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. Copiez 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 appellent 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. Ajoutez un projet de tests de type Nunit appelé **Jungle\_Models\_Tests**
2. Modifiez le namespace par défaut du projet test pour **Jungle\_Models**
3. Ajoutez une référence de projet du projet Models au Projet de tests
4. Comment-Commit-Push pour que tous aient accès au projet de tests

**Coéquipier 1 ou 1 et 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 s’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. Créez/Modifiez un Tests pour ajouter un des scénarios ci-dessus
   2. Exécutez le test: il devrait échouer
   3. 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. Si nécessaire, mettre le code actuel en commentaire (région) et « Refactor » le code
   7. Réexécutez pour valider le « refactor »

**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 publique **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

}

}

**Coéquipier 2 ou/et 3, seul ou en pair programming**

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

1. Créez un dossier **Service** (pour simuler un service) et créez une classe une **TravelReservationService** dedans.
2. Ajoutez une méthode **BookTravel** qui créé une réservation de Travel pour un client

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 s’il y a lieu.

Fonctionnalité ajoutée: Lors de la réservation d'un voyage, la **DepartureDate** de TRAVEL doit être au minimum 2 jours dans le futur, soit aujourd'hui plus 2 jours.

1. Ajoutez dans le projet de tests une classe de test appelée **TravelReservation\_Tests** ou un nom correspondant à votre classe.
2. En TDD, créez les tests/cas de tests et le code afin de répondre à ce besoin