Procès-Verbal de test : CModel

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Élément testé : | | Ces tests seront effectués sur les méthodes de la classe CDatabase, principalement la méthode de connexion à la base MongoDB. Cette méthode permet de se connecter à la base à l’aide des éléments donnés dans le constructeur. Elle retournera aussi les erreurs dans le cas d’un problème de connexion. | | | | | | |
| Objectif du test : | | Ce test vérifiera toutes les options lors de la connexion à la base de données MongoDB. En prenant en compte les différentes erreurs possibles. | | | | | | |
| Nom du testeur : | | Théo Bourgoin | | | Date : | | 21/05/2025 | |
| Moyens mis en œuvre : | | Logiciel : BDD MongoDB, bibliothèque mongoose. | | Matériel : Machine virtuelle hôte | | Outil de développement : Node JS & Express. | | |
| Procédure du test : | | | | | | | | |
| Id | Description du vecteur de test | | Résultat attendu | | Résultat obtenu  (voir le document de plan de test sur le github pour avoir l’échantillon exact des données) | | | Validation (O/N) |
| 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. | | Connexion établie à la base, retour du nom de la collection et du modèle (par défaut). | | « Collection choisie pour le modèle: Test\_Database  Model { dbModel } » | | | O |
| 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. | | Connexion établie à la base, retour du nom de la collection et du modèle personnalisés. | | « Collection choisie pour le modèle: newCollection  Model { Custom\_Model } » | | | O |
| 2 | 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. | | 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) ») | | Données de toute la base (entière) en format JSON (voir annexe) | | | O |
| 3.1 | 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. | | 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 »). | | « 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  }  ] » | | | O |
| 3.2 | 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. | | Connexion à la base établie, retour de données vide : « [ ] » | | « Collection choisie pour le modèle: Test\_Database  Filtre utilisé : { TypeDeDonnee: 'Météo' }  Test 3.2 - Résultats : [] » | | | O |
| 3.3 | 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 ». | | Connexion à la base établie, retour des données ayant comme Capteur « Roseraie », TypeDeDonnee « Temperature » et avec une limite de 3 données. | | « 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  }  ] » | | | O |
| Conclusion du test : | | Les tests des méthodes n’ont donné aucune erreur, ils ont tous été validés sans problème. On en conclut que la classe fonctionne sans problème, elle est donc validée. | | | | | | |

Annexe : Retour du test 2

