Objectif du TP

L'objectif de ce TP est de concevoir des microservices en ASP.NET Core et de les conteneuriser en les mettant sous Docker

Préparation de l'environnement de développement

VS Code

Windows: https://s3.eu-west-3.amazonaws.com/miage-interopandinnov/VSCodeWindows.zip

Linux: https://s3.eu-west-3.amazonaws.com/miage-interopandinnov/VSCode-linux-x64.tar.gz

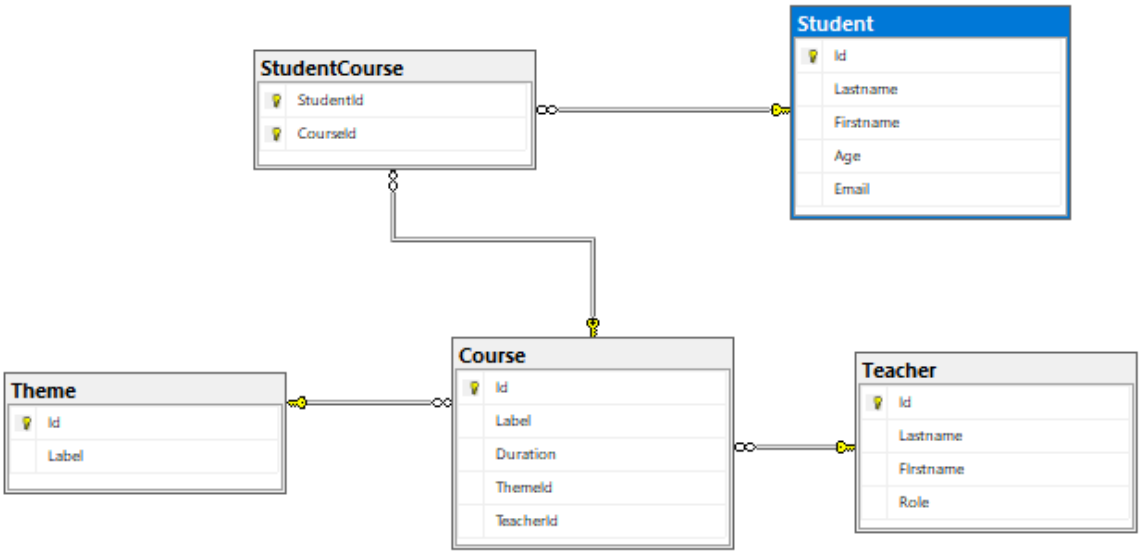
Framework .NET Core

URL: https://dotnet.microsoft.com/download/dotnet-core/2.2

1. Télécharger en fonction de votre OS (NB : Sous linux sans sudo, choisir NET Core Binaries)
2. Suivre les instructions d'installation

BDD

Schéma de la BDD



Installation

1. Installation et lancement du container docker

```
docker run -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=M1@g3.NET' --name sql1 -p 61433:1433 -v <chemin>:/data -d mcr.microsoft.com/mssql/server:2017-latest
```

NB : <chemin> correspondant au dossier de votre git clone.

2. Initialisation de la bdd

```
docker exec -it sql1 /bin/bash
```

```
/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P M1@g3.NET -i /data/Scripts/TP_MIAGE.sql
```

Travail à effectuer

Le but de ce TP est de construire de développer 2 microservices et une interface web consommant ces 2 microservices. L'ensemble de ces microservices développés en ASP.NET Core devront tourner dans des conteneurs Dockers indépendants. Ce sont les 2 microservices qui accèdent à la BDD.

Microservices :

- StudentManager : Microservice permettant d'afficher et de gérer le référentiel des étudiants
- CourseManager : Microservice permettant d'afficher les formations et d'y inscrire un étudiant.

Pour chaque microservice, il faut donc créer le modèle objet, produire la DAL et les contrôleurs permettant d'accéder aux données en REST.

L'application web avec la techno de votre choix permet donc d'effectuer les actions attendues sur les microservices par une IHM Web.

Trucs & astuces

ASP.NET Core

Lignes de commande

1. Création d'un nouveau projet API REST (Dans un dossier projet)

```
dotnet new webapi
```

2. Création d'un nouveau projet Web MVC (Dans un dossier projet)

```
dotnet new mvc
```

3. Installation d'un package nuget

```
dotnet add package <NomDuPackage>
```

4. Installer le générateur de code des contrôleurs

```
dotnet tool install --global dotnet-aspnet-codegenerator --version 2.2.3
```

NB : Penser à installer le package nuget nécessaire dans le projet

5. Générer un controller

```
dotnet aspnet-codegenerator controller -name <NomDuControleur> -m <NomDuModel> -dc
<NomDuContexte> --relativeFolderPath Controllers -api
```

- <NomDuControleur> : Nom du controller cible. Son nommage doit se terminer par Controller
- <NomDuModel> : Nom de la classe à partir de laquelle on va générer le controller
- <NomDuContexte> : Nom du context EntityFramework permettant d'accéder aux données. Ce sera le même nom de contexte pour tous les contrôleurs de l'application. Il sera généré s'il n'existe pas et mis à jour s'il existe.

Packages Nuget utiles (cf http://nuget.org)

- Microsoft.VisualStudio.Web.CodeGeneration.Design
- Swashbuckle.AspNetCore

Connection strings

1. Les chaines de connexions à la base de données sont dans appsettings.json sous cette forme

```
"ConnectionStrings": {
  "TpContext": "Server=sql1,1433;Database=TP_MIAGE;User Id=sa;Password=M1@g3.NET"
}
```

2. En général, dans appsetings.Development.json, on trouve les chaines pour le développement depuis son poste de développement et surcharge celle dans appsettings.json
3. Dans notre cas, dans apssettings.json on aura l'adresse interne du conteneur Docker
4. Chaines de connexion :
   a. En dev : `Server=localhost,61433;Database=TP_MIAGE;User Id=sa;Password=M1@g3.NET`
   b. Pour docker : `Server=sql1,1433;Database=TP_MIAGE;User Id=sa;Password=M1@g3.NET`

Divers

1. Par convention les classes POCO du modèle objet se situe dans un dossier Models

Docker

Lignes de commande

1. Build son application dans un conteneur

```
docker build --rm -f "Dockerfile" -t <NomDeLImage>:<Tag> .
```

   - A Faire dans le dossier de l'application
   - --rm : Enlève les conteneurs intermédiaires après la fin du build
   - -f : spécifie le fichier de définition
   - -t : target
   - <NomDeLImage> : nom de l'image Docker cible (ex : studentmanager)
   - <NomDuTag> : nom du tag lié à la version (ex : 1.0.0-stable / 2.3.2-beta / latest)

2. Exécuter un conteneur

```
docker run --rm -it -p <HostPort>:<ContainerPort>/tcp --name <Name> --link=<ConainerName>:<Alias> <NomDeLImage>:<Tag>
```

   - --rm : Nettoie l'ancien conteneur
   - -it : Interactif
   - -p : Port mapping.
     - <HostPort> : Port sur la machine hôte
     - <Container> : Port sur le conteneur cible
   - --name <Name> : Nommage du conteneur
   - --link=<ConainerName>:<Alias> : Etablissement d'un lien entre conteneurs où <ContainerName> est le nom ou l'id du conteneur auquel on veut accéder et <Alias> est l'alias par lequel sera accessible le conteneur.
   - <NomDeLImage> : nom de l'image Docker cible (ex : studentmanager)
   - <NomDuTag> : nom du tag lié à la version (ex : 1.0.0-stable / 2.3.2-beta / latest)

3. Accéder à l'intérieur du conteneur

```
docker exec -it <Name> /bin/bash
```

   - <Name> : Nom ou id du conteneur

Visual Studio Code

1. F1 ouvre la ligne de commande interne
2. Docker : Add Docker Files to Workspace permet d'ajouter les fichiers nécessaires à la conteneurisation dans votre projet.