Génie Logiciel
Architectures MVC et MVP

BAPTISTE PESQUET

Exemples de code

https://github.com/bpesquet/genie-logiciel/tree/master/examples (MVC)

https://github.com/bpesquet/winforms-architecture-patterns (MVP)

Sommaire

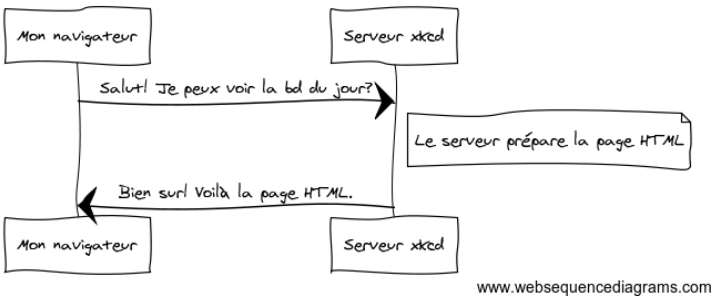
- Architecture MVC
  - Principe
  - Application en C# : ASP.NET Core MVC
- Architectures WinForms
  - Solution basique
  - Architecture en couches
  - MVP

Architecture MVC

Le fonctionnement du web

Modèle **client/serveur** basé sur un mécanisme **requête/réponse** :

1. Le client envoie une demande au serveur
2. Le serveur prépare la réponse
3. Il la renvoie au client



Clients web, serveurs web

Client web : initialise la demande au serveur et exploite ses résultats

Exemples de clients :

- Navigateurs web
- Applications mobiles
- Robots d'indexation

Serveur web : attend, reçoit et traite les demandes des clients

Le protocole HTTP

HTTP = **HyperText Transfer Protocol**

Socle technique du Web

Equivalent sécurisé : HTTPS

Plusieurs types de requêtes possibles (GET, POST, etc)

Principe

MVC = **Modèle-Vue-Contrôleur** (*Model-View-Controller*)

Décomposition d'une application en trois grandes parties :

- **Modèle** : accès aux données et logique métier (*business logic*)
- **Vue** : affichage et interactions avec l'utilisateur
- **Contrôleur** : dynamique de l'application, lien entre Modèle et Vue

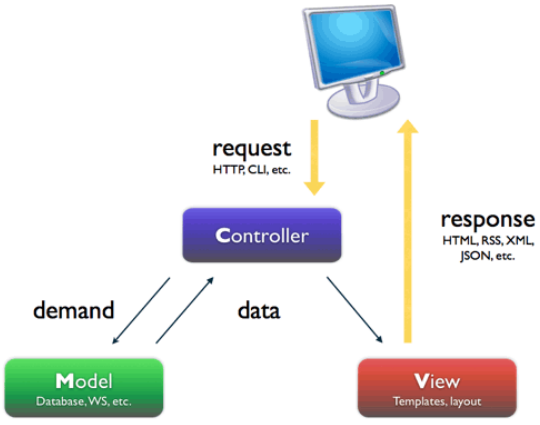
Application du principe de séparation des responsabilités

Histoire

Apparition à la fin des années 1970 pour le langage OO Smalltalk
- Objectif : séparer le code de l'IHM de la logique applicative

Appliqué depuis dans de très nombreux contextes et langages
- PHP : frameworks Symfony, Laravel, etc
- C# : ASP.NET MVC
- JavaScript : frameworks Angular, Ember, etc
- Java : bibliothèques Swing et JSP
- Python : Django
- Ruby : Rails

Anatomie d'un serveur web MVC



Extrait de la documentation du framework Symfony

Avantages et inconvénients

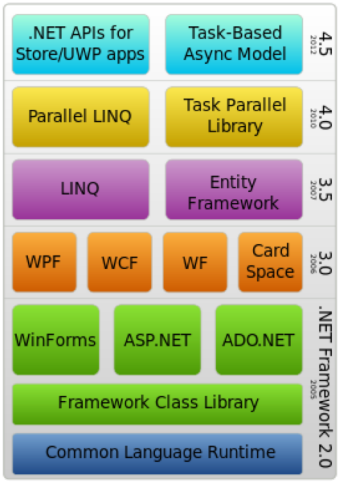
Avantages
- Clarification de l'architecture
- Séparation nette des responsabilités => couplage faible, cohésion forte, maintenance et évolutions facilités

Inconvénients
- Complexification de l'architecture
- Rigidité

.NET et .NET Core

.NET
- Framework créé par Microsoft en 2002
- Uniquement sous Windows (projet Mono : portage sous Linux)
- Propriétaire

.NET Core
- Déclinaison multiplateforme de .NET apparue en 2016
- Open source !
- Ne contient pas tout .NET, mais plus modulaire (basé sur les packages NuGet)
- Future version unique de .NET



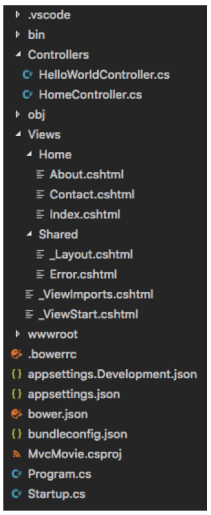
Le framework ASP.NET Core MVC

Environnement serveur web de Microsoft
- Sites web
- API web (backend d'applications mobiles ou de SPA)

Implémente le patron MVC et de nombreuses bonnes pratiques

Léger, moderne et modulaire

Déploiement : Azure, Heroku, IBM Cloud…



Fichier HomeController.cs

```
public class HomeController : Controller {
    public IActionResult Index() { return View();  }
    public IActionResult About() {
        ViewData["Message"] = "Your application description page.";
        return View();
    }
    public IActionResult Contact() {
        ViewData["Message"] = "Your contact page.";
        return View();
    }
    public IActionResult Error() { return View(); }
}
```

Architectures WinForms

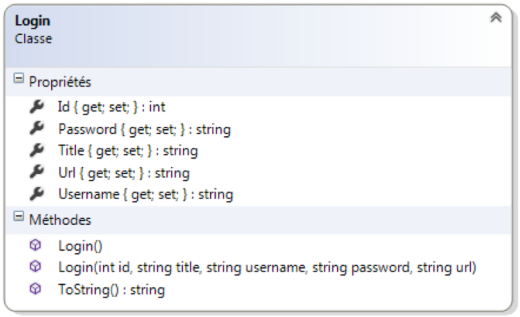
Importance de l'architecture

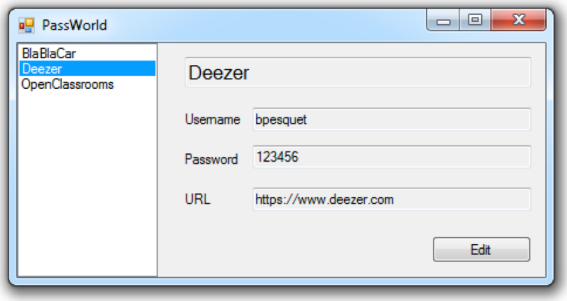
Augmentation de la complexité => besoin de structurer l'application

Séparation des responsabilités
- WinForms = IHM/GUI

Tests unitaires automatisés => modularité nécessaire

Contexte d'exemple





17 sur 29

Solution basique

Les fichiers « code behind » des formulaires contiennent tout le code du projet

- Accès aux données persistantes
- Traitements métier
- Authentification
- Gestion des erreurs
- …

Monobloc : aucune structure !

18 sur 29

Exemple

```
public MainForm() {
    InitializeComponent();
    // Load logins from file
    filePath = @"passwords.xml";
    if (File.Exists(filePath)) {
        StreamReader reader = new StreamReader(filePath);
        loginList = (List<Login>)new XmlSerializer(typeof(List<Login>)).Deserialize(reader);
        reader.Close();
    }
    else { // ... }
    loginLB.DataSource = loginList; // Link view to login list
    loginLB.SelectedIndex = 0; // List should always contain items
}
```


Avantages/inconvénients

Avantages

- Très simple
- Facile à comprendre

Inconvénients

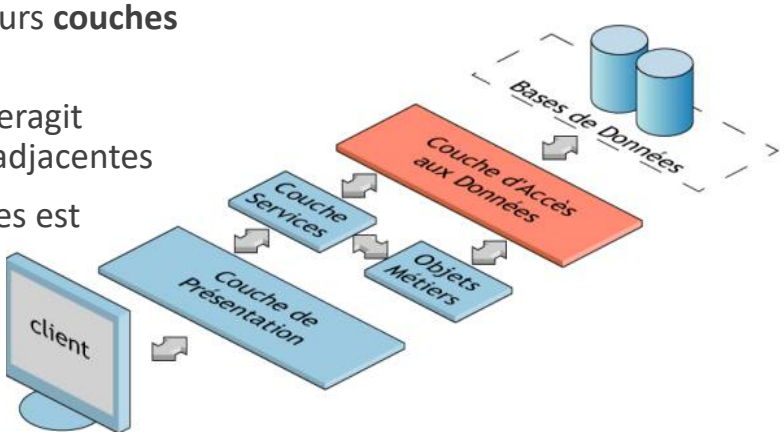
- Pas du tout testable
- Peu évolutif

Architecture en couches

Décomposition d'une application en plusieurs **couches** (*layers*)

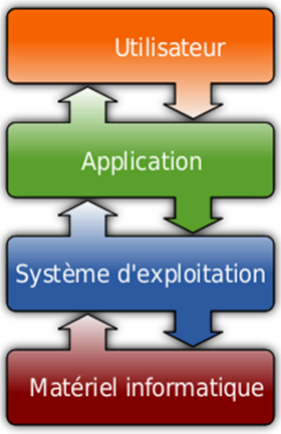
Chaque couche n'interagit qu'avec les couches adjacentes

Le nombre de couches est variable



21 sur 29

Applications



TCP/IP model	Protocols and services	OSI model
Application	HTTP, FTTP, Telnet, NTP, DHCP, PING	Application
		Presentation
		Session
Transport	TCP, UDP	Transport
Network	IP, ARP, ICMP, IGMP	Network
Network Interface	Ethernet	Data Link
		Physical

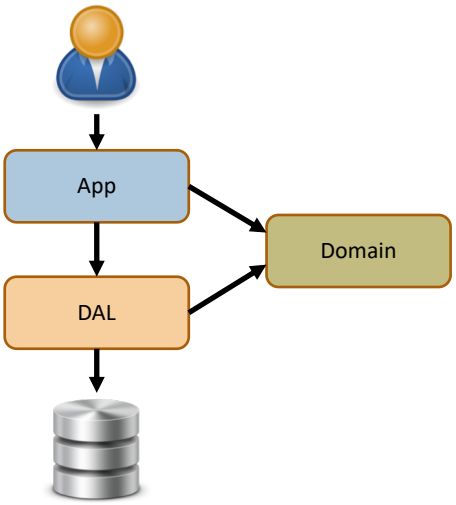
22 sur 29

Exemple

```
public MainForm(ILoginRepository loginRepository)
{
    InitializeComponent();

    this.loginRepository = loginRepository;

    loginLB.DataSource = loginRepository.GetAll();
    loginLB.SelectedIndex = 0;
}
```



Avantages/inconvénients

Avantages

- Premier niveau de séparation des responsabilités
- Complexité raisonnable

Inconvénients

- La logique UI n'est pas testable unitairement

Architecture MVP

MVP = ***Model-View-Presenter***

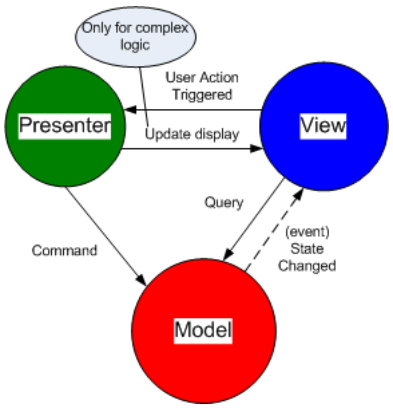
Variante du MVC

1. La vue (et non le contrôleur) reçoit les actions utilisateur
2. Elle délègue leur gestion au *Presenter* qui interagit avec le modèle
3. La vue est ensuite mise à jour

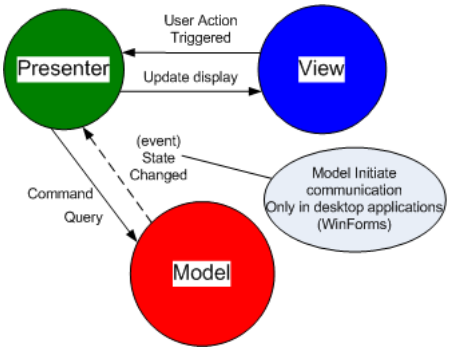
Application du principe de séparation des responsabilités

Deux variantes

Supervising Controller



Passive View



http://aviadezra.blogspot.fr/2008/10/model-view-presenter-design-pattern.html

Exemple : vue passive

```
public interface IMainView  {
    List<Login> LoginList { get; set; }
    int SelectedLoginIndex { get; set; }
    Login SelectedLogin { get; }
    string Title { get; set; }
    string Username { get; set; }
    string Password { get; set; }
    // …
    Presenter.MainPresenter Presenter { set; }
}
```

```
public partial class MainForm : Form, View.IMainView {
  public MainForm() {
    InitializeComponent();
  }
  public List<Login> LoginList {
    get { return (List<Login>)this.loginLB.DataSource; }
    set { this.loginLB.DataSource = value; }
  }
  public string Title {
    get { return titleTB.Text; }
    set { titleTB.Text = value; }
  }
  // …
```


Exemple : *MainPresenter*

```
public MainPresenter(ILoginRepository loginRepository,
View.IMainView mainView) {
    this.loginRepository = loginRepository;
    view = mainView;
    mainView.Presenter = this;
    view.LoginList = loginRepository.GetAll();
    view.SelectedLoginIndex = 0;
}
```

Avantages/inconvénients

Avantages

- Responsabilités clairement séparées
- Logique UI (*Presenter*) testable unitairement

Inconvénients

- Complexité