**Plan de test Classe CModel : Projet Smart Territories**

**1- Identification du test**

Tests\_Classe-CModel

**2- Référence du module testé**

Ces tests seront effectués sur les méthodes de la classe CModel, principalement les méthodes de récupération des données ainsi que la création du modèle. Ces méthodes permettent de mettre en place le modèle qui servira pour tout le modèle, puis l’utiliser pour récupérer et insérer les données dans la base.

**3- Objectif du test**

Ce test vérifiera toutes les options lors de la création du modèle mongoose, lien à la base de données MongoDB. En prenant en compte les différentes possibilités. Les erreurs ne sont pas prises en charge dans cette classe étant donné qu’elles sont étudiées dans la classe correspondant à l’API.

**4- Procédure du test**

*Test 1.1 : Création d'un modèle par défaut*

Ce test correspond au constructeur de la classe, ainsi à la création d’une nouvelle instance de modèle mongoose, avec les valeurs par défaut pour les noms de modèle et de collection, ainsi que le schéma mongoose.

Avec cette mise en œuvre on reçoit dans la console, les lignes :

« Collection choisie pour le modèle: Test\_Database  
Model { dbModel } »

*Test 1.2 : Création d'un modèle personnalisé*

Ce test correspond au constructeur de la classe, ainsi à la création d’une nouvelle instance de modèle mongoose, avec des valeurs définies (il est aussi possible de les changer) pour les noms de modèle et de collection, ainsi que le schéma mongoose.

Avec cette mise en œuvre on reçoit dans la console, les lignes :

« Collection choisie pour le modèle: newCollection  
Model { Custom\_Model } »

Test 2 : Retour de données avec la méthode "getAllMesures()"

Ce test correspond à la méthode retournant toutes les données de la base, une connexion à cette même base à été créée au préalable spécialement pour les tests. Ainsi on se connecte à la base par défaut, puis on retourne toutes les valeurs de celle-ci. Un extrait des données retournées se trouve en annexe.

*Test 3.1 : Retour de données avec des critères correctes (correspondant à la base)*

Ce test correspond à la méthode de retour de données filtrées. Ce même filtre correspondant aux contenant des données que l’on veut retourner. La vérification de tous les critères est faite dans le plan de test de l’API. On utilisera donc des filtres simples pour cette partie des plans de test.   
Le filtre est donc le suivant : « Capteur = Roseraie », on aura aussi une limite de données retournées : 3.

Avec la mise en œuvre présente en annexe et dans la partie 6, on obtient le résultat suivant :

« Collection choisie pour le modèle: Test\_Database  
Filtre utilisé : { Capteur: 'Roseraie' }  
Test 3.1 - Résultats : [  
 {  
 \_id: new ObjectId('68075dccc406b3aac3b7c518'),  
 Capteur: 'Roseraie',  
 TypeDeDonnee: 'Temperature',  
 Date: '2025-03-20 15',  
 Valeur: 17  
 },  
 {  
 \_id: new ObjectId('68075de8c406b3aac3b7c519'),  
 Capteur: 'Roseraie',  
 TypeDeDonnee: 'Temperature',  
 Date: '2025-03-10 15',  
 Valeur: 13  
 },  
 {  
 \_id: new ObjectId('680bed9d97911d8360c458b7'),  
 Capteur: 'Roseraie',  
 TypeDeDonnee: 'PM10',  
 Date: '2025-04-25 14',  
 Valeur: 80  
 }  
] »

Ainsi trois données qui contiennent le capteur « Roseraie ». Le retour du modèle est fait pour savoir que l’on se trouve bien dans la base de test.

*Test 3.2 : Aucun résultat lors du retour*

Ce test correspond à la méthode de retour de données filtrées. Ce même filtre correspondant aux contenant des données que l’on veut retourner. Cependant il faut aussi tester le retour donné si les critères ne sont pas bons. On effectue donc un filtre par TypeDeDonnee « Météo », non présent dans la base, le retour est donc vide.

Avec la mise en œuvre présente en annexe et dans la partie 6, on obtient le résultat suivant :

« Collection choisie pour le modèle: Test\_Database  
Filtre utilisé : { TypeDeDonnee: 'Météo' }  
Test 3.2 - Résultats : [] »

*Test 3.3 : Filtre multi-critères*

Ce test correspond à la méthode de retour de données filtrées. Ce même filtre correspondant aux contenant des données que l’on veut retourner. Plusieurs critères peuvent être introduis dans la requête, on doit donc tout d’abord tester un retour si plusieurs filtres ont été inscrits dans celle-ci. On effectue donc un filtre par Capteur « Roseraie » et TypeDeDonnee « Temperature ».

Avec la mise en œuvre présente en annexe et dans la partie 6, on obtient le résultat suivant :

« Collection choisie pour le modèle: Test\_Database  
Filtre utilisé : { Capteur: 'Roseraie', TypeDeDonnee: 'Temperature' }  
Test 3.3 - Résultats : [  
 {  
 \_id: new ObjectId('68075dccc406b3aac3b7c518'),  
 Capteur: 'Roseraie',  
 TypeDeDonnee: 'Temperature',  
 Date: '2025-03-20 15',  
 Valeur: 17  
 },  
 {  
 \_id: new ObjectId('68075de8c406b3aac3b7c519'),  
 Capteur: 'Roseraie',  
 TypeDeDonnee: 'Temperature',  
 Date: '2025-03-10 15',  
 Valeur: 13  
 },  
 {  
 \_id: new ObjectId('680f617838db2b78143d67f1'),  
 Capteur: 'Roseraie',  
 TypeDeDonnee: 'Temperature',  
 Date: '2025-04-28 13',  
 Valeur: 18  
 }  
] »

**5- Résultats attendus**

Test 1.1 : Connexion à la base établie, création d'un modèle par défaut :   
« Collection choisie pour le modèle: Test\_Database  
Model { dbModel }» Qui nous explique les critères choisis pour le modèle.

Test 1.2 : Connexion à la base établie, création d'un modèle personnalisé, critères de modèle :   
« Collection choisie pour le modèle: newCollection  
Model { Custom\_Model } »

Test 2 : Retour de données avec la méthode "getAllMesures()"  
Connexion à la base établie puis retour de toutes les données de celle-ci. (Vérifiable en se connectant à MongoDB sur la machine de test : Nouveau terminal 🡪 « mongo » 🡪 « use Test\_Database » 🡪 « db.Test\_Database.find().forEach(printjson) »)

Test 3.1 : Retour de données avec des critères correctes (correspondant à la base)  
Connexion à la base établie, retour des données en fonction des critères (3 données ayant comme Capteur : « Roseraie »).

Test 3.2 : Aucun résultat lors du retour  
Connexion à la base établie, retour de données vide : « [ ] »

Test 3.3 : Filtre multi-critères  
Connexion à la base établie, retour des données ayant comme Capteur « Roseraie », TypeDeDonnee « Temperature » et avec une limite de 3 données.

**6- Moyens à mettre en œuvre**

Pour effectuer ces tests il faudra se servir de la machine virtuelle servant de test au code.

Il faudra se positionner dans le code database.js (Dossier Documents/API-ProgObjet/Tests) via un terminal (commande « cd Documents/API-ProgObjet/Tests » pour changer de dossier et « nano test-CModel.js » pour ouvrir le fichier) ou alors l’ouvrir par un explorateur de fichiers.

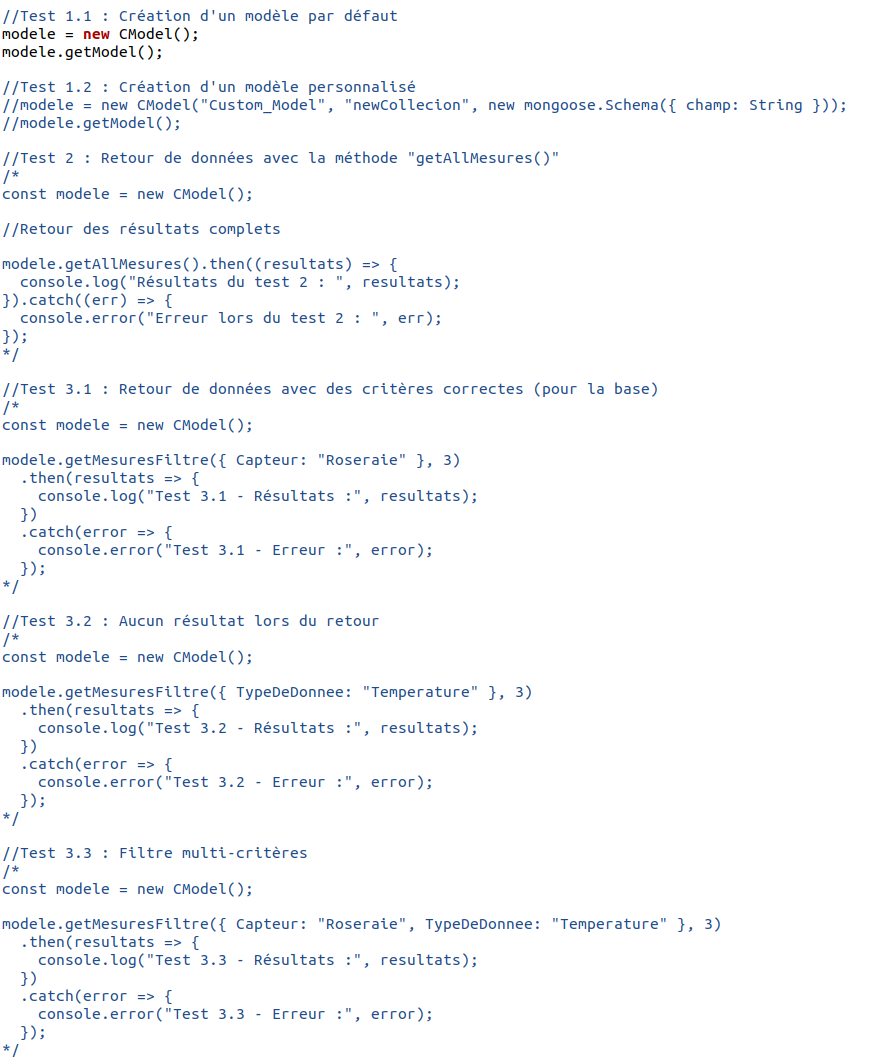
Pour le bon déroulement de ces tests, il faudra modifier le code en fonction de la fiche de test présente à la fin du fichier.

Il faudra mettre en commentaire ( « // ») les parties qui ne correspondent pas au code du test (voir les captures d’écran en annexe)

Pour démarrer le code, il faudra entrer, dans le terminal (connecté au répertoire de test) : « node test-CModel.js »

Annexe : Fiche de test et programme modifié

Test 1.1 :



Test 1.2 :

Une image contenant texte, capture d’écran, Police, document

Le contenu généré par l’IA peut être incorrect.

Test 2 :

Une image contenant texte, capture d’écran, lettre, Police

Le contenu généré par l’IA peut être incorrect.

Test 3.1 :

Une image contenant texte, capture d’écran, Police, lettre

Le contenu généré par l’IA peut être incorrect.

Test 3.2 :

Une image contenant texte, capture d’écran, Police, document

Le contenu généré par l’IA peut être incorrect.

Test 3.3 :

Une image contenant texte, capture d’écran, Police, document

Le contenu généré par l’IA peut être incorrect.

Annexe : Retour du test 2

