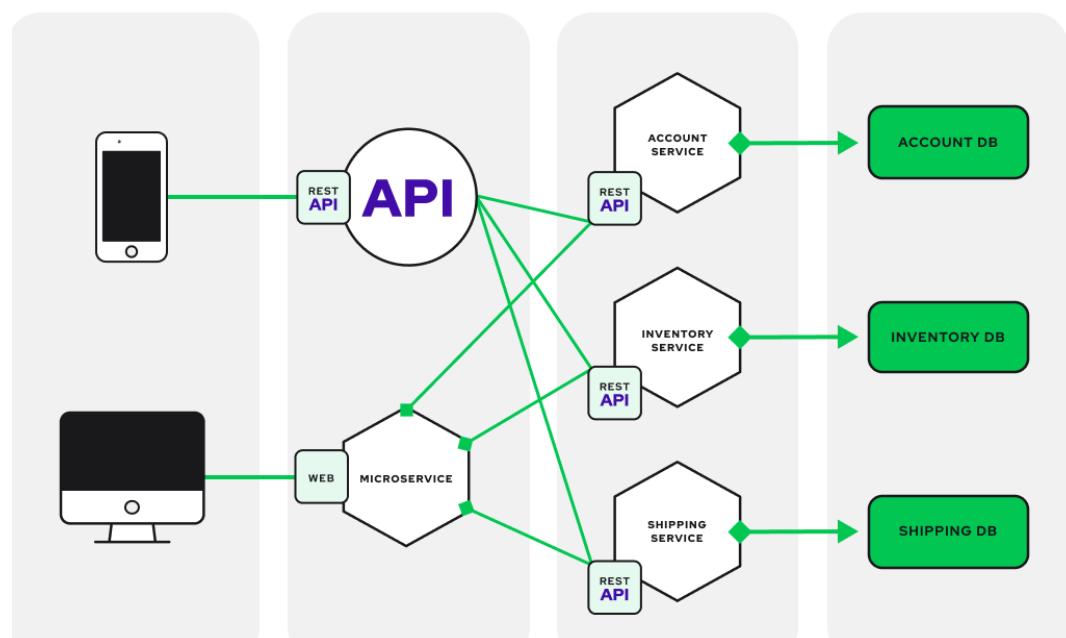


# Architecture microservices

- Approche architecture et organisation du développement logiciel.
- Découpage en petits services indépendants qui communiquent via des API bien définies.
- Simplifient la mise à l'échelle et accélèrent le développement des applications
- Caractéristiques : Autonomie et spécialisation

## Avantages

- Agilité
- Flexibilité pour le dimensionnement
- Facilité de déploiement
- Indépendance de la technologie
- Réutilisabilité du code
- Résilience

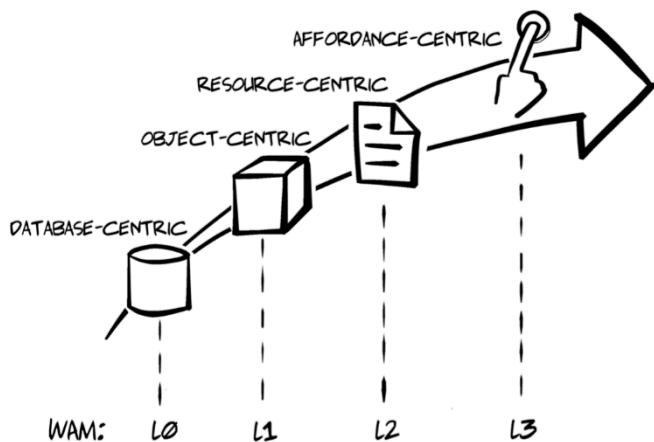


# Niveau de maturité Fonctionnelle

## Maturité des API

L'échelle d'Amundsen a pour objectif de mesurer la maturité des API d'un point de vue fonctionnel

### Web API Design Maturity Model



#### Level 0 : Database Centric

Aucune abstraction entre la couche technique de persistance et ce qui est exposé au client de l'API. Le modèle de données de la base est celui exposé au client. Un changement dans la base impact le client. La complexité technique du système ne lui est pas masqué.

#### Level 1 : Object Centric

On expose des objets techniques (liés à l'implémentation du service) au client plutôt que le modèle de la base directement (level 0). On peut changer, sans impacter le client, en théorie le modèle de BDD mais pas l'implémentation interne du service (framework/middleware).

#### Level 2 : Ressource Centric

L'API permet de découpler l'implémentation interne du service & le modèle de BDD de la ressource exposée au client de l'API. On peut le mettre en place via le Representor Pattern dans lequel le modèle interne du service est remplacé par la représentation d'une ressource avant exposition.

#### Level 3 : Affordance Centric

L'API est orientée cas d'usage & valeur ajoutée pour répondre aux besoins de ses/son consommateur(s). Le client sait directement comment utiliser l'API car elle est intuitive & orientée produit (simple d'usage, markétée & bien packagée, pas besoin d'une notice compliquée, répond à un besoin concret...).

# Quelques exemples d'API

- Paiement sécurisé en ligne

The screenshot shows a "Identification" page from "Verified by VISA". It asks for the cardholder's date of birth. The form fields include:

- Marchand : J'achète.fr
- Montant : 75€
- Date : 10/06/2009
- N° de carte : XXXX XXXX XXXX 1234
- Date de naissance du porteur de la carte :  /  /

A note at the bottom states: "Cette identification est obligatoire pour conclure votre transaction. Si vous refusez de vous identifier, votre achat sera annulé." A link "Ne pas m'identifier et annuler mon achat" is also present.

- Calculs de temps de trajet et tarif en fonction de la localisation



This screenshot shows a user interface for checking store availability. It includes:

- A small image of a store interior.
- A dropdown menu: "Disponibilités en magasin" with "Magasin: Avignon" selected.
- A message: "Disponible en ligne" followed by "Disponible pour l'achat en ligne".
- A progress bar with four green segments labeled "Disponible à Avignon".
- A link: "Voir les prévisions".

Disponible en ligne  
Disponible pour l'achat en ligne

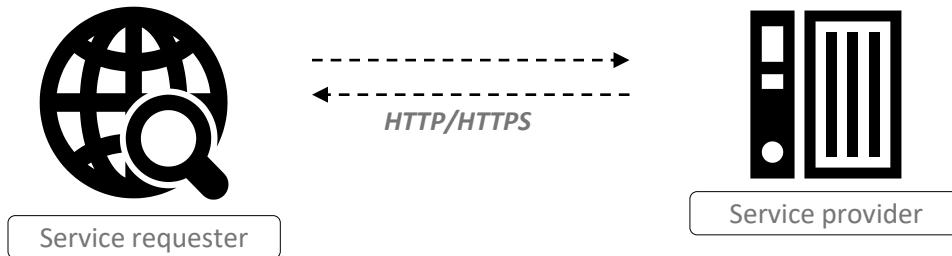
Disponible à  
Avignon  
[Voir les prévisions](#)

37 - 46€

**onepoint.**

# Fonctionnement des API

Le concept fondamental d'une API web est de permettre l'échange de données en utilisant les protocoles HTTP/HTTPS

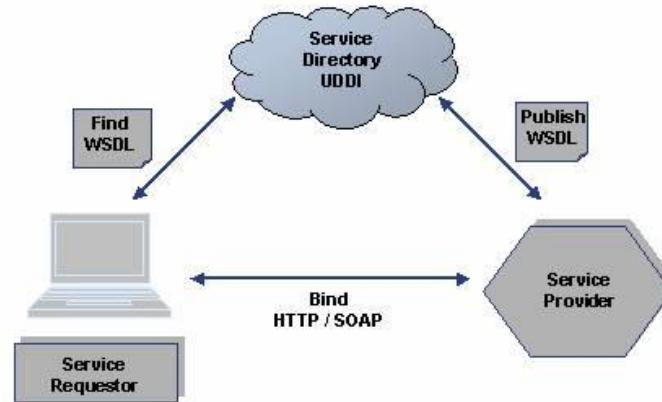


Il s'appuie sur deux composants :

- Structure des données : format spécifique utilisé par les requêtes et réponses API (XML, JSON...)
- Transfert et sécurisation des données : définit la façon dont les données sont échangées et manipulées en fonction des méthodes HTTP utilisées (PUT, POST, GET...) et des protocoles de Web Services utilisés comme SOAP ou REST

# Web Services

- Un web service est une méthode de communication entre deux dispositifs connectés sur le web. C'est un service logiciel fourni à une adresse réseau (endpoint) et qui est actif en permanence (always on)



- Le langage WSDL (Web Services Description Language) est un langage de description d'interface au format XML qui décrit la fonctionnalité offerte par un Web Service.
- Le fichier WSDL Associé au Web Service définit :
  - La manière dont le service peut être appelé
  - Les paramètres qu'il attend
  - La structure de la réponse qu'il fournit

# Web Services

Les web services reposent sur différents protocoles et technologies pour fonctionner

## Architecture

- **REST** : style d'architecture pour les environnements distribués décrivant comment construire des applications sur internet.
- **SOAP** : protocole standard de transmission de message décrit en XML. Il se présente comme une enveloppe pouvant être signée et pouvant contenir des données ou des pièces jointes. Il utilise le protocole HTTP et permet d'effectuer des appels de méthodes à distance.

## Protocoles

- **HTTP** : protocole de communication client-serveur développé pour le World Wide Web. Il est utilisé pour échanger toute sorte de données entre client HTTP et serveur HTTP.

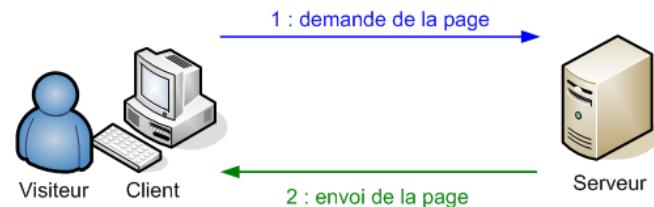
## Langages

- **XML** : Langage de balisage générique. On dit que ce langage est « extensible » car il permet de définir des espaces de noms, c'est-à-dire des langages ayant chacun leur propre vocabulaire et leur propre grammaire.
- **JSON** : JavaScript Objet Notation est un langage léger d'échange de données textuelles avec une syntaxe simple et une structure en arborescence
- **HTML** : Langage de balisage pour représenter les pages web.
- **WSDL** : Langage de description des Web services. Il s'agit de l'interface présentée aux utilisateurs qui indique comment utiliser un web service (types des données échangées, messages, etc.).
- **XPATH** : Langage permettant de requêter des portions dans un document XML.
- **XQUERY** : Langage permettant à la fois d'extraire des informations d'un document XML mais également de réaliser des calculs sur ces dernières.

# Web Services

## Protocole HTTP et requêtes

- HTTP est un protocole permettant la communication entre un client et un serveur. Une requête est toujours composé d'un appel (du client) et d'une réponse (du serveur). Voici un schéma résumant l'échange, les requêtes sont symbolisés par les flèches.



- Syntaxe d'une requête et d'un réponse HTTP:

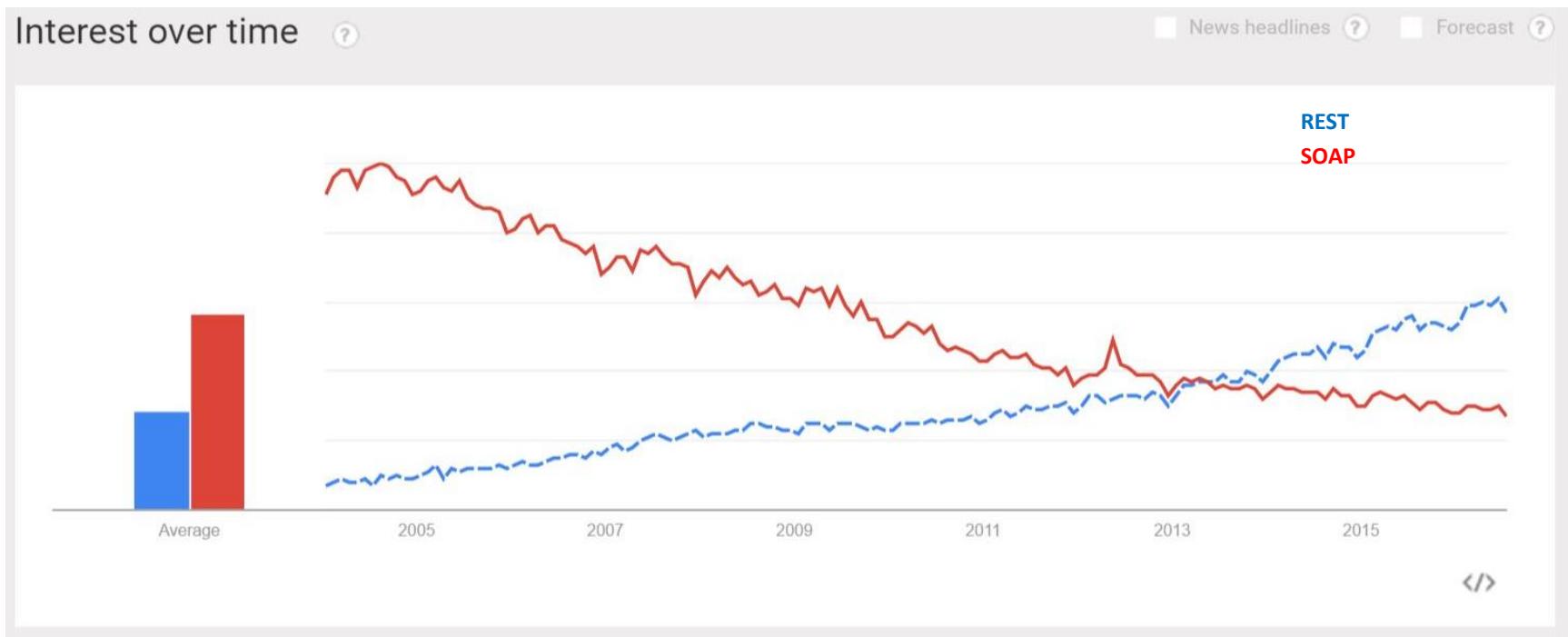
Requête Client	Réponse Serveur
<b>Ligne de commande</b> <ul style="list-style-type: none"><li><u>Syntaxe</u> : Commande, URL, version de protocole.</li><li><u>Exemple</u> : GET /repertoire/page.html HTTP/1.1</li></ul>	<b>Ligne de statut</b> <ul style="list-style-type: none"><li><u>Syntaxe</u> : version, Code-réponse, Texte-réponse.</li><li><u>Exemple</u> : HTTP/1.1 200 OK</li></ul>
<b>En-têtes de la requête</b> <ul style="list-style-type: none"><li><u>Syntaxe</u> : « Nom entête :valeur ».</li><li><u>Exemple</u> : Host: www.site.com</li></ul>	<b>En-têtes de réponse.</b> <ul style="list-style-type: none"><li><u>Syntaxe</u> : « Nom entête :valeur ».</li><li><u>Exemple</u> : Server: Apache/2.2.3</li></ul>
<b>Saut de ligne</b>	<b>Saut de ligne</b>
<b>Corps de la requête</b> sert principalement à transmettre les formulaires HTML.	<b>Corps de réponse</b> Contient le contenu, dans le cas d'une page web le HTML par exemple.

# SOAP vs REST

SOAP	REST
→ Orienté Services	→ Orienté Ressources (pages, images, etc.)
<b>Avantages</b> <ul style="list-style-type: none"><li>Mécanisme d'appel de procédures distantes, on peut s'appuyer sur Soap dans un cadre contractuel.</li><li>Supporte le transactionnel</li><li>Formel, les méthodes et résultats sont spécifiés via des contrats (WSDL).</li><li>Fiable, mécanismes garantissant la fiabilité de la transmission et réception des données.</li><li>Soap est intégré dans la plupart des environnements de développement</li></ul>	<b>Avantages</b> <ul style="list-style-type: none"><li>Léger et simple, messages courts et faciles à décoder pour le navigateur et serveur d'application</li><li>Auto-descriptif : navigation intuitive à travers les ressources.</li><li>Stateless: Adapté pour des opérations simples (consulter, supprimer, modifier, etc.)</li><li>Moins de dépendances sur les outils.</li><li>Architecture similaire aux sites internet.</li></ul>
<b>Inconvénients</b> <ul style="list-style-type: none"><li>Soap est considéré comme trop complexe et verbeux pour des besoins ordinaires.</li><li>Nécessite des outils pour le développement</li></ul>	<b>Inconvénients</b> <ul style="list-style-type: none"><li>Manque de complexité exigés par les transactions d'affaires</li><li>REST est incomplet pour fournir une solution fiable aux entreprises lors d'échanges complexes.</li></ul>



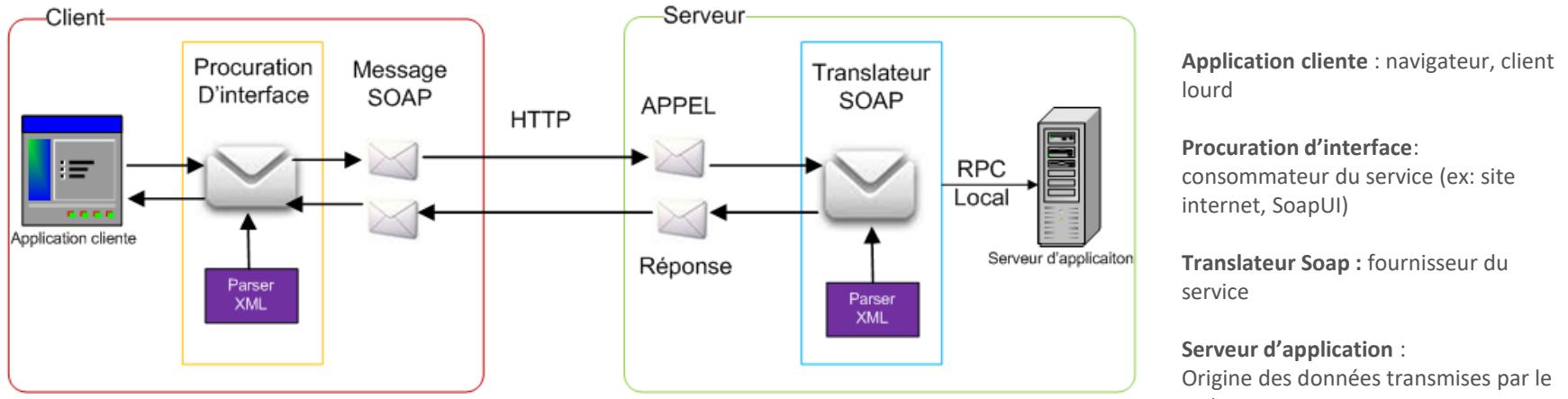
# SOAP vs REST



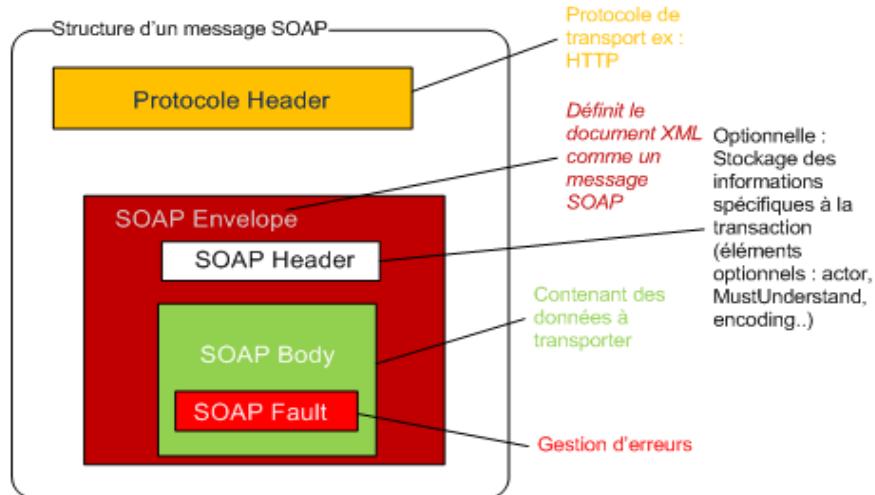
Source : <https://dzone.com/articles/rest-api-vs-soap-web-services-management>

# Architecture du protocole SOAP

- L'invocation de méthodes lors d'une communication Soap entre des serveurs peut être schématisée de la manière suivante :



- La communication est réalisé par l'envoi de messages XML. Voici la structure d'un message SOAP.



# Exemple de messages SOAP

- CurrencyConverter est un service web Soap libre d'accès. Il propose une opération « ConversionRate » accessible sur 2 protocoles (Soap 1.1 et Soap 1.2).
- Voici une trace des échanges Soap effectués lors de la communication entre le serveur et le client :

## → Requête SOAP (client)

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:web="http://www.webserviceX.NET/">  
  
    <soapenv:Header/>  
  
    <soapenv:Body>  
        <web:ConversionRate>  
            <web:FromCurrency>EUR</web:FromCurrency>  
            <web:ToCurrency>USD</web:ToCurrency>  
        </web:ConversionRate>  
    </soapenv:Body>  
</soapenv:Envelope>
```

Données transmises

## → Réponse SOAP (serveur)

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  
    <soap:Body>  
        <ConversionRateResponse xmlns="http://www.webserviceX.NET/">  
            <ConversionRateResult>1.0849</ConversionRateResult>  
        </ConversionRateResponse>  
    </soap:Body>  
</soap:Envelope>
```

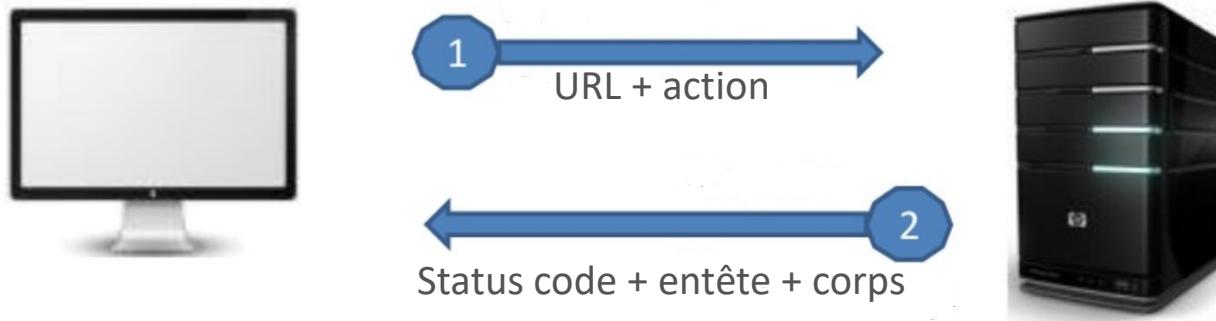
Soap Body (données échangées)

Données renvoyées

# Exemple de message REST

Structure du message REST

Requête      *GET https://eshop.com/books/14675*



Réponse

*HTTP status : 200 OK*

*Allow : GET, HEAD, OPTIONS  
content-type : application/json  
vary : Accept*

{  
  "**id**" : "14675",  
  "**title**" : "Game of Thrones",  
  "**author**" : "George R. R. Martin",  
  "**isbn**" : "1-84356-028-3",  
  "**availableQty**" : "4"  
}

# Commandes et codes HTTP

- Les commandes permettent de spécifier le type de requête et la méthode à utiliser. Voici une liste des commandes :

Commande	Description
<b>GET</b>	C'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource, il doit être possible de répéter la requête sans effet.
<b>HEAD</b>	Cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.
<b>POST</b>	Cette méthode doit être utilisée lorsqu'une requête modifie la ressource.
<b>OPTIONS</b>	Cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.
<b>CONNECT</b>	Cette méthode permet d'utiliser un proxy comme un tunnel de communication.
<b>TRACE</b>	Cette méthode demande au serveur de retourner ce qu'il a reçu, dans le but de tester et d'effectuer un diagnostic sur la connexion.
<b>PUT</b>	Cette méthode permet d'ajouter une ressource sur le serveur.
<b>DELETE</b>	Cette méthode permet de supprimer une ressource du serveur.

- Les codes permettent de renvoyer le résultat de l'exécution d'une requête par un serveur à un client :

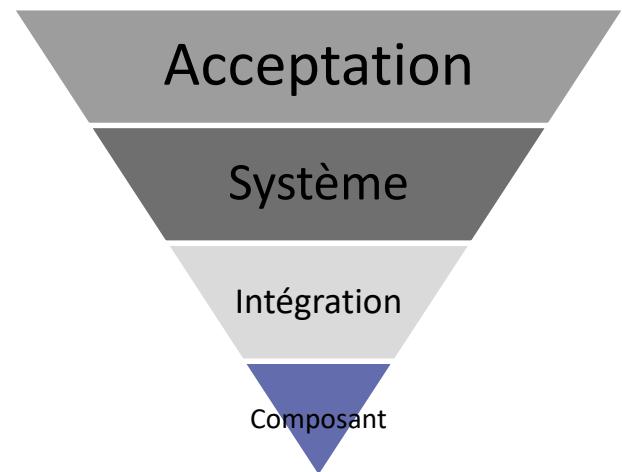
Groupe	Exemple (Code, message et description)
<b>Information</b>	<b>100 : Continue.</b> Attend la suite de la requête
<b>Succès</b>	<b>200 : OK.</b> Requête traitée avec succès <b>201 : Created.</b> Requête traitée avec succès avec création d'un document
<b>Redirection</b>	<b>301 : Moved Permanently.</b> Document déplacé de façon permanente <b>302 : Moved Temporarily.</b> Document déplacé de façon temporaire
<b>Erreur du client / du serveur web</b>	<b>401 : Unauthorized.</b> Une authentification est nécessaire pour accéder à la ressource <b>403 : Forbidden.</b> Le serveur a compris la requête, mais refuse de l'exécuter <b>404 : Not Found.</b> Ressource non trouvée
<b>Erreur du serveur / du serveur d'application</b>	<b>500 : Internal Server Error.</b> Erreur interne du serveur <b>502 : Bad Gateway ou Proxy Error.</b> Mauvaise réponse envoyée à un serveur intermédiaire par un autre serveur.

# Pourquoi tester les API ?

- Les tests IHM présentent des inconvénients
  - Ils ne peuvent être exécutés que lorsque l'IHM est prête
  - Leur temps d'exécution est souvent très long (manuel)
  - Les scripts automatisés sur l'IHM sont souvent très sensibles aux changements et donc difficiles à maintenir
- Exemples :
  - Une suite de tests de régression qui prend plusieurs heures à être exécutée n'est pas viable sur un cycle d'intégration continue où les développeurs ont besoin d'un feedback rapide sur leur build
  - Le développement d'une nouvelle appli où la logique métier est surtout contenue dans les APIs

# Comment tester des API

- Périmètre de test
  - Implémentation cachée de l'application (test sous l'IHM)
- Niveaux de test
  - Composant : API individuelles
  - Intégration : scénarios de bout-en-bout
- Objectifs de test
  - Focaliser sur le fonctionnement attendu des composants avant intégration dans l'application
  - Vérifier les caractéristiques non fonctionnelles (performance, robustesse, sécurité...)
- Techniques de test
  - Utiliser un outil pour envoyer des requêtes et vérifier les réponses
  - Simuler les données ou les flux de contrôle



# Comment tester des API

- Etape 1 : savoir ce que l'API doit faire
  - Documentation (swagger)
  - Information obtenue des développeurs
  - Capture de traces
- Etape 2: écrire des scénarios de test avec vérification des résultats attendus
  - Statut code
  - Présence du message retourné
  - Contenu du message retourné
  - Temps de réponse
- Etape 3 : implémenter les tests dans l'outil
- Etape 4 : exécuter les tests avec l'outil

# Comment tester des API

- Pour les arguments de la requête (valeurs d'entrée)
  - Partitions d'équivalence, analyse aux valeurs limite
- Pour les combinaisons de paramètres
  - Techniques combinatoires (n-wise)
- Pour la gestion des erreurs
  - Gestion des exceptions
  - Partitions d'équivalence sur les sorties (codes d'erreur)
- Pour les scénarios de bout en bout
  - Transitions d'état sur les séquences d'appels API
  - Cas d'utilisation

# Comment tester des API

Les problèmes fréquents :

- Difficulté de connexion/communication avec l'API
- API qui ne retourne aucune réponse
- API qui retourne une réponse non/mal structurée
- API qui retourne une réponse non conforme par rapport à la requête envoyée
- Temps de réponse

# Synthèse des tests API



- Très utiles pour l'équipe de développement et de test
- Très efficaces pour trouver des bugs lors de l'intégration de l'application
- Peuvent vérifier les caractéristiques fonctionnelles et non fonctionnelles de l'application
- Leur automatisation est plus rapide et plus stable que pour les tests IHM
- Apportent une visibilité plus rapide à l'équipe de développement sur le fonctionnement de l'application



- Pas d'interface disponible pour préparer les tests
- Pas de vérification de l'utilisabilité de l'interface (IHM)
- Nécessitent des outils spécifiques
- Demandent des testeurs avec des compétences additionnelles



# Savoir parler des API sans être un expert

Par Guillaume ADIN, Jean-Arnaud GAUTHUN et Yacine DECHMI

## Définition API

**API ou Application Programming Interface** est une interface de programmation d'application proposant un ensemble de fonctions informatiques normalisées qui permettent à une application d'exposer, communiquer et échanger du contenu avec d'autres applications.

## Les 2 grands usages des API

### Fourniture de Service

Exposer un service de bout en bout :

- Service d'authentification (google, Facebook)
- Service de cartographie (google maps)
- Service de paiement (Apple Pay, PayPal, PayLib)
- Service intelligence artificielle (IBM Watson)

### Partage de données

Partager des données en interne ou en externe

- Partager des données RH dans le SI
- Echanger des données entre deux applications
- Exposer des données à des partenaires

## Les erreurs à éviter

**Ne pas définir de format d'échange entre les API ou vers les API**: complexité d'accès + maintenance à risque

**Ouvrir l'accès à tous**: Rendre visible des données confidentielles

**Ne pas mettre de file d'attente**: Surcharger le gestionnaire d'API

**Ne pas déployer de cache\* pour gérer les files d'attentes de requêtes**: Surcharger le gestionnaire d'API

**Ne pas communiquer sur les modes d'accès aux API**: Impossibilité d'utiliser les API

**Ne pas communiquer sur les existences des API dans l'entreprise**: Duplication des API

\*Toutes les réponses à des requêtes identiques sur un laps de temps (à définir) sont les mêmes

## L'API management

**La gestion des API permet de gérer le peuplement des API dans SI, et permet de :**

- Gérer le dispatch sur les API (empêcher les plus demandées soient saturées) – Passerelle API
- Mettre les API à disposition de façon sécurisée
- Gérer le cycle de vie des APIs
- Fournir de l'information sur l'utilisation de l'API
- Monétiser l'utilisation des API

## Matrice d'évaluation by onepoint

### Gouvernance

- Stratégie
- Gouvernance
- Considération juridique

### Business Model

### Business Model

- Chaîne de valeur
- Marketing
- Documentation

### Support

- Opérer les APIs
- Support offert aux développeurs

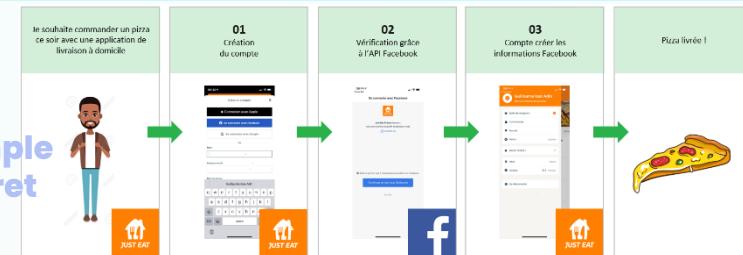
### API Engineering

- Design
- Implémentation
- Qualité
- Infrastructure

### Cyber Sécurité

- Règles de sécurité
- Traçabilité
- Audibilité
- Sécurisation de l'API

Un exemple concret



**onepoint.**

2

---

## Installation et configuration Postman

# Pourquoi utiliser Postman ?

- *Un outil gratuit* de test des webservices principalement en REST.
- *Accessibilité et convivialité*
- Peut-être implémenté sur *multiple système d'exploitation* : Windows, Mac OS, Linux.
- *Une large communauté*
- Les appels d'API *sans avoir à coder des scripts*.
- Fonctionnalités riches.
- Largement répondu.



# Prérequis

1. Poste Windows 10
2. Compte Postman actif : <https://identity.getpostman.com/signup>
3. Télécharger l'application et l'installer : <https://www.postman.com/downloads/>
4. Télécharger l'application FlightApi.zip et dézipper le contenu dans un dossier de travail  
[https://github.com/imhah-hassan/Postman\\_training/blob/master/Demo%20App/FlightApi.zip](https://github.com/imhah-hassan/Postman_training/blob/master/Demo%20App/FlightApi.zip)

# Vue d'ensemble

## POSTMAN - API DEVELOPMENT PLATFORM



# Installation

Télécharger Postman : <https://www.postman.com/downloads/>

## The Postman app

Download the app to get started with the Postman API Platform.



Windows 64-bit



*Vous pouvez créer un nouveau compte*

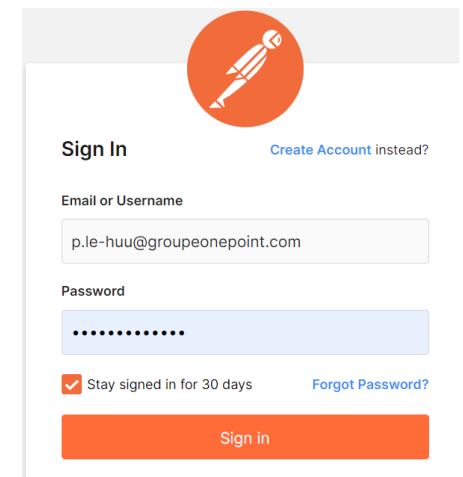


*Connectez vous avec le compte que vous venez de créer !*

Create an account or sign in

Create Free Account

Sign in



The image shows the Postman Sign In page. It features a large orange circular icon with a white pen nib. Below it, the text "Sign In" is displayed, followed by a link "Create Account instead?". There are two input fields: "Email or Username" containing "p.le-huu@groupeonepoint.com" and "Password" showing a series of dots. A checkbox "Stay signed in for 30 days" is checked, and a link "Forgot Password?" is available. At the bottom is a large orange "Sign in" button.

# La page d'accueil

*Page d'accueil*

*Où nos travaux seront organisé*

The screenshot shows the Postman application interface. At the top, there is a navigation bar with links for Home, Workspaces, API Network, and Explore. A search bar is also present. The main content area features a greeting message: "Good afternoon, bruno!" followed by the text "Pick up where you left off.". On the left side, there is a sidebar with the heading "Postman works best with teams" and a sub-section about collaborating in real-time. It includes a "Create Team" button and a "Workspaces" link. On the right side, a modal window titled "Take a shortcut to sending requests" is open, showing a request configuration for a GET method to https://postman-echo.com/get. The modal includes tabs for Params, Auth, Headers, Body, Pre-req, Tests, Settings, and Cookies, along with a "Send" button and a "Query Params" table.

**onepoint.**

# Création du Workspace

Créer un espace de travail pour chacun de vos projets

The screenshot shows the Postman application interface. On the left, there's a sidebar with various workspace and API-related options. The main area shows a search bar and a 'Create Workspace' button. A dropdown menu is open over the 'Create Workspace' button, with the first item being 'morn'. A red circle with the number '1' is on the 'Create Workspace' button. A callout box labeled 'Création du nouveau Workplace' points to the dropdown menu.

*Retrouver ici la liste de vos espaces de travail*

*1*

*Création du nouveau Workplace*

*2*

*Saisir le nom du projet*

*3*

*Choisir Personal*

*4*

*Valider*

The right side of the interface shows the 'Create workspace' dialog. It has fields for 'Name' (containing 'Formation Postman'), 'Summary' (empty), and 'Visibility' settings. Under 'Visibility', the 'Personal' option is selected, indicated by a red circle with '3'. A callout box labeled 'Choisir Personal' points to the 'Personal' radio button. At the bottom of the dialog are 'Create Workspace' and 'Cancel' buttons. A callout box labeled 'Valider' points to the 'Create Workspace' button.

**onepoint.**

# Création d'une collection

Créer une collection (ensemble de requêtes présentant avec une cohérence)

The screenshot shows the Postman interface with two main windows. The left window displays a collection named 'My first collection' containing two folders: 'First folder inside collection' and 'Second folder inside collection'. The right window shows the process of creating a new collection. A tooltip 'Création d'une nouvelle collection' points to the 'New' button in the top right of the main window. Another tooltip 'Saisir le nom de la collection' points to the input field where 'Exercice - Requetes Simple' is typed. A third tooltip 'Ou clic droit sur la collection, et choisir Rename' points to the context menu of an existing collection. The sidebar on the left includes links for Home, Workspaces, API Network, Explore, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History.

*Création d'une nouvelle collection*

*Saisir le nom de la collection*

*Ou clic droit sur la collection, et choisir Rename*

Create a collection for your requests  
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create Collection

New Collection

This collection is empty  
Add a request to start working.

Search Postman

New Collection

Exercice - Requetes Simple

Auth Pre-req. Tests Variables Runs

Type No Auth

# Structure

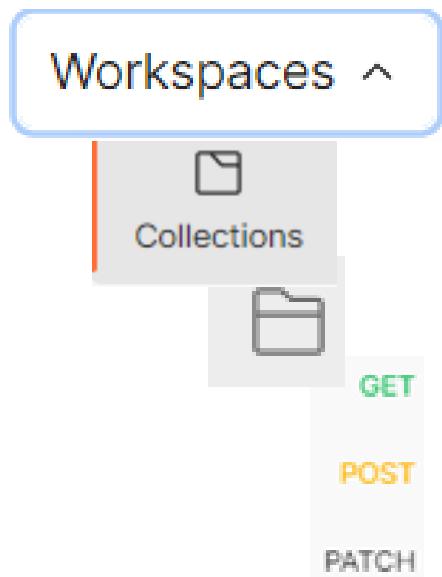
## Organisation des requêtes :

- Chaque Workspace peut contenir plusieurs collections
- Chaque collection peut être organisée avec des dossiers
- Les requêtes peuvent être ajouter dans un dossier ou direction dans la collection

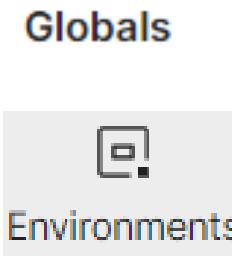
## Organisation des variables

- Les variables peuvent globales ou liées à une collection.
- Les variables peuvent aussi être organiser dans les environnements pour simplifier la maintenance.

## Requêtes



## Variables



# Exercice 0 : Préparation

- 4.      Préparation
  - 4.1.    Création d'un Workspace
  - 4.2.    Création de collection
  - 4.3.    Importer / Exporter une collection
  - 4.4.    Création d'une requête API
  - 4.5.    Liste des requêtes FlightsAPI utilisées

Documentation de l'API Flights :

<https://app.swaggerhub.com/apis/imhah-hassan/PyFlightsApi/Dec22#/>

# 3

---

## Création des requêtes simples

<https://app.swaggerhub.com/apis/imhah-hassan/PyFlightsApi/Dec22#/>

- Consultation (GET)
- Création (POST)
- Mise à jour (PATCH)
- Suppression (DELETE)

# Requête Rest dans Postman

GET <https://api.taiga.io/api/v1/issues?project=404046>

Params • Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 7 hidden

KEY	VALUE
<input checked="" type="checkbox"/> Content-Type	application/json

Params • Authorization Headers (8) Body Pre-request Script

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> project	404046

Params Authorization Headers (10)

Body •

none form-data x-www-form-urlencoded

```
1 {
2   ...
3   "password": "Hassan$2022",
4   "type": "normal",
5   "username": "h.imhah"
6 }
```

Params Authorization Headers (10) Body • Pre-request Script

```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4
5 pm.test("Body matches auth_token string", function () {
6   pm.expect(pm.response.text()).to.include("auth_token");
7 });
8
9 pm.test("Verify user name", function () {
10  var jsonData = pm.response.json();
11  pm.expect(jsonData.username).to.eql("h.imhah");
12 });
```

GET ht

GET

POST

PUT

PATCH

DELETE

COPY

Params Authorization Headers (11) Body • Pre-request Script •

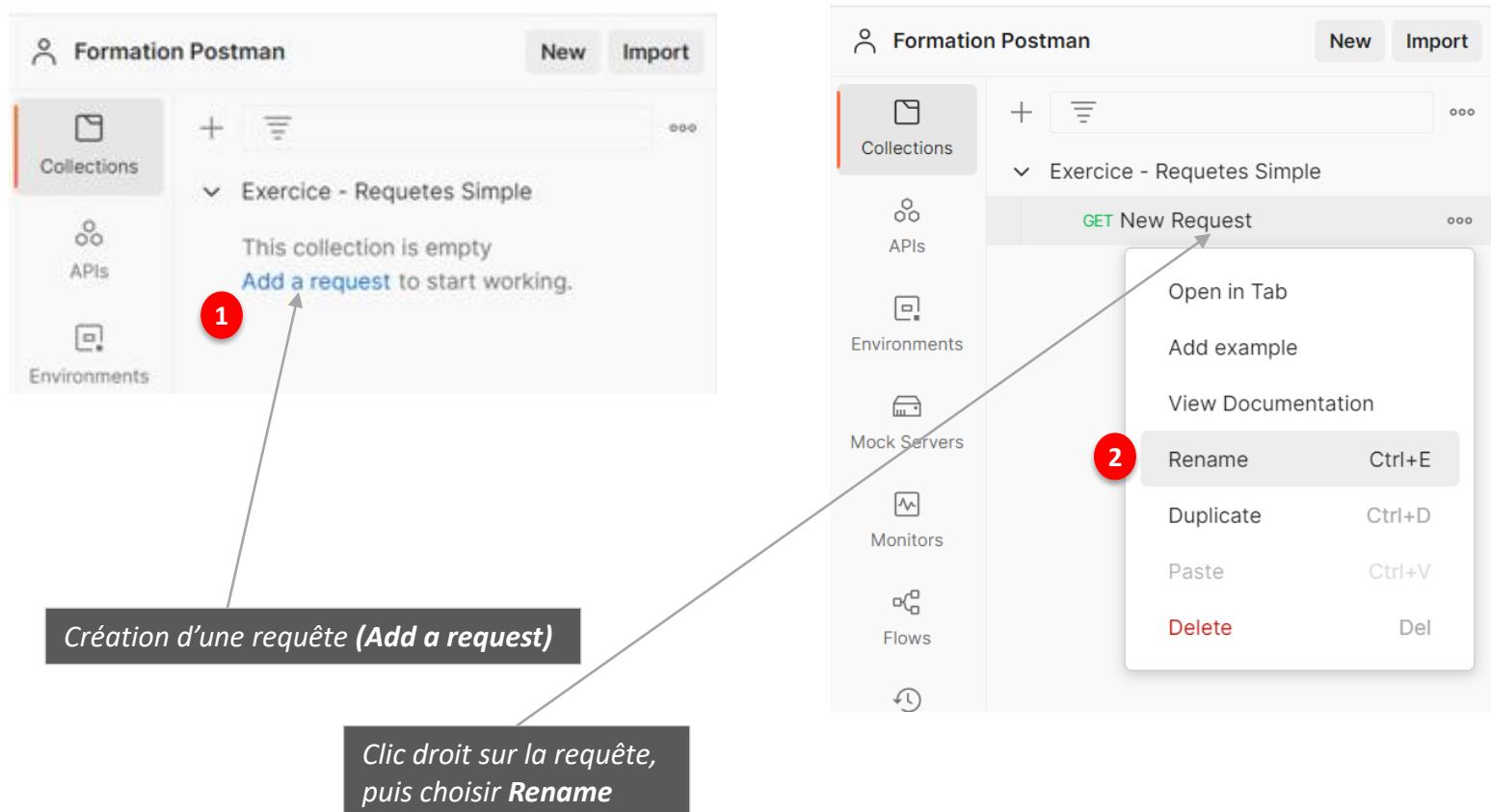
```
1 var issuesCount = pm.environment.get ("issuesCount")
2 console.log (pm.environment.get ("issuesCount"));
3
4 if (issuesCount > 1) {
5   postman.setNextRequest ("issues");
6 }
```

Tests •

onepoint.

# Création d'une requête API

Lancer une requête pour lire les informations d'un vol via son numéro.



# Création d'une requête simple (GET)

*Sélectionner la méthode*

*Coller l'url de la ressource*

The screenshot shows the Postman interface with the following details:

- Method:** GET (highlighted with red circle 1)
- URL:** http://localhost:5000/flights\_api/v1/resources/Flights/{FlightNumber} (highlighted with red circle 2)
- Headers:** (6) (highlighted with red circle 3)
- Send:** button (highlighted with red circle 4)

*Saisir l'identifiant du vol*

*Lancer la requête*

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:5000/flights\_api/v1/resources/Flights/14303 (highlighted with red circle 3)
- Send:** button (highlighted with red circle 4)

N'oubliez pas de sauvegarder (**Save**) (ou **CRL + S**) avant de lancer votre requête (**Send**)

# Résultat d'une requête simple (GET)

En retour une réponse avec un code 200 (ok) et les informations en format JSON du vol.

The screenshot shows the Postman interface with a successful API call. The left sidebar has 'Formation Postman' selected under 'Collections'. The main area shows a 'GET Rechercher d'un vol' request to 'http://localhost:5000/flights\_api/v1/resources/Flights/14303'. The 'Body' tab displays the JSON response:

```
1 "Airline": "NW",
2 "ArrivalCity": "Frankfurt",
3 "ArrivalTime": "07:41 PM",
4 "DepartureCity": "Denver",
5 "DepartureTime": "06:57 PM",
6 "FlightNumber": 14303,
7 "Price": 100.8,
8 "SeatsAvailable": 250,
9 "DayOfWeek": "Friday"
```

A callout arrow points from the text 'Réponse au format json' to the JSON response body. Another callout arrow points from the text 'HTTP Status Codes' to the status code '200 OK' in the response summary.

**HTTP Status Codes**

Code	Description
1XX	INFORMATIONAL
2XX	SUCCESS
3XX	REDIRECTION
4XX	CLIENT ERROR
5XX	SERVER ERROR

# Création d'une requête (GET)

Rechercher un vol par ville de départ, ville d'arrivée et par date

Faire un clic droit sur la collection (Exercice – Requêtes Simple), puis choisir le menu **Add Request**

Renommer la requête

The screenshot shows the Postman interface with the following details:

- Left sidebar:** Collections (selected), APIs, Environments.
- Top bar:** Formation Postman, New, Import, Overview, GET Rechercher d'un vol, GET Recherche par ville de dé..., No Environment.
- Middle section:** Title: Exercice - Requetes Simple / Recherche par ville de départ, d'arrivée et par date. Method: GET, URL: Enter request URL. Params tab is selected.
- Bottom section:** Query Params table with columns KEY, VALUE, DESCRIPTION, and Bulk Edit.

Sélectionner la méthode (GET) et coller l'url de la ressource

The screenshot shows the Postman interface with the following details:

- Left sidebar:** Collections (selected), APIs, Environments.
- Top bar:** Formation Postman, New, Import, Overview, GET Rechercher d'un vol, GET Recherche par ville de départ, d'arrivée et par date.
- Middle section:** Method: GET, URL: http://localhost:5000/flights\_api/v1/resources/Flights?DepartureCity=Paris&ArrivalCity=Denver&Date=2022-12-08. The URL contains a red box around the query parameters.
- Bottom section:** Params tab is selected, showing a table with columns KEY, VALUE, DESCRIPTION, and Bulk Edit. It contains three rows:
  - DepartureCity: Paris (highlighted with a red box)
  - ArrivalCity: Denver
  - Date: 2022-12-08

Pour modifier une requête vous pouvez utiliser l'onglet **Params** au lieu de modifier URL,  
Par exemple, remplacer la valeur Paris par la valeur London pour le paramètre **DepartureCity**

# Création d'une requête (POST)

## Prendre une réservation

Créer une requête et la nommer. Sélectionner la méthode (POST) et coller l'url de la ressource

The screenshot shows the Postman application interface. On the left, there's a sidebar with a '+' icon, a 'Save' button, and a 'More' button. Below these are several items under a 'Exercice - Requetes Simple' section: 'Rechercher d'un vol' (GET), 'Recherche par ville de départ,...' (GET), and 'Prendre une reservation' (POST). The 'Prendre une reservation' item is highlighted with a red box. The main workspace shows a 'POST' method selected, pointing to the URL 'http://localhost:5000/flights\_api/v1/resources/FlightOrders'. Below the URL are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. A 'Cookies' tab is also visible. Under the 'Params' tab, there's a table titled 'Query Params' with columns 'KEY', 'VALUE', 'DESCRIPTION', and 'Bulk Edit'. There is one row in the table with 'Key' and 'Value' fields filled. The right side of the interface has various icons for saving, editing, and deleting requests.

Après la sauvegarde, la requête est tagguée comme étant un requête de type POST

# Création d'une requête (POST)

Envoyer les informations de la nouvelle réservation

*Cliquer sur l'onglet **Body** pour accéder aux formats disponibles*

*Choisir le format **Raw** et préciser **JSON***

Exercice - Requêtes Simple / Prendre une réservation

POST http://localhost:5000/flights\_api/v1/resources/FlightOrders

Params Authorization Headers (8) **Body** \* Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 "Class": "Economy",  
2 "CustomerName": "bruno",  
3 "DepartureDate": "2022-12-18",  
4 "FlightNumber": "15113",  
5 "NumberOfTickets": "2"
```

Saisir les informations de la réservation

*Enregistrer la requête et envoyer la : Réponse est retourné avec l'identifiant de la réservation*

The screenshot shows the Postman interface with the following details:

- Header bar: Body, Cookies, Headers (4), Test Results, 200 OK, 45 ms, 186 B, Save Response.
- Toolbars: Pretty, Raw, Preview, Visualize, JSON dropdown, copy, search.
- Response body:

```
1 "OrderNumber": 81,  
2 "TotalPrice": 277.2
```

An arrow points from the "OrderNumber" value in the JSON response to the "OrderNumber" field in the Headers section of the request builder.

**onepoint.**

# Création d'une requête (PATCH)

Modifier une réservation existante

Appeler la réservation à modifier (81)

Sélectionner la méthode

Exercice - Requetes Simple / Modifier une reservation

PATCH http://localhost:5000/flights\_api/v1/resources/FlightOrders/81

Params Authorization Headers (8) Body **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1  
2     "Class": "Economy",  
3     "CustomerName": "bruno",  
4     "DepartureDate": "2022-12-18",  
5     "FlightNumber": "15113",  
6     "NumberOfTickets": "5".  
7

Saisir les informations de la réservation dont le nouveau nombre de tickets

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

1  
2     "Class": "Economy",  
3     "CustomerName": "bruno",  
4     "DepartureDate": "2022-12-18 18:15",  
5     "FlightNumber": 15113,  
6     "NumberOfTickets": 5,  
7     "OrderNumber": 81,  
8     "TotalPrice": 227.0

200 OK 40 ms 314 B Save Response

Nouvelle valeur prise en compte

# Création d'une requête (DELETE)

Supprimer une réservation existante

The screenshot shows the Postman interface for creating a DELETE request to delete a flight order.

**Left Sidebar:** Shows a list of exercises under "Exercice - Requetes Simple". The "DEL Supprimer une reservation" item is highlighted, with a callout box labeled "Sélectionner la méthode".

**Request URL:** http://localhost:5000/flights\_api/v1/resources/FlightOrders/81

**Method:** DELETE

**Params Tab:** Contains a table for Query Params:

KEY	VALUE	DESCRIPTION
Key	Value	Description

**Body Tab:** Contains the response body:

```
1 true
2 "true": "Order deleted 81 "
```

**Status:** 200 OK

**Callout Boxes:**

- "Appeler la réservation à modifier (81)" points to the URL field.
- "Confirmation de la suppression" points to the response body.

# Consulter la log d'exécution

Consultation des logs d'exécution en cliquant sur Console

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections', 'APIs', and 'Environments' sections. The 'Environments' section is selected. In the main area, a collection named 'Exercice - Requetes Simple' is expanded, showing several requests: 'Rechercher d'un vol' (GET), 'Recherche par ville de départ,...' (GET), 'Prendre une reservation' (POST), 'Modifier une reservation' (PATCH), and 'Supprimer une reservation' (DELETE). The 'Supprimer une reservation' request is selected, showing its details: method 'DELETE', URL 'http://localhost:5000/flights\_api/v1/resources/FlightOrders/81', and various tabs like Params, Headers, Body, etc. Below this, the response body is displayed in JSON format: "true": "Oder deleted 81)". At the bottom, there's a 'Console' tab (which is highlighted with a red box) and a 'Logs' section showing a list of recent API calls. A large arrow points from the 'Console' tab to the 'Clear' button in the logs section. Another arrow points from the 'Logs' section to a text box containing the French sentence 'Pour effacer le contenu cliquer sur Clear.'.

Pour afficher le détail, cliquer sur la requête

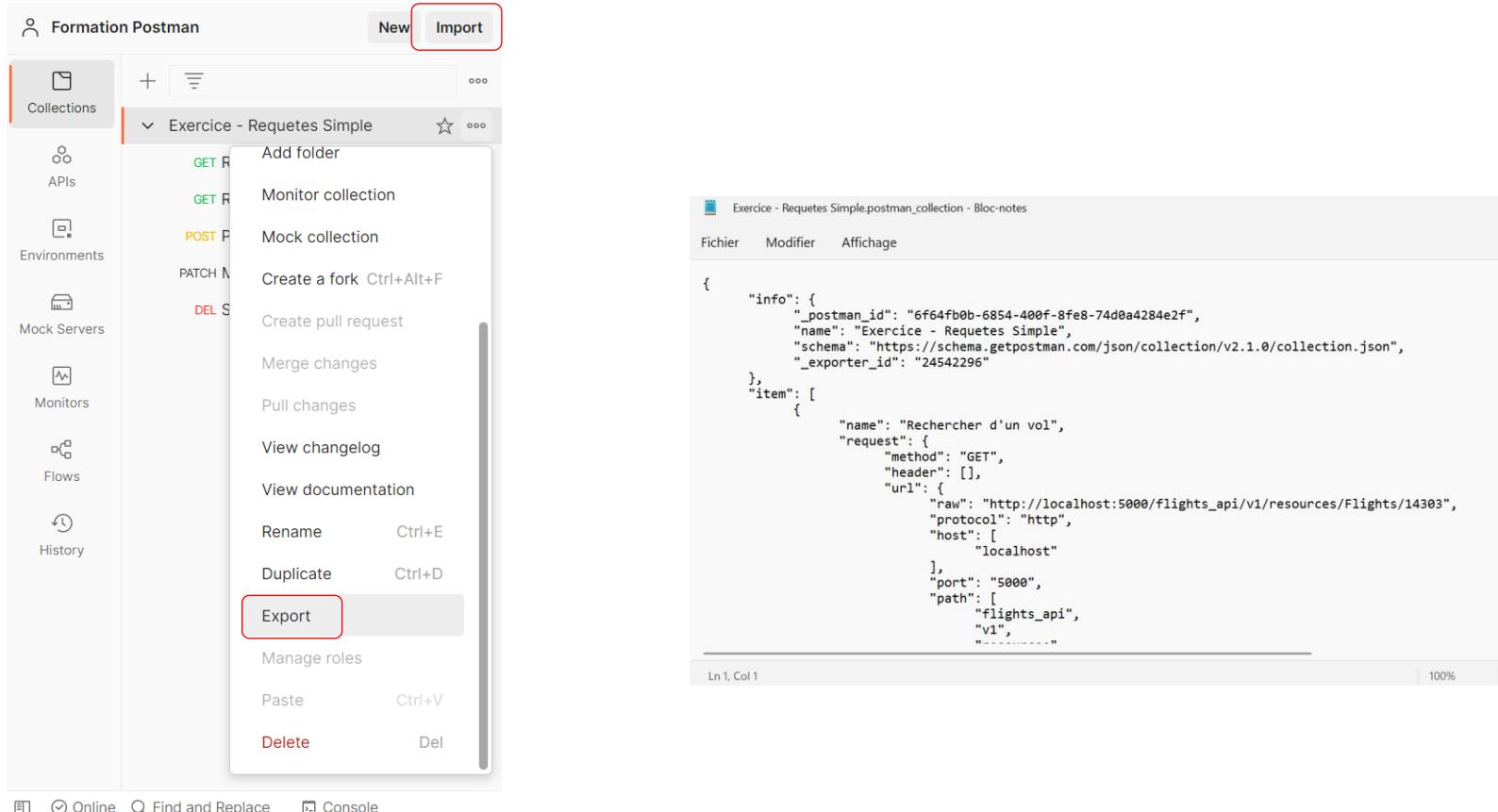
Pour effacer le contenu cliquer sur **Clear**.

**onepoint.**

# Importer / Exporter une collection

Pour exporter une collection, faire un clic droit sur la collection qu'on souhaite exporter,  
Cliquer sur le bouton **Export** dans la popup qui s'afficher,  
Choisir l'emplacement du fichier JSON et confirmer l'enregistrement..

Pour l'import d'une collection, cliquer sur le bouton « Import », puis naviguer jusqu'à votre fichier de collection



# Exercice 1 : Rechercher un vol

## Utiliser le tutorial

5. Exercice 1 : Rechercher un vol
  - 5.1. Requête GetFlight : recherche par numéro
  - 5.2. Requête GetFlights : recherche par ville de départ, d'arrivée et par date
  - 5.3. Consulter le log d'exécution



4

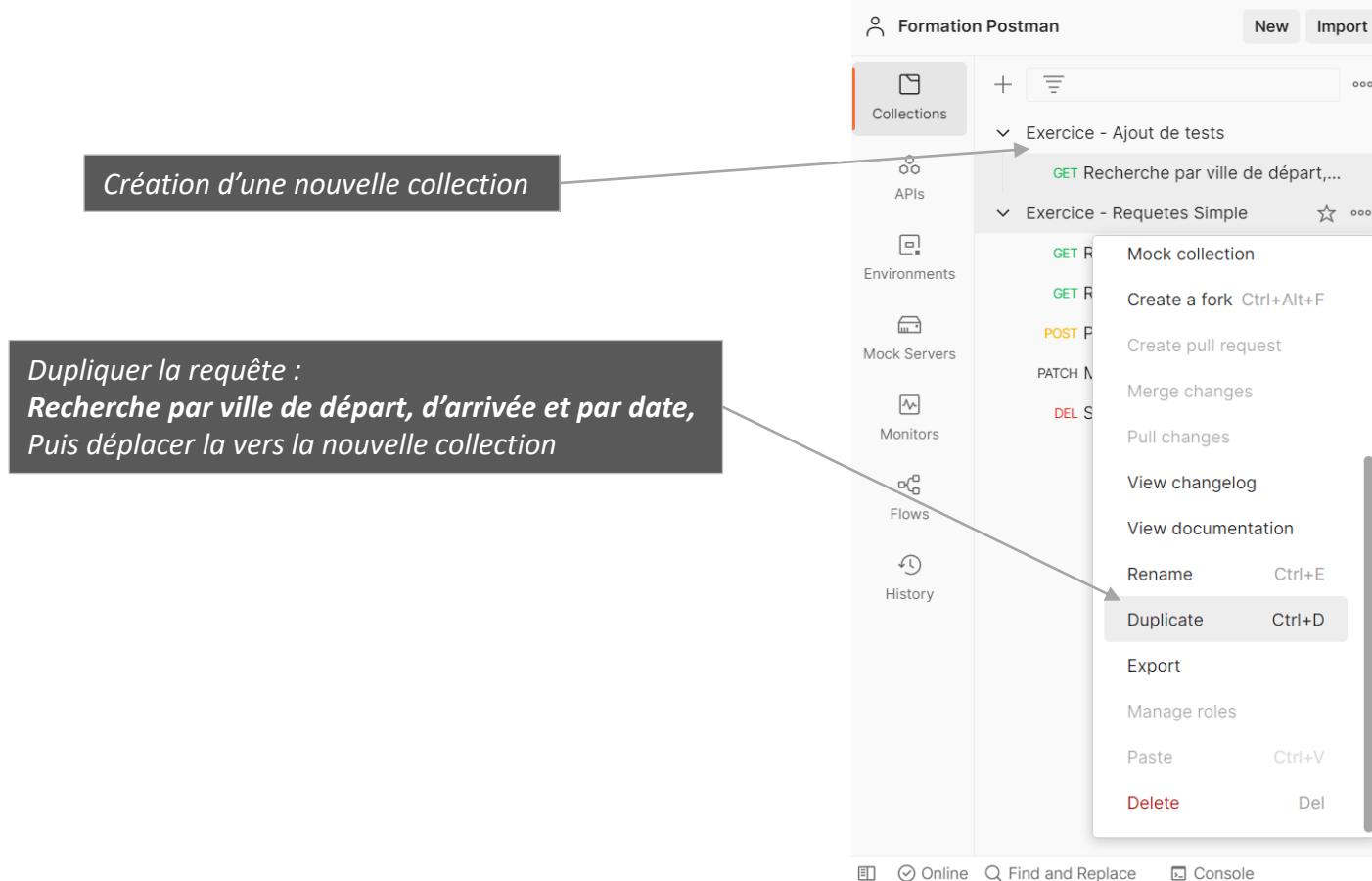
---

# Ajout des vérifications

<https://learning.postman.com/docs/writing-scripts/script-references/test-examples/>

# Vérifier un code retour

Préparation d'une nouvelle collection



Note : On peut également dupliquer une collection

# Vérifier un code retour

Mise au point d'un premier test

The screenshot shows the Postman interface for creating a test script. On the left, the sidebar shows collections like 'Exercice - Ajout de tests' and 'Exercice - Requetes Simple'. The main area is titled 'Exercice - Ajout de tests / Recherche par ville de départ, d'arrivée et par date Copy'. A GET request is defined with the URL: `http://localhost:5000/flights_api/v1/resources/Flights?DepartureCity=Paris&ArrivalCity=Denver&Date=...`. The 'Tests' tab is selected, highlighted by a red circle labeled '1'. Below it, a snippet of JavaScript code is shown:

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(200);  
3 });
```

A callout box with the text 'Le code JavaScript est ajouté automatiquement' points to this code. To the right, the response status is shown as 'Status code: Code is 200' (red circle '2'). Below the status, the response body is displayed as JSON:

```
[{"Airline": "AF", "ArrivalCity": "Denver", "ArrivalTime": "10:30 AM", "DepartureCity": "Paris", "DepartureTime": "08:00 AM", "FlightNumber": 17105, "Price": 139.47, "SeatsAvailable": 250, "DayOfWeek": "Thursday"}, {"Airline": "AF", "ArrivalCity": "Denver", "ArrivalTime": "12:54 PM", "DepartureCity": "Paris", "DepartureTime": "10:24 AM", "FlightNumber": 17109, "Price": 143.47, "SeatsAvailable": 250, "DayOfWeek": "Thursday"}]
```

At the bottom, there are buttons for 'Find and Replace', 'Console', 'Cookies', 'Capture requests', 'Runner', 'Trash', and 'Save Response'.

Notes : N'oublier pas de sauvegarder

Un snippet (« **extrait** ») est une partie d'un code source pouvant être copié-collé dans un modèle plus large.

# Vérifier un code retour

Exécutions de la requête suivie de la vérification

1 - Cas passant :

The screenshot shows the Postman interface with a successful test result. On the left, the sidebar lists a collection named "Exercice - Ajout de tests" with a test case "GET Recherche par ville de départ,...". The main area shows a GET request to "http://localhost:5000/flights\_api/v1/resources/Flights?DepartureCity=Paris&ArrivalCity=Denver&Date=2022-12-08". The "Tests" tab contains a JavaScript assertion:

```
1 pm.test("Status code is 200", function () {  
2     pm.response.to.have.status(200);  
3 });
```

The "Test Results" section shows one pass: "Status code is 200".

La vérification est au statut **Pass** →

2 - En modifiant le nom de la ville par une ville non référencée, le test devient non passant :

The screenshot shows the Postman interface with a failed test result. The setup is identical to the first test, but the URL now includes an invalid city name: "http://localhost:5000/flights\_api/v1/resources/Flights?DepartureCity=Parisparis&ArrivalCity=Denver&Date=2022-12-08". The "Params" tab shows the incorrect values:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
DepartureCity	Parisparis			
ArrivalCity	Denver			
Date	2022-12-08			

The "Test Results" section shows a failure: "Status code is 200 | AssertionError: expected response to have status code 200 but got 500".

La vérification est au statut **Fail** →

# Vérifier le contenu d'une balise Json

Objectif :

Vérifier que la première balise FlightNumber a pour valeur 17105 est présent

Ajouter les Snippets *Response body : Contains String* et *Response body :Json value check*

The screenshot shows the Postman interface with a test script for a flight search API. The test script uses JavaScript assertions to check the response status, presence of a specific tag, and the value of the first FlightNumber.

```
1 pm.test("Le code retour est 200", function () {  
2     pm.response.to.have.status(200);  
3 };  
4  
5 pm.test("La réponse contient le tag FlightNumber", function () {  
6     pm.expect(pm.response.text()).to.include("FlightNumber");  
7 };  
8  
9 pm.test("Le premier vol est le 17105", function () {  
10    var jsonData = pm.response.json();  
11    pm.expect(jsonData[0].FlightNumber).to.eql(17105);  
12 };  
13 };
```

Annotations on the right side of the screenshot:

- 1 Status code: Code is 200
- 2 Response body: Contains string
- 3 Response body: JSON value check
- 4 Test scripts are written in JavaScript, and are run after the response is received. Learn more about [tests scripts](#)
- 5 Response body: Is equal to a string
- 6 Response headers: Content-Type header

Annotations on the left side of the screenshot:

- Les libellés des tests ont été personnalisé
- 1er vol
- 2ème vol

Bottom navigation bar:

- Q Find and Replace
- Console
- Cookies
- Capture requests
- Runner
- Trash

# Vérifier le contenu d'une balise Json

Résultat :

The screenshot shows the Postman application interface with the following details:

- Left Sidebar:** Shows a tree view of collections:
  - Exercice - Ajout de tests
    - GET Recherche par ville de départ,...
  - Exercice - Requetes Simple
    - GET Rechercher d'un vol
    - GET Recherche par ville de départ,...
    - POST Prendre une reservation
    - PATCH Modifier une reservation
    - DEL Supprimer une reservation
- Request Details:** GET request to `http://localhost:5000/flights_api/v1/resources/Flights?DepartureCity=Paris&ArrivalCity=Denver&Date=2023-01-01`.
- Tests Tab:** Contains three test scripts:

```
1 pm.test("Le code retour est 200", function () {  
2     pm.response.to.have.status(200);  
3 });  
4  
5 pm.test("La réponse contient le tag FlightNumber", function () {  
6     pm.expect(pm.response.text()).to.include("FlightNumber");  
7 });  
8  
9 pm.test("Le premier vol est le 17105", function () {  
10    var jsonData = pm.response.json();  
11    pm.expect(jsonData[0].FlightNumber).to.eql(17105);  
12});
```
- Test Results:** 3/3 tests passed.
  - PASS Le code retour est 200
  - PASS La réponse contient le tag FlightNumber
  - PASS Le premier vol est le 17105
- Response Summary:** 200 OK | 41 ms | 1.4 KB | Save Response

# Synthèse des tests

Test	Description
pm.response.to.have.status(200);	Le statut http est 200
pm.expect(pm.response.code).to.be.oneOf([201,202]);	Le statut http est parmi ces valeurs
pm.expect(pm.response.responseTime).to.be.below(100);	Le temps de réponse en ms
pm.response.to.have.header('X-Cache');	Le header existe
pm.expect(pm.response.headers.get('X-Cache')).to.eql('HIT');	Le header a cette valeur
pm.expect(pm.cookies.has('sessionId')).to.be.true;	Le cookie existe
pm.expect(pm.cookies.get('sessionId')).to.eql('ad3se3ss8sg7sg3');	Le cookie a cette valeur
pm.response.to.have.body("OK"); pm.response.to.have.body('{"success":true}');	Le corps de la réponse contient une valeur
pm.expect(pm.response.text()).to.include('Order placed.');	Le texte de la réponse contient une valeur
const response = pm.response.json(); pm.expect(response.age).to.eql(30);	Parser la réponse au format json et vérifier une valeur.

# Exercice 2 : Les tests

## Utiliser le tutorial

- 6. Exercice 2 : Les tests
  - 6.1. Valider le code retour de la requête
  - 6.2. Vérifier la présence d'un texte dans la réponse
  - 6.3. Vérifier le contenu d'une balise JSON



5

---

# Enchainement de plusieurs requêtes

<https://learning.postman.com/docs/writing-scripts/script-references/variables-list/>

# Préparation d'une collection dédiée

Objectif: Simuler le scénario d'utilisation suivant

- Rechercher les vols disponibles avec une ville de départ, une ville d'arrivée et une date  
Selectionner le 1<sup>er</sup> vol disponible
- Prendre une réservation sur ce vol
- Contrôler l'existence de la réservation

Création d'une nouvelle collection en dupliquant les requêtes suivantes

▼ Exercice - Variables Globales

**GET** Recherche par ville de départ, d'arrivée et par date

**POST** Prendre une reservation

**GET** Rechercher une réservation par numéro

Note : la dernière requête n'a pas encore été vu dans cette présentation  
GET [http://localhost:5000/flights\\_api/v1/resources/FlightOrders/{order}](http://localhost:5000/flights_api/v1/resources/FlightOrders/{order})

# Enregistrer dans une variable globale

1<sup>er</sup> requête : Reprendre la requête *Recherche par ville de départ, d'arrivée et par date*

Ajouter le snippet *Set a global variable*,  
puis affecter la variable Numéro\_de\_vol avec la valeur du numéro de vol (jsonData[0].FlightNumber)

The screenshot shows the Postman interface with a test script in the 'Tests' tab of a GET request. The script uses the Mocha.js framework. It first checks if 'FlightNumber' is present in the response. Then, it defines a function to check if the first flight's number is 17105 and if a global variable 'Numero\_de\_vol' is created. Finally, it sets the global variable 'Numero\_de\_Vol' to the flight number from the first item of the JSON response. The 'Body' tab shows the response body and the 'Test Results' tab shows three successful passes.

```
5 pm.test("La réponse contient le tag FlightNumber", function () {  
6 | pm.expect(pm.response.text()).to.include("FlightNumber");  
7 });  
9 pm.test("Le premier vol est le 17105 et la variable globale  
'Numero_de_vol' est créée", function () {  
10 | var jsonData = pm.response.json();  
11 | pm.expect(jsonData[0].FlightNumber).to.eql(17105);  
12 |  
13 | pm.globals.set("Numero_de_Vol", jsonData[0].FlightNumber);  
14 |});  
15 };
```

Body Cookies Headers (4) Test Results (3/3)  
All Passed Skipped Failed

PASS Le code retour est 200  
PASS La réponse contient le tag FlightNumber  
PASS Le premier vol est le 17105 et la variable globale 'Numero\_de\_vol' est créée

Note : Attention, déposer ce snippet dans *Response body :Json value check*, càd juste avant son accolade fermante

# Utiliser la variable enregistrée

2<sup>ème</sup> requête : Reprendre la requête **Prendre une reservation** et ajouter la variable globale

Sur l'onglet **Body**, positionner la variable globale “{{Numero\_de\_vol}}” pour le champ **FlightNumber**

The screenshot shows the Postman interface with a left sidebar containing project navigation and a main workspace for a collection named "Exercice - Variables Globales / Prendre une reservation". In the workspace, a POST request is configured with the URL "http://localhost:5000/flights\_api/v1/resources/FlightOrders". The "Body" tab is active, showing a JSON payload:

```
1 {  
2   "Class": "Economy",  
3   "CustomerName": "bruno",  
4   "DepartureDate": "2022-12-18",  
5   "FlightNumber": "{{Numero_de_vol}}",  
6   "NumberOfTickets": "2"  
7 }
```

Annotations with arrows point from the text "Sur l'onglet Body, positionner la variable globale “{{Numero\_de\_vol}}” pour le champ FlightNumber" to the "Body" tab and the "FlightNumber" field in the JSON payload.

# Utiliser la variable enregistrée

2<sup>ème</sup> requête : Sur la requête **Prendre une reservation**, enregistrer le numéro de réservation

Sur l'onglet Tests, ajouter quelques snippets (tests) et personnaliser-les, comme suit

The screenshot shows the Postman interface with a left sidebar listing exercises and a main panel for an exercise titled "Exercice - Variables Globales / Prendre une reservation". The request method is set to POST, and the URL is http://localhost:5000/flights\_api/v1/resources/FlightOrders. The "Tests" tab is selected, showing the following snippet of code:

```
1 pm.test("Le code retour est 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("La réponse contient le tag OrderNumber", function () {
6     pm.expect(pm.response.text()).to.include("OrderNumber");
7 });
8
9 pm.test("Le numéro de réservation est supérieur à 80", function () {
10    var jsonData = pm.response.json();
11    pm.expect(jsonData.OrderNumber).to.be.greaterThan(80);
12
13    //Ce commentaire pour la nouvelle variable globale
14    pm.globals.set("Num_de_reservation", jsonData.OrderNumber);
15
16 });
17
```

Annotations on the right side explain the code:

- Status code: Code is 200**: Points to the first test case.
- Response body: Contains string**  
(la réponse doit contenir le numéro d'ordre): Points to the second test case.
- Response body: JSON value Check**  
Vérifier que le numéro d'ordre est plus grand que: Points to the third test case.
- Set a global variable**  
(la nouvelle variable global est Num\_de\_reservation): Points to the final line of the code where a global variable is set.

# Utiliser la variable enregistrée

2<sup>ème</sup> requête : Tester la requête *Prendre une reservation* seule

Cliquer sur l'icone Environnement  
pour visualiser les variables globales

The screenshot shows the Postman interface with the 'Environment' tab selected. A global variable 'Numero\_du\_Vol' is defined with an initial value of '17105' and a current value of '17105'. A tooltip at the bottom left explains how to use variables to reuse values and protect sensitive data.

Lancer l'exécution de la requête

The screenshot shows the Postman interface with the 'Test Results' tab selected. Three tests have passed: 'Le code retour est 200', 'La réponse contient le tag Ordernumber', and 'Le numéro de réservation est supérieur à 80'.

Vérifier la présence d'une valeur ou renseigner la (champ éditable)

# Utiliser la variable enregistrée

3<sup>ème</sup> requête : Prendre la requête **Rechercher une réservation par numéro** et ajouter la variable globale

Ajouter la variable globale dans l'url de la requête

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, and Monitors. The main area displays a collection named "Formation Postman". Inside this collection, there is a folder named "Exercice - Variables Globales" which contains three items: "GET Recherche par ville de départ,...", "POST Prendre une reservation", and "GET Rechercher une réservation p...". The third item is currently selected. The URL field for this request is highlighted with a red box and contains the placeholder "http://localhost:5000/flights\_api/v1/resources/FlightOrders/{{Num\_de\_reservation}}". A grey arrow points from the text "Ajouter la variable globale dans l'url de la requête" to this URL field.

# Utiliser la variable enregistrée

Exécuter l'enchainement complet des tests

The screenshot shows the Postman interface with the following annotations:

- An arrow points to the "Runs" tab in the top navigation bar, with the text: "Sélectionner l'onglet **Runs** de la collection".
- An arrow points to the "Run Collection" button at the bottom of the main panel, with the text: "Cliquer sur le bouton **Run Collection**".
- An arrow points to the "Run Exercise – Variables Globales" button in the bottom right corner of the main panel, with the text: "(Dé)Sélectionner les requêtes à exécuter".
- An arrow points to the "Run Configuration" section on the right, with the text: "Choisir comment exécuter votre collection". It includes options for running manually, scheduling runs, or automating via CLI.
- An arrow points to the "Run" button at the bottom right of the configuration panel, with the text: "Run **Exercise – Variables Globales**".

**onepoint.**

# Utiliser la variable enregistrée

## Résultats de l'exécution

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, APIs, Environments, Mock Servers, Monitors, Flows, History.
- Top Bar:** Formation Postman, New, Import, Overview, Exercice - Variables Globales (selected), Run Again, Automate Run, New Run, Export Results, No Environment.
- Run Results Section:**
  - Run Details:** Run on Today, 08:01:54 · View all runs.
  - Table:** Source (Runner), Environment (none), Iterations (1), Duration (404ms), All tests (8), Avg. Resp. Time (20 ms).
  - Test Summary:** All Tests (Passed: 8, Failed: 0, Skipped: 0). View Summary.
  - Test Log:** A list of test cases with their status (Pass or Fail) and assertions.
    - GET Recherche par ville de départ... (Pass) - Le code retour est 200.
    - POST Prendre une réservation (Pass) - La réponse contient le tag FlightNumber.
    - GET Rechercher une réservation p... (Pass) - Le premier vol est le 17105 et la variable globale 'Numero\_de\_vol' est créée.
    - POST Prendre une réservation (Pass) - Le code retour est 200.
    - GET Rechercher une réservation p... (Pass) - La réponse contient le tag Ordernumber.
    - GET Rechercher une réservation p... (Pass) - Le numéro de réservation est supérieur à 80.
    - GET Rechercher une réservation p... (Pass) - Le code retour est 200.
    - GET Rechercher une réservation p... (Pass) - OrderNumber est bien présent.
- Console Section:** Shows network logs with expandable sections for Request Headers, Request Body, Response Headers, and Response Body.

Cliquer sur l'onglet **Console**  
Déplier l'arborescence des logs pour voir les détails

# Exercice 3 : Enchaîner plusieurs requêtes avec les variables globales

7. Exercice 3 : Enchaîner plusieurs requêtes avec les variables globales
  - 7.1. Enregistrer la donnée FlightNumber dans une variable globale
  - 7.2. Utiliser la variable dans la requête BookFlight et enregistrer le numéro de réservation
  - 7.3. Afficher le contenu des variables globales
  - 7.4. Exécuter l'enchainement complet des tests

# 6

---

## Utilisation des variables

<https://learning.postman.com/docs/sending-requests/variables/>

# Gestion des variables d'environnement

Objectif : Faciliter la maintenance des tests

Création d'une nouvelle collection en dupliquant la collection de l'exercice des variables Globales

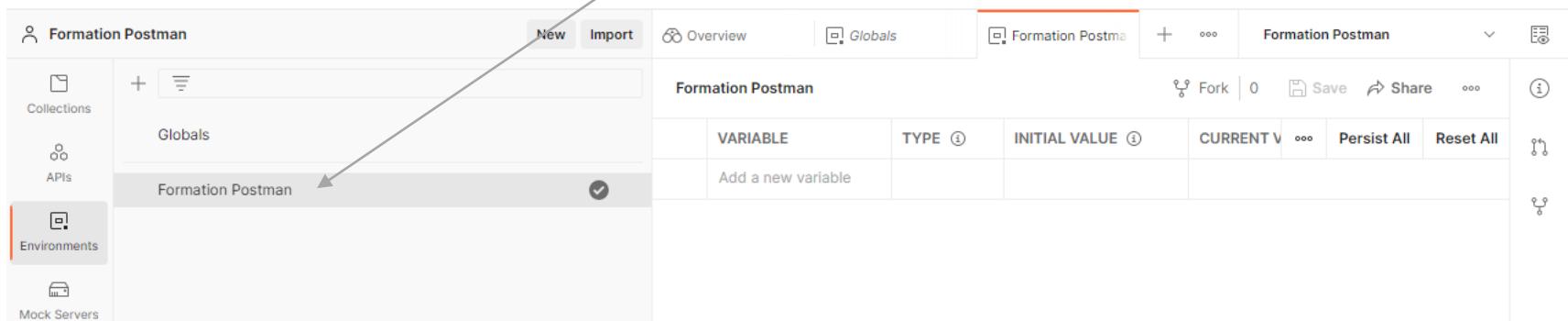
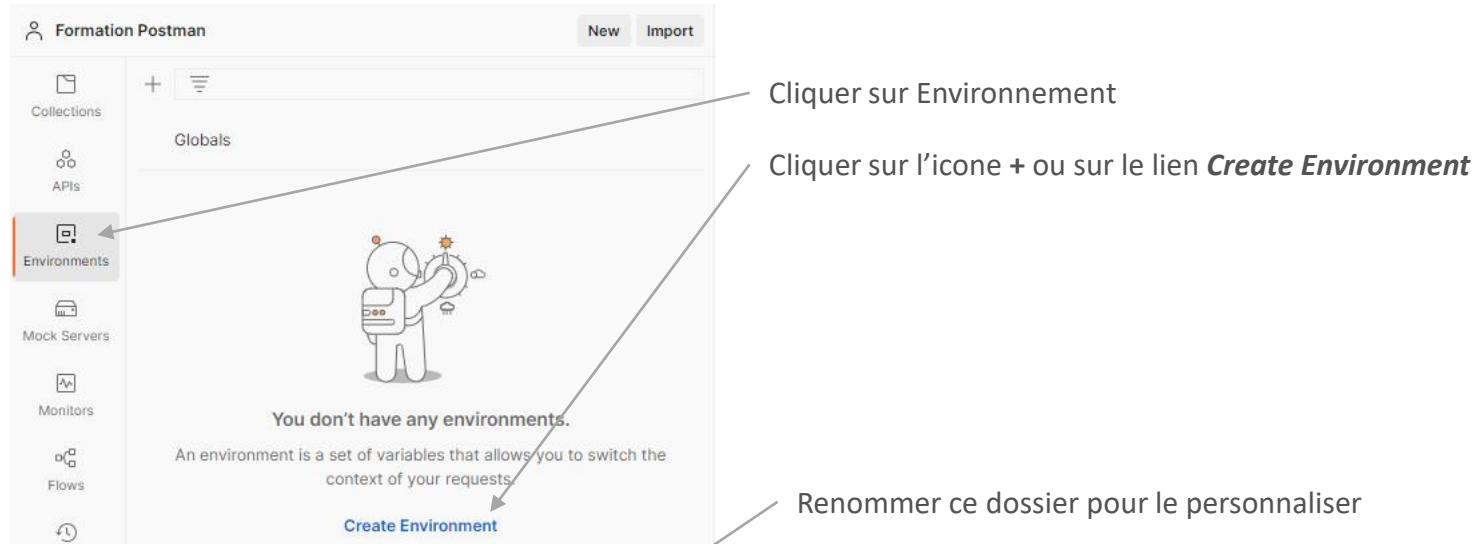
The screenshot shows the Postman application interface. The title bar says 'Formation Postman'. On the left, there's a sidebar with icons for Collections, APIs, Environments, Mock Servers, and Monitors. The 'Collections' icon is highlighted with a red border. The main area shows a list of collections. One collection is expanded, revealing its requests:

- > Exercice - Ajout de tests
- > Exercice - Gestion des Variable d'environnementVariables Glob...
  - GET Recherche par ville de départ, d'arrivée et par date
  - POST Prendre une reservation
  - GET Rechercher une réservation par numéro
- > Exercice - Requetes Simple
- > Exercice - Variables Globales

At the top right, there are 'New' and 'Import' buttons.

# Gestion des variables d'environnement

## Création d'un environnement



onepoint.

# Gestion des variables d'environnement

## Création de nouvelle variable

Cliquer sur le lien **Add a new variable**

The screenshot shows the Postman interface. On the left, the sidebar has tabs for Collections, APIs, Environments, Mock Servers, and Monitors. The 'Environments' tab is selected, and under it, 'Formation Postman' is active. In the main area, there's a table for 'Formation Postman' with columns: VARIABLE, TYPE, INITIAL VALUE, and CURRENT VALUE. Four variables are listed: 'BaseUrl' (initial value: http://localhost:5000/flights\_api/v1/resources, current value: http://localhost:5000/flights\_api/v1/resources), 'Date' (initial value: 2022-12-25, current value: 2022-12-25), 'Departure' (initial value: Paris, current value: Paris), and 'Arrival' (initial value: London, current value: London). A tooltip 'Add a new variable' is shown above the table. At the top right, there are 'Save', 'Share', and other options.

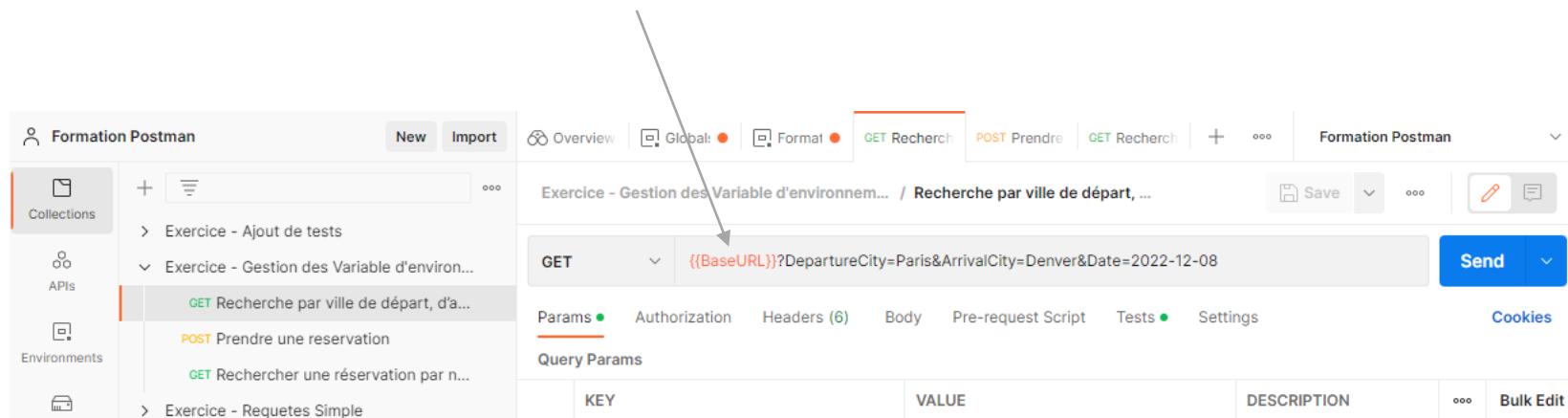
Vérifier que l'environnement est actif

The screenshot shows the Postman interface with the 'Formation Postman' environment active. A tooltip says 'INITIAL VA This environment is already active.' A context menu is open over the table, with the 'Set as active environment' option highlighted. Other options in the menu include Move, Export, Duplicate (Ctrl+D), Create a fork (Ctrl+Alt+F), and Create pull request.

# Gestion des variables d'environnement

## Utilisation de la variable d'environnement

Remplacer `http://localhost:5000/flights_api/v1/resources` par `{{BaseUrl}}` dans toutes les requêtes



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'APIs', 'Environments', and 'Folders'. Under 'Collections', there are several items: 'Exercice - Ajout de tests', 'Exercice - Gestion des Variable d'environnem...', and 'Exercice - Requetes Simple'. The middle section shows a 'GET' request titled 'Recherche par ville de départ, d...'. The URL field contains the placeholder `http://{{BaseUrl}}/flights_api/v1/resources?DepartureCity=Paris&ArrivalCity=Denver&Date=2022-12-08`. An arrow points from the text 'Remplacer {{BaseUrl}} par {{BaseUrl}} dans toutes les requêtes' to the `http://{{BaseUrl}}` part of the URL. The right side of the interface includes tabs for 'Overview', 'Global', 'Format', 'GET Recherch...', 'POST Prendre...', and 'GET Recherch...', along with buttons for 'Save', 'Send', and 'Cookies'.

En modifiant une seule donnée  `{{BaseUrl}}`,  
les requêtes pointeront vers un nouvel environnement

# Gestion des variables d'environnement

## Utilisation d'une variable d'environnement

The screenshot shows the Postman interface with a sidebar containing a tree view of test cases and requests. The main window displays a GET request for flights. In the 'Params' tab, under 'Query Params', there are three entries: 'DepartureCity' (checked), 'ArrivalCity' (checked), and 'Date' (checked). A dropdown menu is open over the 'Value' field of 'DepartureCity', showing a list of global variables: Numéro\_du\_Vol, Num\_de\_reservation, BaseURL, Date, Departure, and Arrival. The 'ArrivalCity' and 'Date' fields also have dropdown menus open, showing similar lists of variables.

Saisir {{ ,  
une liste des variables  
disponibles est proposée

The screenshot shows the Postman interface with a sidebar containing a tree view of test cases and requests. The main window displays a POST request for flight orders. In the 'Body' tab, the raw JSON payload is shown: "Class": "Economy", "CustomerName": "bruno", "DepartureDate": "{{date}}", "FlightNumber": "{{Numero\_du\_Vol}}", and "NumberOfTickets": "2". Arrows point from the double curly braces in the 'DepartureDate' and 'FlightNumber' fields to the dropdown menu in the 'DepartureCity' field of the previous screenshot, indicating that the same list of variables is available here as well.

Saisir les variables  
entre double {{ }}

onepoint.

# Gestion des variables d'environnement

## Consultation des variables d'Environnement / Globales

Récupérer ici la liste des variables

The screenshot shows the Postman application interface. At the top, there's a search bar and various navigation icons. Below the header, a tab bar includes 'Form', 'GET Recl', 'POST Prend', 'GET Recher', 'Exercice', and 'Formation Postman'. A dropdown menu is open next to 'Formation Postman'. On the left, a sidebar titled 'Globals' lists variables: 'Numero\_du\_Vol' and 'Num\_de\_reservation' (both checked), and an option to 'Add a new variable'. The main content area displays two tables. The first table, titled 'Formation Postman', lists variables: 'BaseURL' (initial value: 'http://localhost:5000/flights\_api/v1/resource'), 'Date' (initial value: '2022-12-25'), 'VilleDepart' (initial value: 'Paris'), and 'VilleArrivee' (initial value: 'London'). The second table, titled 'Globals', lists variables: 'Numero\_du\_Vol' (current value: '17105') and 'Num\_de\_reservation' (current value: '92'). An 'Edit' button is located at the top right of each table.

VARIABLE	INITIAL VALUE	CURRENT VALUE
BaseUrl	http://localhost:5000/flights_api/v1/resource	http://localhost:5000/flights_api/v1/resources
Date	2022-12-25	2022-12-25
VilleDepart	Paris	Paris
VilleArrivee	London	London

VARIABLE	INITIAL VALUE	CURRENT VALUE
Numero_du_Vol		17105
Num_de_reservation		92

# Exercice 4 : Variables d'environnement

8. Exercice 4 : Variables d'environnement
  - 8.1. Création des variables d'environnement
  - 8.2. Rendre les requêtes indépendantes de l'adresse ses services Rest
  - 8.3. Utiliser les variables ville de départ, ville d'arrivée et date dans GetFlights
  - 8.4. Modifier BookFlight en utilisant la variable Date
  - 8.5. Exécuter la séquence et visualiser les variables

# Synthèse des variables

Instruction	Description
pm.globals.set('myVariable', MY_VALUE);	Sauvegarder dans variable globale
pm.globals.get('myVariable');	Lire une variable globale
pm.globals.unset('myVariable');	Réinitialiser un variable globale
pm.globals.clear();	Effacer toutes les variables globales
pm.collectionVariables.set('myVariable', MY_VALUE); pm.collectionVariables.get('myVariable'); pm.collectionVariables.unset('myVariable'); pm.environment.clear();	Idem pour les variables de collection
pm.environment.set('myVariable', MY_VALUE); pm.environment.get('myVariable'); pm.environment.unset("myVariable"); pm.environment.clear();	Idem pour les variables d'environnement
pm.variables.get('myVariable');	Idem pour les variables de requête
pm.environment.name	Lire le nom de l'environnement actif
pm.iterationData.get('myVariable');	Lire une donnée sur itération avec un fichier CSV/JSON
pm.variables.replaceIn('{{\$randomFirstName}}');	Utiliser une variable dynamique
console.log(myVar);	Ecrire dans la log d'exécution

# Synthèse des variables dynamiques

Postman dispose de fonctions de génération dynamique des données pour les tests

La liste complète des possibilités :

<https://learning.postman.com/docs/writing-scripts/script-references/variables-list/>

Variable dynamique	Description	Exemple
\$randomCity	City	East Ryanfurt, Jenkinsview
\$randomStreetName	Street name	Mckenna Pines, Schiller Highway, Vandervort Pike
\$randomStreetAddress	Street with number	98165 Tanya Passage, 0695 Monahan Squares
\$randomCountry	Country	Belgium, Antarctica (the territory South of 60 deg S)
\$randomCountryCode	Country code (2-letter)	GY, TK, BG
\$randomPrice	Price	244.00, 301.00
\$randomDatePast	Date in the past	Wed Mar 06 2019 04:17:52 GMT+0800 (WITA)
\$randomDateFuture	Date in the future	Wed Nov 20 2019 20:26:40 GMT+0800 (WITA)
\$randomDateRecent	Recent date	Thu Jun 20 2019 13:29:11 GMT+0800 (WITA)
\$randomCurrencyCode	Currency code	THB, HTG USD, AUD
\$randomCurrencyName	Currency name	Pound Sterling, Bulgarian Lev
\$randomPhrase	Phrase	We need to copy the online CSS microchip!
\$randomEmail	Email from popular email providers	Mable_Crist@hotmail.com, Ervin47@gmail.com
\$randomPassword	Password	v_Ptr4aTaBONsM0, 8xQM6pKgBUndK_J
\$randomLoremWord	Lorem ipsum word	ipsa, dolor, dicta
\$randomFirstName	First name	Dillan, Sedrick, Daniela
\$randomLastName	Last name	Schamberger, McCullough, Becker
\$randomPhoneNumber	Phone number	946.539.2542 x582, (681) 083-2162



7

---

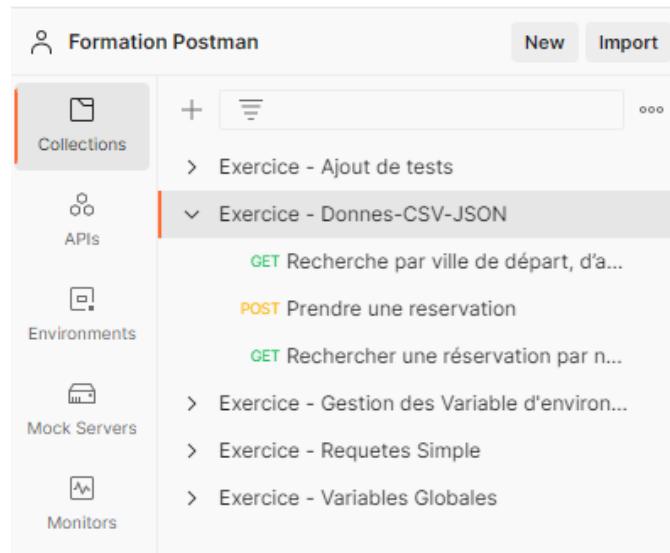
# Utilisation des données CSV/JSON

<https://learning.postman.com/docs/running-collections/working-with-data-files/>

# Utiliser les données dans un fichier CSV / JSON

**Objectif :** Automatiser une exécution d'une requête à partir de la lecture d'un fichier de données

Création d'une nouvelle collection en dupliquant la collection de l'exercice des variables d'environnement



# Utiliser les données dans un fichier CSV / JSON

## Préparation de la requête

Pour récupérer les données du fichier csv, la requête sera renseignée avec des variables portant le nom de la colonne à reprendre

	Departure,Arrival,Date,FlightNumber,Price,SeatsAvailable
1	Seattle,Portland,2023-05-11,1225,138.47,250
2	London,San Francisco,2023-05-20,17022,171.8,250
3	Paris,Denver,2023-05-13,17040,163.8,250
4	Seattle,San Francisco,2023-05-18,1226,139.47,250
5	London,Zurich,2023-05-13,11038,159.8,250
6	Frankfurt,Paris,2023-05-07,11225,145.2,250
7	Paris,Los Angeles,2023-05-19,17122,156.47,250
8	Sydney,London,2023-05-12,11765,179.6,250
9	Seattle,Denver,2023-05-17,1212,125.47,250
10	San Francisco,Paris,2023-05-09,20221,112.2,250
11	
12	

The screenshot shows the Postman interface with a collection named "Exercice - Donnes-CSV-JS...". A GET request is selected with the URL:  `{{BaseUrl}}/Flights?DepartureCity={{Departure}}&ArrivalCity={{Arrival}}&Date={{Date}}`. The "Params" tab is active, showing three parameters: DepartureCity, ArrivalCity, and Date, each with a value of  `{{(Departure)}}` ,  `{{(Arrival)}}` , and  `{{(Date)}}`  respectively. Arrows point from the CSV file above to these parameter values in the Postman interface.

# Utiliser les données dans un fichier CSV / JSON

## Exécution de la requête

Sélectionner la collection *Exercice – Données-CSV-JSON*

Sur l'onglet *Run*, cliquer sur le bouton *Run Collection*

Exercice - Données-CSV-JSON

Runs

Past runs Scheduled runs

Runs triggered for this collection via Collection Runner and Postman CLI.

Last 100 runs Run by Run status Source

Start time Source Duration All tests Passed Failed Skipped

Your collection has not been run yet

Run Collection

# Utiliser les données dans un fichier CSV / JSON

## Exécution de la requête

Cocher la requête prête (la première)

Au niveau du champ **Data**, cliquer sur le bouton **Select File**

The screenshot shows the 'Choose how to run your collection' section with 'Run manually' selected. In the 'Run configuration' section, 'Iterations' is set to 10 and 'Delay' is set to 0 ms. The 'Data' section has a checked checkbox for the first GET request ('Recherche par ville de départ, d'arrivée et par date'). The 'Select File' button next to the file name 'Flights\_data.csv' is highlighted with a red arrow. The 'Data File Type' dropdown is set to 'text/csv'. A 'Preview' button is also visible. At the bottom is a red 'Run Exercice - Donnes-CSV-JSON' button.

Le fichier chargé peut être consulté, cela permet aussi de voir que le fichier est correctement prise en charge

PREVIEW DATA				
Iteration	Departure	Arrival	Date	FlightNumber
1	"Seattle"	"Portland"	"2023-05-11"	1225
2	"London"	"San Francisco"	"2023-05-20"	17022
3	"Paris"	"Denver"	"2023-05-13"	17040
4	"Seattle"	"San Francisco"	"2023-05-18"	1226
5	"London"	"Zurich"	"2023-05-13"	11038
6	"Frankfurt"	"Paris"	"2023-05-07"	11225
7	"Paris"	"Los Angeles"	"2023-05-19"	17122
8	"Sydney"	"London"	"2023-05-12"	11765
9	"Seattle"	"Denver"	"2023-05-17"	1212
10	"San Francisco"	"Paris"	"2023-05-09"	20221

Cliquer sur le bouton **Run Exercice – Donnee-CSV-JSON**

# Utiliser les données dans un fichier CSV / JSON

## Résultats de la requête

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Collections, APIs, Environments, Mock Servers, Monitors, Flows, History.
- Recent Runs:** A section titled "Deux dernières exécutions" (Last two executions) lists "Exercice - Ajout de tests" and "Exercice - Données-CSV-JSON".
- Current Run:** "Exercice - Données-CSV-JSON - Run results" for "Run on Today, 14:42:02".
  - Summary:** All Tests Passed (10), Failed (0), Skipped (0). Iterations: 10, Duration: 911ms, Avg. Resp. Time: 12 ms.
  - Iterations:**
    - Iteration 9:** GET Recherche par ville de départ... {{BaseUrl}}/Flights?DepartureCity={{Departu... / Recherc... 200 OK 13 ms 2.956 KB
      - Pass Le code retour est 200
    - Iteration 10:** GET Recherche par ville de dépa... {{BaseUrl}}/Flights?DepartureCity={{Departu... / Recherc... 200 OK 12 ms 585 B
      - Pass Le code retour est 200
- Console:** Shows API logs for five requests, all successful (200 OK) with response times between 11 ms and 13 ms.
  - ▶ GET http://localhost:5000/flights\_api/v1/resources/Flights?DepartureCity=Paris&ArrivalCity=Los%20Angeles&Date=2023-05-19 200 | 12 ms
  - ▶ GET http://localhost:5000/flights\_api/v1/resources/Flights?DepartureCity=Sydney&ArrivalCity=London&Date=2023-05-12 200 | 11 ms
  - ▶ GET http://localhost:5000/flights\_api/v1/resources/Flights?DepartureCity=Seattle&ArrivalCity=Denver&Date=2023-05-17 200 | 13 ms
  - ▼ GET http://localhost:5000/flights\_api/v1/resources/Flights?DepartureCity=San%20Francisco&ArrivalCity=Paris&Date=2023-05-09 200 | 12 ms
  - ▶ Network
  - ▶ Request Headers
  - ▶ Response Headers
  - ▼ Response Body ↗

```
[{"Airline": "AA", "ArrivalCity": "Paris", "ArrivalTime": "02:23 PM", "DepartureCity": "San Francisco", "DepartureTime": "07:12 AM", "FlightNumber": 20210, "Price": 112.2, "SeatsAvailable": 250, "DayOfWeek": "Tuesday"}, {"Airline": "AA", "ArrivalCity": "Paris", "ArrivalTime": "06:23 PM", "DepartureCity": "San Francisco", "DepartureTime": "11:12 AM", "FlightNumber": 20221, "Price": 112.2, "SeatsAvailable": 250, "DayOfWeek": "Tuesday"}]
```
- Bottom Bar:** Cookies, Capture requests, Runner, Trash, Help, Settings.

Détail de la réponse de la dernière requête

# Utiliser les données dans un fichier CSV / JSON

Exécuter une séquence de test (avec deux requêtes)

The screenshot shows the Postman interface with a sidebar containing a tree of exercises. The main area displays a test sequence for a 'Recherche par ville de départ' request. The 'Tests' tab is selected, showing the following code:

```
1 pm.test("Le code retour est 200", function () {
2     pm.response.to.have.status(200);
3 });
4
5 pm.test("La réponse contient le tag FlightNumber", function () {
6     pm.expect(pm.response.text()).to.include("FlightNumber");
7 });
8
9 pm.test("le numero de vol attendu dans la réponse " + pm.iterationData.get("FlightNumber"), function () {
10    pm.expect(pm.response.text()).to.include(pm.iterationData.get("FlightNumber"));
11});
```

An arrow points from the text 'Et pourvoir faire une comparaison avec les résultats de la réponse' to the 'pm.iterationData.get("FlightNumber")' part of the test script.

Sur la requête **Recherche par ville de départ**

Ajouter la fonction **pm.iterationData.get (« »)** pour récupérer les données du fichier csv (utiliser le même nom)  
Et pourvoir faire une comparaison avec les résultats de la réponse

Departure	Arrival	Date	FlightNumber	Price	SeatsAvailable
Seattle	Portland	2023-05-11	1225	138.47	250
London	San Francisco	2023-05-20	17022	171.8	250
Paris	Denver	2023-05-13	17040	163	250

# Utiliser les données dans un fichier CSV / JSON

La deuxième requête, récupère `{{FlightNumber}}` du fichier CSV

The screenshot shows the Postman interface with two requests:

- Request 1 (Top):** POST `{{BaseUrl}}/FlightOrders`.
  - Body tab selected, showing JSON data:

```
1 {  
2   "Class" : "Economy",  
3   "CustomerName" : "bruno",  
4   "DepartureDate" : "{{Date}}",  
5   "FlightNumber" : "{{FlightNumber}}",  
6   "NumberOfTickets" : "2".  
7 }
```
- Request 2 (Bottom):** POST `{{BaseUrl}}/FlightOrders`.
  - Tests tab selected, showing a PM Test script:

```
1 pm.test("Vérifier le prix total", function () {  
2     var jsonData = pm.response.json();  
3     prix = pm.iterationData.get("Price")  
4     pm.expect(jsonData.TotalPrice).to.eql(2*prix);  
5 });
```

Ici, vérification que  
le prix total (dans la réponse) = Nb de ticket \* Prix (fichier CSV)  
La fonction `pm.iterationData.get()` est utilisée pour récupérer le prix des billets

# Utiliser les données dans un fichier csv et JSON

Lancer l'enchainement des requêtes (voir slides 55 & 56)

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** A tree view of test collections:
  - Exercice - Ajout de tests
  - Exercice - Créer une boucle
  - Exercice - Donnees-CSV-JSON (selected)
  - GET Recherche par ville de départ,...
  - POST Prendre une reservation
  - Exercice - Gestion des Variable d'env...
  - Exercice - Requetes Simple
  - Exercice - Variables Globales
- Top Bar:** Title "Exercice - Donnees-CSV-JSON - Run results", timestamp "Run on Today, 15:35:57", "View all runs" button, "Run Again" button, "Automate Run" dropdown, "+ New Run" button, "Export Results" button.
- Summary Table:** Shows runner information: "Runner" (Formation Postman), "Iterations" (10), "Duration" (1s 727ms), "All tests" (40), "Avg. Resp. Time" (17 ms).
- Test Details:** "All Tests" section: Passed (40), Failed (0), Skipped (0). "View Summary" link.
- Request 1:** POST /Prendre une reservation. Status: 200 OK, 37 ms, 188 B. Test: Pass, Description: Vérifier le prix total.
- Iteration 10:** GET /Recherche par ville de départ, d'ar... Status: 200 OK, 6 ms, 585 B. Three sub-tests:
  - Pass, Description: Le code retour est 200
  - Pass, Description: La réponse contient le tag FlightNumber
  - Pass, Description: le numero de vol attendu dans la réponse 20221
- Request 2:** POST /Prendre une reservation. Status: 200 OK, 15 ms, 187 B. Test: Pass, Description: Vérifier le prix total.
- Bottom Navigation:** Find and Replace, Console, Cookies, Capture requests, Runner, Trash, Help.

onepoint.

# Exercice 5 : Utiliser les données dans un fichier csv et JSON

- 9. Exercices 5 : Utiliser les données dans un fichier csv et JSON
  - 9.1. Utiliser les données dans un fichier csv
  - 9.2. Exécuter la séquence de test
  - 9.3. Utiliser les données dans un fichier json



8

---

# Création d'une boucle

<https://learning.postman.com/docs/running-collections/building-workflows/>

# Création d'une boucle

**Objectif :** Répéter la même séquence de tests

Création d'une nouvelle collection avec les requêtes suivantes :

Rechercher une réservation par le nom du client

GET {{BaseUrl}}/FlightOrders?CustomerName={{customer}}

Supprimer une réservation par numéro

DELETE {{BaseUrl}}/FlightOrders/{order}

## ▼ Exercice - Créer une boucle

GET Rechercher une réservation par le nom du client

DEL Supprimer une réservation par numéro

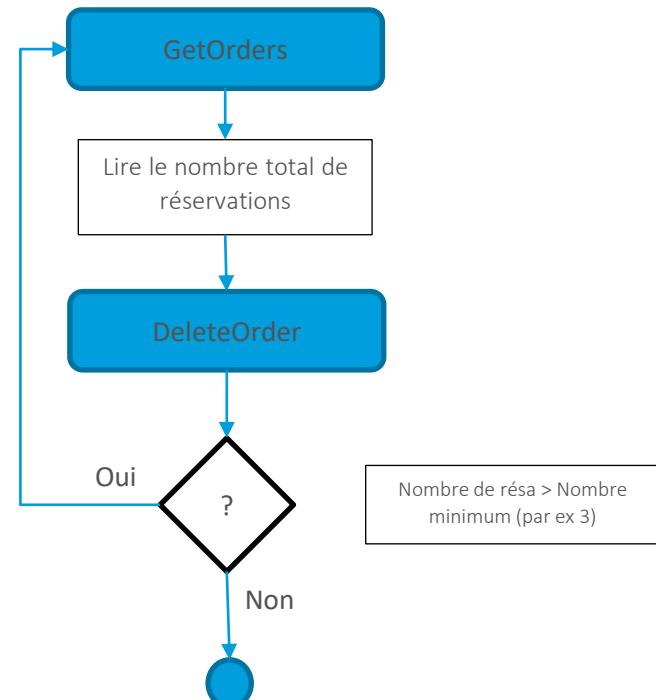
## Création d'une boucle

## Objectif :

## Récupérer la liste des réservations

## Supprimer une réservation

Répéter la boucle si le nombre de réservation est supérieur à 3



# Création d'une boucle

Préparation de la première requête : Récupérer la liste des réservations



The screenshot shows the Postman interface with a project titled "Exercice - Crée une boucle". A GET request is defined with the URL `((BaseUrl))/FlightOrders?CustomerName=bruno`. The "Tests" tab is selected, displaying the following JavaScript code:

```
1 //Vérification du nom du client de la première réservation
2 pm.test("Vérifier le nom du client ", function () {
3     var jsonData = pm.response.json();
4     console.log(jsonData.length)
5     pm.expect(jsonData.length).to.be.gt(1)
6
7     pm.globals.set("Nb_Reservation", jsonData.length);
8 });
9
10 //Récupérerer le nombre de réservation
11 pm.test("Vérifier qu'il existe au moins une réservation", function () {
12     var jsonData = pm.response.json()
13     pm.expect(jsonData[0].CustomerName).to.be.eql("bruno");
14 });
15
16 //Récuperer les numéro de la première réservation
17 pm.test("Vérifier que le numéro de réservation est supérieur à 80", function () {
18     var jsonData = pm.response.json();
19     pm.expect(jsonData[0].OrderNumber).to.be.greaterThan(80)
20
21     //Sauvegarde de la 1er réservation dans une variable globale Num_Reservation
22     pm.globals.set("Num_Reservation", jsonData[0].OrderNumber);
23 });
24
```

Informations récupérées après exécution:

- Identifiant de la première réservation
- Le nombre de réservation total

Où au niveau des logs

Globals		
VARIABLE	INITIAL VALUE	CURRENT VALUE
Num_Reservation		82
Nb_Reservation		63



The Postman console shows the execution of a GET request to `http://localhost:5000/flights_api/v1/resources/FlightOrders?CustomerName=bruno`. The response body is partially visible.

onepoint.

# Création d'une boucle

## Préparation de la deuxième requête : Supprimer une réservation

Récupération de la variable globale du numéro de réservation à supprimer

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main workspace shows a collection named "Formation Postman" with a sub-collection "Exercice - Créer une boucle". Inside this, there are several items: "Exercice - Ajout de tests", "Exercice - Créer une boucle" (which is expanded), "GET Rechercher une réservation p...", "DEL Supprimer une reservation par...", "Exercice - Données-CSV-JSON", "Exercice - Gestion des Variable d'env...", "Exercice - Requetes Simple", and "Exercice - Variables Globales". The "DEL Supprimer une reservation par..." item is highlighted with a red arrow pointing to it from the text above. The main panel displays a DELETE request with the URL `((BaseURL))/FlightOrders/((Num_Reservation))`. The "Params" tab is selected, showing a table with one row: "Key" (Value: "Key") and "Value" (Value: "Value"). Below the table, the "Body" tab is selected, showing a JSON response:

```
1 true
2   "true": "Order deleted 82"
3
```

The status bar at the bottom shows the URL `DELETE http://localhost:5000/flights_api/v1/resources/FlightOrders/82`, a status of `200 | 39 ms`, and a "Console" tab.

onepoint.

# Création d'une boucle

Préparation de la deuxième requête : Réaliser la boucle

Ajouter sur l'onglet **Pre-request Script** de la requête **Supprimer une réservation** :

1- Affichage du nombre total de réservation dans la console. Utiliser le snippet **Get global variable**

2- La boucle est réalisée avec la commande **postman.setNextRequest("")**;

Avec le nom de la requête qui doit suivre : **Rechercher une réservation par le nom du client**

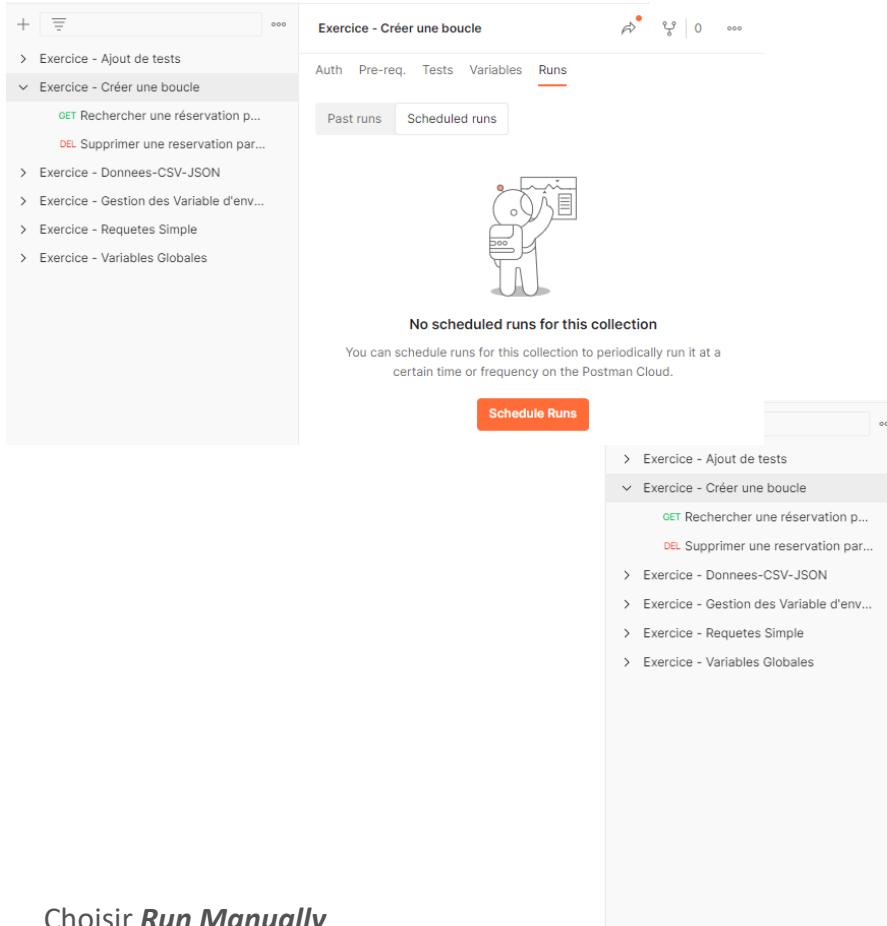
3- Ajouter une condition pour permettre cet enchainement (if condition { } ).

The screenshot shows the Postman interface. On the left, there's a sidebar with a tree view of collections and requests. Under 'Exercice - Créer une boucle', two requests are listed: 'GET Rechercher une réservation p...' and 'DEL Supprimer une reservation par...'. The 'DEL' request is currently selected. The main workspace shows a 'DELETE' request with the URL {{BaseUrl}}/FlightOrders/{{Num\_Reservation}}. The 'Pre-request Script' tab is selected, showing the following JavaScript code:

```
1 var Compteur_Réservation = pm.globals.get("Nb_Réservation");
2 console.log ("le nombre de réservation est : " + Compteur_Réservation);
3
4 if (Compteur_Réservation > 2) {
5 postman.setNextRequest ("Rechercher une réservation par le nom du client");
6 }
7
```

# Création d'une boucle

## Lancement de la boucle



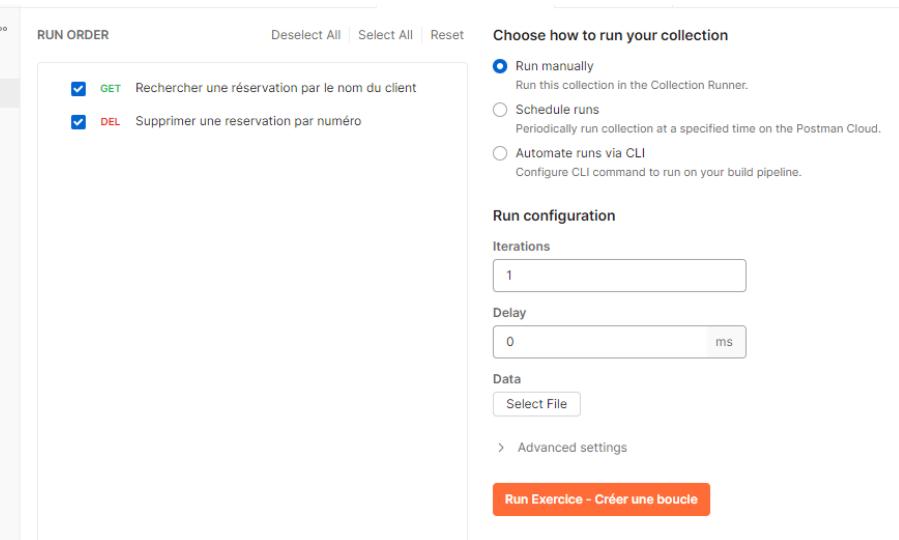
The screenshot shows the Postman interface with the following details:

- Left sidebar:** Shows a tree view of collections:
  - Exercice - Ajout de tests
  - Exercice - Créer une boucle (selected)
  - Exercice - Données-CSV-JSON
  - Exercice - Gestion des Variable d'environnement
  - Exercice - Requêtes Simple
  - Exercice - Variables Globales
- Top navigation bar:** Includes tabs for Auth, Pre-req., Tests, Variables, and Runs (highlighted in red).
- Runs section:** Sub-tabs include Past runs and Scheduled runs. A large icon of a robot holding a monitor is displayed.
- Middle section:** Text states "No scheduled runs for this collection". Below it, a note says: "You can schedule runs for this collection to periodically run it at a certain time or frequency on the Postman Cloud." A prominent orange "Schedule Runs" button is centered.
- Bottom section:** Shows the list of exercises again, with "Exercice - Créer une boucle" expanded to show its sub-requests: GET Rechercher une réservation par nom and DEL Supprimer une réservation par numéro.

Choisir **Run Manually**

Cliquez sur le bouton **Run Exercice – Créer une boucle**

Sur l'onglet **Runs**,  
Cliquer sur le bouton **Schedule Runs**



The screenshot shows the "Schedule Runs" configuration dialog with the following settings:

- Choose how to run your collection:** Radio button selected for "Run manually". Other options include "Schedule runs" and "Automate runs via CLI".
- RUN ORDER:** Lists the selected requests: GET Rechercher une réservation par nom du client and DEL Supprimer une réservation par numéro.
- Iterations:** Set to 1.
- Delay:** Set to 0 ms.
- Data:** Option to "Select File".
- Advanced settings:** Link to further configuration.
- Run button:** Orange "Run Exercice - Créer une boucle" button.

# Création d'une boucle

## Résultats

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows collections, APIs, environments, mock servers, monitors, flows, and history.
- Top Bar:** Includes "Formation Postman", "New", "Import", "Overview", "Supprimer une réservation", "Exercice - Créer une boucle", "Run Again", "Automate Run", "New Run", and "Export Results".
- Run Results:** Title "Exercice - Crée une boucle - Run results" with a timestamp "Run on Today, 12:01:56".
  - Summary table:

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	Formation Postman	1	9s 355ms	204	17 ms
  - Test list:
    - DELETE** Supprimer une réservation par numéro `((BaseURL))/FlightOrders/{{Num_Reservation}}` / Supprimer une r... 200 OK 12 ms 175 B
    - GET** Rechercher une réservation par le nom du client `((BaseURL))/FlightOrders?CustomerName=bruno` / Recher... 200 OK 6 ms 487 B
    - DELETE** Supprimer une réservation par numéro `((BaseURL))/FlightOrders/{{Num_Reservation}}` / Supprimer une r... 200 OK 11 ms 175 B
  - Console output:

```
"le nombre de reservation est : 4"
> DELETE http://localhost:5000/flights_api/v1/resources/FlightOrders/141
> GET http://localhost:5000/flights_api/v1/resources/FlightOrders?CustomerName=bruno
3
"le nombre de reservation est : 3"
> DELETE http://localhost:5000/flights_api/v1/resources/FlightOrders/142
> GET http://localhost:5000/flights_api/v1/resources/FlightOrders?CustomerName=bruno
2
"le nombre de reservation est : 2"
> DELETE http://localhost:5000/flights_api/v1/resources/FlightOrders/143
```

Enchaînent des tests

Décompte au niveau de la console

onepoint.

# Exercice 6 : Création d'une boucle

10. Exercice 6 : Créer une boucle
  - 10.1. Lire le nombre total de réservation
  - 10.2. Réaliser la boucle



9

---

# API Mock / API Monitor

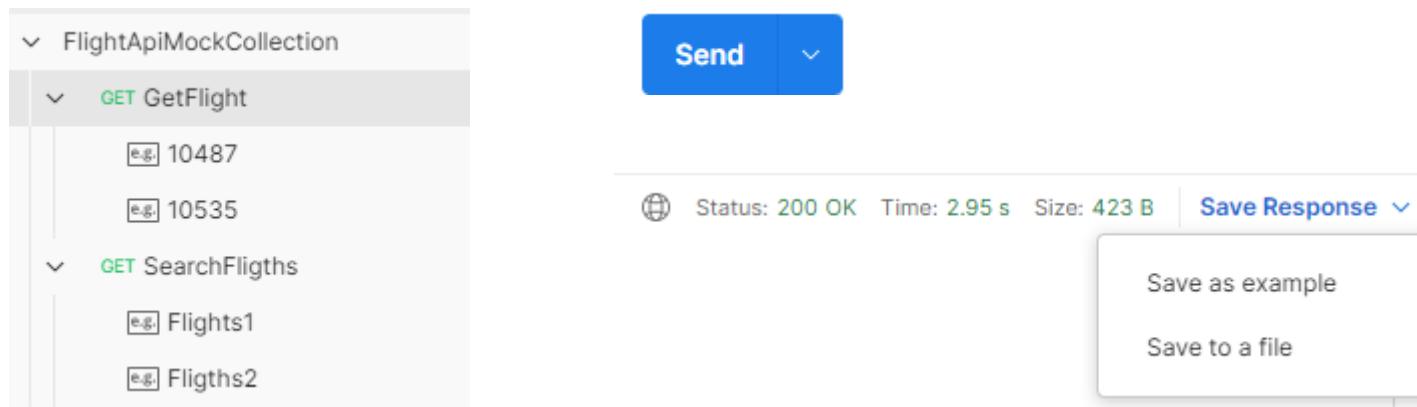
<https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/setting-up-mock/>

# Utilité du Mock API

- Permet de simuler le fonctionnement réel d'un API (bouchon).
- Modifier manuellement la réponse pour anticiper une future évolution de l'API.
- Rendre les tests indépendants d'une API qui n'est pas disponible.
- Economiser le prix des APIs payantes en phase de test

# Processus de création d'un serveur Mock

- Créer une collection comportant les requêtes à simuler.
- Créer un serveur Mock à partir de la collection
- Créer autant d'exemples que nécessaire pour simuler la requête
- Utiliser les requêtes simulées dans vos tests



# Création d'un serveur Mock

The screenshot shows the Postman interface with the 'Mock Servers' tab selected in the sidebar. A collection named 'FlightApiMockServer' is listed under the 'Collections' section, marked with a red circle containing the number 1.

### Create a mock server

1. Select collection to mock    2. Configuration

Create a new collection     Select an existing collection 2

Select a collection you want to mock.

Search collection

FlightApi\_TNR     FlightApi

FlightApiMockCollection

### Create a mock server

Select collection to mock    2. Configuration

Mock Server Name  
 FlightApiMockServer2

Collection  
 FlightApiMockCollection

Environment  
An environment is a group of variables useful for storing and reusing values.  
[Learn more ↗](#)

No Environment

Save the mock server URL as an new environment variable  
This will create a new environment containing URL.

Simulate a fixed network delay  
 No delay selected

Back    Create Mock Server 3

FlightApiMockServer2     Copy URL     View Collection Docs     Edit Configuration

Search your calls...     Refresh Logs



**No mock server calls yet**

To call the mock server, follow these steps:

4

- Send a request to the following mock server URL, followed by the request path:  
`https://ff534a22-3cbd-4b01-8e67-84d9c0594639.mock.pstmn.io`
- Add examples responses to each request that the mock server will return.  
[Learn what examples are and how to use them ↗](#)

**onepoint.**

# Utiliser les requêtes simulées

- La variable url est créée automatiquement dans un environnement FlightApiMockServer
- Exemples d'appels

`{url}/Flights/10535`

`{url}/Flights?DepartureCity=Paris&ArrivalCity=London&Date=2022-12-30`

The screenshot shows the Postman interface with the following details:

- Request URL:** `TestMockServer / Flights_10535`
- Method:** GET
- Query Params:** `{url}/Flights/10535`
- Params:** Key, Value, Description
- Response Status:** 200 OK
- Body (Pretty):**

```
1  {
2      "Airline": "AF",
3      "ArrivalCity": "London",
4      "ArrivalTime": "10:30 AM",
5      "DepartureCity": "Paris",
```
- Console:** Shows the command: `▶ GET https://816fbe85-bcc1-4ee8-837d-502ed1d2c961.mock.pstmn.io/Flights/10535`

# Processus de création d'un monitor

- L'application à surveiller doit être accessible sur internet
- Créer une collection comportant les requêtes à surveiller.
- Créer un Monitor à partir de la collection
- Programmer la fréquence de surveillance
- Lancer la surveillance

## Les requêtes utilisées pour le monitor

[https://api.taiga.io/api/v1/projects/by\\_slug?slug=himhah-demo\\_api](https://api.taiga.io/api/v1/projects/by_slug?slug=himhah-demo_api)

```
pm.test("Project name is Demo_Api ", function () {  
    var jsonData = pm.response.json();  
    pm.expect(jsonData.name).to.eql("Demo_Api");  
    pm.globals.set("ProjectId", jsonData.id);  
});
```

<https://api.taiga.io/api/v1/issues?project={{ProjectId}}>

```
pm.test("Issues count is greater than 0", function () {  
    var jsonData = pm.response.json();  
    pm.expect(jsonData.length).to.greaterThan (0);  
    pm.globals.set("IssueId", jsonData[0].id);  
});
```

<https://api.taiga.io/api/v1/issues/{{IssueId}}>

# Processus de création d'un monitor

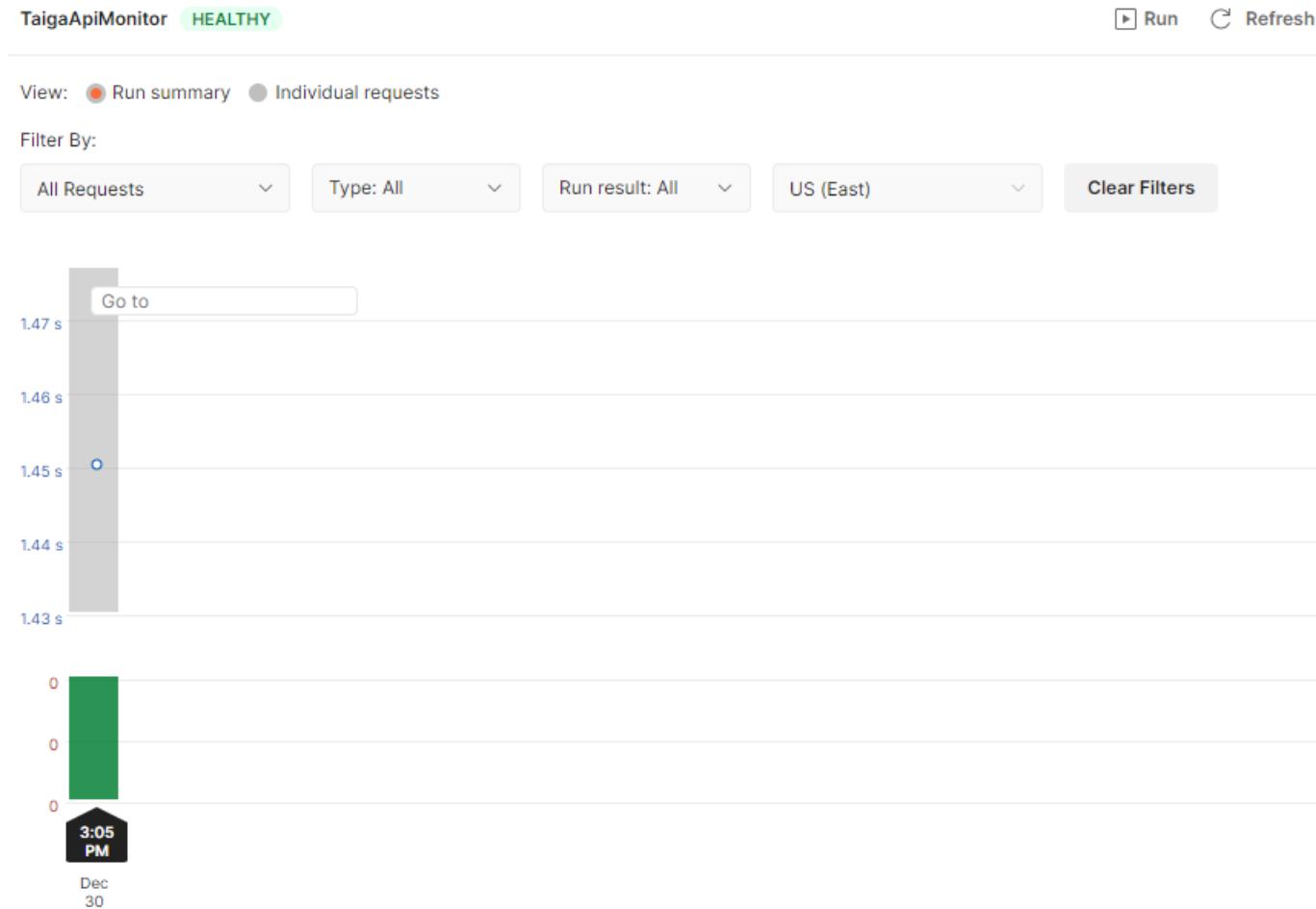
The screenshot illustrates the steps to create a monitor in the Taiga API interface:

- 1** In the left sidebar, click on the **Monitors** icon.
- 2** Click on the **Create a Monitor** button.
- 3** Enter the monitor name **TaigaApiMonitor** and select the collection **Taiga\_Monitor**.
- 4** Set the run schedule to **Every day** at **3:00 PM**, and click the **Create Monitor** button.
- 5** Once created, the monitor will appear in the list with a status message: **This monitor hasn't run yet**. A **Run** button is available to trigger a manual run.

**onepoint.**

# Résultat de la surveillance

Le résultat est aussi consultable sur internet avec votre compte postman <https://identity.getpostman.com/login>



**onepoint.**

# 10

---

## Documentation des APIs

<https://swagger.io/docs/specification/about/>

# Standard OpenAPI

Standard indépendant du langage de programmation.

Une panoplie très riche d'utilitaires.

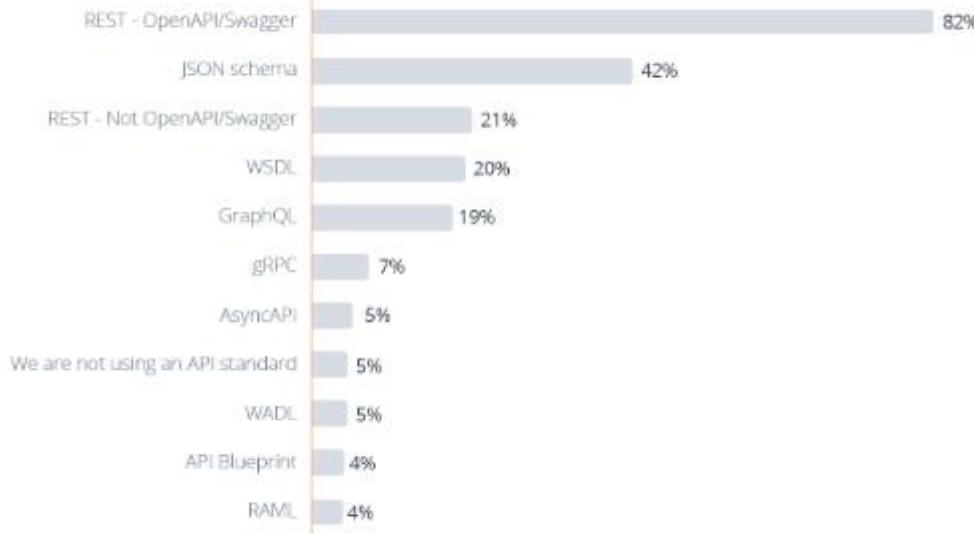
Format lisible et facilement interprétable.

Avantages :

- Validation de la description (Syntaxe, version de la spécification, directives de formatage).
- Validation des données.
- Génération de documentation.
- Génération de code.
- Éditeurs graphiques.
- Gestion des bouchons (Mock)
- Analyse de la sécurité

# Un standard populaire

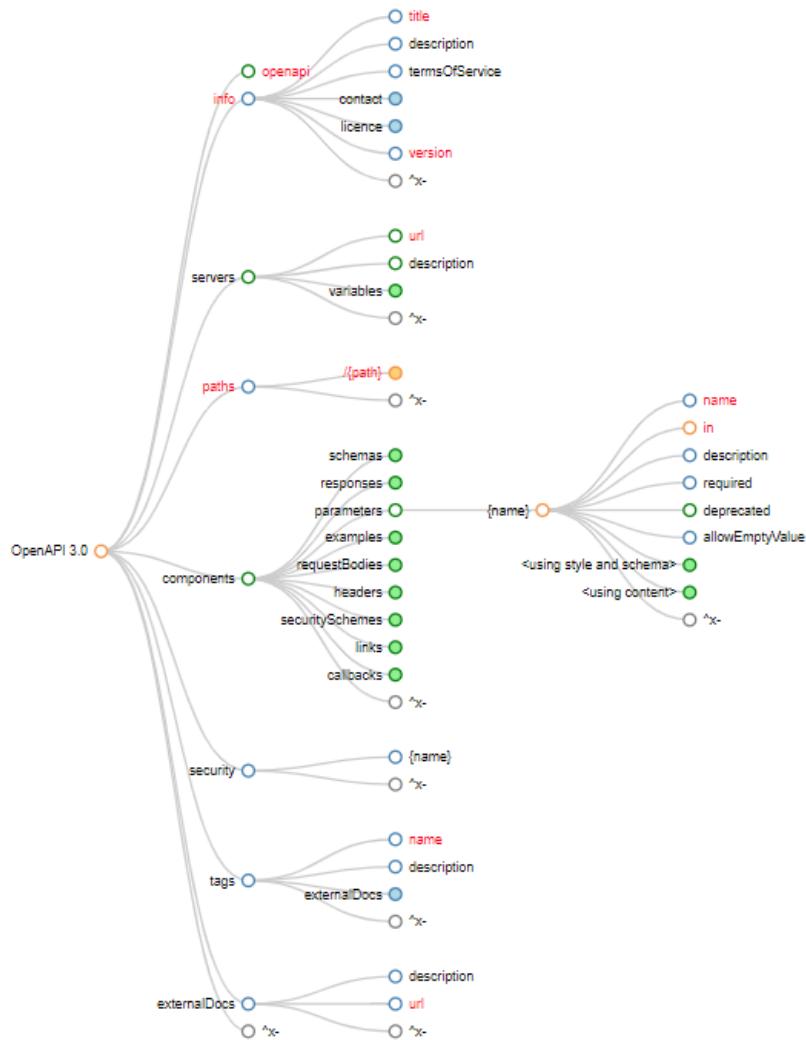
## API Design



82% of organizations use the OpenAPI Specification (formerly known as Swagger) in their API development, up from 69% last year

Source : <https://smartbear.com/resources/ebooks/the-state-of-api-2020-report/>

# Format de la documentation OpenAPI



Source : <https://openapi-map.apihandyman.io/>

# Exemples de documentation openapi

Recherche de la documentation publique sur <https://app.swaggerhub.com/search>

Exemple à explorer :

- <https://app.swaggerhub.com/apis/LizaPolishchuk/SalonsBE/1.0.0#>
- <https://app.swaggerhub.com/apis/DeOliveiraJim/ShopManager/2.0#/ShopAns>

<https://www.postman.com/postman/workspace/postman-open-technologies-convert-postman-collections-into-openapi/overview>

**onepoint.**

# Générer la documentation à partir de Postman

Exporter la collection (format JSON)

Installer postman-to-openapi (nécessite node.js)

```
npm i postman-to-openapi -g
```

Convertir la collection Postman

```
p2o ./path/to/PostmanCollection.json -f ./path/to/result.yml -o ./path/to/options.json
```

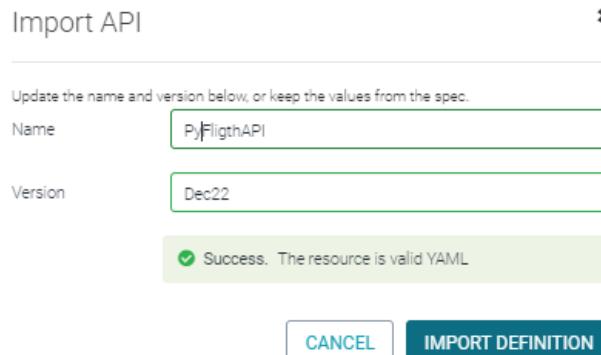
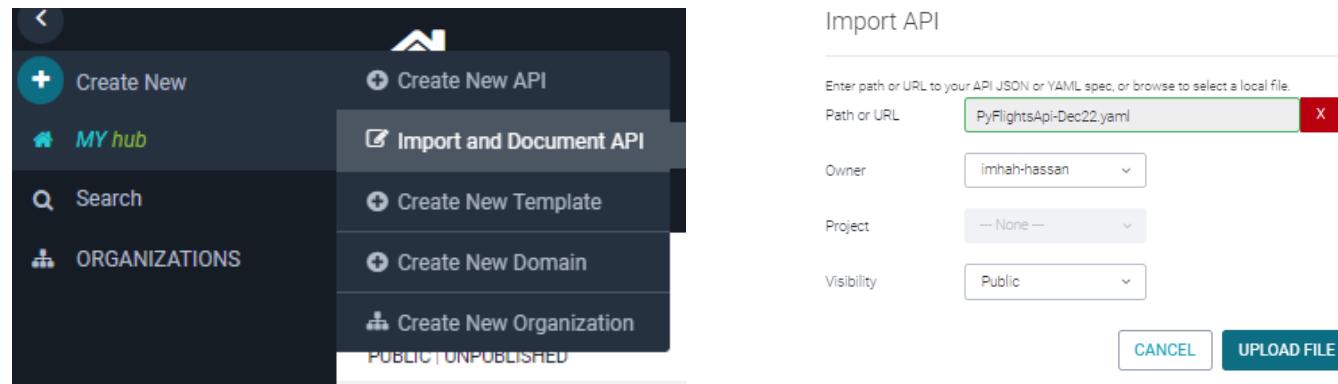
```
S:\github\PyFlightApi\tests>p2o ./FlightApi_TNR.postman.json -f ..\Doc\PyFlightsApi-Dec22.yaml
openapi: 3.0.0
info:
  title: FlightApi_TNR
  version: 1.0.0
servers:
  - url: http://localhost:5000
  - url: http://{{endpoint}}
components:
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
tags:
  - name: Authentication
  - name: Cities
  - name: Flights
  - name: FlightOrders
paths:
  /login:
    get:
```

<https://www.postman.com/postman/workspace/postman-open-technologies-convert-postman-collections-into-openapi/overview>

# Importer la documentation dans swagger.io

Créer un compte sur <https://app.swaggerhub.com/>

Importer la documentation générée



**onepoint.**

# Editer la documentation avec swagger.io

The screenshot shows the swagger.io editor interface with the following components:

- Top Bar:** Buttons for "Show Navigation", "Hide Code Editor", "Show Visual Editor", and "Hide UI Docs".
- Left Sidebar:** A tree view of API endpoints:
  - Cities**: GET /Cities, PATCH /Cities/CMN
  - Flights**: GET /login, POST /Cities, DELETE /Cities/CMN, POST /Flights, GET /RandomFlights, GET /Flights/{FlightNumber}, DELETE /Flights/{CityFlightNum}, PATCH /Flights/20001
  - FlightOrders**: GET /Flights, POST /FlightOrders, DELETE /FlightOrders, PATCH /FlightOrders/{tmpResa}, DELETE /FlightOrders/{tmpResa}
- Code Editor:** A large text area displaying the OpenAPI specification for the API. The code is as follows:

```
openapi: 3.0.0
info:
  title: FlightApi_TNR
  version: Dec22
servers:
  - url: http://localhost:5000
  - url: http://{{endpoint}}
components:
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
    tags:
      - name: Authentication
      - name: Cities
      - name: Flights
      - name: FlightOrders
  paths:
    /login:
      get:
        tags:
          - Flights
        summary: LoginAdmin
        security:
          - basicAuth: []
        parameters:
          - name: Content-type
            in: header
            schema:
              type: string
              example: application/json
        responses:
          '200':
            description: Successful response
            content:
              application/json: {}
    /cities:
      post:
        tags:
          - Flights
        summary: AddCityCasablanca
        requestBody:
```

**Right Panel:** The generated API documentation for "FlightApi\_TNR". It includes sections for Authentication, Cities, and Flights, each listing the corresponding API endpoints.

# 11

---

## Annexes

- Exercices complémentaires
- Exécution en ligne de commande
- Intégration Jenkins
- Exercices complémentaires avec taiga.io
- Présentation de l'application de démo
- Liens utiles

# Exercices complémentaires

1. Prendre un réservation sur une ville de départ et une ville d'arrivée aléatoirement choisit dans une liste de villes (utiliser GET  
[http://localhost:5000/flights\\_api/v1/resources/Cities](http://localhost:5000/flights_api/v1/resources/Cities))
2. Prendre une réservation sur un vol choisi au hazard (utiliser GET  
[http://localhost:5000/flights\\_api/v1/resources/Flights?Date=2023-02-08](http://localhost:5000/flights_api/v1/resources/Flights?Date=2023-02-08))
3. Prendre une réservation avec une date aléatoire dans le future.
4. Lire toutes les réservations prises et vérifier le prix de chaque réservation

# Exercices complémentaires avec taiga.io

- Documentation de l'API taiga.io

<https://docs.taiga.io/api.html>

- Ressources utilisées

<https://docs.taiga.io/api.html#auth-normal-login>

<https://docs.taiga.io/api.html#projects-list>

<https://docs.taiga.io/api.html#issues-list>

- Projet de démo (créer un nouveau projet)

[https://tree.taiga.io/project/himhah-demo\\_api/issues](https://tree.taiga.io/project/himhah-demo_api/issues)

- Exercices

1. Lire les informations du projet à partir de son nom (slug = himhah-demo\_api)
2. Lire la liste des tickets
3. Afficher les détails d'un ticket
4. S'authentifier et utiliser le token récupérer dans la suite des requêtes (header **Authorization: Bearer {{token}}**)
5. Mettre à jour les informations d'un ticket
6. Créer un ticket (utiliser le paramétrage Issue types, severities, priorities)
7. Créer plusieurs tickets à partir d'un fichier Json
8. Supprimer tous les tickets créés.

# Exécution en ligne de commande

<https://learning.postman.com/docs/postman-cli/postman-cli-overview/>

```
powershell.exe -NoProfile -InputFormat None -ExecutionPolicy AllSigned -Command  
"[System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://dl-  
cli.pstmn.io/install/win64.ps1')) »
```

Installation dans %USERPROFILE%\AppData\Local\Microsoft\WindowsApps

Exporter la collection à exécuter et copier le fichier .json dans ce répertoire

Exécuter la collection :

```
postman collection run DemoCLI.postman_collection.json
```

```
DemoCLI
→ project
  GET https://api.taiga.io/api/v1/projects/by_slug?slug=himhah-demo_api [200 OK, 2.89kB]

→ issues
  GET https://api.taiga.io/api/v1/issues?project=404046 [200 OK, 2.89kB]

→ issueByRef
  GET https://api.taiga.io/api/v1/issues?project=404046 [200 OK, 2.89kB]

→ authentication
  POST https://api.taiga.io/api/v1/auth [200 OK, 2.67kB, 479ms]



|                    | executed | failed |
|--------------------|----------|--------|
| iterations         | 1        | 0      |
| requests           | 4        | 0      |
| test-scripts       | 0        | 0      |
| prerequest-scripts | 0        | 0      |
| assertions         | 0        | 0      |


total run duration: 2s
total data received: 19.3kB (approx)
average response time: 439ms [min: 101ms, max: 1012ms, s.d.: 360ms]

Postman api key is required for publishing run details to postman cloud.
Please specify it by using the postman login command
```

**onepoint.**

# Intégration Jenkins

Installer newman (node.js nécessaire)

```
npm install -g newman
```

Exécuter la collection avec newman (exporter la collection depuis Postman)

```
newman run DemoCLI.postman_collection.json
```

Installer Jenkins avec les plugins **Junit**, **Test Results Analyzer**, **Git et Git Client**

Configurer un « Job » Jenkins

- **Gestion de code source : Git**

- Repository URL : <https://github.com/imhah-hassan/PyFlightApi.git>
- Branch : \*/Dec22

- **Construire périodiquement :**

```
H/2 * * * *
```

- **Exécuter une ligne de commande batch Windows :**

```
SET PATH=C:\Apps\Python311;C:\Apps\Python311\Scripts;%PATH%
pip install PyJWT requests Flask python-dateutil
start python flight_rest.py
```

- **Exécuter une ligne de commande batch Windows :**

```
SET PATH=C:\Projets\ApiMonitor\postman-exporter\node_modules\.bin;%PATH%
newman run DemoCLI.postman_collection.json --disable-unicode --reporters cli,junit
```

- **Publier le rapport des résultats des tests Junit**

```
newman/*.xml
```

# Résultats Jenkins

Options Download Test (CSV) Search: Test/Class/Package

Chart	Package/Class/Testmethod	Passed	Transitions	59	58	57	56	55	54	53	52	51	50
<input type="checkbox"/>	➊ (root)	100% (100%)	0	PASSED									
<input type="checkbox"/>	➋ DemoCli	100% (100%)	0	PASSED									
<input type="checkbox"/>	Body matches auth_token string	100% (100%)	0	PASSED									
<input type="checkbox"/>	Check project name	100% (100%)	0	PASSED									
<input type="checkbox"/>	Status code is 200	100% (100%)	0	PASSED									
<input type="checkbox"/>	Verify first issue subject	100% (100%)	0	PASSED									
<input type="checkbox"/>	Verify issue status name	100% (100%)	0	PASSED									
<input type="checkbox"/>	Verify issues count greather than 0	100% (100%)	0	PASSED									
<input type="checkbox"/>	Verify user name	100% (100%)	0	PASSED									

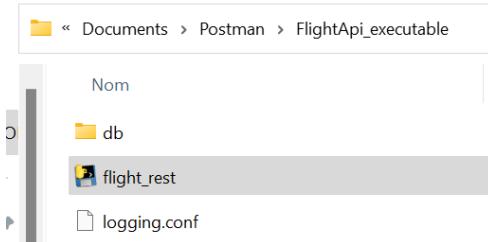
## Top 10 Most Broken Tests

# Installation de FlightApi

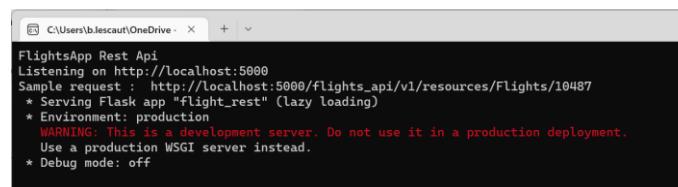
- Récupérer l'archive

<https://onepoint365.sharepoint.com/:u/r/sites/TestIT-Communaute/Documents%20partages/Tutorials/Postman/FlightApi/FlightApi.zip>

- Décompresser la dans un répertoire de travail



- Lancer l'exécutable *flight\_rest*



- Vérifier le bon fonctionnement de l'application

- Copie la requête de l'écran précédent  
([http://localhost:5000/flights\\_api/v1/resources/Flights/10487](http://localhost:5000/flights_api/v1/resources/Flights/10487)), puis
- Coller la dans la barre d'adresse d'un navigateur



# Liste des requêtes FlightApi utilisées

URL : [http://localhost:5000/flights\\_api/v1/resources](http://localhost:5000/flights_api/v1/resources)

Documentation : <https://app.swaggerhub.com/apis/imhah-hassan/PyFlightsApi/Dec22>

Nom de la requête	Ressource	Méthode	Paramètre
Rechercher un vol par son numéro	/Flights/{FlightNumber}	GET	Aucun
Recherche un vol par ville de départ, d'arrivée et par date	/Flights	GET	DepartureCity=Paris&ArrivalCity=Denver&Date=2021-12-08
Rechercher une réservation par le nom du client	/FlightOrders	GET	CustomerName={{customer}}
Rechercher une réservation par numéro	/FlightOrders/{order}	GET	Aucun
Supprimer une réservation par numéro	/FlightOrders/{order}	DELETE	Aucun
Supprimer toutes les réservations	/FlightOrders	DELETE	Aucun

Nom de la requête	Ressource	Méthode	Headers	Body
Prendre une réservation	/FlightOrders	POST	Content-Type : application/json	{ "Class" : "Economy", "CustomerName" : "IMHAH", "DepartureDate" : "2022-12-08", "FlightNumber" : "1060", "NumberOfTickets" : "2" }
Modifier une réservation par numéro	/FlightOrders/{order}	PATCH	Content-Type : application/json	{ "Class" : "First", "CustomerName" : "IMHAH", "DepartureDate" : "2021-12-09", "FlightNumber" : "1060", "NumberOfTickets" : "5" }

# Liste des données de FlightApi utilisées (extrait)

FlightNumber	DepartureInitials	Departure	DayOfWeek	ArrivalInitials	Arrival	DepartureTime	ArrivalTime	Airline	SeatsAvailable
10535	PAR	Paris	Friday	LON	London	8:00 AM	10:30 AM	AF	250
10538	PAR	Paris	Friday	LON	London	10:24 AM	11:24 AM	AF	250
10539	PAR	Paris	Friday	LON	London	10:24 AM	12:54 PM	AF	250
10543	PAR	Paris	Friday	LON	London	12:48 PM	3:18 PM	AF	250
10547	PAR	Paris	Friday	LON	London	3:12 PM	5:42 PM	AF	250
11120	PAR	Paris	Friday	LON	London	3:50 PM	5:31 PM	NW	250
11774	PAR	Paris	Friday	LON	London	9:50 AM	11:31 AM	NW	250
13119	PAR	Paris	Friday	LON	London	6:14 PM	7:55 PM	NW	250

# Liens utiles

- Documentation complète

<https://learning.postman.com/docs/getting-started/introduction/>

- Documentation rapide

<https://postman-quick-reference-guide.readthedocs.io/en/latest/index.html>

- La référence de javascript en Postman :

<https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/>

- Téléchargement : <https://www.postman.com/downloads/>

- <https://blog.nicolashachet.com/developpement-php/larchitecture-rest-expliquee-en-5-regles/>  
<https://www.tutorialspoint.com/soapui>

- <https://www.eewee.fr/postman-cest-quoi>

- Pour comprendre la syntaxe Postman,  
[https://developer.mozilla.org/fr/docs/Learn/Getting\\_started\\_with\\_the\\_web/JavaScript\\_basics](https://developer.mozilla.org/fr/docs/Learn/Getting_started_with_the_web/JavaScript_basics)