ASP.Net MVC

Pointeurs

Pour aller plus loin avec MVC :
Contrôleurs asynchrones, Debug avancé, Tests,
Publication

Méthodes asynchrones

- Principe
  - Utilise la Task Library de .Net 4.0 (VS 2010)
    - Nombreuses méthodes suffixées "Async" sont apparues depuis dans le framework

```
Task<string> HttpClient.GetStringAsync( string requestUri )
```

  - Intégré par la syntaxe C# / VB.Net (.Net 4.5, MVC 4, C# 5, VS 2012)
    - `async` : déclare une méthode qui retourne une tâche en cours d'exécution
    - `await` : pour ne pas attendre la terminaison d'un appel d'une méthode `async`
      - Le code qui suit l'invocation via `await` est placé dans une continuation qui ne sera exécutée que lorsque la tâche sera terminée
  - La signature d'une méthode `async` doit retourner un `Task<T>`
    - Le corps doit faire un `await` et retourner un `T`
    - C# s'occupe du reste

Méthodes asynchrones et Asp.Net MVC

- Nan !
  - Ça ne fait rien gagner
  - Ça complexifie mon code
  - Ça crée un goulot d'étranglement au niveau de la BD
  - C'est contagieux

Méthodes asynchrones et Asp.Net MVC

- Si !
  - Ça ne fait rien gagner ?
    - Ça libère au contraire un thread de IIS pendant le traitement d'une E/S asynchrone (requête de BD, par exemple)
    - Ça permet de lancer plusieurs traitements asynchrones en parallèle
    - Ça n'accélère pas le traitement de la méthode async, ça, c'est vrai
  - Ça complexifie mon code ?
    - Beaucoup moins depuis MVC 4
      - Pas trop dans les cas simples qui sont les plus courant
      - Plus dans les cas où on peut gagner beaucoup et où on joue avec la Task API
  - Ça crée un goulot d'étranglement au niveau de la BD ?
    - Pas trop probable car le pool de threads de IIS est plus stressé que le serveur de BD qui supporte mieux les requêtes parallèles
  - Contagieux ?
    - En général, oui, et il ne faut pas lutter contre !
    - Mais pas vraiment en Asp.Net MVC où les contrôleurs sont les clients finaux

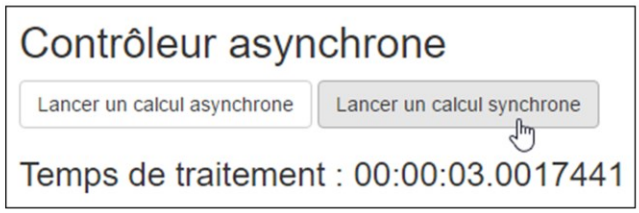
Contrôleur asynchrone - Exemple

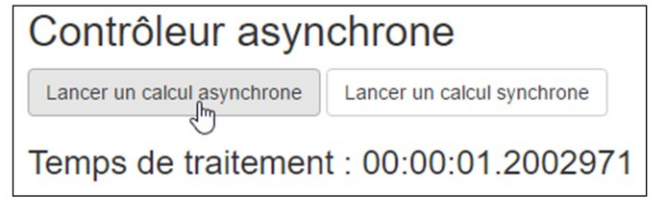
```
public ActionResult Index() {
  return View();
}
private void FaireUnGrosCalculSurUneRessource( int quelleRessource, int complexite ) {
  Thread.Sleep( complexite );
}
private async Task<bool> FaireUnGrosCalculSurUneRessourceAsync( int quelleRessource, int complexite ) {
  return await Task.Run( () => {
    Thread.Sleep( complexite );
    return true;                  ← Retourner void empêcherait de gérer les exceptions avec un simple try / catch
  } );
}
public ActionResult Calculer() {
  ViewBag.DebutTraitement = DateTime.Now;
  FaireUnGrosCalculSurUneRessource( 1, 1200 );
  FaireUnGrosCalculSurUneRessource( 2, 500 );
  FaireUnGrosCalculSurUneRessource( 3, 1100 );
  FaireUnGrosCalculSurUneRessource( 4, 200 );
  return View( "Index" );
}
public async Task<ActionResult> CalculerAsync() {
  ViewBag.DebutTraitement = DateTime.Now;
  var t1 = FaireUnGrosCalculSurUneRessourceAsync( 1, 1200 );
  var t2 = FaireUnGrosCalculSurUneRessourceAsync( 2, 500 );
  var t3 = FaireUnGrosCalculSurUneRessourceAsync( 3, 1100 );
  var t4 = FaireUnGrosCalculSurUneRessourceAsync( 4, 200 );
  await Task.WhenAll(new Task[] { t1, t2, t3, t4 });
  return View( "Index" );
}
```

Contrôleur asynchrone - Exemple

- Exemple - Vue

```
@{ ViewBag.Title = "Contrôleur asynchrone"; }
<h2>@ViewBag.Title</h2>
@using( Html.BeginForm( "CalculerAsync", "ExempleAsync", FormMethod.Post, new { @class = "inline" } ) ) {
  <input type="submit" value="Lancer un calcul asynchrone" class="btn btn-default" />
}
@using( Html.BeginForm( "Calculer", "ExempleAsync", FormMethod.Post, new { @class = "inline" } ) ) {
  <input type="submit" value="Lancer un calcul synchrone" class="btn btn-default" />
}
@if( ViewBag.DebutTraitement != null ) {
  <h3>@($"Temps de traitement : {DateTime.Now - ViewBag.DebutTraitement}")</h3>
}
```





Contrôleurs asynchrones + EF

- Méthodes d'extension async
  - Dans l'espace de nom `System.Data.Entity`
  - Évite de faire trop de `Task.Run`
- Vos actions `async`
  - Il faut faire un `await` de toutes les méthodes `async` utilisées
    - Notamment un `return await AutreAction()` si on enchaine avec une autre action
  - Les redirections ne sont pas problématiques
    - La méthode async retourne un `Task<ActionResult>`
    - Vous devez retourner un simple `ActionResult`, le compilateur s'en arrange

Contrôleurs asynchrones + EF

- Exemple

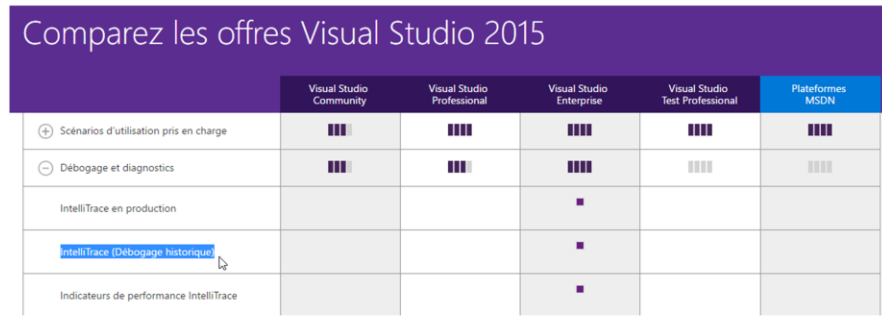
```
public class ExempleAsyncEFController : Controller {
  AFaireContext DC = new AFaireContext();
  public async Task<ActionResult> Index() {    ← Faire un await et retourner un ActionResult, C# s'occupe du reste
    var taches = await DC.Taches.ToListAsync();                ← Ne pas oublier le using System.Data.Entity !
    return View( "Index", taches );
  }
  public async Task<ActionResult> Ajouter( string nomTache ) { ← Ne pas oublier le using System.Threading.Tasks !
    if( nomTache == null )
      return RedirectToAction( "Index" );                                          ← C'est bien un ActionResult
    if( !string.IsNullOrWhiteSpace( nomTache ) ) {
      Tache nvTache = new Tache() { Nom = nomTache, Echeance = DateTime.Now, NbParticipants = 1 };
      DC.Taches.Add( nvTache );
      await DC.SaveChangesAsync();                                  ← Ne pas oublier le using System.Data.Entity !
    }
    return await Index();        ← Index() est de type Task<ActionResult>, await Index() est de type ActionResult
  }
  public async Task<ActionResult> Supprimer( Guid? id ) {
    if( id == null )
      return RedirectToAction( "Index" );
    DC.Entry( new Tache() { Id = id.Value } ).State = System.Data.Entity.EntityState.Deleted;
    await DC.SaveChangesAsync();                                  ← Ne pas oublier le using System.Data.Entity !
    return await Index();
  }
  protected override void Dispose( bool disposing ) { if( disposing ) DC.Dispose(); base.Dispose( disposing ); }
}
```

Intellitrace

☺ Très puissant

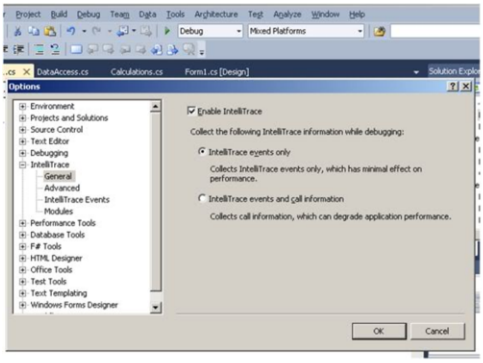
- Voir les événements passés – "*Live Debugging*"
  - Y compris les informations d'appel
- Enregistre les sessions – "*Non-Live Debugging*"
  - Puis possibilité debugger comme en "*live*"
  - Possibilité de sauver dans un fichier

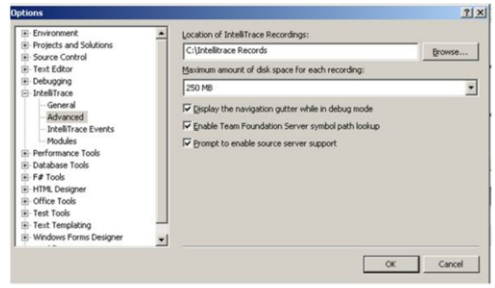
☹ Très payant

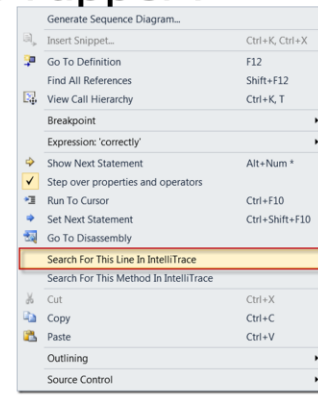


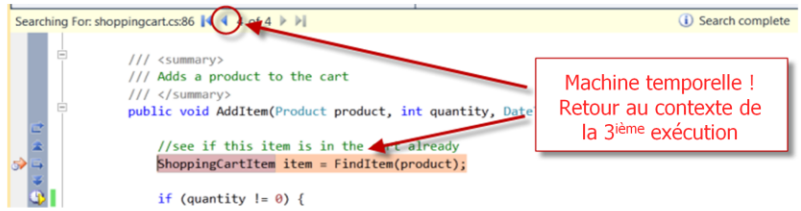
Intellitrace

- Connaître le nombre de fois où la méthode a été exécutée… et retrouver le contexte de l'appel !









Tests unitaires

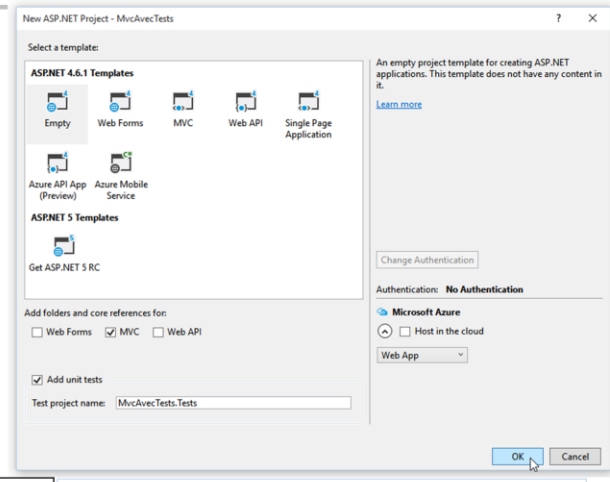
- NAN !
  - Ça prend du temps !
  - Ça me sort le nez du guidon !
  - Ça n'est pas nécessaire !
  - C'est compliqué !
  - Ça peut détecter de faux bugs !
  - Ça permet pas de tester tout le code !

Tests unitaires

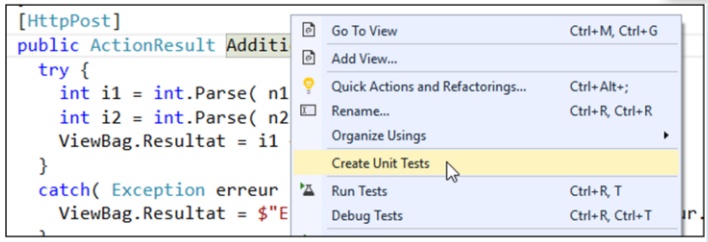
- SI !
  - Ça prend du temps ?
    - On gagne du temps : relancer les tests après une modification, pas refaire des tests
    - Test = ce que ça doit faire, ça aide à comprendre et à écrire la documentation
    - Aide à la réutilisation : copier/coller tests + code puis relancer les tests et faire en sorte que ça marche
  - Ça me sort le nez du guidon ?
    - Me donne confiance : le teste passe, je passe à autre chose le cœur léger !
    - Pas d'angoisse de la page blanche : commencer par écrire les tests, puis les faire fonctionner
  - Ça n'est pas nécessaire ?
    - En effet, pas plus que les tests manuels d'ailleurs, le programme est bien trop simple (et le programmeur bien trop costaud) pour envisager des tests ! ☺
  - C'est compliqué ?
    - Le code de test est souvent trivial
  - Ça peut détecter de faux bugs ? Ça permet pas de tester tout le code ?
    - Le code de test est souvent trivial: difficile d'y faire un bug mais facile de l'y corriger
    - Test + code, peu de risque d'avoir 2 bugs et de rater le test
    - "*Un test partiel lancé 100 fois vaut mieux qu'un test exhaustif pas écrit*" (proverbe tibétain)

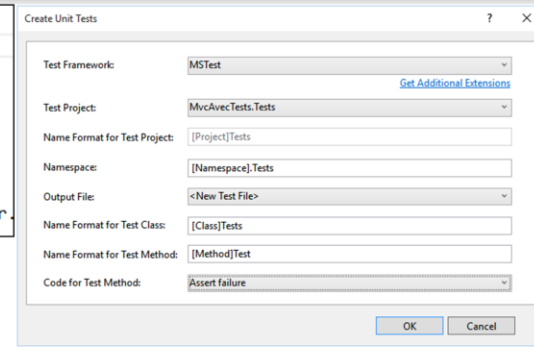
Tests unitaires avec Asp.Net MVC

- À la création du projet



- Ajout d'un test sur une action





HttpContext ? On s'en Mock !

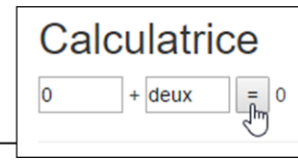
- On peut tester les contrôleurs mais `HttpContext`… ? ? ?
  - Il faut aiguiller `HttpContext.Current` vers un "faux"
    - Si, si, `HttpContext.Current { get; set; }`

```
public static HttpContext CreerUnHttpContextDeTest( string url = "http://pasnecessairementbesoindurl.com/" ) {
  var uri = new Uri( url );
  var httpContext = new HttpContext(
    new HttpRequest( string.Empty, uri.ToString(), uri.Query.TrimStart( '?' ) ),
    new HttpResponse( new StringWriter() ) );
  var nouvelleSession = new HttpSessionStateContainer(
    "id bidon",
    new SessionStateItemCollection(),
    new HttpStaticObjectsCollection(),
    timeout: 100, newSession: true, cookieMode: HttpCookieMode.UseCookies,
    mode: SessionStateMode.InProc, isReadonly: false );
  SessionStateUtility.AddHttpSessionStateToContext( httpContext, nouvelleSession );
  return httpContext;
}
[ . . . ]
public void LActionATesterTest() {
  HttpContext.Current = CreerUnHttpContextDeTest();
  [. . . ]
}
```

Contrôleur à tester

```
public class CalculatriceController : Controller {
  public ActionResult Index() {
    return View();
  }
  [HttpGet]
  public ActionResult Additionner() {
    return RedirectToAction( "Index" );
  }
  [HttpPost]
  public ActionResult Additionner( string n1, string n2 ) {
    try {
      int i1 = int.Parse( n1 );
      int i2 = int.Parse( n2 );
      ViewBag.Resultat = i1 + i2;
    }
    catch( Exception erreur ) {
      ViewBag.Resultat = 0;
    }
    return View( "Index" );
  }
}
```

```
@using( Html.BeginForm( "Additionner", "Calculatrice", FormMethod.Post, new { @class = "form-inline" } ) ) {
  @Html.TextBox( "n1", 0 )
  <text> + </text>
  @Html.TextBox( "n2", 0 )
  <input type="submit" value="=" />
  @ViewBag.Resultat
}
```

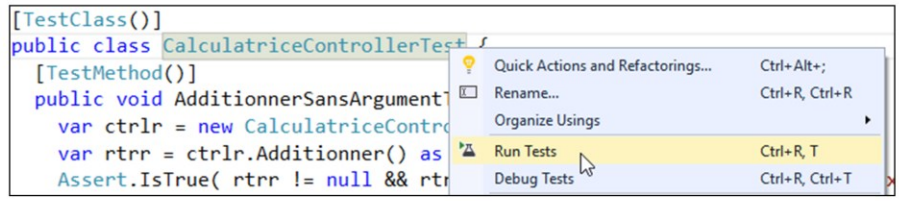


Projet de test associé

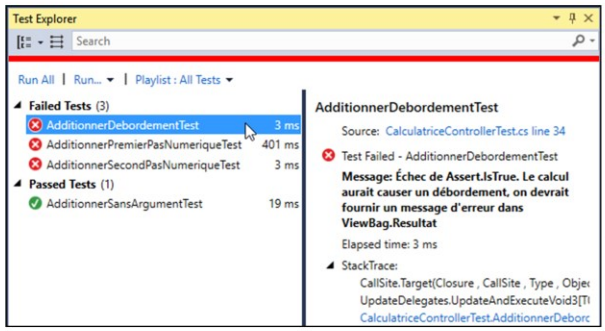
```
[TestClass()] public class CalculatriceControllerTest {
  [TestMethod()] public void AdditionnerSansArgumentTest() {
    var ctrlr = new CalculatriceController();
    var rtrr = ctrlr.Additionner() as RedirectToRouteResult;
    Assert.IsTrue( rtrr != null && rtrr.RouteValues["action"].Equals( "Index" ),
                   "L'action Additionner sans argument doit rediriger vers l'action index" );
  }
  [TestMethod()] public void AdditionnerPremierPasNumeriqueTest() {
    var ctrlr = new CalculatriceController();
    var vr = (ViewResult)ctrlr.Additionner( "truc", "1" );
    int bidon;
    Assert.IsTrue( vr.ViewBag.Resultat is string && !int.TryParse( vr.ViewBag.Resultat, out bidon ),
     "Le premier paramètre n'est pas un entier, on devrait fournir un message d'erreur dans ViewBag.Resultat" );
  }
  [TestMethod()] public void AdditionnerSecondPasNumeriqueTest() {
    var ctrlr = new CalculatriceController();
    var vr = (ViewResult)ctrlr.Additionner( "1", "truc" );
    int bidon;
    Assert.IsTrue( vr.ViewBag.Resultat is string && ! int.TryParse( vr.ViewBag.Resultat, out bidon ),
         "Le 2nd paramètre n'est pas un entier, on devrait fournir un message d'erreur dans ViewBag.Resultat" );
  }
  [TestMethod()] public void AdditionnerDebordementTest() {
    var ctrlr = new CalculatriceController();
    var vr = (ViewResult)ctrlr.Additionner( int.MaxValue.ToString(), int.MaxValue.ToString() );
    int bidon;
    Assert.IsTrue( vr.ViewBag.Resultat is string && !int.TryParse( vr.ViewBag.Resultat, out bidon ),
       "Le calcul aurait causer un débordement, on devrait fournir un message d'erreur dans ViewBag.Resultat" );
  }
}
```

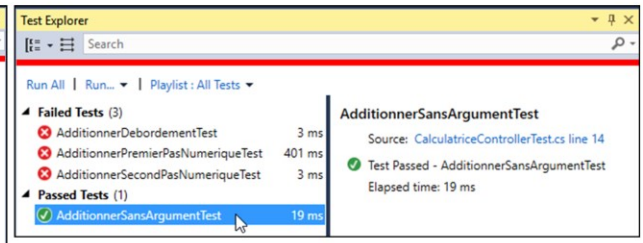
Exécuter les tests

- Lancer



- Examiner



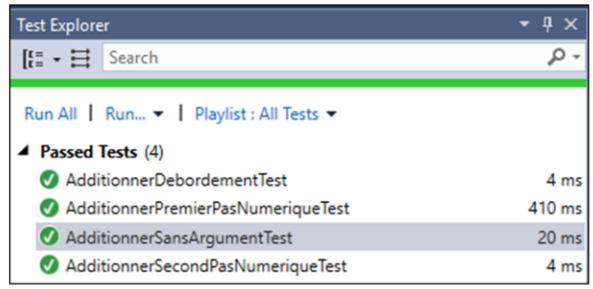


Corriger, ré-exécuter !

- Correction

```
[HttpPost]
public ActionResult Additionner( string n1, string n2 ) {
  try {
    int i1 = int.Parse( n1 );
    int i2 = int.Parse( n2 );
    ViewBag.Resultat = checked( i1 + i2 );
  }
  catch( Exception erreur ) {
    ViewBag.Resultat = $"Erreur {erreur.GetType().Name} : {erreur.Message}";
  }
  return View( "Index" );
}
```

- Ouf, tout va mieux



Ajout des tests à un projet existant

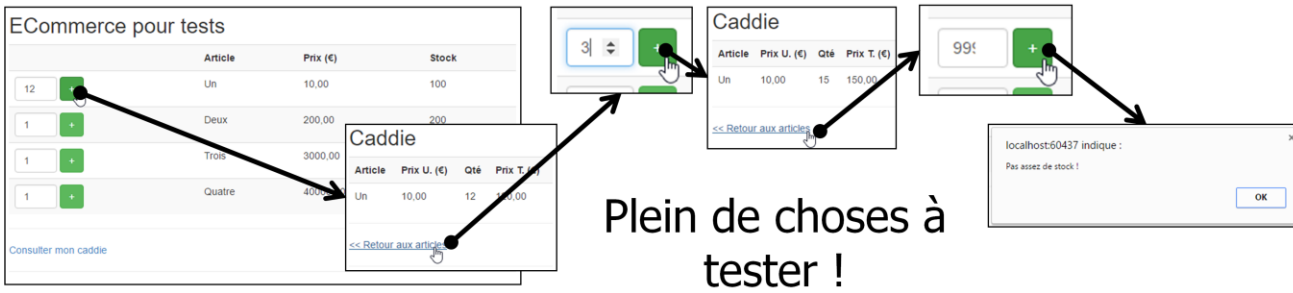
- Le projet existant doit être ajouté dans les références du nouveau projet de test
  - Le projet… et ses dépendances
    - Bibliothèques maison (couche d'accès aux données, …)
    - Assemblies .Net classiques (System.Web, Microsoft.CSharp, …)
    - Packages NuGet (MVC, Entity Framework, …)
- Et n'oubliez pas quelques using

```
using AccesAuxDonnees;
using MonProjetMVC;
using MonProjetMVC.Models;
using MonProjetMVC.Controllers;
using System.Web.Mvc;
```

Exercice

Total	30,6		
M+	M-	MR	MC
7	8	9	x
4	5	6	-
1	2	3	+
±	0	,	/

- Écrire des tests pour la calculatrice ringarde réalisée précédemment
  - Vérifier l'évolution du modèle dans tous les cas de figure
- Ou tester le bon fonctionnement du contrôleur de ECommerce founi



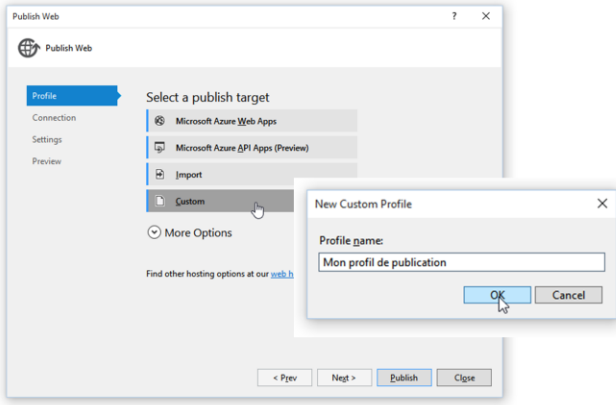
Plein de choses à tester !

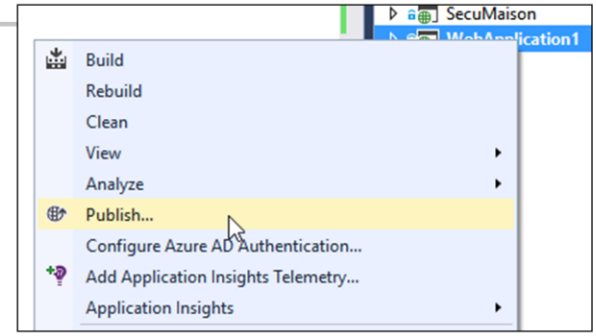
Publication

- Préparer IIS à recevoir l'application
  - Créer une application Web sur IIS
  - Les dépendances
    - Activer la prise en charge du framework cible avec la ligne de commande aspnet_regiis puis l'outil de gestion de IIS (Configurer le pool d'applications)
    - Se méfier des chaines de connexion, notamment de "Integrated Security=true" qui peut demander à changer l'identité utilisée par IIS
  - Sur une ferme n'importe quel serveur peut traiter la requête suivante…
    - Prendre en compte la gestion des sessions
    - Utiliser la même <machineKey> dans les Web.config
  - Publier (enfin)

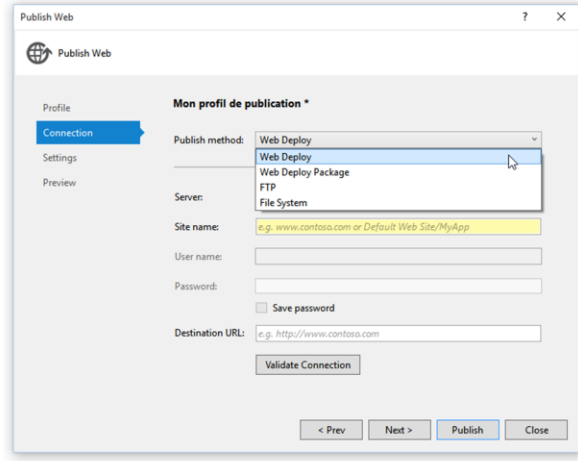
Publication

- Par un simple clic droit



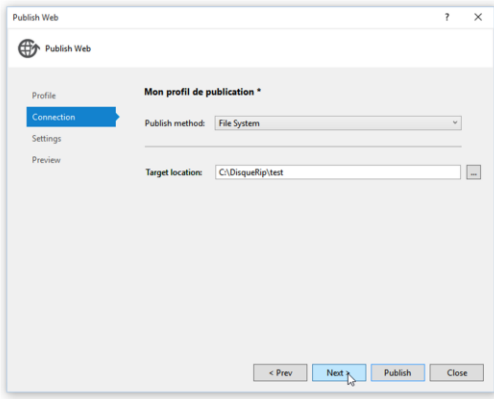


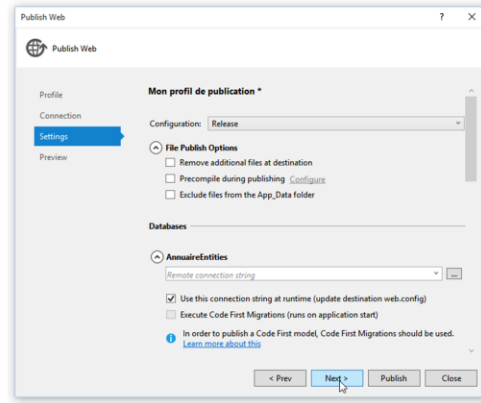
- Plusieurs méthodes



Publication

- Modification de la configuration



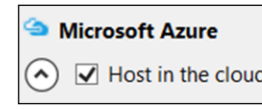


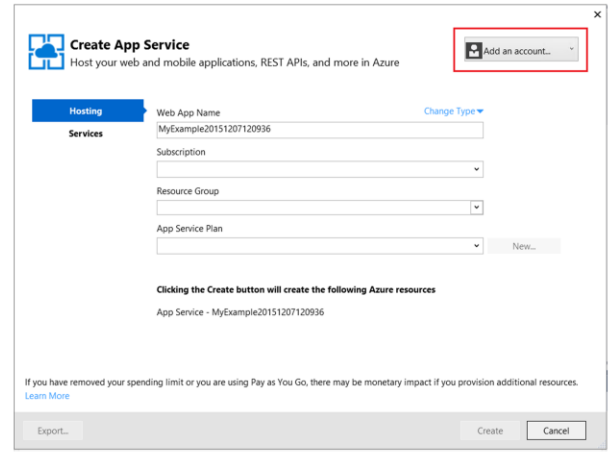
☹ La publication "file système" ne permet pas de gérer les chaines de connexion

- La publication click once des extensions VSTO :
  - https://msdn.microsoft.com/fr-fr/library/bb772100.aspx

Publication Azure

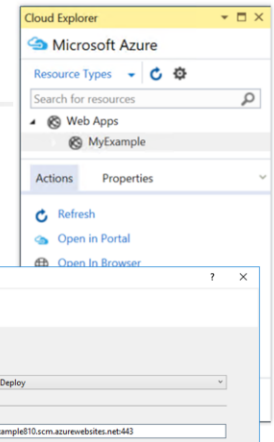
- Azure
  - À la création du projet
    - Web App Name : nom de domaine dans azurewebsites.net
    - Resource group : regroupe les ressources… BD, Appli, VM, …
    - Service Plan : localisation géographique du serveur, son type (mutualisé, …)

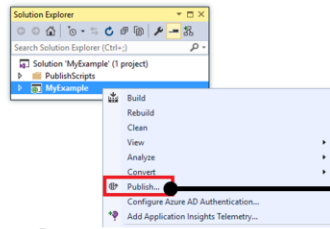


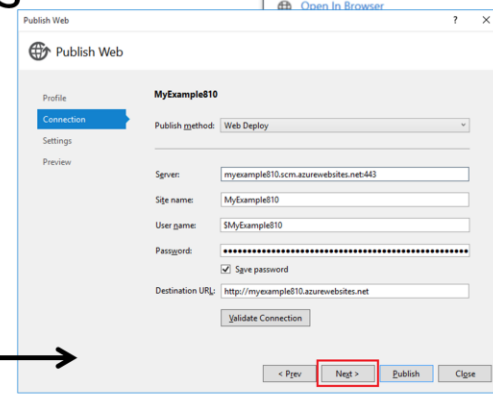


Publication Azure

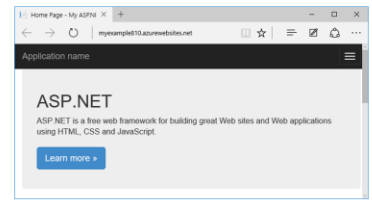
- Le "*cloud explorer*" de Visual Studio donne accès aux ressources du "*resource group*"
- En suite, la publication est standard







- L'appli est maintenant accessible dans le domaine azurewebsites.net



vNext !

- ASP.Net 5 totalement réécrit
  - Cible OSX & Linux
    - Écrire avec Sublime Text and WebStorm, plus nécessairment Visual Studio
  - Plus de Web Forms
    - Pas de réécriture de ASP.Net Web Forms en .Net 5
    - Possible de développer en ciblant le framework 4.6
  - Plus de Html Helpers ➔ Tag helpers
  - Plus de sous-contrôleurs (utilisés par Html.Action), remplacés par des classes dérivés de `ViewComponent`

```
@Component.Invoke( "UnViewComponent" )
```

  - Un seul type de contrôleur pour MVC & Web API
  - ASP.NET Dependency Injection Framework
  - xUnit.net

Asp.Net MVC 6

- Gestionnaires de package
  - Bower (distribue JQuery, AngularJS, Bootstrap, …)
  - NPM (distribue GruntJS)
- Utilitaire JavaScript GruntJS
  - Outil de build de ressources JS / CSS (concaténer, minifier, Less, TypeScript, Tests unitaires JS, …)
  - Nombreux plugins GruntJS
  - + templates pour faciliter l'utilisation de AngularJS

Asp.Net MVC 6

- Avant : Html helpers

```
@model MyProject.Models.Product
@using( Html.BeginForm() ) {
  <div>
    @Html.LabelFor( m => p.Name, "Name:" )
    @Html.TextBoxFor( m => p.Name )
  </div>
  <input type="submit" value="Create" />
}
```

- ASP.Net MVC 6 Tag Helpers

```
@model MyProject.Models.Product
@addtaghelper "Microsoft.AspNet.Mvc.TagHelpers"

<form asp-controller="Products" asp-action="Create" method="post">
  <div>
    <label asp-for="Name">Name:</label>
    <input asp-for="Name" />
  </div>
  <input type="submit" value="Save" />
</form>
```

  - Bien plus *"Html friendly"*