Cours WPF

SIO2 - Antoine MATHAT

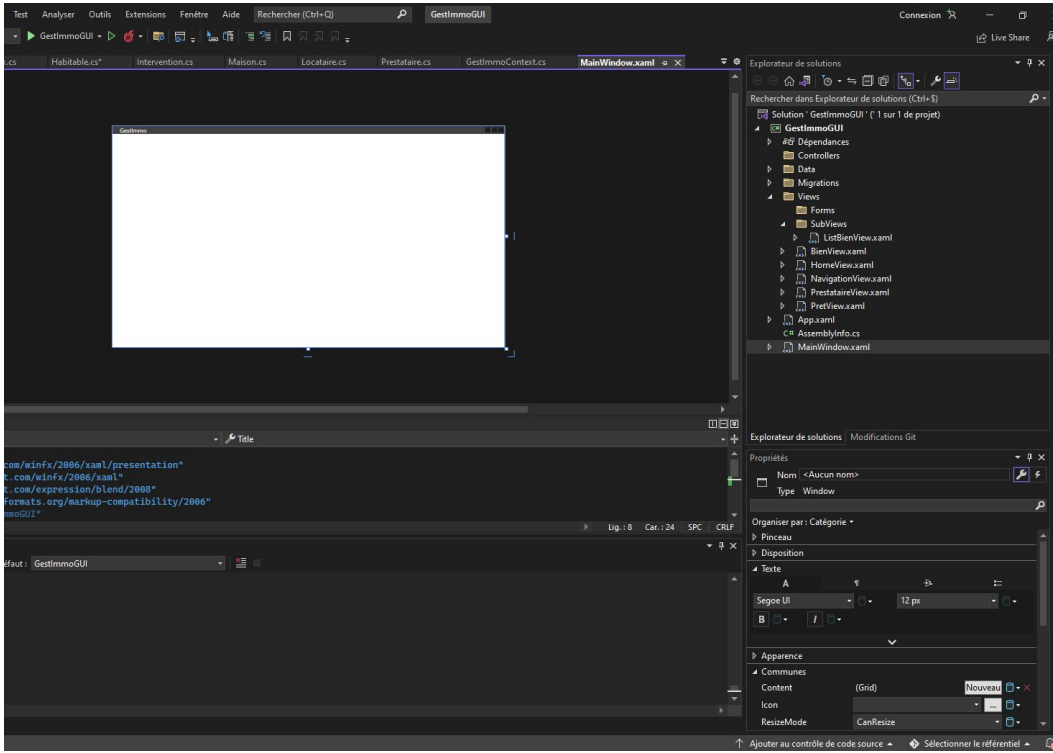
Sommaire

1. Intro WPF
2. Architecture
    a. MVVM
    b. MVC
    c. Fichiers
3. Navigation avec WPF
4. Design Pattern Singleton
5. Vues à développer

Intro WPF

Windows Presentation Framework (WPF)

Framework .NET graphique de Microsoft pour le C# et le Visual Basic.

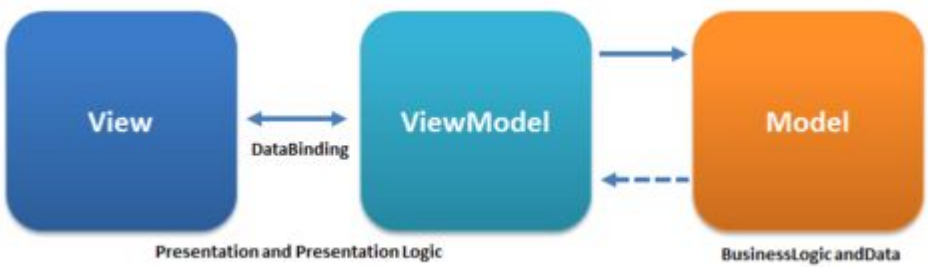


Architecture

Architecture

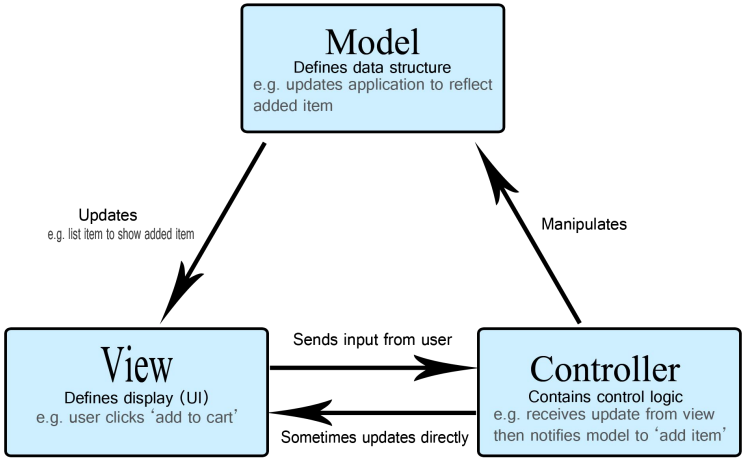
WPF est conçu pour fonctionner avec l'architecture Model View ViewModel (MVVM)

Pré-requis : généricité, Design pattern observer / Observable …

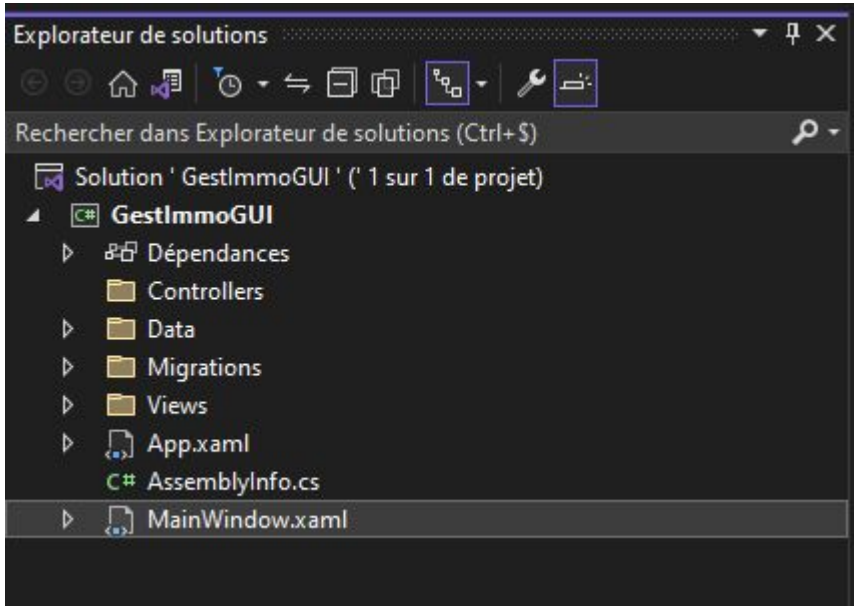


Architecture

Dans un souci de simplicité, nous mettrons en place une architecture MVC. La couche de données (Model) sera gérée par EF6.



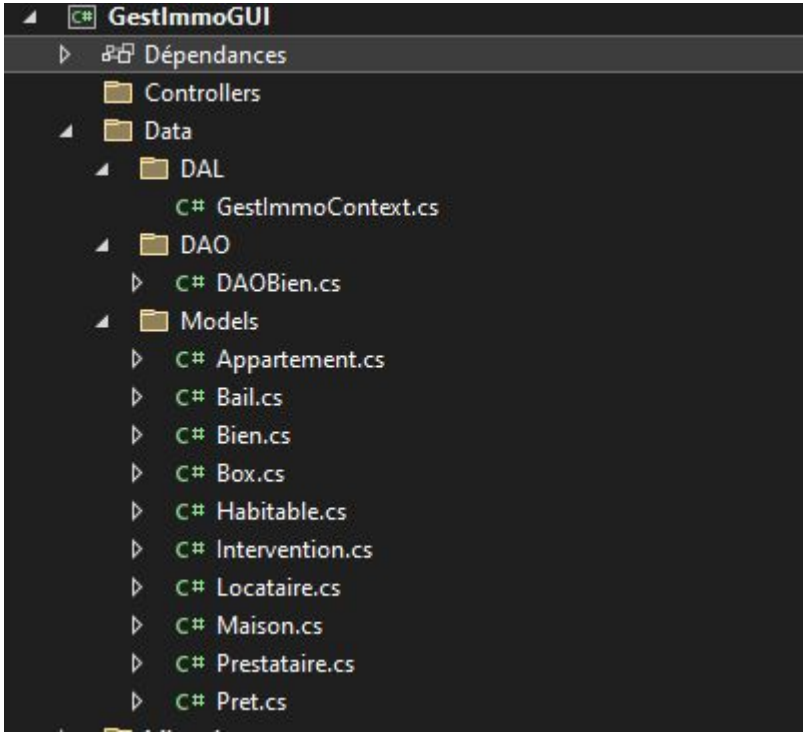
Architecture - Globale



Data / Controllers / Views pour l’architecture MVC

Migrations pour EF6

Architecture - Data



DAL : tous les fichiers relatifs à EF6

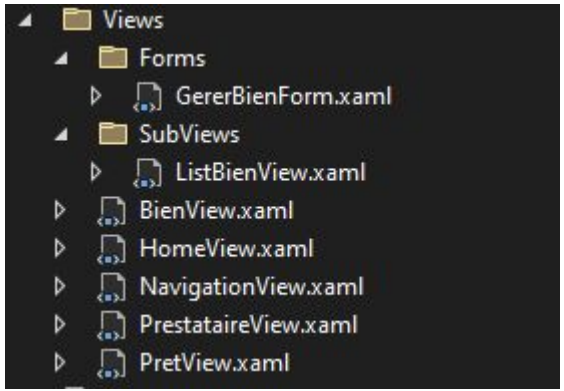
DAO (Data Access Object) : une classe pour chaque modèle. Une DAO contient des requêtes spécifiques pour une table

Models : les modèles

Architecture - Controller

Pour la prochaine fois :)

Architecture - View



Les vues principales à la racine

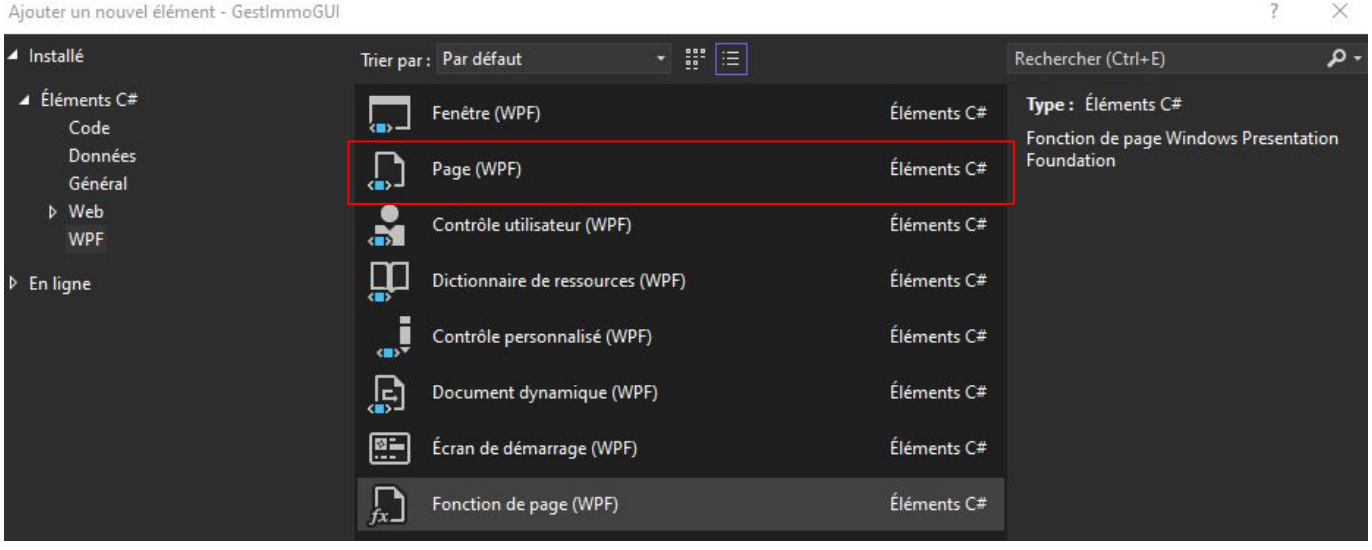
Forms : les formulaires

SubViews : les vues intégrables dans d’autres vues

Navigation avec WPF

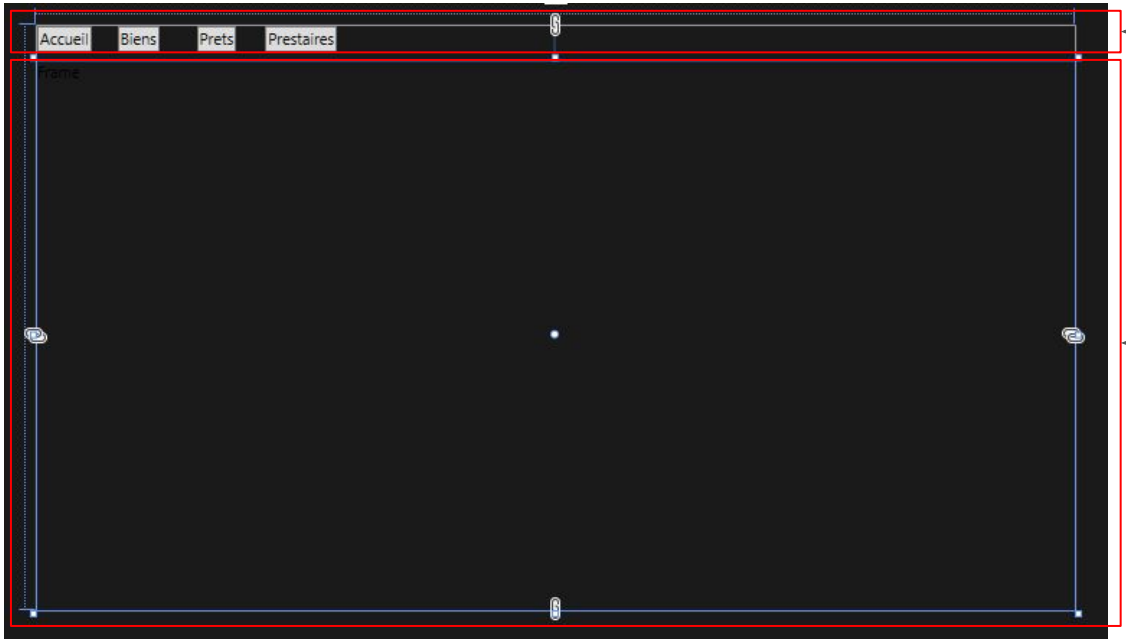
Navigation

- Le point d'entrée de l'application est une **Fenêtre**.
- Nous allons construire nos View avec des **Pages**
- Le contrôle **Frame** nous permet de naviguer vers des vues



Navigation

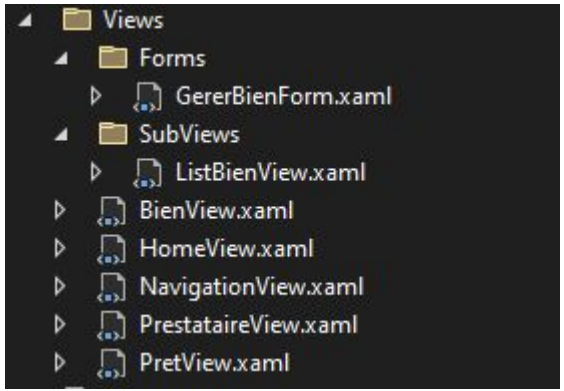
- Créer une Page et la nommer : “NavigationView”



éléments de navigation

Frame pour charger les autres Pages (View)

Navigation



Créer vos Pages (View) qui seront chargées dans une Frame

Navigation

```
public partial class NavigationView : Page
{
    1 référence
    public NavigationView()
    {
        InitializeComponent();
        this.MainFrame.Navigate(new HomeView());
    }
```

Charger une vue au lancement

```
1 référence
private void btnBien_Click_1(object sender, RoutedEventArgs e)
{
    this.MainFrame.Navigate(new BienView());
}
```

Changer de vue lors du clic sur un bouton de navigation

Navigation

```
2 références
public partial class MainWindow : Window
{
    0 références
    public MainWindow()
    {
        InitializeComponent();
        this.Content = new NavigationView();
    }
}
```

Charger la NavigationView au lancement du program dans le constructeur de la MainWindow.

Design Pattern Singleton

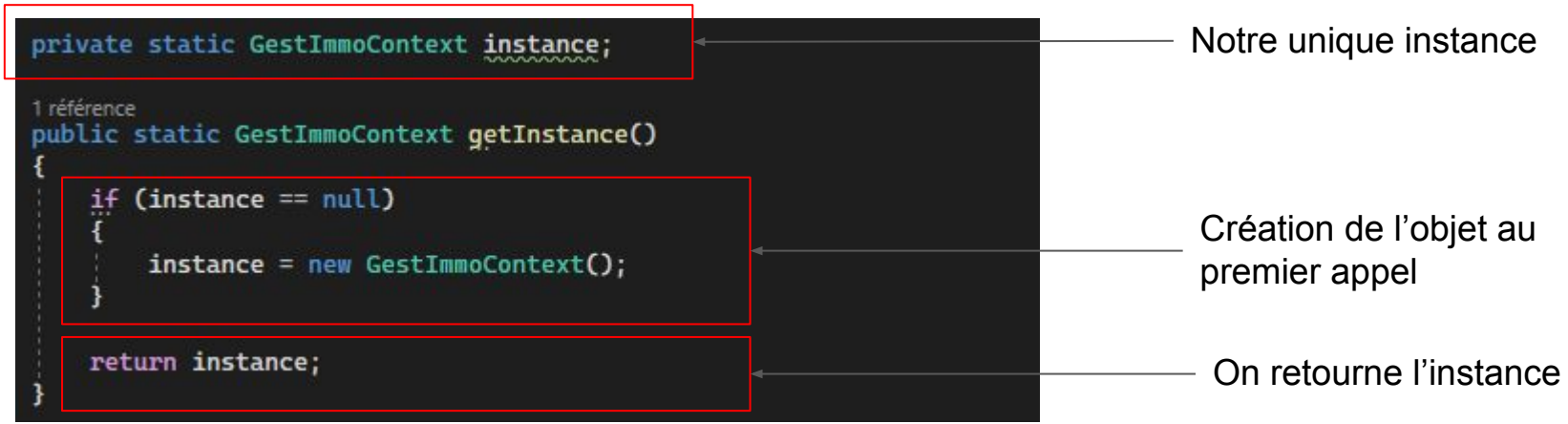
Singleton

Permet d’utiliser une unique instance d’une classe dans l’ensemble d’un programme.

Exemple : Ne pas ré-instancier l’accès à la base de données pour chaque requête

Singleton

Exemple d’un singleton basique dans la classe GestImmoContext



Singleton

Utilisation du singleton pour accéder à des éléments de la base de données

```
GestImmoContext ctx = GestImmoContext.getInstance();

foreach(Bien bien in ctx.Biens)
{
    this.lstBiens.Items.Add(bien.Nom);
}
```

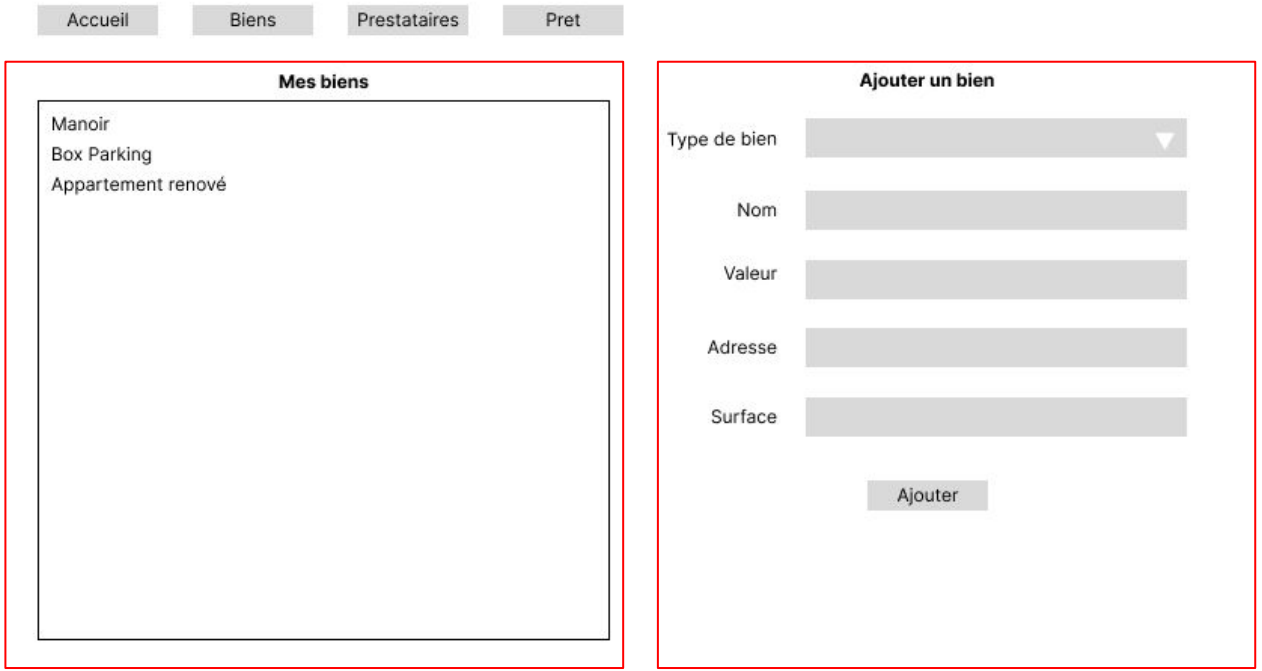
Récupération de l'instance

Design Observer

Observer

Le modèle de design Observateur permet à un abonné de s’inscrire auprès d’un fournisseur et d’en recevoir des notifications.

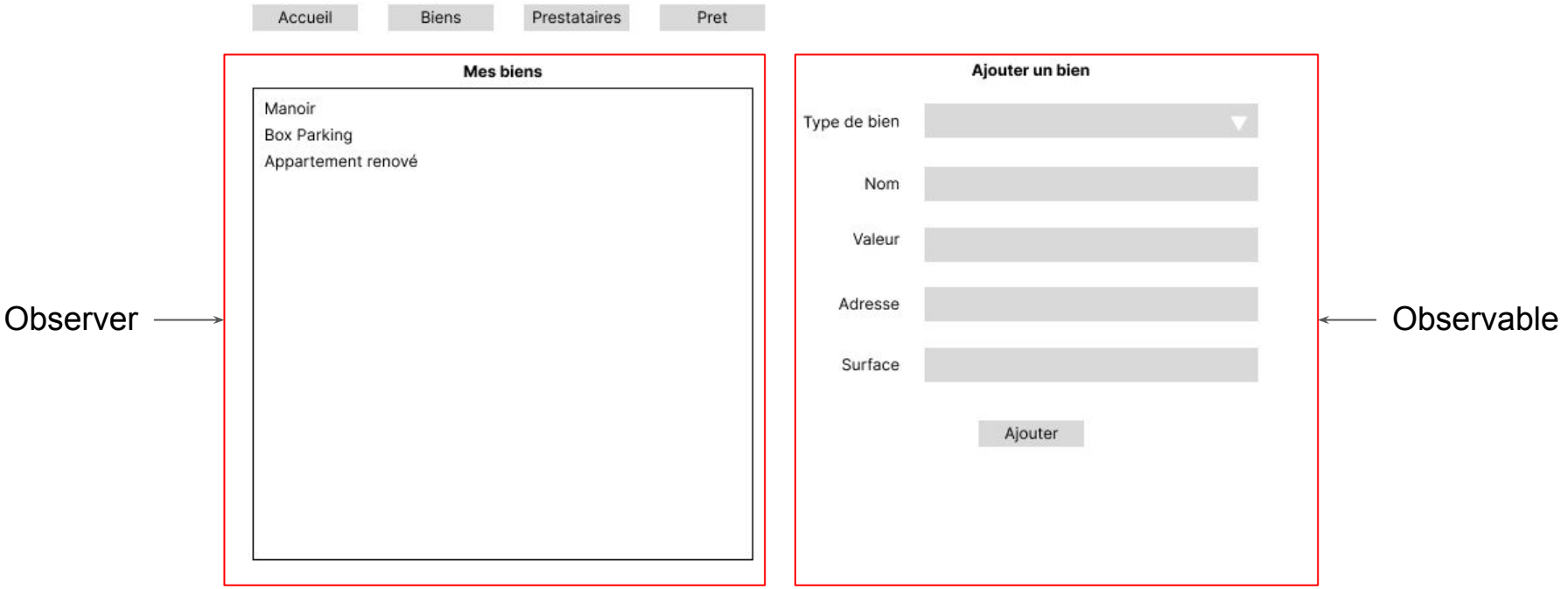
Observer



Les deux composants sont indépendants.

Cependant, ils doivent interagir entre eux.

Observer



Lorsque une **interaction est réalisée** dans la boxFormAjouterBien, un **refresh** est fait pour lstVieBiens

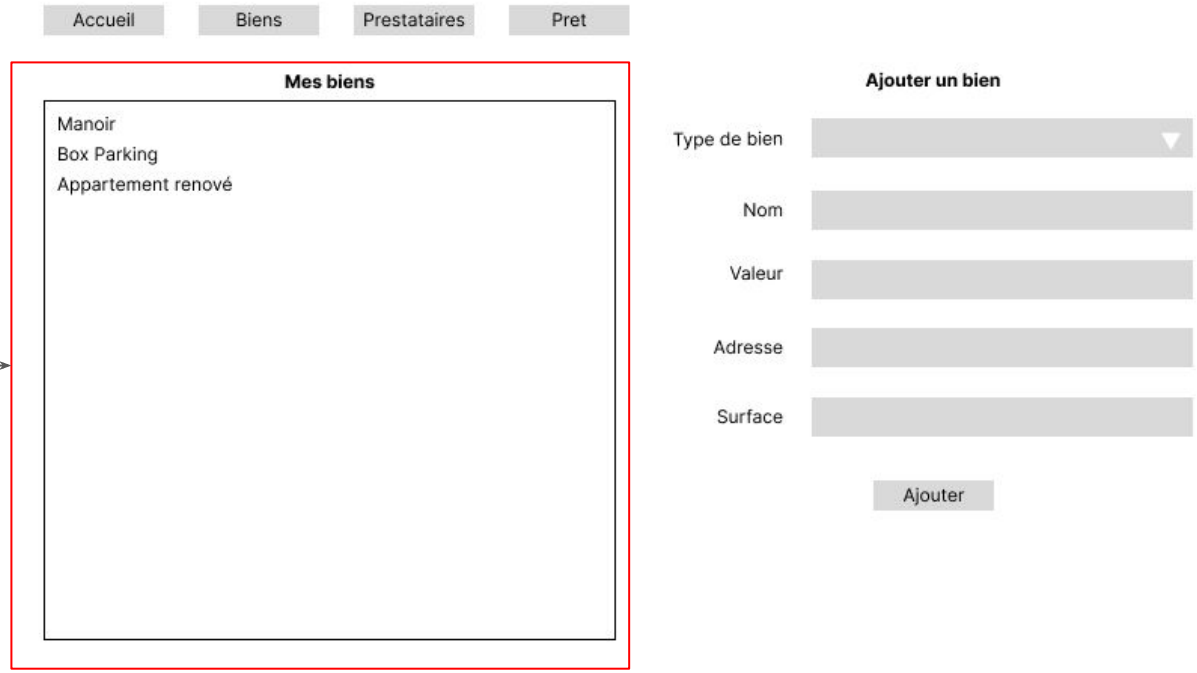
Observer

Ce design pattern utilise des interfaces :

- **IObserver** : Pour la classe qui Observe
- **IObservable** : Pour la classe observée

IObserver

Hérite de IObserver



IObserver

```
namespace GestImmoGUI.Views.Utils
{
    7 références
    public interface IObserver
    {
        3 références
        public void update();
    }
}
```

On surcharge la fonction update dans l’enfant Observer

ListBienView : IObserver

2 références

```
private void refreshList()
{
    GestImmoContext ctx = GestImmoContext.getInstance();

    this.lstBiens.Items.Clear();

    foreach (Bien bien in ctx.Biens)
    {
        this.lstBiens.Items.Add(bien.Nom);
    }
}
```

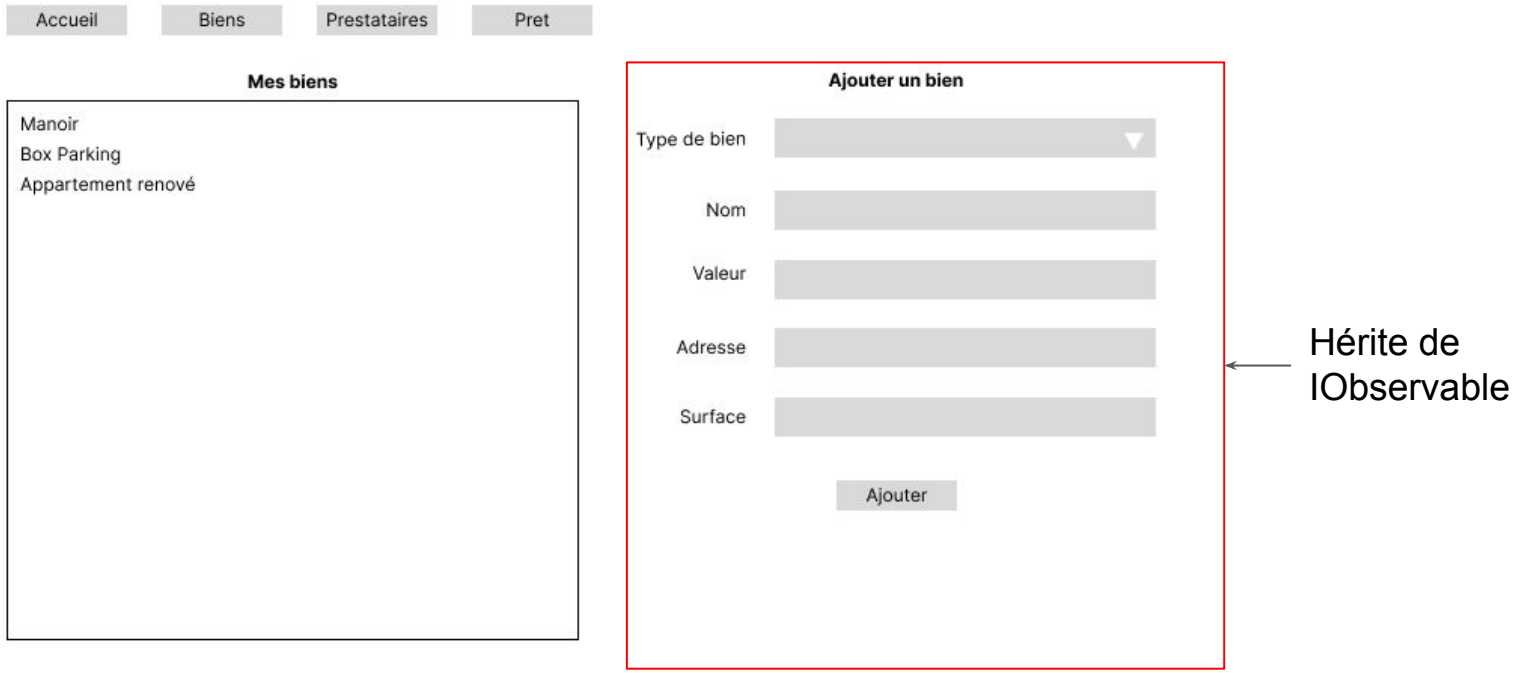
← Refresh des Biens

3 références

```
public void update()
{
    this.refreshList();
}
```

← Méthode héritée

IObservable



IObservable

```
public interface IObservable
{
    5 références
    List<IObserver> Observers { get; set; }

    0 références
    void notifyObservers()
    {
        foreach (IObserver obs in Observers)
        {
            obs.update();
        }
    }
}
```

Permet d’ajouter des Observer et de les notifier en cas d’action

GererBoxForm : IObservable

```
private void btnAjouter_Click(object sender, RoutedEventArgs e)
{
    string nom = this.txtNom.Text;
    int valeur = int.Parse(this.txtValeur.Text);
    string adresse = this.txtAdresse.Text;
    int surface = int.Parse(this.txtSurface.Text);

    Box box = new Box(nom, valeur, adresse, surface);
    GestImmoContext ctx = GestImmoContext.getInstance();
    ctx.Biens.Add(box);
    ctx.SaveChanges();

    this.notifyObservers();
}
```

Ajout d'un bien

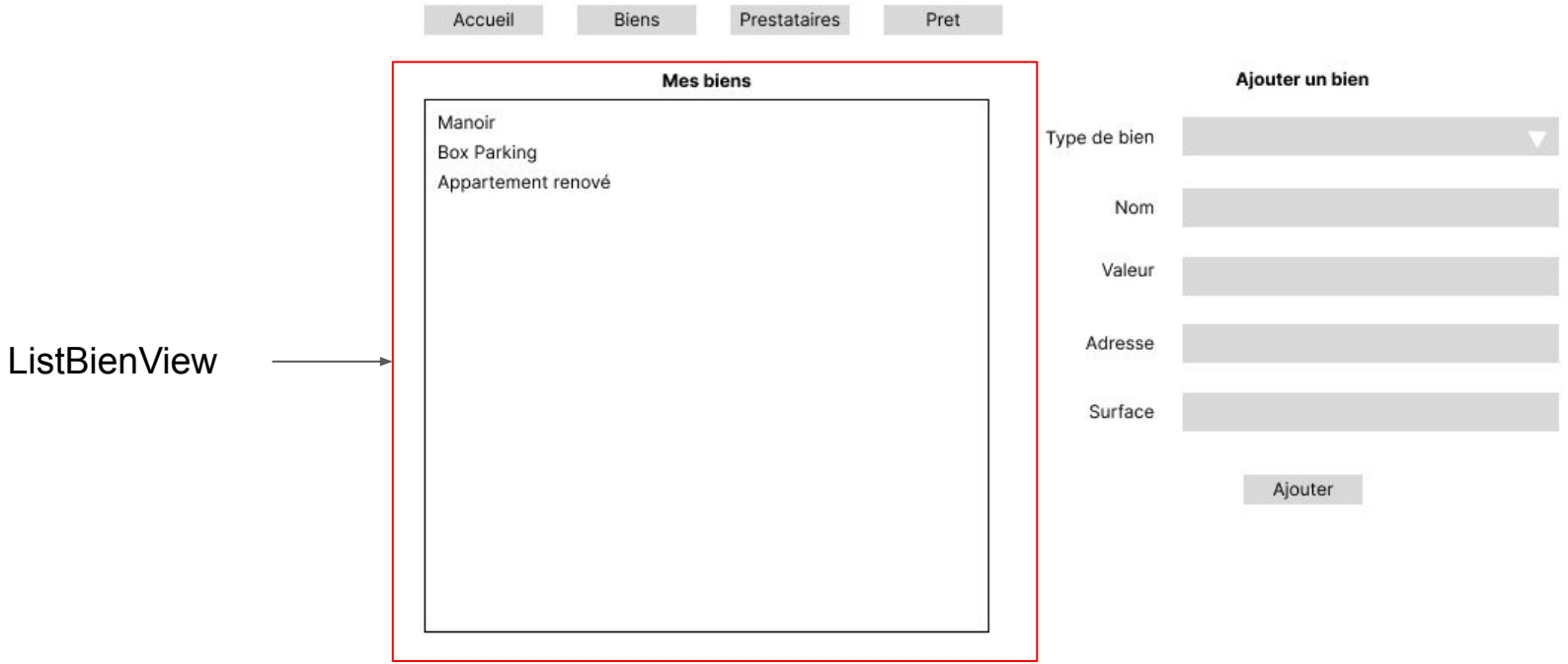
```
1 référence
void notifyObservers()
{
    foreach (IObserver obs in Observers)
    {
        obs.update();
    }
}
```

Notification des Observers

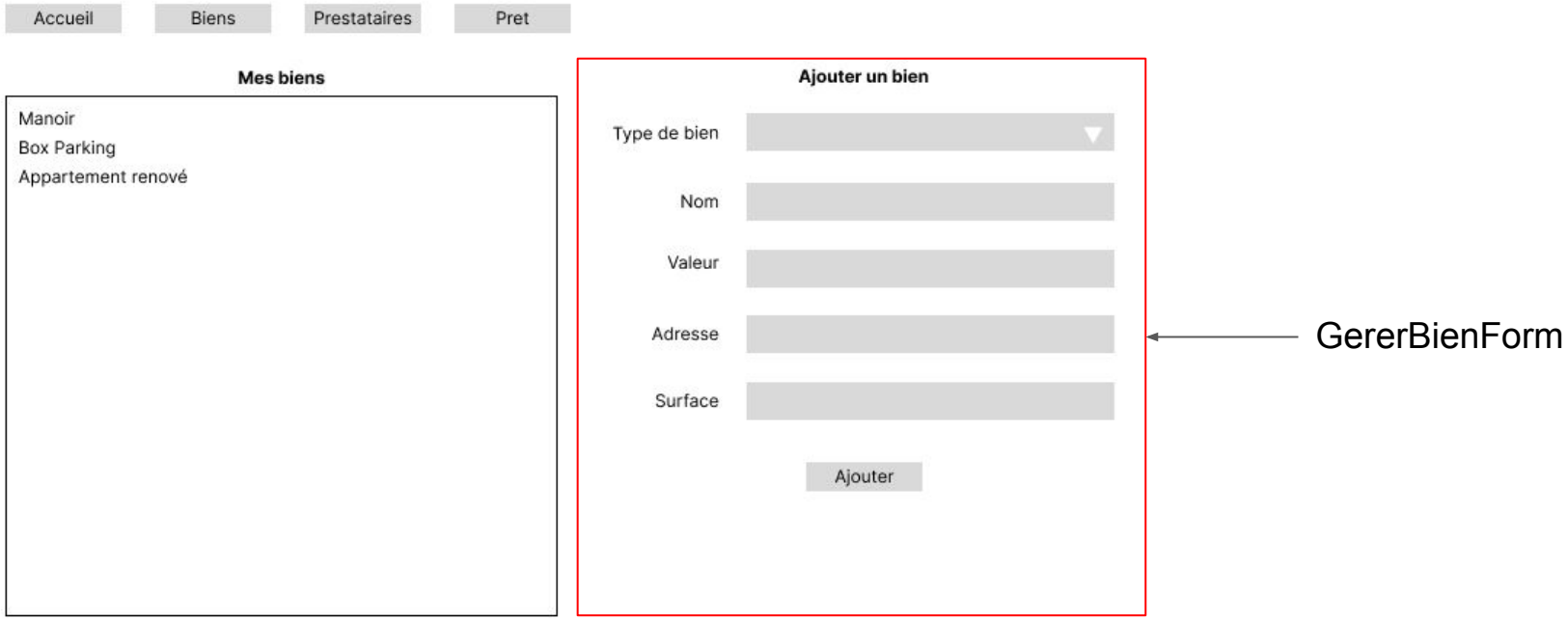
Vues à développer

NavigationView

BienView


Accueil
Biens
Prestataires
Pret
Mes biens
Manoir
Box Parking
Appartement renové
ListBienView
Ajouter un bien
Type de bien
Nom
Valeur
Adresse
Surface
Ajouter

BienView


Accueil
Biens
Prestataires
Pret
Mes biens
Manoir
Box Parking
Appartement renové
Ajouter un bien
Type de bien
Nom
Valeur
Adresse
Surface
Ajouter
GererBienForm

BienView

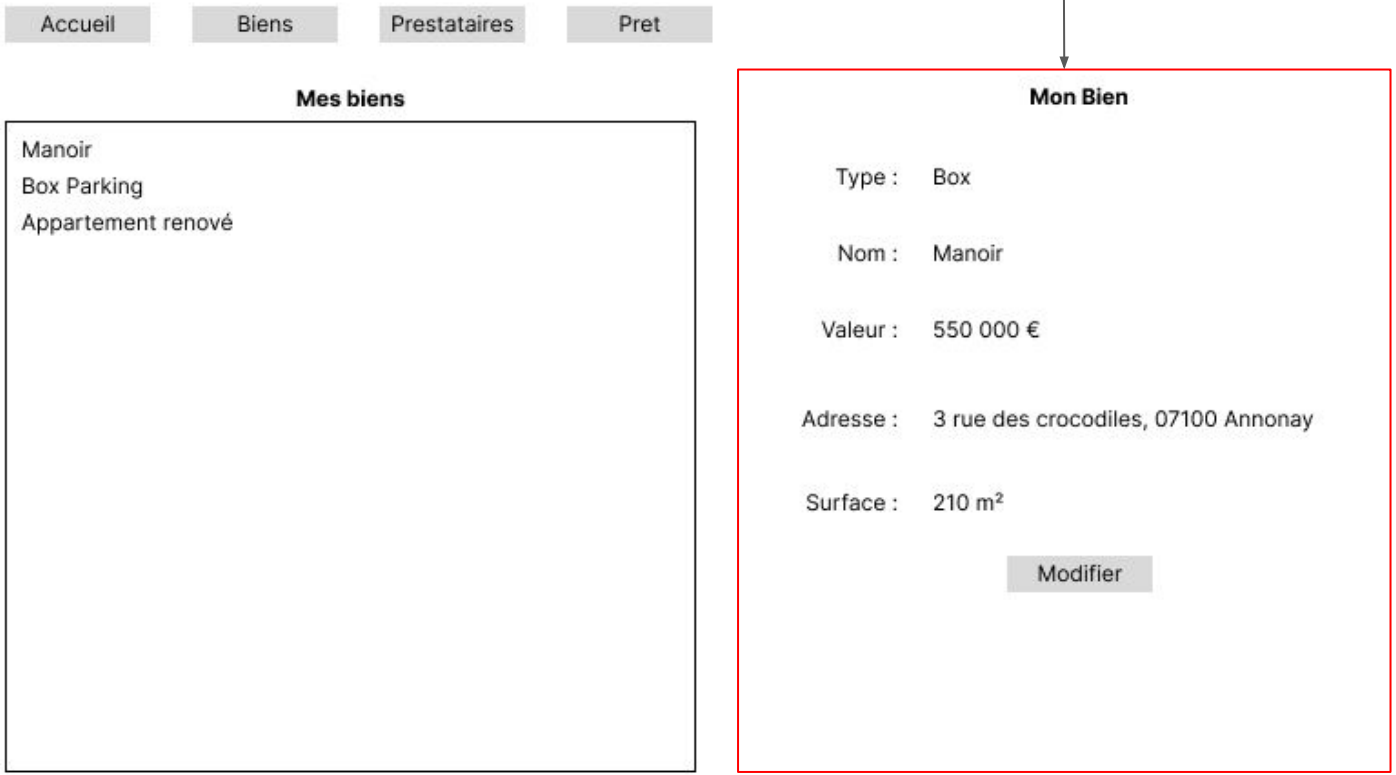
ComboBox pour choisir entre chaque type (Box, Appartement, Maison

Le Form change en fonction du type sélectionné :

- GererBoxForm
- GererMaisonForm
- GererAppartementForm

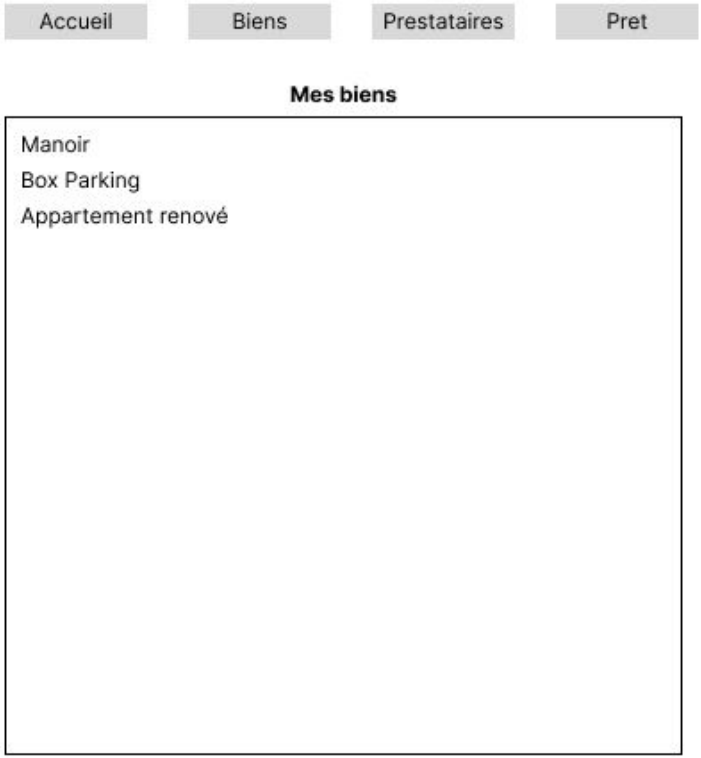
BienView

BienSingleView

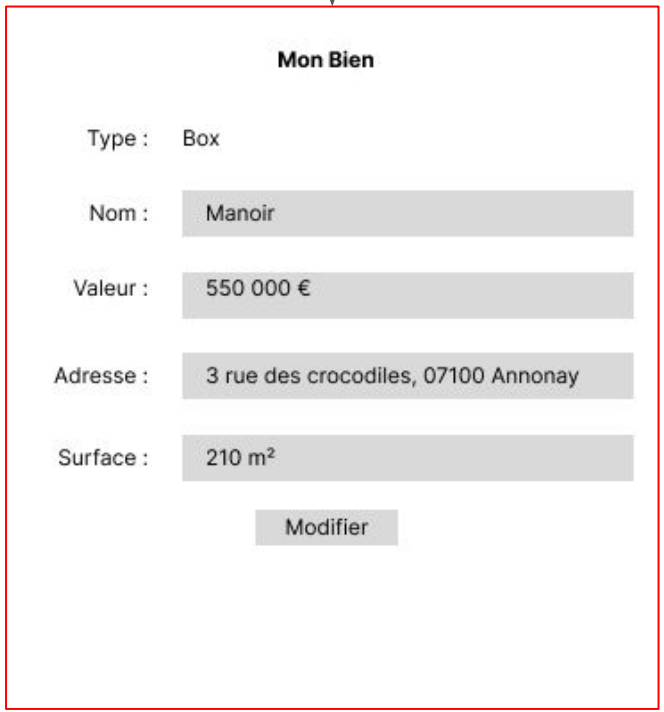


Lors du clic sur le nom d'un bien. Chargement de toutes les infos.

BienView



GererBoxForm

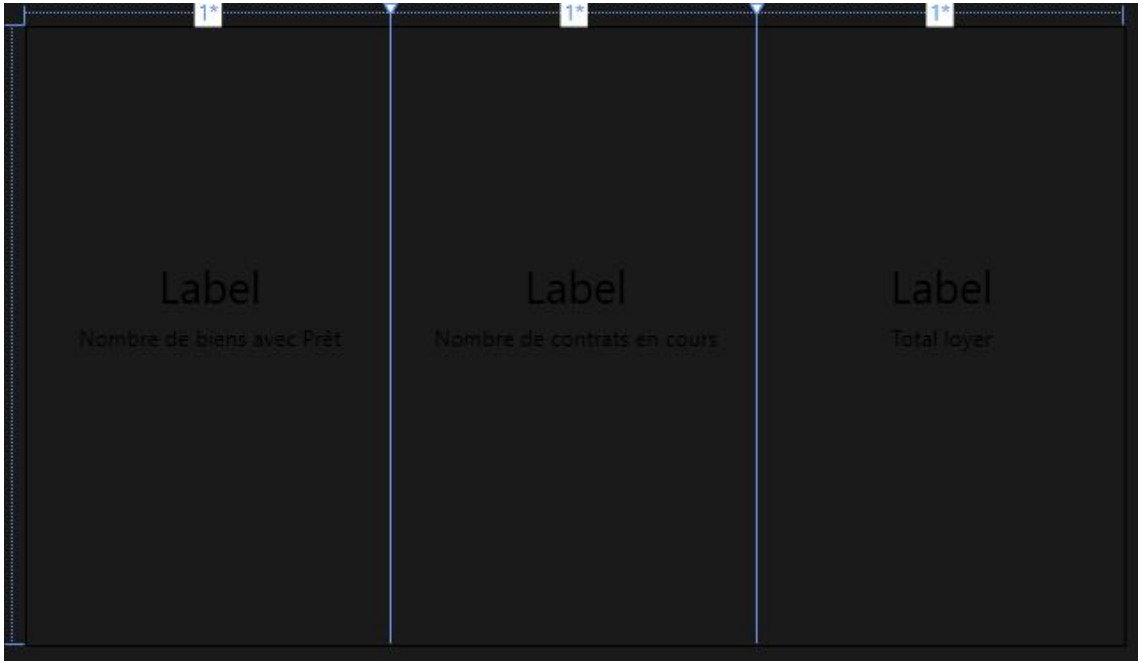


Lors du clic sur le bouton modifier.

Chargement de la bonne Form pour le type de Bien sélectionné

Mise en page WPF

Utilisation des grilles



3 colonnes avec des éléments centrés

Utilisation des grilles

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <Label x:Name="lblBienAvecPret" Content="Label" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="0" FontSize="32" Margin="0, 0, 0, 75"/>
    <Label Content="Nombre de biens avec Prêt" HorizontalAlignment="Center" VerticalAlignment="Center"   Grid.Column="0" FontSize="16"/>

    <Label x:Name="lblNbContrat" Content="Label"  Grid.Column="1" HorizontalAlignment="Center" VerticalAlignment="Center" Margin="0, 0, 0, 75" FontSize="32"/>
    <Label Content="Nombre de contrats en cours" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="1" FontSize="16"/>

    <Label x:Name="lblLoyer" Content="Label" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="2" Margin="0, 0, 0, 75" FontSize="32" />
    <Label Content="Total loyer" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Column="2" FontSize="16"/>
</Grid>
```

Utilisation des grilles

Pour fusionner 2 lignes : Grid.RowSpan="2"

Pour fusionner 2 colonnes : Grid.ColumnSpan="2"