Exercises Design Pattern

Contexte

- Une classe employé avec un id, un nom, un prénom et un salaire.
- Une classe LocalStorage qui est un Singleton et qui contient une liste d'employés. Cette classe nous permettra de simuler une persistance en BDD
- Le salaire doit être toujours supérieur à 0

Application Console

Dans une application console. Faite en sorte qu'on puisse via un menu :

- Ajouter un employé en rentrant ses informations
- Afficher la liste des employés
- Afficher un employé en rentrant son Id

Pattern MVC

Reprenez votre application console et mettre en place 3 couches (3 projets) :

- Views : Garder votre projet d'application console qui contiendra ce qui concerne l'interface. Aucun LocalStorage n'est accepté ici
- Controllers : Créez une bibliothèque de classe et créez un EmployeController qui contiendra la logique de votre application (Par exemple le contrôleur employé aura une méthode qui récupère le nom + prénom + salaire pour créer un objet Employe et le persister en LocalStorage). Aucun Console.WriteLine n'est accepté ici
- Models : Mettez la classe métier et LocalStorage dans ce projet

Pattern MVC avancé

Mettre en place cette architecture :

- DesignPatternAppV3.View : Ne change pas
- DesignPatternAppV3.Controllers : Ne fais que récupérer et vérifier les infos de la vue et formater les infos pour envoyer à la vue
- DesignPatternAppV3.Business : Contient la logique métier de l'application
- DesignPatternAppV3.Business.Contracts : Contient les interfaces qui correspondent aux classes crées dans la partie Business
- DesignPatternAppV3.Repository : Contient la logique de persistance (EmployeRepository et LocalStorage)
- DesignPatternAppV3.Repository.Contracts : Contient les interfaces qui correspondent aux classes crées dans la partie Repository
- DesignPatternAppV3.Entities : Contient la classe métiers

Avec Unity, aucun **new** d'architecture n'est acceptée (ne pas créer une instance Business par exemple mais une classe Employé OK).

Pour ce faire utiliser Unity. Installez-le sur l'application console via le gestionnaire de package nuget au niveau du projet View. Une fois installé rajouter ces lignes au début de la méthode Main :

```
// Créer le container Unity
IUnityContainer unitycontainer = new UnityContainer();

// Lie les implémentations aux interfaces correspondantes
unitycontainer.RegisterType<IEmployeService, EmployeService>();
etc…
```

Pour utiliser les implémentations il faudra ajouter un constructeur spécifique, par exemple dans la classe EmployeService (couche Business) :

```
IEmployeRepository EmployeRepository { get; }

public EmployeService(IEmployeRepository employeRepository)
{
    EmployeRepository = employeRepository;
}
```

Le constructeur sera appelé automatiquement avec la variable bien renseigné en fonction de l'implémentation de IEmployeRepository fournis pendant la création du container.

Une autre façon de récupérer une implémentation est via le container directement (utile pour notre méthode Main de l'application console) :

```
EmployeController employeController = unitycontainer.Resolve<EmployeController>();
```

Dans un deuxième temps, créez une méthode de test sur la couche service en renseignant un mock afin de tester plus facilement

```
var entrepriseService = new EmployeService(new MockStorage());

var employes = entrepriseService.GetEmployes();

Assert.AreEqual(employes.Count, 2);
```

MockStorage étant une classe pour les tests implémentant de façon très basique l'interface IEmployeRepository.