НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка
до бакалаврської роботи
на ступінь вищої освіти бакалавр
на тему: «**Розробка графічного редактора на платформі .NetFramework, використовуючи технологію WPF»**

Виконала: студентка 4 курсу, групи
ПД-41спеціальності:
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Грисюк В.Г.
(прізвище та ініціали)

Керівник Трінтіна Н.А.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Нормоконтроль Трінтіна Н.А.
(прізвище та ініціали)

Київ – 2021

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність – 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
інженерії програмного забезпечення

Негоденко О.В.

“____”_____2021 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Грисюк Владислав Геннадійович
(прізвище, ім’я, по батькові)

Тема роботи: «Розробка графічного редактора на платформі .NetFramework, 1. використовуючи технологію WPF»
Керівник роботи Трінтіна Наталія Альбертівна, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “12”березня 2021 року №65
2. Строк подання студентом роботи 01.05.2021
3. Вихідні дані до роботи:
Microsoft visual studio;
WPF;
GitHub;
SourceTree;
Prism;
Офіційна документація мови програмування C#.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу
1. Титульний слайд

2. Мета, об’єкт дослідження, предмет дослідження
3. Аналіз існуючих аналогів
4. Технічне завдання
5. Засоби програмної реалізації
6. Діаграма діяльності
7. Розроблений додаток
8. Висновки

6. Дата видачі завдання 19.04.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи до	Примітка
1	Вибір теми бакалаврської роботи	19.04.2021	виконав
2	Дослідження актуальності теми	20.04.2021	виконав
3	Розгляд аналогів	23.04.2021	виконав
4	Проектування логіки проекту	24.04-26.04	виконав
5	Конструювання основних сцен проекту	26.05-27.05	виконав
6	Реалізація функцій проекту	27.05-31.05	виконав
7	Здача роботи в деканат	01.06.2021	виконав

Студент _____ Грисюк В.Г.
(підпис) (прізвище та ініціали)
Керівник роботи_____ Трінтіна Н.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи с.57, рис. 54, джерел 12.

Об'єкт дослідження – програмне забезпечення для роботи з файлами мультімедії.

Предмет дослідження – додаток для редагування зображень.

Мета дослідження – розробити десктопний додаток для редагування зображень.

У роботі проведено аналіз існуючих рішень програмного забезпечення, які дозволяють роботу з файлами мультімедії. Розглянуто використання програмних засобів, аналіз їх переваг та недоліків. Було вибрано найкращі програмні інструменти для створення додатку.

Проведено описання використаних програмних засобів та середи розробки.

Реалізована система на мові програмування C#, на основі графічного інтерфейсу .NET Framework, з використанням технології WPF (Windows Presentation Foundation).

Ключові слова: C#, .Net, WPF, XAML, Prism, MVVM, SourceTree, Об’єктно орієнтоване програмування.

ЗМІСТ

ВСТУП

Об'єкт дослідження – програмне забезпечення для роботи з файлами мультімедії.

Предмет дослідження – додаток для редагування зображень.

Мета дослідження – розробити десктопний додаток для редагування зображень.

Новизна проекту – удосконалення функціоналу існуючих додатків, розробка додатку без платного функціоналу та інтерфейсу, який зрозумілий та очевидний для кінцевого користувача.

У дипломному проекті було проведено аналіз між існуючими аналогами фоторедакторів та проведено аналіз недоліків.

Додаток був розроблений на мові програмування C#, на основі графічного інтерфейсу .NET Framework з використанням техології WPF та мови XAML. Всі об’єкти інтерфейсу були створені в графічному редакторі Adobe Photoshop. Додаток має внутрішню ієрархічну структуру, кожен компонент логічно і програмно пов’язаний із суміжним до нього об’єктом. Основними об’єктами меню є кнопки з різним призначенням і візуальним оформленням.

Досліджено та створено додаток, що дозволяє редагувати зображення.

1.АНАЛІЗ ТА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Актуальність

Невід’ємна частина нашого життя – це соціальні мережі.

Перше, що ми робимо коли виїжджаємо на відпочинок, неважливо, на море чи просто на пікнік з друзями - це фотографії.

У наш час більшість людей викладають своє особисте життя в соціальну мережу. Це стало місцем де люди можуть поділитися своїми поглядами, думками, або просто новими фотографіями зі свого курорту. І, звичайно, для цього люди роблять фотографії на кожному кроці. Але, на жаль, далеко не кожне фото виходить так як ми хочемо, тому в наш час стали популярні різні редактори фотографій, такі як Adobe Photoshop, в яких ми може прибрати людину, яка заважає нам в кадрі, або замазати невеликі дефекти нашого обличчя.

\

1.2 Adobe Photoshop

На сьогоднішній день існує дуже багато різних графічних редакторів. Найпопулярніший – це Adobe Photoshop рис. 1.1.

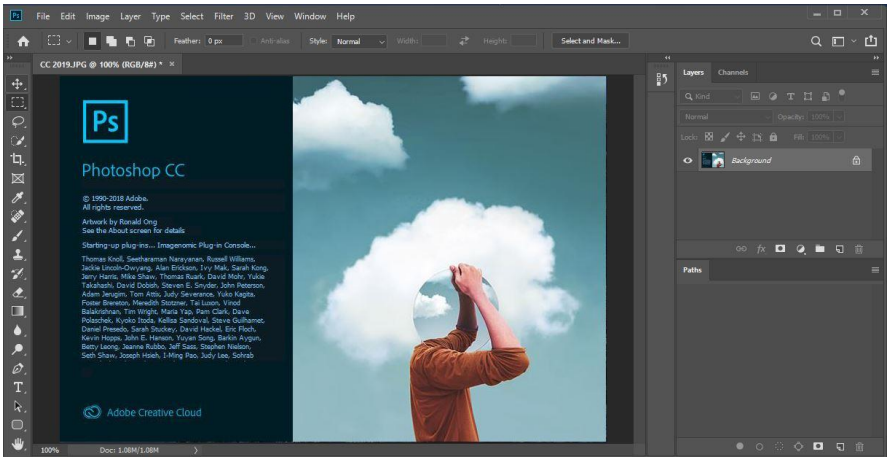


Рисунок 1.1 Adobe Photoshop

Adobe Photoshop – багатофункціональний графічний редактор, розроблений і поширюваний фірмою Adobe Systems. Цей продукт є лідером ринку в галузі комерційних засобів редагування растрових зображень і найвідомішим продуктом фірми Adobe. Часто цю програму називають просто Photoshop (Фотошоп). У наш час Photoshop доступний на платформах Mac OS X/Mac OS і Microsoft Windows. Ранні версії редактора були портовані під SGI IRIX, але офіційна підтримка була припинена, починаючи з третьої версії продукту. Для версії CS і CS6 можливий запуск під Linux за допомогою альтернативи Windows API – Wine.

Фотошоп має дуже великий функціонал, але це негативно влпиває на продуктивності додатка. Це, як і те, що він платний, робить його не дуже хорошим вибором для звичайної людини, зазвичай, такий інструмент як фотошоп використовують професійні художники або фотографи.

1.3 Paint Tool SAI 2

Paint Tool SAI – легкий растровий графічний редактор та програмне забезпечення для малювання для Microsoft Windows, розроблене та опубліковане Systemax Software рис.1.2.

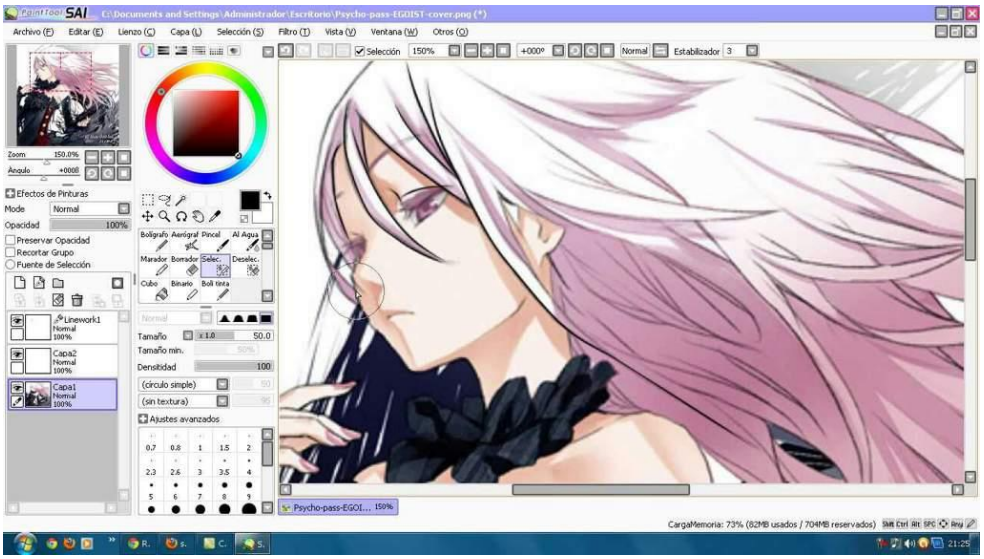


Рисунок 1.2 Paint Tool SAI

Це також досі потужний додаток, який полюбляють художники, за велику кількість інструментів, та їх гнучного налаштування. Але ця програма має ряд недостач, такі як платна підписка, та погана сумісність з Windows 10.

1.4 Постановка технічного завдання

Після проведення аналізу аналогів та визначивши їх недоліки, було вирішено створити додаток на мові c#. Це одна з найкращих мов програмування для створення десктопних рішень з гарним графічним інтерфейсом.

Також важливо обрати середовище розробки серед:

1. Eclipse.
2. IntelliJ IDEA.
3. Microsoft Visual Studio 2019.

Eclipse - це безкоштовне середовище розробки від не комерційної організації Eclipse Foundation. Узагальнюючи, програма – це основа для підключення модулів.

Eclispe достатньо зручне середовище, яке можна використовувати навіть на слабких ПК. Вона містить декілька мов інтерфейсу, включаючи такі мови як російська та українська. Також Eclipse має багато доповнень (робота з БД, сервером, тощо).

IntelliJ IDEA – також популярне середовище. Воно доступне у двох версіях – платній да безкоштовній. Порівнюючи з попереднім середовищем зі списку, IntelliJ IDEA має більше можливостей для відлагодження коду, та більш зручний інтерфейс. Але розробка на мові C# у цьому середовищі може викликати ряд труднощів, якщо ви розробляєте додаток з графічним інтерфейсом, тому що IntelliJ IDEA не має вікна конструктора, тому увесь інтерфейс треба писати у коді.

Microsoft Visual Studio 2019 – це найкращий вибір для розробки десктопного додатку на платформі .NET використовуючи мову програмування C# з графічним інтерфейсом, оскільки і то, і то є продуктами компанії Microsoft і вони відмінно підходять для роботи з один одним.

Є безкоштовна версія Community, яка підтримує всі інструменти розробки для

мови c#, яка ідеально підійде для роботи. Також дане середовище підтримує IntelliSense для оптимізації роботи коду.

А також для удосконалення стандартного IntelliSense ми будемо використувавоти безкоштовну версію плагіну ReSharper.

ReSharper - додаток (add-on), розроблений компанією JetBrains для пришвидшення роботи та автоматизації рефакторингу в середовищі Microsoft Visual Studio. Цей плагін підтримує усі версії Visual Studio.

Здійснює миттєвий статичний аналіз коду (без потреби компіляції), передбачає додаткові засоби пошуку, автозаповнення, навігації, форматування, оптимізації та генерації коду, надає близько 40 автоматизованих рефакторингів, спрощує модульне тестування в таких середовищах як MSTest та NUnit.

Microsoft Visual Studio підтримує UI фреймворк WPF, який є ідеальним рішенням для розробки десктопних додатків. Він дозволить реалізувати всі наступні вимоги до додатку:

1. Можливісь відкривати зображення
2. Можливість додавати фігури до полотна
3. Можливість редагувати такі параметри як яскравість, контраст та гамму.
4. Управління елементами, такі як поворот та перенесення
5. Робота з шарами

WPF дозволить реалізувати адаптивний та гарний інтерфейс, за допомогою мови розмітки XAML.

В ході розробки та тестування додаток буде супроводжуватися буде за допомогою системи контроля версії та управління версіями GIT, використовуючи візуальний додаток SourceTree.

2. ОПИС ПРОГАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Мова програмування C#

C# – сучасний об'єктно-орієнтований мова програмування. C# належить сімейству мов C, і може здатися досить знайомим кожному, хто працював з такими мовами як C, C++, Java .

C# належить до об’єктно-орієнтованої мови, але на ній можна використовувати також і компонентно-орієнтоване програмування. Розробка додатків з кожним днем набуває все більшого розташування до створення програмних компонентів у вигляді автономних пакетів, що допомагають реалізувати окремі функціональні можливості. Головна особливість таких компонентів в тому, що вони являють собою модель програмування з властивостями, методами і подіями. У них є атрибути, що надають декларативні відомості про компоненті. Вони включають всебе власну документацію. C# надає мовні конструкції, безпосередньо підтримують таку концепцію роботи. Завдяки цьому C# підходить для того, щоб створювати та застосовувати програмні компоненти.

C# в значній мірі підтримується Microsoft. Нові функції, синтаксичні конструкції та баги оновлюються швидше, ніж в інших мовах програмування. Ця мова є однією з найпопулярніших мов програмування коли мова йде про додатки з графічним інтерфейсом. Це важливо для розробників, оскільки популярність мови прямо пропорційна тому, наскільки для нього будуть доступні онлайн-матеріали. Найчастіше, інформацію можна знайти в Google або Stack Overflow, для вирішення завдань в розробці і найчастіше можна знайти велику кількість відповідей по C#. Це заощадить велику кількість часу новачкам і, навіть професіоналам, при вирішенні різних завдань в розробці.

Також свою популярність C# набув за рахунок гнучкості, якщо порівнюватиз деякими іншими мовами програмування. Є велика кількість різних типів програм, якіможуть бути написані за допомогою C#, .Net і Visual Studio:

1. Додатки для Windows.
2. Мобільні додатки.
3. Веб-додатки.
4. Ігри.
5. Додатки для Android і ios, які розробляються за допомогою додаткових фреймворків, таких як Xamarin або Mono.

Звичайно, всі ці речі можливо виконувати і за допомогою інших мов програмування, але також, в таких випадках, використовуються сторонні інструменти інших розробників. Розробники, які працюють з C# достатньо довго мають вже набір інструментів, який вони добре знають, які підтримуються Microsoft для розробки різних типів програми.

2.2 WPF

WPF - розшифровується як Windows Presentation Foundation, - це підхід Microsoft до фреймворку GUI, що використовується з фреймворком .NET рис. 2.1.

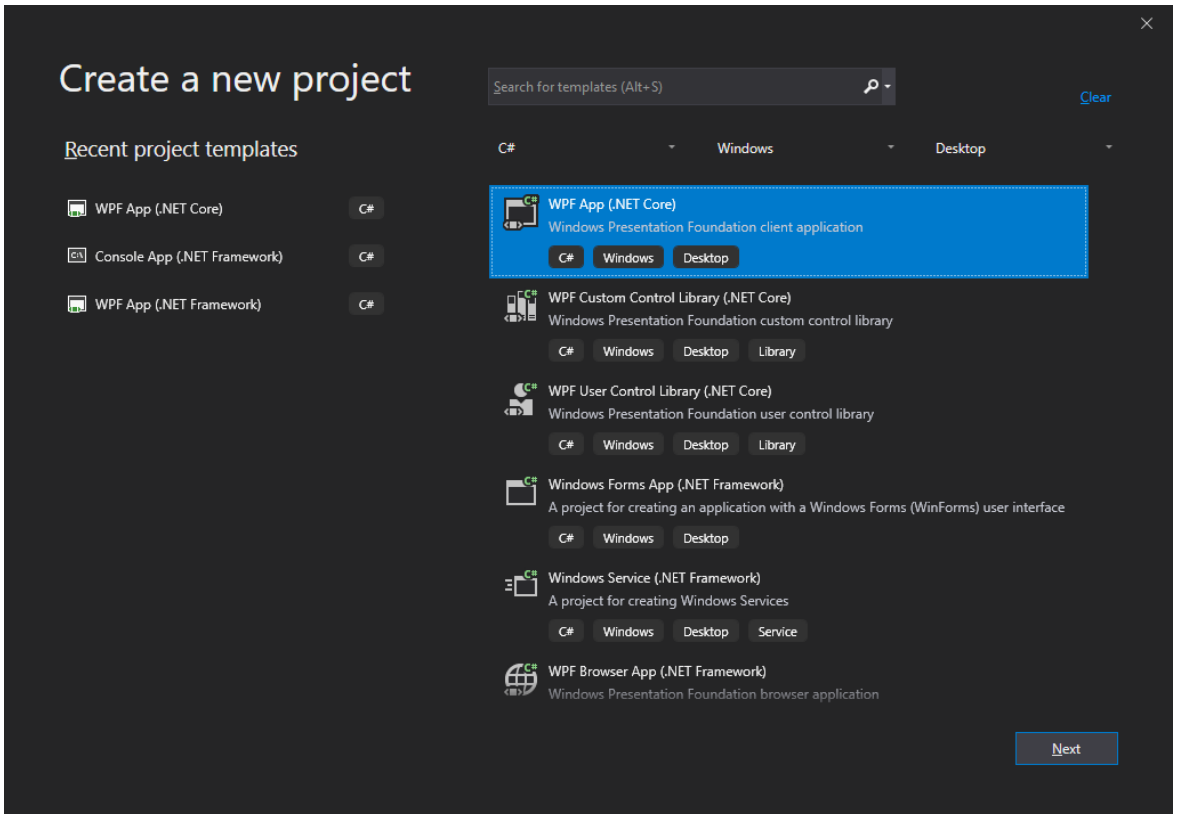


Рисунок 2.1 – Створення проекту використовуючи шаблон WPF

Але що таке GUI? GUI розшифровується як Графічний користувальницький інтерфейс. Windows має графічний інтерфейс для роботи з вашим комп'ютером.

Графічний інтерфейс дає змогу створювати програму з великою кількістю графічних елементів, які, зазвичай, можна швидко та зручно змінити під себе.

Без нього розробник повинен був би малювати сам увесь інтерфейс, та програмувати усі події, як, наприклад, натискання або наведення миші. Це БАГАТО роботи, тому натомість більшість розробників використовуватимуть графічний інтерфейс, який виконуватиме всю основну роботу і дозволить розробникам зосередитися на створенні чудових додатків.

Існує багато різних типів графічних інтерфейсів, але для розробників .NET найбільш поширеними на даний момент є найпопулярніший WinForms та WPF. WPF є найновішим, але Microsoft все ще розвиває та підтримує WinForms через те, що на ньому досі пишуть багато додатків. Між двома фреймворками є досить багато відмінностей, але їх призначення однакове - полегшити створення додатків із графічним інтерфейсом.

WPF - це механізм, який відповідає за створення, відображення та маніпулювання користувальницькими інтерфейсами, документами, зображеннями, елементами керування та медіа в операційних системах Windows 7,8 та пізніших версіях. WPF - це набір бібліотек, які мають усі функції, необхідні для створення, запуску, запуску програмам Windows.

XAML - це нова описова мова програмування, розроблена Microsoft для написання графічних користувацьких інтерфейсів для керованих додатків, з зручною кастомізацією елементів.

XAML використовується для побудови користувальницьких інтерфейсів для Windows та мобільних додатків, які використовують Windows Presentation Foundation (WPF), UWP та Xamarin Forms – мобільних пристроїв. Призначення XAML – створити користувальницькі інтерфейси за допомогою мови розмітки, схожої на XML. Здебільшого ви будете використовувати конструктор для створення XAML, але ви можете вільно використовувати елементи XAML вручну.

XAML використовує формат XML для елементів та атрибутів. Кожен елемент у XAML представляє об'єкт, який є екземпляром типу. Сфера застосування типу (клас, перерахування тощо) визначається простором імен, який фізично знаходиться у збірці (DLL) бібліотеки .NET Framework. Подібно до XML, синтаксис елемента XAML завжди починається з відкритої кутової дужки (<) і закінчується закритою кутовою дужкою (>). Кожен тег елемента також має початковий та кінцевий теги рис. 2.2.

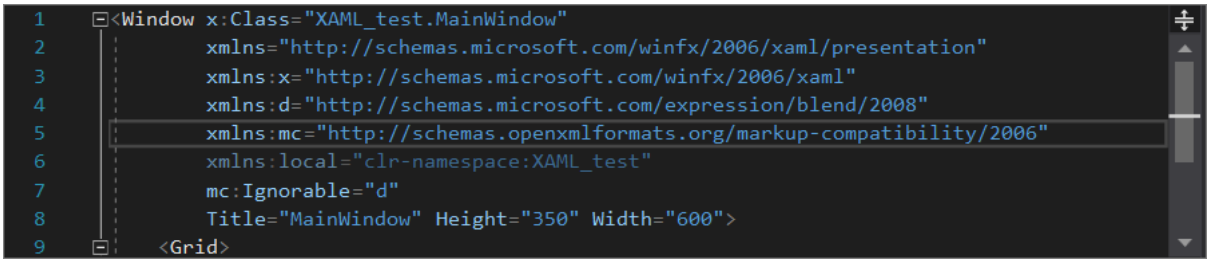


Рисунок 2.2 – приклад використання XAML

Хоча XAML використовується для побудови користувальницьких інтерфейсів у WPF, C# використовується як кодова мова в WPF. Хоча Windows та їх елементи створюються безпосередньо в XAML під час проектування, вони також можуть бути створені під час компіляції методами мови C#.

C# також використовується для написання програмування всіх подій та бізнес-логіки. Усі дії, події та візуалізація виконуються в коді, що стоїть за кодом C#.

WPF підтримує два типи ресурсів, статичні ресурси та динамічні ресурси. Статичні ресурси: Коли XAML завантажується, статичні ресурси будуть виконуватися до фактичного запуску програми. Він буде виконаний лише один раз, а будь-які зміни до словника ресурсів будуть проігноровані.

Динамічні ресурси: Динамічним ресурсам призначаються значення атрибутів під час запуску, але насправді їх не шукають до часу виконання. Це призведе до затримки пошуку ресурсу, поки він не буде необхідний під час виконання. Елемент “Style” рис.2.3 у XAML представляє стиль

“Style” - це спосіб групувати подібні властивості в одному об’єкті “Style” та

застосовувати до кількох об’єктів. Зазвичай стиль додається до ресурсів FrameworkElement. Даний вираз “x:” є унікальним ідентифікатором ключа стилю.

```
<Window.Resources>
        <Style x:Key="MyButtonStyle">
            <Setter Property="Control.FontFamily" Value="Calibri"></Setter>
            <Setter Property="Control.FontSize" Value="18"></Setter>
            <Setter Property="Control.FontWeight" Value="Bold"></Setter>
            <Setter Property="Control.Padding" Value="5"></Setter>
            <Setter Property="Control.Margin" Value="5"></Setter>
        </Style>
</Window.Resources>
```

Рисунок 2.3 – Використання стилю у WPF

Шаблони є невід’ємною частиною дизайну інтерфейсу користувача в WPF. WPF має наступні три типи шаблонів: **Control Template, Items Panel Template та Data Template**.

Control Template або **Шаблон керування** визначає зовнішній вигляд елемента інтерфейсу. Ми можемо встановити новий зовнішній вигляд елемента керування, просто змінивши Control Template елемента управління. Також, даний шаблон ще більш корисний, коли ви пишете власні елементи керування. Ви можете визначити зовнішній вигляд елементів керування. Наприклад, елемент керування кнопкою WPF має прямокутний макет, але за допомогою **ControlTemplates** ви можете створити власну кнопку, яка має круговий макет і змінює свій колір, коли ви наводите курсор миші на неї або натискаєте.

Items Panel Template – визначає стиль макету кастомного або готового елементу.

Data Template – шаблон, який визначає стиль відображення данних.

Також, у WPF доступна прив’язка данних.

Прив’язка дозволяє зв’язати вихідний об’єкт з деяким елементом управління. Існують такі типи прив’язок:

1. Одностороння прив'язка: прив'язує джерело до інтерфейсу.
2. Двостороння прив'язка: прив'язує джерело до інтерфейсу і назад до джерела.

Інтерфейс **INotifyPropertyChanged** дозволяє джерелам взаємодіяти з

інтерфейсом та оновлювати його при зміні значень. Для прив’язки об’єкта або списку до елемента потрібно встановити властивість **DataContext**. Ви можете прив’язати об’єкт або список об’єктів, або можете прив’язати один елемент до іншого. Ви можете встановити DataTemplate з елемента керування, щоб налаштувати спосіб відображення відповідних даних.

У WPF розвинена система тригерів, як в C#.

Тригери використовуються для виконання певних дій при дотриманні зазначених умов. Тригери використовуються для створення візуальних ефектів на елементах управління та елементах фреймворку. Тригери є частиною стилів і завжди визначаються усіма стилями.

В основному існує три типи тригерів, це:

1.Тригер власності.

2.Тригер даних.

3.Тригер подій.

WPF має декілька класів для роботи з зображеннями - Canvas та Image. Canvas має більше можливостей, але використовує багато ресурсів, то ж ми будемо використовувати Image, а редагувати його безпосередньо за допомогою класу WriteableBitmap.

WPF має розширені засоби управління інтерфейсом. Ці елементи керування підтримують такі візуальні дії, як перетягування, налаштування структур живлення та розділів, введення даних та створення ресурсів та шаблонів.

Список основних елементів керування, які підтримує WPF:

1. DatePicker Control.
2. DockPanel Control.
3. ListBox Control.
4. ComboBox Control.
5. MessageBox Control.
6. DataGrid.
7. ProgressBar.
8. TreeView.

Елемент керування **DatePicker** рис.2.5 використовується для створення візуального DatePicker, який дозволяє користувачеві вибрати дату та запускати подію при виборі дати. У цій статті показано, як створити та використовувати елемент керування DatePicker у WPF за допомогою XAML та C #.

Рисунок 2.5 – Приклад використання DatePicker Control

DockPanel використовується для стикування дочірніх елементів у лівому, правому, верхньому та нижньому положеннях відповідних елементів рис.2.6.

Положення дочірніх елементів визначається властивістю Dock відповідних дочірніх елементів та відносним порядком цих дочірніх елементів. Залишається значення за замовчуванням властивості Dock.

Властивість Dock має тип enumDock (перерахування), що має значення Left, Right, Top і Bottom.

```
<DockPanel Name="dcPanel">
    <Button Name="TopRect" DockPanel.Dock="Top" Background="LightGreen" Height="50" Content="Top
    <Button Name="LeftRect" DockPanel.Dock="Left" Background="LightBlue" Width="50" Content="Lef
    <Button Name="RightRect" DockPanel.Dock="Right" Background="LightSalmon" Width="50" Content=
    <Button Name="BottomRect" DockPanel.Dock="Bottom" Background="LightCyan" Height="50" />
    <Button Name="Fill" Background="LightGray" />
</DockPanel>
```

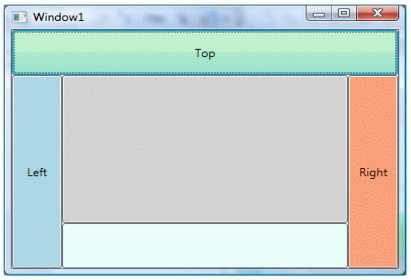


Рисунок 2.6 – Приклад використання DockPanel

WPF ListBox - Елемент XAML ListBox рис.2.7 предствляє можливість створення листів елементів. Властивості Width і Height представляють ширину та висоту ListBox. Властивість Name представляє ім'я елемента керування, який є унікальним ідентифікатором елемента управління. Властивість Margin вказує розташування ListBox на батьківському елементі управління.

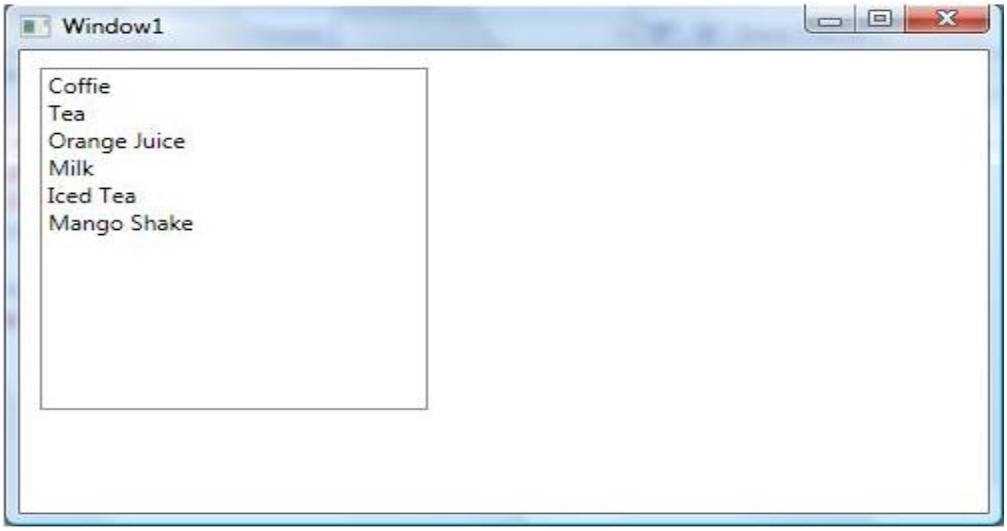


Рисунок 2.7 – Приклад використання ListBox

ComboBox - це елемент керування, який працює як елемент керування ListBox, але одночасно видно лише один елемент із колекції, і натискання на ComboBox робить колекцію видимою і дозволяє користувачам вибрати елемент із колекції. На відміну від елемента керування ListBox, ComboBox не має декількох елементів вибору. Елемент керування

ComboBox - це комбінація трьох елементів керування - кнопки, спливаючого вікна та текстового ящика рис.2.8. Елемент керування кнопкою використовується для показу або приховування доступних елементів, а елемент керування спливаючого вікна відображає елементи та дозволяє користувачеві вибрати один елемент зі списку. Потім елемент керування TextBox відображає вибраний елемент.

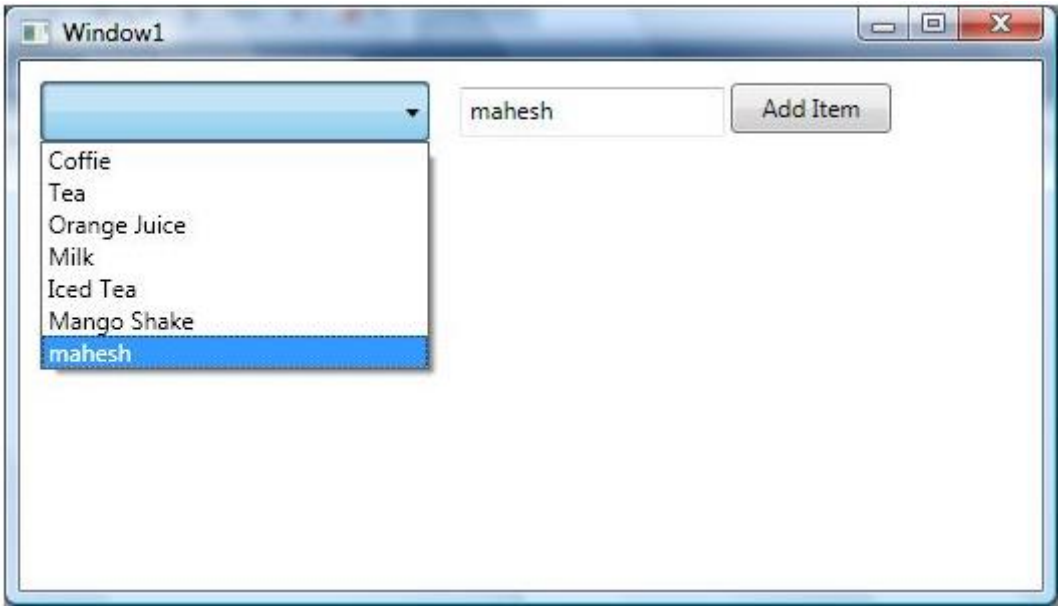


Рисунок 2.8 – Приклад використання ComboBox

Клас **MessageBox** у WPF представляє діалогове вікно модального вікна повідомлення, яке визначено у просторі імен System.Windows рис.2.9.

MessageBox.Show - єдиний метод, який використовується для відображення вікна повідомлення. Метод Show повертає перерахування MessageBoxResult, яке має значення None, OK, Cancel, Yes і No. Ми можемо використовувати MessageBoxResult, щоб визначити, на яку кнопку було натиснуто MessageBox, і вжити відповідних дій.

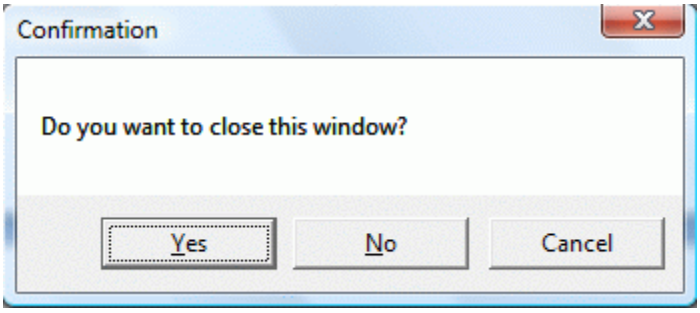


Рисунок 2.9 – Приклад використання MessageBox

2.3 GIT

Що таке "контроль версій"? Контроль версій - це система, яка фіксує зміни у файлі чи наборі файлів з часом, щоб певні версії можна було відновити пізніше. Якщо ви розробник і хочете зберегти поточну версію вашої програми, тоді система контролю версій дуже зручна для цього завдання. Ця технологія дозволяє відновити вибрані файли, повернути весь проект, порівняти зміни з часом, подивитися, хто останній змінив певний файл, що могло спричинити проблему, хто зробив проблему та коли, та багато іншого.

СКВ дає змогу працювати над одним проектом одразу декільком розробникам без шкоди один одному. Існують три типи СКВ: локальна, централізована та розподілена. Далі розглянемо більш детально розподілену систему контролю версій.

Розподілена система контролю версій – у даній системі, користувачі повністю копіюють репозиторій на свій ПК. Тобто, у кожного розробника наявна копія всього початкового коду та змін, котрі були в нього внесені під час розробки проекту.

Це означає, якщо один із серверів вийде з ладу, будь-який інший клієнтський репозиторій може бути скопійований на інший сервер для продовження роботи, без втрати даних. Схема розподільної системи контролю версій зображено на рисунку 2.12

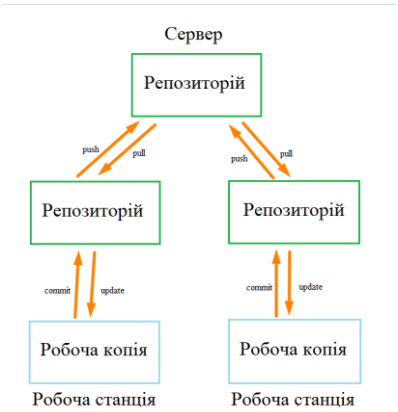


Рисунок 2.12 – Схема розподільної системи контролю версій

Git - це розподілена система контролю версій, яка дозволяє розробникам змінювати файли та співпрацювати з іншими розробниками. Найвідомішими особливостями Git є простий дизайн, швидкість, підтримка нелінійної упаковки, повна децентралізація та якщо необхідно, можливість роботи з великими проектами, які підтримуються декількома командами.

GitHub - це послуга хостингу онлайн-сховищ, яка включає всі функції, які підтримує Git - розподілені функції контролю версій та функції управління вихідним кодом. Використання GitHub з Git надає розробникам можливість зберігати код через Інтернет та легко взаємодіяти з іншими розробниками. Важливою та зручною особливістю розробника є історія змін, яку просто відстежувати, що саме було змінено. В історії змін або коммітів виділяються файли та частини коду, в які були внесені зміни (рис. 2.13). Можливість порівняння частин коду полегшує пошук можливих помилок та налагодження коду.

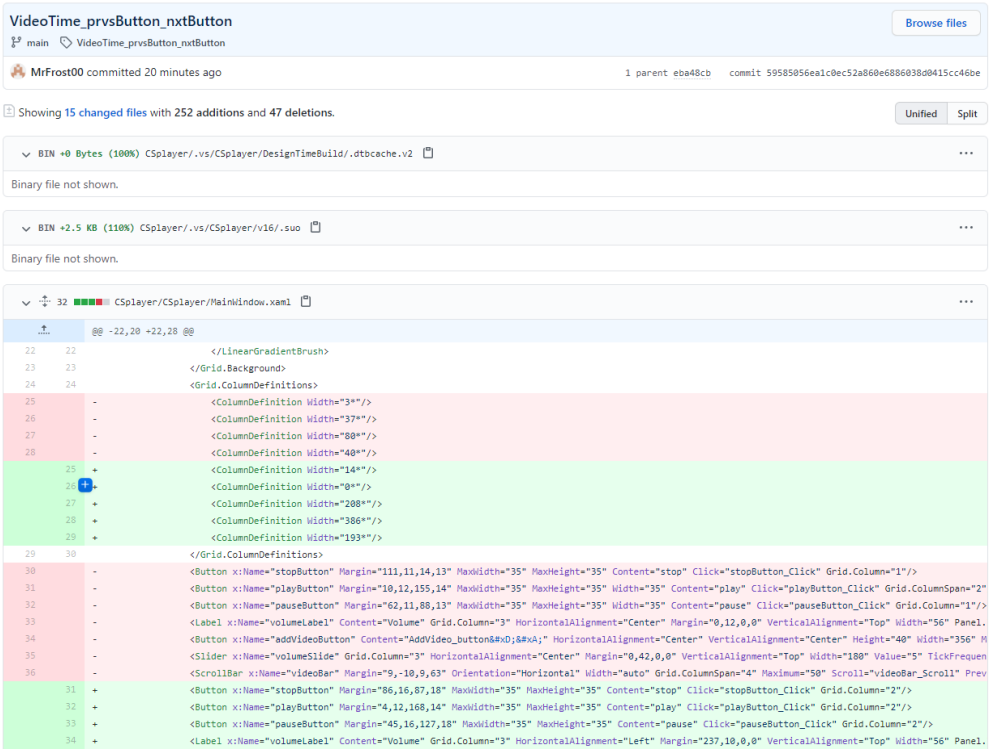


Рисунок 2.13 – Вигляд коміту на сервісі GitHub

То що таке Git у двох словах? Головна відмінність між Git та будь-якою іншою системою контролю версій полягає в тому, як Git працює зі своїми данними. Концептуально більшість інших систем зберігають інформацію як список змін файлів. Ці інші системи (CVS, Subversion, Perforce, Bazaar одночасно) створюють інформацію, яку вони зберігають як набір файлів та зміни, внесені до кожного файлу з часом (це називається контролем версій на основі деталей). Git не думає і не зберігає свої дані таким чином. Фактично Git думає про свої дані більше, як про серію зменшених мініатюрних файлових систем. Кожен раз, коли ви фіксуєте або зберігаєте стан свого проекту, Git в основному робить фотографії того, як він переглядає всі ваші файли одночасно, і зберігає посилання на цей знімок. Щоб ефективніше працювати, якщо файли не були змінені, Git більше не зберігає файл, а лише посилання на попередній файл ідентифікатора, який він уже зберігає. Git думає про свою інформацію більше, ніж про потік зображень. Це важлива відмінність між Git та майже всіма іншими системами контролю версій. Це робить Git більш схожим на міні-файлову систему з неймовірно потужними інструментами, побудованими поверх неї, а не просто на системою контролю версій.

Все в Git має контрольну суму перед тим, як зберігатись, а потім посилається на цю контрольну суму. Це означає, що неможливо змінити вміст будь-якого файлу чи каталогу, щоб git не взнав про це. Ця функціональність вбудована на найнижчих рівнях і є невід'ємною частиною. Ви не можете втратити інформацію або отримати пошкодження файлів. Механізм, який Git використовує для цього контрольного підсумовування, називається хешем SHA-1. Це 40-символьний рядок, що складається з шістнадцяткових символів (0–9 та a – f) і обчислюється на основі вмісту файлу або структури каталогів у Git.

Це надає додатковий захист користувачів гіт і робить його унікальним серед інших систем контролю версій, оскільки, при роботі декількох людей з одним розробником, дуже важко втратити дані, для кожного, оскільки у кожного є свій локальний репозіторій, а контрольна сума відстежує, щоб дані файли не були втрачені.

2.4 SourceTree

Хоч гіт і дуже зручний в плані супроводження проекту, у нього є свої недоліки, один з них – постійні десятки консольних команд, які забирають багато часу та не завжди коректно працюють, оскільки, якщо була допущена помилка – взнати де вона і чому вона виникла – це доволі складна задача.

На допомогу у вирішені цих проблем приходить SourceTree.

SourceTree було створено для того, щоб зробити Git доступним для кожного розробника - особливо тих, хто новачок у Git. Кожна команда Git - це лише один клік за допомогою інтерфейсу SourceTree. Даний додаток спрощує наступний функціонал:

1. Створення та клонування репозиторіїв з будь-якого місця.
2. Commit, push, pull та merge в декілька кліків
3. Розвинена система пошуку помилок та конфліктів.
4. Відстежування всього життєвогу циклу проекту.

Як вже зазначалось, супроводження реалізації додатку буде проводитись за допомогою GitHub. Використання SourceTree спрощує роботу з Гіт до декількох кліків рис.2.14.

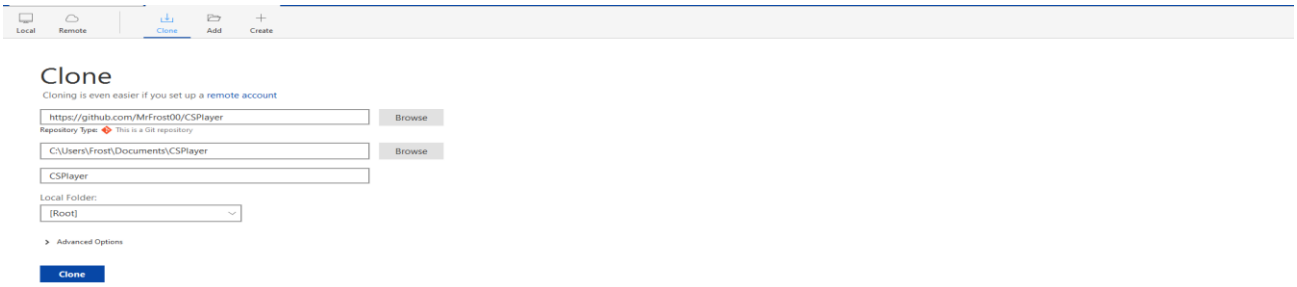


Рисунок 2.14 – Клонування\створення репозиторію

Одним з основого функціоналу гіта є – комміт внесених у проект змін, при

звичайному використанні прийшлось би писати декілька консольних команд, у SourceTree все зводиться до 2 кліків рис.2.15.

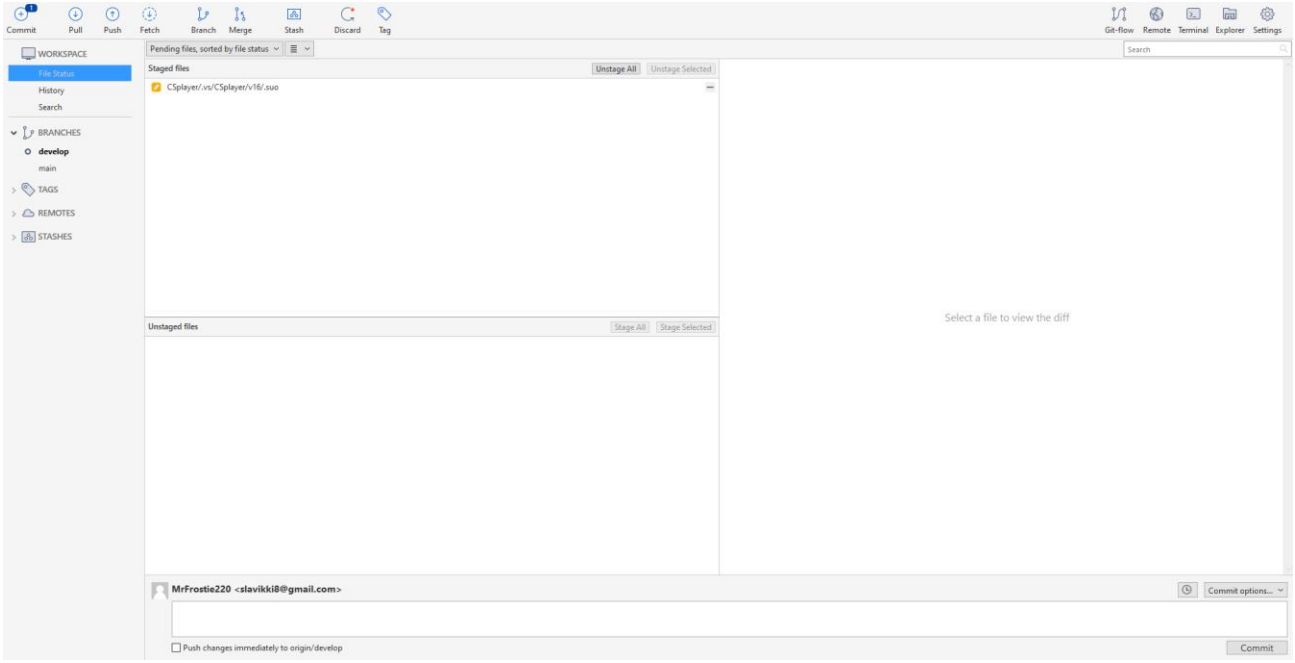


Рисунок 2.15 – Комміт внесених змін

Також даний додаток спрощує систему гілок гіта, оскільки, зазвичай, використовують 2 гілки – master та develop, виникає необхідність їх зливання одна з одною. При опису цього процесу консольними командами – зазвичай виникають труднощі. SourceTree – автоматизував цей процес рис.2.16.

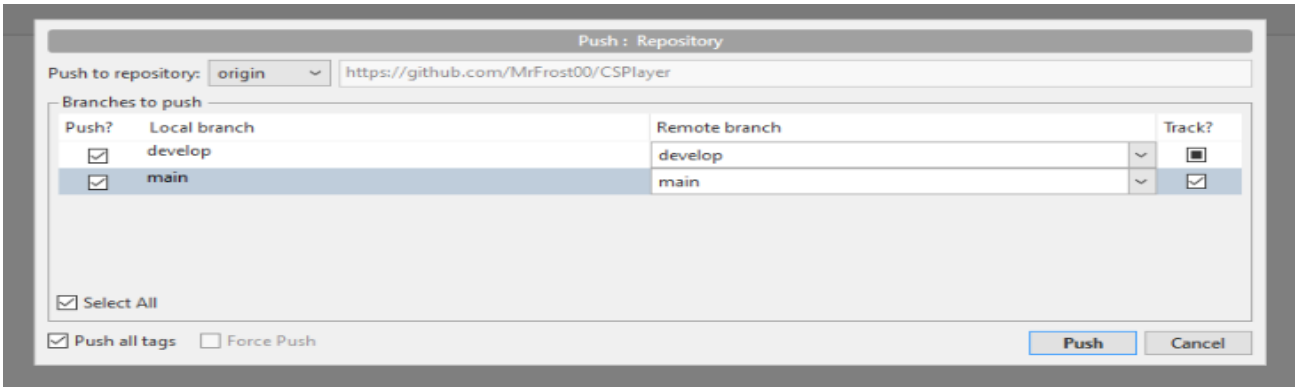


Рисунок 2.16 – Пуш та автоматичне зливання гілок

Також реалізована зручна візуалізація всього життєвого циклу процесу, яку можна відстежувати крок за кроком. В один клік можна побачити всі зміни внесені на потрібному пуші, та в один клік можна відкатити проект до потрібної версії рис.2.17.

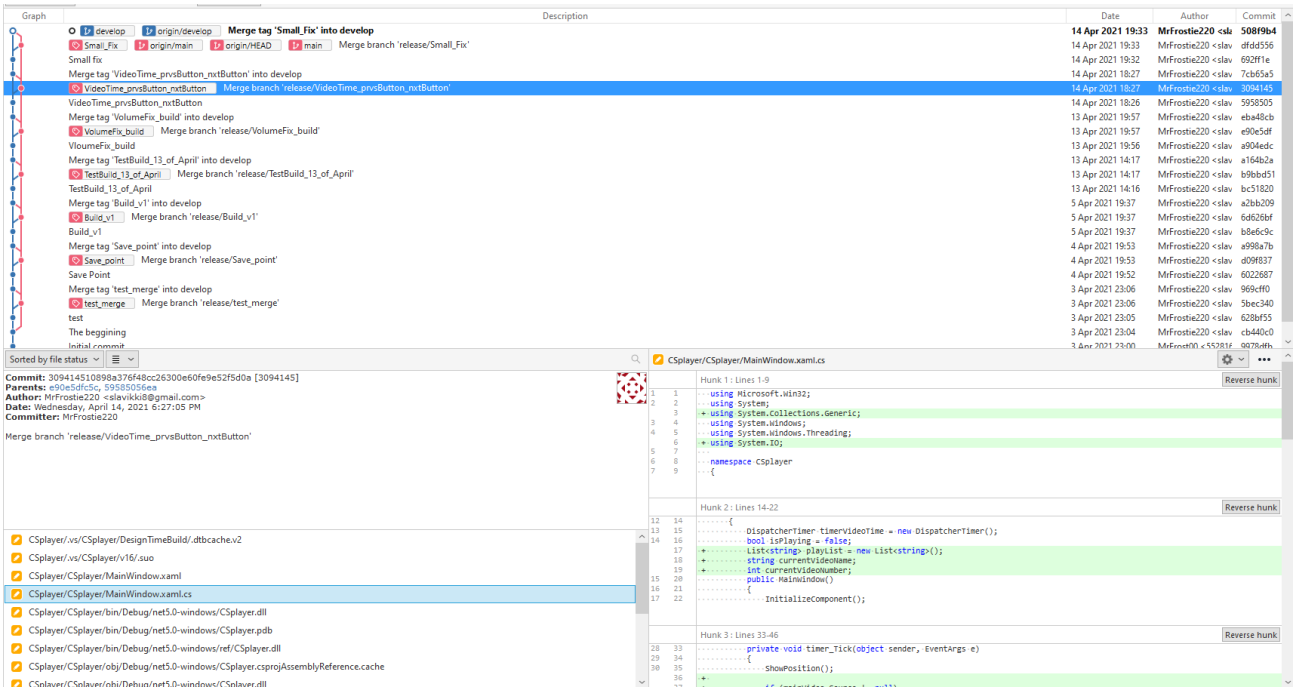


Рисунок 2.17 – Візуалізація життєвого циклу проекту

У великих командах також використровують таку технологію як Git Flow. Вона базується на схожих до звичайного Git технологій, але в ній присутні додаткові гілки: feature, release, hotfix, support. Коли потрібно розробити новий функціонал, його створюють на гілці feature, а, коли вона дороблена, закривають цю гілку та роблять реліз.

3.ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Розробка додатку

Завдання, котре повинен вирішувати застосунок – це надати можливість користувачеві, переглядати відеофайли самих поширених форматів.

Оскільки, даний додаток націлений на аудиторію з базовими знаннями користування комп’ютером - важливо розробити простий та інтуїтивно зрозумілий інтерфейс для зручного користування.

Основні можливості даного програмного забезпечення: можливість перегляду відеофайлів, можливість створення плейлістів, можливість маніпулювати відеофайлом.

Перш за все потрібно завантажити Microsoft visual studio. Завантажити її можна з офіційного сайта Microsoft рис.3.1, але важливо обрати версію Community, оскільки вона безкоштовна та націлена, здебільш, на студентів.

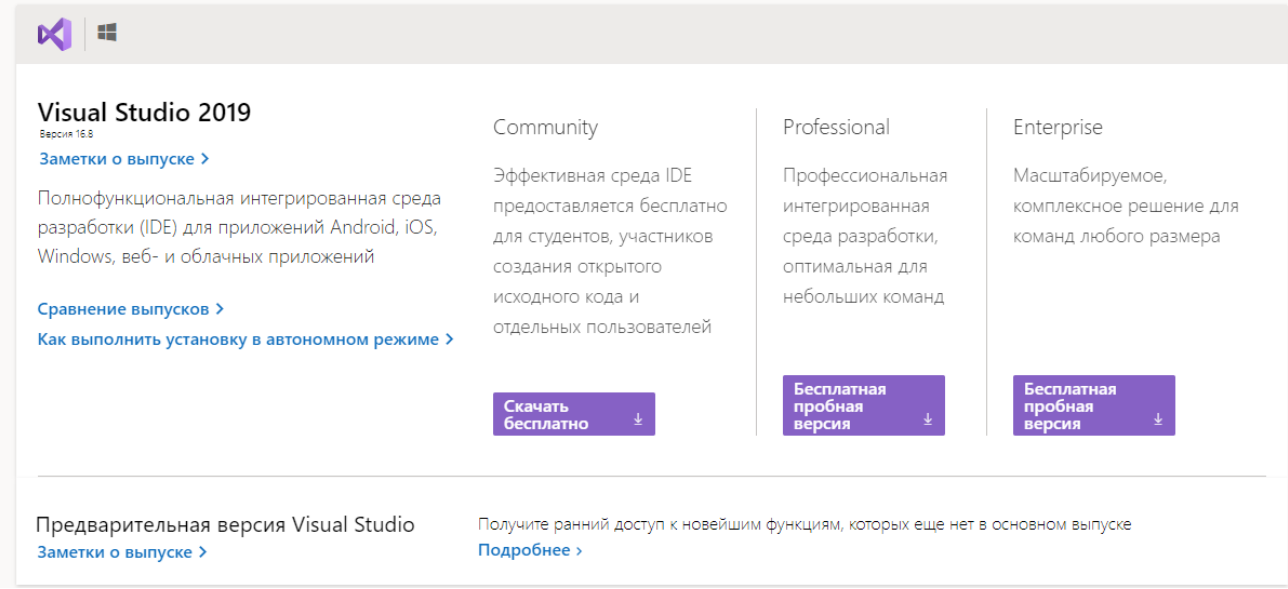


Рисунок 3.1 – Завантаження Visual Studio 2019

Надалі я буду використовувати Visual Studio 2019 Community.

Надалі потрібно створити наш проект, використовуючи шаблон WPF, який було описано раніше. В меню File (Файл) обираємо New (Створити) > Project (Проект). Перед нами відкриється діалогове вікно рис.3.2 створення проекту, в якому ми обираємо шаблон WPF Application.

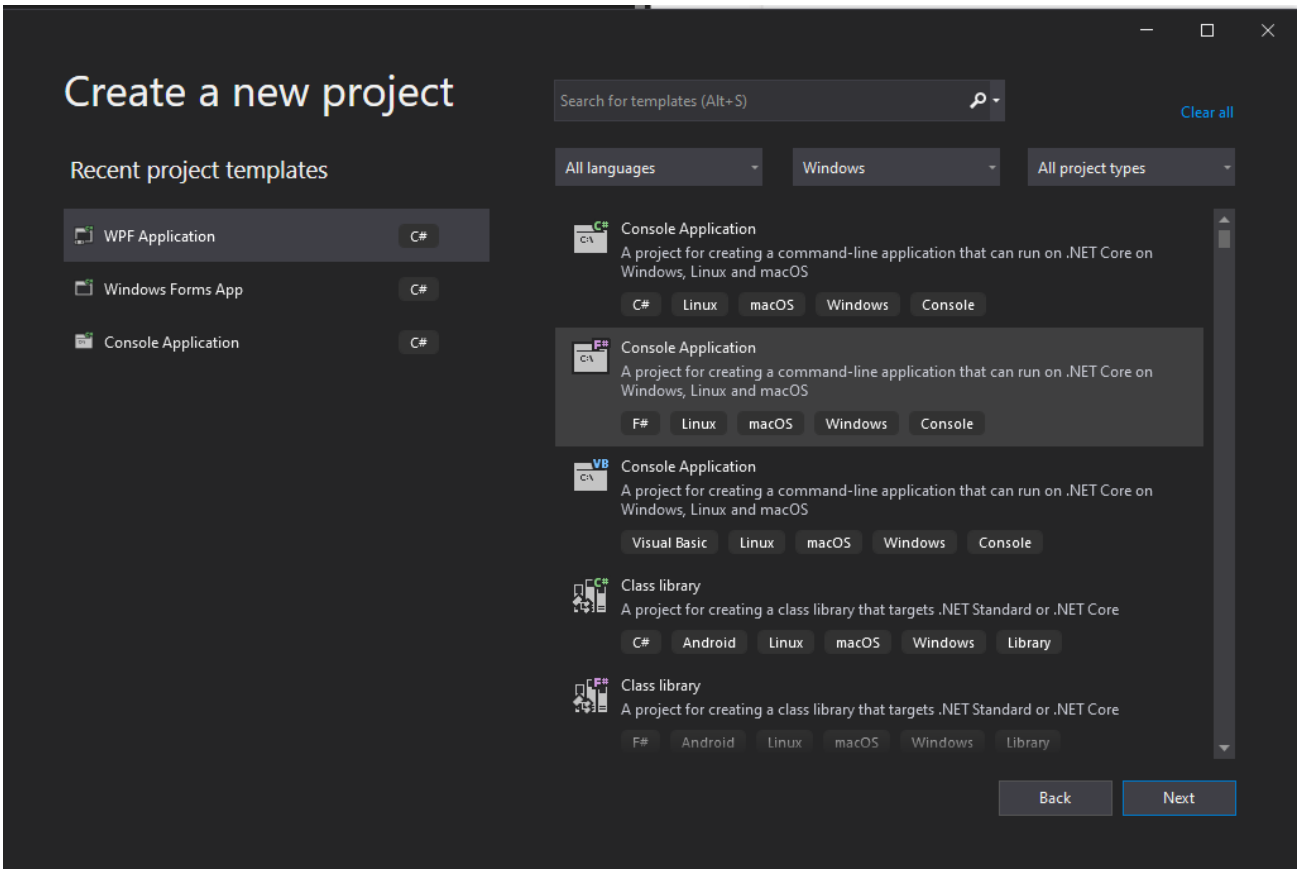


Рисунок 3.2 – Створення проекту

Після встановлення назви, студія автоматично створить на новий проект рис.3.3. В структурі проекту WPF слід виділити, що MainWindow.xaml та MainWindow.xaml.cs 2 основні файли розробки нашого проекту, коли буде запускатися зборка всього додатку – він буде створюватися з реалізованого інтерфейсу у файлі MainWindow.xaml та підкорятись функціоналу реалізованому у MainWindow.xaml.cs

Тепер нам потрібно завантажити бібліотеку Prism, яка буде допомагати нам в проектуванні нашого додатка рис.3.3.

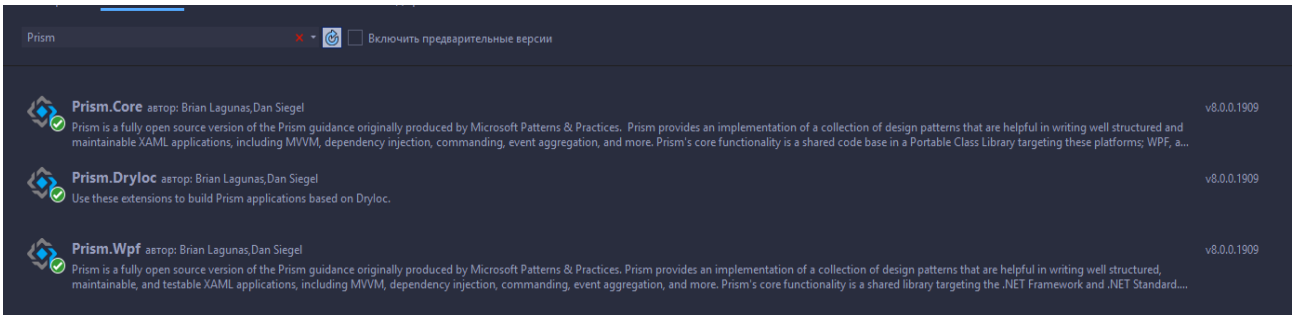


Рисунок 3.3 – Завантаження бібліотеки Prism

В основі MVVM підходу лежить зв’язок між представленням (view) та моделлю представления (viewmodel).

Prism використовує конвенцію імен для зв'язку уявлення з її моделлю - бібліотека шукає модель в папці з ім'ям Viewmodels, а відповідне уявлення в папці з ім'ям Views, таким чином нам не потрібно пов'язувати ці об'єкти в коді. Все що від нас вимагається - створювати файли в правильних папках.

Перш за все нам необхідно повідомити нашому додатку, що теперь за зв’язок відповідає Prism, для цього відкриємо наш головний файл додатку app.xaml.cs, та внаслідуємо класс PrismApplication, та перегрузимо метод CreateShell, в якому зробимо вікно MainWindow головним рис.3.4.

```
11      /// <summary>
12      /// Interaction logic for App.xaml
13      /// </summary>
        Ссылок: 3
14      public partial class App : PrismApplication
15      {
16
            Ссылок: 0
17          protected override Window CreateShell()
18          {
19              return Container.Resolve<MainWindow>();
20          }
21
```

Рисунок 3.4 Встановлення Prism відповідальним за зв’язок моделей з уявленями

Після цього ми можемо зайнятися проектування інтерфейсу нашого додатку.

Головне завдання MVVM підходу - це можливість писати гнучкі, тестовані і легко розширювані додатки, в яких немає тісно пов'язаних об'єктів. Для цього Prism надає нам систему навігації і регіонів. На xaml розмітці головної форми ми створюємо ContentControl, в якому вже буде знаходиться будь-яке з представлень. У атрибуті RegionName нашого створеного ContentControl ми вкажемо ім'я регіону. Ім'я може бути довільним, ми його використовуватимемо коли прив'язуватимемо представлення до регіону рис.3.5.

```
<Window x:Class="PrismTest.Views.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:prism="http://prismlibrary.com/"
        prism:ViewModelLocator.AutoWireViewModel="True"
        Title="{Binding Title}" Height="350" Width="525">
    <Grid>
        <ContentControl prism:RegionManager.RegionName="ContentRegion" />
    </Grid>
</Window>
```

Рисунок 3.5 Створення контейнеру-регіону

Після цього ми можемо створити тестове уявлення з моделлю, щоб побачити чи правильно ми підключили бібліотеку.

Для цього нам потрібно зробити насутпне:

1. У папці Views створимо юзер-контрол з іменем ViewA, та добавимо у нього текст на всю сторінку.
2. У папці ViewModels створимо класс ViewAViewModel, та внаслідуємося від класу BindableBase
3. У файлі App.xaml.cs зареєструємо нашу форму.
4. У коді форми ViewA, помітимо, що ця форма належить до регіону ContentRegion.

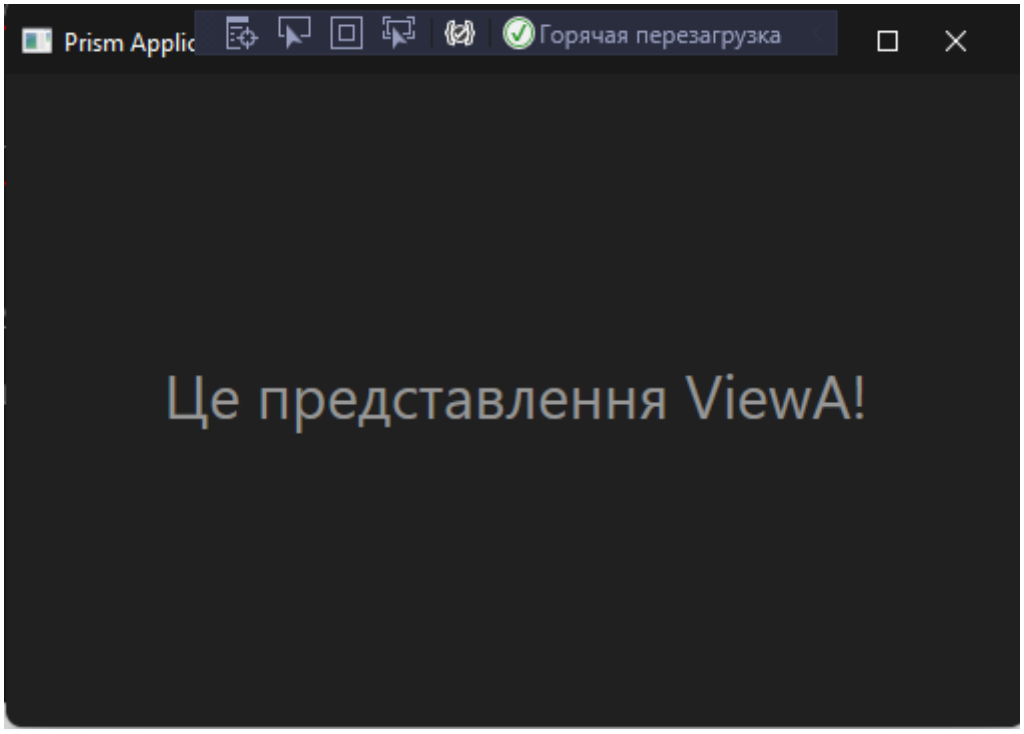


Рисунок 3.6 Результат створення тестового представлення

Ми побачили наш текст, отже Prism працює правильно, і ми можемо почати створювати реальний додаток.

Перш за все, визначемо, що буде уключати у себе головний інтерфейс, а саме:

1. Стандартне меню можливостей такі як створити файл або відкрити фото.
2. Меню з інструментами редагування.
3. Полотно на якому можно редагувати зображення.
4. Меню з шарами.
5. Меню з яке буде включати у себе інструменти для маніпуляцій над шарами або об’єктами на полотні.

Для створення списку з інструментами використуємо ListBox, в який через добавимо список з елементами рис.3.7

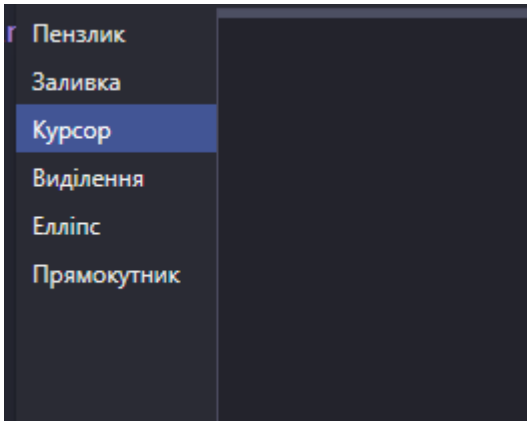


Рисунок 3.7 Меню інструментів

Для взаємодії з полотном та іншими класами, нам потрібно щоб вони знали який інструмент зараз вибраний, але ми не можемо надати їм доступ до нашого об’єекту, бо за порушує принципи MVVM. Для цього створимо класс перерахування та статичний клас CanvasToolSettings, в який ми будемо записувати вибраний інстумент, а усі класи яким потрібна буде інформація, будуть звертатися до статичного класа і отримувати дані. Такий підхід дає нам змогу не передавати усю форму інструментів, а лише те, що потрібно.

```
19  public enum Tool
20  {
21      Pen,
22      Fill,
23      Cursor,
24      Selecting,
25      Ellipse,
26      Rect
27  }
```

Рисунок 3.8 Перерахування інструментів

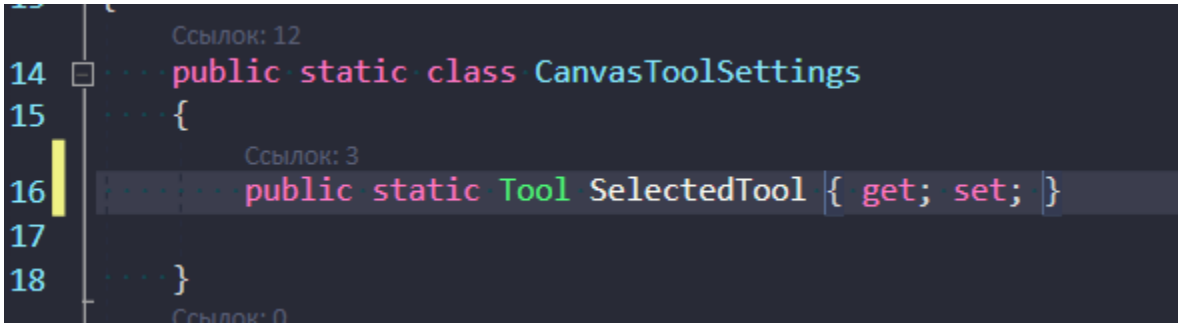


Рисунок 3.9 Статичний клас для налаштувань полотна

Тепер нам потрібно оновлювати вибраний інструмент у класі CanvasToolSetting кожен раз коли ми змінюємо інструмент, але ми не можемо напряму прив’язатися до події зміни списку в меню, бо це створить нам код безпосередньо у формі представлення, а наша задача уникати цього, тому ми повинні використовувати “команди” .

Команди представляють механізм виконання якого-небудь завдання, наприклад, копіювання тексту - коли ми натискаємо Ctrl+C, то ми копіюємо текст у буффер. В процесі копіювання виконується ряд дій, і всі разом ці дії об'єднуються в одну команду. Використання команд допомагає нам скоротити об'єм коду і використати одну і ту ж команду для декількох елементів управління в різних місцях програми. Таким чином, команди дозволяють абстрагувати набір дій від конкретних подій конкретних елементів.

Prism містить у собі класс DelegeteCommand, який реалізує стандартний інтерфейс ICommand, саме цей класс ми і будемо використовувати.

У моделі представлення нашої форми ми створимо властивість SelectedTool класу Tool, та властивість SelectionChangedCommand класу DelegateCommand, який ми прив’яжемо до події зміни елемента у ListBox безпосередньо до представлення, не використовую код форми представлення.

```
private Tool _selectedTool;
ссылка: 1
public Tool SelectedTool
{
    get { return _selectedTool; }
    set { SetProperty(storage: ref _selectedTool, value); }
}
ссылка: 1
public DelegateCommand SelectionChangedCommand { get; set; }
Ссылок: 5
```

Рисунок 3.10 Створення властивостей інструмента та команди зміни вибраного інструмента

```
<ListBox Grid.Row="1" Grid.Column="0" SelectedItem="{Binding SelectedTool, UpdateSourceTrigger=PropertyChanged}">

    <ListBoxItem>Пензлик</ListBoxItem>
    <ListBoxItem>Заливка</ListBoxItem>
    <ListBoxItem>Курсо</ListBoxItem>
    <ListBoxItem>Виділення</ListBoxItem>
    <ListBoxItem>Еллипс</ListBoxItem>
    <ListBoxItem>Прямокутник</ListBoxItem>

    <b:Interaction.Triggers>
        <b:EventTrigger EventName="SelectionChanged">
            <prism:InvokeCommandAction Command="{Binding SelectionChangedCommand}"></prism:InvokeCommandAction>
        </b:EventTrigger>
    </b:Interaction.Triggers>
</ListBox>
```

Рисунок 3.11 Прив’зка команди до події зміни

Щоб прив’язатися до події зміни, ми використали стандартну бібліотеку Interactions від Microsoft.

Тепер, коли ми усе зробили, в нашому класі CanvasToolSetting буде зберігатися акутальний інструмент який ми вибрали на нашій панелі. Це дає змогу отримувати інформацію про нього з будь-якого місця у додатку, при цьому не посилаючись на основну форму меню.

Зараз ми можемо почати розробку основного вікна – полотна. Виділимо основні пункти, які потрібні нам при проектуванні цього елемента:

- Полотно.
- Шари.
- GUI – допоміжний інтерфейс для користувача.

Отже, перш за все, почнемо розробку полотна, для цього ми будемо використовувати стандартний елемент – Image. Хоча в WPF вже є готове рішення під малювання – Canvas, ми не будемо його використовувати, через те, що він занадто повільний та займає багато ресурсів ПК.

Для початку створимо окремий User-control, а потім прив’яжемо його до нашого регіону, як ми зробили це з меню списка інструментів. Після створення вікна, добавио до нього елемент Image, і задамо йому розміри 500 пікселів на 500 пікселів, але пізніше дамо користувачеві самому обирати розмір полотна.

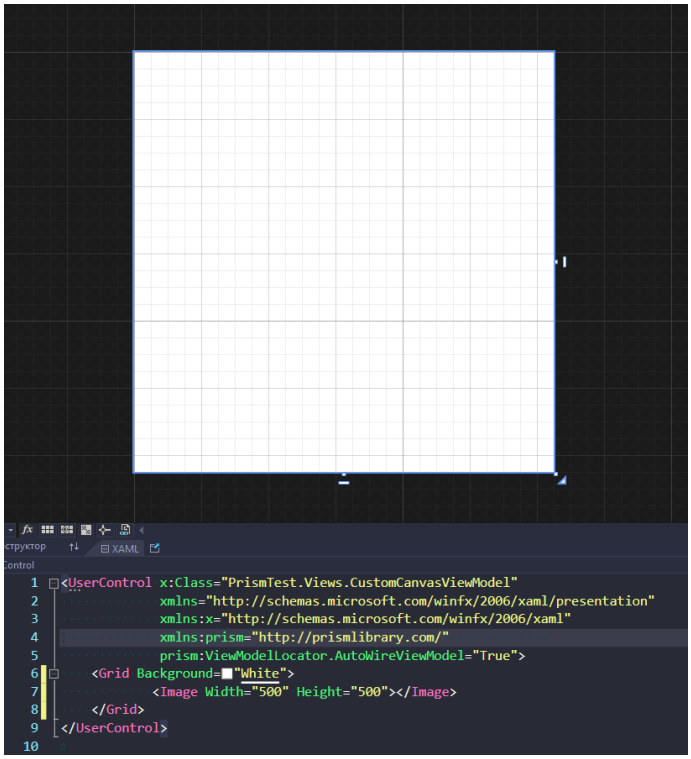


Рисунок 3.12 Створення головного полотна

Тепер нам потрібно обрати спосіб, яким ми будемо відображати елементи на нашому полотні. Варіанти який надає нам стандартний WPF:

- Класс Shape
- Рендер за допомогою класу DrawingVisual
- Малювання безпосередньо на bitmap

Класс Shape містить у собі усі стандартні елементи, такі як малювання лінії, квадрату, еліпсу або зображення. Також декілька корисних функцій, такі як малювання з маскою, або виділення пікселів у обрані йплощі. Але, за такий багатий фунцкіонал, цей клас втрачає свою продуктивність, коли нам потрібно відмальовувати багато об’єетів, також використовуючи цей класс, нам потрібно буде перемальовувати усі об’єкти, при зміні розміру вікна додатку, або при додаванні нового об’єкту до нашого полотна. В нашому випадку користувач може мати до тисячі об’єктів на полотні, тому втрата продуктивності додатка буде тим більше.

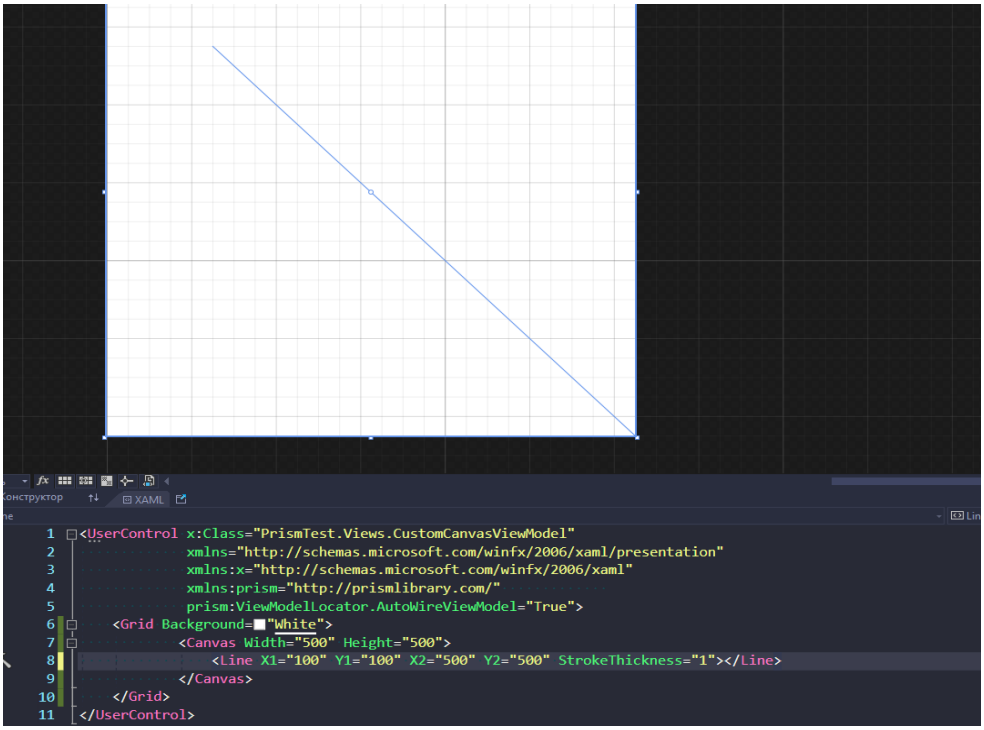


Рисунок 3.13 Створення лінії класу Shape

Клас DrawingVisual буде працювати швидше, оскільки нам не потрібно створювати багатозатратні об’єкти, як у випадку з класом Shape. В ньому ми можем тільки вказувати координати, куди та що ми хочем намалювати, а він вже видасть нам результат. Але цей клас має таку ж недостачу, як і клас Shape, а саме те, що нам потрібно буде проходитись по усьому списку наших елементів, та перемальовувати кожен із них, бо після кожної взаємодії з полотном, клас DrawingVisual стирає всі об’єекти з полотна.

Малювання безпосередньо на bitmap відбувається за допомогою класу WriteableBitmap. За допомогою його ми можемо записувати дані у пам’ять нашого bitmap, але для цього нам потрібно використовувати ключове слово unsafe, яке помічає код як небезпечний.

Взагалі, зазвичай C# не дозволяє звертатися безпосередньо до пам’яті, незважаючи на те, що у його спорідненої мови C++ це дозволено. А зроблено через те, що у C#, за видалення об’єктів з пам’яті відповідаю службовий клас збирача сміття, або GarbageCollector. Він видаляє об’єкт, коли до нього не веде жодного посилання, тобто до такого об’єкту ми більше не можемо ніяк звернутися. Хоча такий і вповільнує виконування програми, це надає змогу розробникам білшьше зосередитися на написанні коду, та абстрагуватися від витоку пам’яті. Але якщо ми не можемо вирішити нашу проблему без втручання безпосередньо до пам’яті, або ми хочемо пришвидшити нашу програму, у мові C# передбачене ключове слово unsafe. Перед його використанням потрібно в проекті дозволити небезпечний код. В цьому блоці ми можемо звертатися безпосередьно до пам’яті, а нашому випадку саме пам'яті об’єкту. Оскількі масив пікселів лежить безперервно, ми можемо пройтись по ним за одну ітерацію, та змінити їх як нам треба. Але треба слідкувати, щоб ми не вийшли за межі масиву, та не переписали не нашу пам’ять, саме через це, ключове слово називається unsafe – тобто небезпечний.

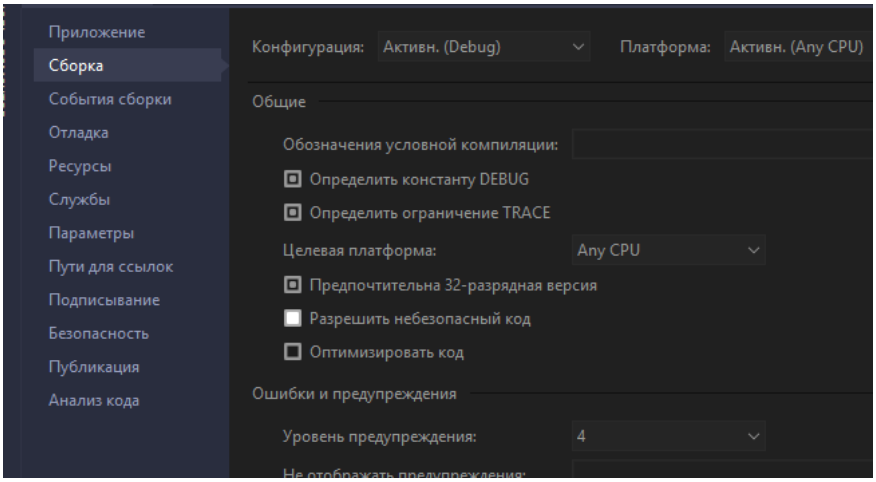


Рисунок 3.14 Дозвіл небезпечного коду в налаштуваннях проекту

```
Ссылок: 0
public CustomCanvasViewModelViewModel()
{
    var writeableBitmap = new WriteableBitmap(pixelWidth: 500, pixelHeight: 500, dpiX: 96, dpiY: 96, PixelFormats.Pbgra32, palette: null);
    unsafe
    {
        // Get a pointer to the back buffer.
        int pBackBuffer = (int)writeableBitmap.BackBuffer;

        // Find the address of the pixel to draw.
        pBackBuffer += 500 * writeableBitmap.BackBufferStride;
        pBackBuffer += 500 * 4;//??

        // Compute the pixel's color.
        int color_data = 255 << 16; // R
        color_data |= 128 << 8;   // G
        color_data |= 255 << 0;   // B

        // Assign the color data to the pixel.
        *((int*)pBackBuffer) = color_data;//??
    }
}
```

Рисунок 3.15 Приклад використання небезпечного коду

В нашому випадку ми будемо застосовувати гібридний варіант, клас DrawingVisual який відповідатиме за малювання, та клас WriteableBitmap, який відповідає за збереження зображення. Щоб перетворити DrawingVisual у Bitmap ми будемо використовувати клас RenderTargetBitmap, який буде брати на себе рендер того, що ми відмалювали на контексті.

Кроки для реалізації такого метода наступні:

1. Створити об’єкт класу WriteableBitmap, який буде виступати як джерело для полотна.

2. Створити об’єкт класу DrawingVisual.

3. З нього отримати об’єкт класу DrawingContext, який має методи малювання основних примітивів та зображень.

4. Намалювати потрібні нам елементи

5. За домогою класу RenderTargetBitmap перетворити DrawingVisual у Bitmap.

6. Скопіювати пікселі RenderTargetBitmap до нашого основного об’єекту WriteableBitmap.

Хоча такий набір дій і виглядає як “важкий”, але насправді це дуже популярний метод відображення динамічних об’єктів у WPF, оскільки стандартні методи цієї платформи не дуже під це заточені. Якщо ми уважніше придивимося на цих кроків, можна зрозуміти, що такий підхід чимось нагадує подвійну буферизацію, яку найчастіше використовують у рендері графіки. Нам також не потрібно відмальовувати усі об’єкти, а лише ті, які додав користувач.

Такий підхід також дозволяє нам вибирати зону нашого полотна, яку ми хочемо оновити, бо якщо ми маємо велику площу, нам не потрібно рендерити усе полотно, якщо користувач додав маленький кружечок.

```
var temp = BitmapFactory.New(WIDTH, HEIGHT);
using (temp.GetBitmapContext())
{
    _rtb.Clear();
    _currentElement.Execute((int)lastPoint.Value.X, (int)lastPoint.Value.Y, (int)newPoint.X, (int)newPoint.Y);
    _currentElement.Render(_drawingVisual, clonRender: false);
    _rtb.Render(_drawingVisual);
    _rtb.CopyPixels(new Int32Rect(0, 0, _rtb.PixelWidth, _rtb.PixelHeight),
        temp.BackBuffer,
        bufferSize: temp.BackBufferStride * temp.PixelHeight, temp.BackBufferStride);

    //temp.AddDirtyRect(new Int32Rect(0, 0, (int)500, (int)500));
}

//WBitmapLayer.Clear(Colors.White);
if (!shouldImmediatlyDraw)
{
    WBitmapLayer.AddDirtyRect(new Int32Rect(0, 0, (int)WIDTH, (int)HEIGHT));
    WBitmapLayer.Blit(new Point(0, 0), _backBuffer, new Rect(0, 0, WIDTH, HEIGHT), Colors.White, WriteableBitmapExtensions.BlendMode.None);
}

WBitmapLayer.Blit(destRect: new Rect(0, 0, WIDTH, HEIGHT), temp, new Rect(0, 0, WIDTH, HEIGHT), WriteableBitmapExtensions.BlendMode.Alpha);
```

Рисунок 3.16 Приклад використання DrawingVisual разом з WriteableBitmap

Спочатку ми створюємо порожній об’єкт класу WriteableBitmap за допомогою статичного метода New класу BitmapFactory, він поверне нам стандартний об’єкт WriteableBitmap з висотою та шириною, які ми передали у конструктор, стандартним DPI, та форматом пікслелів Pbgra32. Далі ми відкриваємо контекст цього об’єкту, і

починаємо проводити манипуляції вже з ним.

Щоб не завантажувати систему стандартними класами фігур, які встроєні у WPF, ми створимо свій клас, в якому будемо зберігати тільки потрібні нам параметри, такі як координати, колір, тощо. Для початку створимо інтерфейс ILightElement, який будуть наслідувати усі наші власні інструменти рис.3.17.

Рисунок 3.17 Набір властивостей та методів які будуть мати інструменти

Це базовий набір властивостей, для наших класів, ми зможемо доповнювати його, якщо нам знадобиться. Властивості відповідають за:

- LeftTop – координата верхнього лівого кута
- RightBottom – координата нижнього правого кута
- Size – розмір елемента (по іксу та ігрику)
- Rotation – поворот елемента
- Opactiy – прозорість
- Brush – колір заливки
- Pen – колір обведення
- Thickness – ширина строки

- Move – метод для зміни позиції елемента на полотні
- Render – метод для відмальовування елемента на полотні
- Execute – метод, котрий визивається при взаємодії в елементом

Кожен елемент буде реалізовувати метод рендеру по-своєму, приклад реалізації інструменту прямокутника рис.3.18.

```
Ссылок: 7
public void Render(DrawingVisual drawingVisual, bool foolRender)
{
    Pen.EndLineCap = PenLineCap.Round;
    Pen.StartLineCap = PenLineCap.Round;
    Pen.LineJoin = PenLineJoin.Round;

    DrawingContext drawingContext = drawingVisual.RenderOpen();

    drawingContext.DrawRectangle(Brush, pen: null, new Rect(LeftTop, RightBottom));
    drawingContext.Close();
}
```

Рисунок 3.18 Метод малювання прямокутника

Тут ми малюємо прямокутник на об’єкті DrawingVisual, з координатами які ми передали у метод Execute. Цей метод просто записує нові координати елемента.

Алгоритм виглядає так:

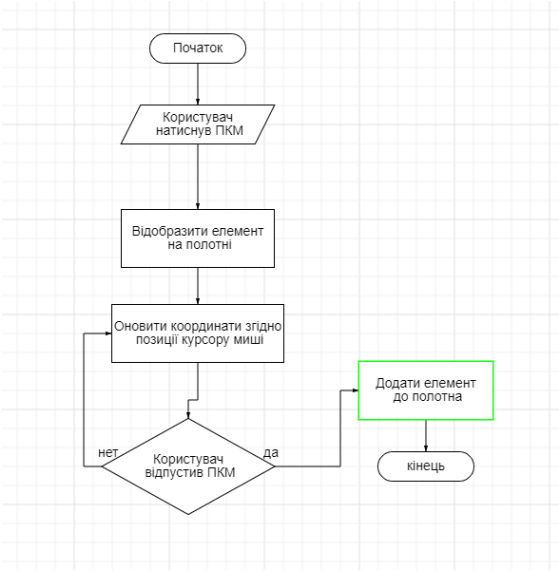


Рисунок 3.19 Блок-схема малювання елемента

Таким чином ми зробили універсальний метод для взаємодії з полотном, який ми можемо використовувати для будь-якого типу, наприклад, еліпсу. Єдиною різницею буде метод рендеру, в якому ми замість DrawRectanlge будемо визивати метод DrawEllipse.

Для полегшення процесу взаємодії між об’єктами класів, створимо класс LightLayer, який матиме властивість шару, і буде відповідальний за рендер елементів, які знаходяться на ньому. Додамо до нього базові властивості та методи, а саме:

- Ширину та висоту
- Масив елементів типу ILightElement
- Об’єкт типу WriteableBitmap
- Метод початку малюванню
- Метод процесу малювання
- Метод завершення малювання

Тепер ми можемо додати вікно, яке буде відповідальним за контроль над шарами: вибір шару, створення, видалення, відобразити або сховати шар. Для цього, як і раніше, свторимо User-control представлення, та представлення-модель. А на форму добавио елемент GroupBox, який буде містити ListBox з нашими шарами.

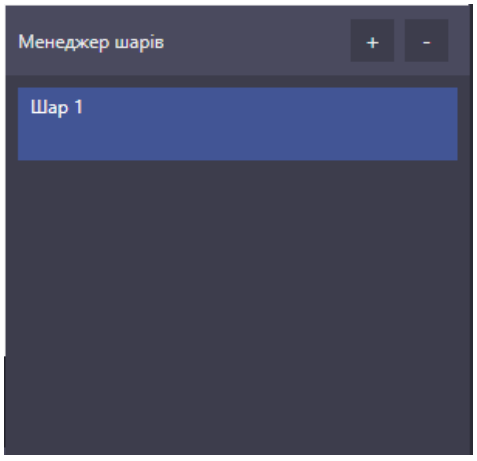


Рисунок 3.20 Представлення менедржеру Шарів

Але тепер в нас виникає проблема – представлення-модель полотна, та представлення-модель шарів – різні класи, які не мають посилань один одного, і мати не можуть, бо це порушує принципи MVVM підходу. Наша задача полягає в наступному: полотно не повинно мати доступу до менеджеру шарів, натомість вони мають взаємодіяти при певних обставинах. Наприклад: ми вибрали новий шар, на якому тепер будемо малювати, і нам потрібно повідомити клас полотна, що тепер в нас вибраний інший шар.

Для цього ми застосуємо патерн проектування “Observer”. Суть цього підходу в наступному, в нас є два типи об’єктів:

- Publisher
- Subscriber

Publisher – це об’єкт, який буде повідомляти усіх своїх підписників, що трапилася якась подія. При цьому йому не важливо мати посилання на усіх, хто підписан, тому що він лише оповіщає, що сталася подія, наприклад, створення шару.

Subscriber – підписник, який підписується на якусь подію, і, коли вона виконується, запускає вказаний алгоритм дій.

Перевага цього метода в тому, що ми можемо підписуватись на події, та сповіщати про неї з будь-якого місця в коді, це дає змогу зробити гнучку архітектуру, яку легко використовувати.

Бібліотека Prism реалізує цей патерн класом PubSubEvent, який ми можемо получити через об’єкт класу IEventAggregator.

Для початку створимо свій клас-подію, з назвою SelectedLayerChanged, який буде передавати аргумент типу ILightLayer – вибраний шар.

```
Ссылок: 2
public class SelectedLayerChangedEvent : PubSubEvent<LightLayer>
{
}
```

Рисунок 3.21 Клас-подія зміни шару

Зараз, кожен раз коли ми захочемо повідомити про зміну шару, ми повинні визвати цю подію:

```
ссылка: 1
private void SelectedLayerChangedHandler()
{
    if (SelectedLayer == null)
        return;
    _eventAggregator.GetEvent<SelectedLayerChangedEvent>().Publish(SelectedLayer);
}
ссылка: 1
```

Рисунок 3.22 Повідомлення про зміну шару

А усі, хто хоче отримувати цю подію, повинен підписатися на неї, та додати метод, який буде виконуватися рис.3.23:

```
_eventAggregator.GetEvent<SelectedLayerChangedEvent>().Subscribe(SelectedLayerChanged, ThreadOption.UIThread,
    keepSubscriberReferenceAlive: true, filter: layer => layer.CanvasId == _canvasId);
```

Рисунок 3.23 Підпис на подію зміни шару

При підписанні на цю подію в нашому класі полотна, ми повинні в параметрах зазначити, що ця подія повинна відбуватися у потоці інтерфейсу користувача, тому що у цьому класі ми працюємо безпосередньо з полотном.

Тепер ми можемо отримати подію зміни шару з будь-якої точки коду. Це дуже зручно – якщо нам потрібно буде добавити якийсь новий функціонал, нам не потрібно знати про те, хто виконує цю зміну, та за яких обставин. Усе, що нам потрібно – безспосередньо подія.

Щоб зв’язати вибраний шар у нашому представленні та представленні-моделі нам потрідно використати потужний інструмент WPF – біндінг. Це дозволяє нам

зв’язувати властивості елементів у представленні з її моделлю. Це реалізується за допомогою слова Binding на формі представлення.

В нашому випадкому потрібно властивість SelectedItem списку, прив’язати до об’єкту SelectedLayer в класі менеджеру шарів рис.3.24.

```
</GroupBox.HeaderTemplate>
<ListBox HorizontalContentAlignment="Stretch" MinHeight="300" SelectedIndex="0" ItemsSource="{Binding Layers}"  SelectedItem="{Binding SelectedLayer}">
    <b:Interaction.Triggers>
        <b:EventTrigger EventName="SelectionChanged">
            <prism:InvokeCommandAction Command="{Binding SelectedLayerChangedCommand}"></prism:InvokeCommandAction>
        </b:EventTrigger>
    </b:Interaction.Triggers>
    <ListBox.ItemContainerStyle>
        <Style TargetType="ListBoxItem" BasedOn="{StaticResource {x:Type ListBoxItem}}">
            <Setter Property="HorizontalContentAlignment" Value="Stretch"></Setter>
        </Style>
    </ListBox.ItemContainerStyle>
    <ListBox.ItemTemplate>
```

Рисунок 3.24 Біндінг вибраного елементу

Тепер коли ми будемо змінювати елемент в нашому списку, він буде також змінюватися у представленні-моделі.

А зараз, після реалізації деяких інструментів та менеджера шарів, ми можемо спробувати щось додати до нашого полотна рис.3.25:

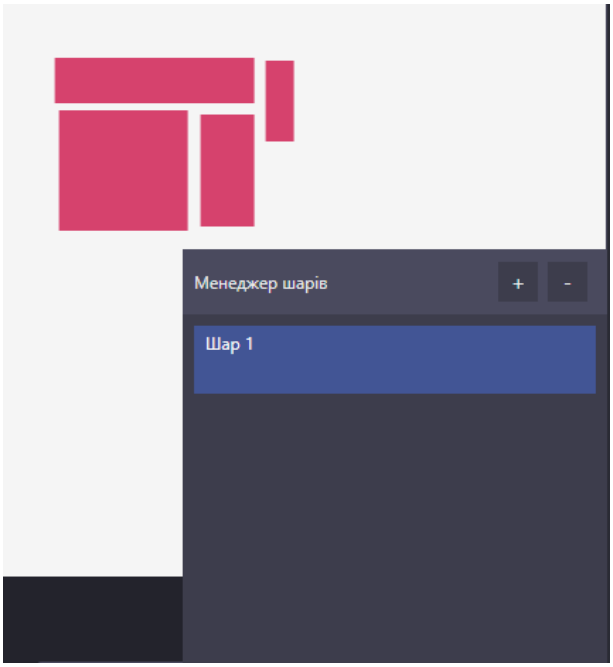


Рисунок 3.26 Приклад малювання на полотні

Тепер реалізуємо можливість пересувати об’єкти інструментом “курсор”. Для цього нам потрібно відстежувати позицію курсора, коли користувач натиснув ПКМ. Для цієї задачі використаємо таку ж саму структуру з класом Interactions, як і раніше рис.3.27. Але тепер подія, до якої ми прив’яжемося – клік ПКМ, а параметром будемо передавати наше полотно, відносно якого будемо отримувати координати курсору миші.

```
<WrapPanel Grid.Column="1" Background="AliceBlue" Width="{Binding WIDTH}" Height="{Binding WIDTH}">
    <i:Interaction.Triggers>
        <i:EventTrigger EventName="MouseDown">
            <prism:InvokeCommandAction Command="{Binding MouseDownCommand}" CommandParameter="{Binding ElementName=_mainCanvas}"><
        </i:EventTrigger>
```

Рисунок 3.27 Прив’язка до події кліку ПКМ

```
ссылка: 1
private async void MouseDown(object e)
{
    //Layers[SelectedLayer].DiscradSelected();
    if (_lastPoint == null)
    {
        _lastPoint = Mouse.GetPosition(e as IInputElement);
        SelectedLayer.Begin(_lastPoint.Value, CanvasToolSettings.SelectedTool);

    }
}
```

Рисунок 3.28 Отримання координат курсору відносно полотна

Після цього ми повинні пройтися по всьому масиву елементів поточного шару, та перевірити чи попадають координати у кордони цього елементу, для цього ми будем порівнювати позицію курсору з властивостями LeftTop та BottomRight, поточного елемента перебору. Якщо ми знаходимо елемент який нам підходить, ми виходимо з циклу, а у властивість SelectedElement класу полотна записуємо цей елемент рис.3.29.

```
foreach (ILightElement ce in _canvasElements.AsEnumerable().Reverse())
{
    if ((orig.X > ce.LeftTop.X && orig.Y > ce.LeftTop.Y) &&
        (orig.X < ce.RightBottom.X && orig.Y < ce.RightBottom.Y) ||

        (orig.X < ce.LeftTop.X && orig.Y < ce.LeftTop.Y) &&
        (orig.X > ce.RightBottom.X && orig.Y > ce.RightBottom.Y))
    {
        _selectedElement = ce;
        CanvasToolSettings.Element = _selectedElement;
        break;
    }

    _selectedElement = null;
}
```

Рисунок 3.29 Пошук елемента на який натиснув користувач

Тепер нам потрібно відобразити цей об’єкт як виділений, для цього створимо клас LightCanvasGUI, який буде малювати допоміжний інтерфейс на окремому полотні. Після цього, кожен раз коли нам потрібно виділити елемент, ми будемо визивати цей клас та передавати йому в аргументах посилання на елемент, з я кого він будет отримувати його кордони, та за ними малювати область виділення рис.3.30.

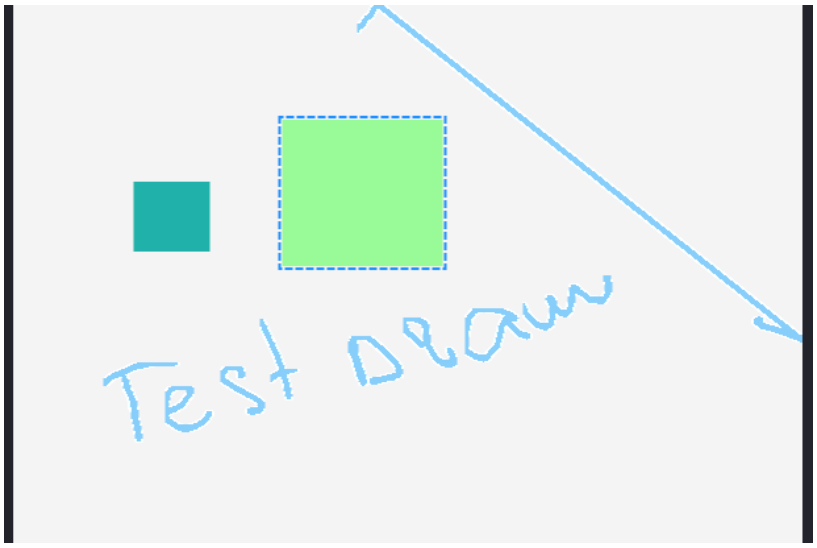


Рисунок 3.30 Приклад виділення елемента (зелений прямокутник)

Коли користувач буде рухати мишу з зажатою ПКМ на виділеному об’єкті, ми будемо змінювати координати цього елементу, і повідомляти шару, щоб він

перемалював вибраний елемент.

Щоб ми могли швидко відмальовувати елемент на новому місці, при цьому не перетираючи старі елементи, ми будемо зберігати весь шар, окрім вибраного елементу у окремий WriteableBitmap який ми назвемо заднім буфером. А при потребі перерендерети поточний елемент на новому місці, ми будем спочатку оновляти основне полотно використовуючи збережену інформацію із заднього буфера, а потім додавати на нього наш елемент.

Для оптимізації цього процесу, ми будемо оновляти не ціле полотно, а лише ту зону, яку займає поточний об’єкт, використочуючи його властивості LeftTop та RightBottom. Щоб реалізувати такий алгоритм, нам потрібно використати вбудовану функцію класу WriteableBitmap – Blit.

Цей метод копіює пікселі з одного Bitmap у другий, при цьому це дуже швидка операція. У цей метод ми будемо передавати параметри прямокутника, зони якого нам потрібно оновити рис.3.31. Завдяки цьому, навіть якщо ми маємо полотно з тисячею елементами, ми будемо витрачати мінімальну кількість ресурсів, адже все полотно буде збережено у задній буфер.

```
foreach (var el in _canvasElements)
{
    if (el != _selectedElement)
    {
        WBitmapLayer.Blit(new Point(el.LeftTop.X, el.LeftTop.Y), GetRenderedElement(el), new Rect(el.LeftTop.X, el.LeftTop.Y, el.RightBottom.X, el.RightBottom.Y), Colors.White, WriteableBitmapEx
    }
}
using (_backBuffer.GetBitmapContext())
{
    WBitmapLayer.CopyPixels(new Int32Rect(0, 0, WIDTH, HEIGHT),
        _backBuffer.BackBuffer, _backBuffer.BackBufferStride * _backBuffer.PixelHeight, _backBuffer.BackBufferStride);
    _backBuffer.AddDirtyRect(new Int32Rect(0, 0, (int)WIDTH, (int)HEIGHT));
}

using (_backBuffer.GetBitmapContext())
{
    WBitmapLayer.Blit(new Point(_selectedElement.LeftTop.X, _selectedElement.LeftTop.Y), GetRenderedElement(_selectedElement), new Rect(_selectedElement.LeftTop.X, _selectedElement.LeftTop.Y, _selectec
}
```

Рисунок 3.31 Збереження елементів у задній буфер, окрім виділеного елемента

Тепер нам потрібно змінювати його позицію, використовуючи метод Move, в якому ми змінюємо LeftTop та RightBottom елементу.

Аналогічно з позицією зробимо і поворот елементу, але тепер будемо визивати метод Rotate рис.3.32.

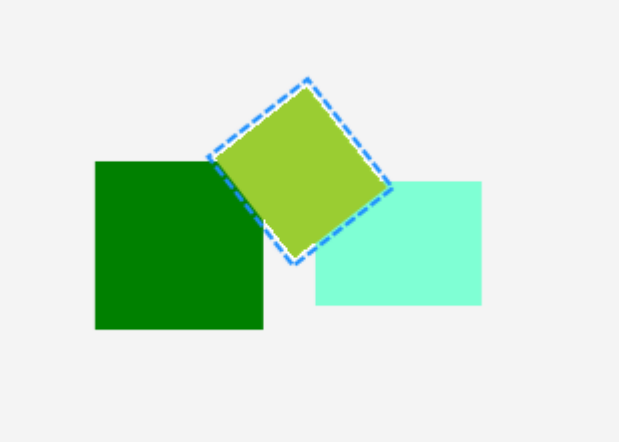


Рисунок 3.32 Приклад повороту элементу

Для зручності, додамо справа до нашого додатка меню, де користувач зомже вибирати потрібні властивості для редагування рис.3.33.

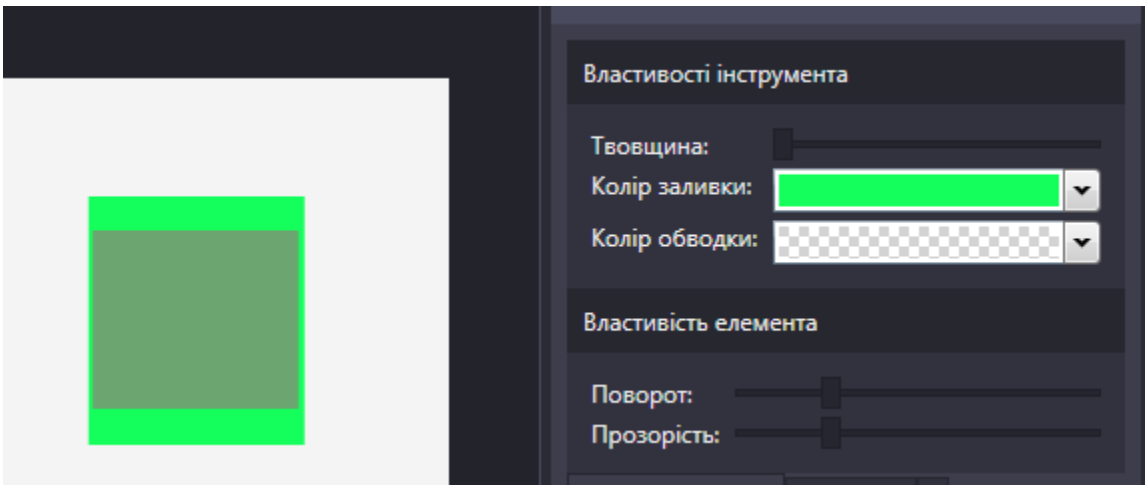


Рисунок 3.33 Меню властивостей

А також стандартне меню, для редагування таких показників як яскравість, контраст та гамма через які мибудемо змінювати властивості вибраного шару рис.3.34.

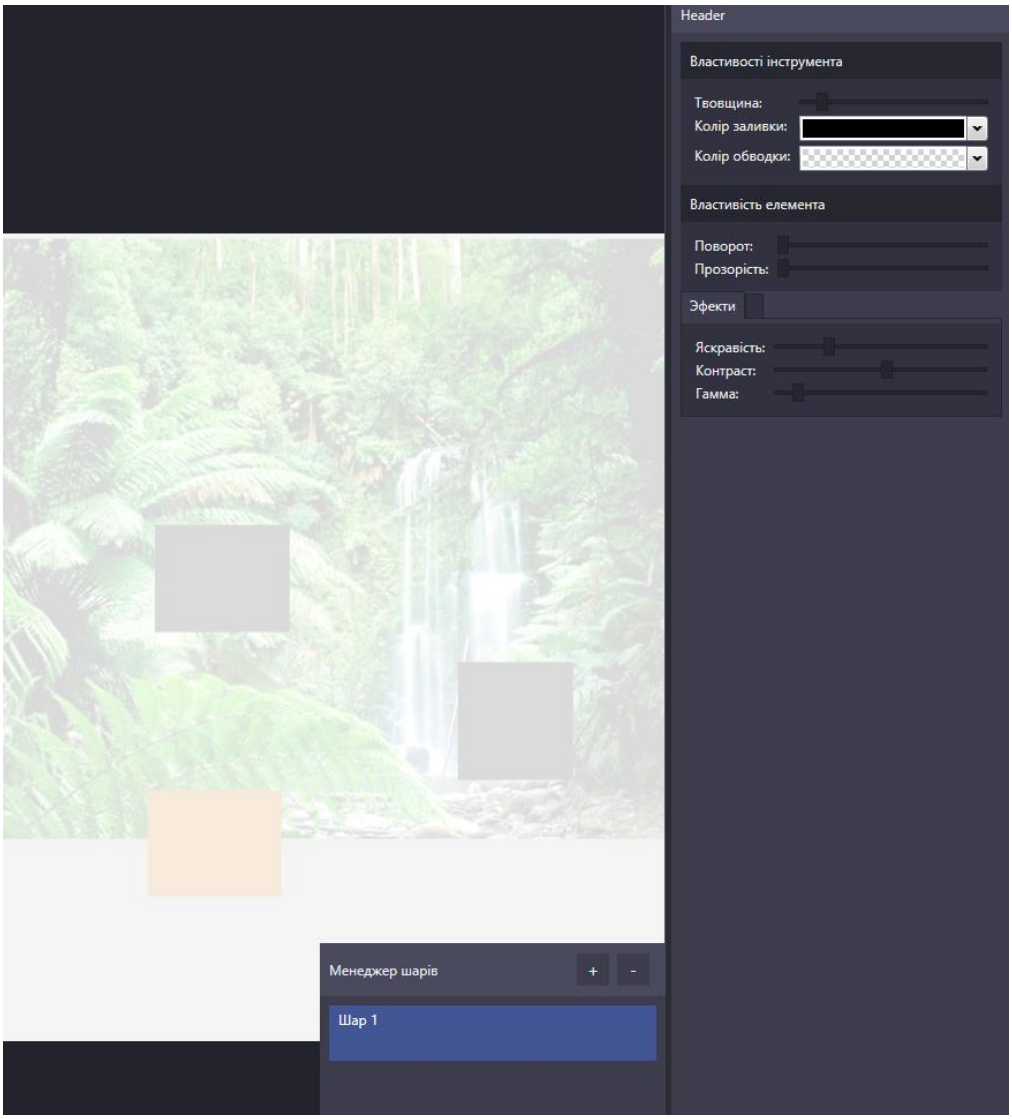


Рисунок 3.34 Редагування параметрів шару

Завдяки підходу з WriteableBitmap ми можемо змінювати ці параметри, у реальному часі та дуже ефективно. Користувач одразу може бачити результат, після зміни одного з повзунків.

4. Продуктивність додатка

4.1 Використання ресурсів ПК

При роботі з даним додатком, якщо ми під час дебага подивимося до показників памя’ті та навантаження процесору, ми побачему, що наша програма використовую не більше 150-200, а в піковому навантаженні 300мб оперативної пам’яті, що є дуже добрим показником.

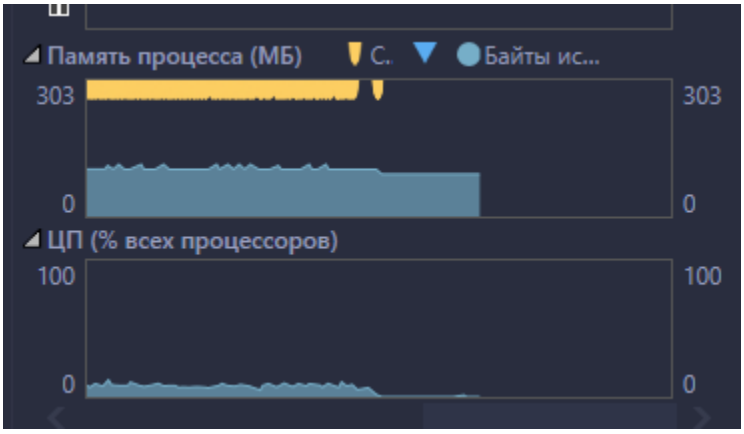


Рисунок 4.1 Монітор ресурсів додатка

У порівнянні з такими аналогами як Adobe Photoshop або Paint Tool SAI, наш додаток у 2-3 рази менше ресурсів ПК користувача, що дає змогу паралельно працювати не тільки в цьому додатку, а й в браузері, грі, тощо.

5. Інструкція користувача

5.1 Характеристики та інтструкція

Для функціонування програми необхідно наступне технічне забезпечення з наступними мінімальними характеристиками:

1. Процесор - Intel Pentium V / 2.2 Ghz або вище.
2. Обсяг оперативної пам'яті - 1024 Mb RAM.
3. Місце на жорсткому диску - 1024 Mb HDD.

Після запуску додатку, для початку редагування файла потрібно відкрити зображення натиснувши на кнопку відкриття зображення та вибрати файл.

Для коректної роботи додатку, рекомендується використовувати Windows 7 або вище.

Також, необхідні наступні умови для запуску Windows 7,8,10 на комп'ютері:

1. 32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2.2 GHz (ГГц) або вище.
2. 1 гігабайт (ГБ) (для 32-розрядної системи) або 2 ГБ (для 64-розрядної системи) оперативної пам'яті (ОЗУ).
3. 16 гігабайт (ГБ) (для 32-розрядної системи) або 20 ГБ (для 64-розрядної системи) простору на жорсткому диску.
4. Графічний пристрій DirectX 10 з драйвером WDDM версії 1.0 або вище.

ВИСНОВКИ

У результаті виконання даної дипломної роботи було розроблено десктопний додаток. У роботі міститься основні принципи розробки додатку.

1. Під час роботи над дипломним проектом було проаналізовано існуючі аналоги, та програми, схожі на розроблений продукт, що знаходяться у вільному

доступі. Було розглянуто проблеми, які виникають під час розробки додатків, які підтримують режим віртуальної реальності.

2. Створено концепцію проекту.

3. Розроблений швидкий та легкий додаток, для редагування зображень

4. Дизайн додатку був розроблений з урахуванням всіх нюансів, а саме: основу дизайну складає XAML, за зв’язок моделей відповідальна бібліотека Prism.

Методи розробки, що були використані у додатку дозволяють використовувати додаток, людям без спеціальних навичок і знань у роботі з комп’ютером.

У подальшому планується розробти:

1. Кросплатформенність додатку.
2. Покращений дизайн.
3. Доповнення функціоналу (можливість редагувати різні типи файлів)

ПЕРЕЛІК ПОСИЛАНЬ

1. Mark J. Price C# 7 and .NET Core: Modern Cross-Platform Development - Second Edition/- Packt Publishing– с. 96 – 120.

2. Ben Albahari, Joseph Albahari C♯ 6.0: Pocket Reference /- O'Reilly Media; 1st edition - с. 117 – 123.

3. ANDREW TROELSEN, Philip Japikse, C# 6.0 and the .NET 4.6 Framework /- Apress, 2015 – с. 1083.

4. Joseph Albahari, Ben Albahari, C# 7.0 in a Nutshell: The Definitive Reference /- "O'Reilly Media, Inc.", 2017 – с. 120.

5. Griffiths I. Programming C# 8.0: Build Cloud, Web, and Desktop Applications / – "O'Reilly Media Inc.", 2019 – с. 140.

6. Коноваленко І.В. Програмування мовою C# 6.0 / – Тернопіль, ТНТУ, 2016. с. 227.

7. Desktop Guide (WPF .NET). [Електронний ресурс]: [Веб-сайт]. –ектронні дані. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-5.0 Дата звернення: 06.04.2021.

8. Dialog boxes overview (WPF .NET). [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/windows/dialog-boxes-overview?view=netdesktop-5.0 Дата звернення: 07.04.2021.

9. Git Branching. [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell Дата звернення: 07.04.2021.

10. Photoshop. About painting tools, presets, and options. [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://helpx.adobe.com/photoshop/using/painting-tools.html Дата звернення: 15.04.2021.

Додаток

ПРЕЗЕНТАЦІЯ

“Розробка графічного редактора на платформі .NetFramework, використовуючи технологію WPF”

Підготував: студент Грисюк В.Г.
Керівник: Трінтіна Н.А.

Мета, об’єект та предмет дослідження

- **Мета роботи**
  - Розробка додатка для редагування зображень
- **Об’єкт дослідження**
  - Існуючі додатки для редагування фото
- **Предмет дослідження**
  - Найбільш ефективний метод відображеня багатьох елементів у WPF

Аналоги

Paint Tool SAI

Платний додаток. Має бідний функціонал редагування зображень, більше орієнтований на малювання з планшетом.

Adobe Photoshop

Додаток вимагає платної підписки. Багато функціоналу для роботи з зображеннями, але має великі вимоги до потужності ПК.

Порівняння

	Adobe Photoshop	Paint Tool SAI
Платний	Так	Так
Вимагає потужний ПК	Так	Ні
Підтримка типів зображень	Більше 10	sai та png
Багато функціоналу для роботи з зображеннями	Так	Ні

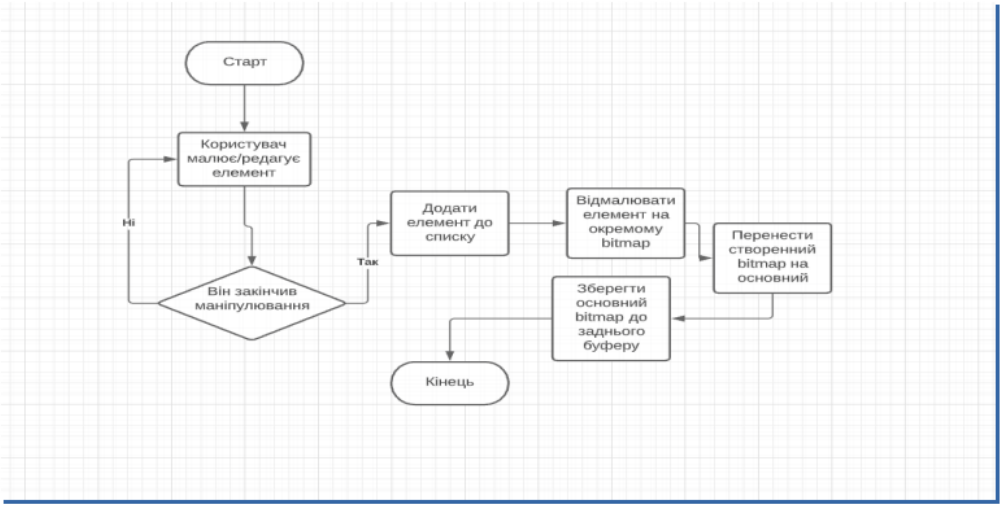
Метод дослідження

- Порівняти ефективність різних методів роботи з графічними елементами, та методами їх відображення у WPF.

Практичне значення отриманних результатів

- Було створено ефективний для роботи з графічними елементами метод, який має перевагу в швидкості в порівнянні зі стандартними методами.

Схема взаємодії користувача з полотном



ВИСНОВКИ

Було створено швидкий та легкий додаток для редагування графічного контенту. Основний алгоритм взаємодії з полотном може бути експортований як бібліотека, та повторно використаний у інших додатках C# WPF.

Участь у конференціях

- **II ВСЕЄУКРАЇНСЬКА НАУКОВО-ТЕХНІЧНА КОНФЕРЕНЦІЯ “СУЧАСНИЙ СТАН ТА ПЕРСПЕКТИВИ РОЗВИТКУ ІОТ”** З ТЕМОЮ “АКТУАЛЬНІСТЬ СТВОРЕННЯ ДОДАТКУ ДЛЯ РЕДАГУВАННЯ ЗОБРАЖЕНЬ”