Exercice : Implémentation d'une API en .Net Core

Sommaire

I) Présentation :

L'objectif de cet exercice va être d'implémenter une solution permettant de générer une API Rest Full en .Net Core, le langage utilisé étant C#.

Sachant que vous n'avez probablement jamais implémenté une API de votre vie, sachez qu'une API Rest est dite « RestFull » lorsque toutes les opérations du CRUD (Create, Read, Update, Delete) peuvent être exécuté via l'API. Ceci vous sera de nouveau expliqué lors de la présentation de l'exercice.

Pour appeler les méthodes de l'Url, il suffira ainsi d'écrire une url sur le navigateur.

Par exemple dans le cadre de notre exercice, l'url suivante « http://localhost:55124/api/question/getAll » permettra de récupérer la liste des questions qui sera présente dans un document Json.

C'est ce que nous allons vous demander d'implémenter en vous guidant progressivement au cours de cet exercice.

Je tiens à vous rappeler que l'objectif final est d'allumer toutes les diodes en vert qui correspondent aux tests effectués pour l'exercice.

Alors commençons dès maintenant à créer notre API !

II) L'Exercice : Une question de questions

Pour commencer, sachez qu'il va falloir créer une API permettant de traiter une liste de questions.

Pour gérer les données, dans un premier temps, vous aurez à votre disposition un document Json, « Questions.json ». Le document sera dans sa version originale via un 2ème document « Questions_initial.json », ce qui va permettre de récupérer la version initiale du document Json après avoir commencer à tester votre API. Elle sera aussi utile pour les tests, il est préférable de ne pas y toucher.

Ensuite, vous devez prendre connaissance des différentes requêtes possibles via une API Restful.

II.1) Objectifs :

En effet, les différents objectifs de cet exercice sont les suivants :

- Récupérer l'ensemble des questions contenus dans le document « Question.json » via une requête de type « GET ALL ». La liste des questions devra être accessible via l'url suivante : « http://localhost:55124/api/question/getAll ». Il s'agira du R pour « Read » du CRUD

- Lire une question précise de l'API via une requête de type « GET » avec le numéro d'identification de la question. Par exemple, si on veut récupérer l'intitulé de la question 3, elle devra être accessible en utilisant l'url suivante : « http://localhost:55124/api/question/get/3 ». Il s'agira là aussi du R du CRUD pour « Read » ou encore lecture.

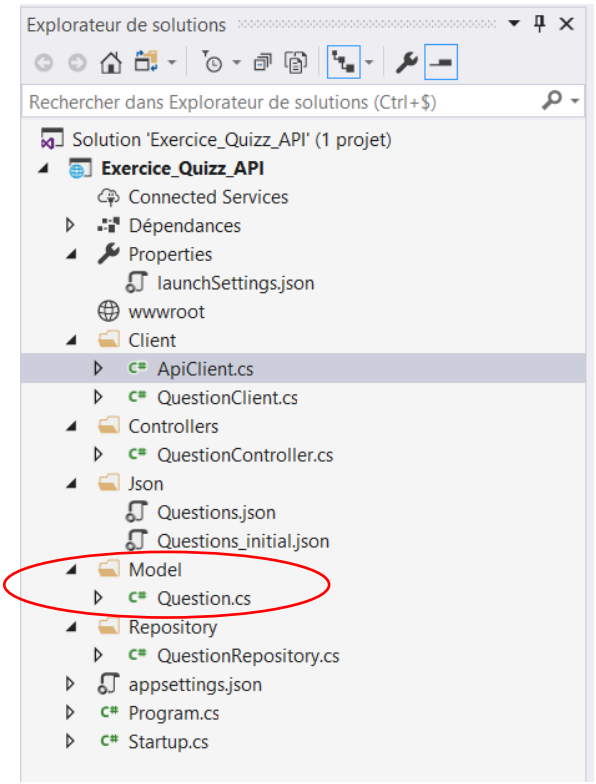
- Ajouter une question dans le document « Question.json » via une requête de type POST. Par exemple si on veut ajouter une question, il faudra fournir l'intitulé de la question avec la réponse, au format Json. Pour cela, il faudra utiliser l'url suivante : « http://localhost:55124/api/question/addQuestion ». Il s'agira du C du CRUD pour « Create ».

- Modifier une question dans le document « Question.json » via les requêtes de type PUT avec le numéro d'identification de la question. Par exemple, si on veut modifier la question 3, elle devra être accessible en utilisant l'url suivante : « http://localhost:55124/api/question/updateQuestion/3 ». Il faudra, comme pour l'ajout d'une question, préciser l'intitulé de la question mais aussi la réponse de la question que l'on voudra modifier. Il s'agira du U du CRUD pour « Update » ou encore mise à jour.

- Supprimer une question dans le document « Question.json » via les requêtes de type DELETE avec là aussi le numéro d'identification de la question. Il faudra bien vérifier au préalable que la question n'a pas déjà été supprimé ou n'existe pas. Par exemple, si on veut supprimer la question 3, il faudra utiliser l'url suivante : « http://localhost:55124/api/question/deleteQuestion/3 ». Il s'agira du D du CRUD pour « Delete » ou encore Suppression.

II.2) Précisions avant de commencer :

Lorsque vous allez récupérer le document, la structure de la solution devrait ressembler à ceci :



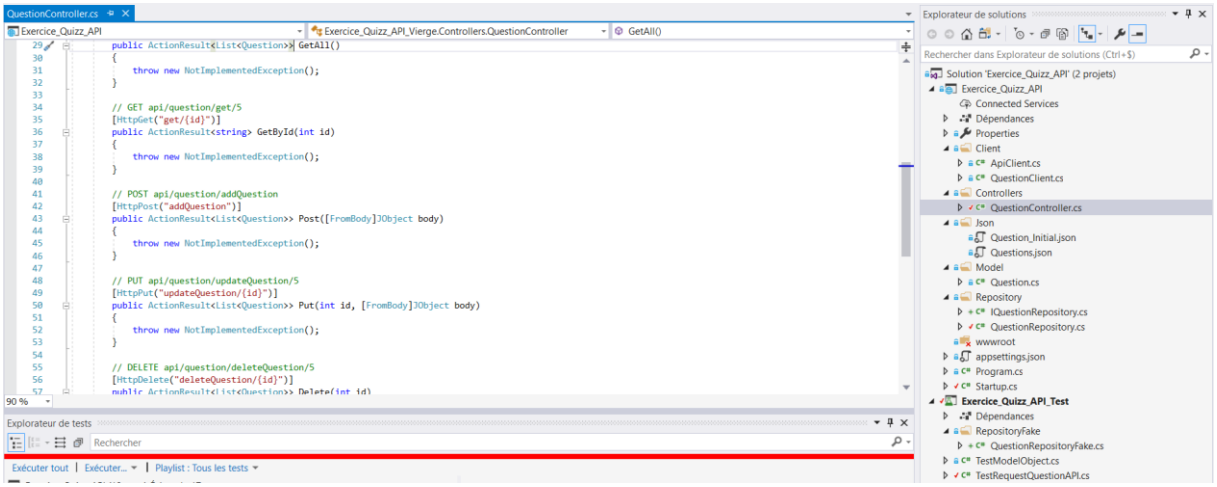
Et quand la classe Question.cs est chargée, elle ressemble à ceci :

```
public class Question
{
    // Créer le model objet Question et son constructeur
    public int QuestionId { get; set; }
    public string QuestionIntitule { get; set; }
    public string Answer { get; set; }

    public Question()
    {

    }

}
```

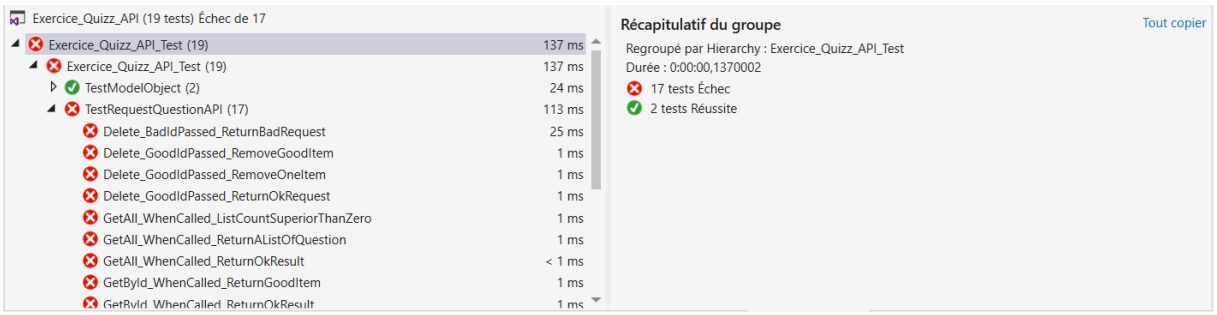
Sachez que la classe « Question.cs » ne doit pas être modifiée, si vous la modifier, que ce soit le nom des paramètres ou encore les types de ces paramètres, des tests permettant de vérifier la classe « Question » passeront aux rouges, ils sont par défaut en vert au 1er lancement des tests. N'hésitez pas à vous appuyer sur ce modèle pour réaliser cet exercice.

II.3) Lancer les tests de l'exercice :

Lorsque vous commencerez cet exercice et que vous lancerez la solution via Visual Studio, la solution devrait ressembler à ceci :



Et si vous lancez les tests, en cliquant sur « Test » puis « Éxécuter » puis « Tous les tests », vous devriez obtenir les résultats suivants dans l'explorateur de Tests :

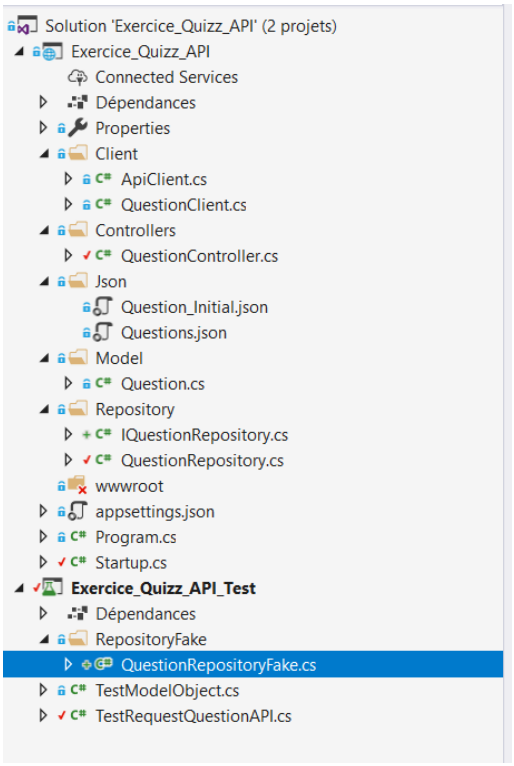


On voit ici que les tests concernant le « Delete » indique bien que la méthode n'a pas été implémentée, en effet c'est à vous de le faire.

De plus, j'ai implémenté la classe « QuestionRepositoryFake », une classe qui permettra de créer un « faux » jeu de donnée, un peu comme ce que pourrait être les fameuses « fake news », des informations qui sont fausses mais qui paraissent vrai.

Vous pouvez vous inspirez de ce « Fake » pour créer le vrai Repository, celui qui servira à l'utilisation de l'API.

Je vous rappelle l'objectif principal est d'implémenter une API, ainsi l'objectif des tests sont de tester la logique, le code que vous avez écrit dans le Controller principal de cette API.




```
namespace Exercice_Quizz_API_Test.RepositoryFake
{
    public class QuestionRepositoryFake : IQuestionRepository
    {
        private readonly List<Question> _questions;

        public QuestionRepositoryFake()
        {
            _questions = new List<Question>()
                {
                    new Question() { QuestionId = 1, QuestionIntitule = "En quelle annee a eu lieu la bataille de Waterloo ?", Answer = "1815" },
                    new Question() { QuestionId = 2, QuestionIntitule = "De quoi serait mort Napoleon ?", Answer = "D'un cancer de l'estomac" },
                    new Question() { QuestionId = 3, QuestionIntitule = "Quelle roi a decide la construction du chateau de Chambord ?", Answer = "Francois 1er"
                    new Question() { QuestionId = 4, QuestionIntitule = "Qui fut elu president de la IIIeme Republique en 1873 ?", Answer = "Mac Mahon" },
                    new Question() { QuestionId = 5, QuestionIntitule = "Quelle est la mere du petit Cesarion ?", Answer = "Cleopatre" },
                    new Question() { QuestionId = 6, QuestionIntitule = "Quelle reine de France appelle-t-on la Grosse Banquiere ?", Answer = "Marie de Medicis'
                };
        }

        public List<Question> AddOrUpdateQuestion(Question question)
        {
            throw new NotImplementedException();
        }
```

Si vous vous sentez prêt, vous pouvez commencer l'exercice dés maintenant, vous possédez suffisament d'informations pour le faire.

Enfin, sachez que j'ai placé un dossier s'appelant « Client » et où se situe deux classes, la classe « ApiClient » et la classe « QuestionClient ». Le but de l'exercice n'étant pas de vous évaluer sur cette partie, sachez que l'implémentation de ces deux classes n'est pas obligatoire, voyez cela comme un bonus à l'exercice que vous pouvez réaliser.

Vous pouvez sauter le chapitre II.4 et allez directement au chapitre II.5 qui vous explique comment lancer les tests de la correction.

Néanmoins si ce n'est pas le cas, pour vous aider dans cette tâche, voici plus de précisions sur les objectifs à atteindre.

II.4) Comment débuter ?

1. Ouvrez le dossier « ExerciceAPIQuizzTest » dans Visual Studio.

2. Une fois ouvert, dans l'explorateur de Solution, vous verrez une solution « ProjetTest ».

3. Déroulez le contenu de la solution et ouvrez les fichiers suivants : « QuestionController.cs », « Question.cs », « QuestionRepository.cs », « QuestionClient »

4. Il faudra ensuite modifier la classe permettant d'accéder aux données, la classe « QuestionRepository.cs »

Pour ce faire vous devez donc compléter les méthodes des autres fichiers.
Pour commencer il va falloir modifier le fichier « QuestionRepository.cs ».

Dans le fichier vous voyez diverses méthodes déjà présentes mais vide :

```
public static class QuestionRepository
{
    0 références | 0 exceptions
    public static List<Question> GetAllQuestions(string path)...
    0 références | 0 exceptions
    public static Question GetQuestion(string path, int questionId)...
    1 référence | 0 exceptions
    public static List<Question> AddOrUpdateQuestion(string path, Question question)...
    0 références | 0 exceptions
    public static List<Question> DeleteQuestion(string path, int questionId)...
}
```

Ces méthodes ont pour rôle d'effectuer la requête du même nom lors de leur appel.

Vous devez donc implémenter les différents types de requête dans ce fichier dans la méthode du nom correspondant afin de finir la mise en place de l'API Restful.

5. Une fois toutes les méthodes complétées, vous devez alors mettre en place l'interaction des méthodes du repository avec celles de votre controller, afin de compléter l'implémentation de l'API Restful

Rendez vous donc dans le fichier « QuestionController.cs » et complétez les méthodes suivantes :

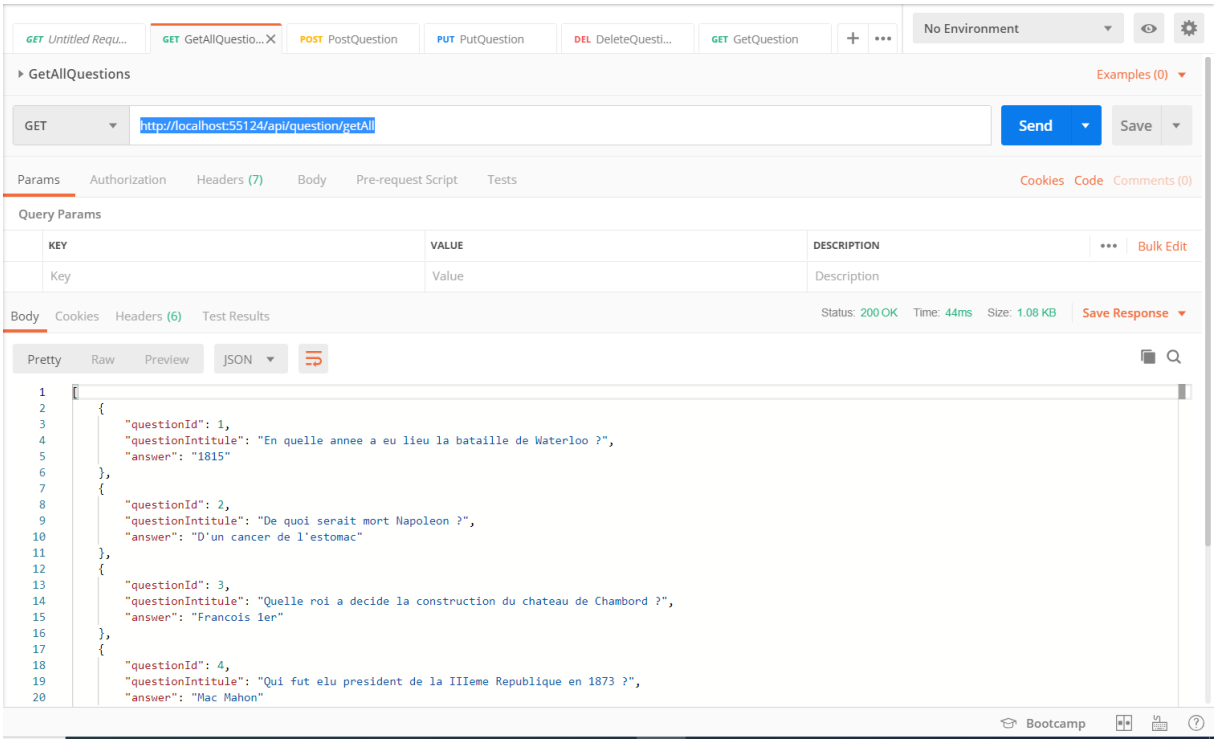
```
// GET api/question/GetAll
[HttpGet("getAll")]
[Produces("application/json")]
public ActionResult<List<Question>> GetAll()...

// GET api/question/get/5
[HttpGet("get/{id}")]
public ActionResult<string> GetById(int id)...

// POST api/question/addQuestion
[HttpPost("addQuestion")]
public ActionResult<List<Question>> Post([FromBody]JObject body)...

// PUT api/question/updateQuestion/5
[HttpPut("updateQuestion/{id}")]
public ActionResult<List<Question>> Put(int id, [FromBody]JObject body)...

// DELETE api/question/deleteQuestion/5
[HttpDelete("deleteQuestion/{id}")]
public ActionResult<List<Question>> Delete(int id)...
```

Pour information, une région s'intitulant « Helpers » existe dans le fichier « QuestionController.cs ». Les méthodes, à compléter sont ici à titre indicatif pour vous donner une idée du travail de factorisation à effectuer.

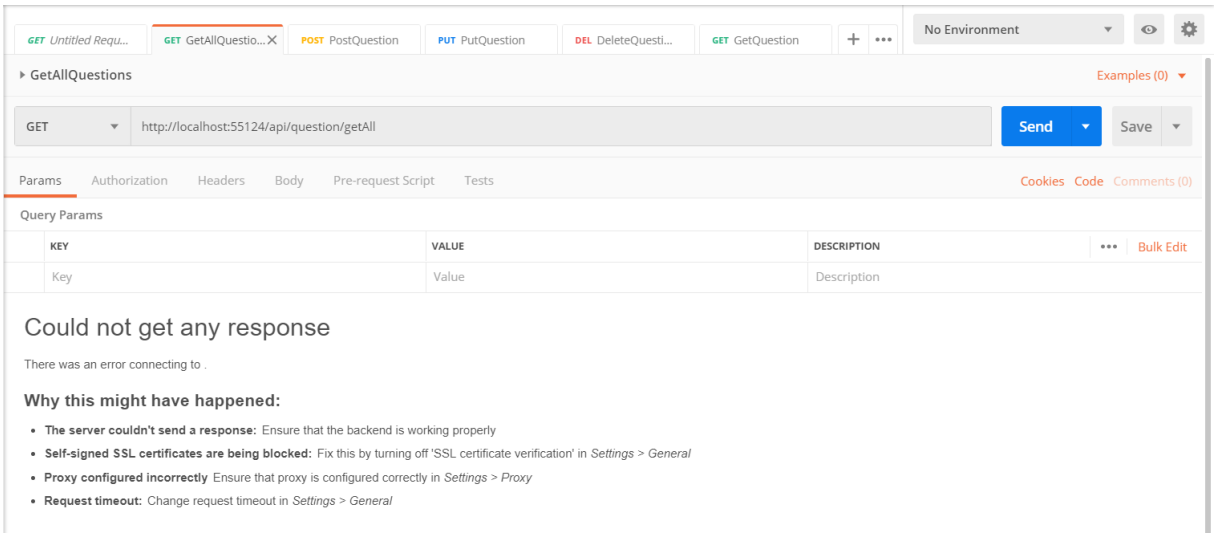
```
#region Helpers

private static Question MappingQuestion(int questionId, JObject body)...

private bool CheckIfQuestionExist(Question questionEnter)...

private ActionResult ExecuteRequestPostOrPut(JObject body, int idParam)...

#endregion
```

La factorisation consiste à appliquer un principe fondamentale de la programmation, « DRY » pour « Don't Repeat YourSelf », pas forcément facile à maîtriser et qui consiste à ne pas répéter son code ou du moins à le répéter le moins possible.

Si vous voulez un outil complémentaire pour tester votre API au fur et à mesure de l'exercice, je vous conseille PostMan, un utilitaire qui permet de tester l'ensemble des requêtes http qu'on vous demande d'implémenter, voici ci-dessous le résultat obtenu lors du lancement de la requête « http://localhost:55124/api/question/getAll »:



N'oubliez pas de lancer le projet permettant de lancer l'API, sinon PostMan vous renverra une réponse de ce type :



Enfin dernier conseil, bien lire le nom des méthodes, ceci permet de savoir quelle est l'objectif de la méthode, ce qu'elle est censé faire.

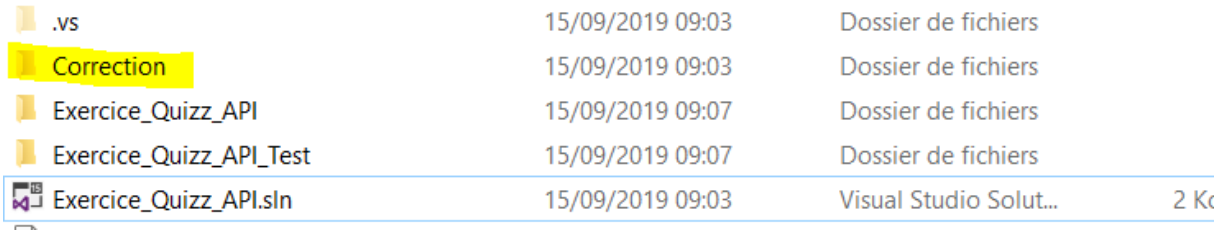
Nous vous avons fourni assez d'information pour commencer l'exercice, n'oubliez pas que vous pouvez me poser toutes les questions concernant cette exercice et qui vous passe par la tête, n'hésitez pas !

II.5) Lancer les tests de la correction :

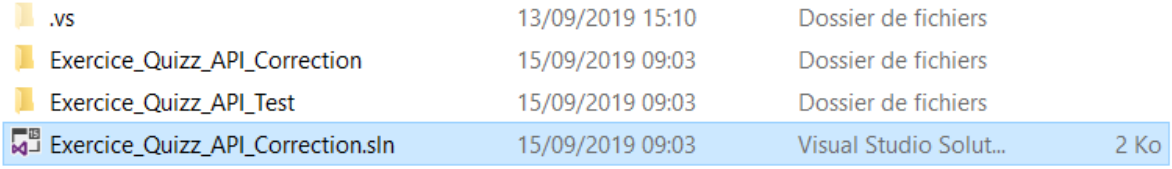
Lorsque vous aurez fini cet exercice, où que le temps imparti sera écoulé, je vous distribuerai un dossier contenant la Correction.

Voici la procédure à suivre pour lancer les tests de la correction :

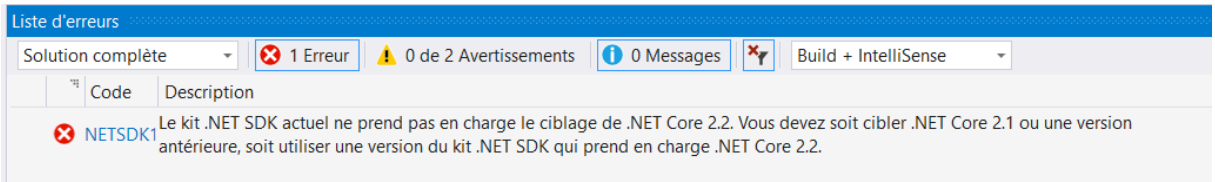
1. Ouvrez le sous dossier « Correction » du dossier que je vous ai transmis :



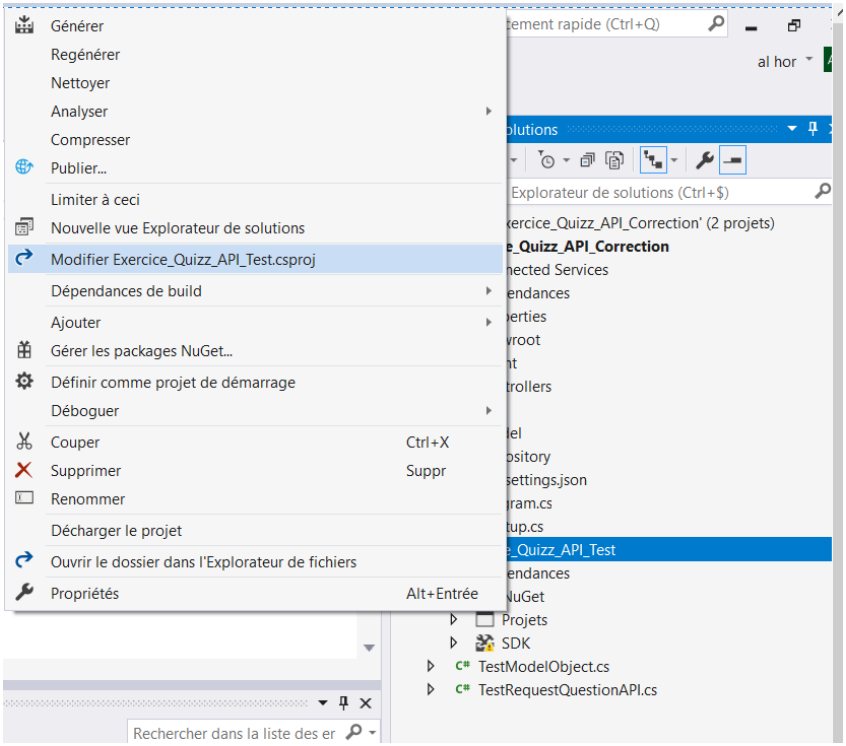
2. Lancer la solution « Exercice_Quizz_API_Correction » en double cliquant sur le fichier du même nom avec l'extension « .sln ». Il vous sera peut-être demandé si vous voulez faire confiance à un projet venant de l'extérieur, dites o.k.



3. Compiler la solution soit en allant sur la solution et en faisant clique droit puis « Régénérer la solution » ou encore en éxécutant le combo Ctrl + Maj + B.Lorsque vous allez lancer la Régénération de la solution vous risquez d'obtenir l'erreur suivante :



Il suffit simplement d'allez sur le projet, de faire clique droit puis de sélectionner « Modifier Exercice_Quizz_API_Test.csproj ».



Ensuite modifier la ligne « netcoreapp2.2 » en « netcoreapp2.1 ».

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.2</TargetFramework>

    <IsPackable>false</IsPackable>
  </PropertyGroup>
```

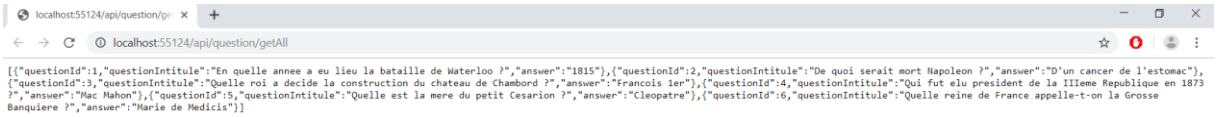
Avant la modification

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.1</TargetFramework>

  <IsPackable>false</IsPackable>
</PropertyGroup>
```

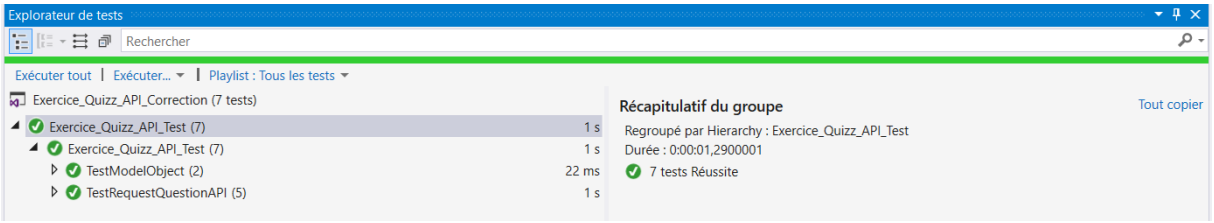
Après la modification

Si vous relancer la génération du projet, l'erreur devrait disparaître.

4. Lancer ensuite le projet « Quizz_API_Correction » en appuyant sur F5, une fenêtre de navigateur devrait se lancer avec le résultat suivant :



5. Vous pouvez maintenant lancer les tests de la correction, vous devriez obtenir le résultat suivant :



Pour conclure !:

Je le répète, si vous avez des questions, n'hésitez pas à me voir et enfin sachez que je viendrai régulièrement voir où vous en êtes, posez toutes les questions qui vous passe par la tête, je suis là pour cela !