**Plan de test UC Consulter les Données : Projet Smart Territories**

**1- Identification du test**

Tests\_UCConsulterLesDonnees\_SmartTerritories

**2- Référence du module testé**

Ces tests seront effectués sur les méthodes de la classe CAPI, plus précisément la méthode getMesuresFiltre et validationParams. Ces méthodes permettent d’obtenir les données de la base en fonction des éléments présents dans la requête. Ces mêmes éléments seront vérifiés avant de recevoir les données.

**3- Objectif du test**

Ce test vérifiera les différentes possibilités de requête de l’API avec différentes données de test pour vérifier du bon renvoi des données ou d’un code d’erreur.

**4- Procédure du test**

***Les données « correctes » reçues par le client seront prises en captures d’écran et insérées en annexe***

Test 1 : Renvoi des données correct en format JSON  
On envoie une requête http correcte au serveur et on reçoit les données attendues. Le fonctionnement est correct pour les différentes requêtes (soit plusieurs numéros de test).

Données de test :

Requête http : [http://127.0.0.1:8080/SmartTerritories/dataFilter?Capteur=Roseraie&TypeDeDonnee=Temperature&Date=2025-03-10 15](http://127.0.0.1:8080/SmartTerritories/dataFilter?Capteur=Roseraie&TypeDeDonnee=Temperature&Date=2025-03-10%2015)

Après avoir effectué ces requêtes, nous est retourné les données correspondant à ces critères, les données comportent l’id de la donnée, le capteur, type de donnée et date associés, ainsi que la valeur ainsi que le code http 200 : « status success »

Test 2 : On envoie plusieurs fois le même élément de requête (ex : TypeDeDonnee) dans une seule et même requête.   
Exemple : on veut les données du capteur de Saint Serge, mais seulement des types de données Température et Humidité, rien d’autre.

*Test 2.1 🡺*

Données de test :

Requête http :   
<http://127.0.0.1:8080/SmartTerritories/dataFilter?TypeDeDonnee=Temperature&Capteur=Saint-Serge&TypeDeDonnee=NO2>

Résultat retourné : données correspondant aux critères ainsi que le code http 200 : « status success ».

*Test 2.2 🡺*

Données de test :

Requête http :

<http://127.0.0.1:8080/SmartTerritories/dataFilter?Capteur=Roseraie&TypeDeDonnee=Temperature&Capteur=Saint-Serge>

Résultat retourné : données et code http 200 : « status success ».

*Test 2.3 🡺*

Données de test :

Requête http :

<http://127.0.0.1:8080/SmartTerritories/dataFilter?Date=2025-03-10 15&TypeDeDonnee=Temperature&Date=2025-03-18 12>

Résultat retourné : données et code http 200 : « status success ».

*Test 2.4 🡺*

Données de test :

Requête http :

<http://127.0.0.1:8080/SmartTerritories/dataFilter?DateDebut=2025-03-10 15&TypeDeDonnee=Temperature&DateDebut=2025-04-25 10>

Résultat retourné : message d’erreur : « Il ne peut pas y avoir plusieurs dates de début. », et code http 400.

*Test 2.5 🡺*

Données de test :

Requête http :   
[http://127.0.0.1:8080/SmartTerritories/dataFilter?DateFin=2025-03-10 15&TypeDeDonnee=Temperature&DateFin=2025-04-25 10](http://127.0.0.1:8080/SmartTerritories/dataFilter?DateFin=2025-03-10%2015&TypeDeDonnee=Temperature&DateFin=2025-04-25%2010)

Résultat retourné : message d’erreur : « Il ne peut pas y avoir plusieurs dates de fin. », et code http 400.

Test 3 : On envoie une requête qui ne contient aucun critère

Données de test :

Requête http :   
<http://127.0.0.1:8080/SmartTerritories/dataFilter?>

ET/OU

Requête http :   
[http://127.0.0.1:8080/SmartTerritories/dataFilter](http://127.0.0.1:8080/SmartTerritories/dataFilter%20)

En toute logique, on s’attend, dans une requête sans critère, à recevoir toutes les données, cependant il peut aussi s’agir d’une faute de frappe. De plus, recevoir toutes les données correspond à un envoi massif de données, c’est pour cela qu’une autre méthode « GET » est créée et que celle-ci ramène à un message d’alerte :

{"message":"ATTENTION : vous êtes sur le point d'afficher toutes les données et la base, utiliser la requête : "/SmartTerritories/allData" si c'est votre demande."}

Ainsi que le code http 303 : « See other »

Test 4 : Renvoi d’une erreur si les critères de la requête ne correspondent pas aux données  
On envoie une requête http au serveur contenant des critères (capteur, type de données, date) non présents dans la base de données. L’API ira donc chercher une donnée présentant un critère non enregistré et donc ne retournera rien.

Données de test :

Requête http :  
<http://127.0.0.1:8080/SmartTerritories/dataFilter?Capteur=Roseraie&TypeDeDonnee=Meteo>

Le type de donnée « Météo » n’existe pas, on n’attend donc pas de données à proprement parler.

Le message suivant est donc retourné :

{"message":"Aucune donnée trouvée avec ces critères"}

Ainsi que le code d’erreur http 444 : « No Response ». Autrement dit aucune donnée n’a été retournée.

Test 5 : Renvoi d'une erreur si le format de la date est mauvais  
On envoie une requête http correcte au serveur cependant le format de la date est différent de « YYYY-MM-DD hh ». Aucune des données n’est enregistrée avec ce format donc aucun retour ne sera fait, même si la date dans un autre format est la même qu’en « YYYY-MM-DD hh ».

Données de test :

Requête http :   
<http://127.0.0.1:8080/SmartTerritories/dataFilter?Date=2025/20/03>

Le format de la date n’est pas bon, la requête est donc erronée et l’erreur appropriée est retournée :

{"message":"Format de date invalide. Utiliser YYYY-MM-DD hh."}

Ainsi que le code d’erreur http 400 : « Bad Request ».

Test 6 : Renvoi d’une erreur si syntaxe mauvaise dans des éléments de la requête  
On envoie une requête http avec des filtres (Capteur / TypeDeDonnee / Date) qui ne correspondent pas à ces trois-là. La requête cherchera donc une donnée avec une entité qui n’existe pas.

Données de test :

Requête http :   
<http://127.0.0.1:8080/SmartTerritories/dataFilter?Capteur=Saint-Serge&TypeDeDonnee=Temperature&Valeur=20>

« Valeur » n’est pas un critère qui est acceptée par l’API, l’erreur est donc récupérée et renvoyée avec un code d’erreur et un message :

{"message":"Paramètres invalides détectés. Seuls 'Capteurs', 'TypeDeDonnee' et 'Date' sont autorisés."}

Ainsi que le code d’erreur http 400 : « Bad Request »

Test 7 : Test de la limite présente en critères possibles

J’ai décidé d’ajouter une limite de données si l’utilisateur souhaite limiter le nombre de données qui lui sont retournées. Il faut donc vérifier que cette limite n’est pas égale à 0 ou négative.

*Test 7.1* *🡺*

Données de test :

Requête http :

<http://127.0.0.1:8080/SmartTerritories/dataFilter?Limite=4>

On retrouve donc les 4 premières données de la table, ainsi que le code http 200 : success.

*Test 7.2 🡺*

Données de test :

Requête http :   
<http://127.0.0.1:8080/SmartTerritories/dataFIlter?Limite=0>

OÙ

Requête http :   
<http://127.0.0.1:8080/SmartTerritories/dataFIlter?Limite=-5>

On retrouve le code d’erreur http 400 avec le message : « Le paramètre ‘Limite’ doit être positif »

Test 8 : Envoi d’une date et d’une plage de date dans la requête

Dans le cas où une date (seule) est entrée avec une plage de dates, le code ignore cette date, on retourne donc un code et un message en conséquence

Données de test :

Requête http :   
[http://127.0.0.1:8080/SmartTerritories/dataFilter?Date=2025-03-10 15&DateDebut=2025-03-10 15&DateFin=2025-04-25 10](http://127.0.0.1:8080/SmartTerritories/dataFilter?Date=2025-03-10%2015&DateDebut=2025-03-10%2015&DateFin=2025-04-25%2010)

On retrouve le code d’erreur 206 et le message : « La date (simple) sera ignorée car vous avez entré une date de début et/ou de fin »

**5- Résultats attendus**

Test 1 : On attend le code retour statut http 200 (success) avec les données correspondant à la requête

Test 2.1 : On attend le code retour statut http 200 (success) avec les données correspondant à la requête

Test 2.2 : On attend le code retour statut http 200 (success) avec les données correspondant à la requête

Test 2.3 : On attend le code retour statut http 200 (success) avec les données correspondant à la requête

Test 2.4 : On attend le code retour statut http 400 : bad request, ainsi que le message : « Il ne peut pas y avoir plusieurs dates de début »

Test 2.5 : On attend le code retour statut http 400 : bad request, ainsi que le message : « Il ne peut pas y avoir plusieurs dates de fin »

Test 3 : Retour statut http 303 : see other, ainsi que le message : « ATTENTION : vous êtes sur le point d'afficher toutes les données et la base, utiliser la requête : "/SmartTerritories/allData" si c'est votre demande. »

Test 4 : Retour statut http 444 : no response, ainsi que le message : « Aucune donnée trouvée avec ces critères »

Test 5 : Retour statut http 400 : bad request, ainsi que le message « Format de date invalide. Utiliser YYYY-MM-DD hh. »

Test 6 : Retour statut http 400 : bad request, ainsi que le message « Paramètres invalides détectés.»

Test 7.1 : Retour statut http 200 (success) avec les données correspondant à la requête.

Test 7.2 : Retour statut http 400 : bad request, ainsi que le message : « Le paramètre Limite doit être positif. »

Test 8 : Retour statut http 206 : partial content, ainsi que le message : « La date (simple) sera ignorée car vous avez entré une date de début et/ou de fin »

**6- Moyens à mettre en œuvre**

Pour effectuer ces tests il faudra se servir de la machine virtuelle servant de test au code. Un échantillon de données sera inscrit sur une base de données de test, ainsi on pourra retourner les données voulues.

Avant d’effectuer ces tests, il faudra démarrer l’API dans le terminal de la machine virtuelle. Il faudra se positionner dans le dossier : « Documents/API-ProgObjet » (cd Documents/API-ProgObjet) puis faire la commande : « node . » pour lancer l’API. Ensuite on pourra se positionner dans le navigateur de la machine pour faire les requêtes et observer les résultats.

Pour vérifier le code http qui sera retourné et le visualiser, on peut ouvrir, dans les paramètres supplémentaires du navigateur (ici Firefox), situés en dessous du bouton de fermeture de page, l’outil de développement web.

**Annexe : Données retournées par les tests**

**Test 1 : Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.**

**Test 2.1 : Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.**

**Test 2.2 : Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.**

**Test 2.3 : Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.**

**Test 2.4 : **

**Test 2.5 : **

**Test 3 : **

**Test 4 : **

**Test 5 : **

**Test 6 : **

**Test 7.1 : Une image contenant texte, capture d’écran, menu, Police

Le contenu généré par l’IA peut être incorrect.**

**Test 7.2 : **

**Test 8 : **