Rencontre 21

Tests

Bases de données et programmation Web

Sommaire

- Tests
  - Motivation
- Tests unitaires
- Tests d’intégration
- Tests fonctionnels (Pas abordé)

❖ Motivation

- ♦ Les tests sont généralement *ennuyeux* à faire, mais être habile avec la création de tests est une **compétence** que les entreprises recherchent.
  - Prenez le temps de bien comprendre les tests, car c'est une lacune qui se remarque rapidement par les employeurs. (Généralement, vous aurez souvent à faire des tests)
- ♦ Sans surprise, les tests **préviennent** des situations embarrassantes pendant la **mise en production** d'un système.
  - Avec les tests, on attrape des problèmes avant qu'ils soient mis en production.
    - C'est normal qu'un bug surgisse pendant la mise en production, mais pas *trente par jour*.
  - Faut-il tout tester ?
    - Non. (De toute façon, en pratique, c'est impossible / chronophage) Par exemple, les fonctions ou procédures triviales (ultra simples) n'ont pas forcément à être testées.
    - Prioriser les fonctionnalités critiques ou qui apportent plusieurs risques.

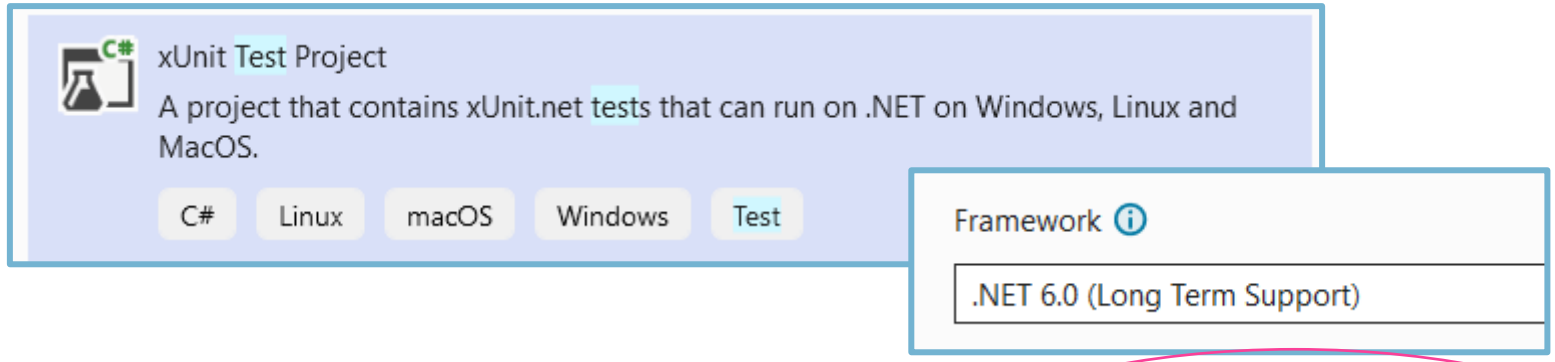
      (Exemples : **input d'un utilisateur** impliqué, interaction avec des **fichiers**, **conversion** de données, etc.)

❖ Motivation

- ♦ « Mon code fonctionne déjà très bien et j'ai tout testé, pourquoi est-ce que je ferais des tests pour ce qui marche déjà à merveille ? »
  - ○ Même si quelque chose **fonctionne parfaitement** présentement, à mesure qu'on ajoutera de nouvelles fonctionnalités (Ex : Itérations Agile), des choses qui fonctionnaient avant **pourraient briser** plus tard.
    - • Les tests ne nous aident pas seulement à vérifier que quelque chose fonctionne maintenant. Ils nous aident aussi à vérifier que quelque chose **continue** de bien fonctionner malgré des changements.
    - • C'est beaucoup plus facile de **trouver ce qui doit être modifié suite à une évolution de la BD** si nos tests nous indiquent ce qui a brisé.
  - ○ Bref, il ne faut pas sous-estimer l'utilité des tests à long terme. On les fait rouler à chaque fois que l'application **évolue**.

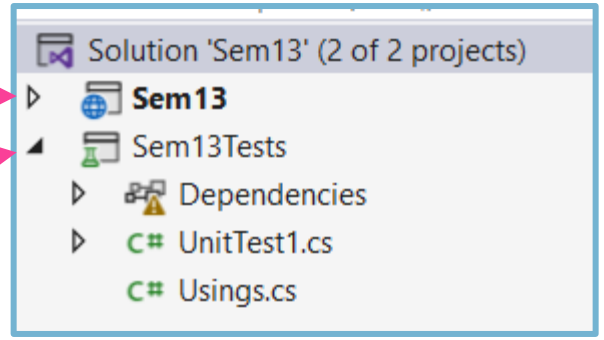
❖ Création d'un projet de test

- ♦ Dans la même solution que notre projet ASP.NET Core 6 MVC, on crée un projet avec le template xUnit Test Project :

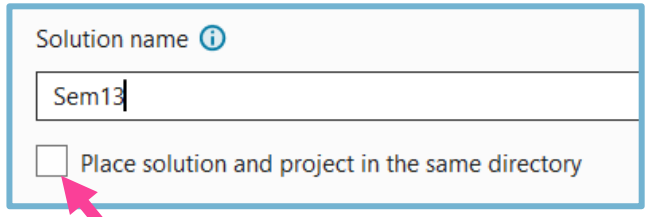


Projet principal

Projet de test



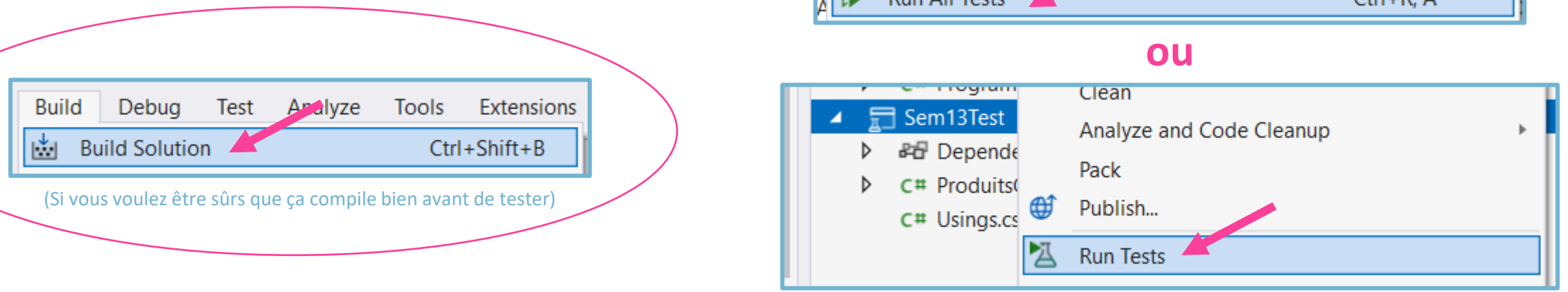
Comme on a plusieurs projets dans notre solution, il est préférable que la solution ne soit pas dans le même dossier.

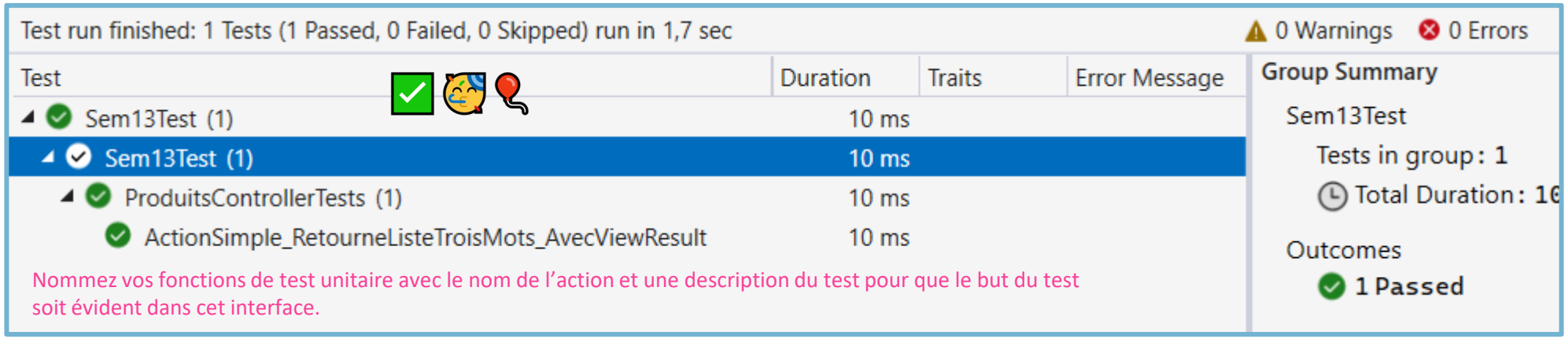


Non 😠❌

Tests

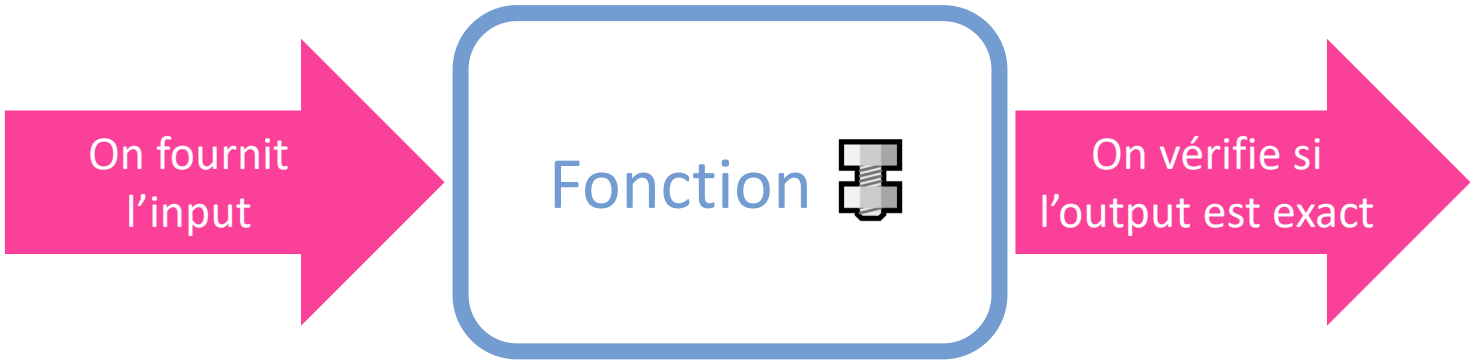
❖ Faire rouler les tests 🚗🚙





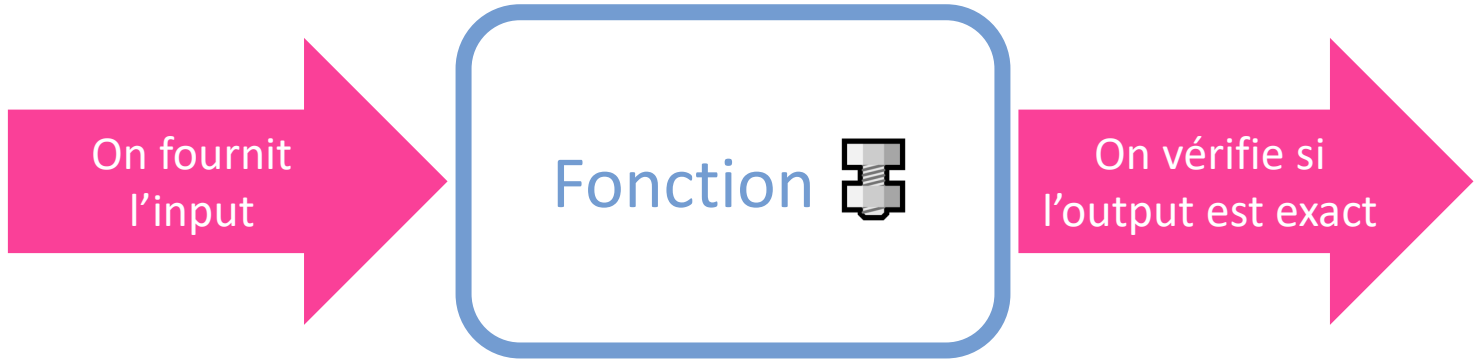
❖ Tests unitaires

- ♦ Le but des tests unitaires est généralement de tester **une seule fonction**, en l'isolant du reste du programme.
  - ○ Si elle a des dépendances ou des paramètres, on les lui fournit méticuleusement.
  - ○ Si elle effectue des actions ou retourne des données, on les analyse méticuleusement.
- ♦ Nous ne pourrons pas vraiment tester l'interaction avec la base de données avec les tests unitaires. (Seulement l'interaction avec le DbContext)



Tests unitaires

- La façon de construire un test se résume aux trois lettres suivantes :
  - A : Arrange (Arranger)
    - On met en place le contexte dans lequel le test doit se faire
    - Inclus la création des objets et variables nécessaires au test
  - A : Act (Agir)
    - On exécute le test, soit l'opération que nous voulons tester
    - On stocke le résultat (on peut aussi passer le résultat directement à l'étape suivante)
  - A : Assert (Auditer)
    - On compare le résultat obtenu (returned result) au résultat attendu (expected result)



Tests unitaires

- Dans les prochaines diapos on teste...
  - Ex 1 : Action sans paramètre (Contrôleur sans dépendance)
    - Retourne List<String>
  - Ex 2 : Action sans paramètre (Contrôleur avec DbContext)
    - Retourne List<Produit> ou Problem
  - Ex 3 : Action avec paramètre (Contrôleur avec DbContext)
    - Ajoute un Produit dans un DbSet si le ModelState est valide
  - Catalogue d'exemples d'assertions

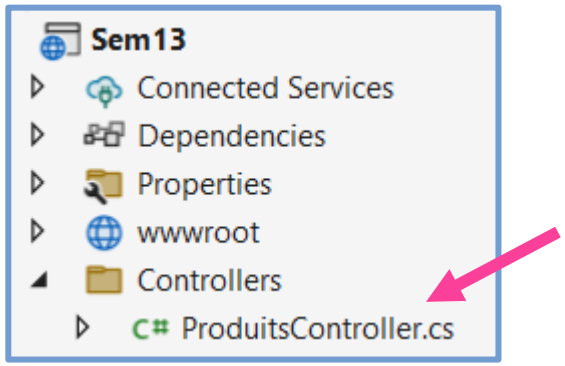
❖ Exemple 1 : Action sans dépendances ni paramètres

- ♦ On voudrait vérifier que la vue Razor recevra bel et bien une **liste** de trois string.

```
public IActionResult ActionSimple()
{
    List<string> mots = new List<string>() { "matelas", "jambon", "selfie stick" };
    return View(mots);
}
```

Le **contrôleur** qui contient cette action n'a aucune injection de dépendance.

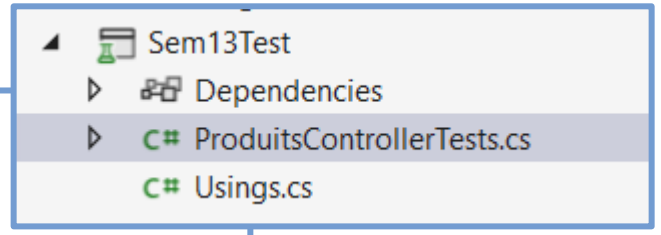
```
3 references
public class ProduitsController : Controller
{
    1 reference | 1/1 passing
    public ProduitsController(){}
```



Ceci est un bon exemple d'action qu'on ne devrait pas tester. Elle a très peu de chances de faillir.

❖ Exemple 1 : Action sans dépendances ni paramètres

♦ Fonction de test dans le projet xUnit Test.



Arrange

Act

Assert

```
public class ProduitsControllerTests
{
    [Fact]
    0 references
    public void ActionSimple_RetourneListeTroisMots_AvecViewResult()
    {
        // Instancier le contrôleur
        ProduitsController controller = new ProduitsController();

        // Appeler l'action à tester et stocker son output
        IActionResult resultat = controller.ActionSimple();

        // Attraper le "ViewResult" et vérifier que c'est bel et bien un ViewResult
        // (ce qui générerait la vue Razor)
        ViewResult viewResult = Assert.IsType<ViewResult>(resultat);

        // Attraper la liste de string dans le ViewResult (et vérifier que c'est une liste de string)
        List<string> model = Assert.IsAssignableFrom<List<string>>(viewResult.Model);

        // Vérifier qu'il y en a trois dans la liste (par exemple)
        Assert.Equal(3, model.Count);
    }
}
```

```
public IActionResult ActionSimple()
{
    List<string> mots = new List<string>() { "matelas", "jambon", "selfie stick" };
    return View(mots);
}
```

❖ Exemple 2 : Action sans paramètre, contrôleur avec DbContext

- ♦ Le défi : Pour pouvoir **instancier** le ProduitsController et appeler l'action Index() pour analyser son output, on doit « **simuler** » un Sem13Context pour le passer dans le constructeur de ProduitsController.
- ♦ En résumé, l'étape Arrange va :
  - ○ Créer un objet simulant un contexte de type Sem13Context
  - ○ Créer un controleur de type ProduitsController

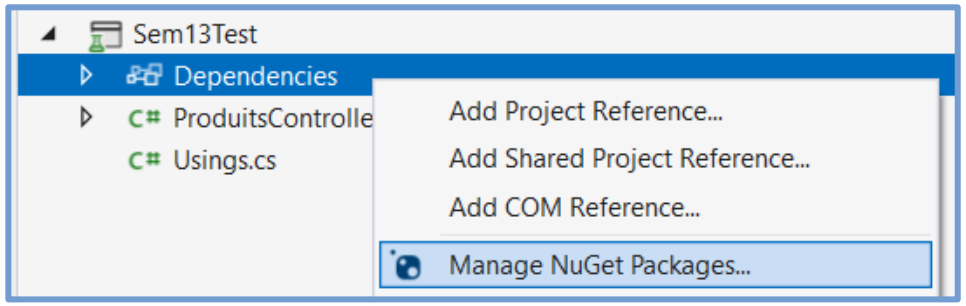
```
public class ProduitsController : Controller
{
    private readonly Sem13Context _context;

    0 references
    public ProduitsController(Sem13Context context)
    {
        _context = context;
    }
```

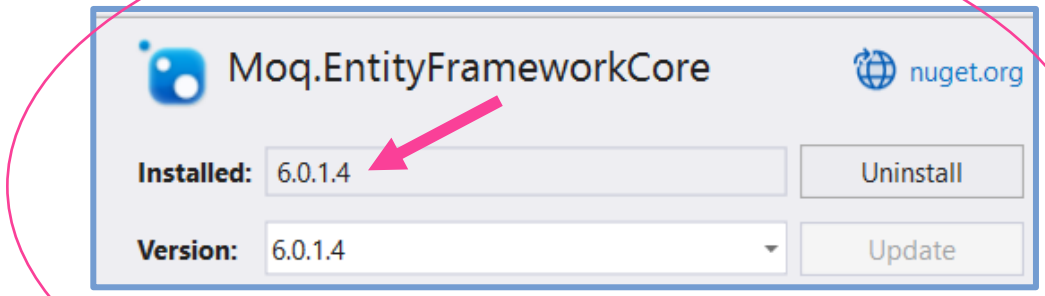
```
public async Task<IActionResult> Index()
{
    if(_context.Produits == null)
    {
        return Problem("L'ensemble Produits est null.");
    }
    return View(await _context.Produits.ToListAsync());
}
```

Si l'application dispose de services (couche d'accès aux données entre les contrôleurs et le DbContext), c'est plutôt le service concerné qu'on devrait *fabriquer*.

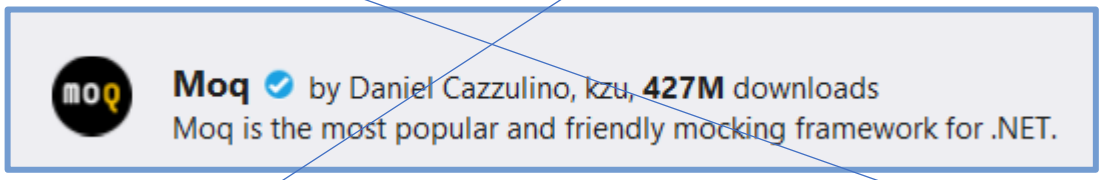
❖ Exemple 2 : Action sans paramètre, contrôleur avec DbContext

- ♦ Librairie « Moq » pour nous aider à simuler un DbContext





Nous utiliserons celle-ci pour sa grande simplicité



Celle-ci est très populaire, mais un peu plus compliquée pour ce dont on a besoin.

❖ Exemple 2 : Action sans paramètre, contrôleur avec DbContext

♦ Le DbContext à simuler

- Il est plutôt simple dans ce cas, il n'y a qu'un seul DbSet.
- En simulant le DbContext, nous fabriquerons un DbSet<Produit> à notre goût.

```
10 references
public partial class Sem13Context : DbContext
{
    0 references
    public Sem13Context(){}

    0 references
    public Sem13Context(DbContextOptions<Sem13Context> options) : base(options){}

    14 references | 1/2 passing
    public virtual DbSet<Produit> Produits { get; set; } = null!;
```

```
public partial class Produit
{
    [Key]
    [Column("ProduitID")]
    15 references | 1/2 passing
    public int ProduitId { get; set; }
    [StringLength(50)]
    16 references | 1/2 passing
    public string Categorie { get; set; } = null!;
    [StringLength(100)]
    16 references | 1/2 passing
    public string Nom { get; set; } = null!;
    [Column(TypeName = "money")]
    16 references | 1/2 passing
    public decimal Prix { get; set; }
    16 references | 1/2 passing
    public int QteStock { get; set; }
    14 references | 1/2 passing
    public bool EstDiscontinue { get; set; }
}
```

❖ Exemple 2 : Action sans paramètre, contrôleur avec DbContext

♦ Méthode de test (Si le DbSet<Produit> est **valide**)

```
public async Task<IActionResult> Index()
{
    if(_context.Produits == null)
    {
        return Problem("L'ensemble Produits est null.");
    }
    return View(await _context.Produits.ToListAsync());
}
```

```
[Fact]
0 references
public async Task Index_ListeProduitsValides()
{
    Mock<Sem13Context> mockContext = new Mock<Sem13Context>();
    List<Produit> produits = new List<Produit>() {
        new Produit() {
            ProduitId = 1, Categorie = "Meuble", EstDiscontinue = false,
            Nom = "Table brune", QteStock = 3, Prix = 399.99M
        },
        new Produit() {
            ProduitId = 2, Categorie = "Meuble", EstDiscontinue = false,
            Nom = "Chaise brune", QteStock = 29, Prix = 49.99M
        }
    };
    mockContext.Setup(x => x.Produits).ReturnsDbSet(produits);

    ProduitsController controller = new ProduitsController(mockContext.Object);
    IActionResult resultat = await controller.Index();

    ViewResult viewResult = Assert.IsType<ViewResult>(resultat);

    List<Produit> model = Assert.IsAssignableFrom<List<Produit>>(viewResult.Model);

    Assert.Equal(2, model.Count);
}
```

(Ce suffixe M est nécessaire car le prix est de type decimal)

Arrange

Act

Assert

❖ Exemple 2 : Action sans paramètre, contrôleur avec DbContext

♦ Méthode de test (Si le DbSet<Produit> est **valide**)

```
public async Task<IActionResult> Index()
{
    if(_context.Produits == null)
    {
        return Problem("L'ensemble Produits est null.");
    }
    return View(await _context.Produits.ToListAsync());
}
```

- Dans le premier bloc, le « Mock DbContext » (le DbContext simulé) est créé et on lui fournit une List<Produit> qui servira de DbSet<Produit>.

- new Mock<T>(), .Setup(), .ReturnsDbSet() sont des méthodes de la librairie Moq qui nous aident à créer des objets **simulés** à partir de la liste produits.

- Le reste de la méthode de test **instancie** le contrôleur (en lui passant le DbContext dans son **constructeur** !) pour pouvoir appeler l'action à tester.
- Les assertions sont similaires à celles dans l'exemple 1.

```
[Fact]
0 references
public async Task Index_ListeProduitsValides()
{
    Mock<Sem13Context> mockContext = new Mock<Sem13Context>();
    List<Produit> produits = new List<Produit>() {
        new Produit() {
            ProduitId = 1, Categorie = "Meuble", EstDiscontinue = false,
            Nom = "Table brune", QteStock = 3, Prix = 399.99M
        },
        new Produit() {
            ProduitId = 2, Categorie = "Meuble", EstDiscontinue = false,
            Nom = "Chaise brune", QteStock = 29, Prix = 49.99M
        }
    };
    mockContext.Setup(x => x.Produits).ReturnsDbSet(produits);

    ProduitsController controller = new ProduitsController(mockContext.Object);
    IActionResult resultat = await controller.Index();

    ViewResult viewResult = Assert.IsType<ViewResult>(resultat);

    List<Produit> model = Assert.IsAssignableFrom<List<Produit>>(viewResult.Model);

    Assert.Equal(2, model.Count);
}
```

(Ce suffixe M est nécessaire car le prix est de type decimal)

❖ Exemple 2 : Action sans paramètre, contrôleur avec DbContext

♦ Méthode de test (Si le DbSet<Produit> est **null**)

○ On pourrait aussi vouloir tester que le return Problem(...) marche bien

```
public async Task<IActionResult> Index()
{
    if(_context.Produits == null)
    {
        return Problem("L'ensemble Produits est null.");
    }
    return View(await _context.Produits.ToListAsync());
}
```

• On remarque cette fois-ci que le DbContext simulé est préparé, mais rien ne lui est fournit pour le DbSet<Produit>. (Ce qui le laisse null)

• Par la suite, on vérifie que Problem est bel et bien retourné. C'est un ObjetResult qui devrait être retourné (et non un ViewResult), dont le type spécifique est ProblemDetails. (Ça correspond à Problem)

Arrange
Act
Assert

```
[Fact]
0 references
public async Task Index_SetProduitsNull_Problem()
{
    Mock<Sem13Context> mockContext = new Mock<Sem13Context>();

    ProduitsController controller = new ProduitsController(mockContext.Object);
    IActionResult resultat = await controller.Index();

    ObjectResult objectResult = Assert.IsType<ObjectResult>(resultat);

    Assert.IsAssignableFrom<ProblemDetails>(objectResult.Value);
}
```

Tests unitaires

❖ Exemple 3 : Action avec paramètre, contrôleur avec DbContext

- ♦ Disons qu'on a une action pour créer un produit... et qu'on utilise un ViewModel pour passer les informations fournies par l'utilisateur de la Vue Razor à l'action.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create(CreationProduitVM cpvm)
{
    if (ModelState.IsValid)
    {
        Produit produit = new Produit()
        {
            ProduitId = 0,
            Categorie = cpvm.Categorie,
            Nom = cpvm.Nom,
            QteStock = cpvm.QteStock,
            Prix = cpvm.PrixEntier + cpvm.PrixDecimal / 100,
            EstDiscontinue = cpvm.EstDiscontinue
        };
        _context.Add(produit);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ModelState.AddModelError("", "Un ou plusieurs champs sont invalides.");
    return View(cpvm);
}
```

❖ Exemple 3 : Action avec paramètre, contrôleur avec DbContext

♦ Voici le Model et le ViewModel utilisés dans cet exemple :

```
public partial class Produit
{
    [Key]
    [Column("ProduitID")]
    14 references
    public int ProduitId { get; set; }
    [StringLength(50)]
    15 references
    public string Categorie { get; set; } = null!;
    [StringLength(100)]
    15 references
    public string Nom { get; set; } = null!;
    [Column(TypeName = "money")]
    15 references
    public decimal Prix { get; set; }
    15 references
    public int QteStock { get; set; }
    13 references
    public bool EstDiscontinue { get; set; }
}
```

L'usage d'un ViewModel pour cette insertion dans la BD n'était pas forcément nécessaire. (L'action aurait pu directement recevoir un Produit) Dans ce cas-ci, c'est par exemple pour pouvoir ajouter certaines contraintes personnalisées sur l'input fourni.

```
public class CreationProduitVM
{
    [Required(ErrorMessage = "Spécifiez un nom pour le produit.")]
    [StringLength(100, ErrorMessage = "Le nom du produit est trop long.")]
    1 reference
    public string Nom { get; set; } = null!;

    [Required(ErrorMessage = "Spécifiez un prix pour le produit")]
    [Range(0, int.MaxValue, ErrorMessage = "Le prix doit être supérieur ou égal à 0")]
    1 reference
    public int PrixEntier { get; set; }

    [Required(ErrorMessage = "Spécifiez un prix décimal pour le produit")]
    [Range(0, 99, ErrorMessage = "La portion décimale du prix doit être entre 0 et 99.")]
    1 reference
    public int PrixDecimal { get; set; }

    [Required(ErrorMessage = "Spécifiez une quantité initale pour le stock")]
    [Range(0, int.MaxValue, ErrorMessage = "La quantité doit être supérieure ou égale à 0")]
    1 reference
    public int QteStock { get; set; }

    1 reference
    public bool EstDiscontinue { get; set; }

    [Required(ErrorMessage = "Sélectionnez une catégorie pour le produit.")]
    1 reference
    public string Categorie { get; set; } = null!;

    0 references
    public List<SelectListItem> Categories { get; } = new List<SelectListItem>
    {
        new SelectListItem{ Value = "Divertissement", Text = "Divertissement"},
        new SelectListItem{ Value = "Électronique", Text = "Électronique"},
        new SelectListItem{ Value = "Meuble", Text = "Meuble"},
        new SelectListItem{ Value = "Vêtement", Text = "Vêtement"},
        new SelectListItem{ Value = "Bijou", Text = "Bijou"}
    };
}
```

❖ Exemple 3 : Action avec paramètre, contrôleur avec DbContext

♦ Ce qu'on veut tester

(On devra fournir un DbContext simulé et un CreationProduitVM à l'action)

• Si le CreationProduitVM passé est **valide**, le produit est-il bien **ajouté** dans le DbContext ? Sommes-nous bien **redirigés** vers l'action Index ?

• Si le CreationProduitVM passé est **invalide**, une **erreur** a-t-elle été ajoutée dans le ModelState ? La Vue Razor **Create** a-t-elle bien été retournée avec le CreationProduitVM passé en paramètre ?

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create(CreationProduitVM cpvm)
{
    if (ModelState.IsValid)
    {
        Produit produit = new Produit()
        {
            ProduitId = 0,
            Categorie = cpvm.Categorie,
            Nom = cpvm.Nom,
            QteStock = cpvm.QteStock,
            Prix = cpvm.PrixEntier + cpvm.PrixDecimal / 100,
            EstDiscontinue = cpvm.EstDiscontinue
        };
        _context.Add(produit);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ModelState.AddModelError("", "Un ou plusieurs champs sont invalides.");
    return View(cpvm);
}
```

❖ Exemple 3 : Action avec paramètre, contrôleur avec DbContext

♦ Méthode de test (si le ModelState est valide)

```
_context.Produits.Add(produit);
await _context.SaveChangesAsync();
return RedirectToAction(nameof(Index));
```

```
[Fact]
0 references
public async Task CreatePost_InsertionProduitValide()
{
    List<Produit> produits = new List<Produit>() {
        new Produit() {
            ProduitId = 1, Categorie = "Meuble", EstDiscontinue = false,
            Nom = "Table brune", QteStock = 3, Prix = 399.99M
        },
    };

    Mock<Sem13Context> mockContext = new Mock<Sem13Context>();
    Mock<DbSet<Produit>> mockDbSet = new Mock<DbSet<Produit>>();

    mockDbSet.Setup(x => x.Add(It.IsAny<Produit>())).Callback<Produit>(x => produits.Add(x));
    mockContext.Setup(x => x.Produits).Returns(mockDbSet.Object);

    CreationProduitVM cpvm = new CreationProduitVM()
    {
        Nom = "Divan 2 places", PrixEntier = 499, PrixDecimal = 99,
        QteStock = 3, EstDiscontinue = false, Categorie = "Meuble"
    };

    ProduitsController controller = new ProduitsController(mockContext.Object);
    IActionResult resultat = await controller.Create(cpvm);

    Assert.Equal(2, produits.Count);
    RedirectToActionResult redirectResult = Assert.IsType<RedirectToActionResult>(resultat);
    Assert.Equal("Index", redirectResult.ActionName);
}
```

Arrange

Act

Assert

❖ Exemple 3 : Action avec paramètre, contrôleur avec DbContext

♦ Méthode de test (si le ModelState est valide)

```
_context.Produits.Add(produit);
await _context.SaveChangesAsync();
return RedirectToAction(nameof(Index));
```

- Oof ! C'est normal que le premier bloc semble intimidant. On **simule** un DbContext, mais cette fois-ci, pour être capable de tester si un élément a bien été **ajouté** dans le DbSet<Produit>, on doit **simuler** un DbSet<Produit>.

- Comme on ne peut pas .Add() quelque chose dans le **DbSet simulé**, on configure le comportement suivant : si un élément est .Add() dans le DbSet simulé, il est plutôt .Add() dans la liste produits, créée plus haut. (Comme ça on pourra vérifier cette liste)

- Par la suite, on prépare un CreationProduitVM valide, on instancie le contrôleur en lui passant le DbContext et on appelle son action.

- Finalement, on peut faire nos assertions. Y a-t-il bien 2 éléments dans la liste ? Avons-nous bel et bien déclenché une redirection ? Cette redirection mène-t-elle vers l'action nommée Index ?

```
[Fact]
0 references
public async Task CreatePost_InsertionProduitValide()
{
    List<Produit> produits = new List<Produit>() {
        new Produit() {
            ProduitId = 1, Categorie = "Meuble", EstDiscontinue = false,
            Nom = "Table brune", QteStock = 3, Prix = 399.99M
        },
    };

    Mock<Sem13Context> mockContext = new Mock<Sem13Context>();
    Mock<DbSet<Produit>> mockDbSet = new Mock<DbSet<Produit>>();

    mockDbSet.Setup(x => x.Add(It.IsAny<Produit>())).Callback<Produit>(x => produits.Add(x));
    mockContext.Setup(x => x.Produits).Returns(mockDbSet.Object);

    CreationProduitVM cpvm = new CreationProduitVM()
    {
        Nom = "Divan 2 places", PrixEntier = 499, PrixDecimal = 99,
        QteStock = 3, EstDiscontinue = false, Categorie = "Meuble"
    };

    ProduitsController controller = new ProduitsController(mockContext.Object);
    IActionResult resultat = await controller.Create(cpvm);

    Assert.Equal(2, produits.Count);
    RedirectToActionResult redirectResult = Assert.IsType<RedirectToActionResult>(resultat);
    Assert.Equal("Index", redirectResult.ActionName);
}
```

❖ Exemple 3 : Action avec paramètre, contrôleur avec DbContext

♦ Méthode de test (si le ModelState est invalide)

```
ModelState.AddModelError("", "Un ou plusieurs champs sont invalides.");
return View(cpvm);
```

- Ici, on a fourni un CreationProduitVM **invalide**, (QteStock est négatif...) mais remarquez qu'on doit nous-mêmes glisser une erreur dans le ModelState. (La validité du CreationProduitVM n'est pas testée par le projet lors du test unitaire, **alors on doit identifier l'erreur nous-mêmes**...)

- Dans les assertions, on a un exemple où on vérifie le contenu du ModelState. Dans cette situation, on veut vérifier si un message précis a été ajouté pour la clé "". (La clé vide sert à configurer une erreur générale à tout le Model)

- On a aussi vérifié si le Model envoyé à la vue est un CreationProduitVM, bien entendu.

```
[Fact]
0 references
public async Task CreatePost_InsertionProduitInvalide()
{
    Mock<Sem13Context> mockContext = new Mock<Sem13Context>();

    CreationProduitVM cpvm = new CreationProduitVM()
    {
        Nom = "Divan 2 places", PrixEntier = 499, PrixDecimal = 99,
        QteStock = -7, EstDiscontinue = false, Categorie = "Meuble"
    };

    ProduitsController controller = new ProduitsController(mockContext.Object);
    controller.ModelState.AddModelError("QteStock", "Range");

    IActionResult resultat = await controller.Create(cpvm);

    ViewResult viewResult = Assert.IsType<ViewResult>(resultat);
    Assert.IsType<CreationProduitVM>(viewResult.Model);
    ModelStateDictionary modelState = viewResult.ViewData.ModelState;
    Assert.True(modelState.ContainsKey(""));
    Assert.Equal("Un ou plusieurs champs sont invalides.", modelState[""].Errors[0].ErrorMessage);
}
```

A

A

A

Catalogue d’exemples d’assertions

- Ici, nous allons voir plusieurs exemples d’assertions ainsi que les actions les plus couramment entreprises en fonction du type de retour attendu

❖ Catalogue d'exemples d'assertions

- ♦ Les actions retournent généralement un IActionResult, la première étape est donc souvent de l'attraper en appelant l'action à tester.
- ♦ Ensuite, le IActionResult retourné peut correspondre à plusieurs types d'objets et on vérifie généralement si c'est le type attendu.

IActionResult **peut être...**

- ViewResult
- BadRequestResult
- RedirectToActionResult
- NotFoundResult
- ObjectResult
- ...

```
return View(await _context.Produits.ToListAsync());
return BadRequest();
return RedirectToAction(nameof(Index));
return NotFound();
return Problem("L'ensemble Produits est null.");
```

```
IActionResult resultat = await controller.Index();
```

suivi de ...

```
ViewResult viewResult = Assert.IsType<ViewResult>(resultat);
BadRequestResult badRequestResult = Assert.IsType<BadRequestResult>(resultat);
RedirectToActionResult redirectResult = Assert.IsType<RedirectToActionResult>(resultat);
NotFoundResult notFoundResult = Assert.IsType<NotFoundResult>(resultat);
ObjectResult objectResult = Assert.IsType<ObjectResult>(resultat);
```

Tests unitaires

```
return View(await _context.Produits.ToListAsync());
```

❖ Catalogue d'exemples d'assertions

♦ ViewResult

- Lorsqu'un ViewResult est retourné, on voudra généralement vérifier le Model envoyé à la vue Razor.

- On récupère le IActionResult retourné par l'action.
- Est-ce que c'est bel et bien un ViewResult ? (Une vue a-t-elle été retournée ?)
- Le model envoyé à la vue est-il un List<Produit> ?
- Y a-t-il bel et bien deux produits dans la liste ?

```
IActionResult resultat = await controller.Index();

ViewResult viewResult = Assert.IsType<ViewResult>(resultat);

List<Produit> model = Assert.IsAssignableFrom<List<Produit>>(viewResult.Model);

Assert.Equal(2, model.Count);
```

❖ Catalogue d'exemples d'assertions

♦ ViewResult

- Lorsqu'un ViewResult est retourné, on peut aussi vérifier le ModelState. (Généralement les erreurs)

Rappel : La clé correspond à la propriété du Model auquel le message d'erreur s'applique. La clé vide "" représente le Model en général pour les messages d'erreur globaux.

```
ModelState.AddModelError("Prix", "Blablabla.");
```

Clé — Message

```
ModelState.AddModelError("", "Erreur générale");
```

- On récupère le IActionResult retourné par l'action.
- Est-ce que c'est bel et bien un ViewResult ?
- On récupère le ModelState.
- Avec .ContainsKey(), on vérifie si une erreur quelconque a bel et bien été ajoutée pour une clé en particulier.
- Avec .Errors[0].ErrorMessage, on vérifie si le message est celui attendu. Dans les cas où il y aurait plusieurs messages d'erreur à la même clé, on utiliserait .Errors[1], .Errors[2], etc.

```
IActionResult resultat = await controller.Create(cpvm);

ViewResult viewResult = Assert.IsType<ViewResult>(resultat);

ModelStateDictionary modelState = viewResult.ViewData.ModelState;

Assert.True(modelState.ContainsKey("Prix"));
Assert.Equal("Blablabla.", modelState["Prix"].Errors[0].ErrorMessage);
```

Tests unitaires

```
return RedirectToAction(nameof(Index));
```

❖ Catalogue d'exemples d'assertions

♦ RedirectToActionResult

- Lorsqu'un RedirectToActionResult est retourné, on pourrait vouloir vérifier si c'est vers la bonne action et / ou le bon contrôleur. (Surtout si l'action peut engendrer des redirections vers plusieurs autres actions)

- On récupère le IActionResult retourné par l'action.
- Est-ce que c'est bel et bien un RedirectToActionResult ? (L'action a-t-elle engendrée une redirection ?)
- On peut vérifier si la redirection a bien été faite vers la bonne action et / ou le bon contrôleur.

```
IActionResult resultat = await controller.Create(cpvm);

RedirectToActionResult redirectResult = Assert.IsType<RedirectToActionResult>(resultat);

Assert.Equal("Index", redirectResult.ActionName);

Assert.Equal("Produits", redirectResult.ControllerName);
```

- Catalogue d'exemples d'assertions
  - Vérifier un DbSet (Donnée ajoutée, supprimée, modifiée, etc.)
    - Les tests des fonctions DbSet.Add(), DbSet.Remove(), etc. ne sont généralement pas pertinents car on est en train de tester EntityFrameworkCore.
    - Ce n'est pas à nous de tester EntityFrameworkCore. À priori, l'équipe derrière cette librairie l'a déjà testé. (Alors ce n'est pas nécessaire de la tester nous aussi)
    - Ce qui pourrait être plus intéressant à tester est l'interaction de nos actions avec notre base de données. Puisque ça n'implique pas seulement une fonction isolée, on utilisera les tests d'intégration pour cela.
    - Si nous avions eu des Services ou des Repositories, tester les .Create(), .Edit(), .Delete(), etc. implémentés par ces classes qui font le pont entre un contrôleur et le DbContext aurait été pertinent dans le cadre des tests unitaires.

Tests d'intégration

❖ Tests d'intégration

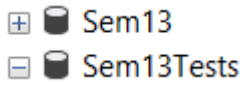
- ♦ Avec les tests d'intégration, l'objectif est de tester l'interaction entre plusieurs composants de notre application.
  - ○ Exemple : La base de données et des contrôleurs de l'application Web
  - ○ Tel que dit à la diapo précédente, tester Entity Framework n'est pas nécessaire, mais tester l'interaction entre la BD et les actions (qui exploitent EntityFramework) devient pertinent.
    - D'ailleurs, le fait d'utiliser des procédures stockées en *Raw SQL* est beaucoup moins infaillible qu'utiliser le DbContext, alors c'est une avenue de tests importante.

- ♦ Les tests d'intégration sont plus compliqués à préparer que les tests unitaires.
  - ○ Ils impliquent généralement plus de code.
  - ○ Si quelque chose peut être testé avec des tests unitaires à la place, il faut prioriser les tests unitaires.

Tests d'intégration

❖ Étapes préliminaires

- ♦ Étape 1 : Créer une nouvelle BD qui servira pour les tests
  - Seulement avec **`CREATE DATABASE Sem13Tests`**. Elle sera vide.



- ♦ Étape 2 : Mettre la BD de test dans le même état que la vraie BD grâce à Evolve
  - (Exécutez les migrations : **`evolve migrate sqlserver`** ...), mais n'oubliez pas de bien changer le nom de la BD dans le ConnectionString. (

- ♦ Étape 3 : Préparer un ConnectionString qui pointe vers la DB de test.

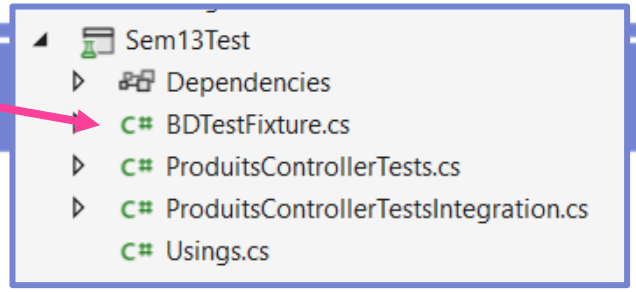
```
"Data Source=.\\SQLEXPRESS;Initial Catalog=Sem13;Integrated Security=True;Persist Security Info=F
```

(Juste changer le nom de la BD)

```
"Sem13": "Data Source=.\\SQLEXPRESS;Initial Catalog=Sem13Tests;Integrated Security=True;Persist Security Info=F
```

Dans le cadre du cours, vous pouvez très bien utiliser la même BD. Notre BD ordinaire est déjà une BD de test !

Tests d'intégration



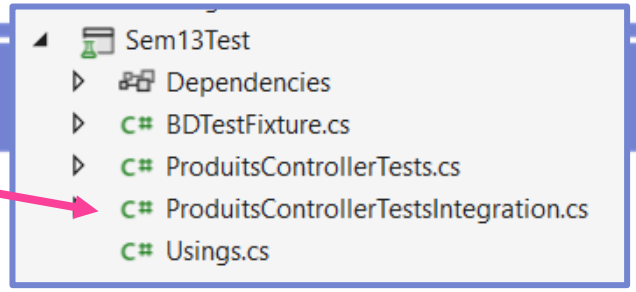
❖ Utiliser une BD de test

- ♦ Cette fois-ci, nous utiliserons un vrai DbContext plutôt que simulé.

- Ceci est un exemple de classe qui permet de créer un DbContext qui pourra être appelé par **plusieurs** classes de test.
- Notez que cette classe aurait été beaucoup plus grande si nous avions utilisé un design Code-First. (Voir dernière diapo)
- « Fixture » est un nom fréquemment donné aux classes qui gèrent un environnement de test utilisé par **plusieurs tests**.

```
3 references
public class BDTestFixture
{
    private const string ConnectionString = "Data Source=.\\SQLEXPRESS;Initial Catalog=Sem13Tests;Integrated Security=True;Persist Se

    2 references | 2/2 passing
    public Sem13Context CreateContext()
    {
        return new Sem13Context(new DbContextOptionsBuilder<Sem13Context>().UseSqlServer(ConnectionString).Options);
    }
}
```

Tests d'intégration

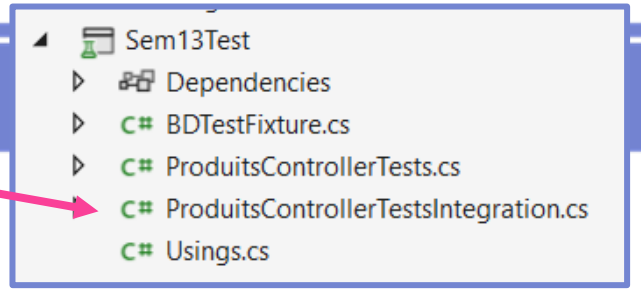


❖ Classe de tests

- ♦ Dans chaque classe de test qui utilise la BD de test...
  - ○ On hérite de IClassFixture<ClasseDeLaBD>.
  - ○ On injecte la ClasseDeLaBD dans le constructeur.

```
1 reference
public class ProduitsControllerTestsIntegration : IClassFixture<BDTestFixture>
{
    3 references | 2/2 passing
    private BDTestFixture Fixture { get; }

    0 references
    public ProduitsControllerTestsIntegration(BDTestFixture fixture)
    {
        Fixture = fixture;
    }
```

Tests d'intégration



- Test sans modifications de données
  - Très similaire aux tests unitaires, sauf qu'on utilise un vrai DbContext

```
public async Task<IActionResult> Index()
{
    if(_context.Produits == null)
    {
        return Problem("L'ensemble Produits est null.");
    }

    return View(await _context.Produits.ToListAsync());
}
```

```
[Fact]
0 references
public async Task Index_VerifierSiDixProduitsObtenus()
{
    using Sem13Context context = Fixture.CreateContext();
    ProduitsController controller = new ProduitsController(context);

    IActionResult actionResult = await controller.Index();
    ViewResult viewResult = Assert.IsType <ViewResult>(actionResult);
    List<Produit> model = Assert.IsAssignableFrom<List<Produit>>(viewResult.Model);
    Assert.Equal(10, model.Count);
}
```

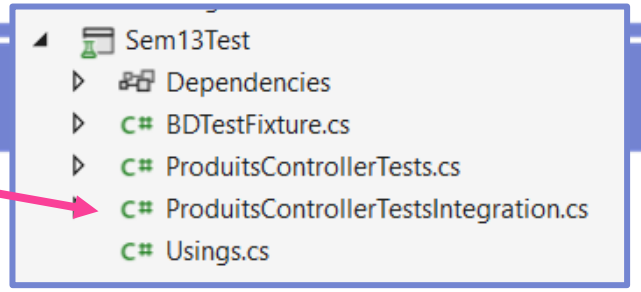
Arrange

Act

Assert

(Explications dans la prochaine diapo)

Tests d'intégration



❖ Test sans modification de données

- ♦ Très similaire aux tests unitaires, sauf qu'on utilise un vrai DbContext

• Si le test n'implique pas de **modifications de données** (Donc surtout des GET par exemple), le test pourrait ressembler largement à un test unitaire.

• Gardez à l'esprit que contrairement à un test unitaire, si ce test échoue, ce n'est pas forcément la faute d'une action dans un contrôleur : ça peut aussi être la faute de la base de données. (Ce qui rend les tests d'intégration moins ciblés)

• Ici, on a mis 10 car on sait que dans la BD de test, les migrations avaient inséré 10 produits au total.

```
[Fact]
0 references
public async Task Index_VerifierSiDixProduitsObtenus()
{
    using Sem13Context context = Fixture.CreateContext();
    ProduitsController controller = new ProduitsController(context);


    IActionResult actionResult = await controller.Index();
    ViewResult viewResult = Assert.IsType <ViewResult>(actionResult);
    List<Produit> model = Assert.IsAssignableFrom<List<Produit>>(viewResult.Model);
    Assert.Equal(10, model.Count);
}
```

Tests d'intégration

❖ Test avec modification de données

- ♦ Il faut prendre quelques précautions pour que ces tests ne changent pas l'état de la BD. (Au risque de créer des conflits avec d'autres tests)

```
[HttpPost]
[ValidateAntiForgeryToken]
3 references | 3/3 passing
public async Task<IActionResult> Create(CreationProduitVM cpvm)
{
    if (ModelState.IsValid)
    {
        Produit produit = new Produit()
        {
            ProduitId = 0,
            Categorie = cpvm.Categorie,
            Nom = cpvm.Nom,
            QteStock = cpvm.QteStock,
            Prix = cpvm.PrixEntier + cpvm.PrixDecimal / 100,
            EstDiscontinue = cpvm.EstDiscontinue
        };
        _context.Produits.Add(produit);
        await _context.SaveChangesAsync();

        return RedirectToAction(nameof(Index));
    }
    ModelState.AddModelError("", "Un ou plusieurs champs sont invalides.");

    return View(cpvm);
}
```

(Explications dans la prochaine diapo)

Arrange

Act

Assert

```
[Fact]
0 references
public async Task CreatePost_InsertionProduitValide()
{
    CreationProduitVM cpvm = new CreationProduitVM()
    {
        Nom = "Divan 2 places", PrixEntier = 499, PrixDecimal = 99,
        QteStock = 3,  EstDiscontinue = false, Categorie = "Meuble"
    };

    using Sem13Context context = Fixture.CreateContext();

    context.Database.BeginTransaction();

    ProduitsController controller = new ProduitsController(context);
    IActionResult resultat = await controller.Create(cpvm);

    context.ChangeTracker.Clear();

    Assert.Equal(11, await context.Produits.CountAsync());

    //context.Database.RollbackTransaction();
}
```

Tests d'intégration

❖ Test avec modification de données

♦ Il faut prendre quelques précautions pour que ces tests ne changent pas l'état de la BD. (Au risque de créer des conflits avec d'autres tests)

• Dans ce cas, le test essaye d'**ajouter** un Produit dans la BD. (Via l'action « Create » du ProduitsController) Il y a ensuite une assertion qui vérifie si la BD possède maintenant bel et bien 11 produits.

• On remarque qu'une transaction est explicitement démarrée avec le DbContext. De plus, on nettoie le ChangeTracker plus bas. La conséquence de ces ajouts est que le produit sera **bel et bien ajouté dans la BD** pendant la transaction. Cela dit, puisque le ChangeTracker est nettoyé et que la transaction n'est jamais Commit, l'insertion du produit sera **annulée** (La transaction sera Rollback) une fois la fonction terminée.

• On pourrait être tentés de faire le Rollback nous-mêmes (la ligne commentée), mais ce n'est pas nécessaire car EF le fera pour nous.

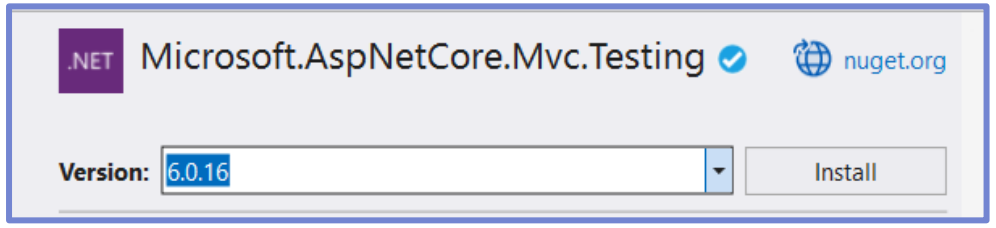
```
[Fact]
0 references
public async Task CreatePost_InsertionProduitValide()
{
    CreationProduitVM cpvm = new CreationProduitVM()
    {
        Nom = "Divan 2 places", PrixEntier = 499, PrixDecimal = 99,
        QteStock = 3,  EstDiscontinue = false, Categorie = "Meuble"
    };

    using Sem13Context context = Fixture.CreateContext();

    context.Database.BeginTransaction();

    ProduitsController controller = new ProduitsController(context);
    IActionResult resultat = await controller.Create(cpvm);

    context.ChangeTracker.Clear();

    Assert.Equal(11, await context.Produits.CountAsync());

    //context.Database.RollbackTransaction();
}
```

❖ Test qui impliquent l'authentification

- ♦ Pas mal plus compliqué !
  - ○ Pour pouvoir utiliser des actions qui nécessitent d'**identifier l'utilisateur** qui envoie la requête (Pas qui ont l'annotation [Authorize], mais bien celles qui fouillent dans le cookie de la requête pour identifier l'utilisateur qui a appelé l'action)
  - ○ Si une action possède l'annotation [Authorize], mais n'a pas besoin d'accéder au cookie de l'utilisateur, elle peut être testée de la même manière que dans les diapos précédentes.

- ♦ Quelques prérequis :

Package à installer




```
public partial class Program
{

}
```

Morceau de code à ajouter à la toute fin de Program.cs dans le projet principal.

Méthode à ajouter dans la classe de test (ou dans une classe Fixture)

```
private WebApplicationFactory<Program> GetApp()
{
    return new WebApplicationFactory<Program>();
}
```

❖ Test qui impliquent l'authentification

♦ Action à tester

- On remarque qu'elle a l'annotation [Authorize], mais surtout, qu'elle accède au cookie fournit par le client.

```
[Authorize]
0 references
public async Task<IActionResult> IndexAvecAutorisation()
{
    if (_context.Produits == null)
    {
        return Problem("L'ensemble Produits est null.");
    }

    IIdentity? identite = HttpContext.User.Identity;
    if(identite != null && identite.IsAuthenticated)
    {
        string pseudo = HttpContext.User.FindFirstValue(ClaimTypes.Name);
        Utilisateur? utilisateur = await _context.Utilisateurs.FirstOrDefaultAsync(x => x.Pseudonyme == pseudo);
        if (utilisateur != null)
        {
            ViewData["utilisateur"] = utilisateur.Pseudonyme;
            return View(await _context.Produits.ToListAsync());
        }
    }

    return Unauthorized();
}
```

❖ Test qui impliquent l'authentification

♦ Méthode de test

Arrange

Act

Assert

```
[Fact]
0 references
public async Task IndexAvecAutorisation_VerifierDixProduits()
{
    WebApplicationFactory<Program> application = GetApp();

    using IServiceScope services = application.Services.CreateScope();
    HttpClient client = application.CreateClient(new WebApplicationFactoryClientOptions
    {
        AllowAutoRedirect = false
    });

    HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Post, "/Utilisateurs/Connexion");
    FormUrlEncodedContent content = new FormUrlEncodedContent(new Dictionary<string, string>()
    {
        {"Pseudonyme", "max" },
        {"MotDePasse", "Salut1!"}
    });
    request.Content = content;

    HttpResponseMessage response = await client.SendAsync(request);

    if (response.StatusCode == HttpStatusCode.Found)
    {
        string? authCookie = response.Headers.GetValues("Set-Cookie").FirstOrDefault();
        if(authCookie != null)
        {
            client.DefaultRequestHeaders.Add("Cookie", authCookie);
            HttpResponseMessage resultat = await client.GetAsync("/Produits/IndexAvecAutorisation");

            Assert.True(resultat.IsSuccessStatusCode);
        }
    }
}
```

```
return Ok(await _context.Produits.ToListAsync());
```

❖ Test qui impliquent l'authentification

♦ Méthode de test

Nous n'irons pas dans les détails, mais voici quelques explications clés :

• Au lieu d'instancier le contrôleur et d'appeler l'action comme on le faisait avant, on doit créer un « client » qui envoie une requête à l'action de notre choix pour se « connecter ».

• On récupère le cookie servi par l'action de connexion.

• On se sert ensuite de ce cookie pour appeler une action qui nécessite l'authentification.

• Malheureusement, comme on travaille avec des requêtes et des réponses HTTP, il n'est pas évident d'accéder aux données retournées. (Les return View() deviennent des pages HTML plutôt que des « ViewResult »)

• Si l'action avait return Ok(produits) comme dans une Web API, il aurait été plus facile d'analyser les données retournées par l'action.

```
return Ok(await _context.Produits.ToListAsync());
```

```
[Fact]
0 references
public async Task IndexAvecAutorisation_VerifierDixProduits()
{
    WebApplicationFactory<Program> application = GetApp();

    using IServiceScope services = application.Services.CreateScope();
    HttpClient client = application.CreateClient(new WebApplicationFactoryClientOptions
    {
        AllowAutoRedirect = false
    });

    HttpRequestMessage request = new HttpRequestMessage(HttpMethod.Post, "/Utilisateurs/Connexion");
    FormUrlEncodedContent content = new FormUrlEncodedContent(new Dictionary<string, string>()
    {
        {"Pseudonyme", "max" },
        {"MotDePasse", "Salut1!"}
    });
    request.Content = content;

    HttpResponseMessage response = await client.SendAsync(request);

    if (response.StatusCode == HttpStatusCode.Found)
    {
        string? authCookie = response.Headers.GetValues("Set-Cookie").FirstOrDefault();
        if(authCookie != null)
        {
            client.DefaultRequestHeaders.Add("Cookie", authCookie);
            HttpResponseMessage resultat = await client.GetAsync("/Produits/IndexAvecAutorisation");

            Assert.True(resultat.IsSuccessStatusCode);
        }
    }
}
```

❖ Tests fonctionnels

- ♦ Les tests fonctionnels ont pour objectif d'effectuer des tests en abordant la perspective d'un utilisateur.
  - ○ Cela implique d'interagir avec l'interface graphique de l'application et d'analyser si le comportement du système est celui attendu par l'utilisateur.
  - ○ Généralement, cela requiert la participation d'un humain si on souhaite avoir des tests complets.
  - ○ Il existe des outils pour automatiser les tests fonctionnels, (donc des outils capables d'interagir avec un interface graphique un peu comme un utilisateur le ferait) mais cela ne représente pas toujours des tests aussi riches qu'avec de vrais utilisateurs.

Tests d'intégration

❖ Utiliser une BD de test

♦ Si nous avions utilisé un design Code-First

Cette classe CRÉE LA BD directement dans le code. (Et on la supprime si elle existait déjà)

• On peut alors en profiter pour faire un seed de données de test.

• Cette manière de créer la BD à partir du DbContext offre plus d'**automatisation**, **mais nous ne l'utiliserons pas** car le DbContext ne permet pas de reproduire facilement notre BD avec tous ses objets. (Il manquerait des contraintes, des procédures stockées, des déclencheurs, etc.)

```
4 references
public class BDTestFixture
{
    private const string ConnectionString = "Data Source=.\\SQLEXPRESS;Initial Catalog=Sem13Tests;Integrated Security=True;Persist Security Info=Fal
    private static bool _databaseInitialized;

    0 references
    public BDTestFixture()
    {
        if (!_databaseInitialized)
        {
            using (var context = CreateContext())
            {
                context.Database.EnsureDeleted();
                context.Database.EnsureCreated();

                context.Produits.AddRange(
                    new Produit() { Categorie = "Meuble", Prix = 149.99M, Nom = "Table de chevet L5220", QteStock = 12, EstDiscontinue = false },
                    new Produit() { Categorie = "Meuble", Prix = 64.99M, Nom = "Chaise brune Youlo", QteStock = 17, EstDiscontinue = false },
                    new Produit() { Categorie = "Bijou", Prix = 5200M, Nom = "Rivière de rubis", QteStock = 1, EstDiscontinue = false }
                );
                context.SaveChanges();
            }

            _databaseInitialized = true;
        }
    }

    3 references | 2/2 passing
    public Sem13Context CreateContext()
    {
        return new Sem13Context(new DbContextOptionsBuilder<Sem13Context>().UseSqlServer(ConnectionString).Options);
    }
}
```