

La programmation en C#

Introduction

Bienvenue dans ce cours dédié à l'apprentissage de la programmation en C#.

Dans la première partie du cours, vous apprendrez à déclarer des variables, maîtriser les types de données, définir des objets avec des classes, et choisir la bonne collection pour gérer des situations complexes.

La seconde partie se concentre sur la gestion du déroulement d'un programme : démarrer avec la fonction Main, utiliser des conditions et des boucles, gérer les erreurs et communiquer avec l'extérieur.

Enfin, dans la troisième partie, vous découvrirez comment écrire un code clair et facile à maintenir. Vous comprendrez les paramètres et les valeurs de renvoi, écrirez des fonctions propres, les testerez et les déboguerez. Vous terminerez par une activité pratique pour créer votre première application.

En suivant ces trois parties, vous allez acquérir une base solide en programmation en C#, vous permettant de créer des programmes efficaces et bien structurés.

Préparez vos outils de travail

Prenez le temps de vous préparer en rassemblant vos outils : Visual Studio, Visual Studio Code et GitHub.

Visual Studio et Visual Studio Code

Visual Studio, ou Visual Studio Code pour les utilisateurs de Mac, est l'environnement de travail préféré des développeurs. Ces outils permettent de travailler avec plusieurs langages, dont C#. Ils sont essentiels pour pratiquer et réaliser les exercices du cours et sont incontournables pour les développeurs C# dans leur quotidien professionnel.

Installer Visual Studio pour Windows

Vous allez effectuer deux étapes pour installer Visual Studio sur Windows :

1. Installer [Visual Studio Community Edition](#) (gratuit).
2. Configurer un environnement de développement complet

Pour Windows

Téléchargeons l'édition gratuite de Visual Studio, Community, pour écrire votre première application .NET.

La première étape est d'installer **Visual Studio Installer**, utilisé pour maintenir à jour l'IDE ou pour toute modification, comme passer sur une édition professionnelle (selon votre besoin). Vous pouvez lancer Visual Studio Installer depuis votre menu Démarrer.

L'installateur va ensuite vous demander de choisir les **espaces de travail** à ajouter à votre installation. Il s'agit de collections de composants nécessaires pour un type d'application donné. En fonction de ce que vous souhaitez créer, vous choisirez l'espace de travail associé. Par exemple, un espace de développement d'applications web n'est pas le même qu'un espace dédié aux applications mobiles.

L'IDE Visual Studio permet de travailler avec les langages suivants : **C#, Visual Basic, JavaScript, TypeScript, XML, HTML, CSS**. Pour illustrer ce cours, nous avons choisi de vous présenter l'exemple d'une application .NET développée avec le langage de programmation C#. Si vous travaillez sur un projet de développement dans un autre langage de programmation, il vous suffit d'installer les espaces de travail correspondants.

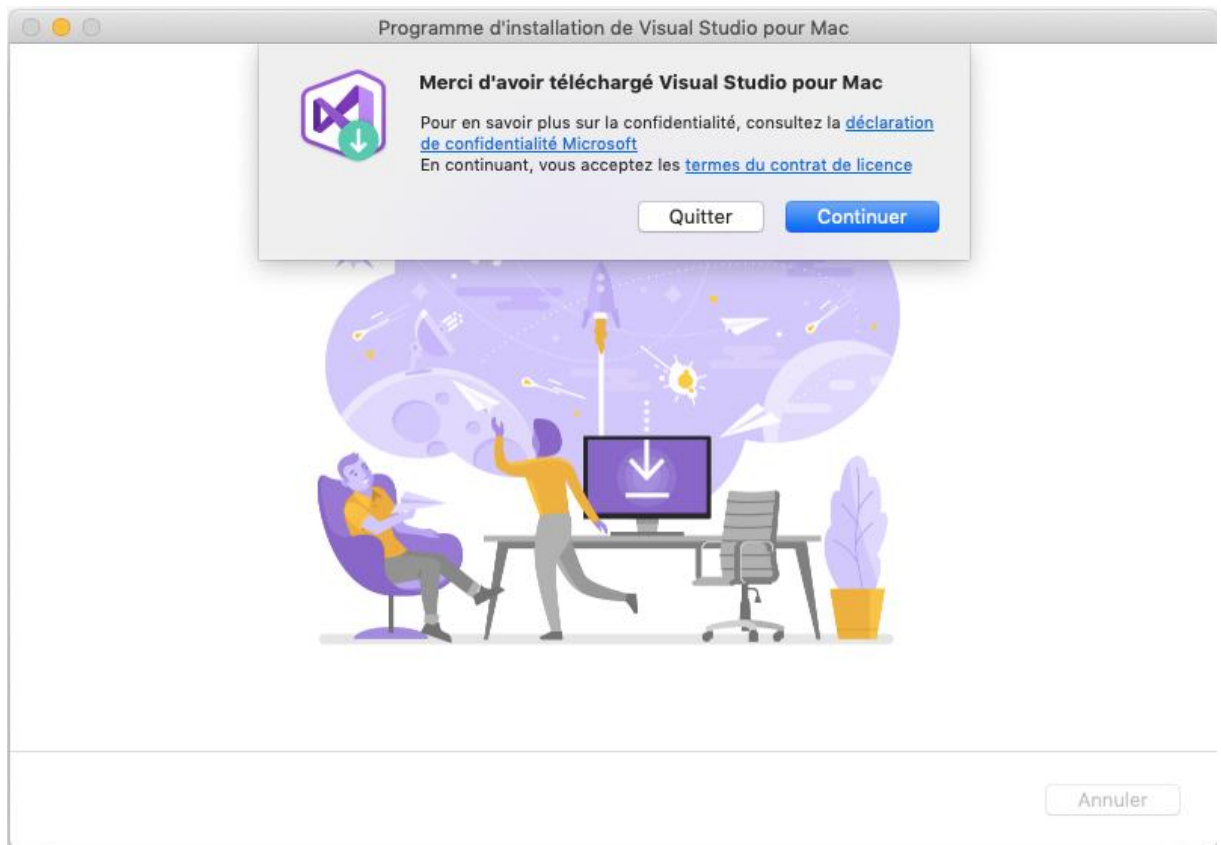
Chaque espace de travail n'a besoin d'être installé qu'une fois, et reste accessible pour autant de projets que vous le souhaitez.

Vous avez également la possibilité de sélectionner des **composants individuels**, en plus ou à la place des espaces de travail. Ces options sont recommandées pour des utilisateurs avancés qui ont des besoins précis. Vous y retrouverez toutes les collections d'outils, services et langages de programmation supportés par Visual Studio.

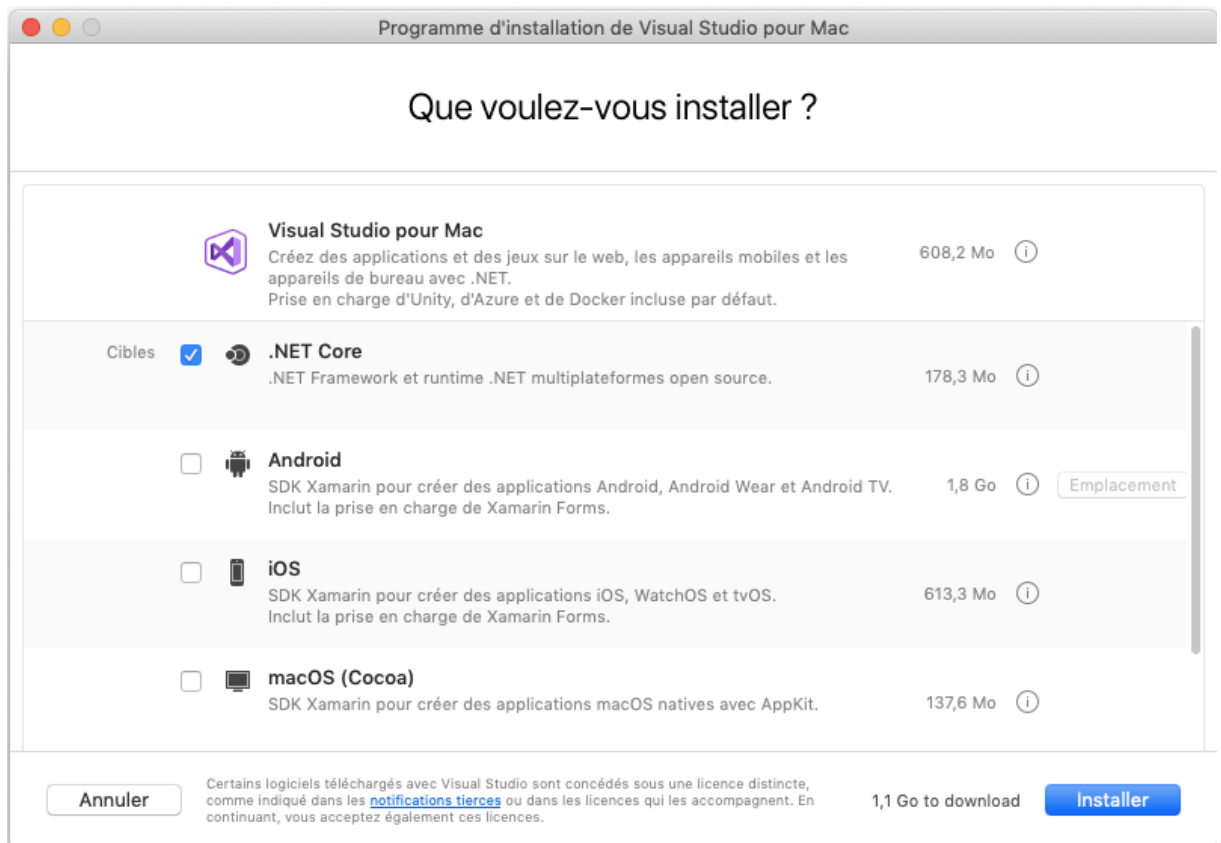
Installer Visual Studio Code pour Mac

Voici les étapes pour installer Visual Studio Code sur Mac :

1. Ouvrez votre navigateur web et allez sur le site officiel de Visual Studio Code à l'adresse suivante : <https://code.visualstudio.com/>.
2. Cliquez sur le bouton "Download" pour télécharger l'installateur approprié.
3. Le fichier que vous téléchargez est l'application déjà prête à l'emploi. Glissez-la dans votre dossier Applications pour la rendre accessible depuis le Launchpad.
4. **Pour macOS**
5. Les étapes d'installation sont les mêmes que pour Windows. Vous pouvez télécharger la dernière version de [Visual Studio Community](#) pour macOS. Ensuite, lancez le fichier téléchargé :



6. Programme d'installation de Visual Studio
7. Le programme d'installation vous propose de choisir les composants à installer.
Dans notre cas, il s'agit de .NET Core :



8.

Choisissez les composants à installer

9. Le tour est joué ! Visual Studio s'ouvre une fois l'installation terminée.

Maintenant, paramétrons GIT. Créez un compte en suivant le cours introductif sur Git.

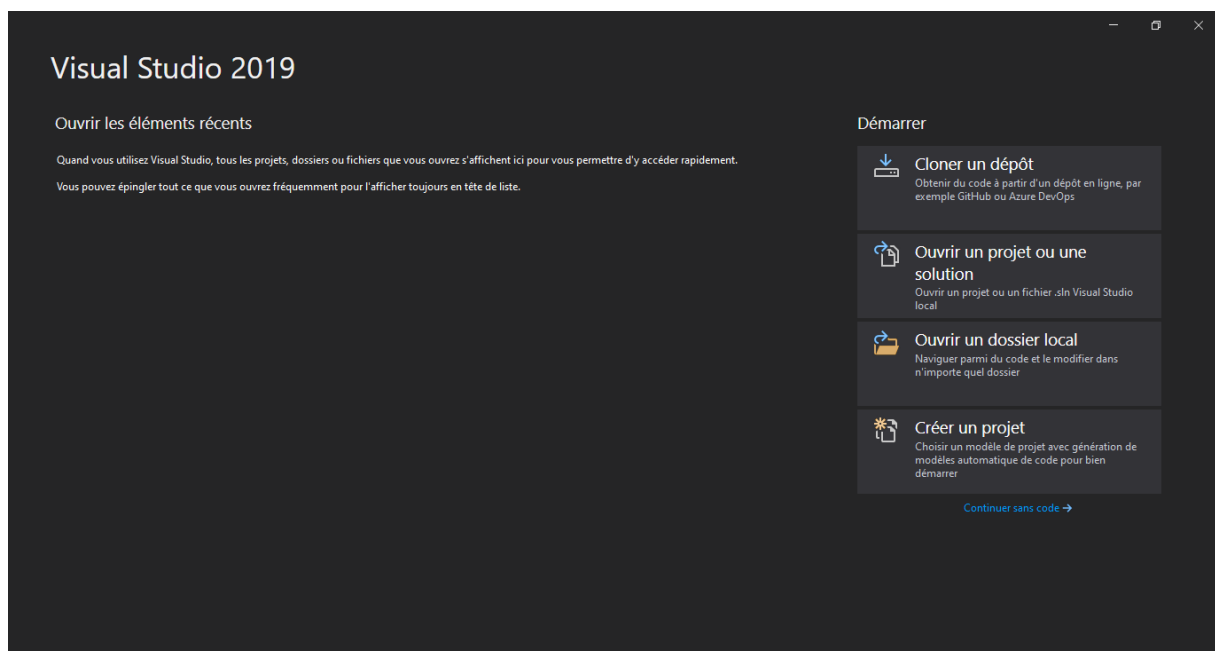
Découvrez l'interface graphique de Visual Studio

Après avoir installé Visual Studio, lancez-le depuis votre menu Démarrer ou un raccourci.

Vous avez désormais accès à son interface graphique, ou **GUI**, pour "Graphical User Interface", en anglais. La GUI est ce avec quoi vous interagissez, qu'il s'agisse d'icônes, de menus ou d'indicateurs divers, avec une souris, un stylet ou votre doigt. Vous utilisez des interfaces graphiques au quotidien, de votre téléphone à votre ordinateur.

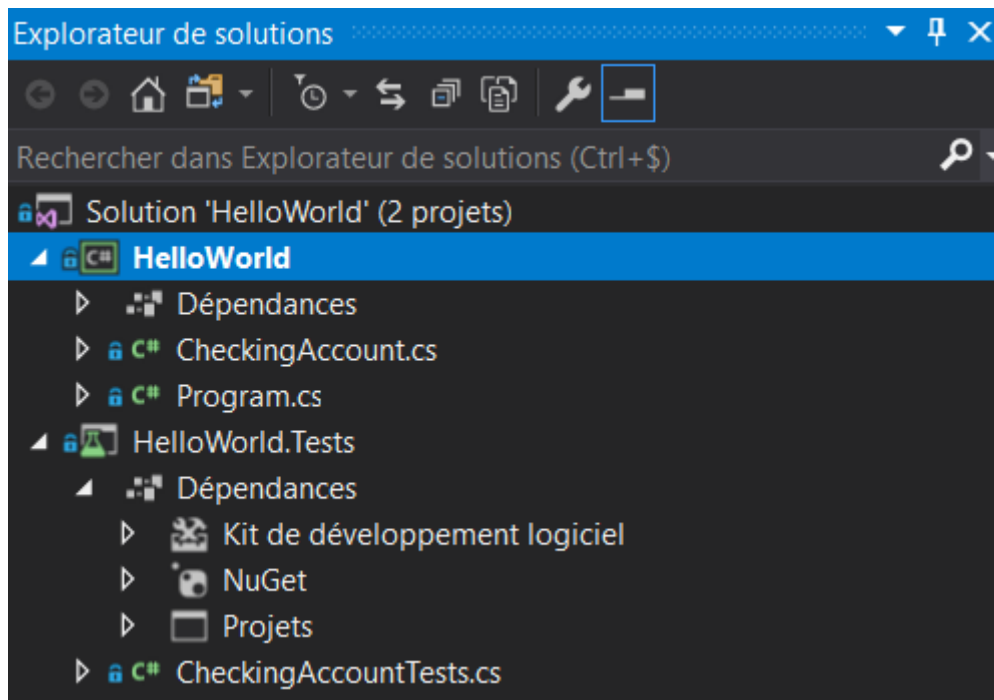
Lorsque vous lancez Visual Studio 2019, la page de démarrage est affichée avec les sections suivantes :

- **Cloner un dépôt** : récupérer le code d'un dépôt en ligne, tel que GitHub.
- **Ouvrir un projet ou une solution** : ouvrir un projet ou un fichier .sln sur votre ordinateur.
- **Ouvrir un dossier local** : ouvrir un répertoire sur votre machine.
- **Créer un projet** : créer un nouveau projet, vide ou depuis un modèle proposé.



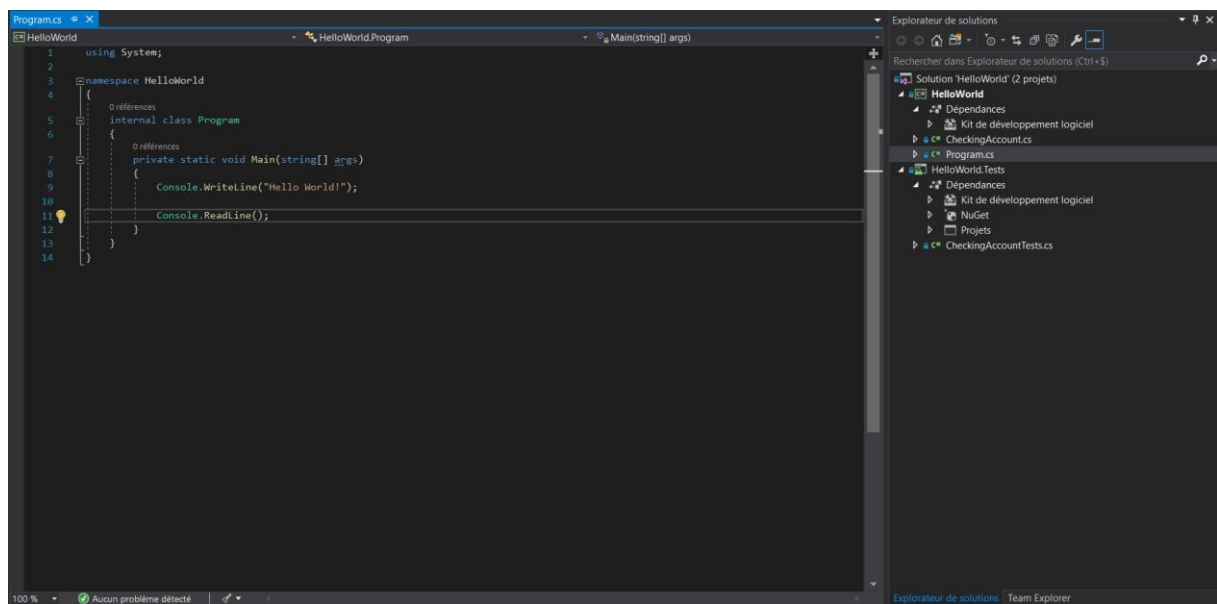
La page de démarrage de Visual Studio

Sur la partie droite de l'écran, vous verrez l'**explorateur de solutions**, qui affiche tous les fichiers, dossiers et bibliothèques de référence utilisés dans votre application.



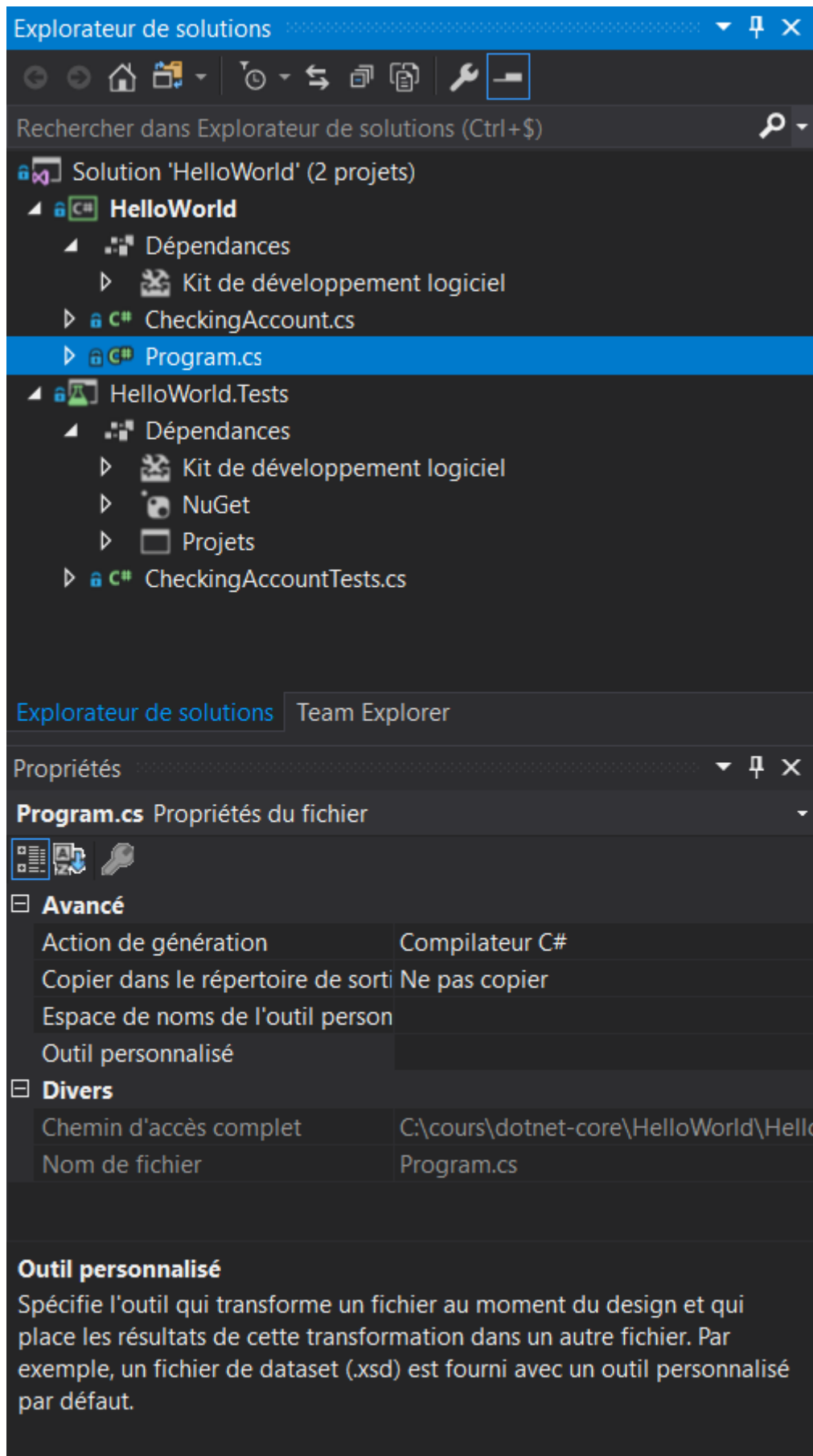
L'explorateur de solutions

Ouvrez un fichier dans l'explorateur. Vous verrez son contenu affiché dans la fenêtre dite **principale**, sur la gauche. C'est l'endroit où vous écrirez et modifierez votre code.



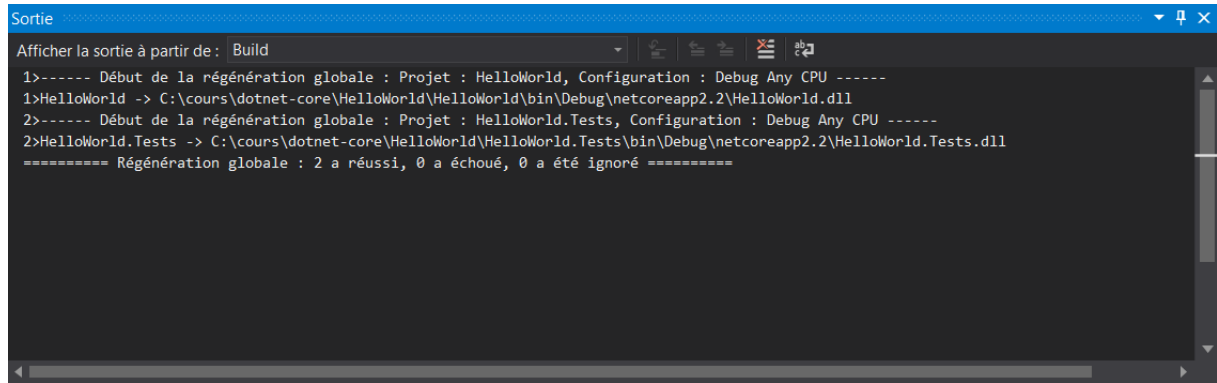
Un fichier sélectionné est affiché dans la fenêtre principale

En dessous de l'explorateur se trouve la fenêtre **des propriétés**. En sélectionnant un élément dans l'explorateur, vous pouvez obtenir plus de détails dans cette fenêtre, et les modifier si nécessaire.



Les propriétés d'un élément sélectionné sont affichées dans la fenêtre des propriétés

En dessous de la fenêtre principale se trouve la fenêtre de **sortie**, qui affiche les messages provenant de la compilation, ainsi que des informations sur les causes possibles.



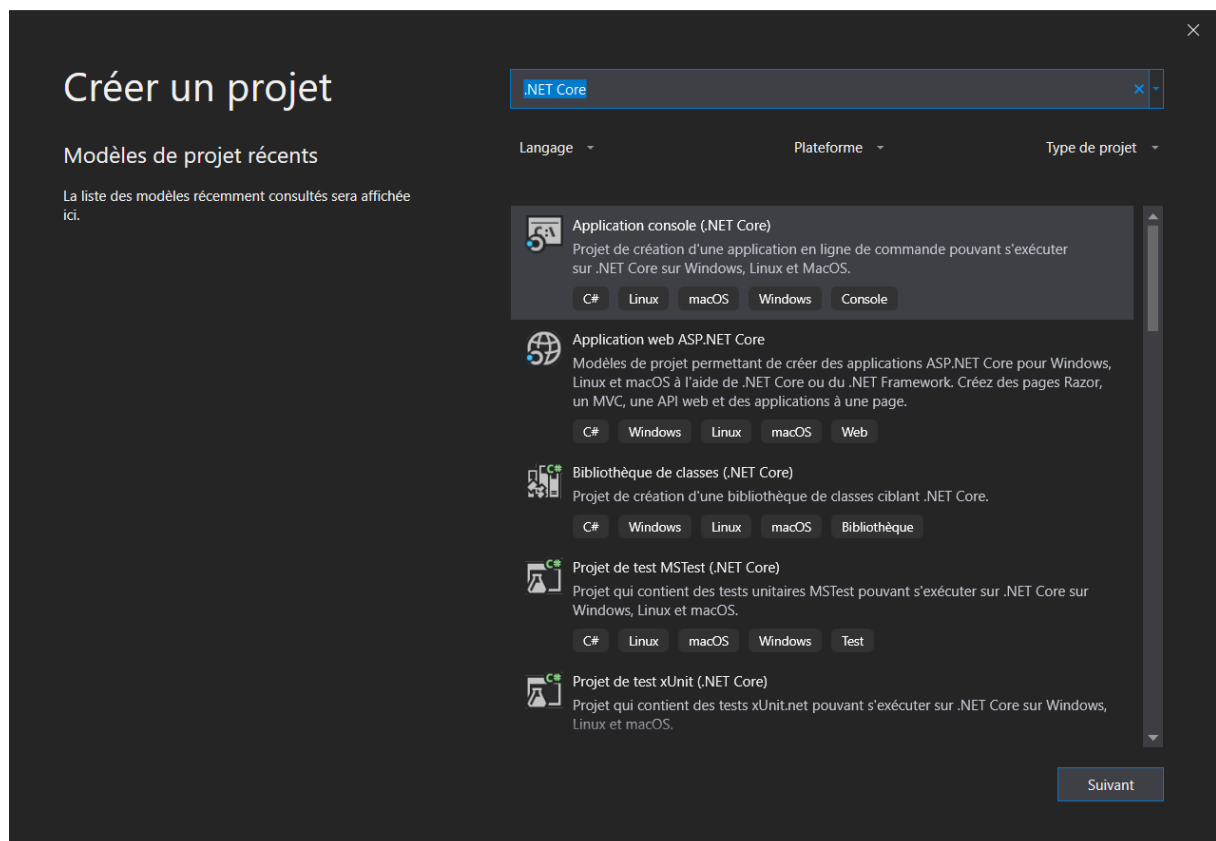
```
Sortie
Afficher la sortie à partir de : Build
1>----- Début de la régénération globale : Projet : HelloWorld, Configuration : Debug Any CPU -----
1>HelloWorld -> C:\cours\dotnet-core\HelloWorld\HelloWorld\bin\Debug\netcoreapp2.2\HelloWorld.dll
2>----- Début de la régénération globale : Projet : HelloWorld.Tests, Configuration : Debug Any CPU -----
2>HelloWorld.Tests -> C:\cours\dotnet-core\HelloWorld\HelloWorld.Tests\bin\Debug\netcoreapp2.2\HelloWorld.Tests.dll
===== Régénération globale : 2 a réussi, 0 a échoué, 0 a été ignoré =====
```

Les résultats d'un build sont affichés dans la fenêtre de sortie

Créez votre premier projet avec Visual Studio

Créons une application à l'aide d'un **modèle** de projet, ou “**template**”, inclus dans Visual Studio.

Les modèles de projet vous permettent de gagner du temps lorsque vous créez une application. Ils fournissent un squelette plus ou moins étoffé (en fonction de votre choix) où vous pouvez commencer à écrire immédiatement sans vous préoccuper des ressources nécessaires pour démarrer. La liste des modèles disponibles est affichée dans la fenêtre “Créer un projet”.



La fenêtre "Créer un projet" affiche les modèles de projet de Visual Studio

En sélectionnant un modèle de projet, vous laissez Visual Studio :

- créer la structure du répertoire ;
- installer tous les fichiers nécessaires ;
- installer les bibliothèques de référence requises.

Une fois le modèle choisi, vous disposez déjà d'une application complète qui peut être exécutée sans avoir à faire le moindre changement.

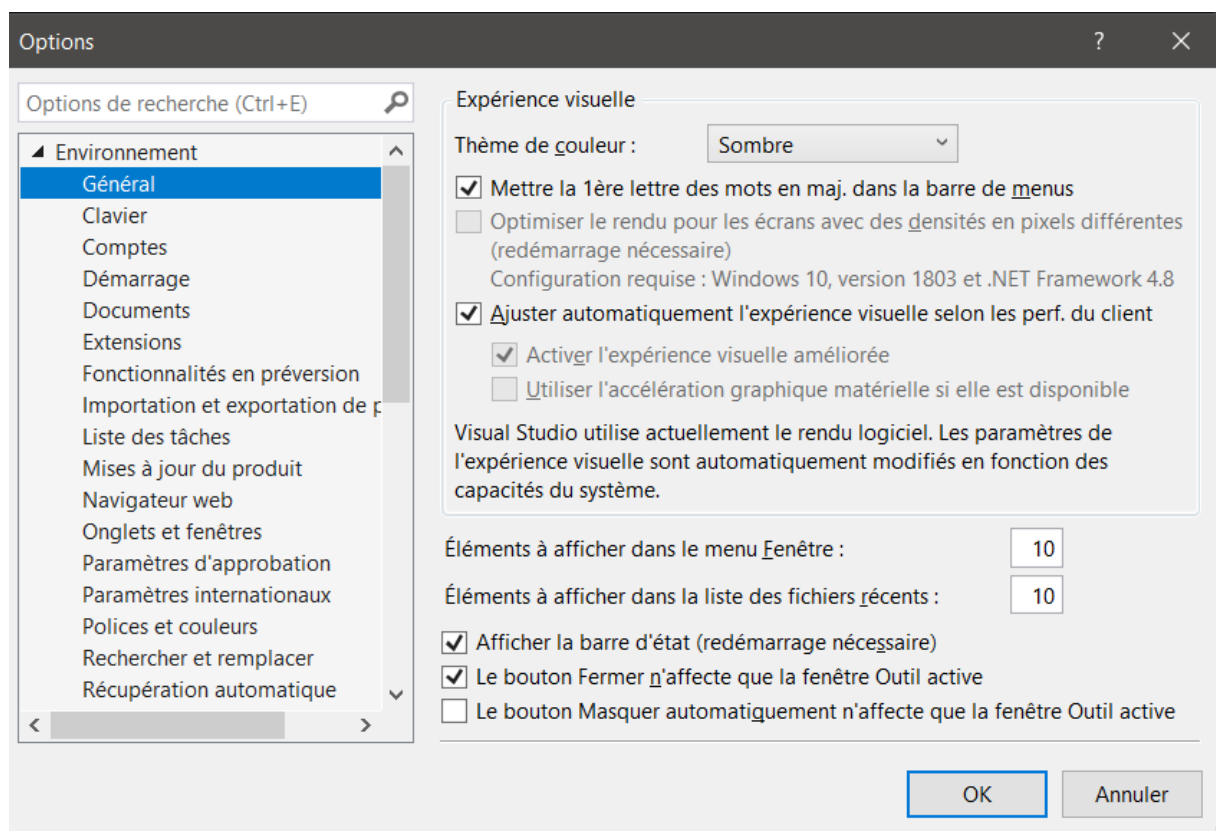
Choisissez ensuite un nom pour votre projet, son emplacement sur le disque, et un nom pour votre nouvelle solution.

Personnalisez votre environnement de développement

Une autre capacité intéressante de Visual Studio est sa configuration. Vous pouvez changer le thème, les polices de caractères, les couleurs des menus et barres d'outils pour créer l'environnement qui vous correspond.

Lorsque nous avons lancé Visual Studio pour la première fois, nous avons personnalisé la présentation en choisissant un **thème**. Ce n'était bien évidemment pas un choix définitif, et vous pouvez dès à présent le changer si vous le souhaitez.

Visual Studio vous offre une grande flexibilité sur la manière dont il s'affiche, et même sur la manière dont il fonctionne. Vous pouvez voir ces options de personnalisation dans le menu **Outils, puis Options**.



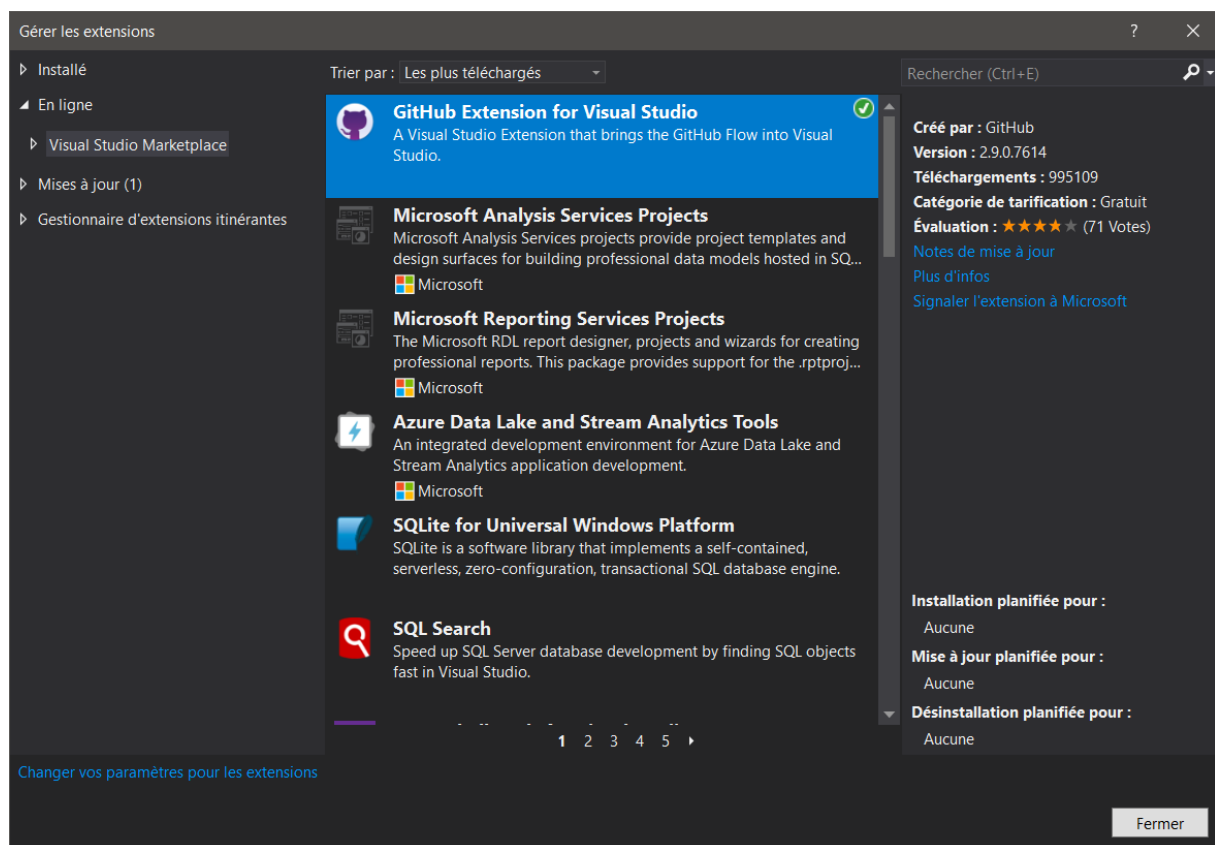
Allez dans Outils > Options pour personnaliser votre IDE

Bibliothèque d'extensions

En plus de pouvoir modifier son apparence selon vos préférences, vous pouvez améliorer Visual Studio avec des fonctionnalités supplémentaires. En allant dans **Extensions**, puis **Gérez les extensions**, vous pouvez choisir des ajouts dans la boutique Visual Studio.

Vous pouvez télécharger et installer d'autres outils, modèles et kits de développement (ou *SDK* pour "Software Development Kit", en anglais), pour vous aider à construire d'autres applications.

La boutique Visual Studio est un espace en ligne auquel n'importe quel développeur peut contribuer. Les membres de la communauté, comme vous, peuvent construire leurs propres extensions et les partager avec le reste des utilisateurs.



Explorez et contribuez à la bibliothèque d'extensions

Suite aux renforts de sécurité de Windows 10, vous devez lancer Visual Studio en mode **Administrateur** pour opérer tout changement. Voici comment procéder :

- cherchez Visual Studio dans le menu **Démarrer** ;
- faites un **clic droit** sur Visual Studio ;
- sélectionnez **Exécuter en tant qu'administrateur** ;
- sélectionnez **Oui** quand la fenêtre de sécurité apparaît.

Galerie NuGet

Grâce à la bibliothèque d’extensions, vous avez l’opportunité d’étendre les capacités de Visual Studio. Mais qu’en est-il de votre application ?

La galerie **NuGet** offre cette même possibilité pour votre application. L’idée est d’ajouter des fonctionnalités à l’aide de code compilé, appelé ***paquet, ou “package”, en anglais.***

Tout comme la boutique Visual Studio, cette galerie permet à tout développeur de partager du code qui lui semble utile à la communauté. Utiliser ces paquets vous permet d’inclure ce code, d’améliorer votre application et de gagner encore plus de temps, en évitant d’écrire ce code supplémentaire.

Pour certains types de projets, vous pouvez suivre l’évolution de vos paquets en consultant le fichier **.csproj**, à la racine de votre solution. De cette manière, vous serez informé de leur mise à jour et vous pourrez les restaurer, s’ils venaient à disparaître de votre application.

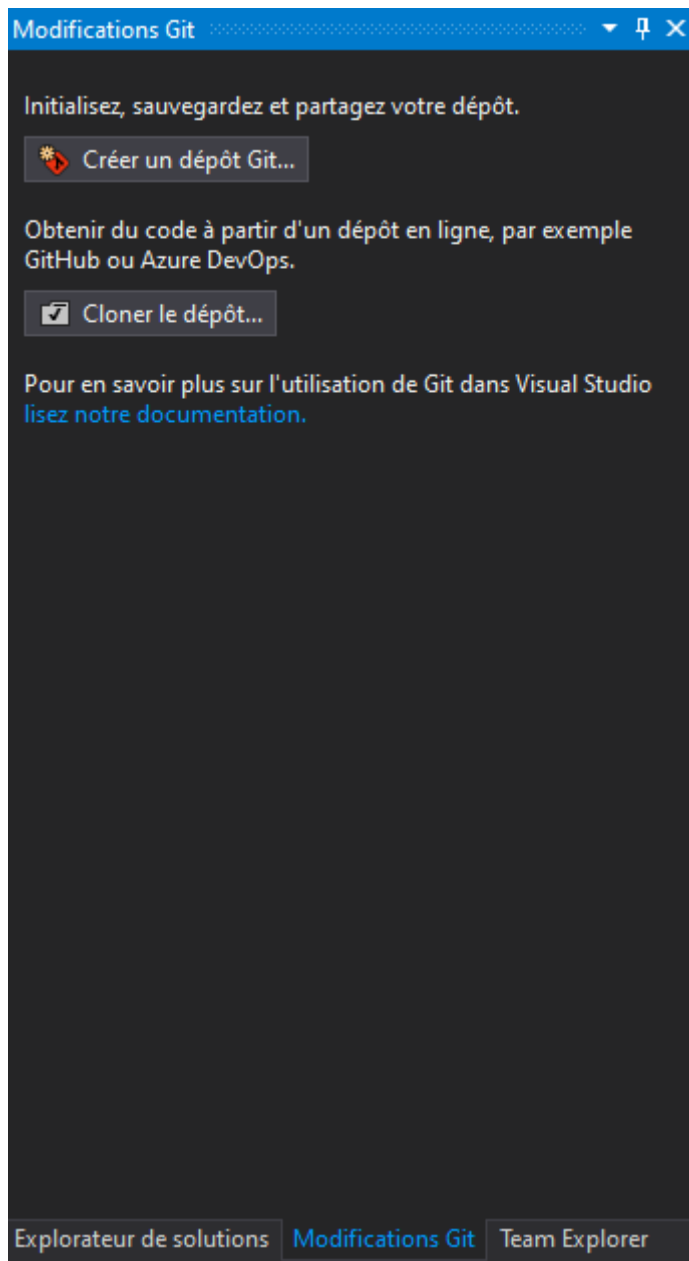
Clonez une application existante

Nous allons travailler sur une application existante stockée sur un service nommé **GitHub**.

Il s'agit d'un des hébergeurs de code les plus connus. Selon l'utilisation que vous en faites, GitHub peut être gratuit et public, ou proposer des abonnements mensuels pour accéder à toutes les fonctionnalités privées.

GitHub utilise **Git**, un système de versionnement de code, qui note tous les changements sur chaque fichier d'un projet, pour tous les collaborateurs. L'objectif de Git est de pouvoir suivre précisément et de manière **exhaustive** l'évolution du code, et de pouvoir revenir en arrière si nécessaire.

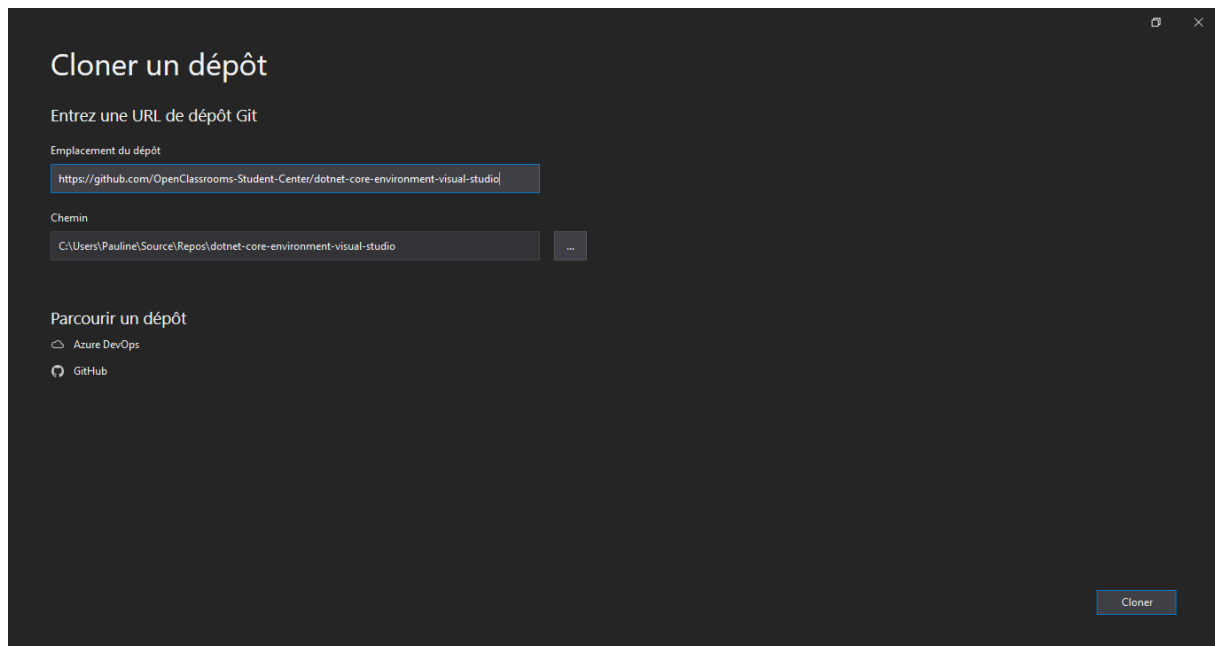
Visual Studio permet de gérer les **dépôts (ou “repositories”, en anglais) sous Git**. Vous trouverez vos dépôts depuis la fenêtre **Modifications Git**.



L'onglet Modifications Git

Dans ce chapitre, nous allons récupérer, c'est-à-dire **cloner**, une application existante. Un clone est simplement la copie téléchargée d'un dépôt en ligne vers votre machine locale. Vous pourrez ensuite ouvrir ce dossier avec Visual Studio pour y apporter vos modifications.

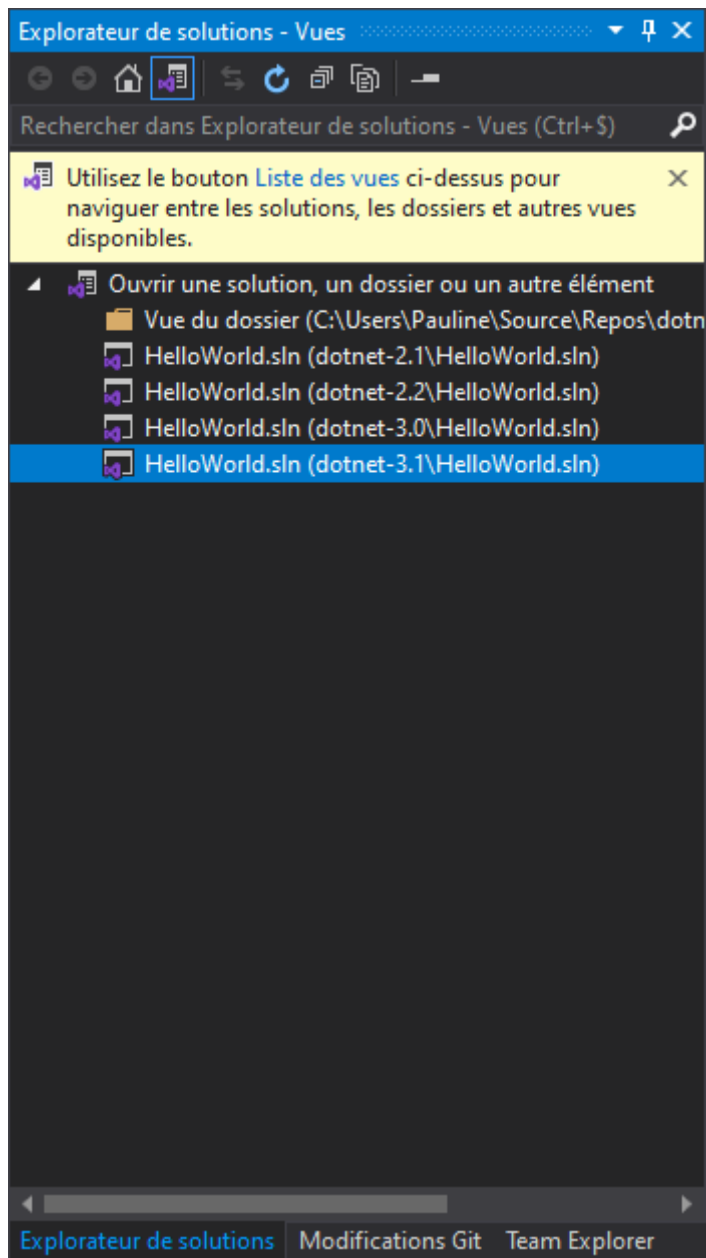
Lorsque vous cliquez sur le bouton **Clone**, il vous sera demandé de fournir l'URL du répertoire en question, et l'emplacement où vous souhaitez le stocker sur votre machine locale.



Cloner un dépôt

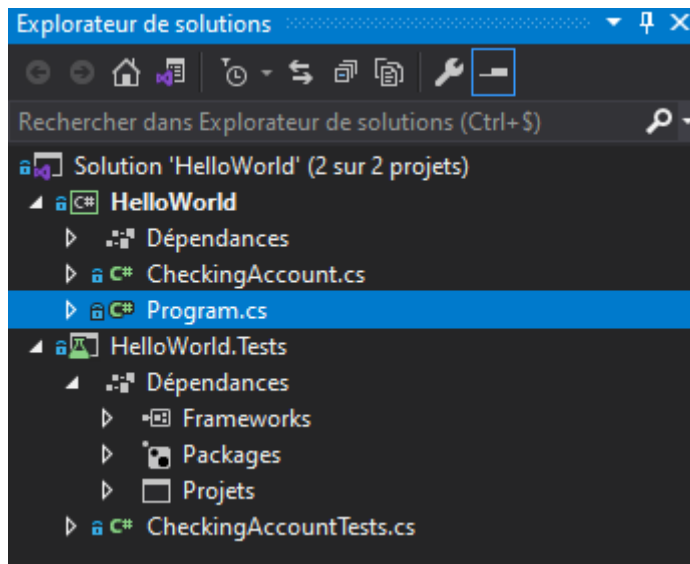
L'URL peut vous être fournie, ou vous pouvez la trouver en parcourant les milliers de dépôts GitHub existant sur [leur site](#). Pour cet exemple, vous pouvez cloner l'application [Hello World](#).

Une fois le dépôt cloné, vous pouvez aller dans la fenêtre **Explorateur de solutions**. Vous y trouverez les différentes solutions présentes dans le dépôt. Choisissez la solution dans le dossier **dotnet-3.1**, ou la version de .NET Core correspondant à celle que vous avez installée.



L'explorateur de solutions avec les dépôts

Une fois la solution sélectionnée, vous la retrouverez dans l'explorateur de solutions, comme dans les chapitres précédents.



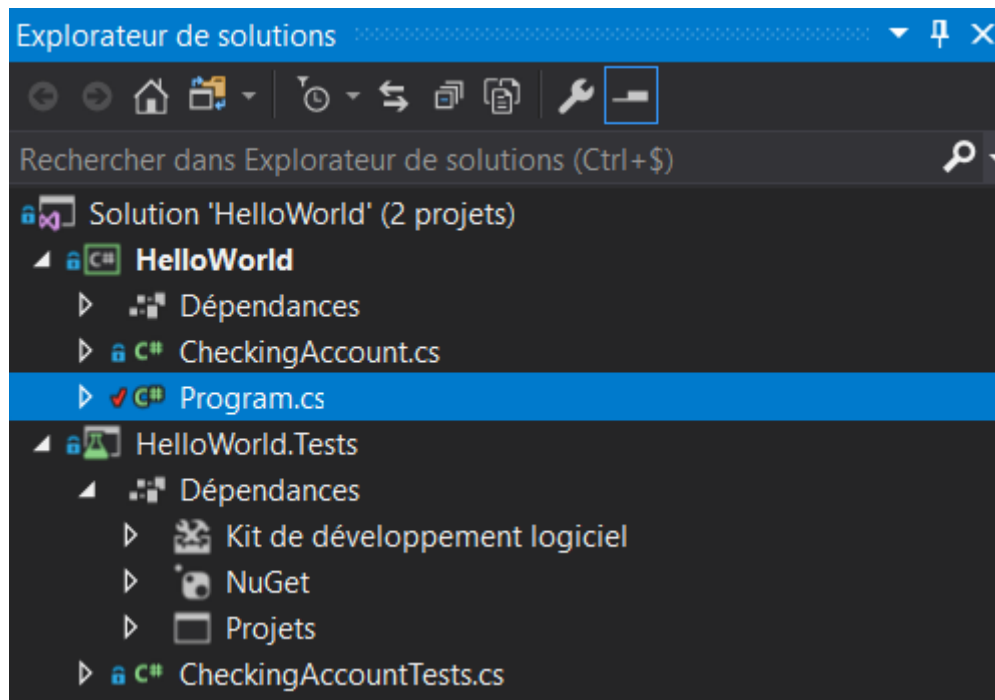
L'explorateur de solutions

En résumé :

- Git est un système de versionnement de code créé pour suivre tout changement sur un fichier, par vous ou vos collaborateurs ;
- GitHub est un hébergeur de code en ligne, ouvert au public. Il est basé sur Git ;
- un clone Git est la copie d'un dépôt en ligne sur votre machine locale ;
- Visual Studio permet de se connecter et de gérer des répertoires Git.

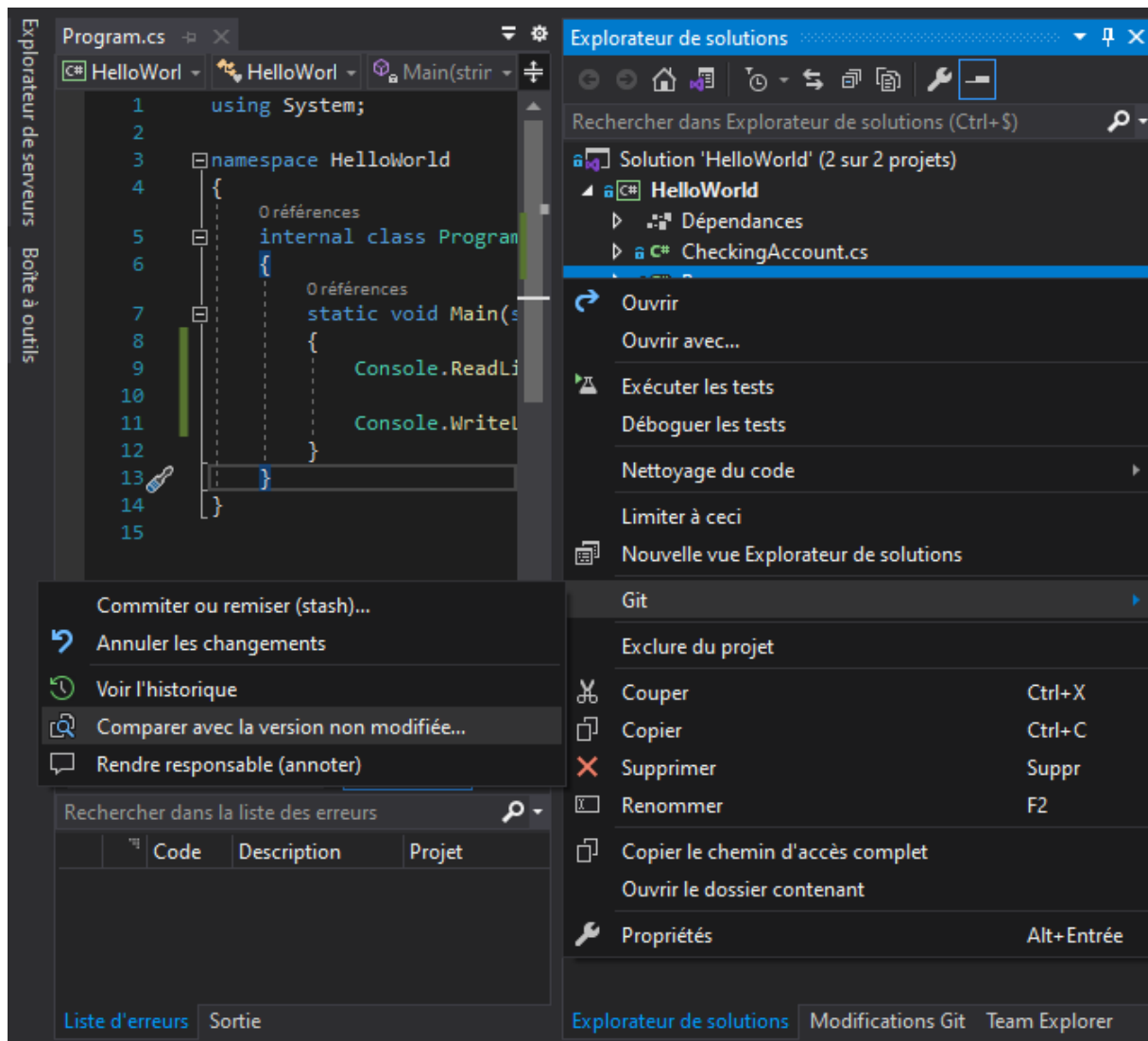
Gérez les conflits avec Git

Dans une application gérée sous Git, Visual Studio indique tout changement avec des marqueurs rouges dans l'explorateur. Ouvrez le fichier Program.cs par exemple, ajoutez une seconde ligne **Console.WriteLine("Hello World again!");** par exemple, juste en dessous de la ligne **Console.WriteLine("Hello World");** et enregistrez le fichier.



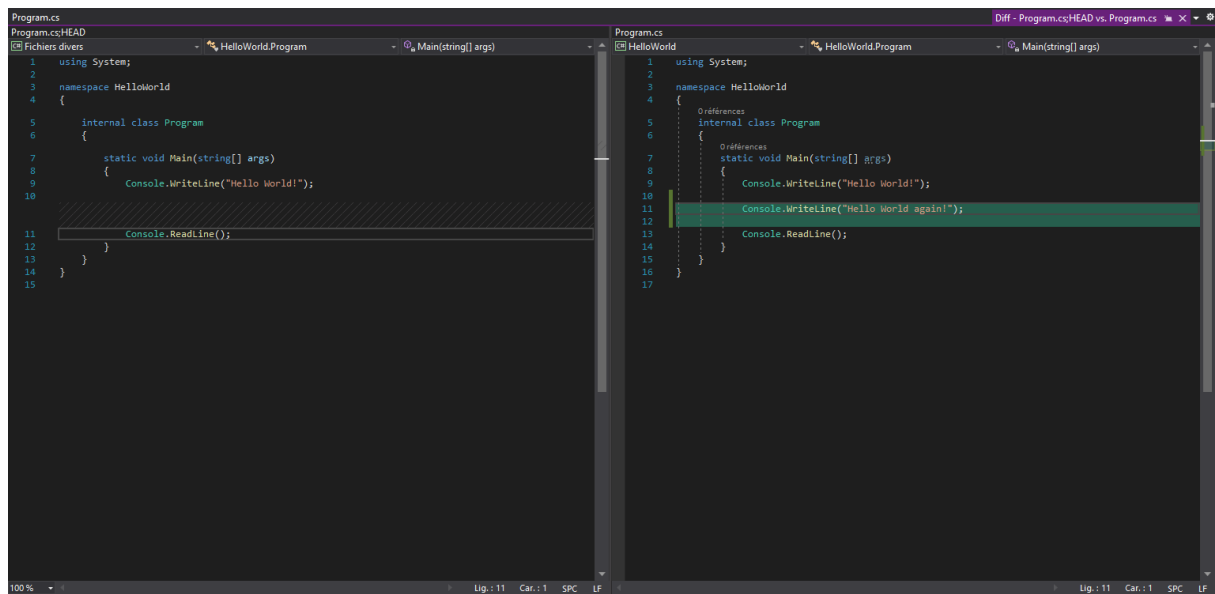
Le fichier modifié est affiché avec une coche rouge

Pour voir les **différences** entre les changements locaux et le fichier d'origine, faites un clic droit sur le nom du fichier, et **choisissez Git > Comparer avec la version non modifiée**.



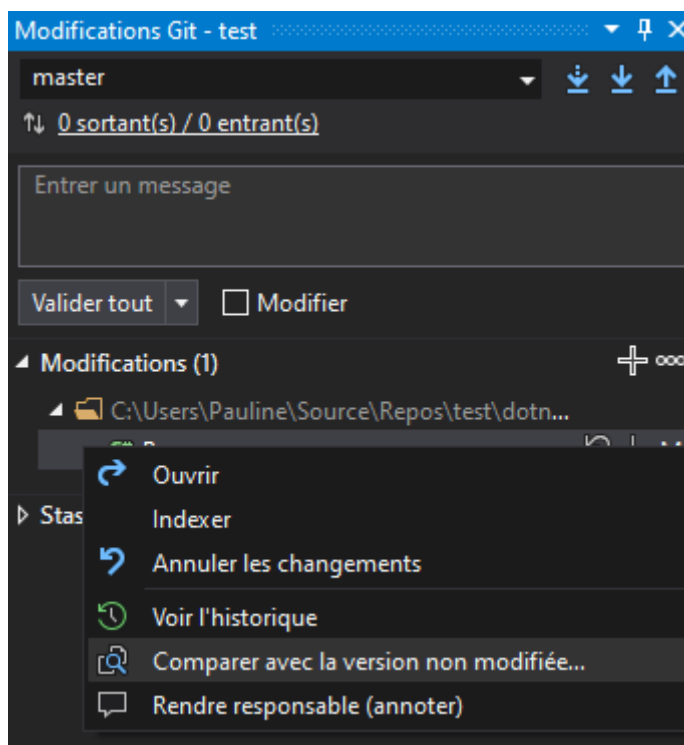
Cliquez sur "Comparer avec la version non modifiée" pour voir les différences

Le fichier actuel et sa version d'origine sont affichés côte à côte. Les ajouts sont surlignés en vert, les suppressions en rouge :



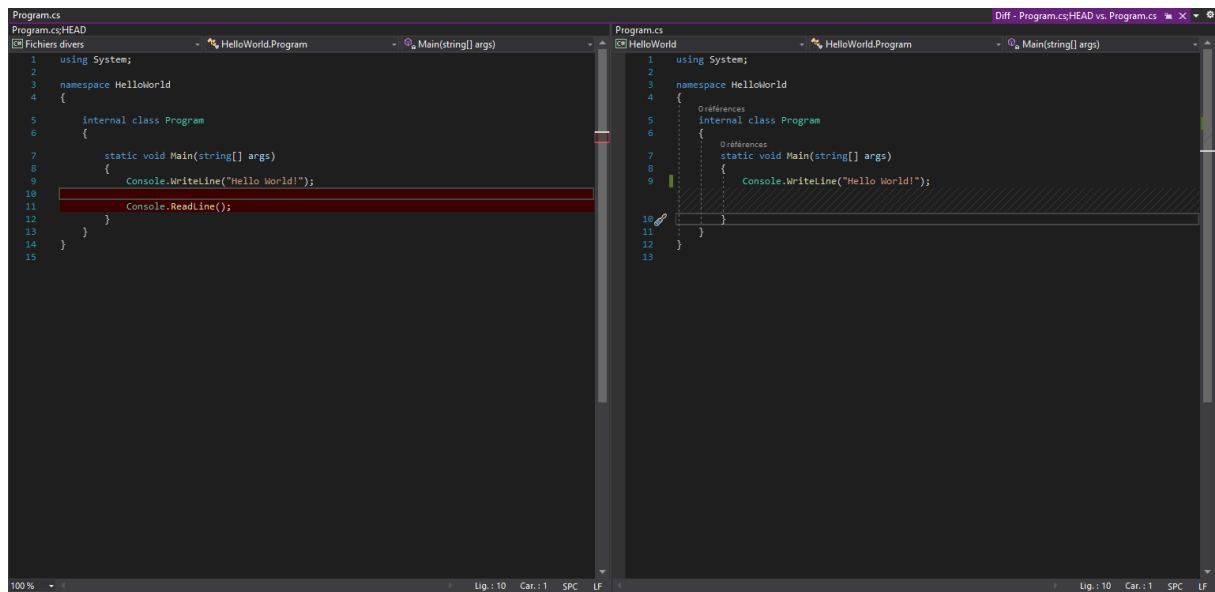
La fenêtre Diff

Supprimez maintenant la ligne **Console.ReadLine();** par exemple, et enregistrez le fichier. Vous pouvez également voir les fichiers qui ont été modifiés dans la fenêtre **Modifications Git**, et ouvrir la fenêtre Diff en faisant un clic droit sur le fichier et en sélectionnant **Comparer avec la version non modifiée**.



Vous pouvez également sélectionner "Comparer avec la version non modifiée" depuis la fenêtre Modifications Git

Vous verrez maintenant dans le Diff qu'une ligne a été supprimée, elle est surlignée en rouge.



Les lignes supprimées sont surlignées en rouge

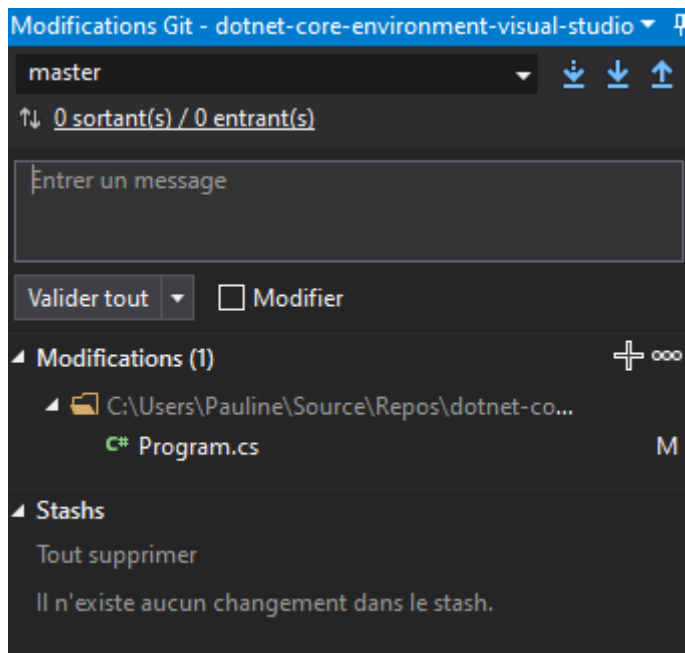
En résumé :

- Visual Studio suit tout changement effectué sur un fichier Git ;
- les fichiers modifiés sont marqués en rouge dans l'explorateur ;
- vous pouvez comparer le fichier actuel avec le fichier non modifié en faisant un clic droit sur son nom, et en sélectionnant **Comparer avec la version non modifiée** ;
- la fenêtre de comparaison affiche le code ajouté en vert, et le code supprimé en rouge.

Envoyez votre code vers le serveur

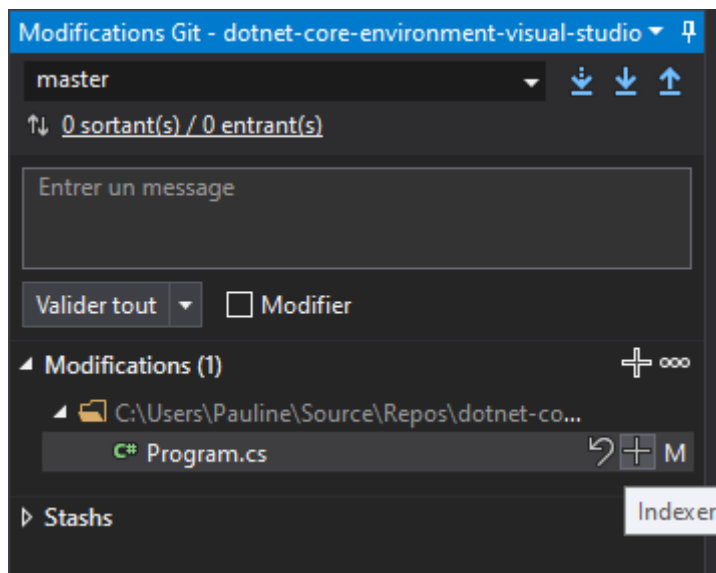
Une fois que vous avez fini d'écrire votre code, vous êtes prêt à envoyer vos changements sur votre dépôt Git, en ligne.

Allez dans la fenêtre **Modifications Git** pour voir la liste de tous les fichiers modifiés.



Sélectionnez "Vérifier la solution" et validez

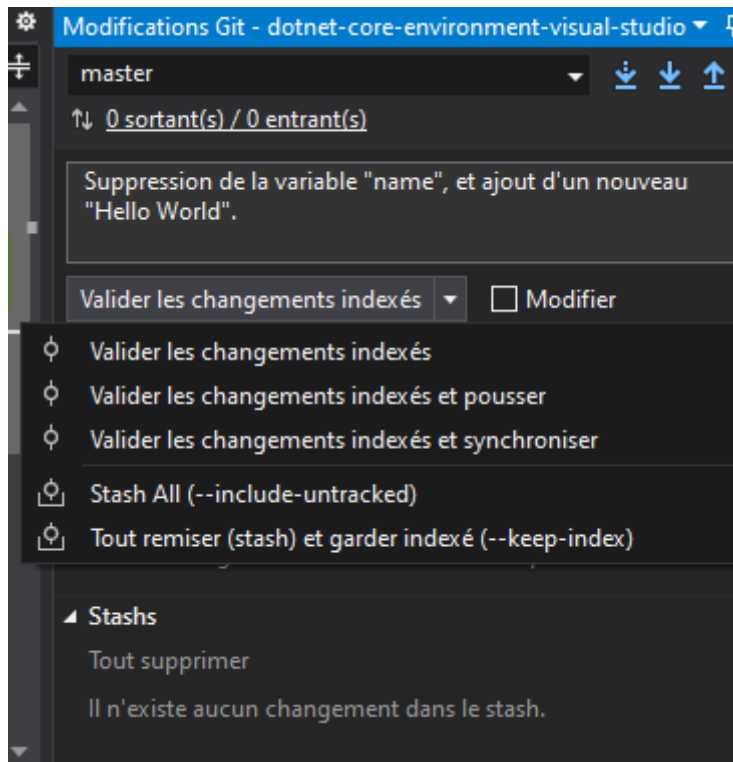
Les fichiers modifiés sont actuellement sous l'onglet **Modifications**. Il va falloir **indexer** les fichiers que vous souhaitez voir apparaître dans la prochaine version de votre code, à l'aide du bouton + qui apparaît à droite des fichiers lorsque vous passez la souris au-dessus. Les fichiers sélectionnés vont être déplacés vers l'onglet **Changements indexés**.



Indexez les fichiers à inclure dans la modification

Comme vous pouvez le constater, toute modification nécessite un commentaire (appelé "Message de validation" sur Visual Studio), sur ce qui a été modifié, et pour quelle raison. Ces détails seront très utiles lorsque vous aurez à consulter l'historique des changements, ou si vous partagez le code avec votre équipe.

Une fois que vous avez écrit votre commentaire, vous pouvez cliquer sur le menu déroulant **Valider les changements indexés**.



Les options de commit

Ce menu vous propose de choisir entre différentes actions :

- **valider les changements indexés** : sauvegarde les changements sur votre dépôt local ;
- **valider les changements indexés et pousser** : sauvegarde les changements sur vos dépôts en local et en ligne ;
- **valider les changements indexés et synchroniser** :
 - sauvegarde les changements sur votre dépôt local,
 - récupère les mises à jour éventuelles du dépôt en ligne,
 - sauvegarde les changements sur votre dépôt en ligne.

Dans quelles situations dois-je choisir une action plutôt qu'une autre ?

Utilisez **Indexer tout** lorsque vous faites un certain nombre de modifications que vous ne souhaitez pas perdre. Envoyer les changements vers le dépôt local signifie que vous marquez une étape dans ces changements, et que vous les considérez de qualité suffisante à un moment donné : vous créez un historique des modifications.

Si vous faites une erreur, ou si vous avez besoin de récupérer un changement antérieur, vous pouvez consulter l'historique pour récupérer ce dont vous avez besoin.

Cependant, si un problème arrivait à votre ordinateur, ou que vous travailliez depuis l'ordinateur d'une autre personne, vous pouvez à tout moment perdre cet historique, et, plus important, le code associé. C'est pourquoi vous allez le plus souvent utiliser **Valider les changements indexés et pousser**, pour créer votre historique, mais surtout le sauvegarder sur votre dépôt en ligne. De cette manière, vous pourrez récupérer le code depuis n'importe où, et le protéger en cas de problème technique avec votre ordinateur.

Enfin, utilisez **Valider les changements indexés et synchroniser** lorsque vous travaillez à plusieurs sur une même application. De cette manière, vous récupérerez les changements de vos collaborateurs en local, et leur enverrez vos changements. Attention cependant à toujours bien récupérer **d'abord** leurs changements ("synchroniser") **avant** d'envoyer les vôtres ("pousser"). Cela vous évitera beaucoup de conflits.

En résumé :

- chaque fichier modifié apparaît dans la fenêtre **Modifications Git** ;
- pour chaque envoi de changement intermédiaire (ou "commit"), un commentaire doit détailler quels changements ont été faits, et pourquoi ;
- il existe trois types d'actions :
 - **valider les changements indexés** : sauvegarde les changements sur votre dépôt local,
 - **valider les changements indexés et pousser** : sauvegarde les changements sur vos dépôts en local et en ligne,
 - **valider les changements indexés et synchroniser** :
 - sauvegarde les changements sur votre dépôt local,
 - récupère les mises à jour éventuelles du dépôt en ligne,
 - sauvegarde les changements sur votre dépôt en ligne.

Découvrez .NET MVC

Qu'est-ce que .NET Core MVC, au juste ?

MVC est l'acronyme utilisé pour désigner le design pattern **modèle-vue-contrôleur**. Il est plus ou moins utilisé dans divers langages et frameworks. Alors, en quoi le MVC de .NET est-il différent ?

ASP.NET MVC est un framework qui permet de créer des applications web performantes. Il applique le **design pattern modèle-vue-contrôleur** à la plateforme de développement ASP.NET. Il s'agit également d'une méthode de **développement agile**, d'une partie centrale de la **plateforme ASP.NET** et d'un outil qui remplace les formulaires web lents et obsolètes. Avec .NET Core, le framework MVC va encore plus loin en rendant aussi possible le développement **d'API RESTful**.

Vous comprenez donc pourquoi .NET MVC est bien plus qu'un simple design pattern. À mesure que vous avancerez dans ce cours, vous percevrez mieux tout ce que .NET MVC peut vous offrir, et notamment sa puissance et sa flexibilité pour les développeurs .NET.

Pourquoi choisir d'apprendre .NET Core MVC ?

Comme nous l'avons vu, .NET Core ne se limite pas au C#. Il s'agit d'une plateforme de développement logiciel complète qui permet aux développeurs de créer des applications compatibles avec tous les systèmes d'exploitation et tous les appareils. Vous pouvez créer des applications de bureau, des applications Web, des applications mobiles et même des applications embarquées. .NET MVC n'est qu'un maillon de cette plateforme. Alors en quoi est-ce utile de l'apprendre ?

La réponse est simple. Pour devenir un développeur .NET **compétitif**, vous devez connaître et comprendre .NET MVC. Si vous voulez être un développeur **convoité**, vous devez acquérir des compétences en développement .NET.

Il existe actuellement bien plus d'offres d'emploi aux États-Unis et dans le reste du monde qui mentionnent le **C#** et/ou le **développement .NET** que *tout autre langage* (selon indeed.com). De plus, .NET MVC est au cœur du développement .NET.

Une fois Visual Studio installé, vous êtes prêt à démarrer. La première application .NET MVC que nous allons créer s'appellera **Watchlist**. Il s'agit d'un simple système permettant de répertorier et noter les films que vous avez vus ou que vous possédez. L'application utilisera une **base de données relationnelle** pour stocker les informations que vous saisissez, notamment la note (de 1 à 5 étoiles) que vous attribuez à chaque film. Vous pourrez ensuite faire des recherches dans votre liste de films, l'afficher et la trier en lui appliquant divers critères.

Ce sera une application intéressante à développer, et elle pourrait même vous être utile une fois terminée. Allons-y !

En résumé

Ce chapitre était bref, mais il vous a fourni quelques informations clés pour vous préparer à la suite :

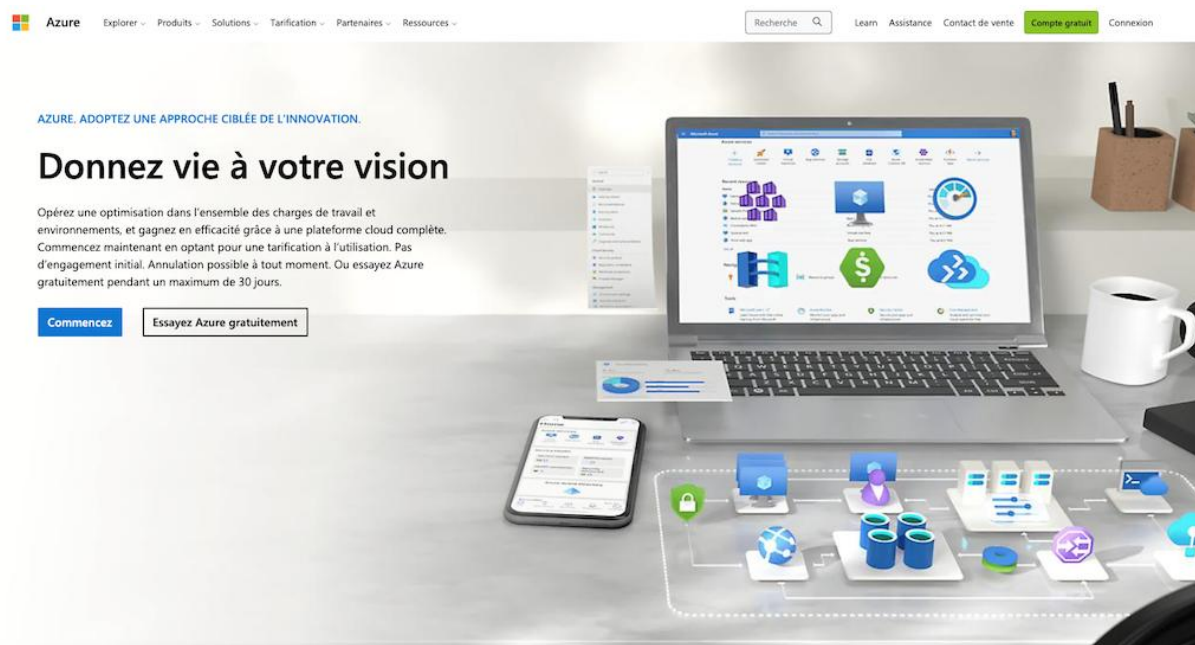
- Vous avez vu en quoi .NET MVC est bien plus qu'un simple design pattern ; il s'agit d'un framework de développement qui utilise le pattern MVC pour créer des applications web vraiment performantes.
- Vous avez appris pourquoi il s'agit d'une partie très importante de votre parcours pour devenir un développeur .NET compétent et en quoi ce framework est au cœur de la plateforme de développement .NET dans son ensemble.

Exécutez votre premier projet .NET MVC

Créez un compte Azure


Si vous disposez déjà d'un compte Azure que vous prévoyez d'utiliser dans le cadre de ce cours, vous pouvez sauter cette section et passer à la section « Créez un nouveau projet ». Sinon, suivez les étapes décrites dans cette section pour créer gratuitement votre propre compte Azure.

Ouvrez une fenêtre dans votre navigateur et accédez à azure.microsoft.com. Vous verrez la page suivante :




Page d'accueil de Microsoft Azure

Tout le monde peut s'inscrire et créer un compte Azure gratuit. Cliquez sur *Compte gratuit* en haut à droite de la page.

 Azure

Explorer ▾ Produits ▾ Solutions ▾ Tarification ▾ Partenaires ▾ Ressources ▾


Recherche 

Learn Assistance Contact de vente Connexion

Créez dans le cloud avec un compte gratuit Azure

Créez, déployez et gérez des applications sur plusieurs clouds, localement et à la périphérie

[Démarrer gratuitement](#) [À l'utilisation](#)



Les services les plus appréciés sont gratuits pendant 12 mois

↓ [Affiche tous les services](#)

+

55+ autres services toujours gratuits

↓ [Affiche tous les services](#)

+

Commencez avec un crédit Azure de USD200*

↓ Vous avez 30 jours pour l'utiliser, en plus des services gratuits.

Voici quelques exemples de ce que vous pouvez accomplir avec Azure

Page Compte gratuit

Cliquez sur le bouton *Démarrer gratuitement* au milieu de l'écran pour commencer l'inscription.

Il vous est maintenant demandé de choisir le compte Microsoft à utiliser pour votre compte Azure. Si vous avez un compte Microsoft et souhaitez l'utiliser, sélectionnez-le dans la liste et suivez les instructions. La configuration est très simple.

Si vous n'avez pas de compte, ou si celui que vous souhaitez utiliser n'est pas dans la liste, choisissez *Utiliser un autre compte*. Vous serez dirigé vers la page de connexion Microsoft.

Si votre compte Microsoft n'était pas répertorié sur la page précédente, vous pouvez vous connecter ici. Si vous n'avez pas de compte Microsoft, ou si vous souhaitez en créer un nouveau, cliquez sur *Créez-en un !*



Créer un compte

xyz@example.com I

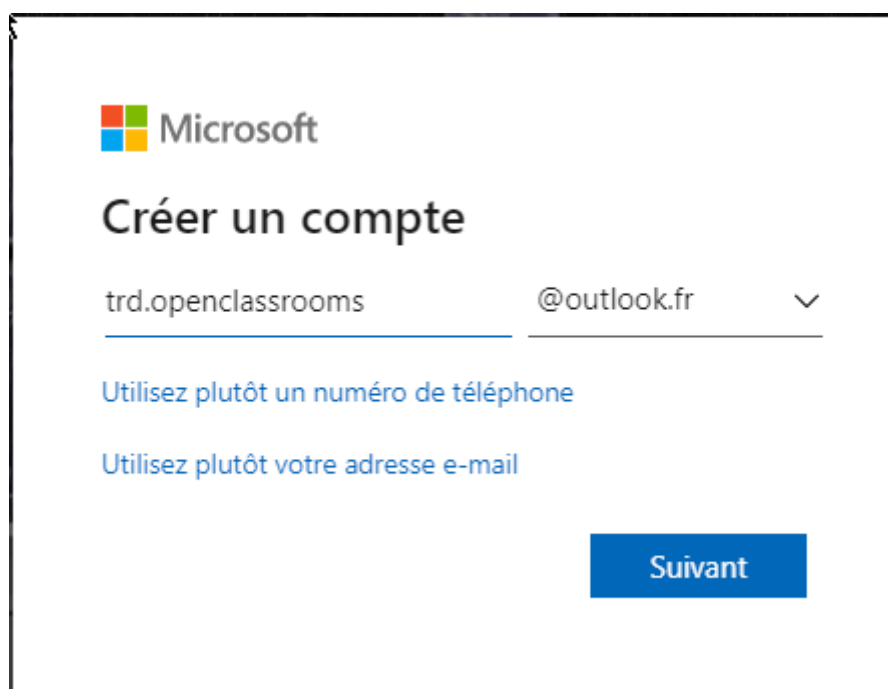
[Utilisez plutôt un numéro de téléphone](#)

[Obtenez une nouvelle adresse e-mail](#)

Suivant

Fenêtre Créer un compte

Vous avez ici la possibilité de créer un nouveau compte Microsoft en utilisant votre adresse e-mail actuelle ou votre numéro de téléphone. Vous pouvez également obtenir une nouvelle adresse e-mail que vous associerez ensuite à votre compte Microsoft. Chaque option suit des étapes similaires : choisissez celle qui en demande le plus pour ne rien rater. Choisissez *Obtenez une nouvelle adresse e-mail*.



Microsoft

Créer un compte

trd.openclassrooms @outlook.fr ▼

[Utilisez plutôt un numéro de téléphone](#)

[Utilisez plutôt votre adresse e-mail](#)

Suivant

Fenêtre Créer une

nouvelle adresse e-mail

Pour ma part, j'ai choisi une nouvelle adresse e-mail outlook.fr que je vais utiliser pour ce compte. Cliquez sur le bouton *Suivant* pour définir un mot de passe.



← trd.openclassrooms@outlook.fr

Créer un mot de passe

Entrez le mot de passe que vous souhaitez utiliser avec votre compte.

.....|


☐ Afficher le mot de passe

☐ J'aimerais obtenir des informations, des conseils et des offres concernant des produits et services Microsoft.

Choisir **Suivant** signifie que vous acceptez le [Contrat de services Microsoft](#) et la [Déclaration sur la confidentialité et les cookies](#).

Toutes les informations demandées sont obligatoires et seront utilisées par Microsoft pour créer votre compte et vous permettre de vous connecter aux produits et appareils Microsoft. Les données fournies seront associées à votre compte afin de personnaliser et synchroniser votre expérience sur différents appareils. Sous réserve de votre choix relatif à la publicité, Microsoft utilisera également ces données pour personnaliser la publicité qui vous est adressée.

Conformément à la loi Informatique et Libertés, vous disposez d'un droit d'accès et de rectification aux données personnelles vous concernant, ainsi que du droit de vous opposer au traitement de vos données. Vous pouvez également nous adresser des instructions spécifiques concernant l'utilisation de vos données après votre mort. Pour exercer ces droits, veuillez suivre les instructions figurant dans la déclaration de confidentialité de Microsoft.

Suivant 

Fenêtre Créer un mot de passe

Après avoir créé le mot de passe pour la nouvelle adresse e-mail et de ce fait, le nouveau compte, cliquez sur le bouton *Suivant* pour passer la vérification anti-robot.



← trd.openclassrooms@outlook.fr

Créer un compte

Veuillez résoudre l'énigme pour que nous sachions que vous n'êtes pas un robot.

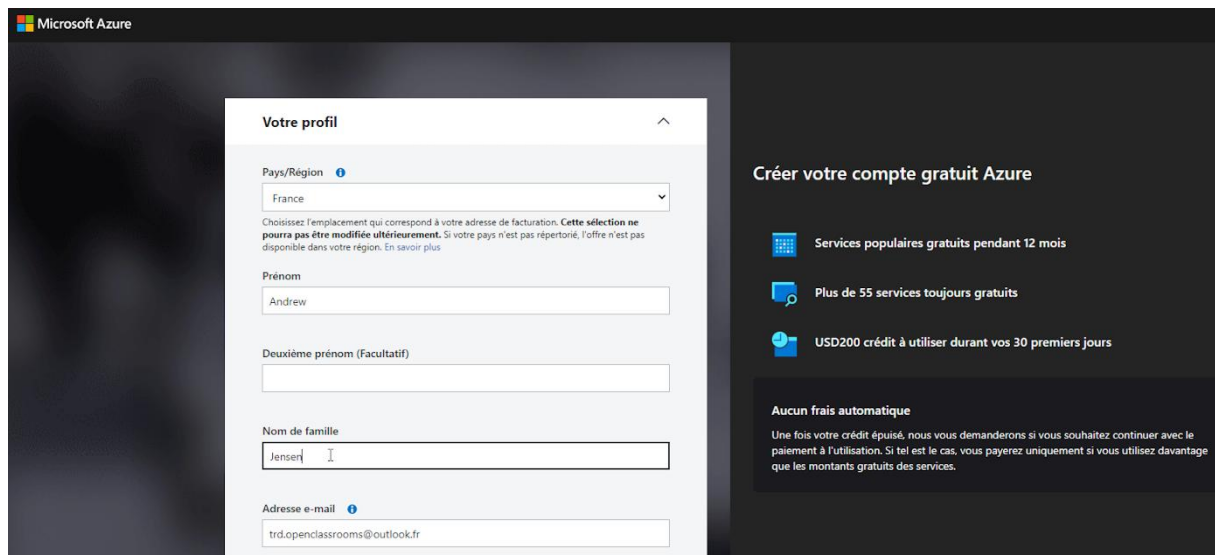


Suivant



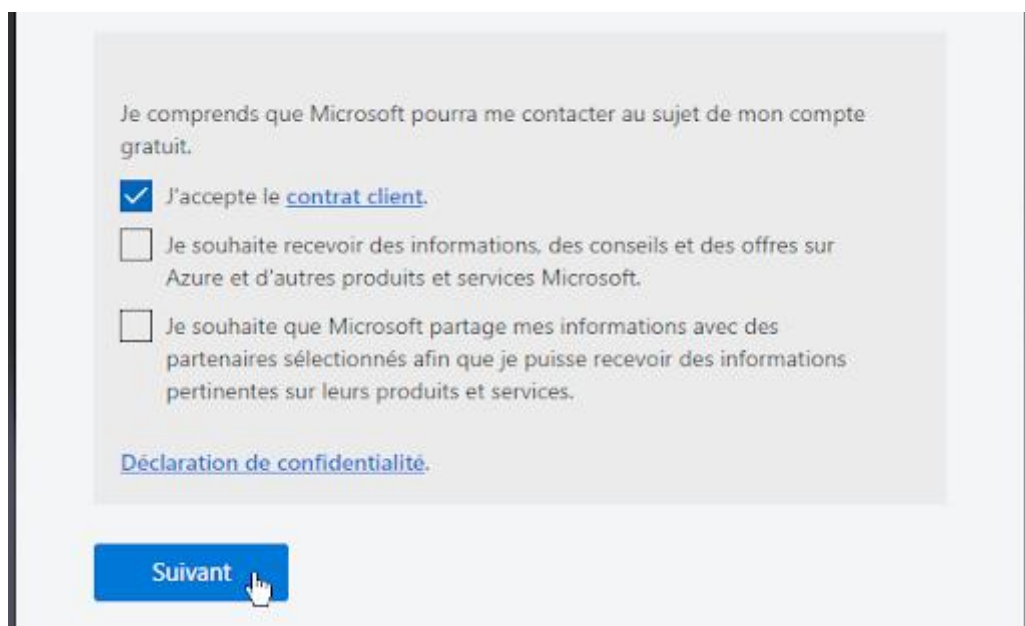
Vérification anti-robot

Cliquez sur le bouton *Suivant* pour passer à l'étape finale de ce processus. À ce stade, votre compte Azure est créé et presque prêt à être utilisé. Il ne reste que quelques informations à remplir sur le profil, puis une vérification d'identité pour prouver que vous êtes bien la personne que vous prétendez être. Remplissez le formulaire, cliquez sur le bouton *Suivant*, puis suivez les instructions supplémentaires.



Formulaire d'information sur le profil

Une fois le formulaire rempli et la vérification d'identité effectuée, cochez la case située à côté du contrat d'abonnement, puis cliquez sur *Inscription*.



Après le profil, l'inscription

La configuration peut prendre quelques instants, mais dès que le processus est terminé, vous êtes redirigé vers une page de confirmation depuis laquelle vous pouvez accéder au portail Azure.

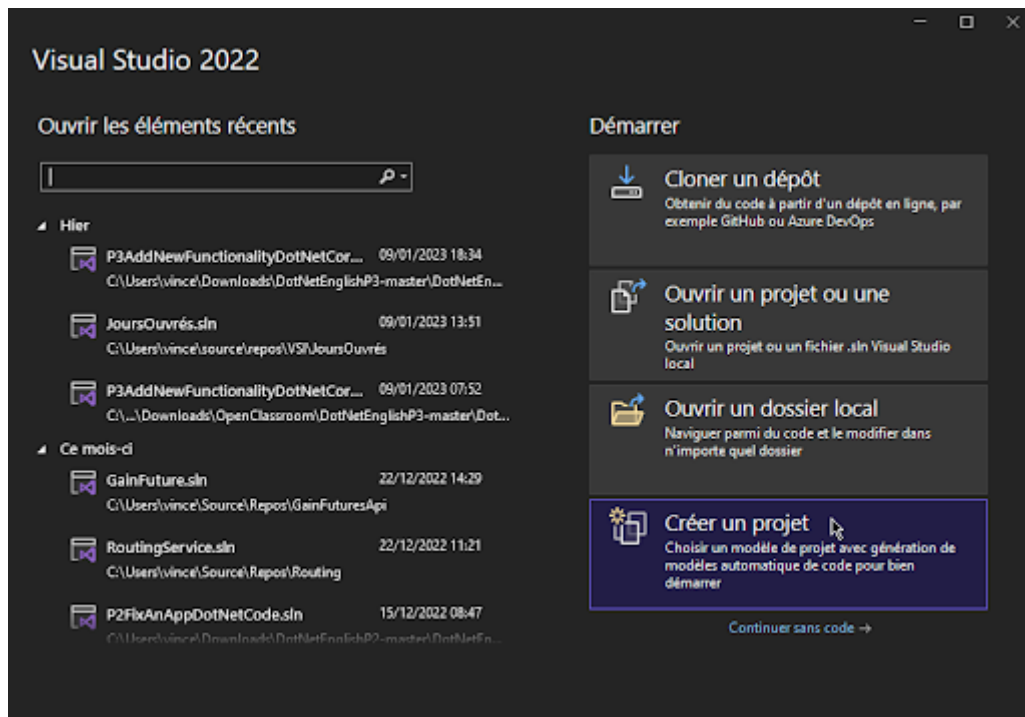
Après votre inscription, vous pouvez accéder à votre portail.

Pas besoin d'accéder au portail puisque rien ne s'y trouve pour le moment. Mais vous disposez désormais d'un compte et d'un portail Azure où vous pouvez héberger toutes les applications web que vous développez, y compris l'application Watchlist que vous êtes sur le point de créer.

Créez un nouveau projet

Vous avez installé Visual Studio et créé votre compte Azure. Vous pouvez maintenant créer votre projet.

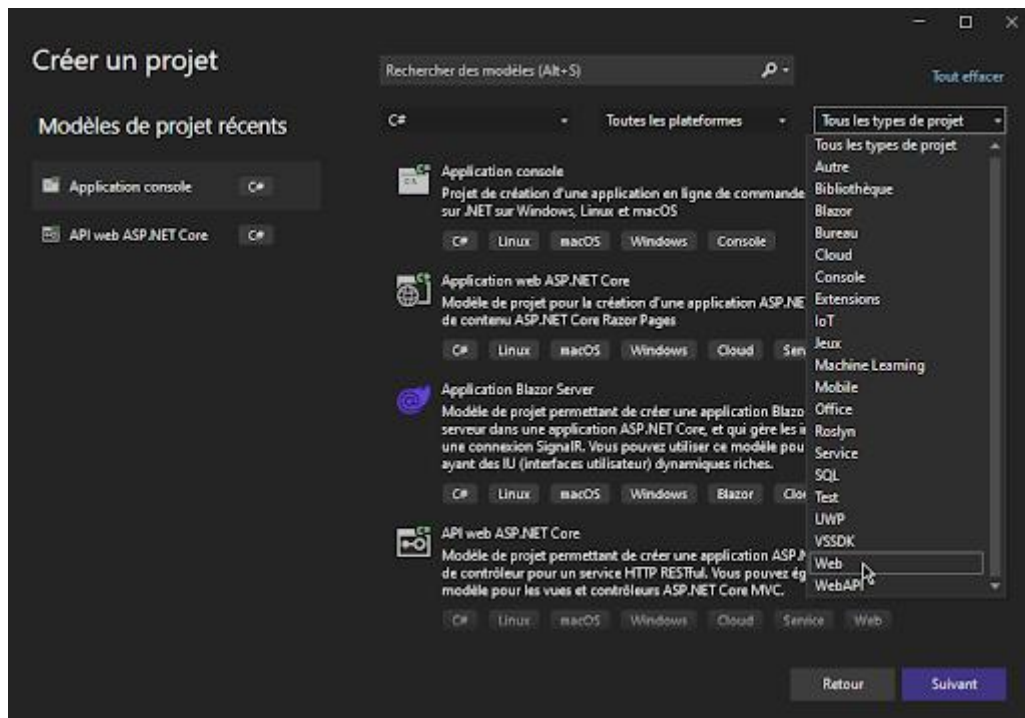
Lancez Visual Studio (à partir du menu *Démarrer*, de l'icône sur le bureau ou de l'icône de lancement rapide). La fenêtre qui s'affiche (ci-dessous) vous offre plusieurs possibilités.



Fenêtre d'accueil de Visual Studio

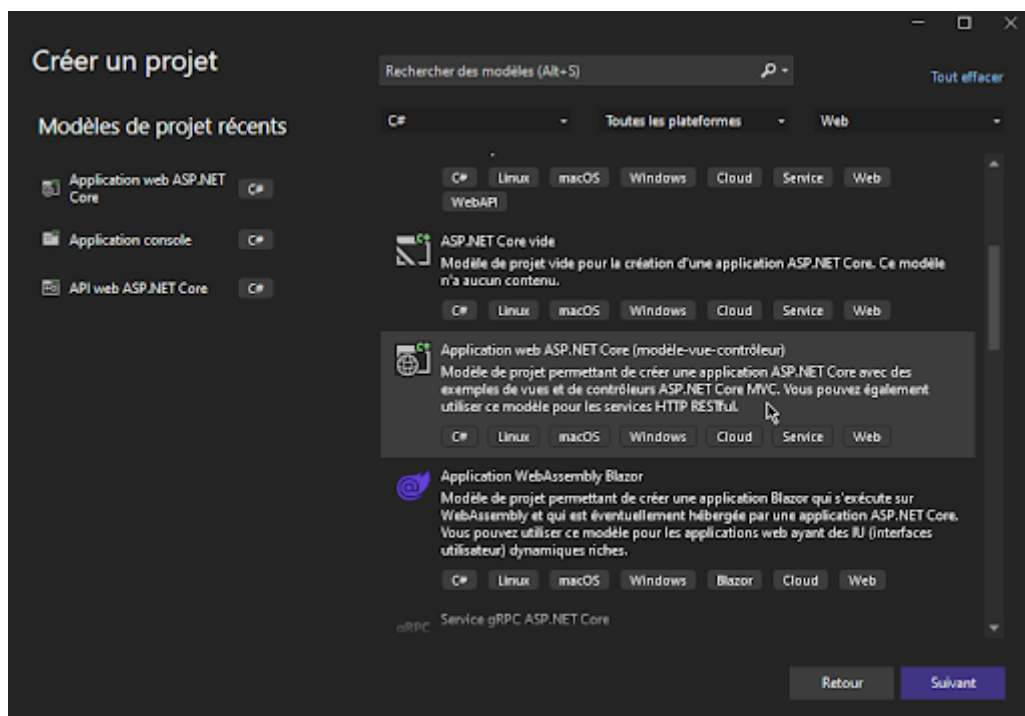
Comme nous voulons créer tout de suite notre projet, sélectionnez *Créer un projet*.

La fenêtre qui suit vous permet de spécifier le type de projet que vous souhaitez créer. NET Core possède un tel choix de propositions qu'il est parfois nécessaire de limiter les choix en utilisant les listes déroulantes *Tous les langages*, *Toutes les plateformes* et *Tous les types de projet*. Choisissez *Web* en bas de la liste déroulante *Tous les types de projet*, comme le montre la capture d'écran ci-dessous.



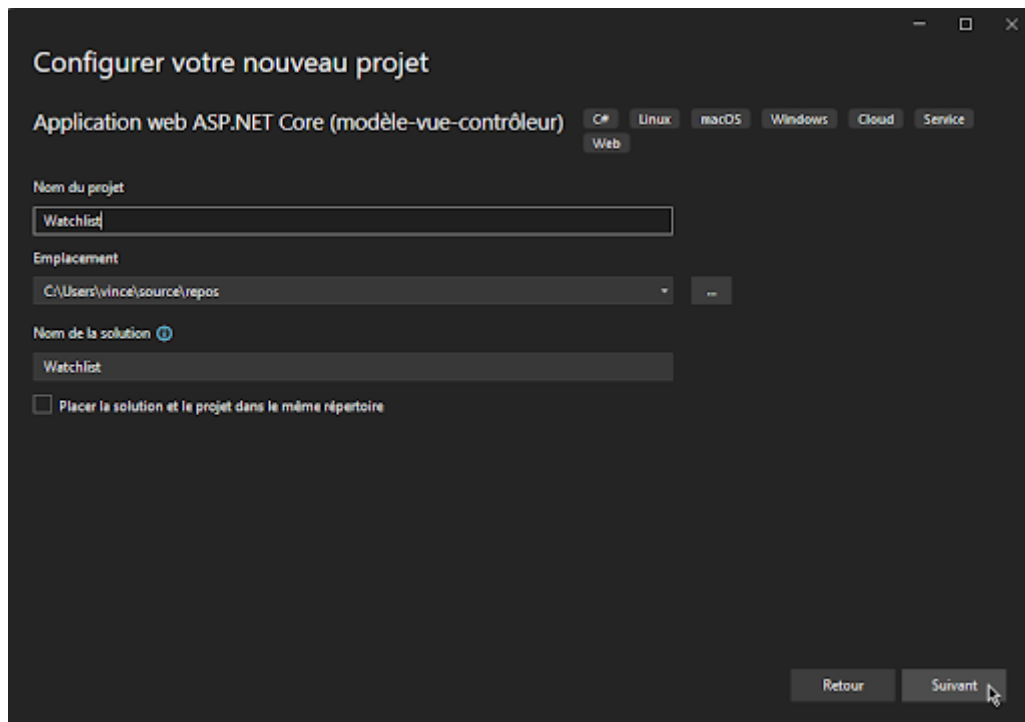
Fenêtre Créer un projet

Cela restreindra la liste des types de projets aux projets Web. Cherchez et sélectionnez *Application Web*, comme indiqué ci-dessous, puis cliquez sur le bouton *Suivant*.



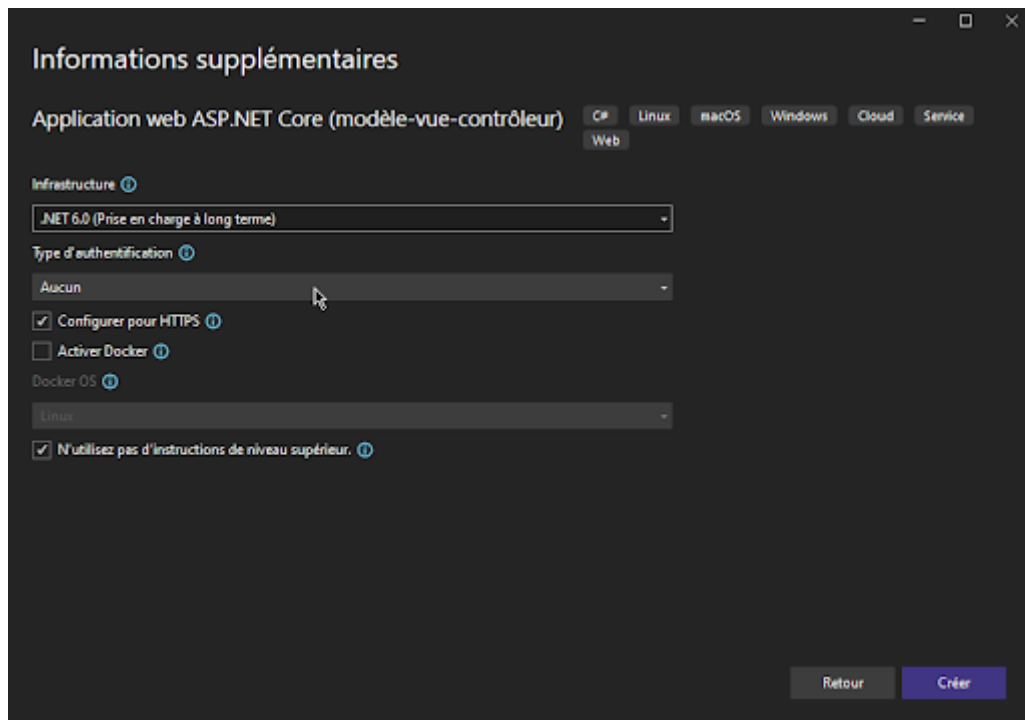
Choisissez Application web ASP.NET Core

La fenêtre qui suit vous permet de configurer votre projet, à savoir lui donner un nom et définir son emplacement. Nommez votre projet Watchlist, sélectionnez l'emplacement dans lequel vous souhaitez le créer, puis cliquez sur *Suivant*.

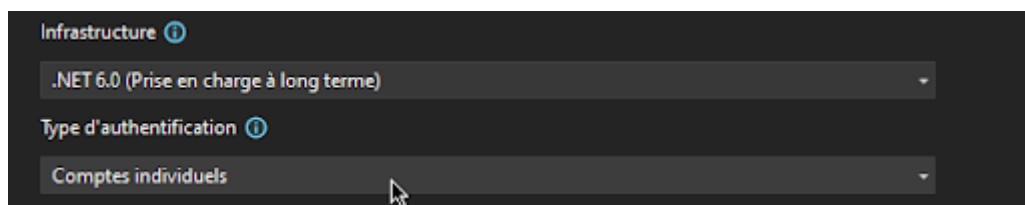


Fenêtre Configurer votre nouveau projet

Remplacez le type d'authentification *Aucun* par un type qui utilise une connexion sécurisée pour chaque utilisateur. Dans la fenêtre qui s'ouvre, sélectionnez *Comptes individuels*. Assurez-vous que l'option *Stocker les comptes d'utilisateurs dans l'application* est sélectionnée dans la liste déroulante, puis cliquez sur OK. Cela va créer votre projet. Entity Framework sera installé et des comptes d'utilisateur sécurisés seront intégrés au projet. Vous en aurez besoin plus tard, en particulier lorsque vous souhaitez qu'on accède à votre application via une connexion sécurisée. Cliquez maintenant sur le bouton *Créer*.



Fenêtre Informations supplémentaires



Fenêtre Modifier l'authentification

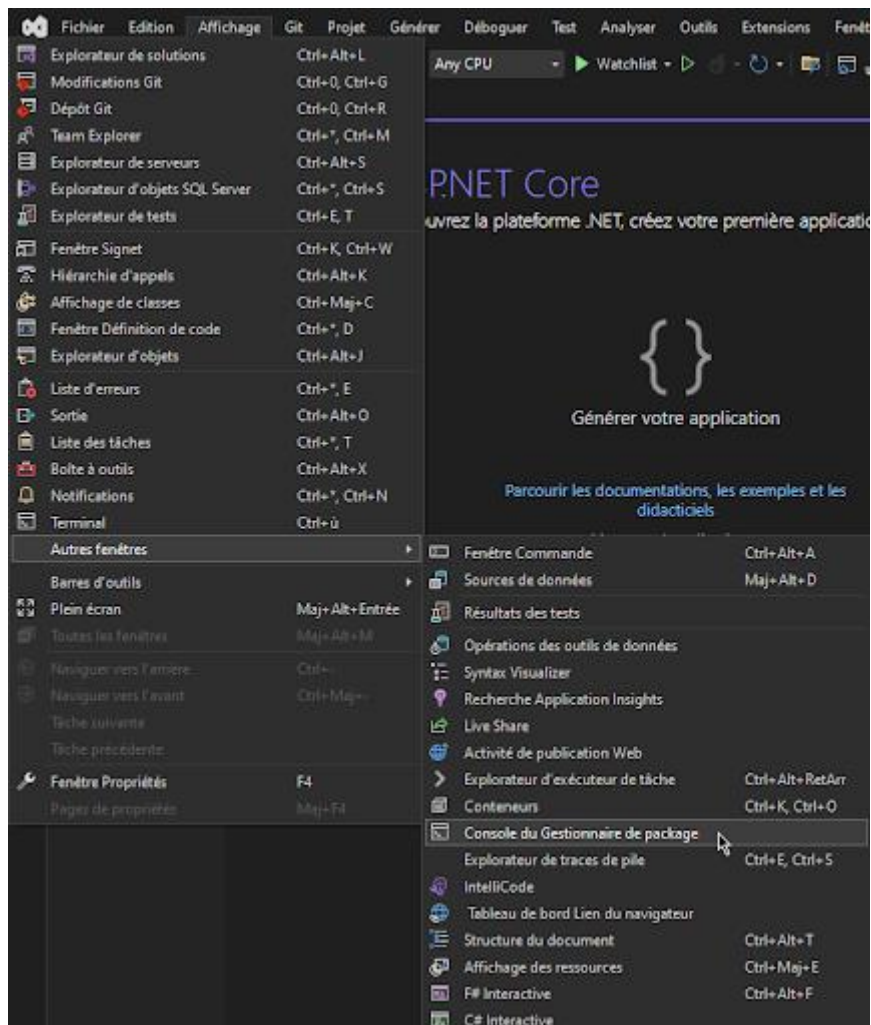
Visual Studio est en train de générer votre application de base. Cette application intègre un système sécurisé de gestion des comptes d'utilisateur, qui prend notamment en charge l'inscription et la connexion. Ces comptes sont créés et gérés à l'aide d'**ASP.NET Core Identity**, qui est inclus dans l'application sous la forme d'une bibliothèque de classes Razor.

Vous pouvez davantage personnaliser les comptes d'utilisateur, mais nous aborderons cela dans prochain cours.

Initialisez la base de données

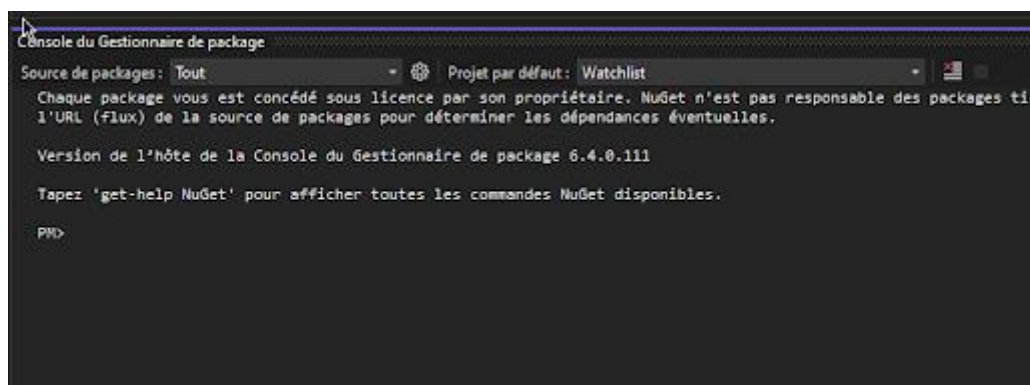
Avant de tester l'application, il reste une dernière étape à franchir pour s'assurer que tout fonctionne correctement. Lorsque vous avez créé le projet, le code pour créer et interagir avec votre base de données a été généré, mais la base de données elle-même n'a pas été mise à jour. Pour ce faire, vous allez utiliser un processus appelé **migration code first**.

Pour exécuter une migration code first, ouvrez la **console du Gestionnaire de package**. Pour trouver cette fenêtre, sélectionnez *Affichage > Autres fenêtres > Console du Gestionnaire de package*.



Comment trouver la Console du Gestionnaire de package

La fenêtre s'ouvre dans la partie inférieure de votre IDE et ressemble à ceci :



Console du Gestionnaire de package

Dans l'invite, saisissez la commande ci-après, puis appuyez sur *Entrée* :

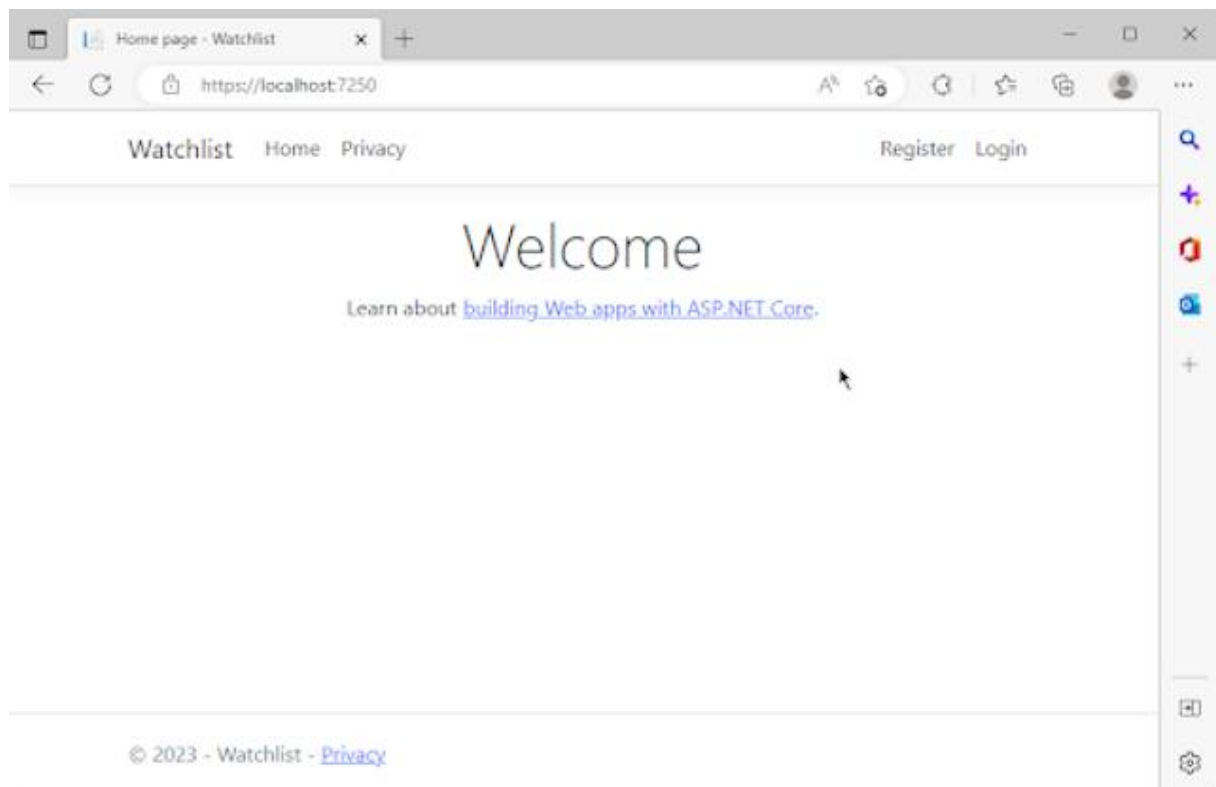
update-database

Cette commande va mettre à jour la base de données en utilisant l'état de migration actuel qui a été généré lorsque vous avez créé le projet.

Inscrivez-vous et connectez-vous

Exécutons maintenant le projet. Appuyez sur F5 pour lancer le débogage. Vous pouvez également utiliser le menu *Déboguer* > *Démarrer* le débogage ou cliquer sur le bouton *IIS Express* dans la barre d'outils.

Lorsque votre code s'exécute, vous devriez voir cette page :



Page d'accueil de votre application

Observez les liens *Register* (Inscription) et *Login* (Connexion) en haut à droite. Ce sont vos liens d'accès aux comptes utilisateurs. Inscrivez-vous en cliquant sur le lien *Register* (Inscription). Vous serez invité à renseigner votre adresse e-mail et un mot de passe.

Les paramètres par défaut d'ASP.NET Core Identity exigent les caractéristiques suivantes pour les mots de passe :

- Une longueur minimale de six (6) caractères ;
- Au moins un (1) chiffre ;
- Au moins un (1) caractère majuscule ;

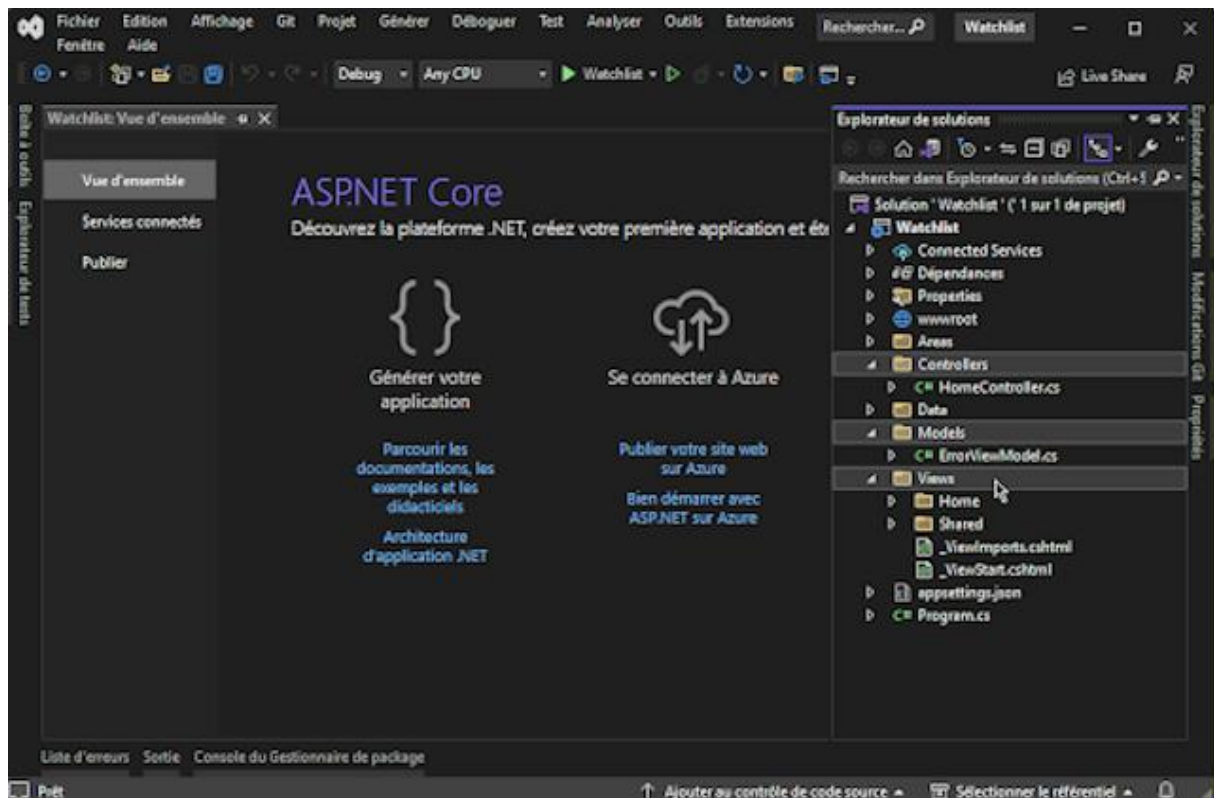
- Au moins un (1) caractère minuscule ;
- Au moins un (1) caractère non alphanumérique ;

Lorsque vous avez saisi les informations demandées, cliquez sur le bouton *Register* (Inscription). Vous verrez alors la même page que la précédente, mais les liens *Register* (Inscription) et *Login* (Connexion) seront remplacés par un message d'accueil et un lien *Logout* (Déconnexion). Vous êtes maintenant connecté en toute sécurité au système !

Si vous souhaitez tester la fonctionnalité de connexion indépendamment de l'inscription, cliquez sur *Logout* (Déconnexion). Lorsque la page s'actualise, cliquez sur le lien *Login* (Connexion) et saisissez votre adresse e-mail et votre mot de passe, puis cliquez sur le bouton *Login* (Connexion). Vous devriez voir un message indiquant que la connexion a réussi.

Découvrez l'arborescence d'un projet .NET MVC

Maintenant que vous avez créé votre application de base et vérifié que les comptes d'utilisateur sont opérationnels, voyons à quoi ressemble ce projet. Dans l'Explorateur de solutions situé sur la droite de votre IDE, vous voyez tous les dossiers et fichiers qui composent votre projet Watchlist. Examinez particulièrement les dossiers Models, Views et Controllers.



Fenêtre Explorateur de solutions dans Visual Studio

Ces dossiers représentent les composants du design pattern MVC que vous utiliserez pour créer cette application. Le dossier Models contient tous les modèles de vue (classes C#) qui seront utilisés pour représenter les données de votre base de données dans le navigateur. Le dossier Views contient toutes les pages HTML qui afficheront les données représentées par les modèles. Le dossier Controllers contient les classes de contrôleur qui traitent les données, construisent les modèles et appellent les vues sur le serveur, afin de les retourner en réponse à chaque requête.

Notez que les noms de chaque contrôleur sont les mêmes que ceux des sous-dossiers du dossier Views. Chaque sous-dossier du dossier Views correspond donc à un contrôleur. C'est ainsi que le mécanisme de routage MVC sait quelle vue doit être rendue et où la trouver. Les requêtes URL suivent le format [domaine] / [contrôleur] / [action] / {id}. Le nom du contrôleur est la première partie de l'URL après le nom de domaine. Il est suivi de l'action du contrôleur qui sera appelée. Le nom de l'action est également le nom de la page HTML du sous-dossier du dossier Views qui correspond au contrôleur. Le paramètre {id} est facultatif et est fourni lors de la récupération d'un élément spécifique de la base de données.

Ainsi, si votre requête d'URL est *https://votredomaine.com/films/create*, vous demandez à votre application d'accéder au contrôleur FilmsController et de trouver la méthode/action create. Le code de cette méthode retourne alors une page HTML dynamique rendue à l'aide de la vue nommée Create.cshtml, qui se trouve dans le dossier /Views/Films. C'est ainsi que le routage fonctionne dans .NET MVC.

J'aimerais attirer votre attention sur un autre sous-dossier du dossier Views : le *sous-dossier Shared*. Toutes les pages placées dans ce dossier peuvent être appelées à partir de n'importe quel contrôleur. Si le contrôleur ne parvient pas à trouver la page demandée dans son sous-dossier Views correspondant, il cherchera une correspondance dans le dossier *Shared*.

Ce dernier contient également deux autres types de vues importants : les dispositions et les vues partielles. Les dispositions contiennent un code qui est commun à plusieurs pages, comme le menu de navigation et d'autres éléments d'en-tête, ainsi que les pieds de page. Les pages de vue individuelles contiennent alors uniquement les éléments de code qui leur sont propres. Par exemple, si une modification doit être apportée aux menus ou aux pieds de page, elle ne doit être effectuée qu'une seule fois (dans la page de disposition). Elle est ensuite répercutée sur toutes les autres pages utilisant cette disposition.

Les vues partielles se composent quant à elles de petits extraits HTML repris sur plusieurs pages. Il peut s'agir par exemple d'un formulaire de contact. Ce formulaire est alors contenu dans la vue partielle, qui est rendue dans la page demandée par une simple ligne de code, sans répéter à chaque fois l'ensemble du code du formulaire.

Nous reviendrons plus en détail sur les dispositions et les vues partielles un peu plus loin. Maintenant, je veux vous parler des modèles Identity et de la façon dont vous les utiliserez pour travailler avec une base de données relationnelle.

Mapping objet-rationnel et contexte de la base de données

Regardez le dossier *Data* de votre projet. Vous y trouverez deux éléments : 1) un dossier *Migrations* et 2) un fichier *.cs appelé *ApplicationDbContext*. Le dossier *Migrations* contient le code généré par Entity Framework pour construire une base de données relationnelle pour votre application. Cette base de données contient toutes les tables nécessaires pour activer, créer et gérer les authentifications sécurisées des utilisateurs.

La classe *ApplicationDbContext* est une classe C# qui hérite de la classe *IdentityDbContext*. C'est elle qui contient les fonctions de base permettant de gérer les comptes d'utilisateur et votre base de données. La classe *ApplicationDbContext* est la représentation codée de votre base de données. Elle sert à interagir avec la base de données par le biais du **mapping objet-relationnel (ORM—Object Relational Mapping)**. L'ORM est un moyen de faire correspondre les entités de la base de données au code C#. Une classe peut ainsi représenter un enregistrement dans une table, et les attributs de la classe les colonnes. Nous n'entrerons pas dans les détails de l'ORM dans ce cours, mais nous y reviendrons dans une autre partie du parcours .NET. Je vais quand même vous montrer comment utiliser l'ORM pour interagir avec la base de données et la mettre à jour pour cette application.

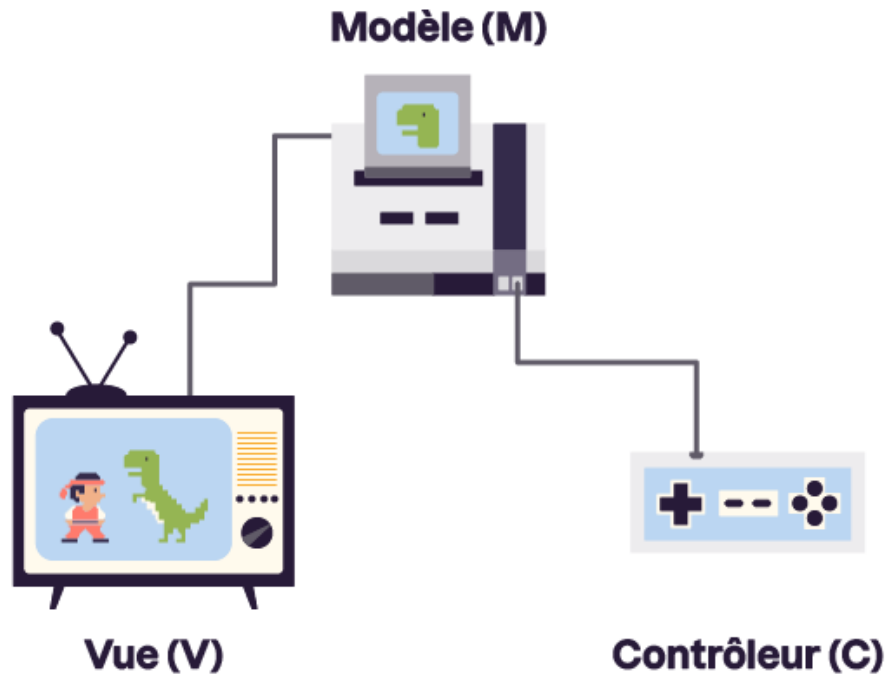
En résumé

Ce chapitre a été riche en enseignements !

- Votre compte Azure est à présent configuré.
- Vous avez réussi à créer et à exécuter votre application.
- Vous avez inscrit un nouvel utilisateur et testé la fonctionnalité de connexion.

Saisissez le principe du MVC

Le principe du MVC



Modèle-vue-contrôleur

Voyons comment appliquer le design pattern MVC en trois étapes :

1. Du concept au modèle de données
2. Du modèle de données au contrôleur
3. Du contrôleur à la vue

Étape 1 : du concept au modèle de données

Lorsque vous créez une nouvelle application .NET MVC, comme pour toute autre application, vous commencez par un concept et une liste de besoins. Les applications MVC sont également axées sur les données. Cela signifie que l'application est conçue pour fonctionner avec un type ou un ensemble spécifique de données. Par conséquent, vous commencerez toujours par travailler sur le modèle de données de l'application.

Comme nous savons précisément ce que nous voulons réaliser, il ne nous reste qu'à traduire notre concept en un modèle de données que l'application peut utiliser.

Nous voulons que notre application Watchlist nous permette de saisir les titres de films et d'autres informations liées, noter les films, et enregistrer toutes ces informations

pour les consulter par la suite. Nous souhaitons également pouvoir afficher la liste des films notés. Ils devraient peut-être être triés par note ou par date de visionnage. Nous devrions également pouvoir afficher uniquement les films qui n'ont pas encore été notés. Pour éviter que cette application ne devienne trop complexe, laissons de côté ces critères d'affichage. Nous n'allons pas créer de page distincte pour afficher tous les détails d'un film, nous ne cherchons pas à imiter Allociné !

Identification des entités (objets)

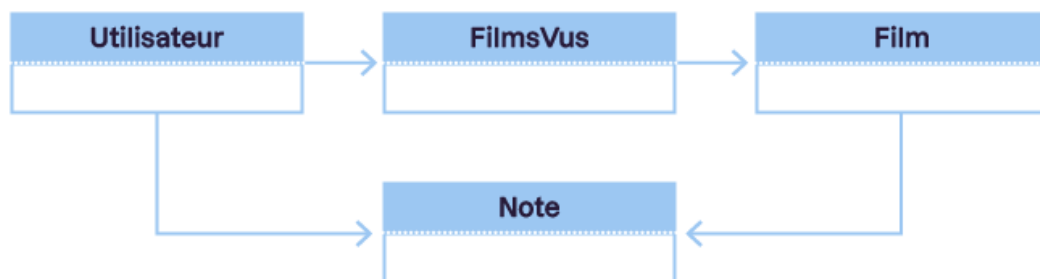
Commençons par ce qui est évident : quelles sont les principales entités de notre schéma conceptuel ? On pourrait dire qu'il y en a quatre :

- Utilisateur : notre système comporte plusieurs utilisateurs.
- Liste des films vus : chaque utilisateur possède une liste des films qu'il a vus.
- Film : chaque liste contient plusieurs films.
- Note : chaque film d'une liste est associé à une note.

Si cette application ne devait être utilisée que par un seul utilisateur, nous pourrions faire de la note un simple attribut de l'objet Film. Toutefois, l'application possédant plusieurs comptes d'utilisateur, chaque utilisateur possède sa propre liste de films et peut donc noter chaque film de sa liste. Alors, comment définir les relations entre ces entités ? Voici mes suggestions :

- Chaque utilisateur dispose d'une seule liste de films.
- Chaque liste contient plusieurs films.
- Plusieurs utilisateurs peuvent noter le même film. Ainsi, la note doit être liée à la fois à l'utilisateur et au film.

Voici un diagramme approximatif du modèle jusqu'à présent. Gardez à l'esprit qu'il n'est pas exhaustif :



Représentation actuelle du modèle

Bien. Vous avez une idée de ce à quoi le modèle pourrait ressembler, mais vous savez aussi qu'il n'est pas encore très précis ni complet. Nous allons voir dès à présent comme le lier à un contrôleur. Nous reviendrons sur le modèle dans le chapitre suivant.

Étape 2 : du modèle de données au contrôleur

Dans le modèle MVC, les contrôleurs sont des zones d'action. Ils représentent et gèrent les requêtes HTTP entrantes et les réponses sortantes. J'ai mentionné le pattern de route (domaine/contrôleur/action) du MVC. La partie contrôleur de la route (ou URL) est le nom du contrôleur, et l'action est le nom de la méthode du contrôleur qui crée la vue que l'utilisateur veut voir. Cela signifie qu'il existe une relation directe, de type one-to-one, entre un modèle et un contrôleur.

Prenons l'exemple de l'entité Film. Si vous construisez un contrôleur pour interagir avec vos objets Film, il sera probablement appelé FilmsController. Toutes les données traitées dans ce contrôleur seront liées à l'objet Film. Le contrôleur vous permettra d'ajouter, de modifier, de supprimer et de lister les films de la base de données en générant une vue (page HTML) correspondant à la requête HTTP.

Après avoir finalisé le modèle de données, vous allez générer automatiquement des contrôleurs et des vues pour chacune des classes C# qui le composent.

Étape 3 : du contrôleur à la vue

La vue est appelée par l'instruction return dans l'action du contrôleur :

```
return View();
```

Cette instruction appelle la méthode View de .NET MVC. Cette méthode recherche dans le dossier Views du projet en cours un fichier de vue dont le nom correspond à l'action et au contrôleur. Elle charge le fichier de disposition correspondant, puis lit le fichier de vue et exécute le code Razor qu'elle trouve tout en produisant le HTML brut. Elle construit ainsi un fichier HTML dynamique à partir des résultats et le retourne au navigateur en réponse à la requête originale.

Nous allons créer nos vues à l'aide de modèles génériques grâce à la génération automatique. Cependant, elles devront être personnalisées et affinées pour fonctionner efficacement.

Avantages des dispositions

Comme mentionné précédemment, les dispositions dans .NET MVC sont tout simplement géniales !

Elles vous permettent de diviser les pages HTML en blocs réutilisables. Tous les éléments communs (ceux qui apparaissent sur plusieurs pages, comme les menus, les barres de navigation, les en-têtes et pieds de page, les fenêtres de discussion, etc.)

peuvent être placés dans un fichier de disposition, tandis que les éléments qui sont exclusifs à une page donnée constituent le fichier de vue spécifique de la page. Les deux fichiers sont ensuite fusionnés en réponse à une requête HTTP et livrés sous la forme d'un seul fichier HTML dynamique.

Dans un projet .NET MVC, il est possible de créer autant de dispositions personnalisées que nécessaire et de les affecter à des vues spécifiques. Vous pouvez même modifier la disposition en fonction de l'utilisateur qui demande la page. Cette personnalisation peut intervenir en fonction du rôle de l'utilisateur, ou même en fonction de l'utilisateur lui-même, si vous avez besoin de ce degré de précision.

Plus tard dans ce cours, je vous montrerai comment tirer le meilleur parti de vos dispositions et accélérer votre développement .NET MVC. Mais avant cela, revenons à la conception de notre modèle. Nous devons le compléter avant de pouvoir faire quoi que ce soit avec les contrôleurs, les vues et les dispositions.

En résumé

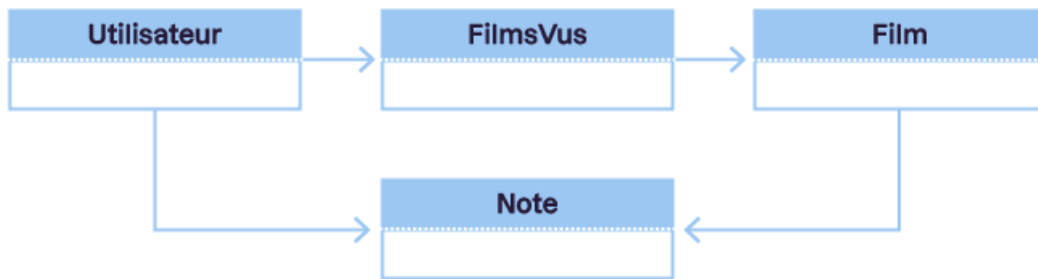
- Dans ce chapitre, nous avons conçu une ébauche de notre modèle de données basée sur le concept de notre application Watchlist.
- Nous avons identifié les entités que vous pourriez avoir besoin de créer pour former le modèle et avons vu comment créer des contrôleurs MVC pour gérer ces entités.
- Nous avons également évoqué la relation directe ou one-to-one qui existe entre les contrôleurs et les modèles, ainsi que la relation entre les actions des contrôleurs (méthodes C#) et les vues MVC (pages .cshtml).
- Enfin, nous avons abordé l'importance de l'utilisation des dispositions dans les applications .NET pour simplifier le développement et la maintenance future des vues.

Construisez le modèle de données de votre application MVC

Concevez un modèle de données

Revenons à notre modèle de base et réfléchissons à la manière de l'affiner et de l'améliorer.

Voici ce avec quoi nous avons commencé :



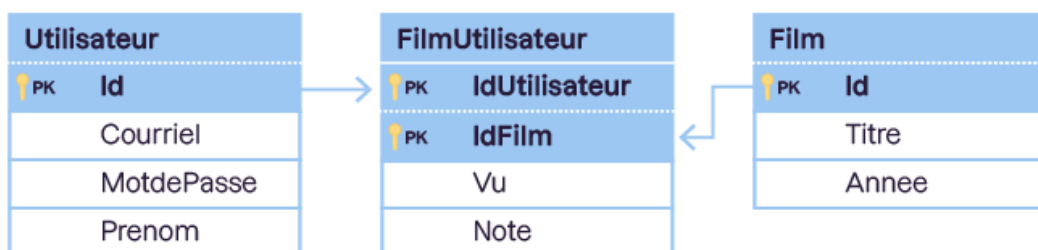
Notre modèle de départ

Commençons à ajouter les propriétés logiquement associées à chaque entité de données et voyons si le modèle tient la route. Pour l'entité Utilisateur, nous n'avons pas besoin de grand-chose pour le moment. Il nous faut juste une adresse e-mail et un mot de passe pour la connexion. Puisque nous utilisons Entity Framework et Identity pour gérer les comptes d'utilisateur, ces propriétés seront déjà présentes dans notre objet Utilisateur, accompagnées d'une propriété Id, qui identifie de manière unique chaque objet. La propriété Id sert également de **clé primaire** dans la table correspondante de la base de données.

Chacun des autres objets doit lui aussi avoir une propriété Id qui servira de clé primaire.

D'après la liste des besoins établie pour notre application, chaque utilisateur n'aura qu'une seule liste contenant plusieurs films. Par conséquent, cette liste a-t-elle besoin de ses propres propriétés ? Nous savons qu'elle appartient à un utilisateur et qu'elle contient des films, mais nous n'avons pas précisé si elle aurait besoin de propriétés supplémentaires. Par conséquent, nous pourrions la considérer comme une propriété de l'entité Utilisateur, une liste d'objets Film.

Voici un autre design possible :



Un autre design plausible

Dans ce schéma, nous avons ajouté des propriétés aux entités et supprimé l'entité Liste de films, en la plaçant dans l'entité FilmUtilisateur. Utilisez les propriétés Id des entités Utilisateur et Film comme **clés primaires composites** pour l'entité FilmUtilisateur. Cela permet d'identifier de manière unique une entité FilmUtilisateur comme appartenant à un utilisateur et à un film précis. Ce type de relation est également connu sous le nom de relation **many-to-many (plusieurs à plusieurs)** dans le vocabulaire des bases de données, ce qui signifie que plusieurs utilisateurs peuvent regarder et noter le même film et que plusieurs films peuvent appartenir au même utilisateur. Comment coder ce modèle ?

Codez les classes du modèle

Nous avons deux classes d'entités primaires : Film et Utilisateur. Nous disposons également d'une autre classe d'entité, FilmUtilisateur, qui représente la relation many-to-many entre les deux autres entités de la base de données.

Chaque classe d'entité doit avoir une clé primaire, généralement représentée par une propriété nommée Id ou un nom approchant. Commençons donc le codage par la classe Film. Tout d'abord, cette classe a besoin d'une propriété Id de type entier, d'un titre de type chaîne et d'une année de type entier. Pour ajouter cette classe, cliquez avec le bouton droit de la souris sur le dossier Data, puis sélectionnez *Ajouter > Classe*. Nommez le fichier de classe *Film.cs* puis cliquez sur le bouton *Ajouter*. Ajoutez ensuite les propriétés nécessaires à la classe Film, comme indiqué ci-dessous :

```
public class Film
{
    public int Id { get; set; }

    public string Titre { get; set; }

    public int Annee { get; set; }
}
```

Passons à l'entité Utilisateur. Celle-ci est un peu plus délicate. Puisque le projet comprend des comptes d'utilisateur individuels, des objets de compte d'utilisateur sont déjà intégrés dans le projet grâce à ASP.NET Identity. Ils sont créés à l'aide d'une classe appelée IdentityUser, qui se trouve dans le package Microsoft.AspNetCore.Identity. La classe IdentityUser contient toutes les informations importantes permettant de sécuriser les connexions des utilisateurs, telles que le nom d'utilisateur, l'adresse e-mail, le mot de passe, le numéro de téléphone, etc. Cependant, elle ne contient pas de champs pour les informations personnelles telles que le nom ou le prénom, l'adresse, la ville, l'État, etc. Heureusement, il s'agit d'une classe que vous pouvez **étendre** à une

autre classe de votre conception qui peut contenir tous les champs supplémentaires que vous souhaitez ajouter.

Il y a un autre point que je tiens à mentionner avant d'ajouter cette classe. ASP.NET Identity possède déjà un objet appelé User qui est réservé à l'accès au compte de l'utilisateur connecté. Par conséquent, il ne faut pas nommer votre classe d'entité utilisateur User. Appelons-la plutôt Utilisateur. Ajoutez donc une autre classe au dossier Data, appelez-la Utilisateur, puis intégrez-y les éléments suivants :

1. La classe doit étendre Microsoft.AspNetCore.Identity.IdentityUser.
2. Elle doit disposer d'un constructeur qui appelle le constructeur de l'objet hérité. Si vous n'avez jamais utilisé cette syntaxe auparavant, voici à quoi elle doit ressembler :

```
public Utilisateur() : base()
{
}
```

3. Elle doit contenir au moins une propriété Prenom de type chaîne de caractères.

Nous allons revenir à notre classe Utilisateur dans un moment. Pour l'instant, intéressons-nous à la classe FilmUtilisateur. Ajoutez cette classe au dossier Data, puis intégrez-lui des propriétés, comme indiqué ci-dessous :

```
public class FilmUtilisateur
{
    public string IdUtilisateur { get; set; }
    public int IdFilm { get; set; }
    public bool Vu { get; set; }
    public int Note { get; set; }
}
```

La classe FilmUtilisateur possède la clé primaire composite dont nous avons fait mention plus haut. Cette clé est composée de deux propriétés : IdUtilisateur et IdFilm. La propriété IdUtilisateur correspond à la propriété Id de la classe Utilisateur, qu'elle a héritée de IdentityUser. La propriété IdFilm correspond à la propriété Id de la classe Film. Pour compléter la classe FilmUtilisateur, vous avez besoin de deux autres propriétés. Ces propriétés utilisent le mot clé **virtual** et représentent la relation entre l'objet FilmUtilisateur et les objets Utilisateur et Film.

```
public virtual Utilisateur User { get; set; }
```

```
public virtual Film Film { get; set; }
```

Les classes Film et FilmUtilisateur étant maintenant terminées, reprenons la classe Utilisateur. Nous devons ajouter une propriété virtuelle pour représenter la liste des films de l'utilisateur. Il s'agit d'une collection ou d'une liste d'objets FilmUtilisateur, et plus précisément, de tous les objets FilmUtilisateur contenant l'identifiant de l'utilisateur. Mettons à jour notre classe Utilisateur pour inclure cette propriété et initialisons-la dans le constructeur, comme indiqué ci-dessous :

```
public class Utilisateur : Microsoft.AspNetCore.Identity.IdentityUser
{
    public Utilisateur() : base()
    {
        this.ListeFilms = new HashSet<FilmUtilisateur>();
    }
}
```

```
public string Prenom { get; set; }
```

```
public virtual ICollection<FilmUtilisateur> ListeFilms { get; set; }
}
```

Le modèle de données de notre projet est maintenant achevé. Compilez le code en générant votre projet (choisissez *Générer > Générer la solution* ou appuyez sur Ctrl-Maj-B). S'il y a des erreurs dans votre code, corrigez-les et recompilez le projet. Recommencez jusqu'à ce qu'il n'y ait plus d'erreurs de compilation.

Vous avez maintenant un modèle de données complet pour votre application Watchlist, mais pas de base de données qui reflète ce nouveau modèle. Dans le chapitre suivant, nous allons mettre à jour la base de données pour qu'elle corresponde à notre code.

En résumé

- Dans ce chapitre, nous avons affiné le modèle de données de l'application Watchlist, en nous éloignant un peu du concept initial. Nous avons peaufiné une conception qui permet à chaque utilisateur de créer sa propre liste de films. Ils peuvent ainsi donner une note individuelle à leurs films, tout en étant en mesure de voir les notes moyennes données par les autres utilisateurs.
- Nous avons codé l'ensemble du modèle sous forme de classes C# et sommes maintenant prêts à transposer ce modèle dans une base de données relationnelle.

Modifiez le contexte DbContext

Vous avez déjà effectué des migrations code first, mais nous devons encore faire quelques ajouts importants à la classe ApplicationDbContext. Tout d'abord, ajoutez les objets DbSet pour vos classes Film et FilmUtilisateur. Ajoutez les lignes suivantes à la classe ApplicationDbContext, juste après le constructeur :

```
public DbSet<Film> Films { get; set; }  
  
public DbSet<FilmUtilisateur> FilmsUtilisateur { get; set; }
```

Remarquez que nous n'avons pas ajouté d'objet DbSet pour la classe Utilisateur. Cela est dû au fait qu'il existe déjà une tableAspNetUsers dans la base de données. Puisque nous avons créé la classe Utilisateur qui hérite de la classe IdentityUser, tout ce que vous ajoutez à Utilisateur sera automatiquement traduit dans la table AspNetUsers de la base de données.

La dernière modification est un peu plus délicate. Dans la version actuelle d'ASP.NET Core, il n'est pas possible de migrer automatiquement des objets ayant des relations many-to-many. Heureusement, ASP.NET inclut l'**API Fluent**. Cette API vous permet de donner à Entity Framework des instructions précises pour créer ou modifier la base de données. Pour ce faire, vous devez remplacer une méthode de votre classe ApplicationDbContext, appelée OnModelCreating, qui est héritée de IdentityDbContext. Vous pouvez le faire n'importe où en dessous du constructeur, dans la classe ApplicationDbContext.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
    base.OnModelCreating(modelBuilder);  
    modelBuilder.Entity<FilmUtilisateur>()  
        .HasKey(t => new { t.IdUtilisateur, t.IdFilm});  
}
```

Recompilez maintenant le projet et corrigez toute nouvelle erreur.

Mettez à jour ASP.NET Identity

Par défaut, ASP.NET Identity utilise l'objet IdentityUser pour tous les comptes d'utilisateur. Nous avons créé la classe Utilisateur pour étendre la classe IdentityUser et pouvoir donner des informations supplémentaires sur les comptes d'utilisateur. Cependant, puisque nous voulons que la classe Utilisateur soit la base des objets Identity, nous devons encore un peu modifier le code. Les deux fichiers à modifier sont *Startup.cs* et *_LoginPartial.cshtml*.

Ouvrez d'abord le fichier *Startup.cs*. Il se trouve à la racine de votre projet. Cherchez la méthode *ConfigureServices*. Vers la ligne 41, vous devriez voir le code suivant :

```
services.AddDefaultIdentity<IdentityUser>()

    .AddDefaultUI(UIFramework.Bootstrap4)

    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Remplacez ce segment de code par celui-ci :

```
services.AddIdentity<Utilisateur, IdentityRole>(options =>
{
    options.User.RequireUniqueEmail = false;
})

    .AddDefaultUI(UIFramework.Bootstrap4)

    .AddEntityFrameworkStores<ApplicationDbContext>()

    .AddDefaultTokenProviders();
```

Remarquez le changement de *IdentityUser* à *Utilisateur* dans la configuration du service *Identity*. Cette modification permettra à .NET Identity d'utiliser votre classe *Utilisateur* pour tous les processus et services *Identity*, comme la configuration de *userManager* et *SignInManager*, que nous verrons plus tard dans ce cours.

La deuxième modification doit intervenir dans *Views > Shared > _LoginPartial.cshtml*. Les trois premières lignes de code de ce fichier sont les suivantes :

```
@using Microsoft.AspNetCore.Identity

@Inject SignInManager<IdentityUser> SignInManager

@Inject UserManager<IdentityUser> UserManager
```

Vous devez ajouter une autre instruction *using* pour l'espace de noms *Data* de l'application, puis remplacer les références à *IdentityUser* par *Utilisateur*, comme indiqué ci-dessous :

```
@using Microsoft.AspNetCore.Identity

@using Watchlist.Data

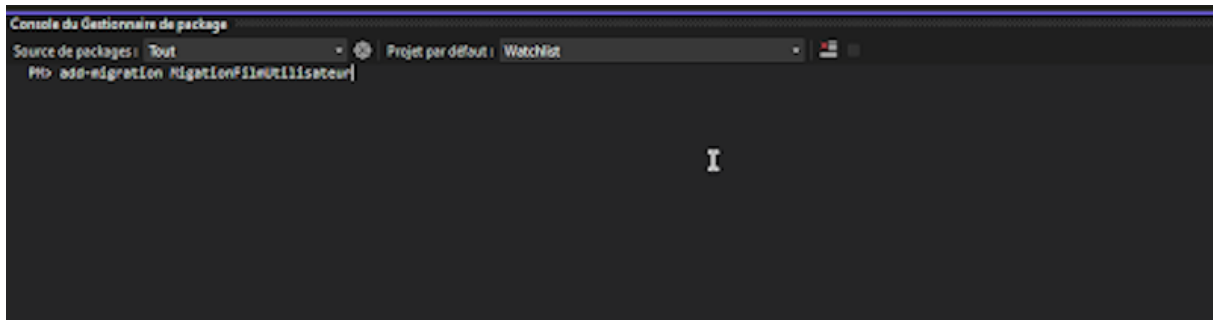
@Inject SignInManager<Utilisateur> SignInManager

@Inject UserManager<Utilisateur> UserManager
```

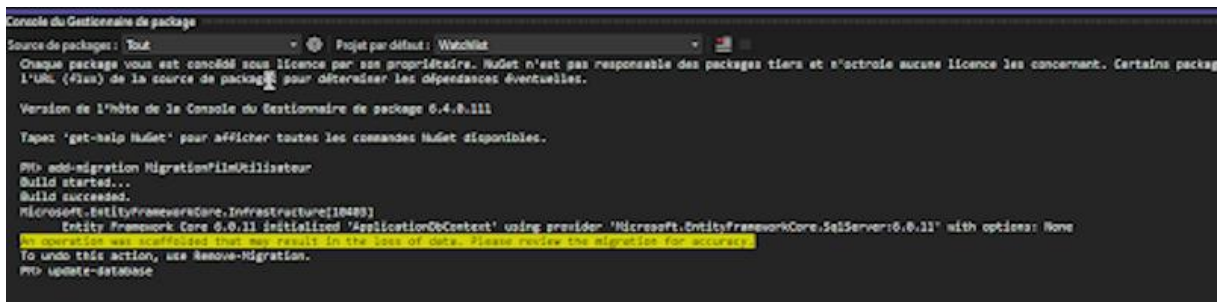
Enregistrez vos modifications et recompilez le projet en corrigeant les éventuelles erreurs de syntaxe.

Effectuez la migration

Maintenant que le contexte de la base de données et le service Identity ont été mis à jour pour utiliser votre classe Utilisateur, le modèle est achevé et peut être migré. Ajoutez une nouvelle migration et nommez-la comme vous le souhaitez, mais n'oubliez pas qu'il est recommandé de lui attribuer un nom représentatif de l'action réalisée ou de son ordre dans la séquence. Vous pouvez l'appeler SecondeMigration ou MigrationFilmUtilisateur, par exemple :



Ajout d'une nouvelle migration dans le Gestionnaire de package



Mise à jour de la base de données dans le Gestionnaire de package

Votre base de données est maintenant à jour et correspond au modèle de données que vous venez de créer, mais vous ne pouvez pas encore faire grand-chose avec. La prochaine étape sera donc de générer automatiquement des contrôleurs et des vues à partir du modèle.

En résumé

Ce chapitre portait sur le paramétrage de la base de données afin de lui permettre de recevoir les données de notre application :

- Nous avons mis à jour le contexte de la base de données pour qu'il corresponde au modèle de données global.
- Ensuite, nous avons effectué une migration code first, qui a créé de nouvelles tables dans la base de données correspondant aux classes C# de notre modèle.

Générez automatiquement les couches de votre application MVC

Qu'est-ce que la génération automatique ?

La génération automatique d'ASP.NET a été introduite dans Visual Studio 2013 et reste un élément important de la plateforme de développement .NET Core.

Il s'agit d'un framework de génération de code automatique qui vous permet d'ajouter rapidement du code modélisé qui interagit avec vos modèles de données. Son utilisation pour générer les contrôleurs et vues MVC de vos modèles permet d'accélérer le développement de toutes les opérations de données standard de votre projet, ainsi que des pages HTML/Razor servant à visualiser ces données.

Utilisez la génération automatique avec Entity Framework

Avant de générer automatiquement vos contrôleurs et vos vues, vous devez créer une classe supplémentaire. Elle ne représente pas une entité spécifique de la base de données, mais plutôt un mélange d'éléments provenant de différentes entités. C'est ce qu'on appelle un **modèle d'affichage**, et vous en avez besoin pour configurer correctement l'affichage de votre liste de films. La liste de films est une collection d'identifiants d'utilisateurs et de films dans la table FilmUtilisateur, et vous devez la rendre lisible et facile à utiliser. C'est là que le modèle d'affichage entre en jeu.

Ajoutez les modèles d'affichage

Ajoutez une nouvelle classe appelée *ModeleVueFilm* dans le dossier Models. Modifiez ensuite la classe pour ajouter l'identifiant du film, son titre, son année, sa présence ou absence dans la liste de l'utilisateur, son statut de visionnage par l'utilisateur et sa note (le cas échéant).

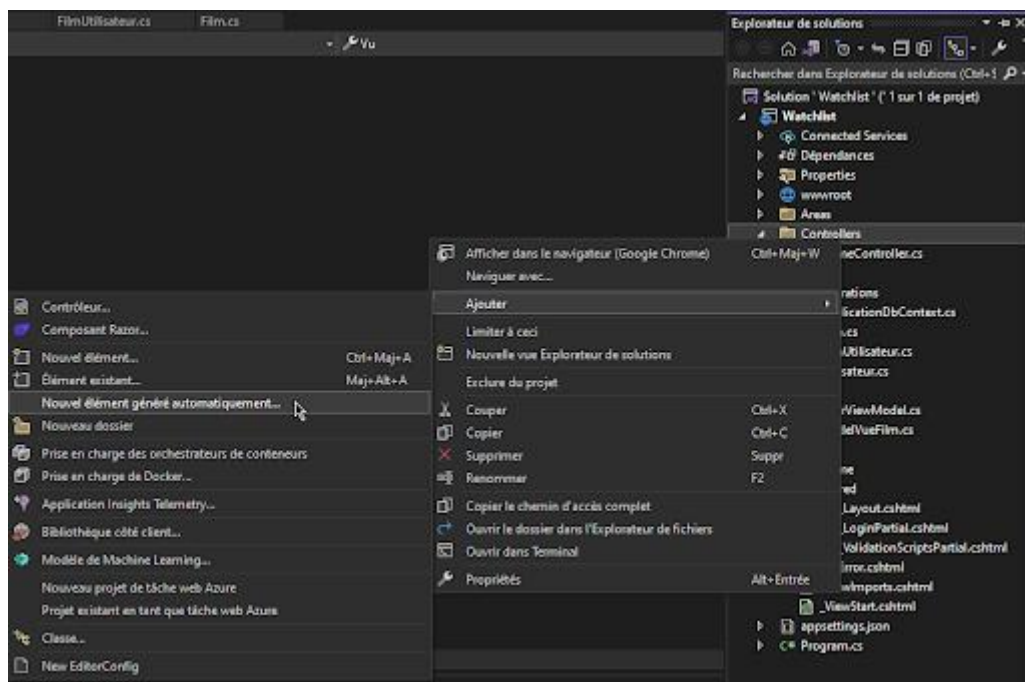
```
public class ModeleVueFilm
{
    public int IdFilm { get; set; }
    public string Titre { get; set; }
    public int Annee { get; set; }
    public bool PresentDansListe { get; set; }
    public bool Vu { get; set; }
    public int? Note { get; set; }
}
```

Générez automatiquement une entité de modèle

Nous allons maintenant générer automatiquement la classe Film. Vous utiliserez souvent cette méthode lors de la création d'applications MVC : veillez à bien comprendre le processus.

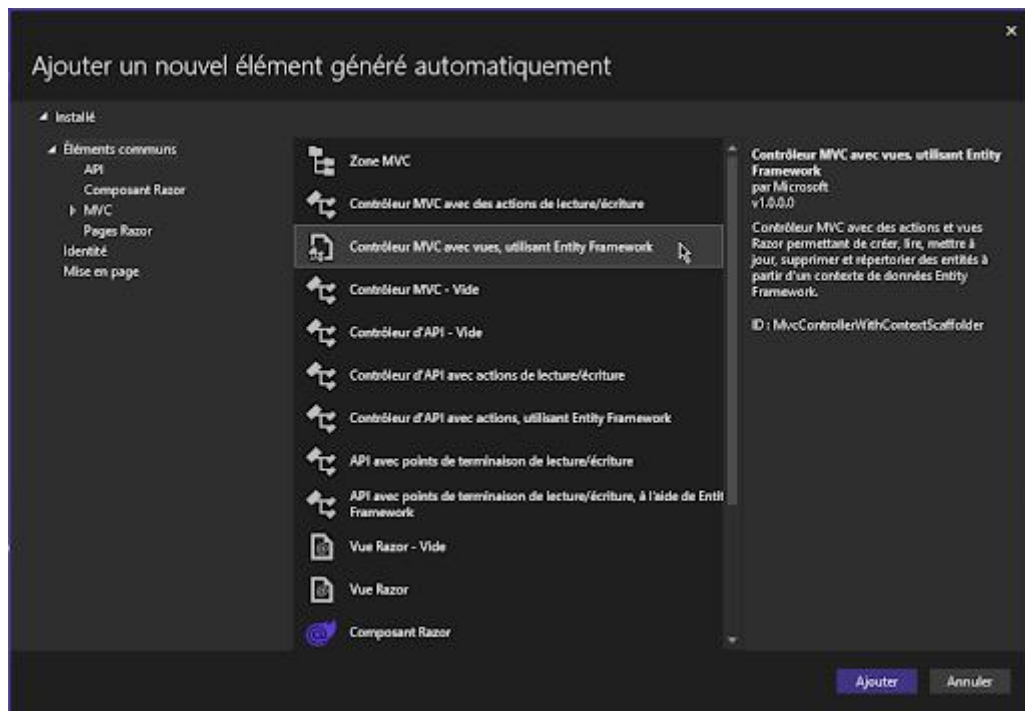
La version de VisualStudio utilisée dans les captures d'écran peut être différente de celle que vous avez téléchargée. Par conséquent, il est possible que votre interface ne soit pas totalement identique à celle présentée dans le cours.

Pour générer automatiquement des contrôleurs et des vues pour une entité de données, cliquez avec le bouton droit sur le dossier Controllers, puis choisissez *Ajouter > Nouvel élément généré automatiquement*.



Dans l'Explorateur de solutions, clic droit sur Controllers, puis Ajouter > Nouvel élément généré automatiquement.

Vous pouvez choisir parmi de nombreuses options, mais ce qui nous intéresse c'est la génération automatique d'un *contrôleur MVC avec vues, utilisant Entity Framework*. Sélectionnez cette option, puis cliquez sur le bouton *Ajouter*.



La fenêtre Ajouter un nouvel élément généré automatiquement présente de nombreuses options.

Vous entrez à présent dans les détails du processus de génération automatique. Sélectionnez la classe du modèle à partir de laquelle vous allez générer automatiquement le contrôleur et les vues (il s'agit de la classe Film), puis sélectionnez votre classe du contexte de la base de données (ApplicationDbContext) et définissez le nom du contrôleur (FilmsController). Assurez-vous que les trois cases sont cochées : nous voulons générer des vues, référencer les bibliothèques de scripts telles que jQuery et Bootstrap, et utiliser une page de disposition (dans ce cas, la page de disposition par défaut convient).

Ajouter Contrôleur MVC avec vues, utilisant Entity Framework

Classe de modèle: Film (Watchlist.Data)

Classe de contexte de données: ApplicationDbContext (Watchlist.Data) +

Vues

- ☒ Générer des vues
- ☒ Bibliothèques de scripts de référence
- ☒ Utiliser une page de disposition

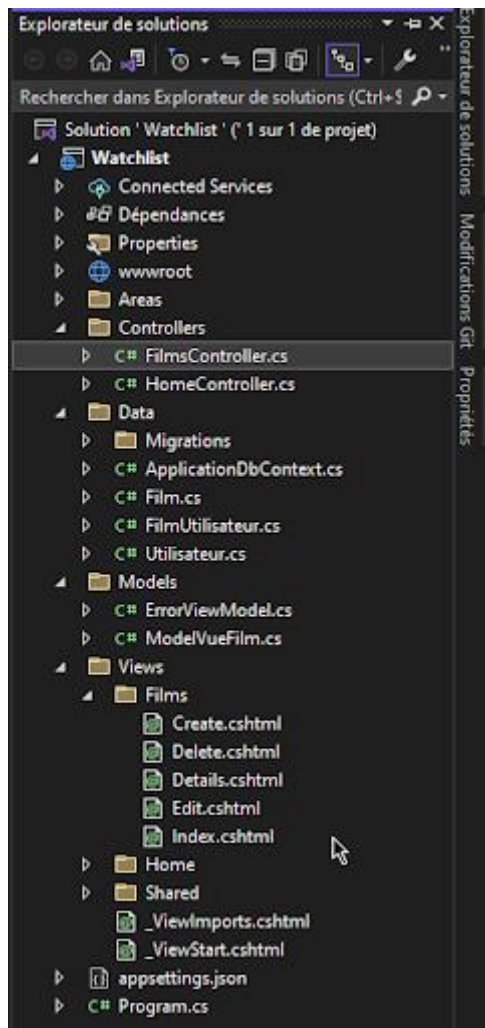
(Laissez vide s'il est défini dans un fichier Razor _viewstart)

Nom du contrôleur: FilmsController

Ajouter Annuler

Vérifiez que les trois vues sont cochées.

Une fois le processus terminé, vous verrez apparaître un nouveau fichier FilmsController.cs dans le dossier Controllers et un nouveau dossier Films dans le dossier Views de votre projet.



Le nouveau fichier FilmsController.cs et le nouveau dossier Films

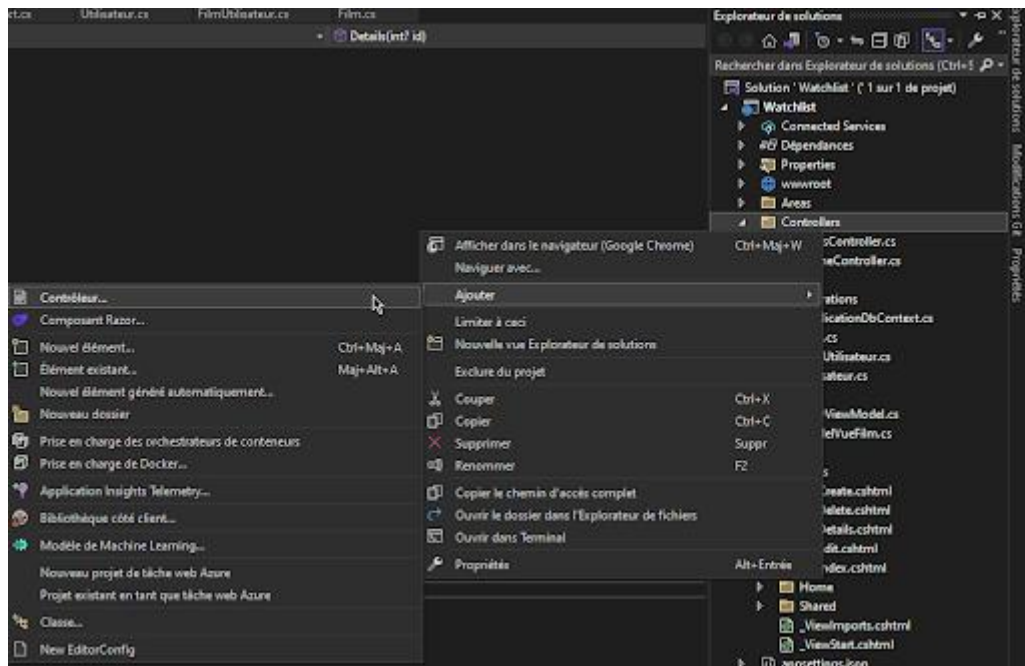
Ces ajouts sont déjà fonctionnels, nous les testerons plus tard dans le cours. Pour l'instant, nous devons générer automatiquement d'autres éléments.

Générez automatiquement des composants individuels

Vous avez déjà généré automatiquement un contrôleur complet avec des vues utilisant Entity Framework pour la classe Film. Je vais maintenant vous montrer comment générer automatiquement des éléments à plus petite échelle, par exemple pour ajouter un ou deux éléments, et non un ensemble complet d'opérations CRUD.

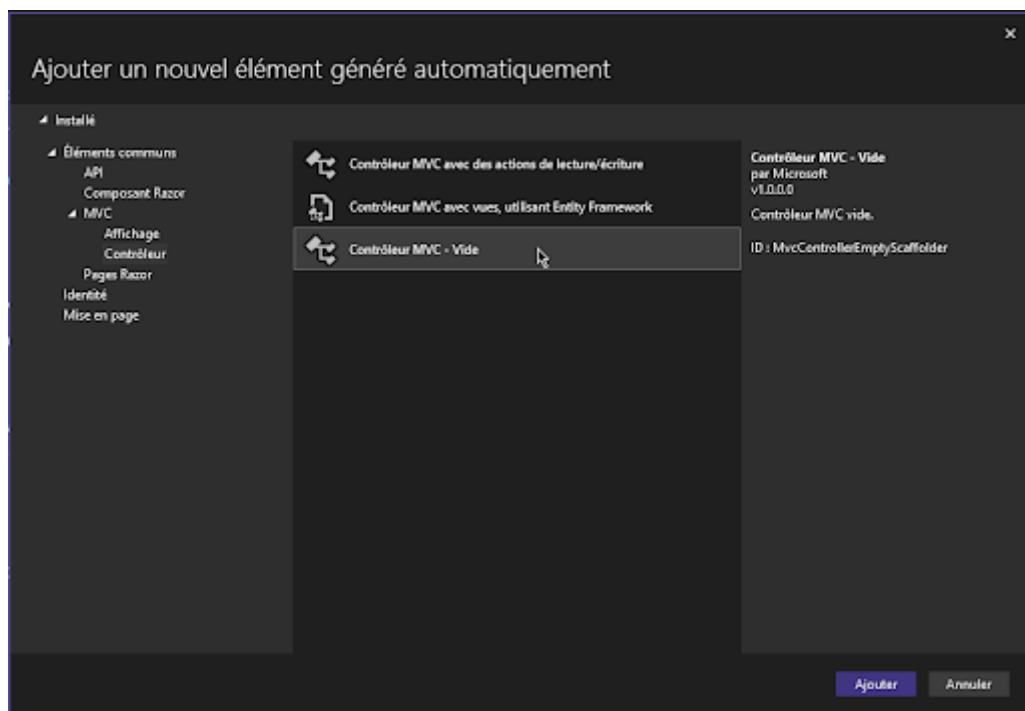
Ajoutez un contrôleur Liste de films

Ajoutons un nouveau contrôleur que vous pouvez utiliser pour créer et afficher votre liste de films. Il s'agira d'un code personnalisé et non pas d'un code généré à partir d'un modèle, vous devez donc commencer par créer un **contrôleur vide**. Pour ce faire, faites un clic droit sur le dossier Controllers, puis choisissez *Ajouter > Contrôleur*.



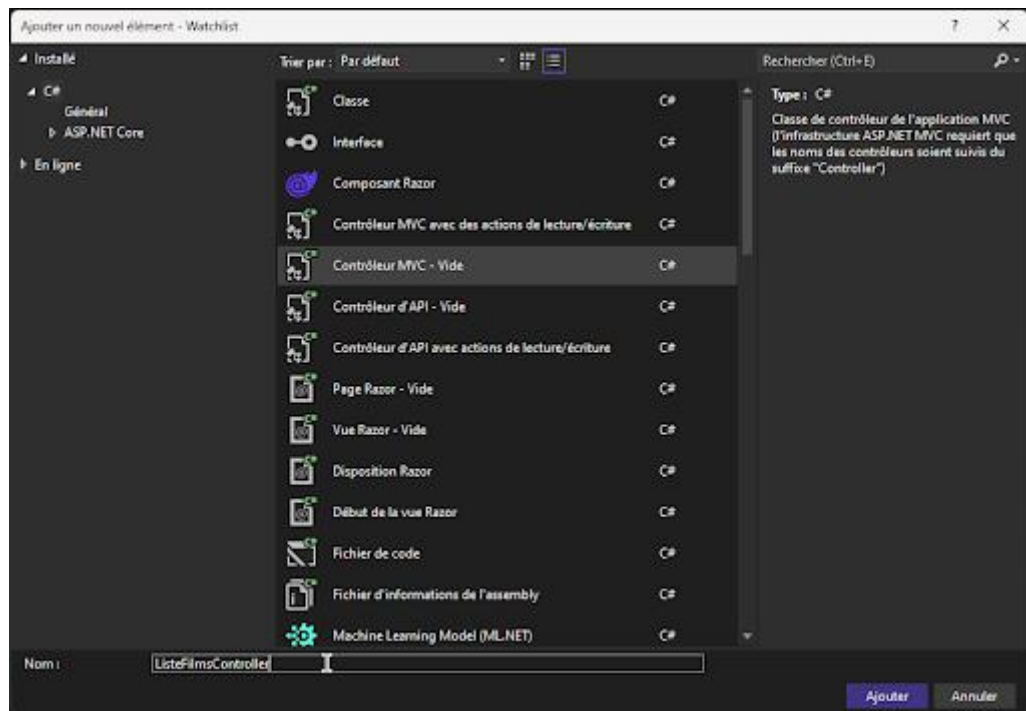
Faites un clic droit sur le dossier Controllers et choisissez Ajouter > Contrôleur.

La fenêtre suivante est similaire à celle qui présente les options de génération automatique que nous avons vue dans la section précédente, mais comporte moins d'éléments. C'est le premier qui nous intéresse : *Contrôleur MVC — Vide*.



Dans la fenêtre Ajouter un nouvel élément généré automatiquement, choisissez Contrôleur MVC — Vide.

Il est maintenant temps de nommer le nouveau contrôleur. Appelons-le *ListeFilmsController*.

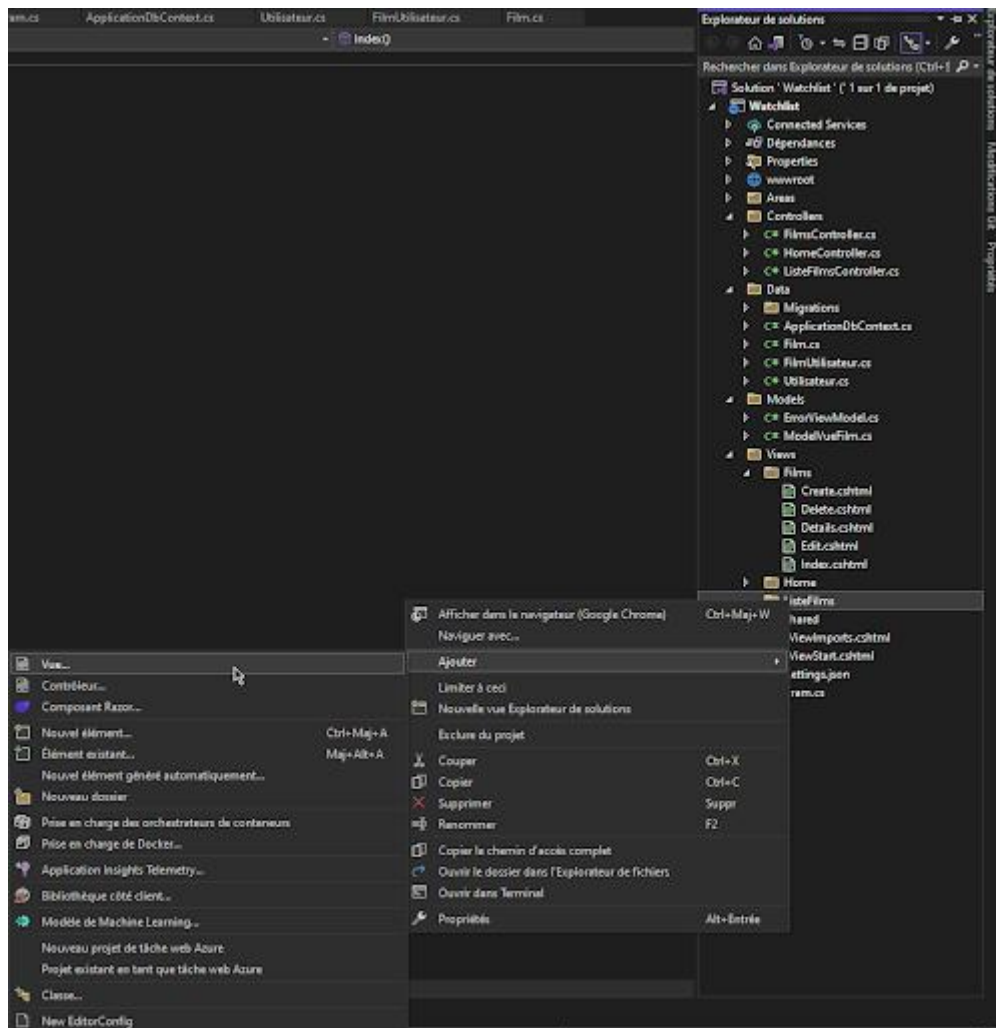


Nommez le contrôleur.

Ajoutez une vue par défaut

Après avoir ajouté une classe `ListeFilmsController` vide, vous devez ajouter une vue unique pour afficher la liste de films.

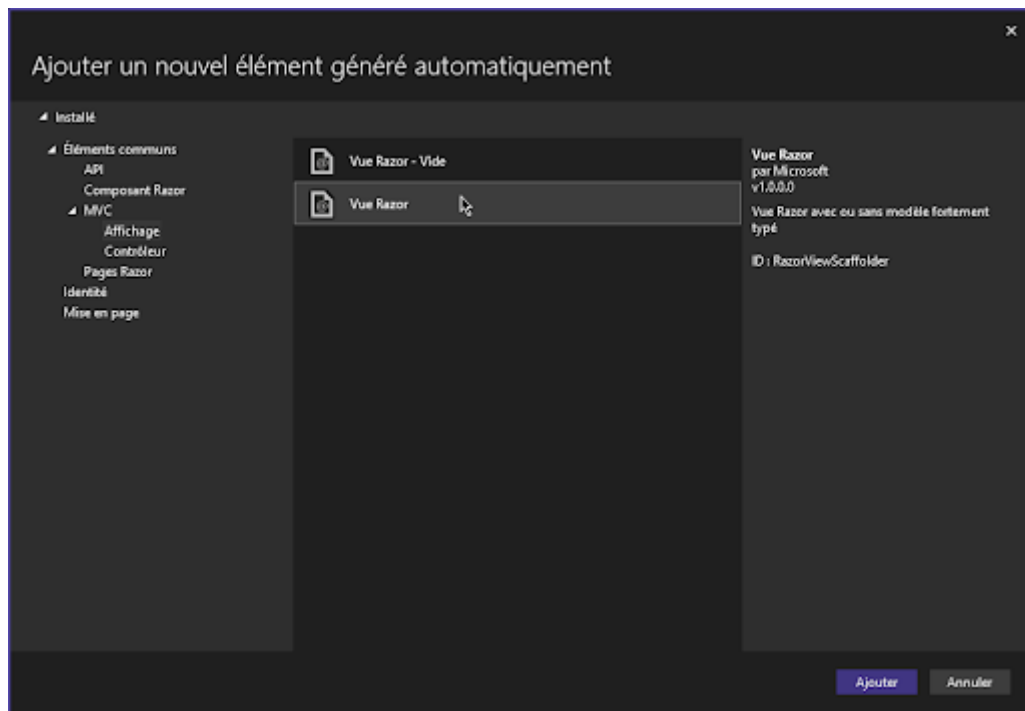
Ajoutez un nouveau sous-dossier dans le dossier `Views` appelé *ListeFilms*. Faites un clic droit sur le nouveau dossier `ListeFilms` et choisissez *Ajouter > Vue*.



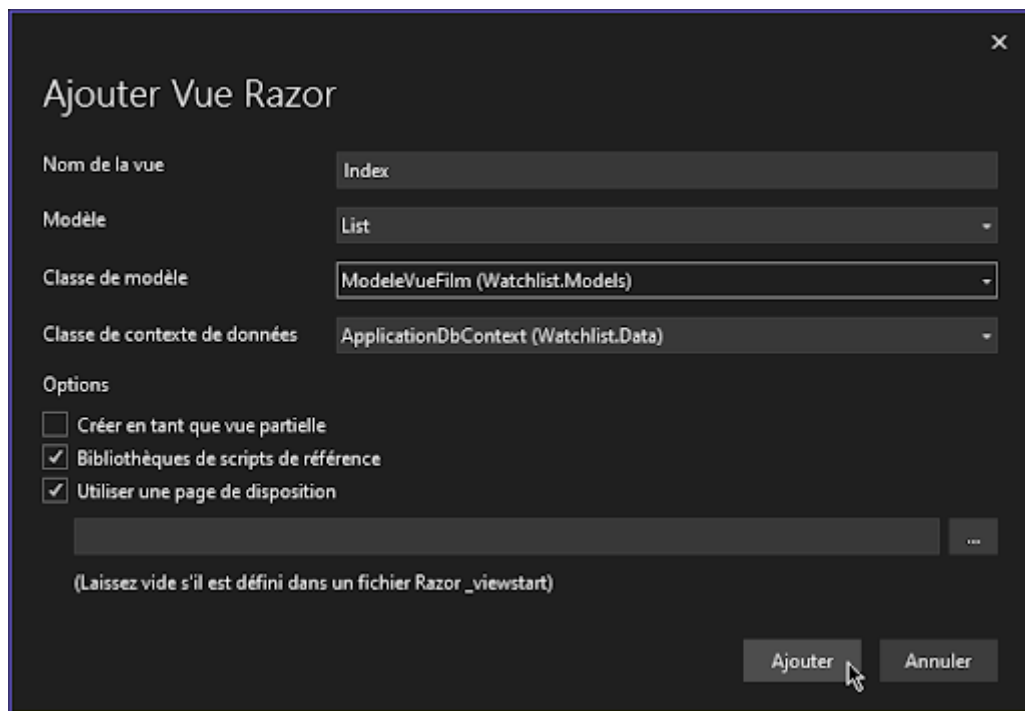
Ajoutez le sous-dossier ListeFilms. Faites un clic droit et choisissez Ajouter > Vue.

C'est ici que vous pouvez définir les types de vues que vous souhaitez ajouter au projet. Vous savez que la liste de films est une liste qui contient des titres de films et des informations supplémentaires, comme des notes, mais c'est toujours et avant tout une liste d'éléments. Par conséquent, vous devez créer une vue en liste.

Nommez cette nouvelle page *Index* afin qu'elle s'affiche par défaut chaque fois que le contrôleur ListeFilmsController est appelé. Pour le type de modèle, sélectionnez *Liste*. Pour la classe de modèle, sélectionnez la classe *ModeleVueFilm* que vous avez créée précédemment. Ensuite, supprimez tout ce qui est présent dans la liste déroulante Classe du contexte de données. On ne veut pas qu'elle soit liée directement à la base de données, car les données se trouveront dans un modèle de vue et non dans un modèle de données (base de données). Enfin, assurez-vous que la case à cocher *Créer en tant que vue partielle* n'est **pas** cochée. En revanche, cochez les cases *Bibliothèques de scripts de référence* et *Utiliser une page de disposition*, puis cliquez sur *Ajouter*.



L'option Ajouter un nouvel élément généré automatiquement



L'option Créer en tant que vue partielle n'est pas cochée, mais les options Bibliothèques de scripts de référence et Utiliser une page de disposition le sont.

Vous avez maintenant un nouveau contrôleur et une nouvelle vue pour gérer la création et l'affichage de la liste des films d'un utilisateur. Mais aucun des deux n'est fonctionnel pour le moment. Vous devez créer la liste de films, ce qui signifie ajouter du code personnalisé au contrôleur ListeFilmsController.

Modifiez le contrôleur `ListeFilmsController`

Le contrôleur `ListeFilmsController` est vide. La seule chose qu'il contient est une méthode appelée `Index`, qui ouvrira la vue `Index.cshtml` que vous venez d'ajouter. Pour remplir cette vue avec les données de la liste de films, vous devez ajouter quelques éléments à ce contrôleur.

Récupérez le contexte de la base de données

Tout d'abord, vous devez accéder à la base de données via l'objet de contexte de base de données, `ApplicationDbContext`. ASP.NET Core utilise l'injection de dépendances pour fournir des instances des objets importants à utiliser dans les contrôleurs. Ainsi, pour avoir accès au contexte de la base de données, vous devez le demander à .NET Core en l'ajoutant comme paramètre dans le constructeur du contrôleur `ListeFilmsController`.

Vous devrez ajouter quelques déclarations `using` au début de votre classe `ListeFilmsController` :

```
using Microsoft.AspNetCore.Identity;
```

```
using Watchlist.Data;
```

```
using Watchlist.Models;
```

Elles permettent d'accéder aux données et aux objets du modèle, ainsi qu'à .NET Core Identity, dont vous aurez besoin pour récupérer les films de l'utilisateur connecté.

Ensuite, pour injecter le contexte de la base de données dans votre contrôleur `ListeFilmsController`, ajoutez le code suivant au-dessus de la classe `ListeFilmsController` :

```
private readonly ApplicationDbContext _contexte;
```

```
public ListeFilmsController(ApplicationDbContext contexte)
```

```
{
```

```
    _contexte = contexte;
```

```
}
```

Dans l'exemple ci-dessus, une variable privée en `readonly` contient une instance du contexte de la base de données pour ce contrôleur. Affectez-le à l'objet de contexte qui est injecté dans le constructeur.

Cela vous permettra de récupérer les films de la liste de l'utilisateur. Vous devez être capable d'identifier l'utilisateur, puisque les films sont associés à son identifiant unique. Par conséquent, vous devez récupérer l'identifiant de l'utilisateur connecté pour obtenir ses films et générer la liste.

Récupérez l'utilisateur connecté

Pour obtenir l'utilisateur connecté, appelez le service UserManager de .NET Identity, qui fournit toutes les méthodes dont vous avez besoin pour obtenir les informations sur l'utilisateur. Pour avoir accès au service UserManager, injectez-le dans le constructeur, comme vous l'avez fait pour le contexte de la base de données.

```
public class ListeFilmsController : Controller
{
    private readonly ApplicationDbContext _contexte;
    private readonly UserManager<Utilisateur> _gestionnaire;

    public ListeFilmsController(ApplicationDbContext contexte,
        UserManager<Utilisateur> gestionnaire)
    {
        _contexte = contexte;
        _gestionnaire = gestionnaire;
    }
}
```

Le service UserManager étant maintenant disponible, vous pouvez écrire du code pour récupérer les informations de l'utilisateur. Une bonne pratique consiste à ajouter une méthode privée dans le contrôleur qui appelle le service, puis une requête HTTP GET publique pour récupérer l'identifiant de l'utilisateur.

```
public class ListeFilmsController : Controller
{
    private readonly ApplicationDbContext _contexte;
    private readonly UserManager<Utilisateur> _gestionnaire;

    public ListeFilmsController(ApplicationDbContext contexte,
```

```

    UserManager<Utilisateur> gestionnaire)
{
    _contexte = contexte;
    _gestionnaire= gestionnaire;
}

private Task<Utilisateur> GetCurrentUserAsync() =>
    _gestionnaire.GetUserAsync(HttpContext.User);

[HttpGet]
public async Task<string> RecupererIdUtilisateurCourant()
{
    Utilisateur utilisateur = await GetCurrentUserAsync();
    return utilisateur?.Id;
}
}

```

Enfin, vous pouvez appeler votre nouvelle méthode `RecupererIdUtilisateurCourant` à partir de la méthode `Index` où vous allez construire le modèle de vue pour la liste de films. Remarquez que la signature de la méthode `Index` a changé, passant d'une méthode synchrone à une tâche asynchrone. Cette évolution est nécessaire pour appeler les méthodes du service `UserManager`.

```

public async Task<ActionResult> Index()
{
    var id = await RecupererIdUtilisateurCourant();
    return View();
}

```

Créez la liste de films de l'utilisateur

Cela nous a demandé un peu de boulot, mais c'était nécessaire pour créer la liste de films spécifique à l'utilisateur connecté. Dans la méthode *Index*, insérez le code surligné suivant :

```

public async Task<IActionResult> Index()
{
    var id = await RecupererIdUtilisateurCourant();
    var filmsUtilisateur = _contexte.FilmsUtilisateur.Where(x => x.IdUtilisateur == id);
    var modele = filmsUtilisateur.Select(x => new ModelVueFilm
    {
        IdFilm = x.IdFilm,
        Titre = x.Film.Titre,
        Annee = x.Film.Annee,
        Vu = x.Vu,
        PresentDansListe = true,
        Note = x.Note
    }).ToList();

    return View(modele);
}

```

Après avoir récupéré l'identifiant de l'utilisateur connecté, une requête LINQ récupère tous les enregistrements *FilmUtilisateur* de la base de données qui le contiennent. Ensuite, à partir des résultats de cette requête, on sélectionne des éléments spécifiques tels que le titre du film, son année, son statut de visionnage par l'utilisateur et sa note. Ensuite, on construit une liste de nouveaux objets *ModeleVueFilm* en utilisant ces éléments. Enfin, on transmet cette liste d'objets *ModeleVueFilm* à la vue via la variable *modele*. Les données sont alors rendues dans un fichier HTML dynamique.

En résumé

Nous avons couvert beaucoup de choses dans ce chapitre !

- Vous avez maintenant une application Watchlist dotée d'un modèle de données complet, de contrôleurs qui gèrent vos entités de données et de vues qui affichent vos données.
- Vous avez appris tout ce qu'il y avait à savoir sur le processus de génération automatique et sur la façon dont il peut être utilisé pour créer des contrôleurs complets avec des vues basées sur une seule classe de modèle, des contrôleurs

plus simples (ou vides) sans vues, ou encore des vues distinctes de la génération automatique du contrôleur.

- Vous avez personnalisé le contrôleur *ListeFilmsController* pour gérer l'affichage des listes de films de vos utilisateurs.

Exécutez et testez votre application

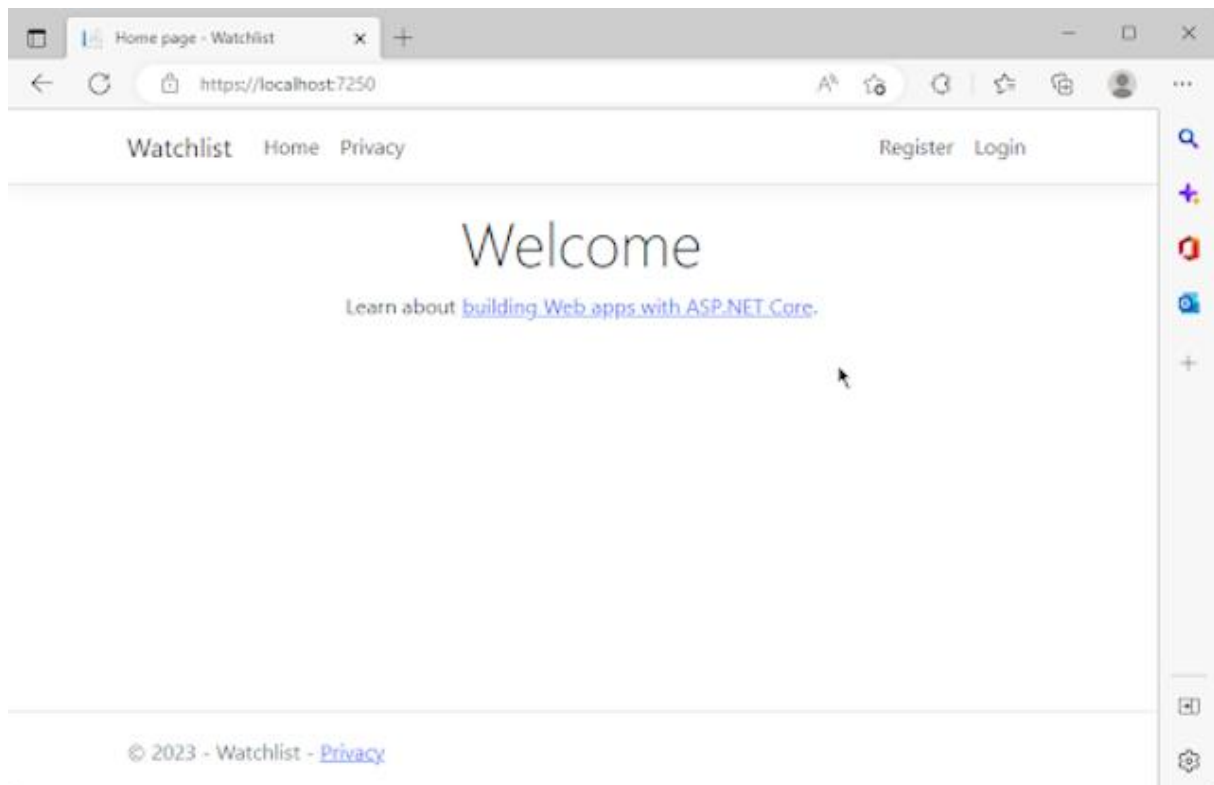
Exécutez l'application en mode débogage

Commençons cette troisième partie en **testant** ce que nous avons déjà réalisé. Ce chapitre a pour but de présenter le code généré par Entity Framework dans vos contrôleurs et vos vues en se basant sur les classes de modèle que vous avez créées. Une fois que vous vous serez familiarisé avec le code de base généré pour ces composants, nous identifierons les éléments que vous devez personnaliser pour répondre aux besoins de vos données et de votre application.

Le débogage de l'application est assez simple à ce stade. Compilez le projet pour vous assurer qu'il n'y a pas d'erreurs (s'il y en a, corrigez-les), puis appuyez sur F5 pour lancer le débogage.

Dans le chapitre 3 de la première partie, vous vous êtes inscrit en tant que nouvel utilisateur de votre application. Lorsque celle-ci est en cours d'exécution, inscrivez un nouvel utilisateur en cliquant sur le lien *Register* (Inscription) en haut à droite de la page. Testez ensuite la connexion pour vous assurer qu'elle fonctionne correctement.

Une fois connecté, vous verrez un écran comme celui-ci :



Page d'accueil de votre application

Il n'y a rien de spécial ici, juste la page d'index par défaut d'une application MVC. Vous remarquerez le message d'accueil en haut à droite, qui reprend l'adresse e-mail que vous avez utilisée pour vous connecter. Il est suivi d'un lien de déconnexion. Le message d'accueil avec votre adresse e-mail est un lien. Si vous cliquez dessus, vous accéderez à une page de gestion de compte où vous pourrez modifier votre mot de passe et d'autres informations personnelles. Cela ne nous concerne pas pour l'instant, mais je voulais simplement attirer votre attention sur son existence.

Vous êtes maintenant connecté, et l'application est prête à être testée. Mais, par où commencer ? Au vu de ce qui s'affiche à l'écran, vous avez bien raison de vous poser cette question. Identifions quelques éléments évidents à tester.

Identifiez les personnalisations nécessaires

La première chose que vous pourriez remarquer lorsque vous regardez la page d'accueil de votre application, c'est qu'elle n'offre aucun moyen à l'utilisateur d'accéder à sa liste de films personnelle ou à l'une des pages que vous avez générées automatiquement pour gérer les films dans la base de données. On peut donc envisager d'ajouter des éléments de menu. Allons-y.

Pas besoin d'arrêter l'application pour le faire. Toutes les modifications que vous souhaitez apporter aux vues peuvent être effectuées pendant que l'application est en cours d'exécution. En revanche, si des modifications doivent être apportées au code du

contrôleur ou du modèle, vous devrez arrêter le débogage et reprendre le développement standard.

Vous pouvez trouver tout le code de navigation dans le menu dans le fichier de disposition de votre application : *Views > Shared > _Layout.cshtml*. Vers la ligne 31 de ce fichier, juste en dessous de l'appel de la vue partielle *_LoginPartial*, vous verrez une liste HTML de balises d'ancrage qui comprend les deux éléments de menu de votre barre de navigation : *Home* (Accueil) et *Privacy* (Confidentialité). Insérez deux nouveaux éléments de menu entre eux. Nommez le premier "Ma liste de films" et faites-le pointer vers la méthode Index du contrôleur *ListeFilmsController*. Nommez le second "Films" et faites-le pointer vers la méthode Index du contrôleur *FilmsController*. Lorsque vous aurez terminé, votre liste devrait ressembler à ceci :

```
<ul class="navbar-nav flex-grow-1">

    <li class="nav-item">

        <a class="nav-link text-dark" asp-area="" asp-controller="Home"
asp-action="Index">Home</a>

    </li>

    <li class="nav-item">

        <a class="nav-link text-dark" asp-area=""
asp-controller="ListeFilms" asp-action="Index">Ma liste de films</a>

    </li>

    <li class="nav-item">

        <a class="nav-link text-dark" asp-area="" asp-controller="Films"
asp-action="Index">Films</a>

    </li>

    <li class="nav-item">

        <a class="nav-link text-dark" asp-area="" asp-controller="Home"
asp-action="Privacy">Privacy</a>

    </li>

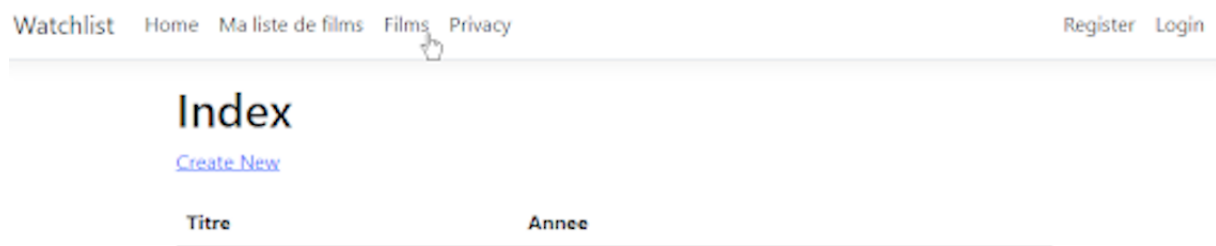
</ul>
```

Enregistrez votre travail et actualisez la page dans votre navigateur. Vous devriez voir vos nouveaux éléments de menu dans la barre de navigation. Testez les éléments de menu

en cliquant sur chacun d'eux et vérifiez qu'ils fonctionnent tous comme vous le souhaitez. Comme vous n'avez pas encore introduit de données, il n'y a pas grand-chose qui s'affiche, mais si les deux images suivantes correspondent aux écrans que vous voyez pour Ma liste de films et Films, vous êtes sur la bonne voie.



Test du nouvel élément "Ma liste de films" dans la barre de navigation.



Test du nouvel élément "Films" dans la barre de navigation.

Exercice

Dans la page d'index des films, vous remarquerez un lien Create New (Créer nouveau) sous le titre Index, en haut de la page. (Ce lien est également présent dans la page d'index de la liste de films, mais nous le modifierons plus tard.)

Prenons quelques minutes pour ajouter certains de vos films préférés à la base de données en utilisant le lien Create New (Créer nouveau) dans la page d'index des films. La page de création de films s'affiche alors. Étudiez les modalités de saisie d'informations proposées par la page. Voyez-vous des améliorations à apporter ? Est-il possible de rendre cette page plus simple à utiliser et de limiter les erreurs ? Comment voudriez-vous que cette page soit présentée et mise en forme si vous deviez créer une page idéale ? Notez ces idées sur une feuille de papier et gardez-la sous le coude. Nous y reviendrons dans le chapitre suivant.

Faites de même pour les pages Edit (Modifier), Details (Détails), Delete (Supprimer) et Index.

Une dernière modification, juste pour le plaisir

Réfléchissons maintenant à l'ergonomie. Ne serait-il pas intéressant que l'utilisateur soit dirigé automatiquement vers sa liste de films lors de sa connexion ? Il n'y a aucun intérêt à le faire arriver sur la page d'accueil. En effet, il est fort probable qu'il utilise l'application pour interagir avec sa liste de films. Voyons comment ajouter cette fonctionnalité, puis exécutons à nouveau l'application. Puisque le code nécessaire se trouve dans un contrôleur, vous devez fermer l'application. Pour ce faire, fermez la fenêtre du navigateur ou cliquez sur le bouton Stop dans la barre d'outils de Visual Studio.

Ouvrez maintenant le contrôleur *HomeController* et examinez la méthode *Index*. Vous pouvez y ajouter une simple vérification pour rediriger l'utilisateur s'il est connecté lorsque la page d'accueil est demandée. Regardez attentivement le code ajouté ci-dessous :

```
public IActionResult Index()
{
    if (User.Identity.IsAuthenticated)
    {
        return RedirectToAction("Index", "ListeFilms");
    }

    return View();
}
```

Ce code supplémentaire exploite l'objet statique *User* fourni par le contrôleur. Il examine la propriété booléenne *IsAuthenticated* de la propriété *Identity* de l'objet *User*. Si l'utilisateur est connecté, il est considéré comme **authentifié**, et cette propriété sera vraie (*true*). Par conséquent, si cette propriété est vraie, vous pouvez rediriger l'utilisateur vers la méthode *Index* du contrôleur *ListeFilms*.

En résumé

Dans ce chapitre, vous avez testé une grande partie des fonctionnalités de notre application :

- Vous avez inscrit un nouvel utilisateur, vous vous êtes connecté avec ce nouveau compte et vous avez ajouté plusieurs films à la base de données.
- Vous avez également utilisé les autres pages CRUD pour les films et noté certaines modifications de conception et de mise en forme à apporter ultérieurement.

Modifiez les vues de votre application MVC

Pourquoi modifier les vues ?

Les vues créées par défaut lorsque vous procédez à la génération automatique des entités du modèle sont généralement fonctionnelles sans modification. Cependant, dans certains cas, vous devrez leur apporter de petits ajustements. Par exemple, les listes de sélection dans les formulaires peuvent afficher l'identifiant de l'élément qu'elles représentent plutôt que son nom. Les dates peuvent être représentées par des zones de texte alors que vous souhaitez qu'elles s'affichent sous la forme d'un sélecteur de date. Dans les vues de liste (généralement la page d'index d'une entité donnée), le tableau qui présente les données peut comporter des champs pas vraiment indispensables, comme l'identifiant d'un élément.

Il est important de faire le point sur le rendu de chaque page générée et de noter les améliorations nécessaires, afin que chaque page reflète vos attentes en matière de conception ainsi que celles de votre client.

Dans le chapitre précédent, vous avez probablement noté quelques améliorations à apporter à certaines vues du projet. Heureusement, tous les formulaires fonctionnent très bien. Cependant, d'un point de vue esthétique, ils n'ont rien d'enthousiasmant. Dans ce chapitre, je veux mentionner certains points particuliers que vous devez prendre en compte lorsque vous utilisez les vues générées, même si la plupart ne s'appliquent pas au projet actuel. Ils revêtiront une importance particulière dans votre future vie professionnelle.

Assurez-vous d'utiliser le bon type de champ de formulaire

Les champs de formulaire peuvent parfois poser problème. Il est particulièrement agaçant de constater que vous ne parvenez pas à saisir vos données correctement parce que le type de champ de formulaire utilisé n'est pas le bon. Voici quelques points sensibles à surveiller :

- Assurez-vous que les bonnes données s'affichent dans les listes. Si une liste permet plusieurs sélections, optez pour une liste à sélection multiple plutôt que pour une liste à sélection simple, même si c'est ce dernier type de contrôle qui est habituellement généré.
- C# utilise des objets `DateTime` à la fois pour les dates et les heures. Si vous avez besoin d'un champ de type heure dans votre formulaire, n'affichez pas un champ de type date, et inversement. Pour des raisons de cohérence, utilisez un sélecteur de date dans votre formulaire et spécifiez également des champs de type heure.
- Si des champs nécessitent uniquement la saisie d'un nombre entier, assurez-vous d'utiliser le bon type d'entrée. Les entrées des espaces de texte sont

particulièrement délicates. Un champ de texte simple est généré pour chaque entrée composée de chaînes de caractères. Entity Framework ne sait pas si vous devez prévoir un emplacement pour un paragraphe ou deux, et certainement pas si vous pourriez avoir besoin d'une entrée au format HTML. Si vous utilisez de telles entrées dans vos formulaires, vous devrez les spécifier et mettre en forme directement. Le moyen le plus simple d'y parvenir est d'utiliser la méthode d'assistance `@Html.TextAreaFor()`. L'utilisation de cette méthode créera l'élément HTML de champ de texte approprié dans votre formulaire. Par exemple :

```
@Html.TextAreaFor(model => model.Commentaires, 8, 0, new {  
    @class = "form-control", required = "required" })
```

Cette instruction crée un nouvel espace de texte pour un élément de modèle appelé *Commentaires*. Le deuxième paramètre indique le nombre de lignes (8), et le troisième paramètre est le nombre de colonnes. Une valeur de zéro signifie que l'espace remplira la largeur de l'espace disponible, tout comme les colonnes. Le dernier paramètre est un objet anonyme qui représente les attributs HTML appliqués au champ. Dans ce cas, la classe CSS *form-control* est appliquée, de même que l'attribut *required*.

Ce ne sont là que quelques exemples de problèmes auxquels il faut faire attention. Comme toujours, surveillez attentivement tous les détails de vos formulaires pour vous assurer qu'ils sont intuitifs et faciles à appréhender pour vos utilisateurs.

Optimisez la cohérence des formulaires

Les formulaires *Create* (Créer) et *Edit* (Modifier) sont presque identiques lorsqu'ils sont générés. Toute modification apportée à l'un doit être répercutée sur l'autre. Vous devez en effet éviter d'avoir un sélecteur de date dans un formulaire et un espace de texte pour le même champ dans l'autre.

Faites également attention à la navigation. Par défaut, chaque formulaire intègre une action d'envoi et une action d'annulation. L'action d'envoi est un bouton intitulé *Create* (Créer) dans le formulaire *du même nom* et *Save* (Enregistrer) dans le formulaire *Edit* (Modifier). L'action d'annulation est un hyperlien *Back to List* (Retour à la liste) dans tous les formulaires. Les développeurs remplacent souvent ce lien par un bouton intitulé *Annuler* ou autre nom similaire en fonction du design choisi. Si vous faites ce choix, veillez à l'appliquer à tous les formulaires de vos différentes vues.

Gérez les erreurs humaines avec la validation

La validation fait déjà partie du processus de génération automatique MVC pour la plupart des types de données. Chaque formulaire qui requiert une saisie sera généré avec la ligne suivante après la balise `<form>` :

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
```

Cette balise indique où s'affichera un résumé des erreurs de validation. Les valeurs qui peuvent être utilisées sont **All**, **ModelOnly** et **None**. `validationSummary.All` affichera les messages de validation au niveau de la propriété et du modèle.

`validationSummary.ModelOnly` affichera uniquement les messages de validation qui s'appliquent au niveau du modèle. `validationSummary.None` indique à cette balise de ne rien faire.

Toutes les erreurs de saisie de données seront indiquées par un texte rouge sous le champ de saisie. Le code HTML qui génère de telles erreurs ressemble à ceci :

```
<span asp-validation-for="Title" class="text-danger"></span>
```

Cette ligne définit un message de validation pour un élément de modèle appelé `Title`. Si la saisie est obligatoire et que rien n'est indiqué dans ce champ, un message de validation indiquant que le champ est obligatoire s'affiche sous le champ de saisie.

Ajoutez des films à la liste des films

Avez-vous remarqué qu'il n'existe aucun moyen d'ajouter des films à votre liste de films ? Vous pouvez les ajouter à la base de données, ainsi que les modifier et les supprimer, mais vous n'avez actuellement aucun moyen de les placer dans votre liste de films. C'est un gros problème pour notre application !

Plusieurs options s'offrent à vous, je vais vous en présenter quelques-unes :

1. Ajouter/supprimer des films directement à partir de la page de liste des films (Index) via des boutons individuels Ajouter/Supprimer pour chaque film
2. Ajouter/supprimer des films directement à partir de la page de liste des films (Index) via des cases à cocher à côté de chaque film
3. Ajouter un film via la page Create (Créer)
4. Ajouter/supprimer un film via la page Edit (Modifier)
5. Ajouter/supprimer un film via la page Details (Détails)

Chacune de ces options présente des avantages et des inconvénients. Les options 1 et 2 sont les plus efficaces et les plus pratiques, mais elles nécessitent un codage JavaScript supplémentaire et un appel de type API au contrôleur. Les options 3 à 5 conviennent à une application basée uniquement sur MVC, mais n'offrent pas une interaction idéale et pourraient entraîner une certaine insatisfaction chez l'utilisateur. Dans cette optique, je suggère l'option 1. C'est la plus simple des deux premières, et son utilisation me permet de vous montrer comment les contrôleurs MVC peuvent également servir de passerelle vers des fonctions d'API.

Puisque vous disposez déjà de la vue, il vous suffit de la modifier et d'apporter quelques modifications au contrôleur *FilmsController*. Concrètement, vous devez :

1. indiquer si le film figure dans la liste de films de l'utilisateur ;
2. insérer des boutons Ajouter/Supprimer dans chaque ligne du tableau ;
3. ajouter du JavaScript pour gérer les clics sur les boutons et appeler une nouvelle action issue du contrôleur *FilmsController* ;
4. mettre à jour la méthode *Index* dans le contrôleur *FilmsController* pour créer le modèle en utilisant les objets *ModelVueFilm* au lieu des objets *Film* ;
5. ajouter une nouvelle méthode (*AjouterSupprimer*) au contrôleur *FilmsController*.

Écrivons un bout de code supplémentaire.

Indiquez si le film figure dans la liste de films de l'utilisateur

Pour ce faire, vous devez modifier le modèle que la page utilise. Heureusement, nous en avons un qui est parfait dans la classe *ModelVueFilm* que nous avons écrite plus tôt dans le cours. Voici la définition du modèle actuel, sur la première ligne de la page *Index* :

```
@model IEnumerable<Watchlist.Data.Film>
```

Remplacez-la par ceci :

```
@model IEnumerable<Watchlist.Models.ModeleVueFilm>
```

Ensuite, vous devez ajouter une colonne supplémentaire au tableau. Ajoutez-en une à l'en-tête et au corps du tableau, entre la colonne *Année* et la dernière colonne. Nommez l'en-tête *Présent dans la liste*.

Dans le corps du tableau, la nouvelle colonne doit afficher un bouton avec un signe plus (+) si le film n'est pas dans la liste de films de l'utilisateur, et un signe moins (-) dans le cas contraire. Chaque bouton doit avoir son identifiant, ainsi qu'un attribut de données qui contient la valeur de l'attribut de modèle *PresentDansListe*. Par exemple :

```
<button id="@item.MovieId" data-val="@item.InWatchlist" class="btn">
```

```
    @(item.PresentDansListe ? "-" : "+")
```

```
</button>
```

Cet exemple montre comment utiliser la syntaxe Razor, indiquée par le signe @ précédant un nom de variable, pour accéder aux données de notre modèle. Insérez l'*identifiant* de chaque film comme attribut *Id* du bouton correspondant. Ajoutez également *presentDansListe-val* comme paramètre et donnez-lui la valeur de l'attribut *PresentDansListe* de l'objet *ModeleVueFilm*. Pour afficher le signe approprié

sur le bouton, vérifiez la valeur de l'attribut *PresentDansListe*. Si la valeur est vraie (true), affichez le signe moins, sinon affichez le signe plus.

Utilisez JavaScript pour gérer l'interaction avec l'utilisateur

Les boutons ajoutés doivent tous appeler une action du contrôleur FilmsController lors du clic. Pour que le script soit appelé au bon moment, ajoutez-le dans une section Razor appelée Scripts, en bas de la page. Ajoutez le script suivant en bas de la page Index :

```
@section Scripts {  
  
<script>  
  
    jQuery(document).ready(function () {  
  
        $('.btn').click(function (e) {  
  
            var btn = $(this);  
  
            var idFilm = btn.attr('id');  
  
            var valFilm = btn.attr('presentDansListe-val') == "False" ? 0 : 1;  
  
            $.get('/Films/AjouterSupprimer?id=' + idFilm + '&val=' + valFilm,  
function (data) {  
  
            if (data == 0) {  
  
            btn.attr('presentDansListe-val', 'False');  
  
            btn.html(' + ');  
  
            }  
  
            else if (data == 1) {  
  
            btn.attr('presentDansListe-val', 'True');  
  
            btn.html(' - ');  
  
            }  
  
            });  
  
            });  
  
        });  
  
</script>
```

}

Ce script capture chaque clic sur le bouton, récupère les attributs *Id* et *presentDansListe-val* du bouton, puis appelle la méthode *AjouterSupprimer* du contrôleur *FilmsController* et lui transmet ces valeurs dans l'URL. Ensuite, en fonction de la valeur renvoyée (présence ou absence du film dans la liste de films de l'utilisateur), il modifie le signe affiché sur le bouton.

Cependant, pour que cela fonctionne, vous devez écrire cette méthode *AjouterSupprimer* dans le contrôleur *FilmsController*. C'est ce que nous ferons dans le prochain chapitre.

En résumé

Dans ce chapitre, vous avez découvert certains problèmes et pièges particuliers auxquels vous pourriez être confronté lors de l'utilisation de vues générées automatiquement dans .NET MVC. Concrètement, vous avez appris :

- l'importance de s'assurer que le type d'entrée HTML correspond au type de l'élément du modèle qu'il représente ;
- l'importance de maintenir la cohérence entre toutes les vues de votre projet, tant au niveau de la représentation des types de données que de la présentation des formulaires et des pages eux-mêmes ;
- comment ASP.NET gère la validation des entrées de formulaire dans les pages HTML.

Optimisez les contrôleurs de votre application MVC

Priorité à la sécurité

Chaque fois que vous créez une nouvelle application web ASP.NET, vous avez la possibilité d'ajouter un module d'authentification. Pour notre application Watchlist, nous avons choisi l'authentification par des comptes individuels. L'authentification, c'est tout simplement le processus par lequel votre application vérifie l'identité d'un utilisateur, généralement par le biais d'un nom d'utilisateur et d'un mot de passe. Cette fonctionnalité est déjà opérationnelle, et vous l'avez testée en inscrivant un nouveau compte d'utilisateur.

Il existe un autre mécanisme de sécurité dans ASP.NET appelé autorisation. Vous en apprendrez beaucoup plus à ce sujet dans le cours sur la sécurité .NET, mais je tiens à vous le présenter rapidement ici. Pour simplifier, l'autorisation est la façon dont vous décidez à quelles parties de votre site les utilisateurs peuvent accéder. Par exemple, voulez-vous permettre aux utilisateurs non inscrits de consulter les listes de films de

vos abonnés ? Peuvent-ils ajouter des films à votre base de données, en modifier ou en supprimer ? L'autorisation vous permet de contrôler ces différents éléments. Elle est gérée grâce à l'**attribut de données Authorize**.

Cet attribut n'est pas difficile à implémenter. Je vous recommande de dresser une liste des contrôleurs et des actions de contrôleur de votre application, et d'attribuer un niveau d'accès à chacun d'eux. Commencez par ces quatre catégories d'accès :

1. Tous les utilisateurs
2. Utilisateurs inscrits (authentifiés)
3. Types précis d'utilisateurs inscrits
4. Utilisateurs précis

Une fois que vous avez identifié le type d'accès requis par les classes et les actions (méthodes) de votre contrôleur, vous pouvez ajouter l'attribut de données approprié à cette classe ou méthode. Vous pouvez sécuriser une classe entière ou des méthodes individuelles, et la syntaxe est simple. Par exemple, pour autoriser uniquement les utilisateurs inscrits à accéder à tout élément de la classe `FilmsController`, vous devez ajouter l'attribut de données `Authorize` directement au-dessus de la déclaration de la classe. Pour utiliser cet attribut, vous devez également ajouter une nouvelle déclaration `using` :

```
using Microsoft.AspNetCore.Authorization;
```

```
[Authorize]
```

```
public class FilmsController : Controller
```

```
{
```

```
    ...
```

```
}
```

L'utilisation de cet attribut au-dessus de la définition de la classe sécurise l'ensemble de la classe. Toute requête auprès de l'une des méthodes de la classe nécessitera l'authentification de l'utilisateur par son nom d'utilisateur et son mot de passe. Si l'utilisateur n'est pas connecté, il sera automatiquement redirigé vers la page de connexion.

Vous pouvez également sécuriser des méthodes spécifiques tout en laissant les autres méthodes du même contrôleur librement accessibles. Pour ce faire, vous ne devez pas placer l'attribut `Authorize` au-dessus de la définition de la classe, mais au-dessus de

chaque méthode à sécuriser. Seules ces méthodes nécessiteront l'authentification de l'utilisateur.

Nous pourrions aller plus loin sur le sujet, par exemple en expliquant comment sécuriser des sections de votre application en fonction du type d'utilisateur ou même du nom d'utilisateur. Toutefois, ce n'est pas indispensable pour ce cours et ce projet.

Liez les données

Les actions du contrôleur reçoivent des données provenant de requêtes HTTP. Ces données ne sont pas transmises en des types de données .NET prédéfinis. Écrire du code pour récupérer chaque valeur et la convertir de chaîne de caractères en type .NET approprié serait compliqué. Le modèle dynamique s'en charge pour vous. La liaison dynamique de données (modèle) :

- récupère les données envoyées à partir des routes URL, des champs de formulaire et de la requête ;
- transmet les données à la méthode du contrôleur par le biais de ses paramètres ;
- convertit les données de la chaîne en types de données .NET appropriés.

Par défaut, toutes les propriétés de modèle envoyées à partir d'un formulaire sont liées à une méthode du contrôleur correspondant avec Bind. La syntaxe se présente comme suit :

[HttpPost]

[ValidateAntiForgeryToken]

```
public async Task<IActionResult> Create([Bind("Id,Titre,Annee")] Film film)
{
    if (ModelState.IsValid)
    {
        _contexte.Add(film);
        await _contexte.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(film);
}
```

Si une propriété est absente de la liste Bind, sa valeur ne sera pas ajoutée aux données de l'objet entrant. Si vous ajoutez des propriétés à vos modèles de données *après* avoir généré automatiquement vos contrôleurs, assurez-vous de les insérer dans les listes Bind des méthodes de vos contrôleurs.

Ajoutez une nouvelle fonctionnalité à Watchlist

Dans le chapitre précédent, nous avons apporté plusieurs modifications à la page Index des films pour permettre d'ajouter et de supprimer facilement des films de la liste de films de l'utilisateur. Une fois ces opérations terminées au niveau du front-end, vous devez apporter plusieurs changements au contrôleur *FilmsController* pour que la vue soit effectivement modifiée.

Tout d'abord, vous avez besoin de l'identifiant de l'utilisateur pour créer sa liste de films à partir de la base de données. Cela signifie que vous devez avoir accès au UserManager, que vous pouvez injecter dans le contrôleur par le biais de son constructeur, tout comme vous le faites avec l'objet de contexte de la base de données. Par exemple :

```
public class FilmsController : Controller
{
    private readonly ApplicationDbContext _contexte;
    private readonly UserManager<Utilisateur> _gestionnaire;

    public FilmsController(ApplicationDbContext contexte,
        UserManager<Utilisateur> gestionnaire)
    {
        _contexte = contexte;
        _gestionnaire = gestionnaire;
    }

    ...
}
```

Récupérez ensuite les données de l'utilisateur actuel avec UserManager.

```
public class FilmsController : Controller
```

```

{
    private readonly ApplicationDbContext _contexte;
    private readonly UserManager<Utilisateur> _gestionnaire;

    public FilmsController(ApplicationDbContext contexte,
        UserManager<Utilisateur> gestionnaire)
    {
        _contexte = contexte;
        _gestionnaire = gestionnaire;
    }

    [HttpGet]
    public async Task<string> RecupererIdUtilisateurCourant()
    {
        Utilisateur utilisateur = await GetCurrentUserAsync();
        return utilisateur?.Id;
    }

    private Task<Utilisateur> GetCurrentUserAsync() =>
        _gestionnaire.GetUserAsync(HttpContext.User);

    ...
}

```

Maintenant que vous pouvez obtenir l'identifiant de l'utilisateur, vous pouvez déterminer quels films se trouvent dans sa liste de films pour construire le modèle. Pour l'instant, la méthode Index envoie une liste d'objets *Film* à la page Index. Vous devez changer ce comportement pour qu'elle envoie une liste d'objets *ModeleVueFilm*.

```

public class FilmsController : Controller
{

```

...

```
public async Task<IActionResult> Index()
{
    var idUtilisateur = await RecupererIdUtilisateurCourant();
    var modele = await _contexte.Films.Select(x =>
        new ModeleVueFilm
        {
            IdFilm = x.Id,
            Titre = x.Titre,
            Annee = x.Annee
        }).ToListAsync();
    foreach(var item in modele)
    {
        var m = await _contexte.FilmsUtilisateur.FirstOrDefaultAsync(x =>
            x.IdUtilisateur == idUtilisateur && x.IdFilm == item.IdFilm);
        if(m != null)
        {
            item.PresentDansListe = true;
            item.Note = m.Note;
            item.Vu = m.Vu;
        }
    }
    return View(modele);
}
```

Vous pouvez maintenant ajouter la méthode *AjouterSupprimer* au contrôleur *FilmsController*. Cette méthode doit renvoyer un objet *JsonResult* au lieu d'une vue. De cette façon, vous pouvez mettre à jour le DOM de la page Index sans

recharger toute la page. Déterminez si le film est ajouté ou retiré de la liste de films, et renvoyez une valeur en conséquence. Le plus simple est de retourner -1 s'il n'y a pas de changement, 0 si le film est supprimé, et 1 si le film est ajouté. Définissez la méthode comme suit :

[HttpGet]

```
public async Task<JsonResult> AjouterSupprimer()
```

```
{
```

```
}
```

Ajoutez maintenant une variable pour stocker la valeur de retour et récupérer l'identifiant de l'utilisateur.

[HttpGet]

```
public async Task<JsonResult> AjouterSupprimer()
```

```
{
```

```
    int valret = -1;
```

```
    var idUtilisateur = await RecupererIdUtilisateurCourant();
```

```
}
```

Ensuite, vérifiez la valeur du paramètre valret pour savoir si vous ajoutez ou supprimez un film. Si la valeur est 1, le film est déjà dans la liste de films et doit donc être supprimé. Si la valeur est 0, le film n'est pas dans la liste de films et doit être ajouté.

[HttpGet]

```
public async Task<JsonResult> AjouterSupprimer()
```

```
{
```

```
    int valret = -1;
```

```
    var idUtilisateur = await RecupererIdUtilisateurCourant();
```

```
    if (valret == 1)
```

```
    {
```

```
        // s'il existe un enregistrement dans FilmsUtilisateur qui contient à la fois l'identifiant de l'utilisateur
```

```
        // et celui du film, alors le film existe dans la liste de films et peut
```

```

// être supprimé

var film = _contexte.FilmsUtilisateur.FirstOrDefault(x =>
    x.IdFilm == id && x.IdUtilisateur == idUtilisateur);

if (film != null)
{
    _contexte.FilmsUtilisateur.Remove(film);

    valret = 0;
}

}

else
{
    // le film n'est pas dans la liste de films, nous devons donc
    // créer un nouvel objet FilmUtilisateur et l'ajouter à la base de données.
    _contexte.FilmsUtilisateur.Add(
        new FilmUtilisateur
        {
            IdUtilisateur = idUtilisateur,
            IdFilm = id,
            Vu = false,
            Note = 0
        }
    );

    valret = 1;
}

// nous pouvons maintenant enregistrer les changements dans la base de données
await _contexte.SaveChangesAsync();

// et renvoyer notre valeur de retour (-1, 0 ou 1) au script qui a appelé

```

```
// cette méthode depuis la page Index  
  
return Json(valret);  
  
}
```

En résumé

- Dans ce chapitre, vous avez appris à renforcer la sécurité de vos contrôleurs avec l'attribut *Authorize*.
- Vous avez également appris comment .NET MVC établit la liaison entre les attributs de données des vues et les actions (méthodes) des contrôleurs.
- Enfin, vous avez apporté plusieurs modifications à votre contrôleur *FilmsController* pour que les changements apportés à la vue *index* dans le chapitre précédent fonctionnent comme vous l'aviez prévu, complétant ainsi la spécification fonctionnelle de votre application Watchlist.

Modifiez l'apparence de votre application

Utilisez les modèles Bootstrap

Tous les projets web Visual Studio sont générés avec les bibliothèques CSS Bootstrap et jQuery incluses. Il est donc facile d'ajouter de nouveaux modèles Bootstrap pour modifier l'apparence de votre application sans avoir à modifier la majorité du HTML/CSS des vues, sauf pour obtenir un design personnalisé.

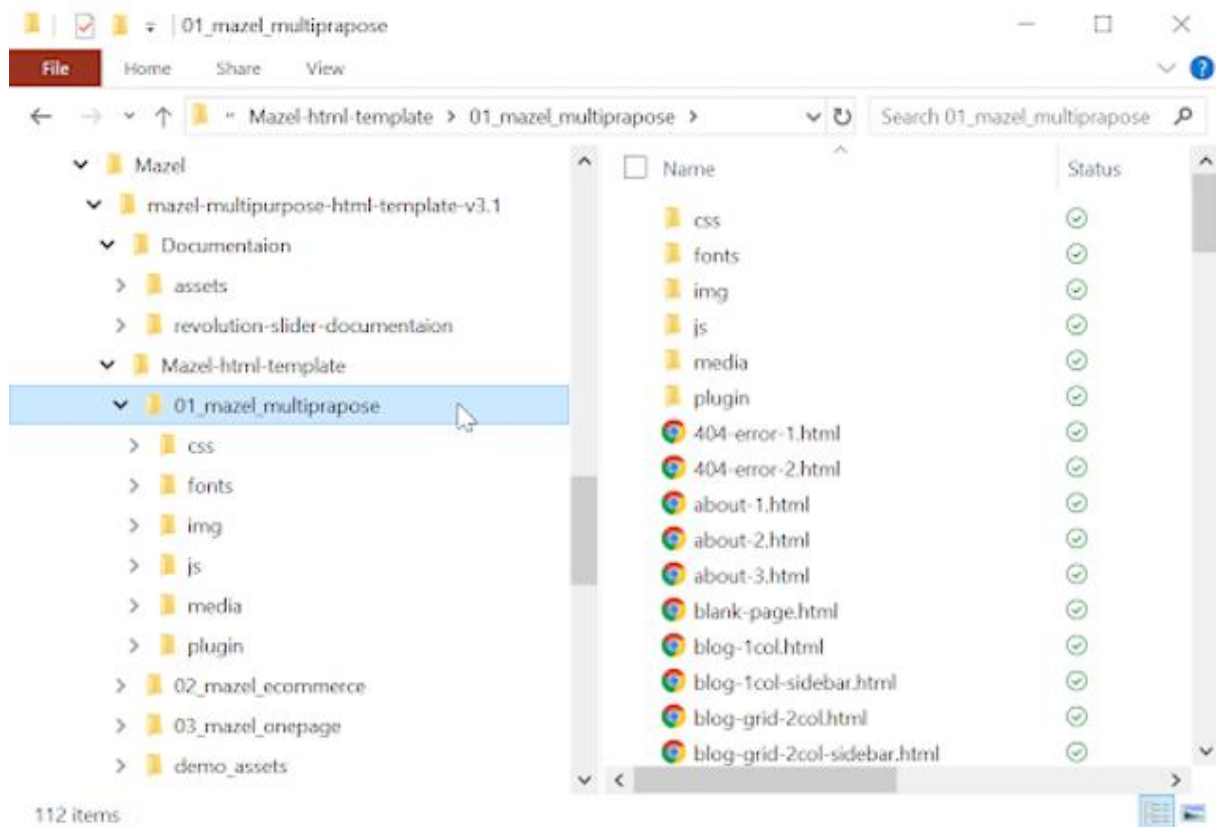
Les entreprises et les particuliers du monde entier créent et publient des modèles Bootstrap que des développeurs comme vous et moi peuvent acheter et utiliser dans des applications. L'un de mes sites préférés pour trouver ces modèles est wrapbootstrap.com. Ce site propose des centaines de modèles et vous permet d'affiner vos choix par catégorie (polyvalent, affaires, administration, page unique, etc.), par version de Bootstrap (3.x, 4.x, toutes), ou par popularité, prix et autres critères. Vous pouvez également effectuer une recherche générale si vous avez une idée précise de ce que vous voulez.

Pour cette démonstration, j'ai acheté un modèle appelé [Mazel](#), qui m'a coûté la modique somme de 10 €. Vous n'êtes pas obligé d'utiliser le même modèle. En vérité, vous n'avez pas besoin du tout d'en utiliser un pour faire fonctionner l'application. Je veux simplement vous montrer comment des modèles tels que celui-ci peuvent embellir vos applications.

Lorsque vous téléchargez l'un de ces modèles, vous obtenez un fichier ZIP que vous devez extraire. Après avoir décompressé l'archive du modèle Mazel, je me retrouve avec deux dossiers principaux : *Documentation* et *Mazel-html-template*. Les instructions

spécifiques au modèle se trouvent dans le dossier Documentation. Il est conseillé de les lire avant d'intégrer un modèle à votre projet.

Il existe trois variantes du modèle dans le dossier Mazel-html-template. Je vais utiliser la variante polyvalente, illustrée ci-dessous :



Le modèle polyvalent

Bien que ce ne soit pas compliqué, incorporer votre nouveau modèle à votre projet ne se fait pas en un claquement de doigts. Je vais donc vous guider tout au long de ce processus.

Comprenez le fonctionnement des dispositions .NET MVC

J'ai ajouté les ressources du modèle au projet Watchlist. La prochaine étape consiste à configurer une page de disposition pour qu'elle puisse être appliquée à toutes les autres pages du projet, puis à configurer une page d'exemple pour que vous puissiez voir le modèle en action.

En résumé

Ce chapitre portait sur l'embellissement de notre application à l'aide de modèles CSS Bootstrap :

- Vous avez appris à chercher et à télécharger des modèles sur wrapbootstrap.com, une excellente source de modèles à petits prix.

- Vous avez également appris à intégrer un modèle dans votre projet MVC, ainsi qu'à mettre à jour votre page de disposition et d'autres pages de contenu pour les mettre en forme à l'aide du modèle.

Publiez votre application

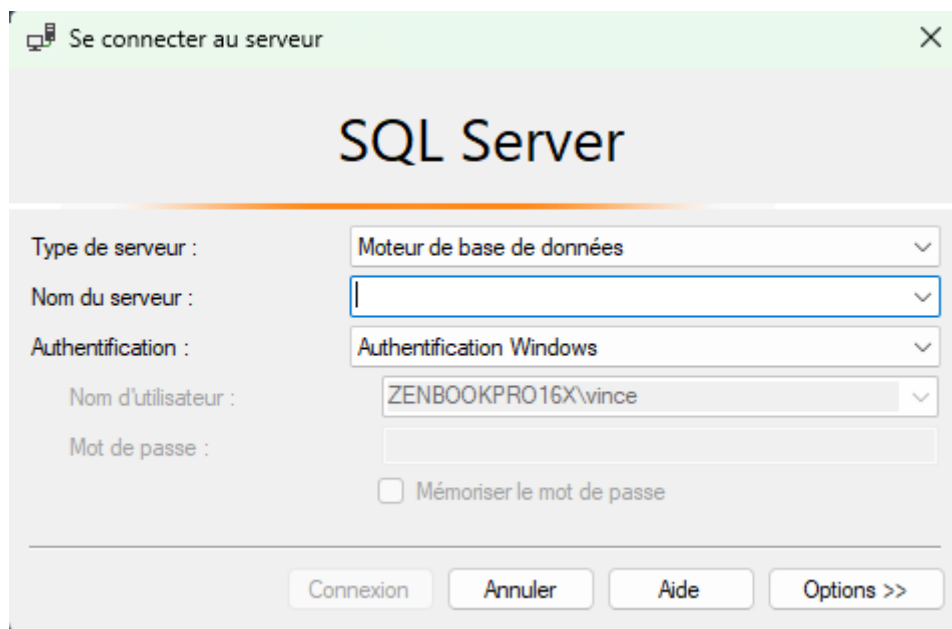
Migrez la base de données vers Azure

Il existe plusieurs façons de migrer une base de données locale vers Azure. La méthode choisie peut dépendre de la nécessité d'inclure ou non le schéma de la base de données dans la migration. Par exemple, si tout ce dont vous avez besoin est une nouvelle base de données vide avec un schéma qui correspond à votre modèle de données MVC, les migrations code first sont une excellente solution. En revanche, si vous avez déjà des données dans votre base et que vous devez les préserver lors de la migration vers Azure, vous devrez peut-être envisager une autre option.

Vous trouverez diverses méthodes en ligne, dont la plupart sont longues et compliquées. La plus simple que je connaisse est celle de SQL Server Management Studio (SSMS). Nous allons l'utiliser ici pour vous apprendre à migrer votre base de données locale vers Azure. Vous pouvez [télécharger SSMS](#) si vous ne l'avez pas encore installé. Une fois cette opération effectuée, vous n'aurez plus qu'à déplacer votre base de données. La suite de ce chapitre vous guidera étape par étape.

Connectez-vous à votre serveur local

Lorsque vous lancez SSMS, la fenêtre de connexion à SQL Server s'affiche :

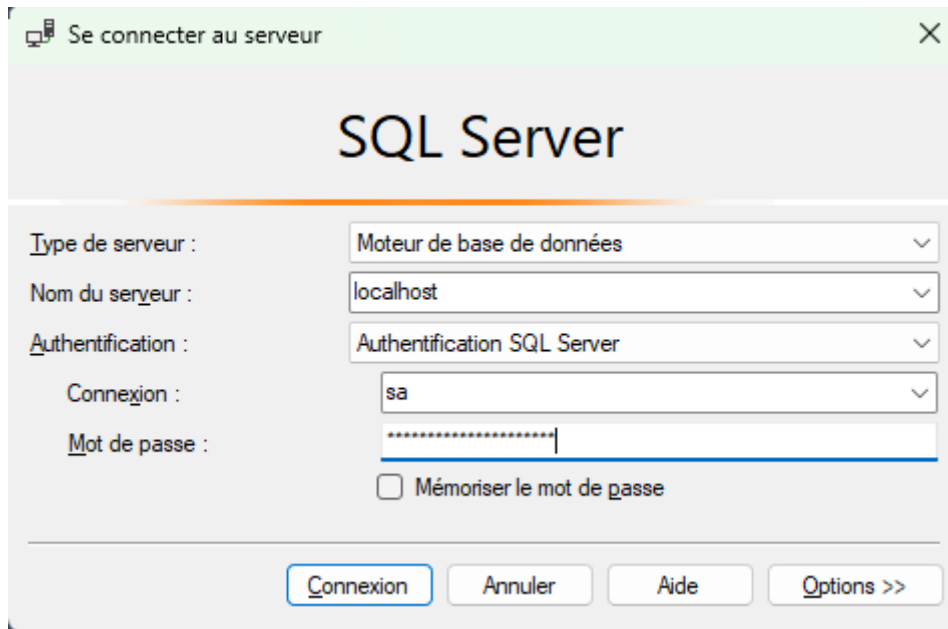


SQL Server :

connexion au serveur

Pour sélectionner un serveur, cliquez sur la liste déroulante Nom du serveur. Vous verrez toutes les instances SQL Server détectées par SSMS. Vous pouvez en sélectionner une

si vous voyez celle qui vous intéresse. Toutefois, si rien n'apparaît ou si vous devez vous connecter à un autre serveur, vous pouvez cliquer sur <Parcourir...>. Vous pouvez également saisir (ou coller) le nom du serveur auquel vous souhaitez vous connecter, comme celui qui figure dans la chaîne de connexion de votre application :



SQL Server :

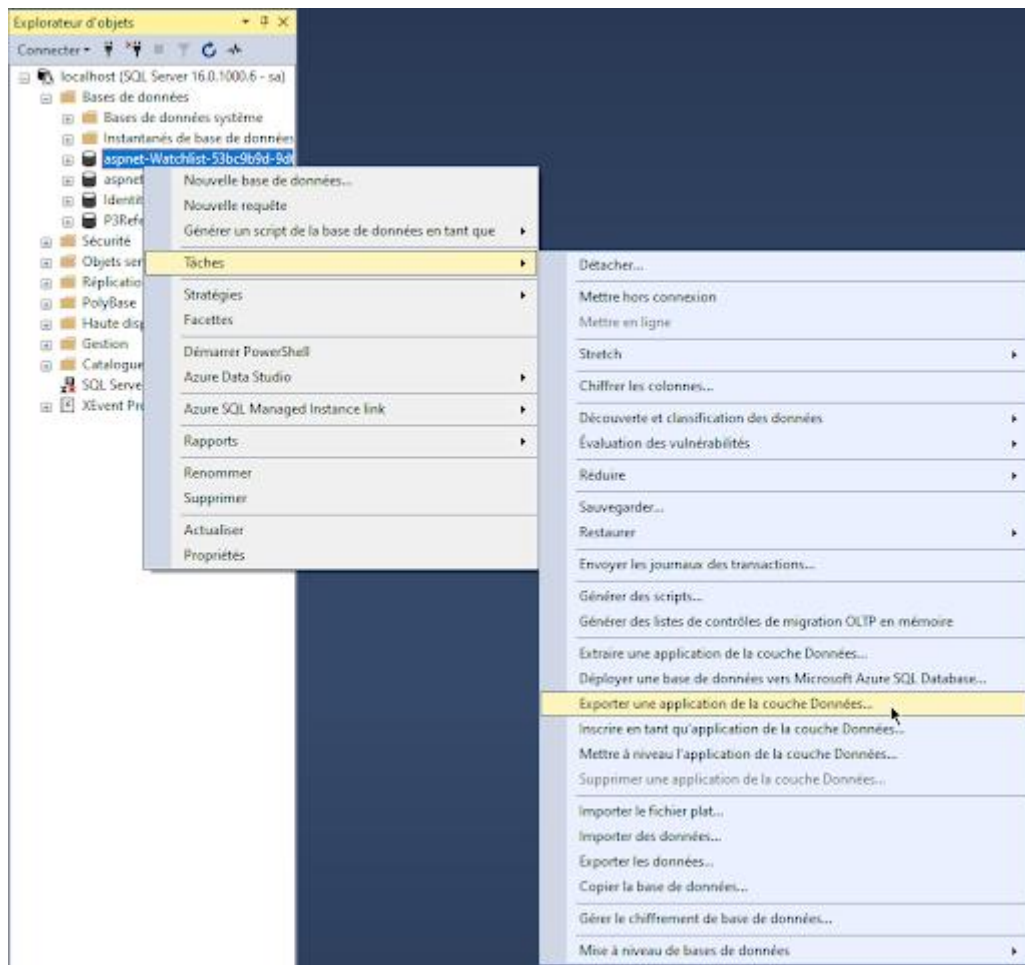
connexion au serveur avec le nom de serveur mentionné

À ce stade, vous pouvez modifier le type d'authentification si nécessaire. Pour les installations locales, votre type d'authentification sera très probablement l'authentification Windows, qui correspondra par défaut à votre nom d'utilisateur de connexion Windows. Vous pouvez donc cliquer sur Connexion pour établir la connexion au serveur. Si vous vous connectez à un serveur en ligne ou en réseau, sélectionnez Authentication SQL Server dans la liste déroulante. Vous le ferez plus tard, lorsque vous configurerez votre serveur Azure. Pour l'instant, restons-en à l'installation de votre serveur local et à l'authentification Windows.

Localisez et exportez votre base de données locale

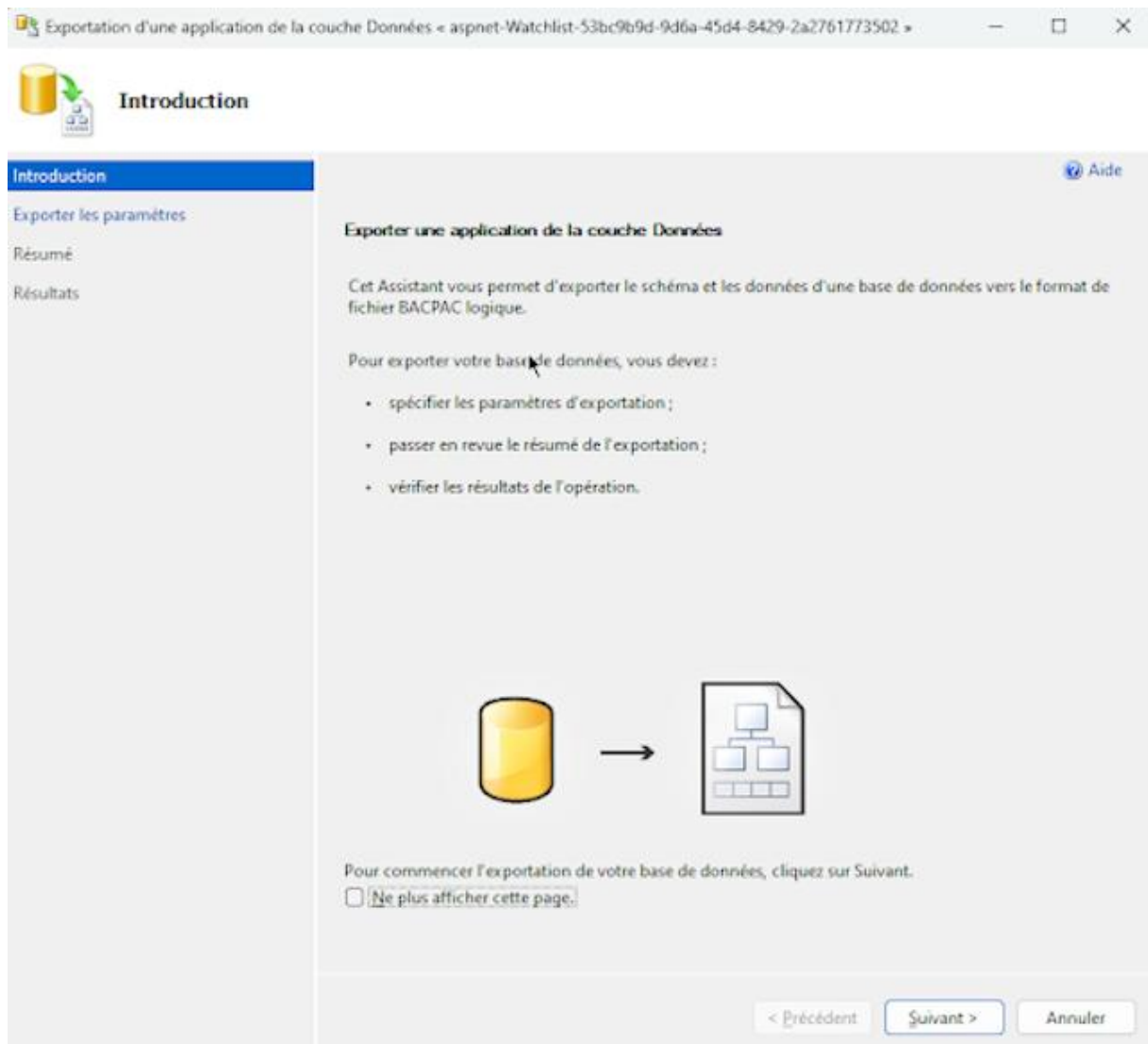
Après vous être connecté à votre serveur local dans SSMS, vous verrez le contenu du serveur listé dans le volet de l'*Explorateur d'objets* de SSMS. Développez le dossier *Bases de données* pour voir la liste des bases de données sur le serveur. Vous verrez une base de données qui commence par *aspnet-Watchlist*.

Faites un clic droit sur la base de données Watchlist, puis choisissez *Tâches > Exporter une application de la couche Données...*



Recherchez

et exportez votre base de données locale.



Dans la fenêtre d'introduction qui s'affiche, cliquez sur le bouton Suivant.

Dans la fenêtre suivante, choisissez *Enregistrer sur le disque local* et cliquez sur *Parcourir*.

Exportation d'une application de la couche Données « aspNet-Watchlist-53bc9b9d-9d6a-45d4-8429-2a2761773502 »

Exporter les paramètres

Introduction

Exporter les paramètres

Résumé

Résultats

Exporter les paramètres

Cette opération va créer un fichier BACPAC qui contient le contenu logique de votre base de données. Pour continuer, spécifiez l'emplacement où vous voulez créer le fichier BACPAC, puis cliquez sur Suivant. Pour spécifier un sous-ensemble de tables à exporter, utilisez l'option Avancées.

Paramètres Avancé

☒ Enregistrer sur le disque local

☐ Enregistrer dans Microsoft Azure

Compte de stockage : Se connecter...

Conteneur :

Nom de fichier : aspNet-Watchlist-53bc9b9d-9d6a-45d4-8429-2a2761

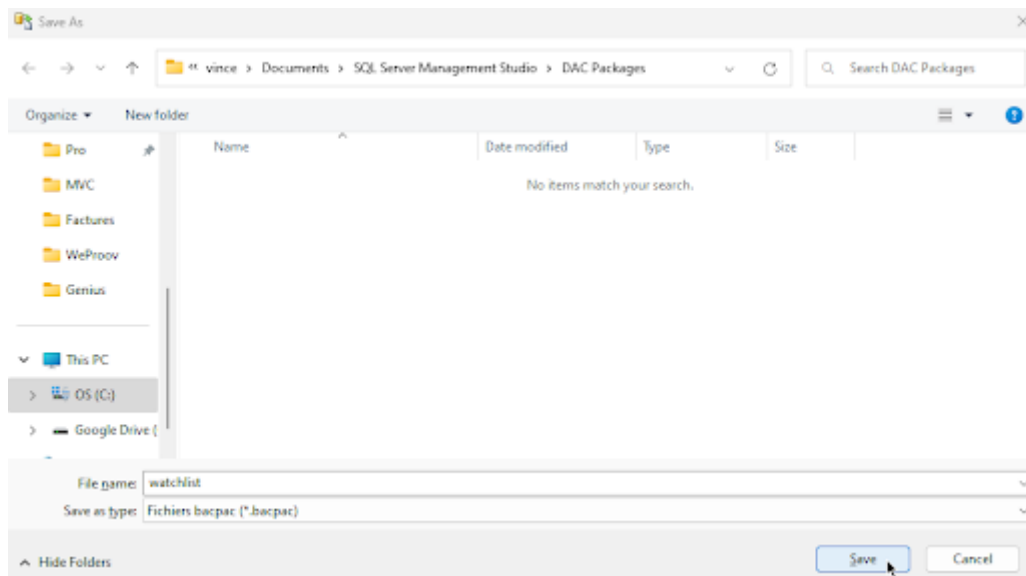
Nom de fichier temporaire :

C:\Users\vince\AppData\Local\Temp\aspnet-Watchlist-53bc9b9d-9d6a-45d4-8429- Parcourir...

< Précédent Suivant > Annuler

Fenêtre Exporter les paramètres

Choisissez l'emplacement dans lequel vous souhaitez enregistrer le fichier, donnez-lui un nom et cliquez sur *Enregistrer*.



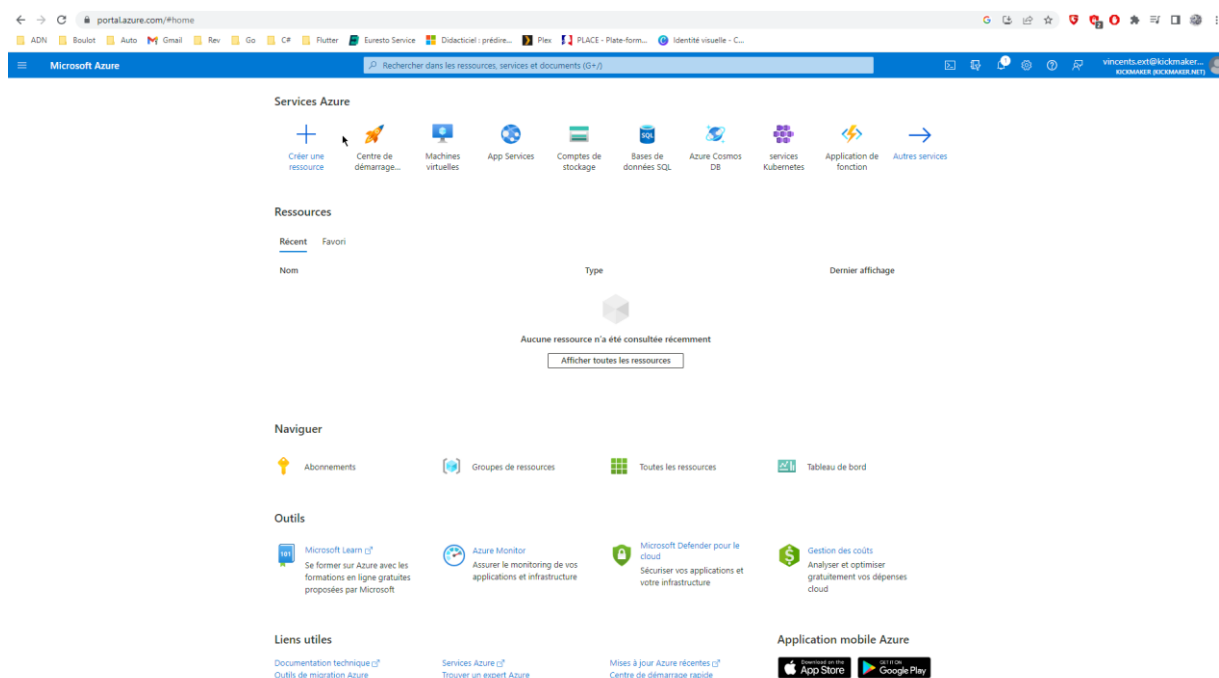
Pour exporter

vosre base de données, vous devez l'enregistrer sous forme de fichier .bacpac.

Cliquez sur *Suivant*, puis sur *Terminer*. Votre base de données est maintenant exportée vers le fichier .bacpac que vous avez nommé dans la fenêtre Parcourir. Cliquez sur *Fermer* à la fin de l'exportation.

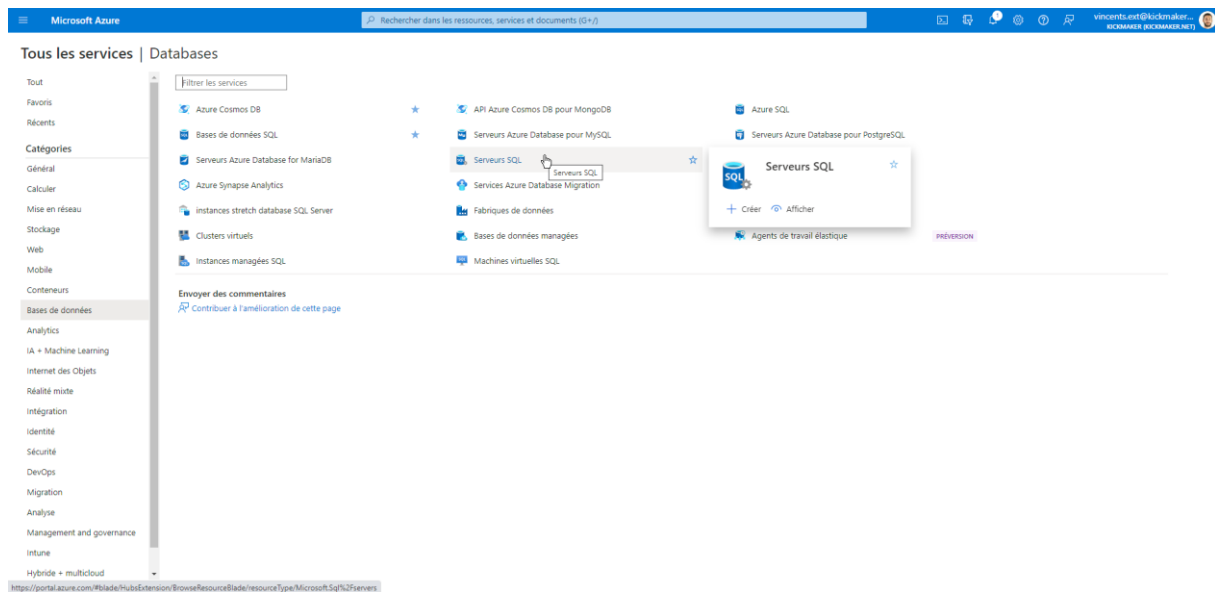
Créez une nouvelle instance SQL Server sur votre compte Azure

Il est temps de configurer un endroit pour héberger votre base de données sur Azure. Connectez-vous à votre compte Azure et cliquez sur *Tous les services* dans la liste des options à gauche de votre tableau de bord.



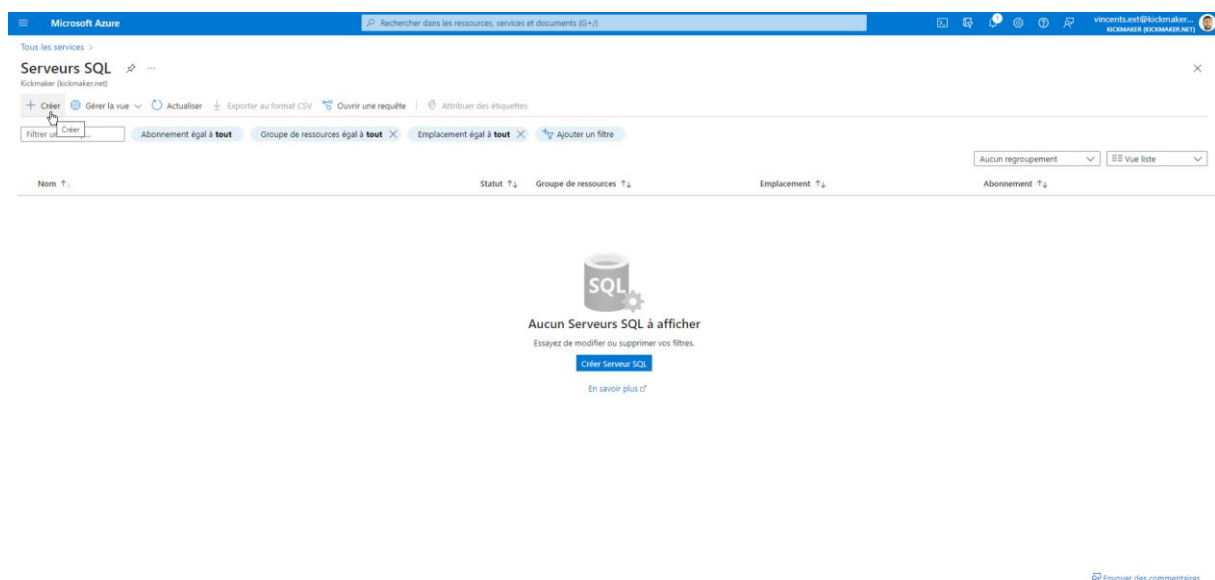
Configurez un emplacement pour héberger votre base de données sur Azure.

Sélectionnez *Bases de données*, puis *Serveurs SQL*.



Étape suivante : sélectionnez Bases de données > Serveurs SQL

Si vous avez déjà configuré un serveur, il sera répertorié ici et vous pourrez l'utiliser pour héberger votre base de données. Sinon, cliquez sur le bouton *Créer* en haut de la page.



Si vous n'avez pas encore configuré de serveur, cliquez sur *Créer*.

Remplissez le formulaire, en créant un nouveau *Groupe de ressources* si nécessaire, puis cliquez sur *Créer*.

Remplissez le formulaire pour créer un nouveau serveur.

Lorsque le serveur a été déployé avec succès, un message de réussite s'affiche en haut à droite de l'écran. Vous pouvez alors cliquer sur le bouton *Actualiser* pour voir apparaître votre nouveau serveur.

Cliquez sur le nom du nouveau serveur, puis cherchez l'option *Mise en réseau* et cliquez dessus.

Cliquez sur Mise en réseau.

Dans le nouveau volet, cliquez sur *Ajouter une adresse IP cliente*, puis sur *Enregistrer*.

testsecure | Mise en réseau

Serveur SQL

Rechercher

Gestion des données

Sauvegardes

Bases de données supprimées

Groupes de basculement

Historique d'importation/exportation

Sécurité

Mise en réseau

Microsoft Defender pour le cloud

Transparent Data Encryption

Identité

Audit

Performances intelligentes

Paramétrage automatique

Recommandations

Supervision

Journaux

Automatisation

Accès au réseau public

Les points de terminaison publics autorisent l'accès à cette ressource via l'Internet en utilisant une adresse IP publique. Une applica toujours une autorisation d'accès appropriée. [En savoir plus](#)

Accès au réseau public

Désactiver

Réseaux sélectionnés

Les connexions provenant des adresses IP configurées dans la section Règles de pare-f autorisée. [En savoir plus](#)

Réseaux virtuels

Autorisez les réseaux virtuels à se connecter à votre ressource en utilisant des points de terminaison de service. [En savoir plus](#)

Ajouter une règle de réseau virtuel

Règle	Réseau virtuel	Sous-réseau	Plage d'adre...	État du point de...	Groupe de resso...	At
-------	----------------	-------------	-----------------	---------------------	--------------------	----

Règles de pare-feu

Autorisez certaines adresses IP Internet publiques à accéder à votre ressource. [En savoir plus](#)

Ajouter l'adresse IPv4 de votre client (176.133.116.11)

Ajouter une règle de pare-feu

Nom de la règle	Adresse IPv4 de début	Adresse IPv4 de fin
-----------------	-----------------------	---------------------

Cliquez sur Ajouter une adresse IP cliente avant de cliquer sur Enregistrer.

Cela ajoutera l'adresse IP de votre ordinateur à la liste des adresses IP autorisées à accéder au serveur et vous permettra de vous y connecter en utilisant SSMS.

Maintenant, dans la liste des options du serveur, cherchez *Propriétés* et cliquez dessus.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header with the Microsoft Azure logo and a search bar. Below the header, the breadcrumb navigation shows 'Accueil > Serveurs SQL > testsecure'. The main content area is titled 'testsecure | Propriétés' and shows the 'Propriétés' tab selected in the left-hand menu. The main content area displays various server details including status, name, location, and subscription information.

Statut	Statut
Disponible	

Nom du serveur	Nom du serveur
testsecure.database.windows.net	

Emplacement	Emplacement
West Europe	

Identité managée affectée par le système	Identité managée affectée par le système
Non configuré	

Connexion d'administrateur du serveur	Connexion d'administrateur du serveur
vincent.s	

Administrateur Active Directory	Administrateur Active Directory
Non configuré	

Groupe de ressources	Groupe de ressources
sql	

ID d'abonnement	ID d'abonnement
e21f9bc1-8628-4ea8-b87b-fec8d184b354	

Nom de l'abonnement	Nom de l'abonnement
Azure subscription 1	

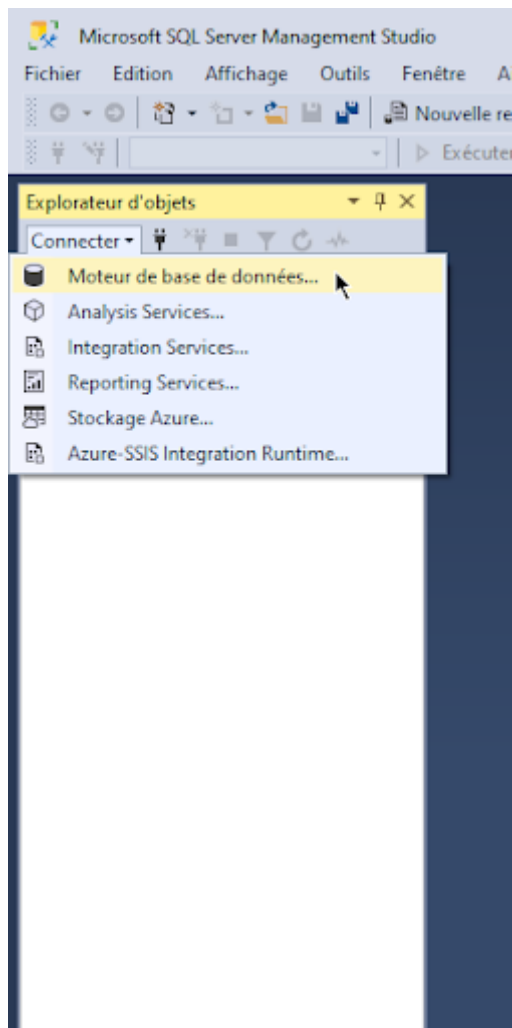
ID de ressource	ID de ressource
/subscriptions/e21f9bc1-8628-4ea8-b87b-fec8d184b354/resourceGroups/sql/providers/Microsoft.Sql/servers/testsecure	

Dans la liste des options du serveur, cliquez sur Propriétés.

Copiez le nom du serveur figurant dans le volet des propriétés, puis ouvrez SSMS.

Connectez-vous à votre instance Azure SQL Server

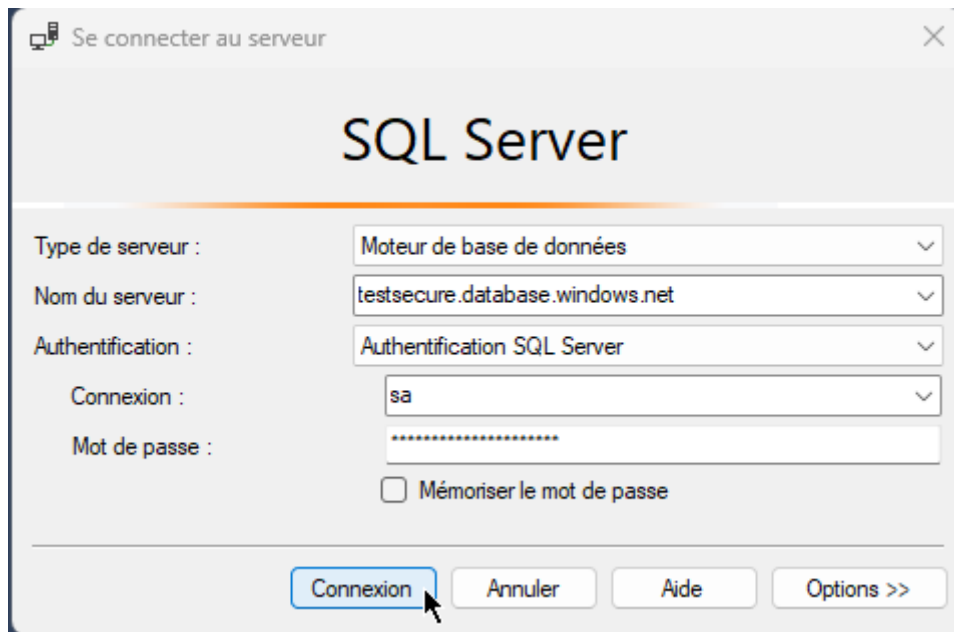
Dans l'Explorateur d'objets de SSMS, cliquez sur *Connecter* > *Moteur de base de données*.



Connecter > Moteur de base de données

Dans la fenêtre Se Connecter au serveur :

1. Collez le nom du serveur que vous avez copié depuis Azure dans le champ *Nom du serveur* ;
2. Choisissez *Authentication SQL Server* dans la liste déroulante *Authentication* ;
3. Saisissez le nom d'utilisateur et le mot de passe que vous avez créés dans Azure lorsque vous avez configuré le serveur ;
4. Cliquez sur *Connexion*.



La fenêtre Se connecter au serveur.

Votre serveur Azure devrait maintenant apparaître dans l'*Explorateur d'objets*.

Créez une nouvelle base de données à partir du fichier .bacpac exporté

Développez votre nouveau serveur Azure dans l'*Explorateur d'objets*, puis cliquez avec le bouton droit de la souris sur le dossier *Bases de données* et choisissez *Importer des données de la couche Applications*.

Dans la fenêtre Introduction, cliquez sur *Suivant*.

Dans la fenêtre Paramètres d'importation, cliquez sur le bouton *Parcourir* à côté de la case *Importer à partir du disque local* et cherchez le fichier .bacpac que vous avez créé plus tôt dans le chapitre. Sélectionnez le fichier et cliquez sur *Ouvrir*, puis cliquez sur *Suivant*.

Dans l'écran Paramètres de la base de données, vous devez configurer les ressources de performance que vous souhaitez utiliser pour votre base de données. Par défaut, il s'agit d'une base de données SQL Azure standard, avec une taille de 250 Go et un objectif de service S2. Il s'agit d'une option coûteuse, surtout pour un projet de tutorat, et nous allons donc apporter quelques modifications. L'option Basic est la moins chère, alors choisissez-la dans la liste déroulante *Édition de la base de données Microsoft Azure SQL*. La taille de la base de données passera à 2 Go et l'objet de service à *Basic*.

Configurez les ressources de performance pour la base de données.

Cliquez sur *Suivant*, puis sur *Terminer*. Lorsque le processus d'importation est terminé, cliquez sur *Fermer*. Votre base de données Watchlist se trouve maintenant sous le dossier *Bases de données* de l'*Explorateur d'objets*.

Ajoutez la nouvelle chaîne de connexion à votre projet

Vous devez maintenant indiquer à votre application d'utiliser la nouvelle base de données Azure au lieu de votre base de données locale. Pour ce faire, vous devez ajouter une nouvelle chaîne de connexion. Revenez à votre projet dans Visual Studio et ouvrez le fichier *appsettings.json*.

Vous pouvez ajouter plusieurs chaînes de connexion dans ce fichier, séparées par des virgules. Ajoutez une virgule à la fin de la ligne correspondant à "DefaultConnection", puis ajoutez une nouvelle chaîne de connexion pour votre base de données Azure avec les informations suivantes :

```
"AzureConnection": "Server=[nom du nouveau serveur];Database=[nom de la nouvelle base de données];
```

```
User ID=[nom d'utilisateur];Password=[mot de passe];MultipleActiveResultSets=true"
```

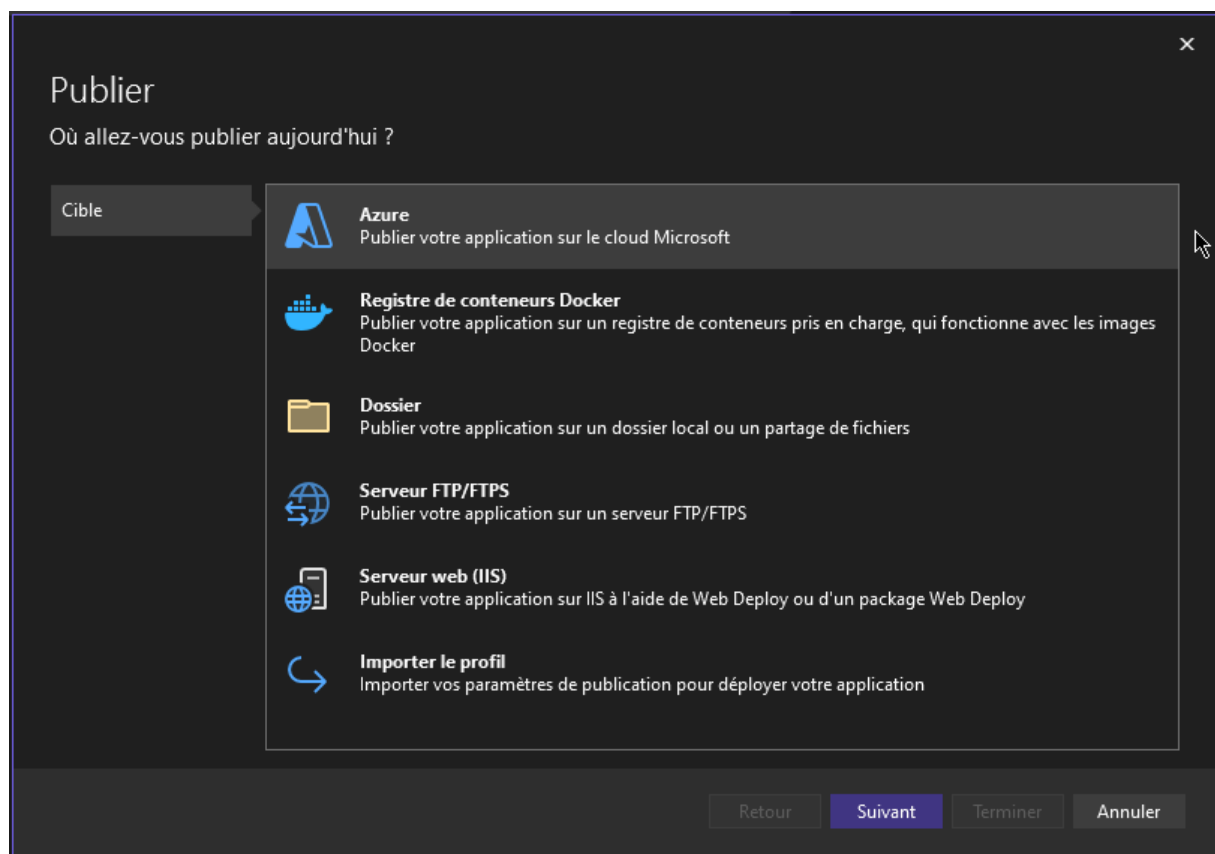
Notez que le paramètre `Trusted_Connection=True;` n'apparaît pas dans la nouvelle chaîne de connexion. Ce paramètre permet aux connexions locales (à des fins de développement) de contourner la sécurité par nom d'utilisateur/mot de passe pour une installation locale de SQL Server. Il ne fonctionnera pas sur Azure.

Ouvrez maintenant le fichier `Startup.cs`. Localisez le code qui contient la chaîne de connexion (vers la ligne 40) et remplacez `"DefaultConnection"` par `"AzureConnection"` .

Recompilez votre solution. Il est temps de publier votre application sur Azure.

Publiez votre application

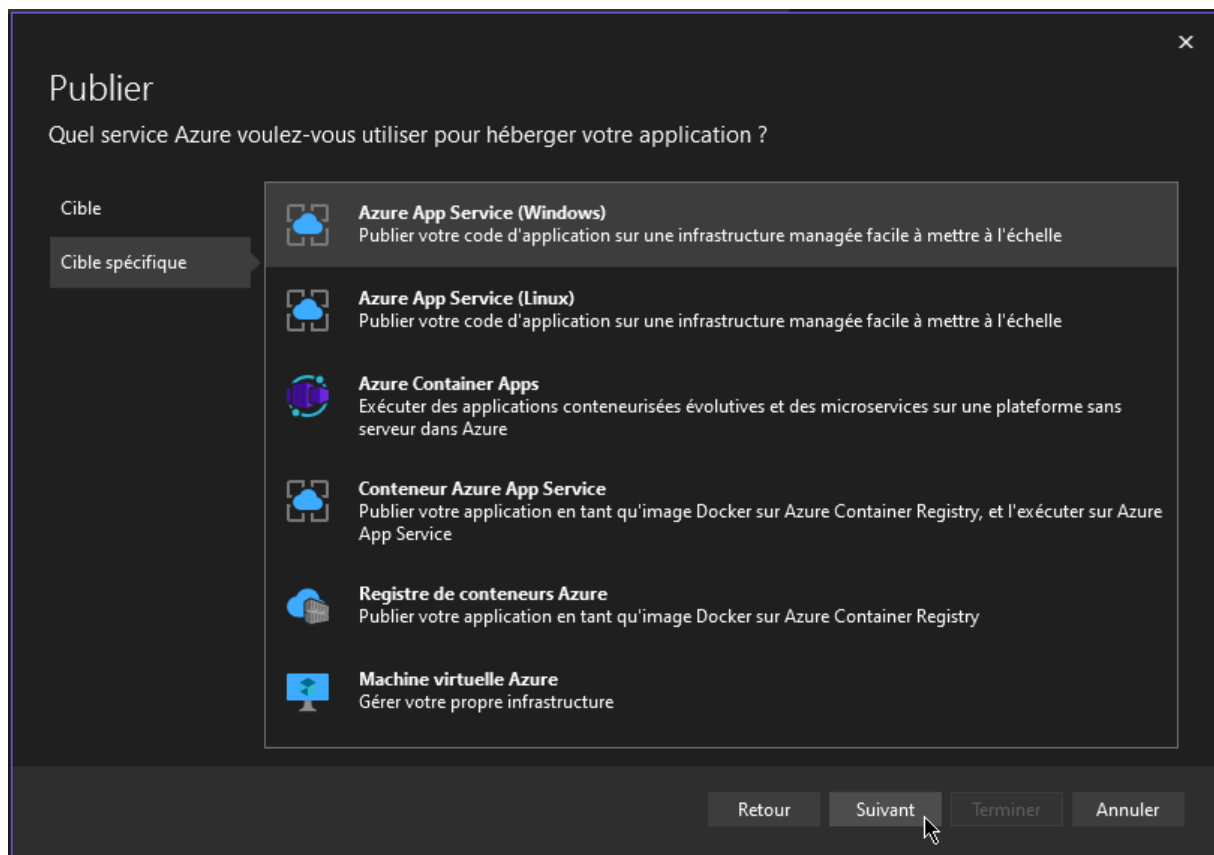
Maintenant que vous avez migré votre base de données Watchlist locale vers votre serveur SQL Azure, il est temps de publier votre application. Cette opération peut être réalisée entièrement dans Visual Studio. Pour commencer, cliquez avec le bouton droit de la souris sur le nom du projet dans l'Explorateur de solutions, puis choisissez *Publier*. L'écran suivant s'affiche :

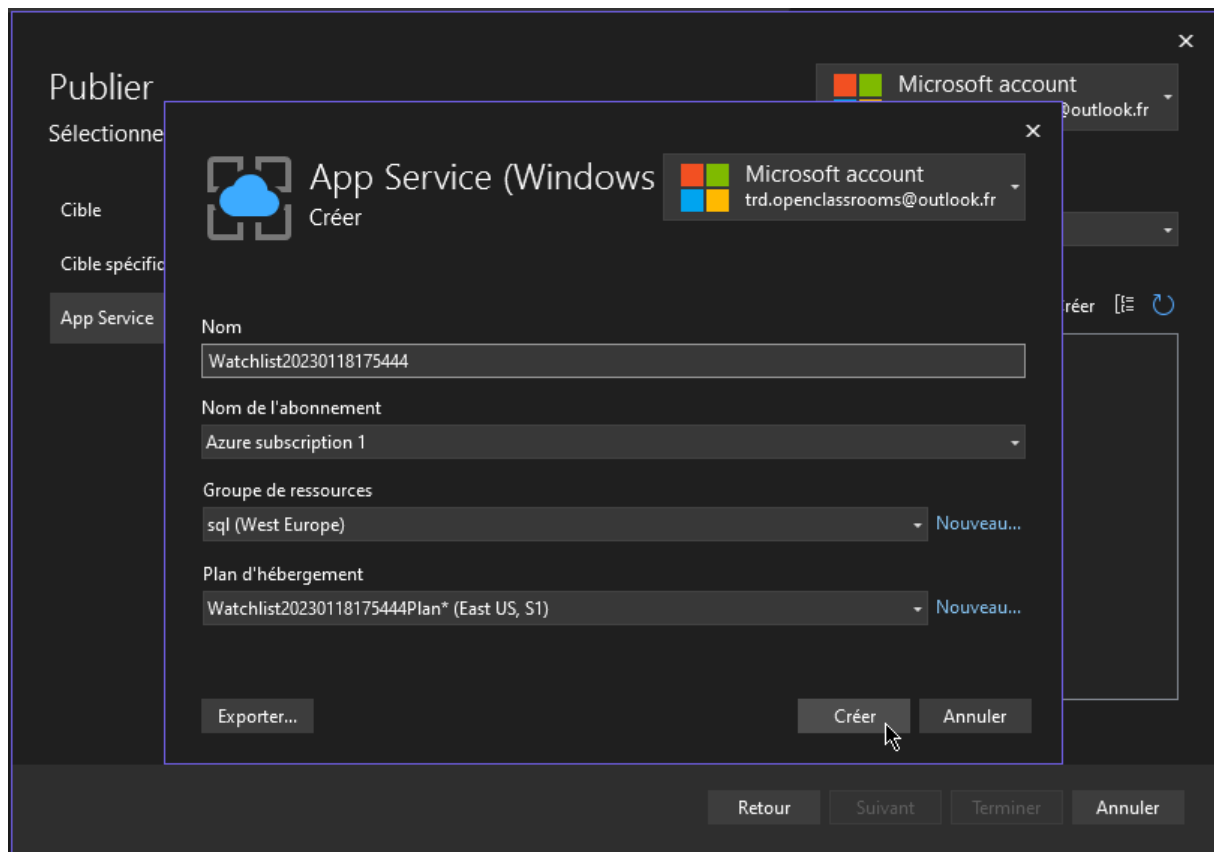


Fenêtre Choisir une cible de publication

Un **App Service Azure** est une version publiée et active de votre application. Puisque vous n'en avez pas encore créé, assurez-vous que la case d'option *Créer un nouveau* est sélectionnée, puis choisissez *Créer un profil* dans le bouton de liste

déroulante en bas de la fenêtre. Lorsque vous cliquez sur *Créer un profil*, la fenêtre suivante s'affiche :



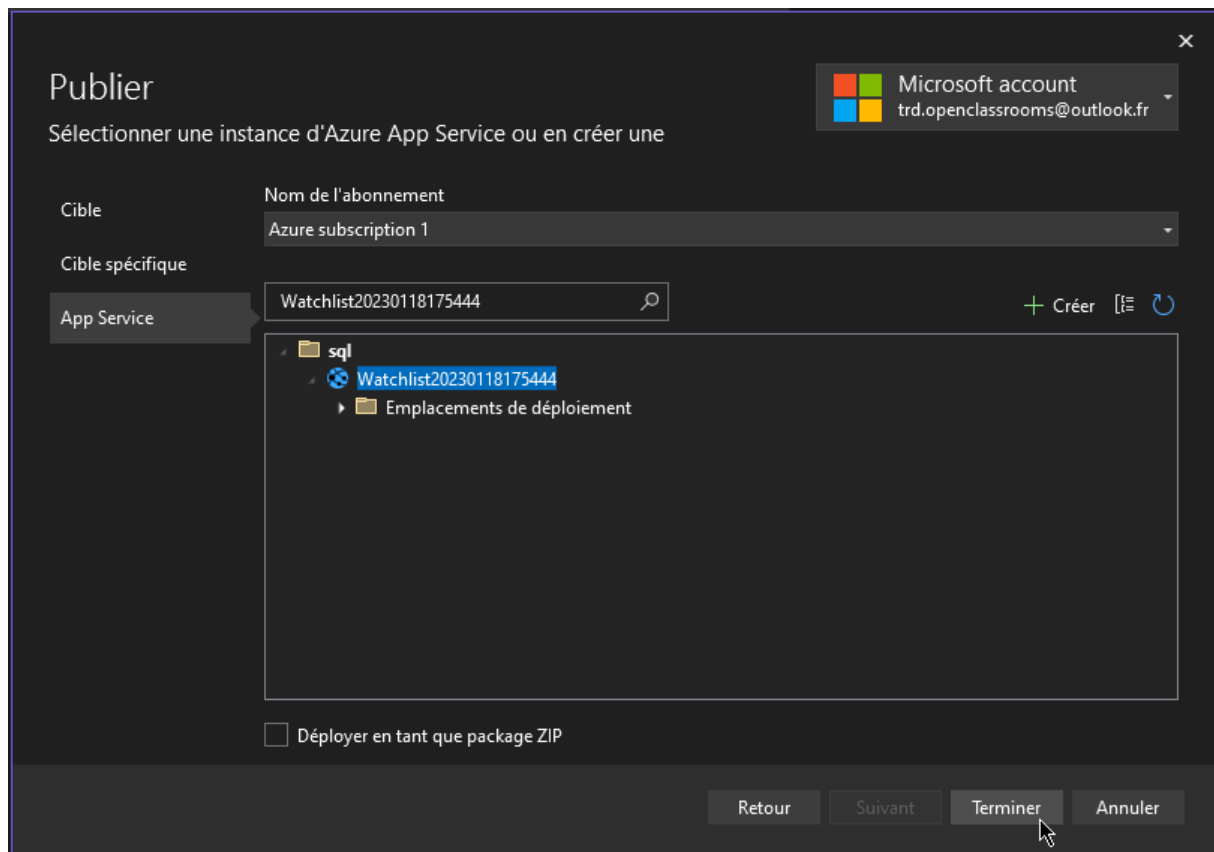


Créez un nouvel App Service.

Ensuite, sélectionnez le type d'abonnement. Vous utiliserez probablement l'option Paiement à l'utilisation. Choisissez alors le groupe de ressources et le plan d'hébergement. Si vous ne les avez pas encore créés, vous pouvez le faire en cliquant sur le lien *Nouveau* à côté de chaque élément. Enfin, installez Application Insights si vous souhaitez disposer d'outils de diagnostic d'application pour votre application en ligne. Je m'en passerai pour mon application.

Une fois que vous avez indiqué toutes les options requises, cliquez sur **Créer**.

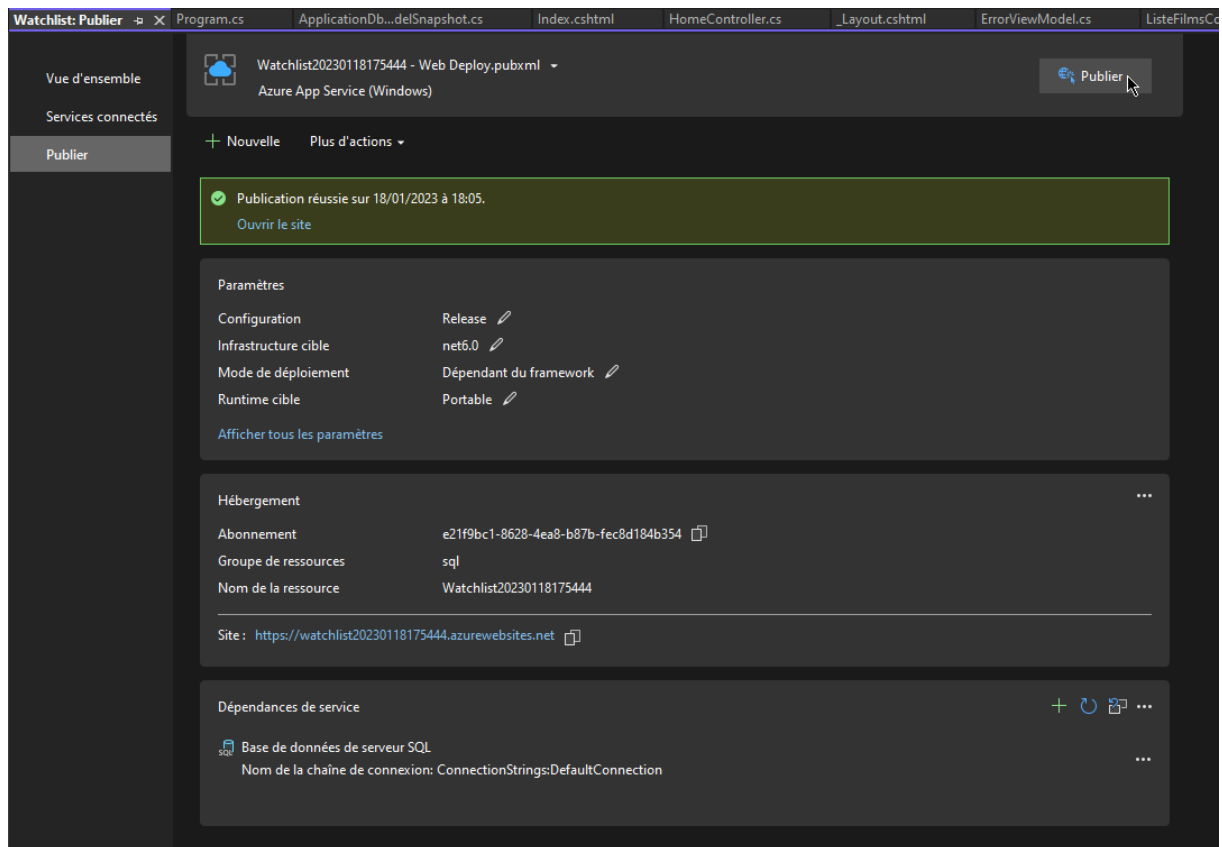
Votre application sera alors déployée sur Azure en tant qu'App Service, mais il vous reste une dernière étape à effectuer. Vous verrez la fenêtre suivante dans Visual Studio lorsque l'application sera prête à être publiée :



Avant de cliquer sur Publier, recherchez l'option Modifier.

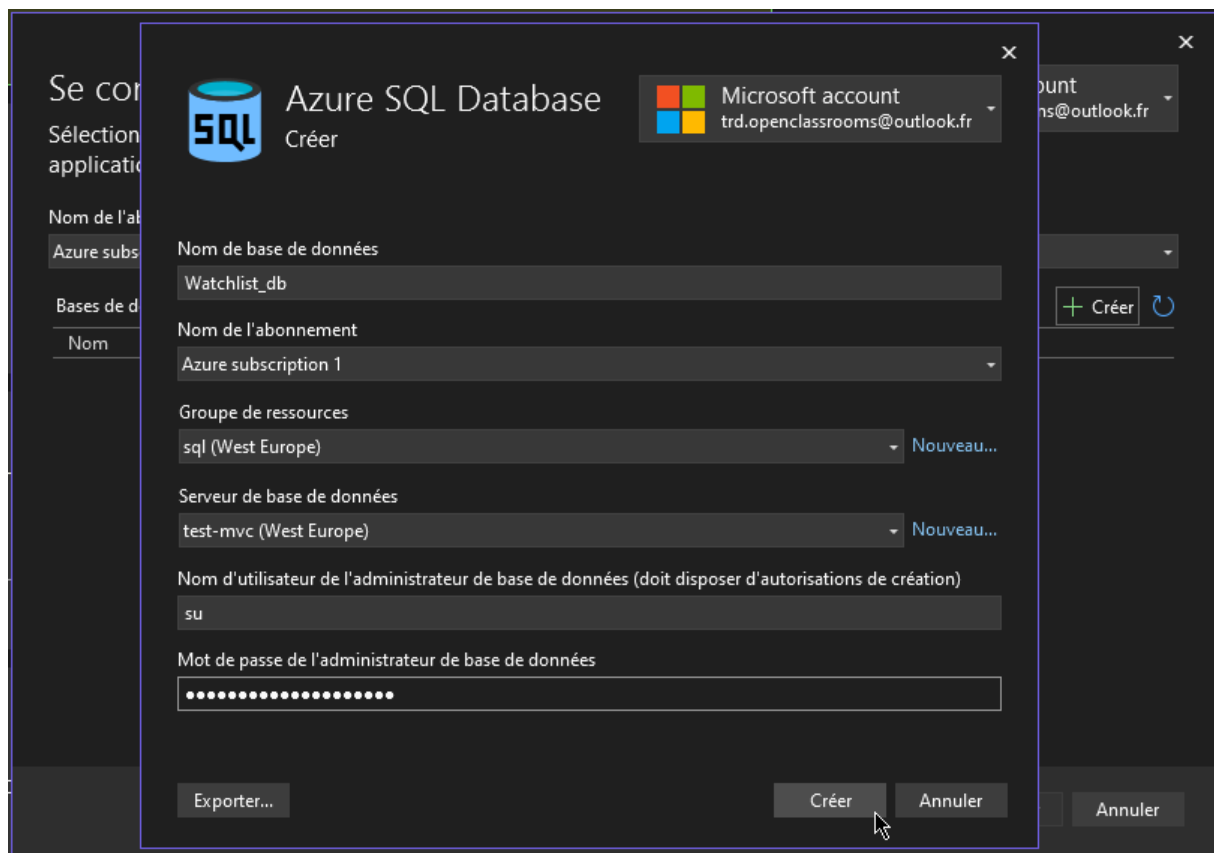
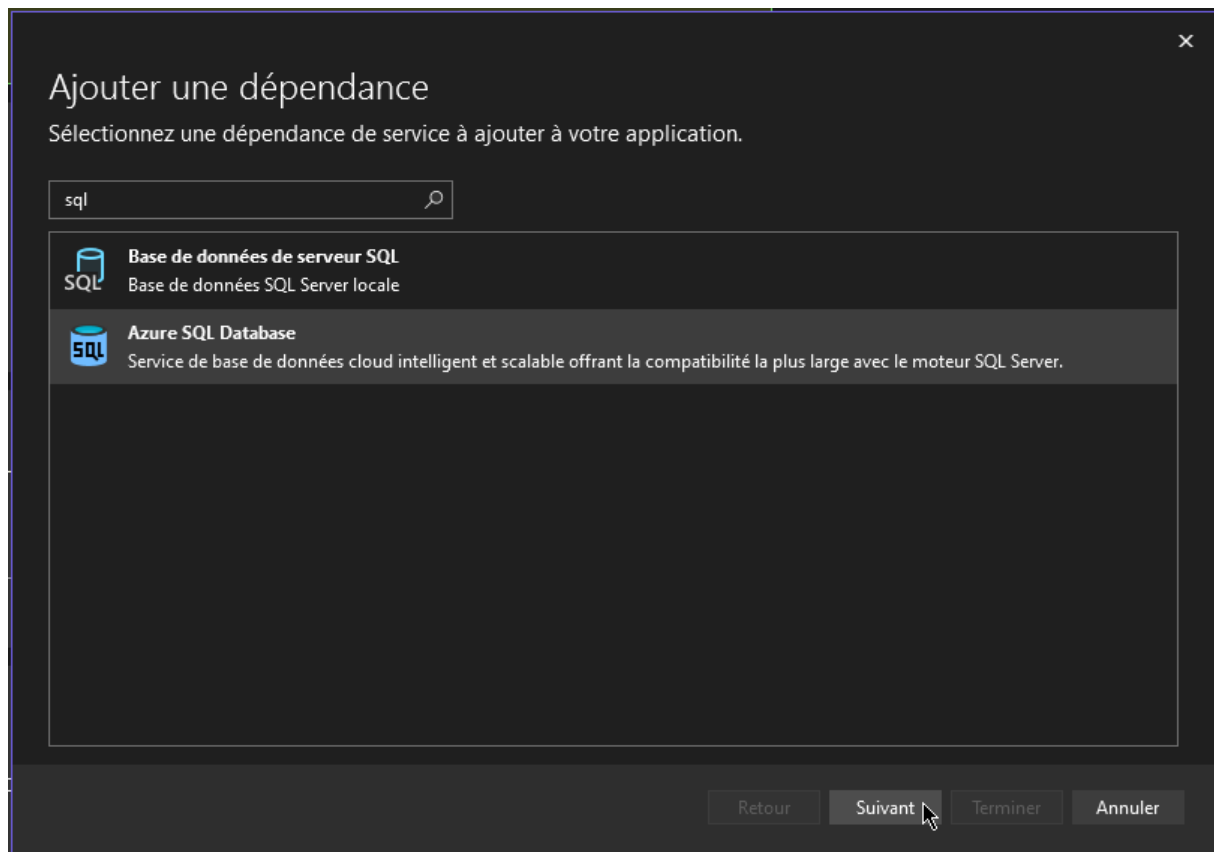
Vous avez maintenant une configuration de publication pour votre application, mais il vous reste encore quelques points à voir. Sous la liste déroulante de publication, quatre liens sont proposés. Cliquez sur *Modifier*.

Dans la fenêtre qui s'affiche, cliquez sur *Valider la connexion* pour vous assurer que l'App Service et votre configuration sont liés correctement. Lorsque la connexion est validée, cliquez sur *Suivant*.



Validez la connexion.

Dans la fenêtre Paramètres, sélectionnez l'option Débogage dans le menu déroulant Configuration, puis cliquez sur la flèche pointant vers le bas à côté de Bases de données pour développer les options de base de données. Cochez la case sous *AzureConnection* pour indiquer à l'application d'utiliser cette chaîne de connexion. Assurez-vous que la case *DefaultConnection* n'est pas cochée. Cliquez sur *Enregistrer*.



Modifiez les paramètres.

✓ Publication réussie sur 18/01/2023 à 18:05.
Ouvrir le site

Paramètres

Configuration

Release

Infrastructure cible

net6.0

Mode de déploiement

Dépendant du framework

Runtime cible

Portable

Afficher tous les paramètres

Hébergement

Abonnemente21f9bc1-8628-4ea8-b87b-fec8d184b354

Groupe de ressourcessql

Nom de la ressourceWatchlist20230118175444

Site : <https://watchlist20230118175444.azurewebsites.net>

Dépendances de service

Azure SQL Database: Watchlist_db

Nom de la chaîne de connexion: AzureWatchlistDb

✓ Connecté

Base de données de serveur SQL

Nom de la chaîne de connexion: ConnectionStrings:DefaultConnection

Cliquez maintenant sur *Publier*. Une fois le processus terminé, votre nouvelle application s'ouvrira dans votre navigateur par défaut.

Configurez un domaine personnalisé

L'attribution d'un domaine personnalisé à votre application est une bonne chose, et vous pouvez le faire pour n'importe quel Azure App Service. Cette opération est coûteuse et n'est donc pas requise pour ce cours. Toutefois, vous savez comment la réaliser si vous le souhaitez. Voici les éléments que vous devez prendre en compte avant d'obtenir un domaine personnalisé :

1. Mettez à niveau le niveau de tarification de votre App Service sur Azure. Le niveau de prix le plus bas qui autorise les domaines personnalisés est S1, qui coûte environ 74,40 \$ par mois.
2. Achetez le domaine souhaité auprès de votre fournisseur de domaines préféré. Le coût peut varier de quelques dollars à plusieurs milliers, en fonction du domaine.
3. Achetez un certificat SSL (Secure Sockets Layer) auprès de votre fournisseur de domaine ou d'un autre vendeur. L'option la moins chère coûte généralement entre 5 et 6 \$ par mois, ou entre 65 et 70 \$ par an.

4. Téléversez votre certificat SSL sur votre compte Azure et liez-le à votre App Service. Vous trouverez des instructions spécifiques dans la [documentation de Microsoft Azure ici](#).
5. Assignez et configurez le(s) domaine(s) souhaité(s) dans les propriétés de votre App Service.

Dans le but de limiter les dépenses des étudiants, nous n'irons pas plus loin pour ce cours. Cela vous donne au moins une idée de ce que vous devrez faire lorsque vous serez prêt à attribuer un domaine personnalisé à l'une de vos applications.

En résumé

Votre application Watchlist est en ligne !

- Dans ce chapitre, vous avez appris à utiliser SQL Server Management Studio (SSMS) pour migrer une base de données locale vers Azure et l'utiliser en ligne.
- Vous avez appris à publier une application web sur Azure.
- Vous avez découvert ce dont vous aviez besoin pour affecter un domaine personnalisé à un App Service Azure.

Testez votre application sur Azure

Ce processus devrait être simple. La base de données que vous avez utilisée pour tester votre application en local est maintenant en ligne, ainsi que votre application. À moins qu'il n'y ait des problèmes avec votre chaîne de connexion ou la modification que vous avez apportée au fichier *Startup.cs*, tout devrait fonctionner exactement comme en local.

Testez votre application en vous connectant à l'aide des informations d'identification que vous avez définies lors de votre inscription initiale. Votre liste de films devrait s'afficher, si vous en avez une. Testez les éléments du menu en cliquant sur chacun d'eux et vérifiez que la navigation fonctionne comme prévu.

Ensuite, ajoutez de nouveaux films à la base de données. Puis ajoutez-en quelques-uns à votre liste de films. Lorsque vous avez des films et une liste de films, évaluez quelques titres. Continuez à tester les fonctionnalités jusqu'à ce que vous soyez convaincu que tout fonctionne comme vous le souhaitez. Faites les corrections éventuellement nécessaires.

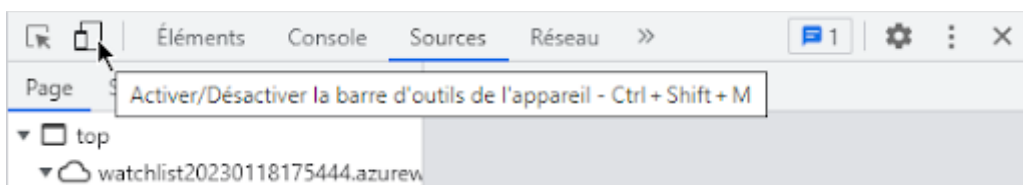
Finalisez votre application en ligne

La dernière étape de votre application consiste à corriger tous les problèmes d'affichage que vous avez pu remarquer pendant les tests. Comme votre application repose sur Bootstrap, elle est compatible avec les appareils mobiles. Je vous recommande de la

tester avec plusieurs tailles d'écran pour vous assurer que tout se passe comme vous le souhaitez.

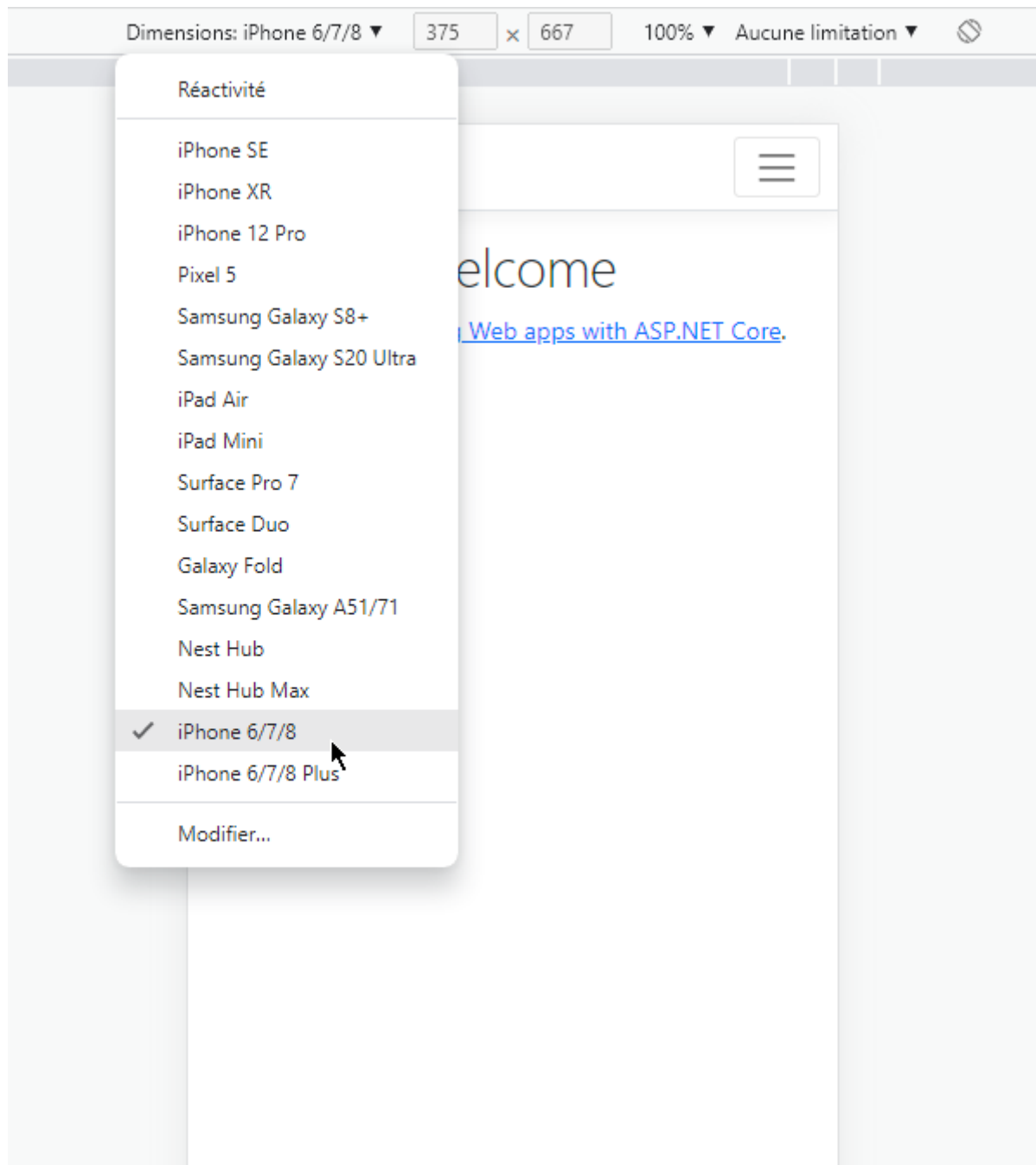
Il existe deux façons de procéder. Premièrement, vous pouvez redimensionner la fenêtre de votre navigateur et voir comment les pages se comportent. Cette méthode est tout à fait valable pour les tests initiaux, mais si vous souhaitez rendre votre application accessible au public, vous devez la tester à l'aide d'émulateurs d'appareils mobiles.

Heureusement, Google Chrome et d'autres navigateurs intègrent de tels émulateurs dans leurs fonctions de débogage. Dans Chrome, appuyez sur F12 pour faire apparaître l'environnement de débogage. Dans la barre de menu, sur le côté gauche, vous verrez une icône représentant des appareils mobiles. Il s'agit du bouton permettant d'activer les émulateurs. Vous pouvez également activer ce menu en appuyant sur Ctrl + Maj + M.



Bouton d'activation des émulateurs

Par exemple, l'image suivante montre le modèle Bootstrap que j'ai choisi s'affichant dans un émulateur pour les modèles iPhone 6/7/8 :



Modèle Bootstrap

Pour essayer différents émulateurs, sélectionnez-les dans le menu déroulant en haut à gauche du menu d'affichage des émulateurs.

En résumé

Dans ce chapitre, vous avez apporté la touche finale à votre application :

- Vous avez publié votre application en ligne sur Azure, et avez testé ses fonctionnalités en ajoutant des films et des notes personnelles à la base de données, ainsi qu'à votre liste de films.

- Enfin, vous avez testé le rendu de votre application pour plusieurs appareils à l'aide des émulateurs intégrés dans les outils de débogage de Google Chrome.

Quelle est la prochaine étape pour votre application ?

Vous avez réussi ! Vous avez suivi l'intégralité de ce cours sur la création d'applications web avec ASP.NET Core MVC. Votre application Watchlist est fonctionnelle, élégante et publiée sur Azure. Elle dispose de sa propre URL. Vous vous demandez peut-être ce que vous pouvez faire de plus.

Comme pour toutes les applications, vous pourriez lui ajouter d'innombrables fonctionnalités et possibilités. À moins de fixer un point auquel vous considérez qu'une application est complète et son cycle de vie terminé, c'est un processus sans fin. Voici une **liste de fonctionnalités** que vous pourriez envisager d'ajouter pour rendre votre application Watchlist encore plus utile :

- Ajouter les genres des films.
- Ajouter le tri et le filtrage à la liste globale des films, ainsi qu'aux listes de films des utilisateurs.
- Ajouter des attributs relatifs aux acteurs et à l'équipe technique des films.
- Ajouter la possibilité d'effectuer une recherche dans la liste complète des films (recherche par titre, année, producteur, réalisateur, acteur, etc.).
- Ajouter des critiques détaillées.
- Ajouter des rôles pour les utilisateurs (admin, etc.).
- Ajouter des comptes famille avec des profils individuels au sein de chaque famille.
- Afficher les vues CRUD sous forme de boîtes de dialogue modales au lieu de pages séparées.

Ce ne sont là que quelques possibilités classiques que vous pourriez envisager d'intégrer à votre application Watchlist. En continuant à travailler sur votre application, vous trouverez sans doute des idées que vous voudrez essayer d'implémenter. Cherchez des moyens d'améliorer vos applications en termes d'expérience globale pour vos utilisateurs, puis prenez le temps d'expérimenter et de découvrir. Le codage est toujours un exercice d'apprentissage par la pratique. Alors, ne lâchez rien ! Vous vous êtes bien débrouillé jusqu'à présent !