**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**Львівський національний університет імені Івана Франка**

Факультет прикладної математики та інформатики

**Звіт**

**про виконання** **курсової роботи**

**з дисципліни “Програмне забезпечення”**

Студента

Групи ПМП-32

Стовби Олександра

Керівник роботи

Доцент Ящук Ю.О.

Львів – 2020

**Тема**

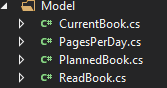
Читай книжки: список книжок, які вже прочитав, які планую читати, короткі враження про книжку, зберігати дані в базу, відображати кількість прочитаних сторінок графіком за певний період, нагадувати про заплановані книжки.

**Опис**

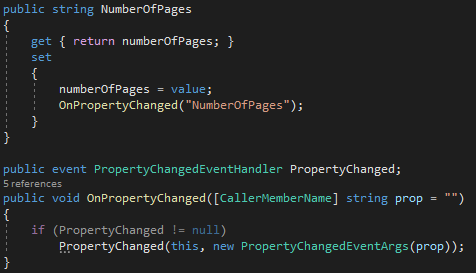
Проєкт розроблений на мові C# з використанням WPF та архітектурним шаблоном MVVM. Наявна робота з базою данних (Entity Framework) та SQlite.

Патерн **MVVM** (Model-View-ViewModel) дозволяє відокремити логіку додатку від візуальної частини . Даний патерн є архітектурним, тобто він задає загальну архітектуру програми.

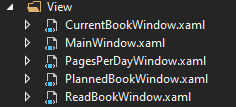
Model - модель описує використовувані в додатку дані. Тобто у папці Model реалізовані класи всіх сутностей, які є основними в проєкті.



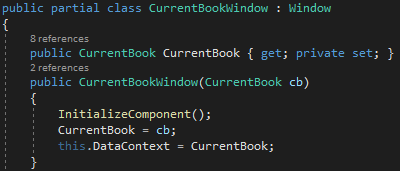
Модель реалізує інтерфейс INotifyPropertyChanged, який дозволяє повідомляти систему про зміни властивостей моделі. Завдяки цьому полегшується прив'язка до подання, хоча пряма взаємодія між моделлю і представленням відсутня.



View - визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до WPF уявлення - це код в xaml, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.



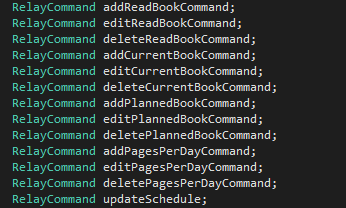
Хоча вікно (клас Window) в WPF може містити як інтерфейс в xaml, так і прив'язаний до нього код C #, проте в ідеалі код C # не повинен містити якийсь логіки, крім хіба що конструктора, який викликає метод InitializeComponent і виконує початкову ініціалізацію вікна . Вся ж основна логіка додатки виноситься в компонент ViewModel.



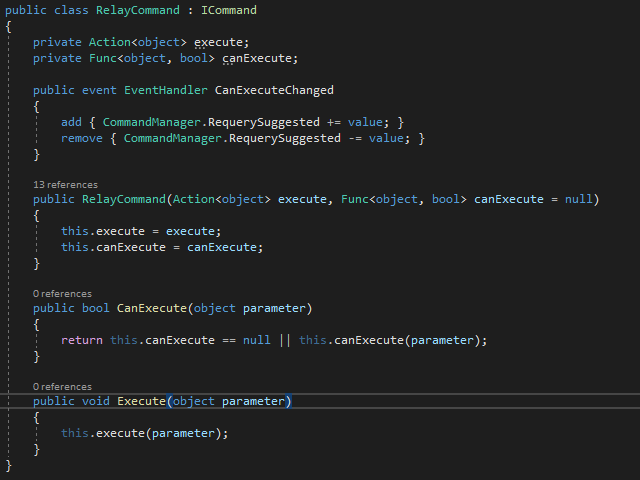
Однак іноді в файлі пов'язаного коду все може знаходитися певна логіка, яку важко реалізувати в рамках паттерна MVVM у ViewModel.

ViewModel – зв’язує Model i View через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в поданні.

Оскільки елементи уявлення, тобто візуальні компоненти типу кнопок, не використовують події, то View взаємодіє з ViewModel за допомогою команд.



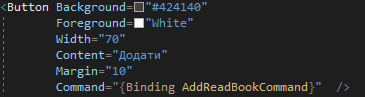
Для цього ми додатково маємо реалізувати клас RelayCommand, щоб створювати команди, які в MVVM використовуються замість кліків.



Наприклад, користувач хоче додати нову книгу. Він натискає на кнопку і тим самим відправляє команду під ViewModel. А ViewModel вже отримує передані дані і відповідно до них оновлює модель.



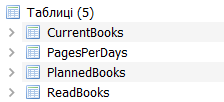
А в View додаємо для кнопки цю команду



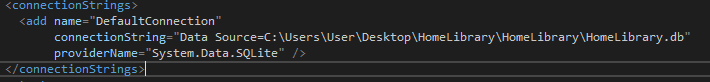
Підсумком застосування патерну MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в подальшому модифікувати і підтримувати.

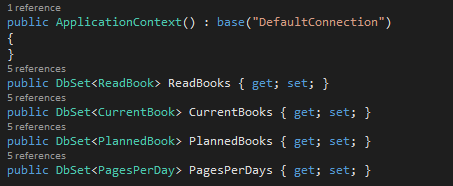
**Робота з базою данних**

У базі даних створено 4 таблиці з сутностями, які є в Model.



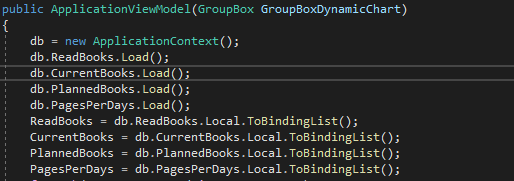
Підключаємо БД до проєкту і створюємо клас ApplicationContext, який буде головним чином взаємодіяти з БД.

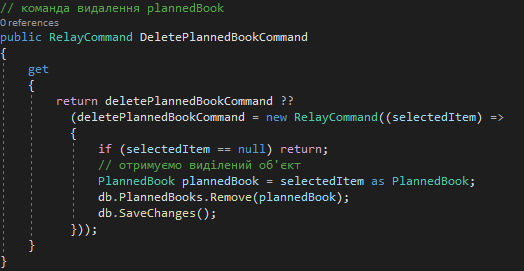




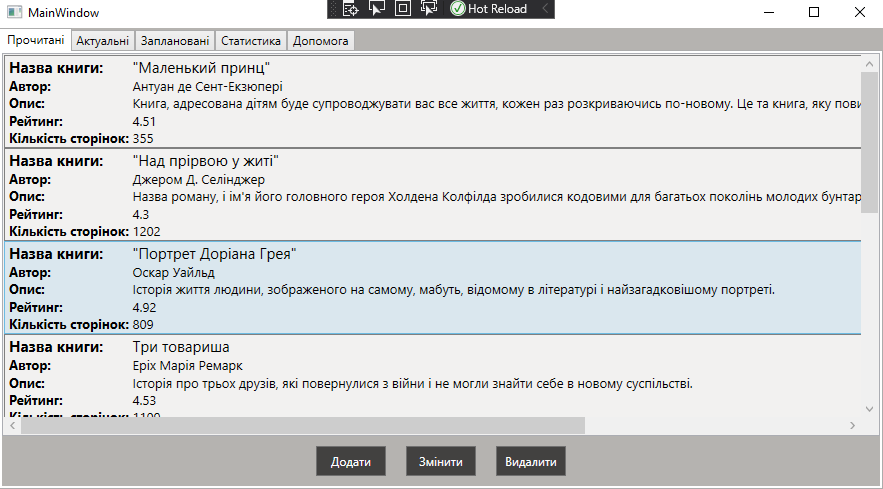
В цьому класі ми створюємо 4 поля з такими ж назвами як і таблиці в БД, щоб все працювало вірно.

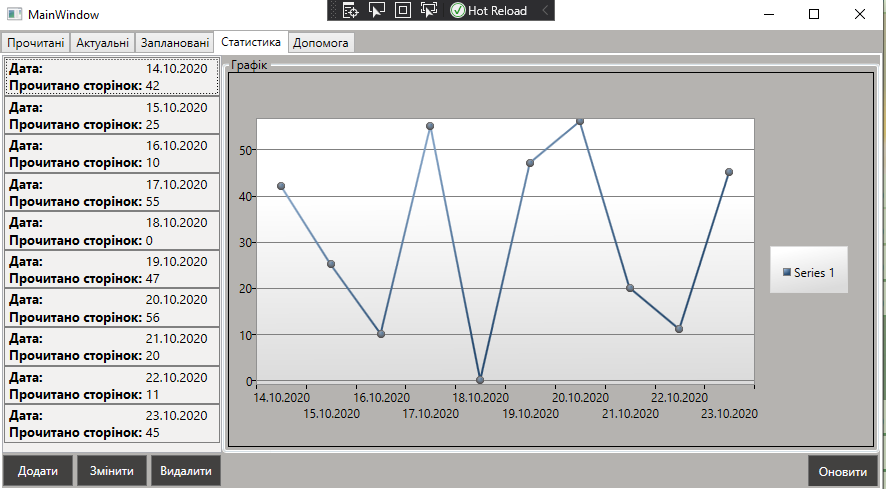
В класі ViewModel створюємо зміну bd і тепер можемо взаємодіяти з нею. Ми можемо загрузити данні з Бд методом Load, а також взаємодіяти з таблицями. Наприклад додавати, змінювати або видаляти данні.



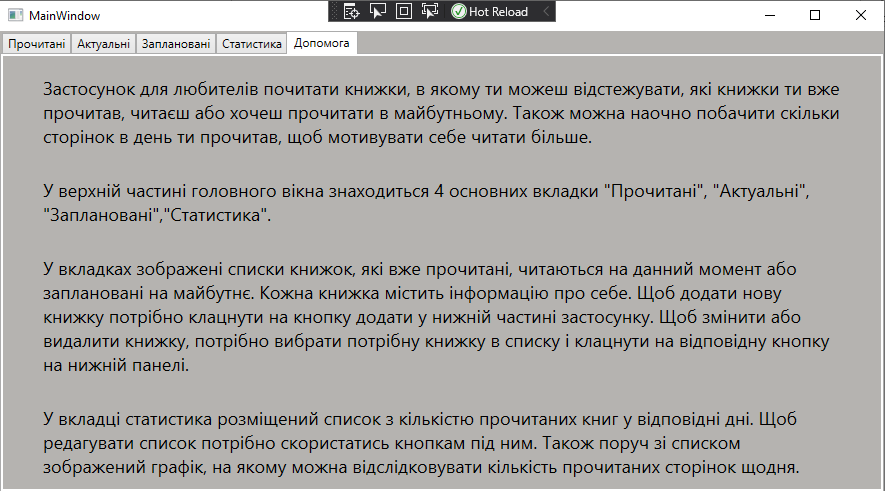


**Зовнішній вигляд програми**

****

****

Функціонал описаний у вкладці Допомога.

****