Contexte : MediaTek86
Application : MediatekDocuments (application de bureau C#) exploitant rest_mediatekdocuments (API REST en PHP)

Plan de tests

Tests unitaires sur les classes du package model

But du test	**Action de contrôle**	**Résultat attendu**	**Bilan**
Contrôler le constructeur de la classe Abonnement pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Abonnement complet et lire chacune des propriétés : • IdAbonnement • IdRevue • TitreRevue • DateCommande • DateFinAbonnement • Montant	Propriétés égales aux valeurs fournies.	OK
Contrôler la désérialisation JSON de la classe Abonnement pour voir si toutes les propriétés sont correctement peuplées.	Désérialiser une chaîne JSON venant de l'API REST représentant un objet de type Abonnement : {"idAbonnement":"00027","idRevue":"10008","titreRevue":"L'Obs","dateCommande":"2025-04-10","dateFinAbonnement":"2025-04-30","montant":30.0} Et lire chacune de ses propriétés.	Propriétés correspondant exactement aux valeurs du JSON.	OK
Contrôler la méthode métier *GetAbonnementsExpirantDans30Jours* : cas où **un abonnement expire** dans les 30 jours.	Appeler *GetAbonnementsExpirantDans30Jours* sur une liste contenant : - un abo expirant dans 10 jours - un abo déjà expiré - un abo expirant dans 60 jours	La liste retournée contient une seule entrée (l'abonnement expirant dans 10 jours) et inclut le titre "Telerama".	OK

Contrôler la méthode métier *GetAbonnementsExpirantDans30Jours* : cas où **aucun abonnement expire** dans les 30 jours.	Appeler *GetAbonnementsExpirantDans30Jours* sur une liste contenant : - un abo déjà expiré depuis 30 jours - un abo expirant dans 90 jours	La liste retournée est vide.	OK
Contrôler le constructeur de la classe Dvd pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Dvd complet et lire chacune des propriétés : • Id • Titre • Image • Duree • Realisateur • Synopsis • IdGenre • Genre • IdPublic • Public • IdRayon • Rayon	Toutes les propriétés valent les valeurs passées en argument.	OK
Contrôler la désérialisation JSON de la classe Dvd pour voir si toutes les propriétés sont correctement peuplées.	Désérialiser une chaîne JSON venant de l'API REST représentant un objet de type Dvd et lire chacune de ses propriétés.	Propriétés correspondant exactement aux valeurs du JSON.	OK
Contrôler le constructeur de la classe Categorie pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Categorie complet et lire chacune des propriétés : • Id • Libelle	Toutes les propriétés valent les valeurs passées en argument.	OK

Contrôler la méthode ToString() de la classe Categorie pour voir si elle retourne le libelle.	Créer un objet de type Categorie avec libelle contenant : "Aventures".	La chaîne retournée est "Aventures".	OK
Contrôler le constructeur de la classe Commande pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Commande complet et lire chacune des propriétés : • Id • Titre • IdLivreDvd • DateCommande • NbExemplaires • Suivi	Toutes les propriétés valent les valeurs passées en argument.	OK
Contrôler la désérialisation JSON de la classe Commande pour voir si toutes les propriétés sont correctement peuplées.	Désérialiser une chaîne JSON venant de l'API REST représentant un objet de type Commande et lire chacune de ses propriétés.	Propriétés correspondant exactement aux valeurs du JSON.	OK
Contrôler le constructeur de la classe Document pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Document complet et lire chacune des propriétés : • Id • Titre • Image • IdGenre • Genre • IdPublic • Public • IdRayon • Rayon	Toutes les propriétés valent les valeurs passées en argument.	OK

Contrôler le constructeur de la classe Etat pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Etat complet et lire chacune des propriétés : • Id • Libelle	Toutes les propriétés valent les valeurs passées en argument.	OK
Contrôler le constructeur de la classe Exemplaire pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Exemplaire complet et lire chacune des propriétés : • Numero • DateAchat • Photo • IdEtat • IdDocument	Toutes les propriétés valent les valeurs passées en argument.	OK
Contrôler le constructeur de la classe Genre pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Genre complet et lire chacune des propriétés : • Id • Libelle	Toutes les propriétés valent les valeurs passées en argument.	OK
Contrôler la méthode ToString() de la classe Genre pour voir si elle retourne le libelle.	Créer un objet de type Genre avec libelle contenant : "Comédie".	La chaîne retournée est "Comédie".	OK
Contrôler le constructeur de la classe Livre pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Livre complet et lire chacune des propriétés : • Id • Titre • Image • Isbn • Auteur • Collection • IdGenre	Toutes les propriétés valent les valeurs passées en argument.	OK

	• Genre • IdPublic • Public • IdRayon • Rayon		
Contrôler le constructeur de la classe Public pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Public complet et lire chacune des propriétés : • Id • Libelle	Toutes les propriétés valent les valeurs passées en argument.	OK
Contrôler la méthode ToString() de la classe Public pour voir si elle retourne le libelle.	Créer un objet de type Public avec libelle contenant : "Tous publics".	La chaîne retournée est "Tous publics".	OK
Contrôler le constructeur de la classe Rayon pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Rayon complet et lire chacune des propriétés : • Id • Libelle	Toutes les propriétés valent les valeurs passées en argument.	OK
Contrôler la méthode ToString() de la classe Rayon pour voir si elle retourne le libelle.	Créer un objet de type Rayon avec libelle contenant : "Littérature étrangère".	La chaîne retournée est "Littérature étrangère".	OK
Contrôler le constructeur de la classe Revue pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Revue complet et lire chacune des propriétés : • Id • Titre • Image	Toutes les propriétés valent les valeurs passées en argument.	OK

	• IdGenre • Genre • IdPublic • Public • IdRayon • Rayon • Periodicite • DelaiMiseADispo		
Contrôler la désérialisation JSON de la classe Revue pour voir si toutes les propriétés sont correctement peuplées.	Désérialiser une chaîne JSON venant de l'API REST représentant un objet de type Revue et lire chacune de ses propriétés.	Propriétés correspondant exactement aux valeurs du JSON.	OK
Contrôler le constructeur de la classe Utilisateur pour voir si toutes les propriétés sont correctement initialisées.	Instancier un objet Utilisateur complet et lire chacune des propriétés : • Id • Nom • Login • Service	Toutes les propriétés valent les valeurs passées en argument.	OK

Tests fonctionnels dans Postman (fonctionnalités de l'API)

But du test	**Action de contrôle**	**Résultat attendu**	**Bilan**
Contrôler le bon fonctionnement de la route GET /abonnement.	Définir un script de test Post-response dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la	• Status = 200 • Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms	OK

	requête Envoyer une requête HTTP GET sur {{baseUrl}}/abonnement		
Contrôler le bon fonctionnement de la route GET /livre.	Définir un script de test Post-response dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la requête Envoyer une requête HTTP GET sur {{baseUrl}}/livre	• Status = 200 • Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms	OK
Contrôler le bon fonctionnement de la route GET /dvd.	Définir un script de test Post-response dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la requête Envoyer une requête HTTP GET sur {{baseUrl}}/dvd	• Status = 200 • Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms	OK
Contrôler le bon fonctionnement de la route GET /revue.	Définir un script de test Post-response dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la requête Envoyer une requête HTTP GET sur {{baseUrl}}/revue	• Status = 200 • Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms	OK
Contrôler le bon fonctionnement de la route	Définir un script de test Post-response	• Status = 200	OK

GET /exemplaire (id=00001).	dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la requête Définir un script de test Post-response dans la requête qui : • Vérifie que le tableau soit non vide • Vérifie que chaque exemplaire a bien l'ID demandé (id=00001) • Vérifie que chaque exemplaire contient les bonnes clés Envoyer une requête HTTP GET sur {{baseUrl}}/revue	• Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms • Le tableau est non vide • Les exemplaires ont l'ID 00001 • Les exemplaires ont les bonnes clés JSON	
Contrôler le bon fonctionnement de la route GET /commande.	Définir un script de test Post-response dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la requête Envoyer une requête HTTP GET sur {{baseUrl}}/commande	• Status = 200 • Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms	OK
Contrôler le bon fonctionnement de la route GET /commandeparid (id=00002).	Définir un script de test Post-response dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la requête	• Status = 200 • Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms • Le tableau est non vide • Les commandes ont les	OK

	Définir un script de test Post-response dans la requête qui : • Vérifie que le tableau soit non vide • Vérifie que chaque commande contient les bonnes clés Envoyer une requête HTTP GET sur {{baseUrl}}/commandeparid	bonnes clés JSON	
Contrôler le bon fonctionnement de la route POST /src/index.php?table=connexion Contrôler le bon fonctionnement de la connexion utilisateur à l'application.	Définir un script de test Post-response dans la collection qui : • Vérifie le statut HTTP = 200 • Vérifie que la réponse est un JSON valide • Vérifie que le header soit valide • Vérifie le temps de réponse de la requête Définir un script de test Post-response dans la requête qui : • Vérifie que le tableau soit non vide • Vérifie la réponse et les champs utilisateurs. Envoyer une requête HTTP GET sur {{baseUrl}}/src/index.php?table=connexion	• Status = 200 • Le corps de la réponse est parsable en JSON • le header Content-Type est application/json • Temps de réponse < 500ms • Le tableau est non vide • La réponse reçoit bien les champs utilisateurs importants : id, nom, login et service.	OK
Contrôler le bon fonctionnement de l'échec de la connexion utilisateur à l'application lorsque l'utilisateur entre des identifiants incorrects.	Définir un script de test Post-response dans la requête qui : • Vérifie que le code métier = 401 • Vérifie le message d'erreur • Vérifie que la clé résultat est vide	• Status code métier = 401 • Message d'erreur = authentification incorrecte • La clé result est vide	OK