

Choix de l'architecture du projet

Première approche des deux solutions proposées

Objectifs

- Vous familiariser avec les deux architectures proposées
- Vous faire découvrir ce qu'est AngularJS
- Vous faire découvrir ce qu'est ASP.NET MVC

Prérequis

- Vous devez avoir déjà travaillé avec des solutions Visual Studio, avoir défini plusieurs projets dans une solution et avoir exploité les références entre ces différents projets (et les using correspondant aux différents namespaces à disposition).
- Vous devez avoir déjà travaillé avec ASP.NET Web API et Entity Framework, particulièrement en mode Code First.
- Vous devez avoir déjà travaillé avec Git et disposer d'un repository GitHub.

Tous ces prérequis ont été abordés lors des séances précédentes.

Introduction

Cet énoncé a pour but de vous faire découvrir les deux approches qui vous sont proposées pour l'implémentation de votre projet.

Sur base d'un même domaine d'application, vous allez réaliser une première implémentation en utilisant une approche SPA, en utilisant AngularJS et ASP.NET Web API. Vous réaliserez ensuite une deuxième version de la solution, en utilisant ASP.NET MVC.

Démarrage du projet

Assurez-vous de disposer d'un clone de votre repository GitHub. Vous y conserverez votre travail.

Vous trouverez sur le repository GitHub <https://github.com/samiroquai/Henalux> une solution Visual Studio que vous utiliserez comme point de départ de ces manipulations.

Cette solution contient un modèle que vous allez utiliser dans les deux implémentations que vous créerez. Il exploite Entity Framework Code First pour générer une base de données sur base de la classe SchoolContext et de ses propriétés DbSet (et des classes impliquées).

Récupérez cette solution et collez-la dans votre repository GitHub local. N'oubliez pas en fin de séance d'ajouter vos modifications au repository, de réaliser votre commit, puis de pusher vos modifications. **Attention, vérifiez que votre username+password sont bien supprimés de la connection string avant de réaliser le push.**

Modifiez la connection string du projet pour pointer vers vm-sql. Pour le nom de la base de données, utilisez SchoolXXXXX où XXXXX est votre etuXXXX.

Implémentation de la première solution: ASP.NET MVC

Ajoutez à votre solution un nouveau projet de type Web Application.

Configurez ce nouveau projet pour être de type ASP.NET MVC (sans Web API), voir screenshots ci-dessous. Ne l'hébergez pas dans le cloud.

Add New Project

Recent

Installed

Recent

Visual C#

Windows

Universal

Windows 8

Classic Desktop

Web

Android

Cloud

Extensibility

iOS

Reporting

Silverlight

Test

WCF

Workflow

Other Languages

Online

.NET Framework 4.5.2

Sort by: Default

Search Installed Templates (Ctrl+E)

ASP.NET Web Application

Visual C#

Class Library (Package)

Visual C#

Console Application (Package)

Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights


☐ Add Application Insights to Project

Help you understand and optimize your application.

[Learn more](#)

[Privacy statement](#)

sa²muel.scholtes@outlook.com (...)

 Reenter your credentials

[Click here to go online and find templates.](#)

Name: WebApplication1

Location: C:\Sources\Henalux\ChoixArchitecture

Browse...

OK

Cancel

New ASP.NET Project - AspNetMVCSolution

Select a template:

ASP.NET 4.5.2 Templates

Empty

Web Forms

MVC

Web API

Single Page Application

Azure API App (Preview)

Azure Mobile Service

ASP.NET 5 Preview Templates

Empty

Web API

Web Application

Add folders and core references for:

☐ Web Forms ☒ MVC ☐ Web API

☐ Add unit tests

Test project name: AspNetMVCSolution.Tests

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

[Learn more](#)

Change Authentication

Authentication: Individual User Accounts

Microsoft Azure

☐ Host in the cloud

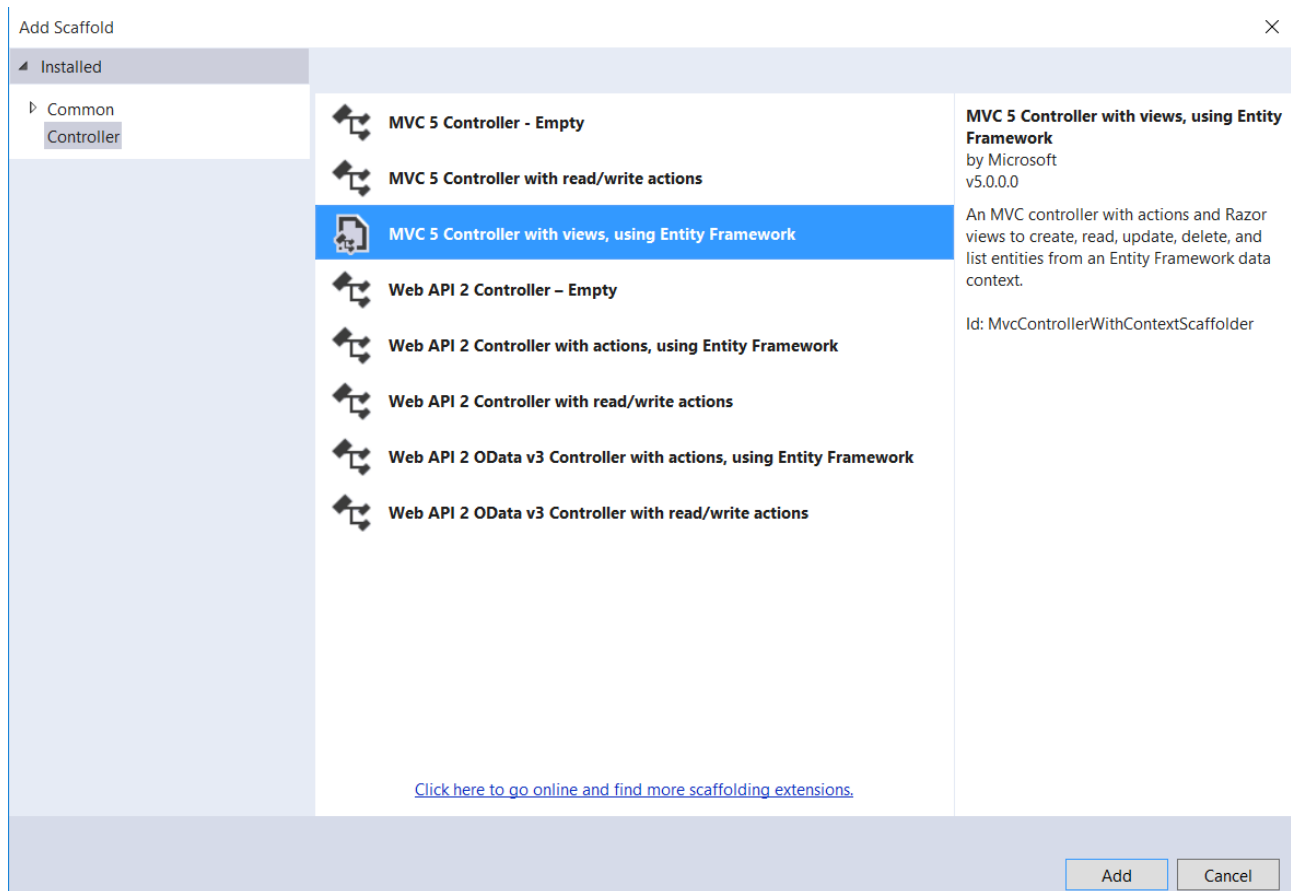
Web App

OK

Cancel

Ajoutez à ce nouveau projet une référence vers le modèle.
Compilez ensuite votre solution.

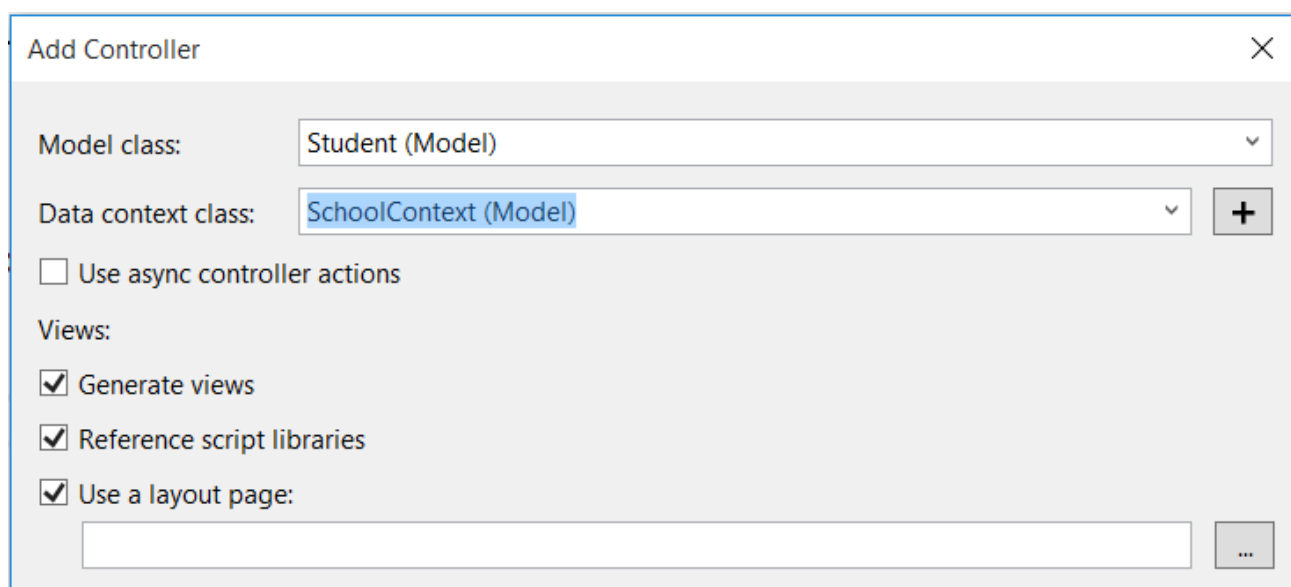
Nous allons ensuite créer les vues de gestion des étudiants du modèle. A cette fin, nous allons utiliser une fonctionnalité d'ASP.NET MVC qui génère des controllers et des vues automatiquement, sur base de votre modèle. Ajoutez un nouveau controller à votre projet. Ce controller doit être de type MVC 5 Controller with views, Using Entity Framework (voir screenshot).



Vous arriverez ensuite dans un assistant qui va vous demander plusieurs informations. Vous devez renseigner:

- la classe du modèle pour laquelle du code doit être généré
- le contexte Entity Framework qui permet d'assurer le mapping avec la base de données
- les préférences de génération de code.

Basez-vous sur l'illustration ci-dessous pour configurer l'assistant.



Une fois que l'assistant a terminé son travail, examinez les controllers et les vues de votre projet (voir folders respectifs). Vous devriez voir le code généré pour vous par ASP.NET MVC.

Lancez votre application Web (assurez-vous qu'il s'agit bien du projet de démarrage).

Rendez-vous alors sur la page listant les étudiants. ASP.NET MVC a généré pour vous des controllers et des vues, mais n'a pas mis à jour le menu, vous allez donc devoir utiliser l'URL pour accéder à la vue en question. A cette fin, utilisez l'URL <http://localhost:59396/Students>. Remarque: le numéro de port sera sans doute différent lorsque vous lancerez votre application. Adaptez cette url en conséquence.

Vous devriez alors voir apparaître la liste des étudiants enregistrés. Vous pouvez jouer avec les différentes pages générées pour vous.

Examinez la manière dont les URL's sont traitées. Le routing ASP.NET MVC est un élément important de cette technologie, qui repose sur des conventions de nommage de vos vues, controllers et actions.. Vous pouvez en apprendre plus à cette URL: <https://msdn.microsoft.com/en-us/library/cc668201.aspx>

Réalisez la même démarche pour créer l'administration des cours.

Vous avez terminé avant les autres? Ajoutez les liens nécessaires au menu afin de permettre un accès facile aux pages de gestion des cours et des étudiants. Vous n'en avez pas assez? Essayer dans la vue de détail d'un étudiant de lister les cours auxquels l'étudiant est inscrit.

Implémentation de la seconde solution : SPA avec ASP.NET Web API et AngularJS

Créez un nouveau projet de type Web Application dans la solution existante. Choisissez le template "Empty" et cochez la case Web API, tel qu'illustré ci-dessous.

New ASP.NET Project - SPASolution

Select a template:

ASP.NET 4.5.2 Templates

Empty Web Forms MVC Web API Single Page Application

Azure API App (Preview) Azure Mobile Service

ASP.NET 5 Preview Templates

Empty Web API Web Application

An empty project template for creating ASP.NET applications. This template does not have any content in it.

[Learn more](#)

Change Authentication

Authentication: **No Authentication**

Microsoft Azure

Host in the cloud

Web App

Add folders and core references for:

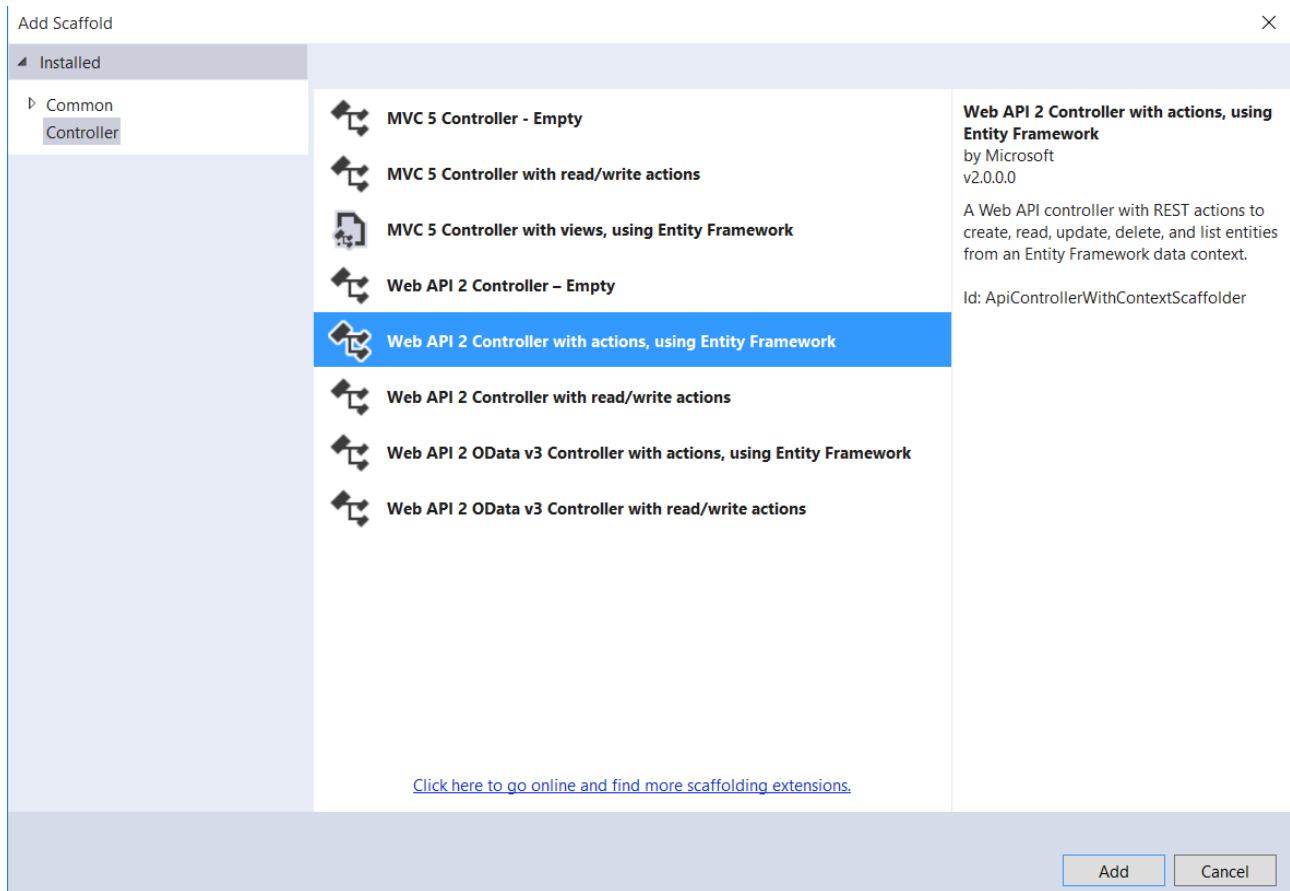
☐ Web Forms ☐ MVC ☒ Web API

☐ Add unit tests

Test project name: SPASolution.Tests

OK Cancel

Une fois le projet créé, ajoutez une référence vers votre modèle et ajoutez-y un nouveau controller de type Web API 2 Controller with actions, using Entity Framework.



Du code va à nouveau être généré pour vous, sur base des informations de l'assistant qui va s'afficher. Renseignez les mêmes informations que précédemment (Student, SchoolContext) et laissez le nom "StudentsController" par défaut.

Le résultat de la génération est un controller REST pour la ressource Student.

Définissez ce projet comme étant celui de démarrage, puis lancez le debugging. Testez votre nouveau controller à l'aide de Fiddler et d'une requête HTTP GET envoyée à l'URL <http://localhost:XXXXX/api/Students> (XXXXX à remplacer par votre numéro de port). Vous devriez obtenir une réponse Json contenant les étudiants.

En cas de problème, recompilez votre solution et ajoutez le controller à nouveau.

Ajoutez ensuite le package Nuget AngularJS.Core ainsi que le package AngularJS.Route à votre projet. Un dossier Scripts devrait apparaître. Il contient les fichiers javascript qui s'exécuteront côté client, dans le navigateur de l'utilisateur.

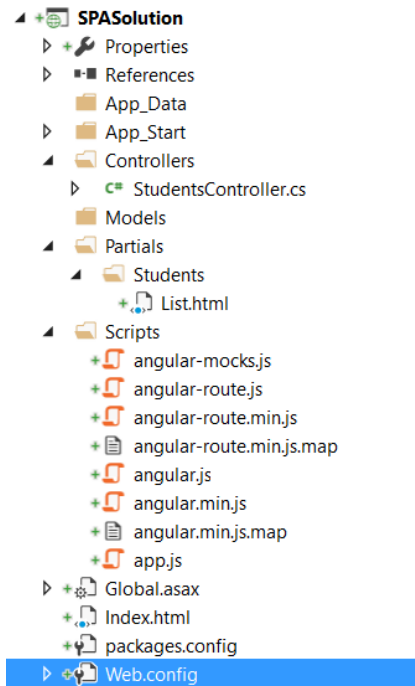
Ajoutez une page Index.html à votre projet. Dans cette page, insérez le code qui vous sera communiqué via le PasteBin disponible pendant la séance.

Dans cette page, vous faites charger les scripts d'AngularJS. Vous initialisez également votre application AngularJS en appelant un script nommé app.js. Ce fichier n'existe pas encore. Créez donc un nouveau fichier javascript nommé app.js sous le dossier "Scripts" de votre projet.

Ensuite, créez un nouveau dossier, sous la racine de votre projet, qui se nommera “Partials”. C’est la que les différentes vues seront définies. Dans ce dossier, créez un sous-dossier “Students” dans lequel vous créerez une page List.html. Cette page sera responsable de l’affichage de la liste des étudiants.

Vous pouvez récupérer le code de cette page sur le PasteBin disponible pendant la séance.

L’arborescence de votre projet devrait ressembler à l’illustration ci-dessous.



Lancez votre projet et naviguez vers la page index.html. La liste des étudiants devrait s’afficher. Le design est très très rudimentaire, le focus n’étant pas sur ce point pour cette démarche.

Lors de l’affichage de la page Index.html, les librairies AngularJS ont été chargées.

```
<html>
<head>
  <title></title>
  <meta charset="utf-8" />
  <script src="Scripts/angular.js"></script>
  <script src="Scripts/angular-route.js"></script>
```

Votre application également. Celle-ci définit que par défaut, c’est la vue “liste des étudiants” qui est affichée. Le navigateur a donc émis une requête HTTP pour récupérer la page /partials/Students/List.html. La définition de cette vue par défaut se fait dans votre application AngularJS (voir fichier app.js).

```
angular.module('SchoolApp', ['ngRoute'])
  .config(['$routeProvider',
    function ($routeProvider) {
      $routeProvider
        .when('/', {
          templateUrl: 'partials/Students/List.html',
          controller: 'StudentsCtrl'
        })
        .otherwise({ redirectTo: '/' });
    })
  .run(function () {
    // ...
  });
})
```

La vue en question est associée au contrôleur “StudentsCtrl” (voir configuration de l’application dans l’illustration ci-dessus). Dès lors, lorsque la vue est affichée, StudentsCtrl s’initialise en appelant votre Web API pour récupérer la liste des étudiants.

```
.controller('StudentsCtrl', function ($scope, $http) {  
    $http.get("/api/students").success(function (data, status, headers, config) {  
        $scope.students = data;  
    }).error(function (data, status, headers, config) {  
        console.log("something went wrong!!!");  
    });  
});
```

une fois la requête Http terminée avec succès, la liste des étudiants (accessible via “data”) est assignée comme valeur à la propriété students de l’objet \$scope. Il s’agit d’un objet fourni par AngularJS qui préserve l’état de vos vues et auquel vos vues se bindent (considérez \$scope comme votre ViewModel). Un peu comme dans vos applications UWP, la vue détecte les changements survenant dans le \$scope. Dès lors que \$scope.students reçoit des valeurs, la vue se met à jour.

```
:div>  
  <table>  
    <thead>  
      <tr>  
        <td><b>Nom</b></td>  
        <td><b>Prénom</b></td>  
        <td><b>Date de naissance</b></td>  
        <td><b>Pays</b></td>  
      </tr>  
    </thead>  
    <tbody>  
      <tr ng-repeat="student in students">  
        <td>{{student.FirstName}}</td>  
        <td>{{student.LastName}}</td>  
        <td>{{student.BirthDate}}</td>  
        <td>{{student.Country}}</td>  
      </tr>  
    </tbody>  
  </table>  
</div>
```

Pour chaque étudiant, une ligne est ajoutée à la table. Chaque ligne contient 4 cellules (prénom, nom, date de naissance, pays). Erratum: l’illustration ci-jointe contient une erreur (inversion nom/prénom).

Maintenant que vous êtes en possession d’une application AngularJS couplée à une Web API, créez une seconde vue, qui affiche la liste des cours. A cette fin vous devrez:

- ajouter un nouveau controller Web API qui retourne les cours
- ajouter une nouvelle vue html qui liste les cours
- ajouter un nouveau controller javascript à votre application AngularJS
- configurer votre application AngularJS pour associer la vue au nouveau controller javascript
- appeler votre controller Web API depuis votre controller AngularJS et garnir la propriété \$scope pour que la vue soit notifiée de la présence de cours à afficher
- ajouter un lien vers la nouvelle vue à la page index.html (en dehors de la div décorée avec l’attribut ng-view).