OS12 – Développement C# ASP.NET
TD4 – Tests Unitaires

Ce TD est la suite directe du TD3 et utilise l'application créée précédemment.
Pensez à bien enregistrer votre projet de manière à pouvoir le retrouver lors des prochaines séances.

1 Introduction

Un avantage du MVC par rapport aux WebForms, est qu'il est beaucoup plus simple de réaliser des tests unitaires. En WebForms, la vue (`aspx`) est fortement liée au code behind (`aspx.cs`), la plupart des méthodes du code behind (lié aux événements) sont dépendantes de contextes et l'exécution d'une page est beaucoup plus compliqué.

On retrouve des problèmes similaires à WPF sans pattern MVVM.

Par exemple, lors du clic sur un bouton, il peut y avoir des traitements :

1. Avant l'initialisation de la page (`OnPreInit`)
2. A l'initialisation de la page (`OnInit`)
3. Après le chargement de la page (`OnInitComplete`)
4. Avant le chargement de la page (`OnPreLoad`)
5. Au chargement de la page (`OnLoad`)
6. Après le chargement de la page (`OnLoadComplete`)
7. Sur l'événement Clic (`OnClik`)
8. Avant le rendu (`OnPreRender`)
9. Pendant le rendu (`OnRender`)
10. Après le rendu (`OnRenderComplete`)
11. …

Bref il peut y avoir du traitement dans tous ces événements, il est donc quasiment impossible de tester proprement une page WebForms.

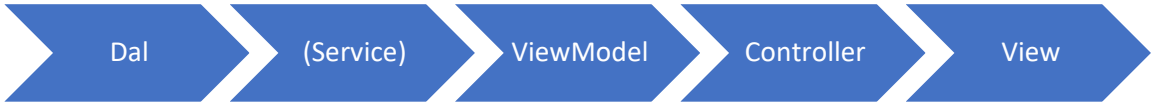
En MVC, une route redirige vers un contrôleur et le contrôleur renvois un `ActionResult`, qui peut être une vue, une redirection, une page d'erreur, ou une exception.

2 Test de la partie Vote

Comme nous l'avons fait à de nombreuses reprises au S3 lors du cours WPF, nous allons réaliser les tests de la couche d'accès aux données.

Il faut toujours tester les couches de la plus basse (proche de la base de données) à la plus haute (l'Interface Homme Machine).

Dans avons donc en général :



La première étape est de s'assurer que la base de l'application (la couche d'accès aux données) est valide.

Une fois que nous nous sommes assurés du bon fonctionnement de la couche `Dal` nous pourrons tester les couches supérieures de l'application, en se basant sur le fait que les couches inférieures sont valides.

Nous n'allons pas écrire les tests de l'ensemble de l'application, nous allons nous concentrer sur la partie vote, que nous allons tester de bout en bout.

2.1 Prérequis

Nous allons travailler dans le projet `OrganisationSoiree.Tests`. Si le projet n'existe pas (normalement il a été créé lors du TD2), le créer (Nouveau projet → Test → Projet de test unitaire).

Avant de commencer, nous avons besoin des packages nugets suivants (à ajouter au projet `OrganisationSoiree.Tests`) :

- EntityFramework
- Microsoft.AspNet.Identity.Core

Ajouter également la chaine de connexion (`SoireeContext`) depuis le fichier `Web.config` de votre projet web vers le fichier `App.config` du projet de test en ajoutant `\App_Data` après `|DataDirectory|` au milieu de la chaîne.

2.2 Test de la couche d'accès aux données

Créer un répertoire *`DAL`* dans le projet de test (`SondageSoiree.Tests`). Ajouter la classe de test unitaire *`DalUnitTest`* dans ce dossier.

Ecrire les méthodes des tests :

- `CreerSondageTest`
- `AjouterVoteTest`
- `RenvoieResultatTest`
- `RenvoieTousLesSondagesTest`

Notes :

On utilise Entity Framework directement, pour l'initialisation des données et pour la validation.
On ne créera pas de tests spécifiques pour les méthodes « `exists` » car elles sont très simples. (On rajoutera le test de ces méthodes dans les méthodes de tests « `création` »).

2.3 Test des contrôleurs

Nous allons commencer par modifier notre contrôleur afin de pouvoir tester plus de cas. Modifier le paramètre `int id` par le type nullable `int? id`.

Le paramètre `id` est optionnel (d'après la définition de la route par défaut). Le fait de ne pas mettre un type nullable dans le contrôleur fonctionne, nous avons bien une page d'erreur, mais nous ne pouvons pas gérer l'erreur finement.

Il est donc préférable d'écrire les actions des contrôleurs de la sorte :

```
public ActionResult MonAction(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    //Code du controleur...

    return View();
}
```

1. Modifier les méthodes du contrôleur vote en suivant l'exemple précédent.
2. Créer un dossier `Controllers` dans le projet de test (s'il existe déjà, supprimer son contenue)
3. Rajouter les fichiers `TestControllerBase.cs` et `VoteControllerUnitTest.cs` dans ce dossier (Cette classe contient une méthode pour définir / modifier l'utilisateur connecté.)
4. Compléter les méthodes (la méthode `VoteGetValide` est déjà complétée et sert d'exemple). Il y a deux catégories de méthodes :
    a. Les méthodes `VoteGet` (ou `ResultatGet`) qui test les requêtes GET au serveur
    b. Les méthodes `VotePost` qui teste les méthodes d'envois de données au serveur.

Notes :

Les méthodes marquée `TestInitialize` et `TestCleanup` sont exécutée avant et après chaque teste et permettant d'avoir à chaque test un sondage « neuf » et une liste d'élève.

On remarquera que nous avons oublié de gérer pleins de cas dans notre contrôleur (id n'existant pas, etc…) c'est d'ailleurs un des intérêts majeurs de faire des tests unitaires, cela nous force à réfléchir à tous les cas possibles.

Corriger le contrôleur pour qu'il soit valide avec les tests. (Il faudra surement rajouter des méthodes dans la partie Dal, donc il faudra de nouveaux tests dans la partie Dal !)

Aide :

La méthode `FillControllerContext(Controller, Eleve)` sert à simuler la connexion d'un élève à l'application.

La méthode `ValidateModel(Controller, object)` sert à actualiser le ModelState du contrôleur.

3 Test des contrôleurs, suite

Nous avons donc maintenant testé la partie vote / résultat de bout en bout.

La notion de test de contrôleurs étant nouvelle, vous allez écrire les tests unitaires des contrôleurs `SondagesController` et `RestaurantController` :

1. Créer les deux classes nécessaires (`SondageControllerTest` et `RestaurantControllerTest`)
2. Ecrire les méthodes de tests. Attention à bien découpé les méthodes, une méthode ne doit tester qu'un cas spécifique.
3. Exécuter les tests sur le code et corriger les contrôleurs en fonction des résultats. (Il sera, là encore, surement nécessaire d'ajouter des méthodes à la partie Dal).