Laboratoire: semaine 6

# Objectif(s)

* Convertir les méthodes Action des controllers et Repositories en asynchrones
* Ajouter des tests unitaires en TDD

# Créer le repo pour les laboratoires individuels

## Création de la branche de la séance

1. Dans Gitkraken, clonez votre repository de laboratoire
2. Créez la branche de la séance **S06\_Async\_TDD**
3. *Comment-commit-push*

**Répartir le travail entre les coéquipiers :**

# Méthodes et action Async

## Convertir les Repositories

1. Copiez l’Interface et la classe Repository générique afin que les méthodes soient async
2. Nommez les éléments en ajoutant **async** à la fin

**Exemple :**

public async Task AddAsync(T entity)

{

await dbSet.AddAsync(entity);

}

public async Task<T> GetAsync(int id)

{

return await dbSet.FindAsync(id);

}

1. Coipez les Interfaces et les classes des Repositories spécifiques afin que les méthodes soient async

## Convertir les controllers

1. Modifiez les procédures1Action des controllers
2. Modifiez les appels aux repositories afin qu’ils appellent les repositories **async**

**Exemple :**

public async Task<IActionResult> Index()

{

IEnumerable<Book> objList = await \_unitOfWork.Book.GetAllAsync(includeProperties:"Publisher,Subject");

return View(objList);

}

1. Modifiez les procédures1Action des controllers qui appelent des ViewModels

**Exemple :**

//GET CREATE

public async Task<IActionResult> Create()

{

IEnumerable<Subject> SubList = await \_unitOfWork.Subject.GetAllAsync();

IEnumerable<Publisher> PubList = await \_unitOfWork.Publisher.GetAllAsync();

BookVM bookVM = new BookVM()

{

Book = new Book(),

SubjectList = SubList.Select(i => new SelectListItem

{

Text = i.Name,

Value = i.Id.ToString()

….

1. Comment-Commit-Push

**1 des Coéquipiers**

# Tests Driven Development

## Créer le projet de tests pour les models

1. Dans le projets MVC, ajoutez un dossier **Sercice** et mettre une classe vide dedans (git n’aime pas les dossiers vides)
2. Ajoutez un projet de tests de type Nunit appelé **Jungle\_Models\_Tests**
3. Modifiez le namespace par défaut du projet test pour **Jungle\_Models**
4. Ajoutez une référence de projet du projet Models au Projet de tests
5. Comment-Commit-Push pour que tous aient accès au projet de tests

**1 Coéquipier ou 2 en pair programming**

## Ajouter une validation de models: Date du travel

1. Ajoutez dans le projet de tests une classe de test appelée **Travel\_Tests** ou un nom correspondant à votre classe.

Vous devez implémenter la nouvelle règle de validation en mode TDD: vous devez donc mettre votre code en commentaire (idéalement dans une région) pour le code de test ET le code dans la classe de validation afin de voir le REFACTOR si il y a lieu.

Fonctionnalité ajoutée: Lors de création d'un voyage (Travel), la **DepartureDate** doit être au minimum 10 jours dans le futur, soit aujourd'hui plus 10 jours.

**Liste des scénarios:**

* + Valide au dessus de la limite: Aujourd'hui + 20 jours
  + Valide sur la limite: Aujourd'hui + 10 jours
  + Non valide date null Throw Exception
  + Non valide date erronée: 2022-30-30 Throw Exception
  + Non valide en dessous de la limite: Aujourd'hui - 15 jours

1. En TDD, créez les tests/cas de tests et le code afin de répondre à ce besoin
2. Créez/Modifiez un Test pour tester le code existant (création de Travel seulement)
   1. Mettre le code précédent en commentaire (région) si il y a lieu, créez/Modifiez un Tests pour ajouter un des scénarios ci-dessus
   2. Exécutez le test: il devrait échoué
   3. Mettre le code actuel en commentaire (région), ajoutez SEULEMENT le code nécessaire au passage du test
   4. Exécutez le test
   5. Corrigez jusqu'à ce que ça passe si nécessaire (étapes 4 et à 6)
   6. Refactor le code si nécessaire
   7. Ré-exécutez pour valider le refactor
   8. Passez au scénario suivant

**Piste pour le code:**

1. Dans la classe **Travel,** ajoutez un attribut de validation personnalisé nommée [**ValidationDepartureDate]** à DepartureDate.
2. Dans le projet Models, créez une classe public **ValidationDepartureDate** qui hérite de ValidationAttribute

public class ValidationDepartureDate: ValidationAttribute

{

private readonly Func<DateTime> \_dateTimeMinProvider;

public **ValidationDepartureDate** ()

: this(() => DateTime.Now.AddDays(10))

{

}

public ValidationDepartureDate (Func<DateTime> dateTimeMinProvider)

{

\_dateTimeMinProvider = dateTimeMinProvider;

ErrorMessage = "Date must be minimum in ten days";

}

public override bool IsValid(object value)

{

//Code de validation ICI

}

}

**1 Coéquipier ou 2 en pair programming**

## Ajouter une validation de models: options choisies pour une réservation de travel

1. Ajoutez dans le projet de tests une classe de test appelée **TravelOptions\_Tests** ou un nom correspondant à votre classe.

Vous devez implémenter la nouvelle règle de validation en mode TDD: vous devez donc mettre votre code en commentaire (idéalement dans une région) pour le code de test ET le code dans la classe de validation afin de voir le REFACTOR si il y a lieu.

Fonctionnalité ajoutée: Valider les options ajoutées à une réservation de Travel : Maximum de 3 options, l’option choisie pour la réservation ne doit pas se répéter. Validez le nombre d’éléments dans la liste, le contenu et le sans doublon ainsi que la somme des options.

Il vous est suggéré de créer une classe publique **ChooseOptions** et le mettre dans un dossier **Service** (pour simuler le service de booking).

**Liste des scénarios du TDD:**

* + Valide aucune option choisie
  + Valide 1 option choisie
  + Valide trois options choisies (même identiques pour le 1er passage TDD)
  + Non valide quatre options choisies
  + Valide 2 options sur 3 différentes
  + Valide 3 options sur 3 différentes
  + Non valide 2 options sur 3 identiques
  + Valide 0 pour aucune option choisie
  + Valide total d’une option
  + Valide total de trois options

1. En TDD, créez les tests/cas de tests et le code afin de répondre à ce besoin
2. Créez/Modifiez un Test pour tester le code existant (création de Travel seulement)
3. Mettre le code précédent en commentaire (région) si il y a lieu, créez/Modifiez un Tests pour ajouter un des scénarios ci-dessus
4. Exécutez le test: il devrait échoué
5. Mettre le code actuel en commentaire (région), ajoutez SEULEMENT le code nécessaire au passage du test
6. Exécutez le test
7. Corrigez jusqu'à ce que ça passe si nécessaire (étapes 4 et à 6)
8. Refactor le code si nécessaire
9. Ré-exécutez pour valider le refactor
10. Passez au scénario suivant