



Service Web .Net

Cours n°3 : Business Logic & Unit Test

Pierre-Loïc CHEVILLOT – Sébastien BEREIZIAT

Pierre-loic.chevillot@capgemini.com – sebastien.bereiziat@capgemini.com

Plan

- Business Logic
 - Architecture
 - Dependencies Injection
- Tests
 - Fonctionnelles / Techniques
 - Framework de tests (MSTest, Xunit, ...)

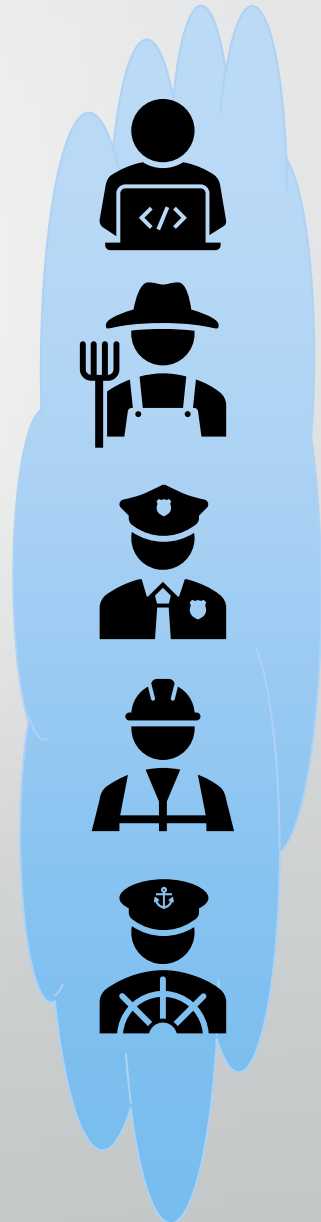
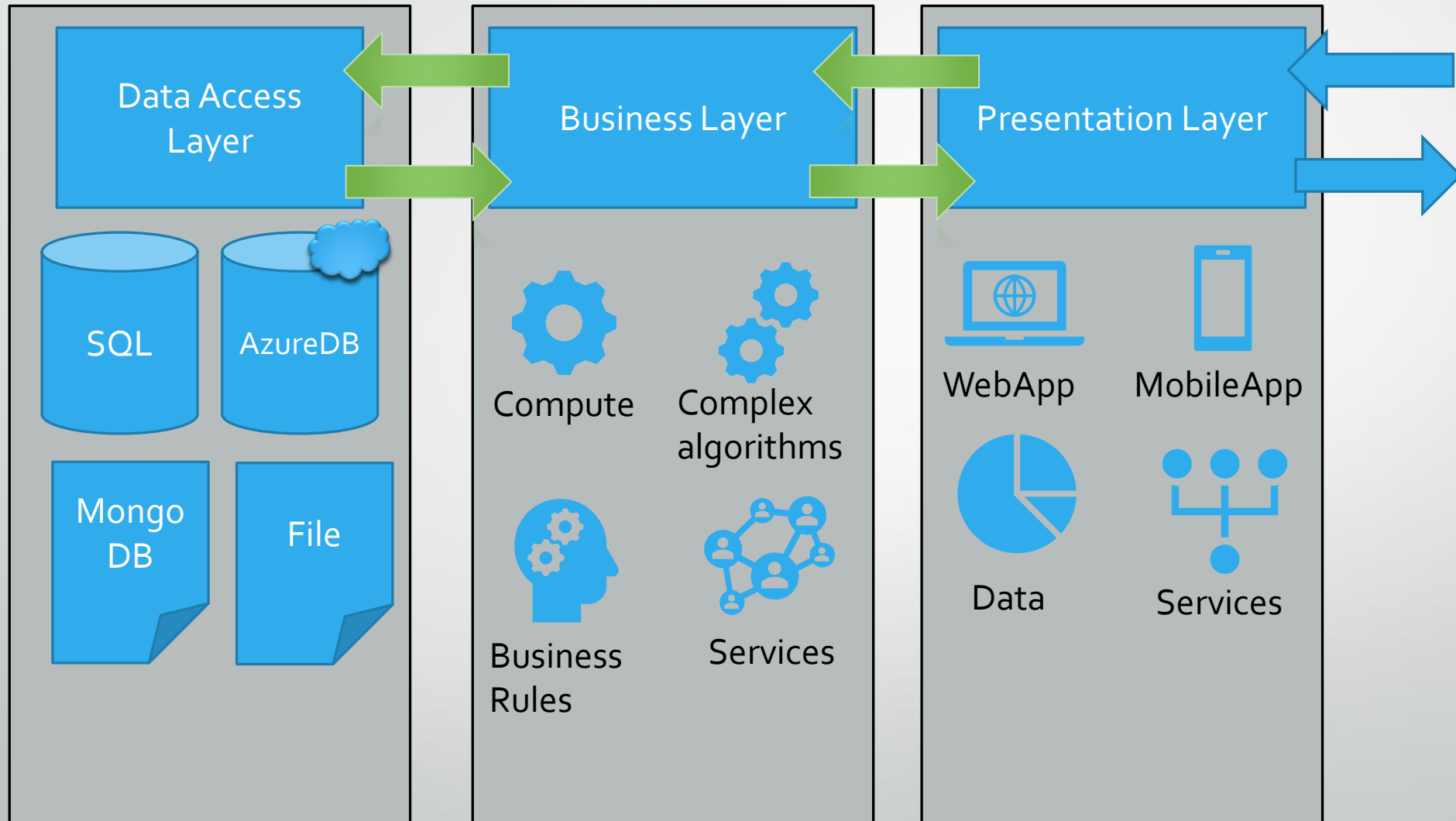
Architecture logicielle

- L'**architecture logicielle** décrit d'une manière symbolique et schématique les différents éléments d'un ou de plusieurs systèmes informatiques, leurs interrelations et leurs interactions.
- Un modèle d'architecture logicielle ne décrit pas ce que doit réaliser un système informatique, mais comment il est conçu afin de répondre aux spécifications (dimensionnement, sécurité, hébergement, ...)

Architecture en couches

- But : Modéliser une application comme un empilement de X couches logicielles
- 3-Tiers est la modélisation la plus couramment utilisée
 - Couche de présentation : Correspond à l'affichage de vos données à l'utilisateur final, et le dialogue avec celui-ci
 - Couche d'application (ou métier) : Correspond à la mise en œuvre des règles de gestion et de logique applicative
 - Couche de données : Correspond à la persistance des données.
- Avantage :
 - SoC (Separation of Concern) : chaque couche gère son domaine

Architecture en couches



Injection de dépendances - DI

- Instancier un object « new » = imposer un couplage fort
- Création d'un service => peu de couplage
- Injection par Interface
 - Constructeur demande un contrat d'interface
 - Utilisation de l'interface dans le code, pas d'utilisation de new

Inversion de contrôles IoC

- Conteneur qui doit être instancié et retournée au client
 - Evite l'appel d'une instanciation explicite (new)
 - Cache des instances nécessaires
- Mappage Interface / Implémentations au moment du démarrage de l'application

IoC + DI

- Faible couplage
 - Maintenabilité
 - Réduction temps de maintenance d'une applicatio
 - Permet de tester avec plusieurs configuration (Mock)
- Développement
 - Permet de se focaliser sur des petits modules réutilisables
 - Moins sensibles aux effets de bords
- Utilisation
 - Mécanique Interfaces a respecter
 - .NetCore facilite l'utilisation car IoC + DI au cœur de l'architecture avec un moteur IoC puissant et simple a utiliser

IoC + DI

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class BikeController : ControllerBase
{
    private readonly IBikeRepository repo;
    0 references
    public BikeController(IBikeRepository rep)
    {
        repo = rep;
    }
}
```

```
// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<BikeStoreContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddScoped<IBikeRepository, BikeRepository>();

    services.AddControllers();

    services.AddSwaggerGen();
}
```



Tests

Tests

- Fonctionnel : valider la fonctionnalité demandée d'une application sans connaître le code sous jacent.
- Technique : valider le comportement ET résultat d'une section de code source via toutes les possibilités (intran)
- Assurer la maintenabilité de l'application
 - A chaque évolution, une campagne de test fonctionnels est effectués pour s'assurer que les anciennes fonctionnalités, et les nouvelles sont conformes aux exigences
 - Modification du code source, passage des tests techniques afin de s'assurer que le code fonctionne de manière identique. Pas d'effet de bord de la modification du code source
- Documentation
 - Comprendre le fonctionnement d'une application via le carnet de test
- Implémentation
 - Avant d'implémenter une nouvelle fonctionnalité (TDD – test driven development)
 - Après avoir implémenter une nouvelle fonctionnalité
 - A la découverte d'une anomalie
 - Retrodoc : Découverte d'une nouvelle fonctionnalité

Test Technique

- Unitaire
 - Isoler : test uniquement d'une seule méthode en l'isolant du reste des couches, et autres méthodes publiques (Mock)
 - Validation d'un résultat attendu contre le résultat de la méthode. Les données en intrant sont définies et statique.
- Intégration
 - Validation du fonctionnement d'une fonctionnalités en se basant sur le code
- Mise a jour des tests a chaque modification de l'application, même mineure.

Framework tests

- Permet de coder et automatiser les tests
- Fonctionnel : Selenium
- Technique : Xunit, MSTest, NUnit, ..

Xunit / Mock (Moq)

- OpenSource Test Framework
- Evolution avec C#

- Opensource
- Permet de créer des « Fake » implémentation d'objet depuis une abstraction / Interface
- Simuler facilement des données
- Isolation d'une couche via un Fake

Xunit

```
[Fact]
✓ | 0 references
public void GetTaxedPrice()
{
    var fakeRepo = Mock.Of<IBikeRepository>();
    Mock.Get(fakeRepo).Setup(x => x.GetProductById(1)).Returns(new BikeDto...);

    TaxeService service = new TaxeService(fakeRepo);

    var bikeResult = service.GetBikeById(1);
    Assert.NotNull(bikeResult);
    Assert.NotNull(bikeResult.TaxAdded);
    Assert.NotNull(bikeResult.TaxValue);
    Assert.NotNull(bikeResult.PriceTaxInc);


    Assert.Equal(20, bikeResult.TaxAdded);
    Assert.Equal(120, bikeResult.PriceTaxInc);
}
```

- Fact = test unique valider par des asserts
- Theory = Enchainement du meme test en variant des variable d'entrée. Le test boucle avec la liste des InlineData que vous renseignez.

Moq

```
0 references
public TaxeService(IBikeRepository repo)
{
    _repo = repo;
}

0 references
public BikeDto GetBikeById(int id)
{
    var bike = _repo.GetProductById(id);
    bike.TaxValue = TAX;
    bike.TaxAdded = bike.ListPrice * (TAX / 100);
    bike.PriceTaxInc = bike.ListPrice + bike.TaxAdded;
    return bike;
}
```

 (local variable) BikeDto bike

```
var fakeRepo = Mock.Of<IBikeRepository>();
Mock.Get(fakeRepo).Setup(x => x.GetProductById(1)).Returns(new BikeDto
{
    ProductId = 1,
    ListPrice = 100
});

TaxeService service = new TaxeService(fakeRepo);
```