Projet P3 - Openclassrooms - Testez l'implémentation d'une nouvelle fonctionnalité .NET

Protocoles de test

Tests Unitaires :

Objectif Test 1 : « Le produit doit porter un nom – MissingName »

Test 1 : Cas de spécification : Erreur "MissingName"

- **Étape 1 (Arrange) :**
  - On crée un produit `pVMTest` avec une valeur `null` pour l'attribut `Name`.
  - On utilise l'outil `Validator` lié à la classe des annotations pour tester le cas.
- **Étape 2 (Act) :**
  - On lance le test de validation pour `pVMTest`.
  - On récupère les résultats de la validation dans une liste.
  - On place les éléments dans un objet booléen.
- **Étape 3 (Assert) : Vérification**
  - L'objet doit être non-validé (false).
  - Il ne doit y avoir qu'un seul message dans la liste des messages d'erreur.
  - Le message d'erreur retourné dans la liste doit être « MissingName ».

Objectif Test 2 : « Le champ Prix est obligatoire - MissingPrice »

Test 1 : Cas de spécification - Erreur "MissingPrice"

- **Étape 1 (Arrange) :**
  - On crée un produit `pVMTest` avec une valeur `null` pour l'attribut `Price`.
  - On utilise la classe `ValidationContext` pour récupérer les résultats d'une tentative de validation.
  - Les résultats sont enregistrés dans une liste.
- **Étape 2 (Act) :**
  - On teste la validation via la méthode `TryValidateObject` qui prend les trois éléments en paramètre (l'instance, le contexte et les résultats).
- **Étape 3 (Assert) : Vérification**
  - On vérifie que la validation échoue (false).
  - Il ne doit y avoir qu'un seul message dans la liste des messages d'erreur.
  - Le message d'erreur retourné dans la liste doit être « MissingPrice ».

Objectif Test 3 : « Le prix doit être un nombre décimal – PriceNotANumber »

Test 1 : Cas de spécification - Erreur "PriceNotANumber" & "PriceNotGreaterThanZero "

- **Étape 1 (Arrange) :**
    - On crée un produit `pVMTest` avec une valeur non-numerique pour l'attribut `Price`.
    - On utilise la classe `ValidationContext` pour récupérer les résultats d'une tentative de validation.
    - Les résultats sont enregistrés dans une liste.
- **Étape 2 (Act) :**
    - On teste la validation via la méthode `TryValidateObject` qui prend les trois éléments en paramètre (l'instance, le contexte et les résultats).
- **Étape 3 (Assert) : Vérification**
    - On vérifie que la validation échoue (false).
    - Il doit y avoir deux messages dans la liste des messages d'erreur.
    - Les messages d'erreur retournés dans la liste doivent être « PriceNotANumber » et « PriceNotGreaterThanZero ».

Objectif Test 4 : « Le prix doit être supérieur à zéro – PriceNotGreaterThanZero »
Test 1 : Cas de spécification - Erreur "PriceNotGreaterThanZero"

- **Étape 1 (Arrange) :**
    - On crée un produit `pVMTest` avec une valeur zéro pour l'attribut `Price`.
    - On utilise la classe `ValidationContext` pour récupérer les résultats d'une tentative de validation.
    - Les résultats sont enregistrés dans une liste.
- **Étape 2 (Act) :**
    - On teste la validation via la méthode `TryValidateObject` qui prend les trois éléments en paramètre (l'instance, le contexte et les résultats).
- **Étape 3 (Assert) : Vérification**
    - On vérifie que la validation échoue (false).
    - Il ne doit y avoir qu'un seul message dans la liste des messages d'erreur.
    - Le message d'erreur retourné dans la liste doit être « PriceNotGreaterThanZero ».

Objectif Test 5 : « Le champ Quantité est obligatoire - MissingQuantity »
Test 1 : Cas de spécification - Erreur "MissingQuantity"

- **Étape 1 (Arrange) :**
    - On crée un produit `pVMTest` avec une valeur `null` pour l'attribut `Stock`.
    - On utilise la classe `ValidationContext` pour récupérer les résultats d'une tentative de validation.
    - Les résultats sont enregistrés dans une liste.
- **Étape 2 (Act) :**
    - On teste la validation via la méthode `TryValidateObject` qui prend les trois éléments en paramètre (l'instance, le contexte et les résultats).

- **Étape 3 (Assert) : Vérification**
    - On vérifie que la validation échoue (false).
    - Il ne doit y avoir qu'un seul message dans la liste des messages d'erreur.
    - Le message d'erreur retourné dans la liste doit être « MissingStock ».

Objectif Test 6 : « La quantité doit être un nombre entier – QuantityNotAnInteger »

Test 1 : Cas de spécification - Erreur "QuantityNotAnInteger"

- **Étape 1 (Arrange) :**
    - On crée un produit `pVMTest` avec une valeur non-entière pour l'attribut `Stock`.
    - On utilise la classe `ValidationContext` pour récupérer les résultats d'une tentative de validation.
    - Les résultats sont enregistrés dans une liste.
- **Étape 2 (Act) :**
    - On teste la validation via la méthode `TryValidateObject` qui prend les trois éléments en paramètre (l'instance, le contexte et les résultats).
- **Étape 3 (Assert) : Vérification**
    - On vérifie que la validation échoue (false).
    - Il doit y avoir deux messages dans la liste des messages d'erreur.
    - Les messages d'erreur retournés dans la liste doivent être « StockNotAnInteger » et « StockNotGreaterThanZero ».

Objectif Test 7 : « La quantité doit être supérieure à zéro – QuantityNotGreaterThanZero »

Test 1 : Cas de spécification - Erreur "QuantityNotGreaterThanZero"

- **Étape 1 (Arrange) :**
    - On crée un produit `pVMTest` avec une valeur zéro pour l'attribut `Stock`.
    - On utilise la classe `ValidationContext` pour récupérer les résultats d'une tentative de validation.
    - Les résultats sont enregistrés dans une liste.
- **Étape 2 (Act) :**
    - On teste la validation via la méthode `TryValidateObject` qui prend les trois éléments en paramètre (l'instance, le contexte et les résultats).
- **Étape 3 (Assert) : Vérification**
    - On vérifie que la validation échoue (false).
    - Il ne doit y avoir qu'un seul message dans la liste des messages d'erreur.
    - Le message d'erreur retourné dans la liste doit être « StockNotGreaterThanZero ».

Tests d'Intégration :

Objectif Test 1 : Vérifier l'ajout de produit dans la base de données

Test 1 : Cas de spécification - Ajout correct d'un produit via la méthode SaveProduct

- **Étape 1 (Arrange) :**
  - On crée un produit `ProductViewModelTest` avec les attributs nécessaires (`Name` , `Price` , `Stock`).
  - On configure le contexte et les services nécessaires pour l'ajout du produit.
- **Étape 2 (Act) :**
  - On ajoute le produit à la base de données via la méthode `SaveProduct` .
- **Étape 3 (Assert) : Vérification**
  - On vérifie que le produit a été ajouté à la base de données.
  - Le produit doit être supprimé après vérification pour maintenir l'intégrité des tests.

Objectif Test 2 : Vérifier la suppression de produit dans la base de données
Test 1 : Cas de spécification - Suppression correcte d'un produit via la méthode DeleteProduct

- **Étape 1 (Arrange) :**
  - On crée et ajoute un produit `ProductViewModelDeleteTest` à la base de données.
  - On configure le contexte et les services nécessaires pour la suppression du produit.
- **Étape 2 (Act) :**
  - On supprime le produit de la base de données via la méthode `DeleteProduct` .
- **Étape 3 (Assert) : Vérification**
  - On vérifie que le produit a été supprimé de la base de données.

Objectif Test 3 : Vérifier la récupération des infos produit dans la base de données
Test 1 : Cas de spécification - Obtention des infos produit via la méthode GetProduct

- **Étape 1 (Arrange) :**
  - On crée et ajoute un produit `ProductViewModelGetTest` à la base de données.
  - On configure le contexte et les services nécessaires pour la récupération du produit.
- **Étape 2 (Act) :**
  - On récupère le produit de la base de données via la méthode `GetProduct` .
- **Étape 3 (Assert) : Vérification**
  - On vérifie que les informations récupérées correspondent au produit ajouté.
  - Le produit doit être supprimé après vérification pour maintenir l'intégrité des tests.

Objectif Test 4 : Vérifier la mise à jour du stock d'un produit dans la base de données

Test 1 : Cas de spécification - Mise à jour de la valeur de stock d'un produit via la méthode SaveProduct

- **Étape 1 (Arrange) :**
  - On crée un produit `ProductViewModelUpdateTest` avec les attributs nécessaires (`Name`, `Price`, `Stock`).
  - On ajoute le produit à la base de données via la méthode `SaveProduct`.
- **Étape 2 (Act) :**
  - On met à jour le stock du produit via la méthode `UpdateStock` de `ProductService`.
- **Étape 3 (Assert) : Vérification**
  - On vérifie que le stock du produit a bien été mis à jour dans la base de données.
  - Le nouveau stock doit correspondre à la valeur mise à jour.
  - Le produit doit être supprimé après vérification pour maintenir l'intégrité des tests.