Prosit Retour - Restauration

Infos et rôles

Date : 06/02/2025
Début : 8h35
Fin : 9h55
Durée : 1h20

Tuteur : Mohammed
Animateur : Mathéo
Secrétaire : Lucas
Scribe : Clément
Gestionnaire : Ludovic

Mots-clés

- Régression logiciel
- ASP.NET
- MVC/MVVM
- Évolutivité
- Migrer
- Modularité
- Logique Métier
- .NET 6.0
- Design Pattern avancé
- Couche de présentation

- Maintenabilité
- C#
- Gestion de versions
- POO Avancé
- UML
- Principes SOLID
- Communication technique

Contexte

2F Roaming a résolu des problèmes techniques mais fait face à des coûts élevés et une architecture rigide. (Ludovic)

Ils souhaitent redéfinir l'architecture de l'application. (Jérémy)

Problématique

Comment séparer les différents couches d'une application tout en garantissant une maintenabilité et la ré usabilité.

Contraintes

- Imposition des Frameworks modernes
- Évolutivité/Maintenabilité
- Réduction des coûts au maximum
- POO Avancé

Livrable(s)

- UML
- Code (Design Pattern)
- Plan formation

Généralisation

- Modèle d'architecture d'application

Hypothèses

- MVC —> Model View Controller ?
- MVVM —> Model View View Model ?

Permettrai d'architecturer le code et la communication entre les controller (cerveau), les model et la view (affichage) ?

L'intérêt est de sectoriser le code ?

Pourquoi deux méthodes différents :

- 1 pour la technique et 1 pour le client ?
- 1 plus ancien et 1 plus récent ?

- 3 couches :
  - Présentation
  - Logique métier
  - Accès donné

- Principe SOLID pour :
  - La modification ne doit pas entrainer de "casser" les bonnes pratiques.

- ASP .NET quésaco ?
  - Permet de faire du backend et de l'applicatif ?

Pourquoi être sur une ancienne version de .NET ?

- .NET 6.0 présente des avantages par rapport à .NET 8.0
- Problème de fiabilité ? → Une ancienne version est potentiellement travaillé pendant plus longtemps ?
- Versions plus récentes → Plus de sécurité ?
- Pas de rétrocompatibilité ? Nouvelles versions peuvent changer l'utilisation complète de la techno.
- Tant que cela fonctionne on ne change pas ? Devoir former toute une team de Senior qui ont l'habitude d'une certaine techno → Coûts temps migration et formation

---

UML à faire ? Lesquels ?

- Classe ? Si l'archi est à refaire ? Peut-être des bonnes pratiques à respecter qui ne le sont pas
- Séquence ? Nouveau membre dans l'équipe
- USER CASE ? Présentation pour une personne extérieure, pas trop de détail ni trop exhaustif.

Plan d'action

- Tableau comparatif
  - MVC et MVVM
  - .NET 6.0 et .NET 8.0

- Voir la compatibilité de plusieurs technos
  - ASP. NET Core, Blazore, Razore...

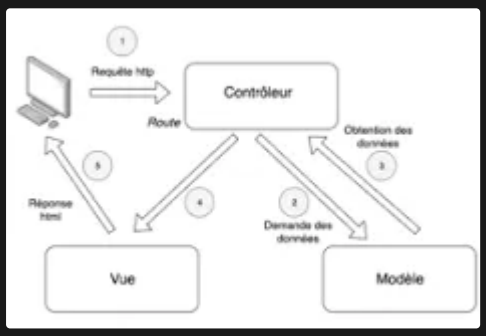
- Voir qu'est ce que la POO Avancé
  - Faire un petit projet entre différentes versions de .NET en incluant la POO et le Design Pattern

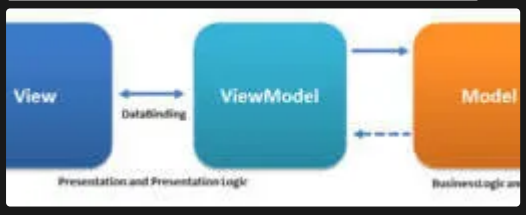
- UML
  - Faire des exemples de schémas selon les pistes de solutions (Classe, Séquence, USER CASE)

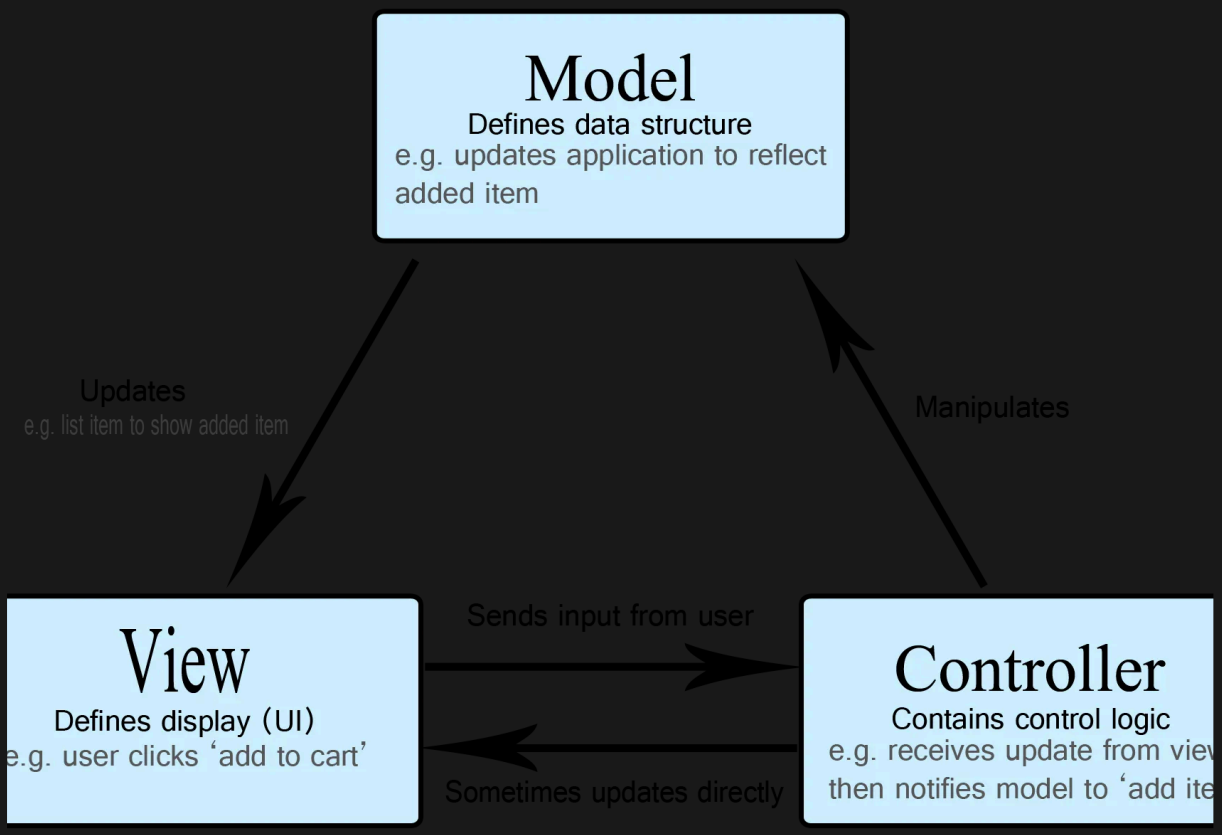
- Plan formation

Présentation tableaux comparatifs et modèles MVC / MVVM

MVC :






Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view
then notifies model to 'add ite

Dotnet 6 vs 8

au comparatif : .NET 6.0 vs .NET 8.0

e	.NET 6.0	.NET 8.0
de sortie	Novembre 2021	Novembre 2023
de support	LTS (Long-Term Support) jusqu'en **novembre 2024**	LTS (Long-Term Support) jusqu'en **novem 2026**
rmance	Amélioration par rapport à .NET 5, mais encore perfectible	Optimisations significatives, notamment p le garbage collector et les performances d l'IA/ML
r	Blazor Server et WebAssembly disponibles	Introduction de **Blazor United** pour fusior Server et WebAssembly
(Multi- rm App UI)	Nouvelle plateforme pour les applications mobiles et desktop, mais encore jeune	Plus stable et amélioré avec de meilleures performances
t ionnalités	Support des APIs modernes mais certaines limitations	Améliorations des APIs existantes et ajout nouvelles fonctionnalités
ité	Meilleure sécurité par rapport aux versions précédentes	Renforcement de la sécurité avec des mise jour majeures
pérabilité	Bonne interopérabilité avec C++, JavaScript, et autres langages	Améliorations sur l'interopérabilité avec le nouvelles plateformes et cloud

nclusion : .NET 8.0 est une évolution majeure de .NET 6.0 avec des améliorations en nances, stabilité et nouvelles fonctionnalités, notamment pour Blazor et MAUI. Il est recomm ter .NET 8.0 pour bénéficier du support à long terme et des meilleures performances.

Compatibilité de plusieurs technologies

mpatibilité des technologies ASP.NET Core, Blazor et Razor

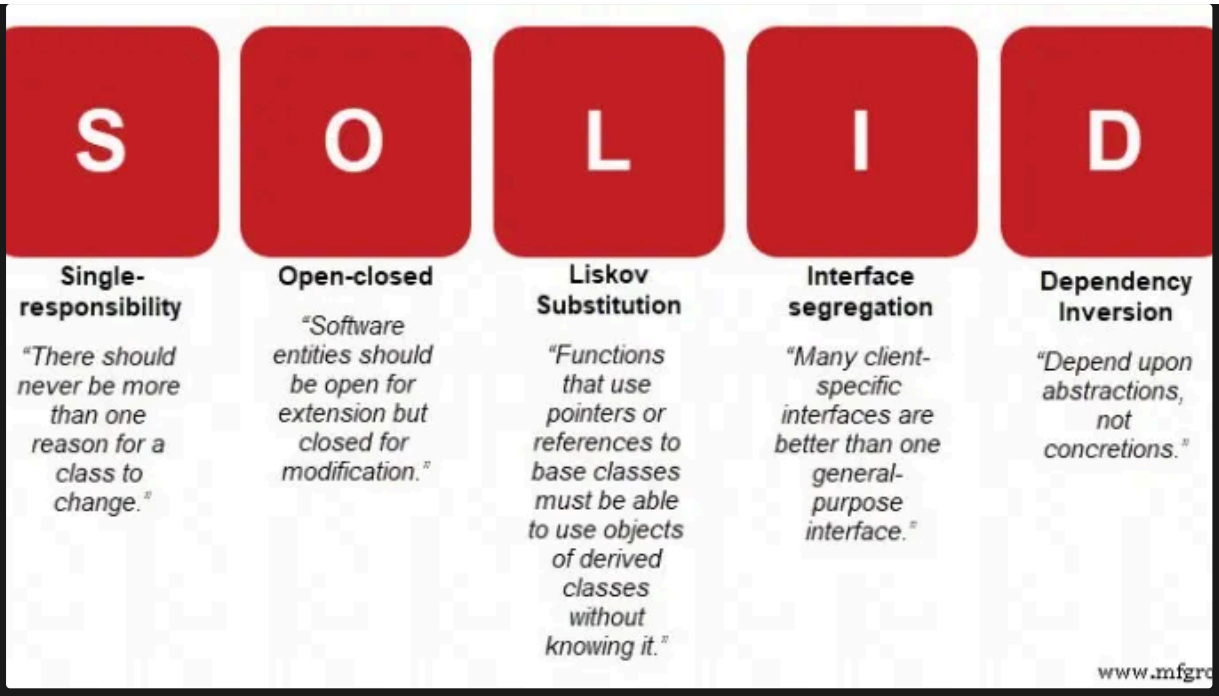
i un tableau comparatif des compatibilités et utilisations des technologies **ASP.NET Core**, **Blazor**, or :

chnologie	Type d'application	Compatibilité	Rôle principal	Rendement
P.NET Core	Applications web backend et API	Windows, Linux, macOS (cross-platform)	Framework pour créer des applications web et APIs REST	Très performant e scalable
azor Server	Applications web interactives	Fonctionne côté serveur, nécessite une connexion permanente	Remplace JavaScript pour le développement front-end en C#	Faible latence, ma nécessite un serveur
azor ebAssembly	Applications web SPAs (Single Page Applications)	Fonctionne côté client dans le navigateur	Exécute du C# directement dans le navigateur sans besoin de serveur	Performance proche de JavaScript
azor Hybrid AUI)	Applications desktop et mobiles	Windows, macOS, iOS, Android	Utilise .NET MAUI pour des apps natives avec Blazor	Bonne performan mais dépend du framework MAUI
zor	Génération dynamique de contenu HTML	Intégré à ASP.NET Core et Blazor	Langage de templating basé sur C# pour les vues web	Rapide et simple utiliser dans les pages web
zor Pages	Pages web dynamiques (alternative à MVC)	Windows, Linux, macOS	Permet de créer des applications web sans utiliser MVC	Simplicité et maintenabilité améliorées

Description POO Avancé et Principe SOLID

POO avancer :

- Rappel SOLID



Exemple pour Integration segregation

Mauvais exemple :

arp — Copier — Modifier

```
blic interface IWorker {
  void Work();
  void Eat();


blic class Robot : IWorker {
  public void Work() { /* OK */ }
  public void Eat() { throw new NotImplementedException(); } // ❌ Problème !
```

Bon exemple (segmentation de l'interface) :

arp — Copier — Modifier

```
blic interface IWorkable {
  void Work();


blic interface IEatable {
```