Laboratoire 1

Création d'un projet

Utiliser le Repository Git que vous avez fait. Créez-en un si ce n'est pas le cas. Vous pouvez créer une branche pour chaque semaine e.g. S01. Faites un pull de cette branche.

Vous avez besoin du zip accompagnant ce word.

Comprenez, puis roulez ces commandes. UTILISEZ l'invite de commande Windows pour que les variables fonctionnent.

```
SET SLNNAME=CEM.A20._3w6.MaisonReve.S01
SET PROJNAME=MaisonReve
dotnet new globaljson --sdk-version 5.0.100-preview.4.20258.7 --
output %SLNNAME%
dotnet new sln -o %SLNNAME%
cd %SLNNAME%
dotnet new mvc --no-https --output %PROJNAME%.Web
dotnet new classlib --output %PROJNAME%.Database
dotnet new xunit --output  %PROJNAME%.Test
dotnet sln add %PROJNAME%.Web
dotnet sln add %PROJNAME%.Database
dotnet sln add %PROJNAME%.Test
cd %PROJNAME%.Web
dotnet add reference ../%PROJNAME%.Database
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --
version 3.1.6
dotnet add package Microsoft.EntityFrameworkCore.InMemory --version 3.1.6
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design --
version 3.1.4
dotnet add package  Microsoft.EntityFrameworkCore.Design  --version 3.1.6
dotnet add package  Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation  --
version 3.1.7
dotnet add package System.Linq --version 4.3.0
echo # Projet Web de %PROJNAME% > readme.md
cd ..
cd %PROJNAME%.Test
dotnet add reference ../%PROJNAME%.Database
dotnet add reference ../%PROJNAME%.Web
dotnet add package Microsoft.EntityFrameworkCore.InMemory --version 3.1.6
echo # Projet Test de %PROJNAME% > readme.md
cd ..
cd %PROJNAME%.Database
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --
version 3.1.6
dotnet add package Microsoft.EntityFrameworkCore.InMemory --version 3.1.6
dotnet add package System.Linq --version 4.3.0
echo ==Projet DAL de %PROJNAME% > readme.md
mkdir Models
```

```
echo # Modèles de %PROJNAME%  > Models\readme.md
mkdir Context
echo # DbContext de %PROJNAME%  > Context\readme.md
mkdir Initializer
echo # Initialisateur de BD de %PROJNAME%  > Initializer\readme.md
mkdir Repository
echo # Les Repositories de %PROJNAME%  > Repository\readme.md
del Class1.cs
cd ..
echo # Solution %PROJNAME%  > readme.md
dotnet build
dotnet run  --project %PROJNAME%.Web
```

Sur cette série de commande, nous avons

- Étape 1 - On crée une solution
- Étape 2 - On crée les trois projets et on les ajoute dans la solution
- Étape 3 - On ajoute des package nuget et on référencie les projets entre eux.
- Étape 4 - On génère et on roule l'application pour vérifier que tout fonctionne.
- Vous pouvez fermer l'application une fois validé par Ctrl+C.

Ajouter une vue partielle pour la partie de navigation

On veut modifier le modèle par défaut pour sortir le <nav/> du _Layout.cshtml. Pour ce faire, on va utiliser l'outil SDK

`dotnet aspnet-codegenerator`

(documentation : https://docs.microsoft.com/fr-fr/aspnet/core/fundamentals/tools/dotnet-aspnet-codegenerator?view=aspnetcore-3.1)

Cet outil permet de générer les différentes composantes d'ASP.NET Core.

- Aller dans le projet MaisonReve.Web et sur l'invite de commande et exécutez

```
dotnet aspnet-codegenerator view  _topNav Empty -outDir Views/Shared  -partial
```

Ceci ajoutera une vue vide dans Views/Shared.

- Remplacer les fichiers dans MaisonReve.Database avec ceux contenu dans le .zip afin d'avoir accès à un repo. Ce repo utilise des données statiques comme vous avez vu dans le dernier cours.

- À la suite de ce remplacement, utilisez la commande aspnet-codegenerator pour faire vos vues. Afin de bien connaître la commande, utilisez -h pour l'aide et aspnet-codegenerator afin de créer les vues Index, Create, Edit, Delete, Details.

En exemple ici :

```
PS C:\WS\t-gv\3w6\3w6-maisonreve-exemple\CEM.A20._3w6.MaisonReve> dotnet aspnet-codegenerator -p MaisonReve.Web -h


Usage: aspnet-codegenerator [arguments] [options]

Arguments:
  generator  Name of the generator. Check available generators below.

Options:
  -p|--project            Path to .csproj file in the project.
  -n|--nuget-package-dir
  -c|--configuration      Configuration for the project (Possible values: Debug/ Release)
  -tfm|--target-framework  Target Framework to use. (Short folder name of the tfm. eg. net46)
  -b|--build-base-path
  --no-build

Available generators:
  area      : Generates an MVC Area.
  controller: Generates a controller.
  identity  : Generates an MVC Area with controllers and
  razorpage : Generates RazorPage(s).
  view      : Generates a view.


RunTime 00:00:04.23
```

Puis

```
PS C:\WS\t-gv\3w6\3w6-maisonreve-exemple\CEM.A20._3w6.MaisonReve> dotnet aspnet-codegenerator -p MaisonReve.Web view -h


Usage: aspnet-codegenerator [arguments] [options]

Arguments:
  generator  Name of the generator. Check available generators below.

Options:
  -p|--project            Path to .csproj file in the project.
  -n|--nuget-package-dir
  -c|--configuration      Configuration for the project (Possible values: Debug/ Release)
  -tfm|--target-framework  Target Framework to use. (Short folder name of the tfm. eg. net46)
  -b|--build-base-path
  --no-build

Selected Code Generator: view

Generator Arguments:
  viewName     : Name of the view
  templateName : The view template to use, supported view templates: 'Empty|Create|Edit|Delete|Details|List'

Generator Options:
  --model|-m                        : Model class to use
  --dataContext|-dc                 : DbContext class to use
  --referenceScriptLibraries|-scripts : Switch to specify whether to reference script libraries in the generated views
  --layout|-l                       : Custom Layout page to use
  --useDefaultLayout|-udl           : Switch to specify that default layout should be used for the views
  --force|-f                        : Use this option to overwrite existing files
  --relativeFolderPath|-outDir      : Specify the relative output folder path from project where the file needs to be generated, if not specified, file will
  --controllerNamespace|-namespace  : Specify the name of the namespace to use for the generated controller
  --partialView|-partial            : Generate a partial view, other layout options (-l and -udl) are ignored if this is specified
  --useSqlite|-sqlite               : Flag to specify if DbContext should use SQLite instead of SQL Server.


RunTime 00:00:03.01
PS C:\WS\t-gv\3w6\3w6-maisonreve-exemple\CEM.A20._3w6.MaisonReve>
```

Il faut utiliser le modèle suivant : MaisonReve.Database.Models.Database

- Si vous vous trompez, supprimer le fichier généré et recommencer.
- Suivez la documentation de la commande pour générer vos vues.
- Faites de même pour générer le contrôleur. Vous ne pouvez pas utiliser le modèle pour le contrôleur.
- Ouvrez Visual Studio Code dans le dossier contenant le .sln.
- Ajoutez les Assets après l'analyse du code par OmniSharp (dites oui, et sélectionner MaisonReve.Web!)
- Implanter les actions du contrôleur avec le repo fourni. Le repo est sous MaisonReve.Database.Repository.
- Ajouter le singleton et utiliser le dans le contrôleur.
- Pour brièveté vous pouvez implanter seulement Index, Delete et Create.
- Dans les vues, cachez le input Id, et vérifier que les liens d'action ont aussi le Id.
- Faites F5 pour tester le tout.

Les commandes peuvent être utiliser sous Visual Studio aussi.

Fin du laboratoire 1