

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА
ПРИРОДОКОРИСТУВАННЯ

**Навчально-науковий інститут автоматичної, кібернетики та
обчислювальної техніки**

"До захисту допущена"
Зав. кафедри комп'ютерних наук та
прикладної математики

«___» _____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

Розробка настільного застосунку для створення відеофайлів

Виконала: Шеметовська Єлизавета Юріївна
(прізвище, ім'я, по батькові)

(підпис)

група КНз-51

Керівник: доцент, к.п.н. Рощенюк А.М.
(науковий ступінь, вчене звання, посада, прізвище, ініціали)

(підпис)

Рівне – 2023

Національний університет водного господарства та природокористування

(повне найменування вищого навчального закладу)

НН інститут **автоматики, кібернетики та обчислювальної техніки**

Кафедра **комп'ютерних наук та прикладної математики** Освітньо-кваліфікаційний рівень **бакалавр**

Галузь знань 12 «Інформаційні технології»

(шифр і назва)

Спеціальність 122 «Комп'ютерні науки»

(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

" ____ " _____ 20__ року

ЗАВДАННЯ **НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

Шеметовській Єлизаветі Юріївні

(прізвище, ім'я, по батькові)

1. Тема роботи Розробка настільного застосунку для створення відеофайлів

керівник роботи Роценюк А.М., к. пед. н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "26" травня 2023 року

№ 449

2. Строк подання студентом роботи 31.05.2023

3. Вихідні дані до роботи технології, бібліотеки та пакети для розробки настільного застосунку для запису і відтворення відеоконтенту.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити настільний застосунок для запису та відтворення відеоконтенту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) мультимедійна презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	доцент Роценюк А.М.	7.11.22	7.11.22
Розділ 2	доцент Роценюк А.М.	12.12.23	12.12.23
Розділ 3	доцент Роценюк А.М.	7.03.23	7.03.23

7. Дата видачі завдання 01.11.2022

КАЛЕНДАРНИЙ ПЛАН


№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Вивчення літератури за обраної тематикою	01.11.2022 – 11.11.2022	виконала
2.	Формулювання постановки задачі	11.11.2022 – 16.12.2022	виконала
3.	Розробка алгоритму	16.12.2022 – 20.01.2023	виконала
4.	Здійснення програмної реалізації	20.01.2023 – 07.03.2023	виконала
5.	Розробка настільного застосунку	07.03.2023 – 7.04.2023	виконала
6.	Відлагодження програмного продукту	07.04.2023 – 21.04.2023	виконала
7.	Підготовка звіту кваліфікаційної роботи	21.04.2023 – 15.05.2023	виконала
8.	Підготовка мультимедійної презентації	15.05.2023 – 24.05.2023	виконала
9.	Підготовка до виступу	24.05.2023 – 30.05.2023	виконала

Студент

(підпис)

Шеметовська Є.Ю.
(прізвище та ініціали)

Керівник роботи



(підпис)

Роценюк А.М.
(прізвище та ініціали)

ЗМІСТ

РЕФЕРАТ	5
ВСТУП	6
РОЗДІЛ 1. АНАЛІЗ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РЕАЛІЗАЦІЇ ДАНОГО ЗАСТОСУНКУ	8
1.1. Історія створення відео та форматів	8
1.2. Аналіз переваг та недоліків існуючих застосунків	10
РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ТА ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ	13
2.1. Середовище розробки Microsoft Visual Studio	13
2.2. Мова програмування C# та графічна підсистема WPF	15
2.3. Мова користувальницького інтерфейсу XAML	18
2.4. Огляд потрібних бібліотек для створення та відтворення відео	25
2.5. Створення інсталятора для настільного застосунку	29
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ НАСТІЛЬНОГО ЗАСТОСУНКУ	35
3.1. Архітектура даного застосунку	35
3.2. Реєстрація та авторизація	39
3.3. Програмна реалізація навігації	44
3.4. Розробка логіки для запису відео різного виду	45
3.5. Перегляд відео та бібліотека записаних файлів	50
3.6. Сценарій роботи користувачького інтерфейсу	52
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	61
ДОДАТКИ	62

РЕФЕРАТ

Кваліфікаційна робота: 70 с., 42 рисунки, 3 додатки, 15 джерел.

Метою кваліфікаційної роботи є дослідження та створення єдиного програмного середовища, яке повинно надати можливість створювати відеофайли двох типів для користувачів.

Об'єкт дослідження – процес запису та відтворення відео різних типів.

Предметом дослідження – методи та засоби для запису відео різних форматів при використанні настільного застосунку за допомогою сучасних бібліотек і пакетів .NET.

Методи дослідження – мова програмування C# та графічна підсистема WPF.

Оскільки відеоконтент є досить актуальним у сучасному світі, то реалізований настільний застосунок, використавши новітні бібліотеки та пакети для запису якісного та унікального контенту. Для розробки додатку були використані наступні мови та засоби програмування: C#, WPF, XAML, WIX та з допомогою реляційних баз даних в середовищі SQL Server Management Studio (SSMS).

Ключові слова: C#, WPF, НАСТІЛЬНИЙ ЗАСТОСУНОК, XAML, MVVM, WIX.

ВСТУП

Відеоконтент є невід’ємною частиною життя у сучасному світі. З кожним днем з’являються нові можливості для запису відеофайлів на різних платформах, які надалі використовуються для навчання, роботи, соціальних мереж та різних цілей. Нині є великий попит на настільні додатки для створення відеофайлів, щоб користувачі могли швидко та зручно записувати відео.

Новизна роботи полягає в розробці настільного застосунку для створення унікального відеоконтенту, використовуючи новітні технології, що будуть легко підтримувані в майбутньому. Застосунок буде відкритий для модифікацій та додаванню додаткового функціоналу залежно від потреб користувачів.

Швидкі додатки для створення відео контенту може бути дуже корисним інструментом для навчання. Студенти можуть краще засвоювати матеріали, переглядаючи різні відеоінструкції, навчальні відео та презентації. Також викладачі можуть створювати власний контент, записувати відео уроки, втілювати власні унікальні ідеї для заохочення студентів.

За статистикою багато людей мають операційну систему Windows. На сьогодні для розробки настільних додатків найпоширенішими мовами є C# та Java. Вони є актуальними нині через швидку та ефективну роботу, незважаючи на кількість функціоналу та обсяг даних.

Великою перевагою настільних додатків є доступ до ресурсів ПК. Вони можуть використовувати різні ресурси без обмежень такі як: веб-камери, мікрофони, файлові системи тощо. Це може бути корисним для спеціалізованих додатків.

Настільні додатки є дуже популярними та багато різних людей їх використовують, тому що вони забезпечують підтримку та оновлення, завжди можна додати потрібний функціонал, який відповідає потребам користувачів. Зазвичай розробники документують код та процес розробки даного продукту для подальшої підтримки продукту.

Настільні додатки є більш безпечнішими, ніж веб-ресурси, оскільки вони інсталиуються локально на ПК та розробники можуть налаштувати механізми шифрування, що будуть забезпечувати високий рівень безпеки.

Досягнення поставленої мети кваліфікаційної роботи вимагає дослідження та виконання таких завдань:

- Дослідити варіанти використання бібліотек, плагінів для створення такого контенту.
- Дослідити можливості записування аудіо та відео одночасно та зберігати у відповідному форматі.
- Дослідити, як налаштувати доступ до до відеокамери та екрану ПК.
- Додати створення облікового запису, щоб мати доступ до реалізованого функціоналу.
- Забезпечити зрозумілий та зручний графічний інтерфейс для користувача.
- Забезпечити можливість вибору шляху на ПК для збереження відеофайлів.
- Розробити можливість мати свою бібліотеку із створених відео, відтворення їх, а також перегляд інших відео, вибраних у провіднику.
- Розробити різні конвертори для роботи з форматами відео та аудіо-файлами.

РОЗДІЛ 1. АНАЛІЗ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ РІШЕНЬ ДЛЯ РЕАЛІЗАЦІЇ ДАНОГО ЗАСТОСУНКУ

1.1. Історія створення відео та форматів

До винайдення кінофільмів було кілька спроб створити рухомі зображення. Одним із найперших прикладів був зоотроп, який представляв собою циліндр із прорізами, які дозволяли глядачам бачити низку зображень у швидкій послідовності. Перший кінофільм був створений наприкінці 19 століття кількома винахідниками, зокрема Томасом Едісоном і братами Люм'єри [2]. Такі фільми були короткими і часто склалися з простих та елементарних дій. Це відбувалося без звуку, а просто рухи людей та танці.

На початку 20 століття кіноіндустрія почала процвітати, і Голлівуд став центром кіновиробництва. Технологічний прогрес дозволив знімати більш довгі та складні фільми, а наприкінці 1920-х до фільмів додали звук.

У 1950-1960-х роках телебачення стало популярним засобом трансляції відеоконтенту. Це призвело до розробки нових технологій, таких як відеофільми, які дозволяють записувати та відтворювати відео вміст. Це також був час, коли була винайдена перша портативна відеокамера, що полегшило людям створення власних відео.

У 1980-х роках поява персональних комп'ютерів і цифрових технологій зробила революцію у виробництві відео. Редагування та спеціальні ефекти тепер можна виконувати в цифровому вигляді, а високоякісне відео можна створювати за допомогою настільних комп'ютерів.

У 2000-х і 2010-х роках висока чіткість і технологія 4K стали більш поширеними, дозволяючи отримувати ще більш реалістичні та деталізовані зображення.

Сьогодні створення відео доступніше, ніж будь-коли, завдяки смартфонам, комп'ютерам та іншим мобільним пристроям, які дозволяють людям записувати та редагувати відео в будь-якому місці.

Історія створення різних форматів відео тісно пов'язана з еволюцією відео технологій відтворення відео.

Найпершими відеоформати були аналоговими і включають такі формати, як VHS, Betamax і Video 8. Ці формати використовувалися в 1980-90-х роках. Загалом відео записували на магнітну стрічку. Хоча у 1990-х роках також почали з'являтися формати цифрового відео. У 1983 році були представлені одні із перших цифрових форматів Digital Betacam. Інші цифрові формати включали DVCAM, DVCPRO та MiniDV.

Уже у 2000-х роках почали з'являтися формати високої чіткості (HD). Одним із перших форматів HD був представлений у 2003 році – HDV. Інші формати HD включали AVCHD, XDCAM і HDCAM. У 2010-х роках почали з'являтися 4K та інші формати надвисокої чіткості (UHD). Ці формати мають набагато вищу роздільну здатність, ніж формати HD, і включають такі формати, як 4K UHD, 8K UHD та HDR.

Разом із цими розробками в технологіях запису та відтворення відео також відбувся прогрес у технологіях кодування та стиснення відео. Ці технології дозволяють стискати відео в файли меншого розміру, що полегшує їх зберігання та обмін в мережі Інтернет.

Поширені формати відеофайлів сьогодні [7]:

1. MP4 – це формат, який сумісний майже з будь-якою системою. Він може містити будь-яку версію MPEG-4 і H.264.

2. MKV – це формат мультимедійного контейнера, який відомий своєю функцією зберігання необмеженої кількості відео, зображень, аудіо, субтитрів та інших файлів в одному. Цей мультимедійний контейнер Matroska є безкоштовним і відкритим форматом і в основному використовується для онлайн-якості HD.

3. AVI є одним із найстаріших контейнерів і більше не використовується дуже поширений, але все ще широко підтримується завдяки багатьом існуючим зміст є в ньому. Крім того, він підтримує велику кількість кодеків.

4. MOV – це контейнер, пов’язаний із Apple QuickTime Player і є його внутрішній формат. Швидше за все, всередині файлу MOV ви знайдете Відеодані MPEG-4. Тому в більшості випадків ви можете перейменувати файл MOV у файл MP4, і він працюватиме так само. Основна відмінність файлів MOV і MP4 полягає в тому, що файли MOV іноді захищені від копіювання. Він запобігає обміну та відтворенню неавторизованими користувачами.

Нині існує багато різноманітних відео форматів, кожен із яких має свої сильні та слабкі сторони. Вибір правильного формату залежить від кількох факторів, зокрема передбачуваного використання відео, доступної пам’яті та пропускної здатності, а також можливостей відтворення на різних пристроїв.

1.2. Аналіз переваг та недоліків існуючих застосунків

При аналізі програм для створення відео, слід завжди враховувати кілька ключових факторів. Ці фактори включають функціональність програми, платформу та технологію, що використовуються для розробки програми, а також досвід користувача та зручність використання програми.

По-перше, при оцінці аналогів важливо враховувати особливості і функціональність програми. Популярні програми для створення відео можуть містити такі функції, як редагування відео, спеціальні ефекти, редагування аудіо та можливість експорту відеофайлів у різні формати. Важливо оцінити, які функції є найбільш важливими для вашого передбачуваного використання.

По-друге, платформа та технічний стек, що використовуються для розробки програми, можуть впливати на продуктивність, масштабованість і зручність обслуговування програми. Стек технологій може включати мови програмування, фреймворки, бібліотеки та інструменти, які використовуються для розробки програми. Важливо тестувати аналоги, щоб переконатися, що вони побудовані з використанням надійного та безпечного стеку технологій, який можна підтримувати та масштабувати протягом тривалого часу.

По-третє, досвід користувача та зручність використання програми мають вирішальне значення для успіху програми. Додаток повинен бути інтуїтивно зрозумілим і простим у використанні, з чистим і сучасним інтерфейсом, який дозволяє користувачам створювати високоякісні відео швидко і легко. Важливо оцінювати аналоги, щоб переконатися, що вони забезпечують позитивний досвід користувача та відповідають потребам вашої цільової аудиторії.

Існує декілька аналогів застосунку для запису відео. Наприклад: OBS (Open Broadcaster Software) та Camtasia.

Нами був проведений аналіз наведених аналогів і можна виділити основні переваги та недоліки:

OBS(Open Broadcaster Software) застосунок є потужним та універсальним програмним забезпеченням для потокового передавання та запису, яке пропонує широкі можливості налаштування та сумісність із різними платформами (рис. 1.2.1). Однак його ресурсомісткий характер, складність інтерфейсу, обмежені можливості підтримки та відсутність вбудованих можливостей редагування відео є деякими факторами, які варто враховувати при використанні. Настільний застосунок написаний на мові програмування C/C++, що дозволяє OBS надавати широкі можливості та забезпечувати надійну продуктивність під час захоплення, кодування та потокової передачі аудіо- та відео контенту. Однак цей додаток має обмеження для підтримки та вдосконалення існуючих функцій.

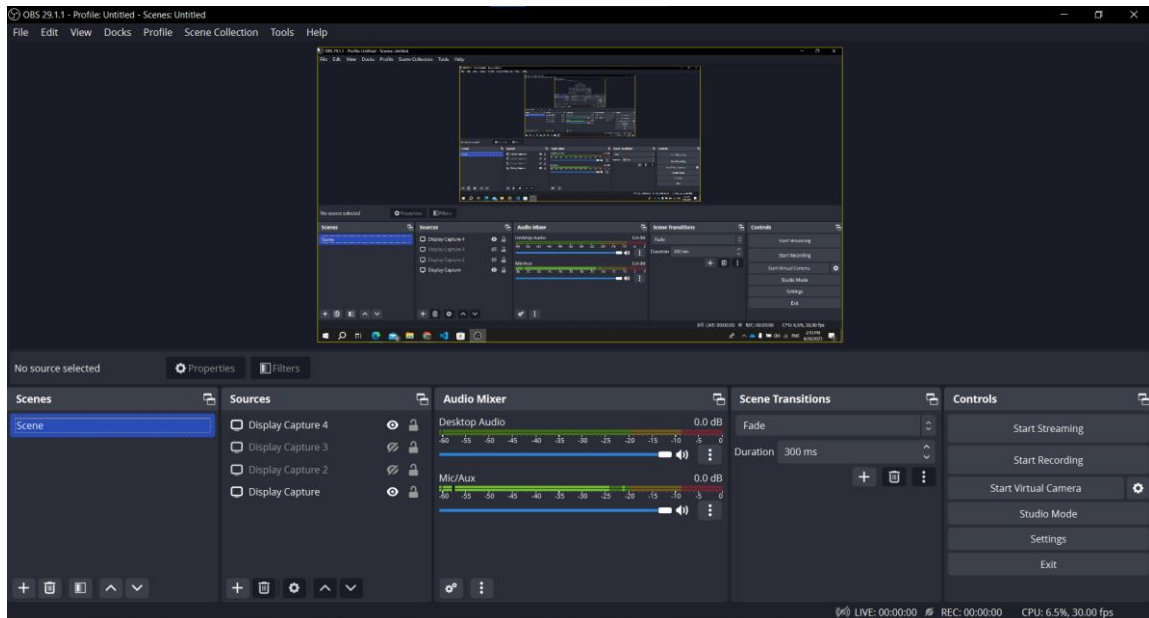


Рис. 1.2.1. Візуалізація OBS застосунку

Camtasia є багатофункціональним програмним забезпеченням для запису екрана та редагування відео, яке пропонує потужні можливості для створення високоякісного відео. Основні функціональні можливості та частини програми, які потребують високої продуктивності, зазвичай реалізуються за допомогою C++, а графічний інтерфейс користувача та інші частини розроблені за допомогою C#. Застосунок є популярним вибором для творців контенту та викладачів через його інтуїтивно зрозумілий інтерфейс, розширені функції редагування та параметри інтерактивного вмісту. Однак його вища ціна, крива навчання, вимоги до ресурсів і обмежені функції співпраці є факторами, які слід враховувати під час оцінки того, чи підходить Camtasia для конкретних потреб і бюджету.

Виходячи з всіх вище перерахованих факторів, дивлячись на кількість додатків для створення відео, важливо враховувати функціональні можливості додатка, платформу та технологію, використані для розробки додатка, а також досвід користувача та простоту використання додатка. Ретельно оцінюючи ці фактори, розробники можуть вибрати найбільш відповідний застосунок для задоволення потреб своїх користувач.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ТА ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1. Середовище розробки Microsoft Visual Studio

Середовище розробки Microsoft Visual Studio – це інтегроване середовище для розробки програмних продуктів від Microsoft. Дане середовище дозволяє створювати різноманітні програмні продукти: консольні програми, програми з графічним інтерфейсом, наприклад додатки Windows Forms або Windows Presentation Foundation, а також Web-додатки тощо.

Microsoft Visual Studio 2022 – це найновіша версія популярного інтегрованого середовища розробки (IDE), яке широко використовується розробниками для створення програмних додатків на різних платформах (рис. 2.1.1).

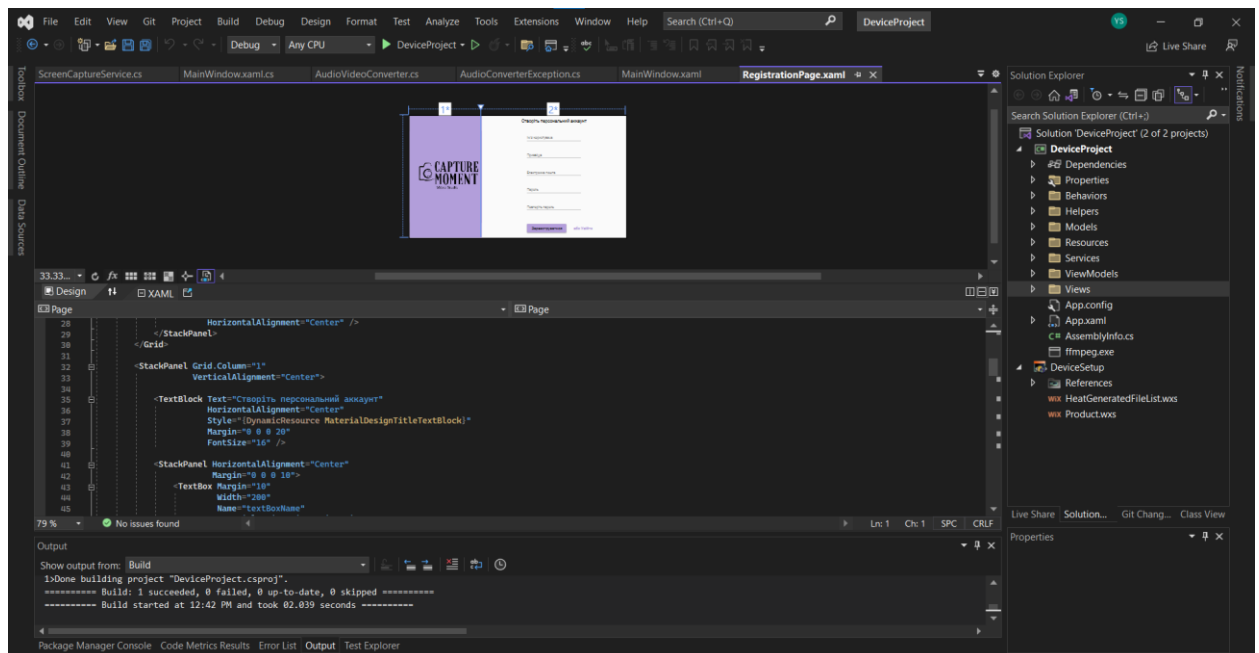


Рис. 2.1.1. Візуалізація середовища розробки настільного застосунку

Перевагами даного середовища є:

1. Visual Studio 2022 підтримує розробку програм для Windows, Android, iOS і веб-платформ.

2. Редактор коду у VS 2022 надає такі розширені функції, як автозавершення коду та навігація по коду, щоб допомогти розробникам швидко та легко писати кращий код.

3. Вбудована технологія IntelliSense, яка значно покращує та спрощує процес написання коду, допомагає підвищити продуктивність.

4. Visual Studio 2022 легко інтегрується зі службами Azure, що полегшує розробникам розгортання та керування своїми програмами в хмарі.

5. У Visual Studio 2022 суттєво покращено продуктивність, що забезпечує швидкий час запуску, редагування коду та збірки.

Недоліками є:

1. Visual Studio 2022 вимагає потужного комп'ютера з високою оперативною пам'яттю та характеристиками ЦП, які можуть бути доступні не всім.

2. Інтерфейс Visual Studio 2022 може бути трішки складним, особливо для новачків, які не звикли працювати з IDE.

З виходом нової версії VS було створення багато додаткових функцій:

1. Hot Reload дозволяє розробникам бачити зміни, внесені до їх коду, без необхідності перезапускати свою програму. Це є дуже корисним при розробленні настільних застосунків, а саме при покращенні дизайну.

2. VS 2022 використовує 64-розрядну систему, що дає змогу повністю використовувати ресурси сучасних комп'ютерів і забезпечувати більш високу продуктивність.

3. IntelliCode тепер включає моделі машинного навчання, які надають точніші пропозиції щодо доповнення та форматування коду. Працює як потужний набір інструментів завершення коду. Інструменти мають можливість розпізнавати контекст коду: імена змінних, функції та тип згенерованого коду.

4. Visual Studio 2022 містить нові інструменти налагодження, такі як Time Travel Debugging, який дозволяє розробникам налагоджувати проблеми, які виникали в минулому.

5. Можливість проведення сеансів роботи в режимі реального часу (Live Share), які забезпечують можливість налагодження та редагування у реальному часі для команд незалежно від мов програмування, які використовуються, або типів програм, які створюються.

2.2. Мова програмування C# та графічна підсистема WPF

Мова програмування C# – це об'єктно-орієнтована мова, яка розроблена та підтримувана компанією Microsoft. Вважається мовою високого рівня, яка розроблена як проста, потужна та безпечна для типів [5, 25]. Широко використовується для розробки настільних програм Windows, веб-програм і мобільних програм для пристроїв Android та iOS.

C# інтегрується з іншими мовами .NET: C# є частиною платформи .NET, що означає, що він може взаємодіяти з іншими мовами .NET, такими як Visual Basic і F#. C# і Visual Basic .NET є двома з багатьох мов програмування, доступних для .NET. Загалом Common Language Runtime (CLR) платформи є спільним середовищем виконання для декількох мов програмування, включаючи C++, F#, Cobol .NET.

C# є строго типізованою мовою, що означає, що вона перевіряє типи змінних під час компіляції, щоб запобігти пов'язаним з типом помилкам під час виконання. Також, C# є кросплатформною мовою, що означає, що її можна використовувати для розробки програм для Windows, Linux і macOS.

Мова C# розроблялась, як мова прикладного рівня CLR. CLR (Common Language Runtime) – це середовище виконання в .NET Framework, яке запускає код та допомагає полегшити процес розробки, надаючи різні служби [4, 28]. Власне кажучи, він відповідає за керування виконанням програм .NET незалежно від мови програмування .NET (рис. 2.2.1).

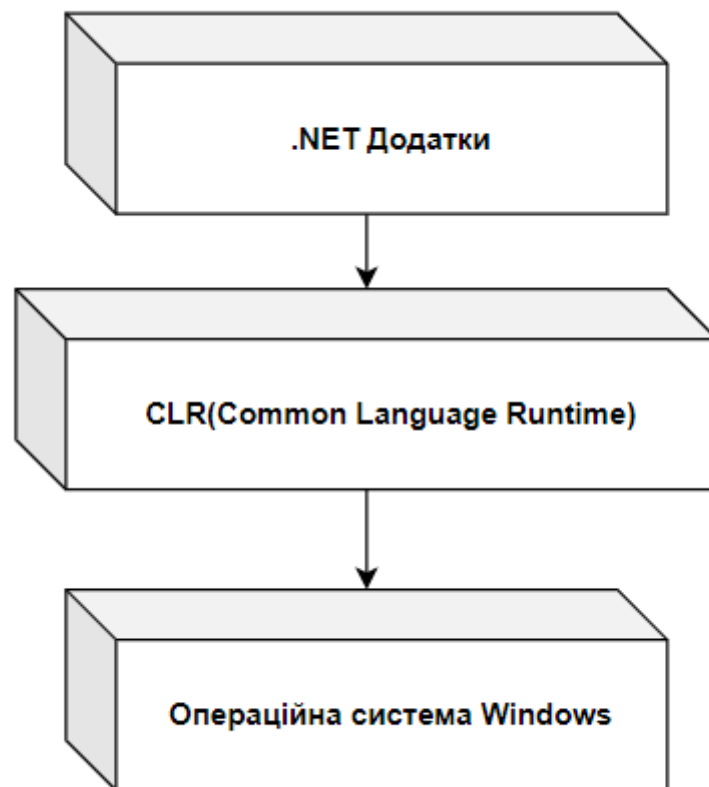


Рис. 2.2.1. Структура CLR(Common Language Runtime)

WPF(Windows Presentation Foundation) – це структура графічного інтерфейсу користувача (GUI), розроблена Microsoft для створення настільних програм на платформі Windows [3, 10]. WPF було вперше представлено у випуску .NET Framework 3.0 2006 року.

WPF використовує XAML (Extensible Application Markup Language) для розробки інтерфейсів користувача, що є мовою на основі XML, яка відокремлює дизайн інтерфейсу користувача від логіки програми [13]. WPF має чіткий розподіл завдань між дизайном інтерфейсу користувача та бізнес-логікою, що полегшує підтримку та оновлення програм з часом. Система WPF дуже гнучка, що полегшує створення складних макетів і їх динамічне коригування.

Також варто зазначити, що Windows як і раніше підтримує додатки, засновані на фреймворку .NET і таких технологіях, як Windows Forms та WPF (рис. 2.2.2).

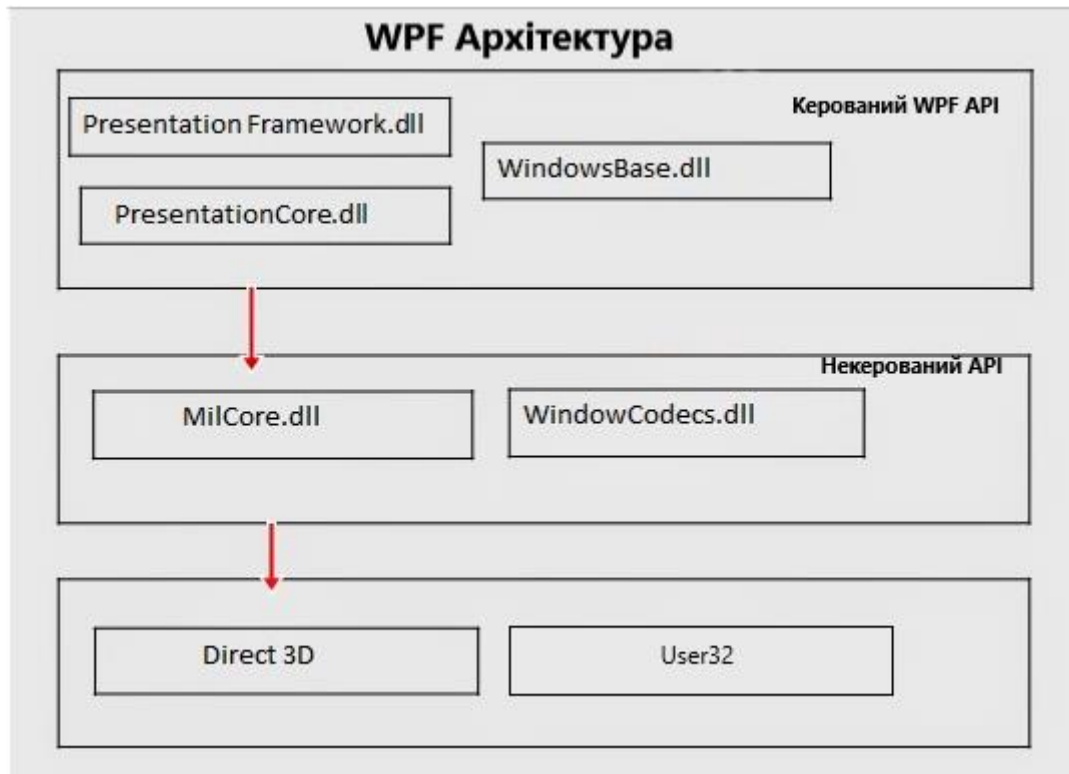


Рис. 2.2.2. Схематичне відображення архітектури WPF

WPF система дозволяє створювати дуже сучасні та зручні інтерфейси для користувача, забезпечуючи підтримку ряду елементів інтерфейсу, 2D і 3D графіки, мультимедіа та анімації. Також є можливість створювати користувацькі елементи керування, на основі наявних елементів керування, або нові з нуля, що полегшує створення повторно використовуваних компонентів інтерфейсу користувача. Це перевагою для розробників, тому що це є можливістю створювати власні унікальні компоненти.

Наступною перевагою WPF є потужна система компоновання (grid layout), яка дозволяє динамічно впорядковувати та змінювати розміри елементів інтерфейсу користувача, що полегшує створення інтерфейсів користувача, які адаптуються до різних розмірів екрана та роздільної здатності.

Наступне, WPF надає потужні можливості зв'язування даних (data binding), дозволяючи розробникам легко прив'язати дані до елементів інтерфейсу користувача та створювати адаптивні інтерфейси користувача, які автоматично оновлюються, коли базові дані змінюються. Тобто, можна прив'язати будь-яку властивість, змінну до компонента (рис. 2.2.3).

```
Content="{Binding RecordingControlButtonText}" Command="{Binding ControlRecording}"
```

Рис. 2.2.3. Прив'язка даних у XAML

Наразі, WPF все ще активно розробляється та підтримується корпорацією Microsoft, тому і є популярним вибором для створення настільних програм для Windows.

2.3. Мова користувальницького інтерфейсу XAML

XAML(Extensible Application Markup Language) – є потужною та гнучкою мовою розмітки, заснована на XML, для опису графічного інтерфейсу користувача [3, 20]. Зазвичай XAML використовується в програмах, побудованих на платформі .NET, включаючи настільні, веб- та мобільні програми.

XAML не взаємодіє безпосередньо із середовищем CLR .NET Framework та не вважається однією з поширених мов, які використовують CLR. Натомість XAML має власну систему типів. Проте так як WPF використовує систему аналізу XAML, то система побудована на основі CLR та її типів. Під час аналізу XAML для WPF типи XAML зіставляються з типами CLR для створення представлення середовища виконання.

Можна зіставити типи XAML з будь-якою іншою системою типів незалежно від рівня специфікації мови XAML, це може бути не обов'язково CLR, але для цього знадобиться використання іншого аналізатора XAML.

Також є додаткові ключові моменти щодо XAML:

- Extensibility – XAML підтримує розширюваність за допомогою спеціальних елементів керування, стилів і ресурсів. Можна створювати власні елементи та поведінку, щоб покращити функціональність і зовнішній вигляд різних програм.

- Data Binding – XAML надає потужні можливості зв'язки даних(binding), дозволяючи прив'язувати елементи інтерфейсу користувача до властивостей, змінних та методів, що відкриті для доступу. Це забезпечує автоматичну синхронізацію інтерфейсу користувача з основними даними, запобігання змін даних вручну та підвищуючи швидкість реагування програми.

- Separation of Concerns – XAML дозволяє чітко розділити проблеми між інтерфейсом користувача та логікою програми. Інтерфейс користувача може бути розроблений і змінений дизайнерами за допомогою UI інструментів, тоді як бекенд розробники можуть зосередитися на написанні бізнес-логіки на C#. Наприклад використання MVVM паттерна для розділення UI інтерфейсу та бізнес-логіки.

- Styling and Animation – XAML підтримує створення різних стилів і шаблонів, які визначають візуальний вигляд і поведінку елементів інтерфейсу користувача. Також включає підтримку анімацій та переходів, що дозволяє розробникам створювати візуально заохочуючі елементи для користувачів.

Перейдемо до синтаксису файлу XAML. Розпочнемо з кореневого елемента (root).

У XAML кожен файл XAML повинен мати лише один кореневий елемент. Кореневий елемент служить відправною точкою для визначення структури та вмісту інтерфейсу користувача. Він інкапсулює всі інші елементи в ієрархії XAML. Як правило, кореневий елемент може бути Window або Page для сторінки, ResourceDictionary для зовнішнього словника або Application для визначення програми.

Далі, простори імен (namespaces), що використовуються в XAML для посилань і використання елементів із зовнішніх бібліотек або визначення

настроюваних елементів. Простори імен XAML дозволяють зіставляти простори імен XML із просторами імен CLR і пов'язувати їх із префіксом. Цей префікс використовується для посилання на елементи з цього простору імен у файлі XAML.

Щоб оголосити простір імен використовується атрибут `xmlns` та `xmlns:x`. Він додається до кореневого елемента та включає URI простору імен XML і відповідний префікс.

Наприклад, цей код оголошує простір імен із префіксом "behaviors" і зіставляє його з простором імен CLR "DeviceProject.Behaviors":

```
xmlns:behaviors="clr-namespace:DeviceProject.Behaviors"
```

Якщо брати префікс `x`, то є перелік найпоширеніших конструкцій [3, 42]:

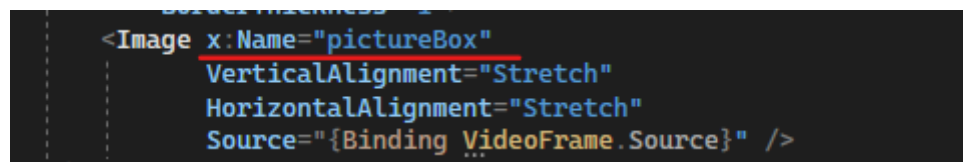
- **x:Key:** встановлює унікальний ключ для кожного ресурсу в `ResourceDictionary`, `x:Key`, ймовірно, відповідатиме за 90% використання `x:`, яке можна побачити у типовій розмітці.

```
<Style x:Key="MaterialDesignRaisedLightButton" TargetType="{x:Type ButtonBase}" BasedOn="{StaticResource MaterialDesignRaisedButton}"></Style>
```

І потім використання в кодї:

```
<Button Style="{StaticResource MaterialDesignRaisedLightButton}" />
```

- **x>Name:** цей атрибут можна додати до будь-якого елемента об'єкта XAML (рис. 2.3.1), де потрібно мати змогу посилатися на створений екземпляр середовища виконання як частину вашої логіки коду.

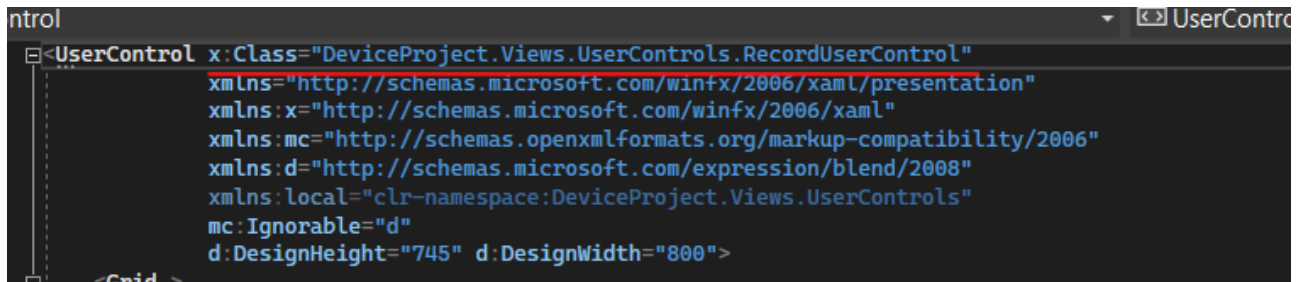


```
<Image x:Name="pictureBox"
        VerticalAlignment="Stretch"
        HorizontalAlignment="Stretch"
        Source="{Binding VideoFrame.Source}" />
```

Рис. 2.3.1. Приклад використання `x>Name` у кодї

- **x:Class:** визначає простір імен і ім'я класу для класу, який забезпечує код для сторінки XAML (рис. 2.3.2). Програма повинна мати такий клас для

підтримки коду позаду, і тому можна завжди бачити відображення x:, навіть якщо немає ресурсів.



```

<UserControl x:Class="DeviceProject.Views.UserControls.RecordUserControl"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:local="clr-namespace:DeviceProject.Views.UserControls"
  mc:Ignorable="d"
  d:DesignHeight="745" d:DesignWidth="800">

```

Рис. 2.3.2. Приклад використання x:Class у кодї

- **x:Type**: створює посилання на тип на основі імені типу. Це використовується для визначення атрибутів, які приймають Type, наприклад Style::TargetType, хоча часто властивість має власне перетворення рядка в тип таким чином, що використання розширення розмітки x:Type є необов'язковим.

```

<Style x:Key="MaterialDesignRaisedLightButton" TargetType="{x:Type
  ButtonBase}" BasedOn="{StaticResource MaterialDesignRaisedButton}"></Style>

```

Це означає щоб ми використовуємо тип ButtonBase і додаємо кілька додаткових стилей.

- **x:Null**: атрибут, що знадобиться для присвоєння нульового значення в XAML. Наприклад, коли рядок вибраний у DataGrid, ми присвоємо значенню null.

```

<Style.Triggers>
  <Trigger Property="DataGridCell.IsSelected" Value="True">
    <Setter Property="Background" Value="{x:Null}" />
  </Trigger>
</Style.Triggers>

```

В загальному просторі імен у XAML забезпечують спосіб організації та повторного використання компонентів інтерфейсу користувача в кількох файлах і проектах XAML.

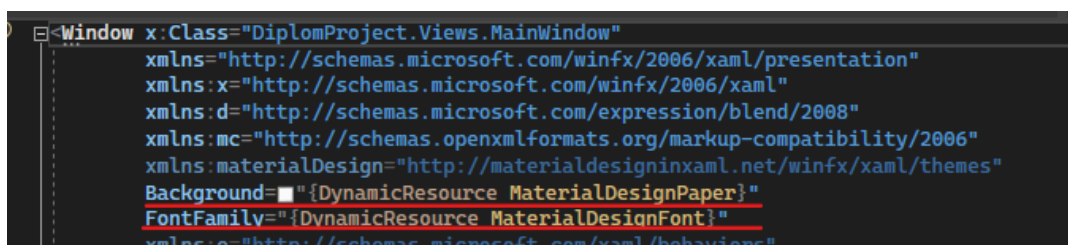
Синтаксис розмітки XAML використовується для створення класу або структури CLR шляхом оголошення елемента XML. У синтаксисі елемента об'єкта примірник починається з лівої кутової дужки (<), за якою йде ім'я типу

класу або структури. Атрибути можуть бути оголошені в елементі об'єкта, розділені пробілами та необов'язково після скісної риски (/) і прямої кутової дужки (>), щоб закрити елемент, або відкриваючої прямої кутової дужки (>), щоб дозволити додатковий вміст усередині елемент. Наявність відповідного закриваючого тегу необхідна для правильного вкладення та балансу з іншими тегами.

Фігурні дужки ({ та }) вказують на використання розширення розмітки. Це конкретне використання вказує процесору XAML обробляти значення атрибута інакше, ніж літеральний рядок або значення, яке перетворюється на рядок.

Найпоширенішими розширеннями розмітки є Binding, яке використовується для виразів зв'язування даних, а ресурс посилається на StaticResource і DynamicResource. Використовуючи розширення розмітки, ви можете використовувати синтаксис атрибутів для надання значень властивостей, навіть якщо властивість зазвичай не підтримує синтаксис атрибутів. Розширення розмітки часто використовують проміжні типи виразів, щоб увімкнути такі функції, як відкладення значень або посилання на інші об'єкти, які існують лише під час виконання.

Ось наприклад підключення стилів Material design системи (рис. 2.3.3), використання DynamicResource, що свідчить про можливу зміну стилю під час виконання програми [11].



```

<Window x:Class="DiplomProject.Views.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
  Background="{DynamicResource MaterialDesignPaper}"
  FontFamily="{DynamicResource MaterialDesignFont}"
  xmlns:e="http://schemas.microsoft.com/xaml/behaviors"

```

Рис. 2.3.3. Приклад використання DynamicResource

Також можна використати прив'язку даних до елементу (рис. 2.3.4).

```
<Image x:Name="pictureBox"
        VerticalAlignment="Stretch"
        HorizontalAlignment="Stretch"
        Source="{Binding VideoFrame.Source}" />
```

Рис. 2.3.4. Приклад прив'язки даних(binding) до елементу

Якщо нам потрібно використати прив'язку, але застосувати додатково реалізовану логіку, то можна використати конвертори. Вони можуть змінювати дані з одного типу на інший, наприклад ми використали для передачі значення індекса вибраного елемента в списку, то ми можемо написати конвертер, який поверне значення індексу у списку. Щоб зробити це, потрібно створити клас, який реалізує інтерфейс `IValueConverter`, а потім реалізувати методи `Convert` і `ConvertBack`.

```
public class IndexConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        if (value is ListViewItem item && item.DataContext is AssetModel fileItem)
        {
            ListView? listView = ItemsControl.ItemsControlFromItemContainer(item)
as ListView;
            if (listView != null)
            {
                int index = listView.Items.IndexOf(fileItem);
                return index;
            }
        }

        return -1;
    }

    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
```

```

if (value is int index)
{
    ListView listView = parameter as ListView;
    if (listView != null && index >= 0 && index < listView.Items.Count)
    {
        return listView.Items[index];
    }
}

return Binding.DoNothing;
}
}

```

Слід також знати про Attached behaviors and behaviors при розробці клієнтської частини. Зазвичай ці поведінки використовуються у фреймворках на основі XAML, таких як WPF і Xamarin.Forms. Це концепції розробки програмного забезпечення, що дозволяють інтегрувати додаткову функціональність до існуючих компонентів інтерфейсів, не змінюючи їх основного коду.

Отже, Attached behaviors дозволяє приєднувати додаткові властивості, методи та обробники подій до існуючих елементів інтерфейсу, які спочатку не є частиною визначеного API. Вони сприяють чіткому розподілу інтересів шляхом інкапсуляції логіки поведінки в окремому класі. Потім ці властивості або події можна приєднати до будь-якого елемента інтерфейсу користувача в XAML, розширюючи їх поведінку.

Attached behaviors зазвичай реалізуються як прикріплені властивості (attached properties) або вкладені події (attached events).

Синтаксис для оголошення вкладеної властивості в XAML:

```

<FrameworkElement>
    <FrameworkElement.AttachedProperty>
        <!-- Значення властивості або додаткові елементи XAML -->
    </FrameworkElement.AttachedProperty>
</FrameworkElement>

```


Синтаксис для прикріплення обробника подій до прикріпленої події в XAML:

```
<FrameworkElement AttachedEvent="Event_Handler_Method" />
```

Щодо Behaviors, вони зазвичай асоціюються з такими фреймворками, як Blend SDK, є способом інкапсуляції багаторазово використовуваної логіки інтерактивності та приєднання її до елементів інтерфейсу користувача.

Behaviors можна приєднати до елементів інтерфейсу за допомогою синтаксису, специфічного для фреймворку (наприклад, за допомогою колекції «Interaction.Behaviors»). Для цього потрібно завантажити пакет: **Microsoft.Xaml.Behaviors.Wpf** та підключити його за допомогою префікса в кореневому елементі.

Синтаксис налаштування властивості Behavior подібний до налаштування властивостей звичайного елемента, наприклад:

```
<FrameworkElement>
  <Interactivity:Interaction.Behaviors>
    <Behaviors:CustomBehavior SomeProperty="Value" />
  </Interactivity:Interaction.Behaviors>
</FrameworkElement>
```

2.4. Огляд потрібних бібліотек для створення та відтворення відео

Emgu CV – це міжплатформна оболонка .Net для бібліотеки обробки зображень OpenCV [8]. Дозволяє працювати функціям OpenCV викликається з сумісних .NET мов, таких як C#, VB тощо. Обгортку можна скомпілювати в Mono і працювати на пристроях Windows, Linux, Mac OS X, iPhone, iPad і Android. Для реалізації алгоритму потрібно встановити такі пакети (рис. 2.4.1).

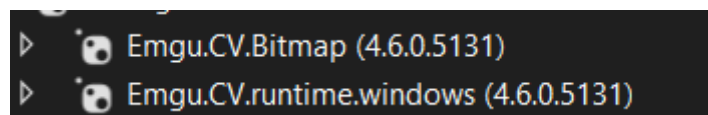


Рис. 2.4.1. Пакети для налаштування Emgu CV

Далі потрібно обробити event (подію) коли Image is grabbed (ImageGrabbed). І всередині встановлюємо для Image компоненту картинку, яку ми отримуємо за допомогою веб-камери.

Також у коді потрібно реалізувати Converter при встановленні картинки (рис. 2.4.2), він для конвертування Mat (Emgu CV image container) у Bitmap картинку, яку ми можемо використовувати для WPF елементів.

```
public class BitmapSourceConvert
{
    [DllImport("gdi32")]
    private static extern int DeleteObject(IntPtr o);
    public static BitmapSource ToBitmapSource(Mat image)
    {
        using (System.Drawing.Bitmap source = image.ToBitmap())
        {
            IntPtr ptr = source.GetHbitmap();

            BitmapSource bs = System.Windows.Interop.Imaging.CreateBitmapSourceFromHBitmap(
                ptr,
                IntPtr.Zero,
                Int32Rect.Empty,
                System.Windows.Media.Imaging.BitmapSizeOptions.FromEmptyOptions());

            DeleteObject(ptr);
            return bs;
        }
    }
}
```

Рис. 2.4.2. Конвертер для перетворення Mat image container у Bitmap

NAudio – це аудіо бібліотека з відкритим вихідним кодом для середовища .NET, розроблена by Mark Heath [12]. Він надає широкий спектр функцій для роботи з аудіо в програмах .NET, включаючи відтворення та запис аудіо, перетворення аудіо файлів тощо.

Записуємо спочатку аудіо у .wav file (рис. 2.4.3). Після чого можемо конвертувати у .mp4.

```

1 reference
private void RecordAvailableDataOfWaveSource(object sender, WaveInEventArgs e)
{
    if (waveFileWriter != null)
    {
        waveFileWriter.Write(e.Buffer, 0, e.BytesRecorded);
        waveFileWriter.Flush();
    }
}

```

Рис. 2.4.3. Уривок коду для запису аудіо

FFmpeg – це програмне забезпечення з відкритим вихідним кодом, яке надає набір інструментів обробки відео та аудіо [9]. Використовуючи FFmpeg у WPF, ви маєте змогу безпосередньо читати, записувати, конвертувати та маніпулювати відео- та аудіофайлами з програми.

Щоб налаштувати та використовувати FFmpeg у програмі, потрібно завантажити найновішу версію, далі додати у папку проекту, і додати налаштування (рис. 2.4.4).

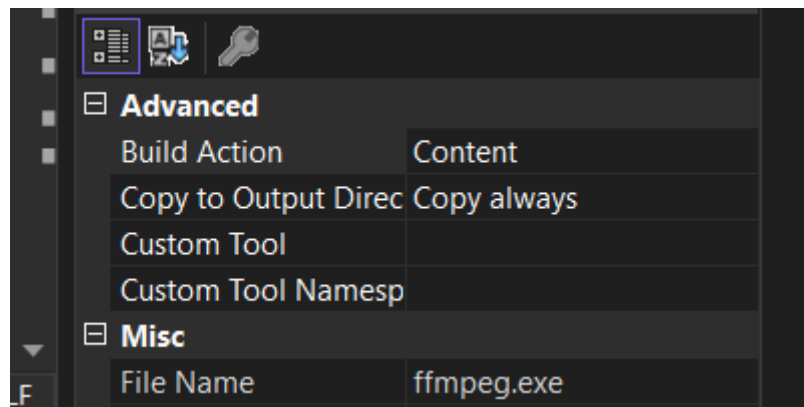


Рис. 2.4.4. Налаштування ffmpeg.exe файлу

Це дасть можливість у коді прописати шлях до папки Debug і використати exe. За допомогою FFmpeg, ми можемо об'єднати записані відео та аудіофайли в єдиний синхронізований вихідний файл. Причиною такого додаткового процесу є обмеження WPF щодо записування відео безпосередньо з аудіо.

Для вирішення цієї проблеми був створений конвертер, який приймає окремі відео- та аудіофайли .mp4 як вхідні дані та використовує можливості

FFmpeg для об'єднання їх в один файл. Цей процес гарантує належну синхронізацію аудіо з відповідним відео, що забезпечує плавний перегляд.

Наприклад, ось деякі flags, які будуть корисними для нашого проекту:

- `-i`` – використовується для позначення вхідного файлу.
- `-c:v`` – параметр визначає відео стандарт, який буде використовуватися для вихідного відео. У цьому випадку вказано копію, що означає, що відеопотік буде скопійовано з входу на вихід без будь-якого перекодування. Також це є перевагою, коли потрібно зберегти оригінальну якість відео та лише маніпулювати звуком.

- `-c:a aac`` – параметр визначає аудіо стандарт, який буде використовуватися для вихідного відео. `aac`` вказано тут, щоб використати стандарт Advanced Audio Coding (AAC). AAC – це кодек, який часто використовується для високоякісного стиснення звуку.

- `-c:v libx264`` – параметр визначає відеокодек, який буде використовуватися для вихідного відео. У цьому випадку вказано `libx264``, популярний відеокодек із відкритим кодом, відомий своїм високоякісним стисненням. Він забезпечує хороший баланс між якістю відео та розміром файлу.

- `-crf 23`` – параметр встановлює коефіцієнт постійної швидкості (CRF) для кодування відео. Значення CRF визначає рівень стиснення та якість вихідного відео. Нижчі значення призводять до вищої якості, але більшого розміру файлу, тоді як вищі значення призводять до нижчої якості, але меншого розміру файлу. У цьому випадку вказано значення CRF 23, що забезпечує баланс між якістю та розміром.

- `-preset medium`` – параметр встановлює попередні налаштування кодування для відео. Попереднє налаштування визначає швидкість і ефективність кодування. У цьому випадку вказується пресет "MEDIUM", який забезпечує баланс між швидкістю кодування та розміром вихідного файлу.

- `-b:a 128k`` – параметр встановлює бітрейт аудіо для вихідного відео. Бітрейт аудіо визначає кількість даних, які використовуються для представлення

аудіо за одиницю часу. У цьому випадку вказано бітрейт 128 кбіт/с (кілобіт на секунду), що забезпечує баланс між якістю звуку та розміром файлу.

2.5. Створення інсталятора для настільного застосунку

WiX Toolset – безкоштовний інсталятор [14]. Він заснований на XML, що означає, що замість інтерфейсу користувача ви можете використовувати один файл xml для створення інсталяційного пакета.

XML (Extensible Markup Language) – це мова розмітки, яка використовується для кодування документів у форматі. Він розроблений як гнучкий і адаптований, що робить його хорошим вибором для широкого спектру додатків, від обміну даними до веб-сервісів.

Однією з головних переваг XML є його розширюваність. На відміну від інших мов розмітки, наприклад HTML, які розроблені для певних цілей і мають обмежену гнучкість, XML можна адаптувати до широкого діапазону програм шляхом визначення спеціальних тегів і атрибутів, які відповідають потребам програми.

Для використання цього інструменту у VS потрібно додати розширення (extension) (рис. 2.5.1) [10].



Рис. 2.5.1. Розширення Wix Toolset

Далі потрібно створити проект(рис. 2.5.2).



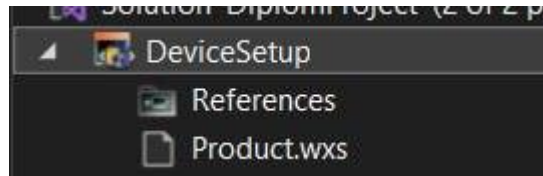


Рис. 2.5.2. Створення проекту для інсталяції застосунку

Відразу додаємо розширення для UI частини, для створення зручного інтерфейсу (рис. 2.5.3). Додавляємо Reference:

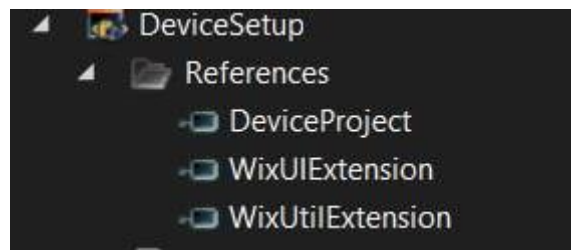


Рис. 2.5.3. Додавання references для інсталятора

Налаштовуємо інсталятор, щоб при побудуванні проекту (build the project) він автоматично перевіряв, які файли потрібно встановити для нашого WPF застосунку. Для цього відкриваємо .wixproj файл, це є основним файлом для інсталятора. Його можна відкрити у Notepad, але для кращого прочитання коду використовуємо застосунок Visual Studio Code (рис. 2.5.4).

```

DeviceSetup.wixproj X
C:\Users\yshem> source > repos > WpfApplication > DiplomProject > DeviceSetup > DeviceSetup.wixproj
1  <?xml version="1.0" encoding="utf-8"?>
2  <Project ToolsVersion="4.0" DefaultTargets="Build" InitialTargets="EnsureWixToolsetInstalled" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
3    <PropertyGroup>
4      <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
5      <Platform Condition=" '$(Platform)' == '' ">x86</Platform>
6      <ProductVersion>3.10</ProductVersion>
7      <ProjectGuid>78812e5b-09aa-48f4-869b-be3035b0efa6</ProjectGuid>
8      <SchemaVersion>2.0</SchemaVersion>
9      <OutputName>DeviceSetup</OutputName>
10     <OutputType>Package</OutputType>
11   </PropertyGroup>
12   <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|x86' ">
13     <OutputPath>bin\$(Configuration)\</OutputPath>
14     <IntermediateOutputPath>obj\$(Configuration)\</IntermediateOutputPath>
15     <DefineConstants>Debug</DefineConstants>
16   </PropertyGroup>

```

Рис. 2.5.4. Основний файл для налаштування інсталятора

Налаштування в цьому файлі робимо наступні:

1. Об'являємо змінну (constant) із значенням шляху до потрібних файлів

```

</PropertyGroup>
<PropertyGroup>
  <DefineConstants>HarvestPath=..\DeviceProject\bin\Debug\net6.0-windows</DefineConstants>
</PropertyGroup>

```

2. Далі, створюємо компонент, де вказуємо шлях, назву файла (який буде зберігати всю інформацію про те, що має встановити інсталацію) та прописуємо нашу змінну, яку створили на попередньому шляху.

```

-->
<Target Name="BeforeBuild">
  <HeatDirectory
    Directory="..\DeviceProject\bin\Debug\net6.0-windows"
    PreprocessorVariable="var.HarvestPath" OutputFile="HeatGeneratedFileList.wxs"
    ComponentGroupName="HeatGenerated" DirectoryRefId="INSTALLFOLDER"
    AutogenerateGuids="true" ToolPath="$(WixToolPath)" SuppressFragments="true"
    SuppressRegistry="true" SuppressRootDirectory="true" RunAsSeparateProcess="$(RunWixToolsOutOfProc)" />
</Target>

```

Тепер можемо побудувати проект (build the project), і бачимо з'явився додатковий файл (рис. 2.5.5).

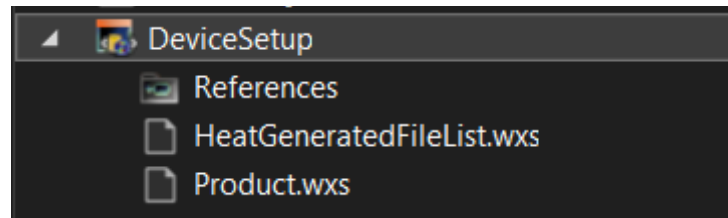


Рис. 2.5.5. Вигляд проекту

Що містить тепер дані про наш WPF застосунок (рис. 2.5.6):

```

<?xml version="1.0" encoding="utf-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
  <Fragment>
    <DirectoryRef Id="INSTALLFOLDER">
      <Component Id="cmpD81A82E7E0F9B2DC0CC1DAF713155282" Guid="*">
        <File Id="fil104E0D1FF161FF9A99EEEF48D7433AA4" KeyPath="yes" Source="$(var.HarvestPath)\DeviceProject.deps.json" />
      </Component>
      <Component Id="cmp4DAB86289E830FF622B3FBDF6881D70A" Guid="*">
        <File Id="fil18373D45FAC66604A34C25F792F19767" KeyPath="yes" Source="$(var.HarvestPath)\DeviceProject.dll" />
      </Component>
      <Component Id="cmp614A2AE936E1038FFA0CD36B7A736BC1" Guid="*">
        <File Id="filD8CC7BCBBEE556CA17C3A8A0EBF8C22" KeyPath="yes" Source="$(var.HarvestPath)\DeviceProject.exe" />
      </Component>
      <Component Id="cmp24712F823A66F7DF71BB54B21C5B2D01" Guid="*">
        <File Id="fil1830217014874A7DDDC924176E3837350" KeyPath="yes" Source="$(var.HarvestPath)\DeviceProject.pdb" />
      </Component>
      <Component Id="cmp8830C21EECED62CE8B528CC7026F7010" Guid="*">
        <File Id="fil2D7293B2D5AEECC63B179961F2840655" KeyPath="yes" Source="$(var.HarvestPath)\DeviceProject.runtimeconfig.json" />
      </Component>
      <Component Id="cmp9CD70704A7E005E61D381624B3C38308" Guid="*">
        <File Id="filE0BDBF419D818607A8C37F812BA88552" KeyPath="yes" Source="$(var.HarvestPath)\DiplomProject.deps.json" />
      </Component>
      <Component Id="cmp7D9A468A769CE1DEAB1B374276EA0773" Guid="*">
        <File Id="fil4ED7D350F8096F4C5FD2A3D8A22789A1" KeyPath="yes" Source="$(var.HarvestPath)\DiplomProject.dll" />
      </Component>
      <Component Id="cmp8837EFD10F44A66F1432521A14CE767E" Guid="*">
        <File Id="fil7A860DCF86A4A280E6FA3DC9DF54A123" KeyPath="yes" Source="$(var.HarvestPath)\DiplomProject.exe" />
      </Component>
    </DirectoryRef>
  </Fragment>

```

Рис. 2.5.6. Файл із інформацією про всі dependencies (залежності)

Інтерфейс користувача до інсталяційного файлу WIX відіграє важливу роль у створенні легкого та інтуїтивно зрозумілого способу встановлення додатків для Windows. Завдяки можливостям гнучкості, які надає набір інструментів WIX, розробники можуть створювати інтерфейси користувача, які відповідають конкретним потребам їхніх програм і користувачів.

Тому наступний етап є створення інтерфейсу для інсталятора. Уривок коду для налаштування інтерфейсу:

```
<Property Id="WIXUI_INSTALLDIR" Value="INSTALLFOLDER" />
<UI Id="WixUI_InstallDir">
```

```
  <TextStyle Id="WixUI_Font_Normal" FaceName="Tahoma" Size="8" />
  <TextStyle Id="WixUI_Font_Input" FaceName="Tahoma" Size="9" />
  <TextStyle Id="WixUI_Font_Bigger" FaceName="Tahoma" Size="12" />
  <TextStyle Id="WixUI_Font_Title" FaceName="TahOWma" Size="9"
  Bold="yes" />
```

```
  <Property Id="DefaultUIFont" Value="WixUI_Font_Normal" />
  <Property Id="WixUI_Mode" Value="Minimal" />
```

```
  <DialogRef Id="ErrorDlg" />
  <DialogRef Id="FatalError" />
  <DialogRef Id="FilesInUse" />
  <DialogRef Id="PrepareDlg" />
  <DialogRef Id="ProgressDlg" />
  <DialogRef Id="ResumeDlg" />
  <DialogRef Id="UserExit" />
  <DialogRef Id="WelcomeDlg" />
```

```
  <Publish Dialog="WelcomeDlg" Control="Next" Event="NewDialog"
  Value="InstallDirDlg">1</Publish>
```

```
  <Publish Dialog="InstallDirDlg" Control="Back" Event="NewDialog"
  Value="WelcomeDlg">1</Publish>
```



```

<Publish Dialog="InstallDirDlg" Control="Next" Event="SetTargetPath"
Value="[WIXUI_INSTALLDIR]" Order="1">1</Publish>
<Publish Dialog="InstallDirDlg" Control="Next" Event="DoAction"
Value="WixUIValidatePath" Order="2">NOT
WIXUI_DONTVALIDATEPATH</Publish>
<Publish Dialog="InstallDirDlg" Control="Next" Event="SpawnDialog"
Value="InvalidDirDlg" Order="3">
<![CDATA[NOT WIXUI_DONTVALIDATEPATH AND
WIXUI_INSTALLDIR_VALID<>"1"]]>
</Publish>
<Publish Dialog="InstallDirDlg" Control="Next" Event="NewDialog"
Value="VerifyReadyDlg" Order="4">
WIXUI_DONTVALIDATEPATH OR WIXUI_INSTALLDIR_VALID="1"
</Publish>
<Publish Dialog="InstallDirDlg" Control="ChangeFolder"
Property="_BrowseProperty" Value="[WIXUI_INSTALLDIR]"
Order="1">1</Publish>
<Publish Dialog="InstallDirDlg" Control="ChangeFolder"
Event="SpawnDialog" Value="BrowseDlg" Order="2">1</Publish>

<Publish Dialog="VerifyReadyDlg" Control="Back" Event="NewDialog"
Value="InstallDirDlg" Order="1">NOT Installed</Publish>

<Publish Dialog="ExitDialog" Control="Finish" Event="EndDialog"
Value="Return" Order="999">1</Publish>
</UI>
<UIRef Id="WixUI_Common" />

```

Зберігаємо та знову робимо Build the project. Далі в папці bin/Debug можемо знайти наш .MSI file (Microsoft Software Installer) (рис. 2.5.7).

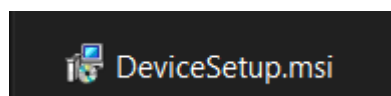
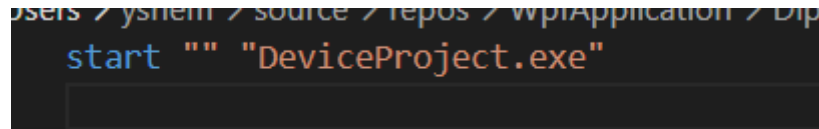


Рис. 2.5.7. Вигляд файлу .msi для встановлення застосунку

Можна додати додаткову функцію для зручного запуску .exe файлу встановленого застосунку, що ж зробимо деякі налаштування. Додаємо .bat (Windows Batch File) файл за допомогою якого, після інсталяції застосунку, наш файл буде автоматично запускатися (рис. 2.5.8).



```
users / yshem / source / tepos / wpiApplication / Dip
start "" "DeviceProject.exe"
```

Рис. 2.5.8 Створення .Bat файлу для автозапуску застосунку

Далі потрібно налаштувати в кодї. Додаємо дію для запускання цього файлу, який автоматично запусить .exe після встановлення всіх файлів.

```
<CustomAction                                Id="LaunchApplication"
  ExeCommand="[INSTALLFOLDER]AutoRunExe.bat"
  Directory="INSTALLFOLDER" Execute="deferred" Return="asyncWait" />

<InstallExecuteSequence>
  <Custom      Action="LaunchApplication"      After="InstallFiles">NOT
Installed</Custom>
</InstallExecuteSequence>
```

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ НАСТІЛЬНОГО ЗАСТОСУНКУ

3.1. Архітектура даного застосунку

Архітектурні патерни є важливою частиною створення програм Windows Presentation Foundation (WPF) на C#. Архітектурні патерни, такі як Model-View-ViewModel (MVVM) і Model-View-Controller (MVC), допомагають розділити компоненти різних частин програми [1, 129]. Це робить код легшим для розуміння та підтримки, а також зменшує ймовірність допущення великих помилок.

Сприяючи узгодженості в тому, як розробляється програма, архітектурні патерни допомагають переконатися, що код добре організований і простий для розуміння. Це може допомогти зменшити ризик помилок та спростити для нових розробників ознайомлення з базою коду.

Model-View-ViewModel (MVVM) – це шаблон проектування, який використовується в розробці програмного забезпечення та походить від корпорації Майкрософт, яка займається саме дизайном моделі презентації. Він базується на моделі Model-view-controller (MVC) і розроблений для сучасних платформ розробки користувачів (WPF), де UX-дизайнер має інші вимоги, ніж більш «традиційні». Шаблон використовує основні конструктивні особливості WPF, дозволяє легко тестувати функціональність процесів і допомагає розробникам та дизайнерам працювати разом над проблемою технічної слабкості [1, 132].

Для досягнення цього патерну нами було реалізовано 3 компоненти [15]:

- **View:** представлення визначено в XAML і не повинно мати жодної логіки в коді. Він прив'язується до моделі перегляду лише за допомогою зв'язування даних (data binding).

- **Model:** Модель відповідає за надання даних у спосіб, який легко використовуватиме WPF. Він повинен відповідним чином реалізувати `INotifyPropertyChanged` та/або `INotifyCollectionChanged`.

- **ViewModel**: Модель перегляду в додатку або абстракція перегляду. Він надає дані, пов'язані з представленням (View), і показує поведінку для представлень, зазвичай за допомогою команд.

Перейдемо до реалізації цього шаблону.

Спершу нами реалізовано `INotifyPropertyChanged` (рис. 3.1.1). Для цього ми створили клас, який наслідує інтерфейс з усіма потрібними властивостями та методами. Далі прописали і реалізували:

```

1 reference
public abstract class BindableBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    2 references
    protected void OnPropertyChanged(string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    4 references
    protected virtual bool SetProperty<T>(ref T storage, T value, [CallerMemberName] string propertyName = null)
    {
        if (EqualityComparer<T>.Default.Equals(storage, value)) return false;

        storage = value;
        OnPropertyChanged(propertyName);

        return true;
    }

    0 references
    protected virtual bool SetProperty<T>(ref T storage, T value, Action onChanged, [CallerMemberName] string propertyName = null)
    {
        if (EqualityComparer<T>.Default.Equals(storage, value)) return false;

        storage = value;
        onChanged?.Invoke();
        OnPropertyChanged(propertyName);

        return true;
    }
}

```

Рис. 3.1.1. Реалізація інтерфейсу `INotifyPropertyChanged`

Далі ми реалізували інтерфейс `ICommand`, який ми використовували для подій при натисканні на кнопки (рис. 3.1.2).

```

3 references
public class Command : ICommand
{
    private readonly Action<object> _action;

    2 references
    public Command(Action<object> action)
    {
        _action = action;
    }

    0 references
    public void Execute(object parameter)
    {
        _action(parameter);
    }

    0 references
    public bool CanExecute(object parameter)
    {
        return true;
    }

    public event EventHandler CanExecuteChanged
    {
        add { }
        remove { }
    }
}

```

Рис. 3.1.2. Реалізація інтерфейсу ICommand

Далі використали класи безпосередньо для елементів. Для кнопки прописали команду, яка буде реагувати на подію OnClick() (рис. 3.1.3).

```

border.BorderBrush=gray BorderThickness=3
<Button Command="{Binding ControlRecording}"
        BorderBrush="Transparent" BorderThic

```

Рис. 3.1.3. Оголошено команду для кнопки

Наступне, у MainViewModel класі, ми створили властивість, і у конструкторі присвоїли значення (рис.3.1.4)

```

1 reference
public ICommand ControlRecording
{
    get => _controlRecording;
}

```

```

_controlRecording = new Command(_ => PerformControlRecording());

```

```

1 reference
private async void PerformControlRecording()
{
    if (IsRecordingInProgress)
    {
        IsEnabledRecordingControlButton = false;
        await _videoRecorder.StopRecordingAsync();
        IsEnabledRecordingControlButton = true;
        RecordingControlButtonText = "Record";
    }
    else
    {
        IsEnabledRecordingControlButton = false;
        await _videoRecorder.StartRecordingAsync();
        IsEnabledRecordingControlButton = true;
        RecordingControlButtonText = "Stop";
    }
    _isRecordingInProgress = _videoRecorder.RecordingInProgress;
}

```

Рис. 3.1.4. Уривок коду для обробки події OnClick()

Наступним кроком реалізували `INotifyPropertyChanged` в `MainViewModel`. Це потрібно для того, щоб ми могли контролювати, якщо властивість має зміни.

Отже, при створенні властивості ми встановили значення для приватного поля, і тим саме могли контролювати чи значення було зміненим. Наприклад (рис. 3.1.5):

```

1 reference
public bool IsRecordingInProgress
{
    get => _isRecordingInProgress;
    private set => SetProperty(ref _isRecordingInProgress, value);
}

```

Рис. 3.1.5. Створення властивості

Також ми при встановленні значення викликами метод, який виконував якусь дію (рис. 3.1.6).

```

1 reference
public bool IsRecordingInProgress
{
    get => _isRecordingInProgress;
    private set => SetProperty(ref _isRecordingInProgress, value, DoSomething);
}

1 reference
private void DoSomething()
{
    // Do something when the value will be changed
}

```

Рис. 3.1.6. Створення властивості з можливістю виконання певної дії при зміні значення

Це все що було потрібно для реалізації MVVM патерну у WPF застосунку, що є важливим аспектом при будіванні архітектури проекту. Адже було важливо у кодї розділяти UI частину від бізнес логіки, для того щоб цей застосунок в майбутньому розробники могли модифікувати без додаткових зусиль.

3.2. Реєстрація та авторизація

Реєстрація та авторизація є важливими аспектами створення безпечних і надійних програм WPF (Windows Presentation Foundation) та забезпечити безперебійну роботу як для нових, так і для існуючих користувачів.

Спершу ми створили базу даних, щоб зберігати користувачів. Для конфігурації використали MS SQL Server. Отже, спершу ми підключилися до серверу та створили нову database (рис. 3.2.1).

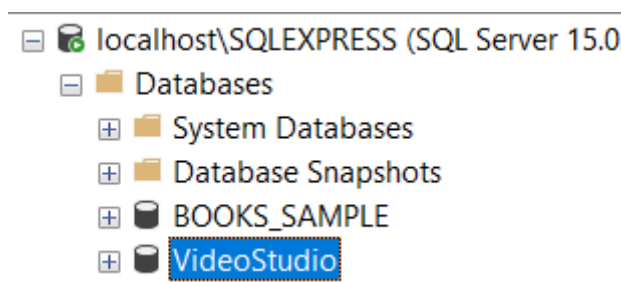


Рис. 3.2.1. Створення database

Далі створили таблицю `Користувачі` та добавили потрібні колонки (рис. 3.2.2).

Column Name	Data Type	Allow Nulls
FirstName	nvarchar(50)	<input type="checkbox"/>
LastName	nvarchar(50)	<input type="checkbox"/>
Email	nvarchar(50)	<input type="checkbox"/>
Password	nvarchar(MAX)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Рис. 3.2.2. Створення таблиці та полів

Форма реєстрації користувача зазвичай надається для збору необхідної інформації від нових користувачів, такої як ім'я користувача, пароль, адреса електронної пошти та будь-які додаткові дані профілю. У WPF ця форма розроблена з використанням XAML і може містити різні елементи інтерфейсу користувача, такі як текстові поля, мітки та кнопки.

Під час реєстрації важливо перевірити введені користувачем дані, щоб переконатися, що вони відповідають певним критеріям. Це включає перевірку, чи ім'я користувача чи адресу електронної пошти вже зайнято, перевірку надійності пароля та застосування будь-яких певних правил чи обмежень. WPF надає механізми перевірки даних, включаючи вбудовані правила перевірки або спеціальну логіку перевірки.

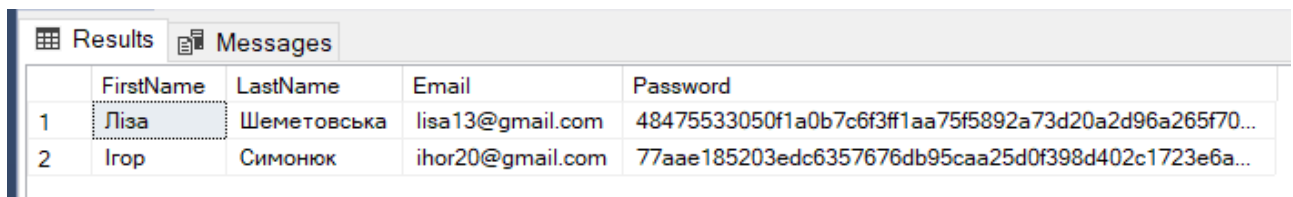
Також важливо обробляти та повідомляти про будь-які помилки, які можуть виникнути, як-от некоректне значення або проблеми з підключенням до бази даних або збої доставки електронної пошти. Програми WPF можуть відображати повідомлення про помилки або надавати відгуки, щоб направляти користувачів через процес реєстрації.

Щоб реалізувати реєстрацію для користувача, ми розпочали з XAML сторінки. Створили Page з потрібними полями для вводу, добавили прив'язку даних та відповідні стилі(Додаток Б).

Далі налаштували логіку підключення до бази даних та реєстрації користувача:

```
public bool RegisterUser(DbUser user)
{
    try
    {
        string hashedPassword = HashPassword(user.Password!);
        using SqlConnection connection = new(CONNECTION_STRING);
        string query = "INSERT INTO Користувачі (FirstName, LastName,
Email, Password) VALUES (@FirstName, @LastName, @Email, @Password)";
        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@FirstName", user.FirstName);
        command.Parameters.AddWithValue("@LastName", user.LastName);
        command.Parameters.AddWithValue("@Email", user.Email);
        command.Parameters.AddWithValue("@Password", hashedPassword);
        connection.Open();
        command.CommandType = CommandType.Text;
        command.ExecuteNonQuery();
        return true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return false;
    }
}
```

Добавили кілька користувачів, і в базі даних можна бачити 2 записи (рис. 3.2.3).



The screenshot shows a 'Results' window with a table containing two rows of user data. The columns are 'FirstName', 'LastName', 'Email', and 'Password'. The first row has 'Ліза' as the first name and 'Шеметовська' as the last name. The second row has 'Ігор' as the first name and 'Симонюк' as the last name. The email addresses are 'lisa13@gmail.com' and 'ihor20@gmail.com' respectively. The passwords are long alphanumeric strings.

	FirstName	LastName	Email	Password
1	Ліза	Шеметовська	lisa13@gmail.com	48475533050f1a0b7c6f3ff1aa75f5892a73d20a2d96a265f70...
2	Ігор	Симонюк	ihor20@gmail.com	77aae185203edc6357676db95caa25d0f398d402c1723e6a...

Рис. 3.2.3. Записи користувачів у БД

Далі відбувається процес авторизації, що є перевіркою особи користувача. Зазвичай це включає перевірку комбінації імені користувача та пароля зі збереженим набором облікових даних. WPF надає різні механізми аутентифікації, включаючи автентифікацію на основі форм, автентифікацію Windows і спеціальну автентифікацію Це гарантує, що лише авторизовані користувачі можуть виконувати обмежені операції або переглядати конфіденційні дані.

Механізми контролю доступу гарантують, що лише авторизовані користувачі можуть взаємодіяти з певними елементами інтерфейсу користувача або виконувати певні операції. WPF пропонує такі елементи керування, як кнопки, меню або стовпці сітки даних, які можна увімкнути або вимкнути залежно від рівня авторизації користувача. Контроль доступу також можна застосувати програмним шляхом у файлах із кодом.

Спочатку ми створили візуалізацію для цієї сторінки. Коли користувач вводить коректні дані, нам було потрібно також витягнути 2 поля – Ім'я та Прізвище, яке ми використовували для подальшої роботи. Для цього ми використали команду SqlDataReader, що надає спосіб читання прямого потоку рядків із бази даних SQL Server.

```
public User? LoginUser(string? username, string? password)
{
    User user = new();
    using SqlConnection connection = new
SqlConnection(CONNECTION_STRING);
    var hashedPassword = HashPassword(password!);
    string query = "SELECT FIRSTNAME, LASTNAME FROM Користувачі
WHERE Email=@Email AND Password=@Password";
    SqlCommand command = new SqlCommand(query, connection);
    command.Parameters.AddWithValue("@Email", username);
    command.Parameters.AddWithValue("@Password", hashedPassword);

    connection.Open();
```

```

using SqlDataReader reader = command.ExecuteReader();

if (reader.Read())
{
    int nameIndex = reader.GetOrdinal("FirstName");
    string retrievedName = reader.GetString(nameIndex);
    int lastnameIndex = reader.GetOrdinal("Lastname");
    string retrievedLastnameName = reader.GetString(lastnameIndex);
    user = new User()
    {
        FirstName = retrievedName,
        Lastname = retrievedLastnameName
    };
}
else
{
    MessageBox.Show("Invalid username or password!");
    return null;
}
return user;
}

```

І реалізували функцію HashPassword, яка використовується для реєстрації та авторизації:

```

private string HashPassword(string password)
{
    using (SHA256 sha256 = SHA256.Create())
    {
        byte[] hashedBytes =
sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < hashedBytes.Length; i++)
        {
            stringBuilder.Append(hashedBytes[i].ToString("x2"));
        }
        return stringBuilder.ToString();
    }
}

```

```

    }
}

```

3.3. Програмна реалізація навігації

У нашому застосунку ми використали елемент керування Frame, що дозволяє здійснювати навігацію в межах програми. Це частина навігаційної системи і зазвичай використовується для розміщення та відображення різних сторінок або переглядів. Елемент керування Frame служить контейнером для розміщення сторінок у програмі WPF. Він забезпечив область вмісту, де можна завантажувати та відображати сторінки у нашому застосунку.

Для впровадження цього компонента у наш застосунок, ми використали елемент в XAML та метод NavigationService в коді.

В MainWindow.xaml створили елемент та у MainWindow.xaml.cs добавили методи для навігації до різних вікон чи сторінок.

```
<Frame x:Name="mainFrame" NavigationUIVisibility="Hidden" />
```

```

public void NavigateToLoginPage()
{
    mainFrame.Navigate(new LoginPage(this, _mainViewModel));
}
public void NavigateToRegistrationPage()
{
    mainFrame.Navigate(new RegistrationPage(this));
}
public void NavigateToVideoRecorderPage(string email)
{
    mainFrame.Navigate(new VideoRecorderPage(this, email,
    _mainViewModel));
}

```

І коли нам потрібно було перейти на іншу сторінку ми викликали ці методи, наприклад при натисканні на кнопку.

Також у програмі нами було реалізовано User Controls, щоб переключатися між ними, потрібно було додати певну логіку, а саме:

```
<Grid x:Name="GridMain" />
```

```
private void Logout_Click(object sender, RoutedEventArgs e)
{
    _mainWindow.NavigateToLoginPage();
}
```

```
private void RecordMenu_Click(object sender, RoutedEventArgs e)
{
    GridMain.Children.Clear();
    _userControl = new RecordUserControl(_mainViewModel);
    GridMain.Children.Add(_userControl);
}
```

```
private void LibraryMenu_Click(object sender, RoutedEventArgs e)
{
    GridMain.Children.Clear();
    _userControl = new LibraryUserControl(_mainViewModel);
    GridMain.Children.Add(_userControl);
}
```

3.4. Розробка логіки для запису відео різного виду

Загалом нами була реалізована сторінка для запису відеоконтенту, що містить три опції для користувача, а саме: запис відео за допомогою камери, запис екрану та знімок екрану (при використанні веб-камери). Ми розробили можливість зробити Знімок екрану, і він активується, коли розпочинається запис відео.

Ми розробили цю сторінку як UserControl та ми могли переключатися між сторінками, що є великою особливістю. Тобто, якщо йшов запис екрану, то ми могли переглянути записані відео, тому що весь процес є асинхронним.

Асинхронне програмування особливо корисне для програм з графічним інтерфейсом користувача, де блокування потоку виконання може призвести до

замерзання програми та погіршення враження користувача [4, 792]. За допомогою асинхронних операцій можна запускати важкі обчислення або довгі запити до бази даних без блокування головного потоку інтерфейсу. Багатопоточність може бути досягнута шляхом запуску багатьох асинхронних операцій, які виконуються паралельно та асинхронно.

Отже, для запису відео за допомогою веб камери використовувалася бібліотека EmguCV та NAudio, що ми аналізували та реалізували у розділі 2.

Відповідну реалізацію для запису відео з відеокамери можна переглянути у додатку В.

Проте при розробці проекту, було ми виявили проблему, а саме: WPF має обмеження для запису відео відразу зі звуком. Тому нами було проаналізовано багато інформації і згодом нами було вирішено це, розробивши конвертер для комбінування відео та аудіо файлів. Для комбінації файлів потрібно було викликати ffmpeg.exe та виконати відповідну команду. Це дозволило нам створити один вихідний файл (у нашому випадку .mp4), що комбінує записане відео та звук. Нижче наведений приклад конвертора з нашого застосунку:

```
private static readonly string ffmpegExePath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "ffmpeg.exe");
public static void CombineVideoAndAudio(string video_path, string
audio_path, string outputVideoName)
{
    string arguments = $"-i \"{video_path}\" -i \"{audio_path}\" -c:v copy -c:a
aac \"{outputVideoName}\"";
    ProcessStartInfo processStartInfo = new()
    {
        FileName = ffmpegExePath,
        Arguments = arguments,
        UseShellExecute = false,
        RedirectStandardError = true,
        RedirectStandardOutput = true,
        CreateNoWindow = true
    };
    Process ffmpegProcess = new() { StartInfo = processStartInfo };
}
```

```

ffmpegProcess.Start();
ffmpegProcess.BeginOutputReadLine();
ffmpegProcess.BeginErrorReadLine();
ffmpegProcess.WaitForExit();
File.Delete(video_path);
File.Delete(audio_path);
ffmpegProcess.Dispose();
}

```

Коли відбувся запис відео за допомогою веб-камери, то можна було зробити скріншот відповідної картинки. Для цього нашій застосунок прочитував відповідний актуальний capture та ми створювали файл з картинкою.

Ось наведений приклад з нашого коду:

```

public void TakeScreenshot()
{
    using var frame = new Mat();
    capture.Read(frame);

    if (!frame.IsEmpty)
    {
        Bitmap bitmap = frame.ToBitmap();
        BitmapImage image = ConvertBitmapToBitmapImage(bitmap);

        // Save the image to a file
        var outputFilePath =
Path.Combine(DeviceProject.Properties.Settings.Default.PathMediaFolder,
$" {Guid.NewGuid()}_image.jpg");
        SaveBitmapImageToFile(image, outputFilePath);

        bitmap.Dispose();
    }
}

private void SaveBitmapImageToFile(BitmapImage bitmapImage, string filePath)
{

```

```

using var fileStream = new FileStream(filePath, FileMode.Create);
BitmapEncoder encoder = new JpegBitmapEncoder();
encoder.Frames.Add(BitmapFrame.Create(bitmapImage));
encoder.Save(fileStream);

}

```

І остання опція на цій сторінці була – запис екрану. Для цього нам потрібно було спершу визначити, який звук буде записаний. Нами було представлено два варіанти: використовувати вбудований/підключений мікрофон або звуковий вихід комп'ютера.

Нижче наведений код для старту запису екрану з нашого коду:

```

public void StartRecording(bool isMicSource)
{
    _isMicSource = isMicSource;
    screenCapture = new ScreenCapture();
    screenCapture.StartCapture();
    IsRecording = true;
    waveFilePath = FileHelpers.GetWaveFilePath();
    if (isMicSource)
    {
        waveSource = new WaveInEvent();
        waveSource.WaveFormat = new WaveFormat(44100, 16, 2);
        waveFileWriter = new WaveFileWriter(waveFilePath,
waveSource.WaveFormat);
        waveSource.DataAvailable += RecordAvailableDataOfWaveSource!;
        waveSource.StartRecording();
    }
    else
    {
        capture = new WasapiLoopbackCapture();
        capture.WaveFormat = new WaveFormat(44100, 16, 2);
        waveFileWriter = new WaveFileWriter(waveFilePath, capture.WaveFormat);
        capture.DataAvailable += RecordAvailableDataOfWaveSourceDesktop!;
        capture.StartRecording();
    }
}

```



```

    }

    timer.Start();
}

```

Та відповідна функція для записування відео у файл. Ми використовували `DispatcherTimer` для роботи запису екрану. Знімок екрану відбувався в певний проміжок часу та записувався у файл кожного разу, і в кінцевому результаті ми отримали вихідний файл.

```

private void Timer_Tick(object sender, EventArgs e)
{
    if (IsRecording)
    {
        screenCapture.CaptureFrame();
        byte[] frameData = screenCapture.GetFrameBytes();
        using var stream = new FileStream(videoFilePath, FileMode.Append);
        stream.Write(frameData, 0, frameData.Length);
    }
}

```

Нами було зроблено додаткове вікно, де користувач міг вибрати налаштування, за допомогою яких відбувалися подальші дії. Ми зберігали налаштування у файл `Settings` (рис. 3.4.1), що давало можливість нам використовувати налаштування у будь якому файлі відповідного `solution`.

Якщо ми вибирали `Application`, то ми встановлювали відразу значення для певного налаштування.

Якщо ми обирали `User`, то у коді могли встановити значення:

```

Properties.Settings.Default.IsDesktopSource = IsDesktopSource.ToString();
Properties.Settings.Default.Save();

```

	Name	Type	Scope	Value
	PathMediaFolder	string	User	
▶	IsDesktopSource	string	User	
*			Application	
			User	

Рис. 3.4.1. Налаштування файлу для зберігання даних

3.5. Перегляд відео та бібліотека записаних файлів

У програмі нами була реалізована так звана бібліотека із записаних файлів. Можна було переглянути записане відео/фото, також була можливість видалення із провідника та списку.

Натиснувши на Переглянути, відкривалося вікно, де можна було відтворювати вибраний файл.

Відео можна було переглядати, поставити на паузу та завершити перегляд [6, 479].

Нижче наведений код XAML, що показує реалізацію відтворення відео:

```
<MediaElement Name="mePlayer" Grid.Row="1" LoadedBehavior="Manual"
    Stretch="None" />
<StatusBar Grid.Row="2" x:Name="statusBar">
    <StatusBar.ItemsPanel>
        <ItemsPanelTemplate>
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="Auto" />
                </Grid.ColumnDefinitions>
            </Grid>
        </ItemsPanelTemplate>
    </StatusBar.ItemsPanel>
    <StatusBarItem>
        <TextBlock Name="lblProgressStatus">00:00:00</TextBlock>
```

```

</StatusBarItem>
<StatusBarItem Grid.Column="1" HorizontalContentAlignment="Stretch">
    <Slider Name="sliProgress"
        Thumb.DragStarted="sliProgress_DragStarted"
        Thumb.DragCompleted="sliProgress_DragCompleted"
        ValueChanged="sliProgress_ValueChanged" />
</StatusBarItem>
</StatusBar>

```

Нижче представлений приклад "Розпочати–Призупинити–Зупинити" команд для реалізації відтворення відеоконтенту з нашого коду:

```

private void Play_CanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = (mePlayer != null) && (mePlayer.Source != null);
}
private void Play_Executed(object sender, ExecutedRoutedEventArgs e)
{
    mePlayer.Play();
    mediaPlayerIsPlaying = true;
}
private void Pause_CanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = mediaPlayerIsPlaying;
}
private void Pause_Executed(object sender, ExecutedRoutedEventArgs e)
{
    mePlayer.Pause();
}
private void Stop_CanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = mediaPlayerIsPlaying;
}
private void Stop_Executed(object sender, ExecutedRoutedEventArgs e)
{
    mePlayer.Stop();
    mediaPlayerIsPlaying = false;
}

```

}

3.6. Сценарій роботи користувацького інтерфейсу

Для встановлення настільного застосунку для запису відеофайлів спочатку необхідно завантажити його на ПК. Для цього користувачу потрібно запустити файл інсталятора .msi (рис. 2.5.7). Після чого користувач буде бачити наступні вікна з інсталяцією, де йому потрібно вибрати шлях до встановлених файлів та натиснути Install (рис. 3.6.1– 3.6.2):

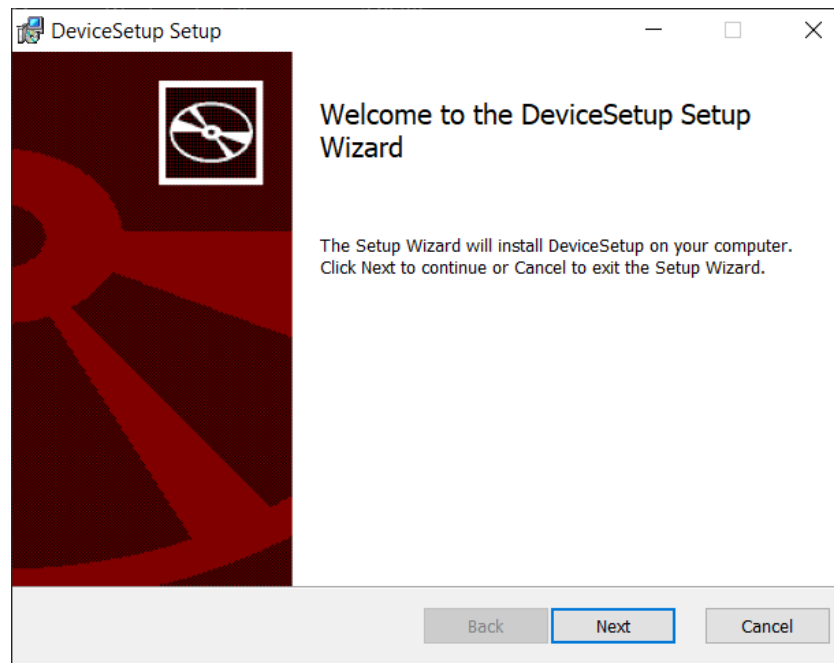


Рис. 3.6.1. Встановлення WPF застосунку

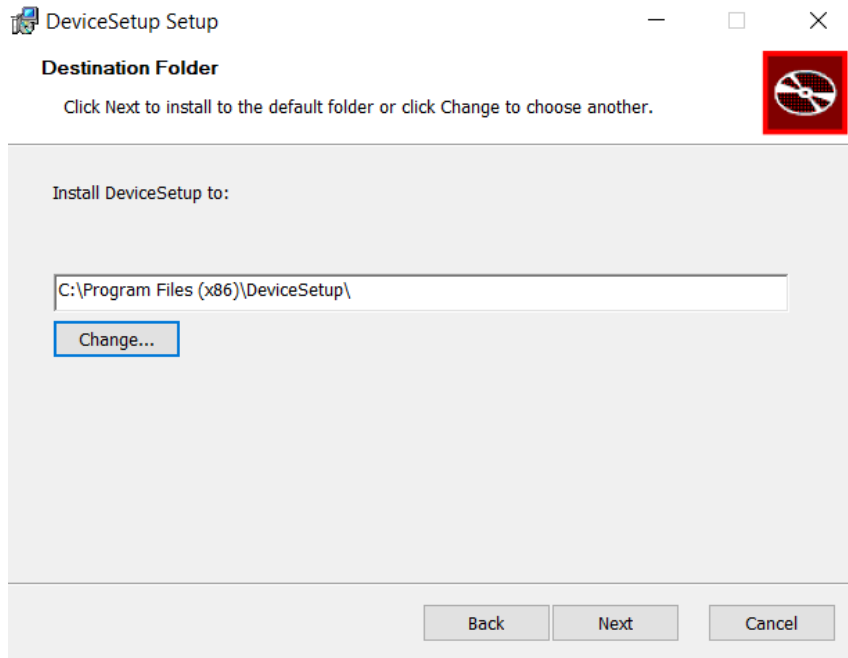


Рис. 3.6.2. Встановлення WPF застосунку

Після чого користувач може перейти до вибраного каталогу і переконатися, чи файли були завантажені, що є відповідно доказом для успішного завантаження на ПК (рис. 3.6.3).

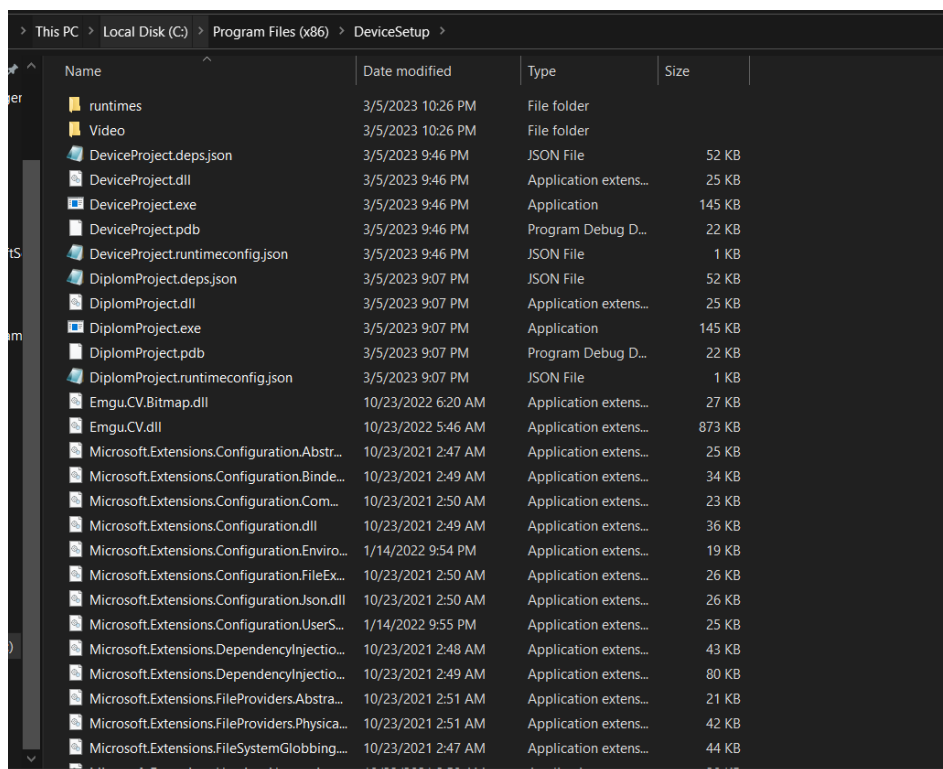


Рис. 3.6.3. Провідник із відкритим шляхом до файлів застосунку

При розробці даного інсталера, нами було налаштовано автоматичний запуск .exe файлу, тому після успішного завантаження користувач зможе бачити початкове вікно нашого додатку (рис. 3.6.4). Якщо користувач вже зареєстрував свій персональний аккаунт раніше, то він повинен вести свої облікові дані для подальшого використання даного застосунку.

Якщо ж користувач не є зареєстрованим, то потрібно натиснути на "Немає аккаунту?" кнопку, щоб перейти до сторінки реєстрації (Додаток Б), де він повинен ввести свої дані для реєстрації. Після успішної реєстрації, вікно автоматично перейде до сторінки авторизації знову, де користувачу потрібно повторно ввести облікові дані.

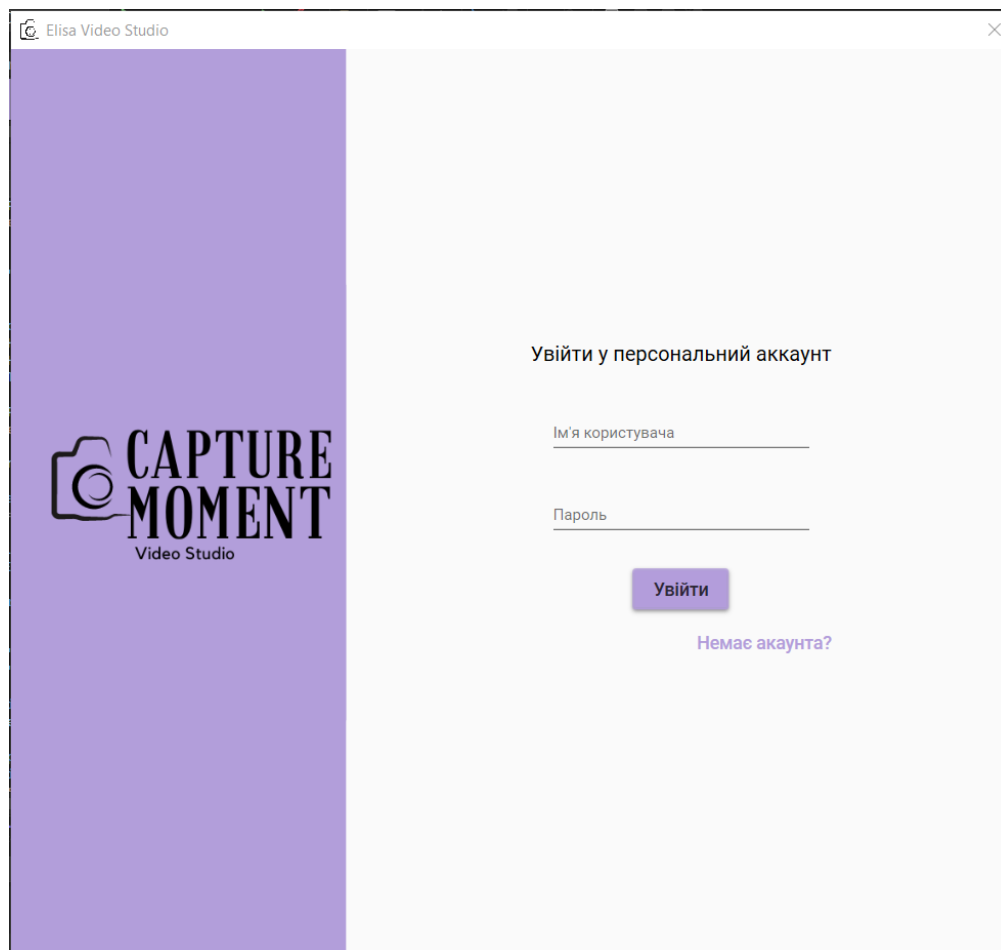


Рис. 3.6.4. Візуалізація вікна авторизації

Після успішної авторизації у верхньому куту header користувач матиме можливість бачити своє ім'я та прізвище(рис. 3.6.5).



Рис. 3.6.5. Візуалізація header після успішної авторизації

Наступним кроком користувачу необхідно зробити налаштування, що є важливим для запису відеоконтенту. Для цього слід натиснути на три крапки у правому верхньому куті та обрати "Налаштування" у меню (рис. 3.6.6).

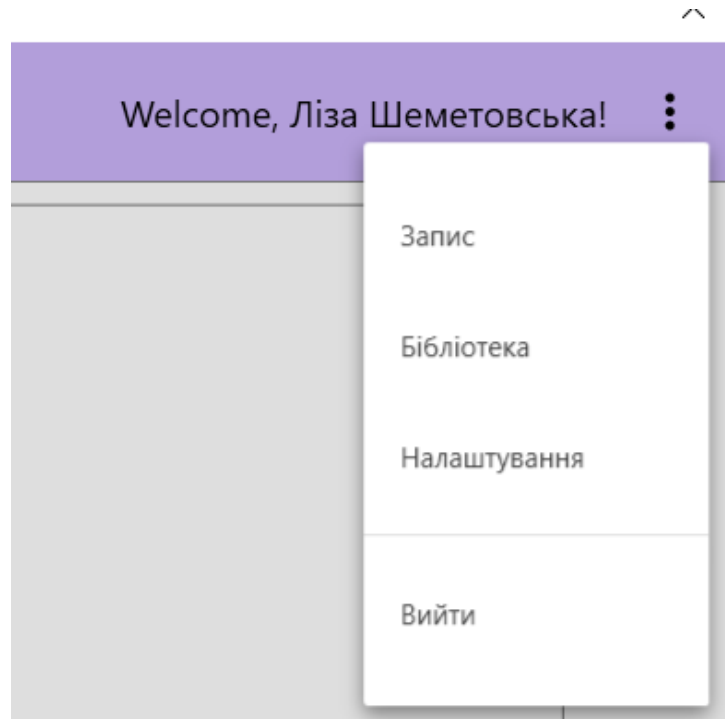
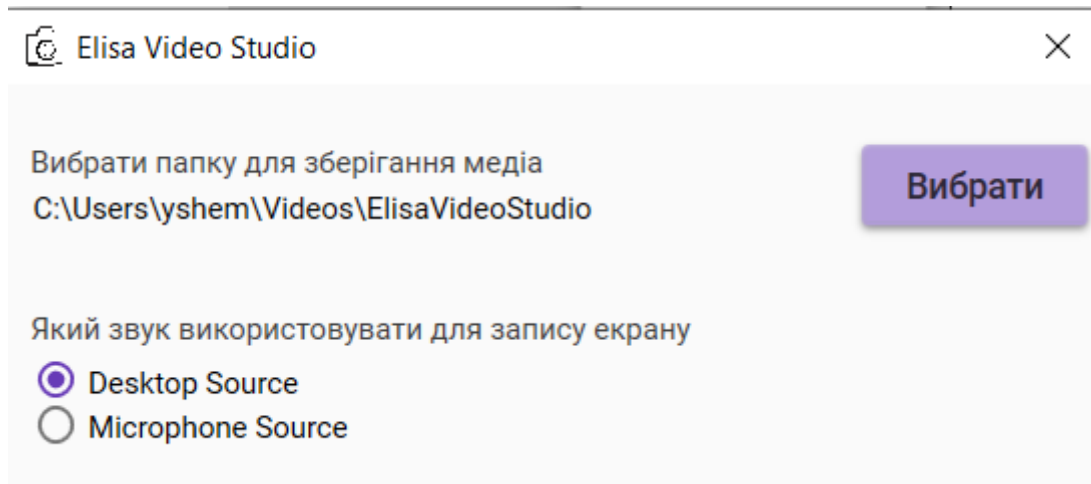


Рис. 3.6.6. Візуалізація меню у правому верхньому куті header

Натиснувши на "Налаштування", відкриється вікно (рис. 3.6.7). Тут нам потрібно налаштувати шлях, де будуть зберігатися відеофайли, та який звук буде використовуватися при записі екрану.



3.6.7. Візуалізація вікна Налаштування

Відповідно після налаштувань, користувач матиме можливість записувати відео двох видів (рис. 3.6.8), вибравши "Запис" в меню(рис. 3.6.6.).

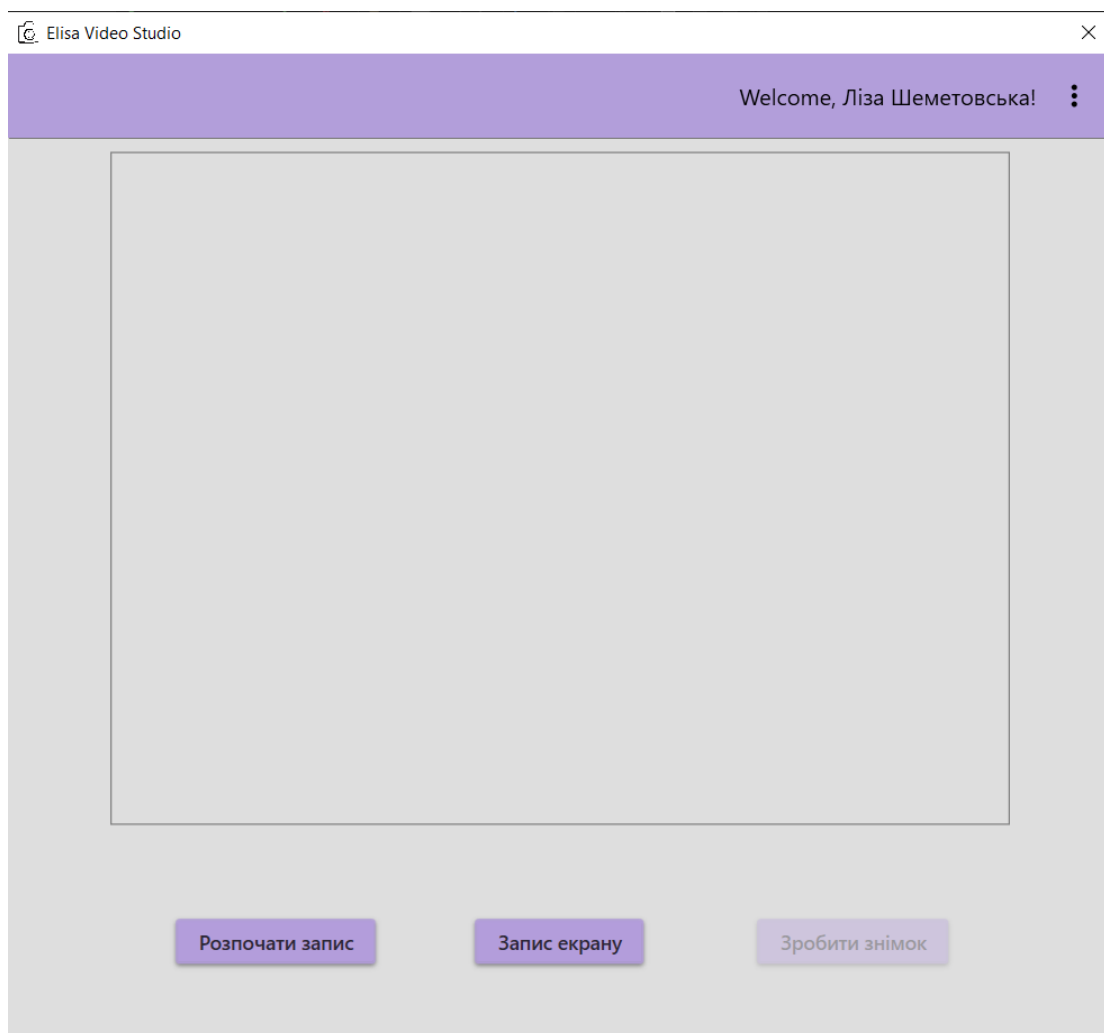


Рис. 3.6.8. Візуалізація вікна для запису відео

Також у меню користувач може вибрати пункт "Бібліотека" (рис. 3.6.9). Якщо відкрити, то він зможе побачити список записаних відео/фото. Також, натиснувши на три крапки є можливість видалити відео з провідника та переглянути відео або фото.

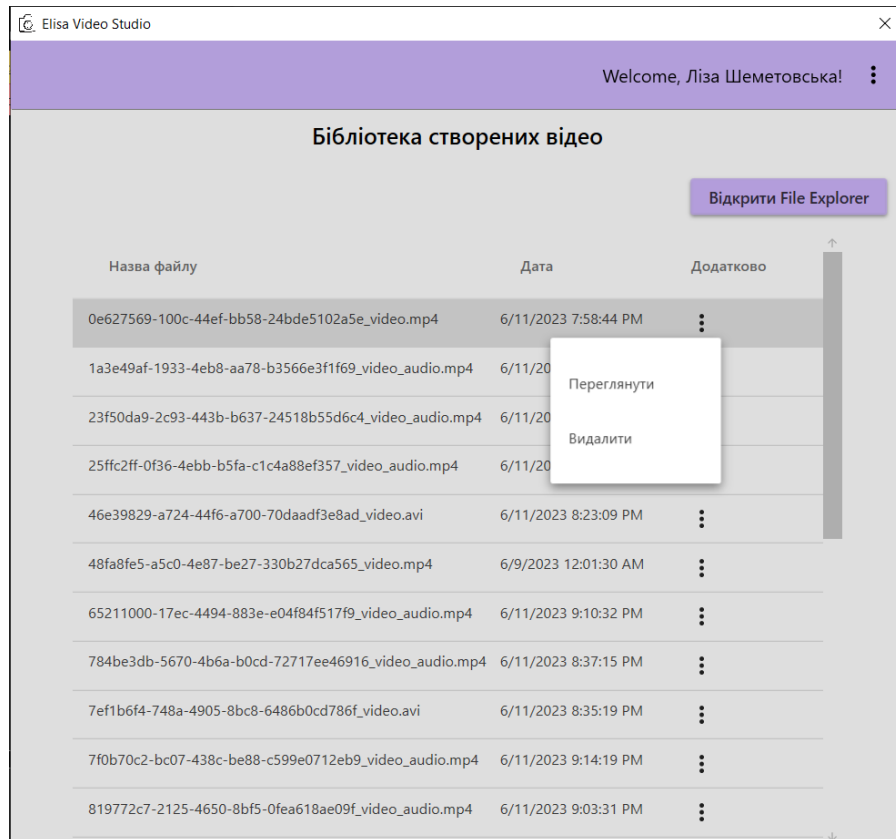


Рис. 3.6.9. Візуалізація сторінки Бібліотека

Вікно перегляду відео/фото містить три кнопки для відтворення/призупинення/зупинки відео, а також повзунок, якого можна вручну пересувати відповідно до потрібного часу (рис. 3.6.10).

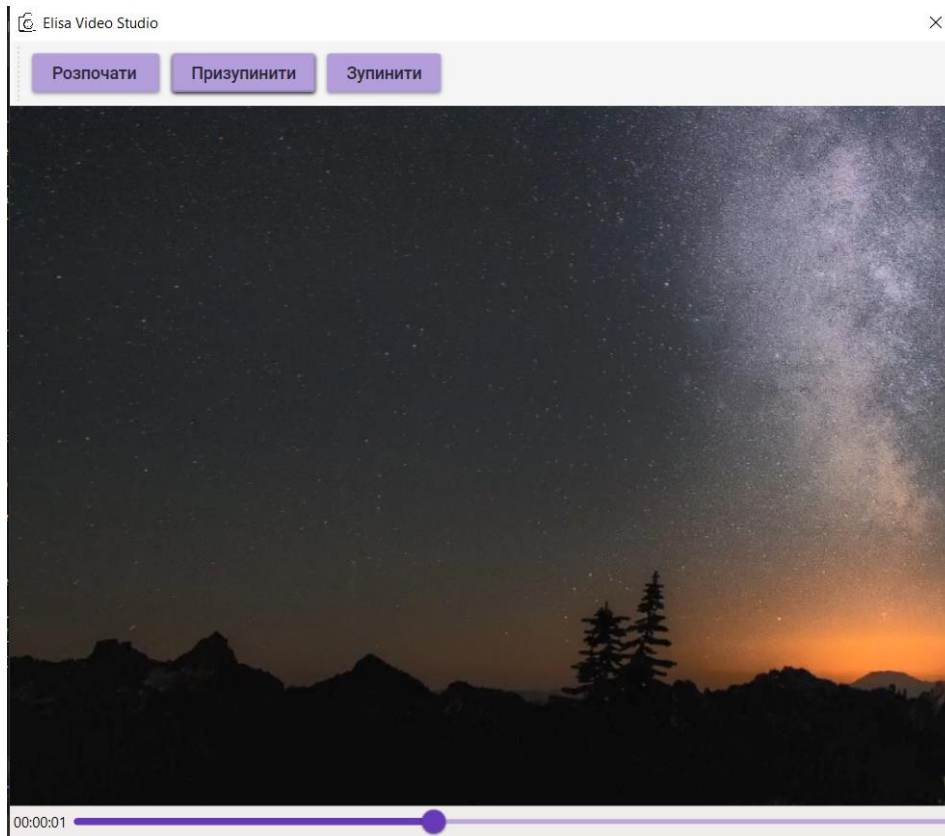


Рис. 3.6.10. Візуалізація вікна для перегляду відео/фото

Після завершення всіх дій користувачу необхідно вийти з облікового запису. Це можна зробити натиснувши опцію "Вийти" у меню (рис. 3.6.6).

ВИСНОВКИ

У кваліфікаційній роботі було проведено аналіз та дослідження, де розглянуто існуючі рішення для реалізації настільного застосунку, аналоги, переваги та недоліки настільного застосунку.

Додаток був створений за допомогою C# .NET, WPF, XAML, WIX extension та елементів SQL Server Management Studio (SSMS), що дозволяє працювати в автоматизованому режимі.

Було розроблено зручний та якісний інтерфейс для користувача, що є важливим аспектом при роботі з великими обсягами медіа даних. Усі операції відбуваються після реєстрації або авторизації користувача, що дозволяє одночасно надати обмежений доступ до даних великій кількості звичайних користувачів та захистити дані від несанкціонованого доступу.

Реалізація архітектури даного застосунку мала важливу роль для розробників в подальшому. Використання MVVM патерну було найоптимальнішим варіантом для розділення бізнес логіки від UI компонентів (представлення).

При розробці програмного продукту важливо було ознайомитися з різними бібліотеками та пакетами, які потрібно реалізовувати у коді. Було проведено аналіз всіх можливих бібліотек, та вибрано більше доступні та добре-підтримувані бібліотеки та пакети. Це обов'язково для розробки новітнього застосунку з можливістю покращення функціоналу.

Важливо було оцінити всі можливі проблеми, з якими можна зіткнутися при розробці. Однією з проблем було виявлено запис відео зі звуком, тому було витрачено більше часу для знаходження відповідної бібліотеки, що працює з аудіо, та розробки конверторів для комбінування файлів у один вихідний.

Перевірка повної валідності розробленого застосунку за допомогою однієї оцінки була складною через різні фактори. Однак було забезпечено надійність і сумісність системи програмного забезпечення, ретельно протестувавши її на різних операційних системах і широкому діапазоні типів комп'ютерів.

Майбутні перспективи програми включають потенційні вдосконалення та модернізацію, які можуть покращити її функціональність та досвід користувача. Одним із шляхів розвитку є можливість оновлювати або замінювати бібліотеки на основі конкретних потреб і уподобань користувачів. Ця гнучкість дозволяє адаптуватися до технологій, що розвиваються, і включати нові функції чи вдосконалення, коли вони стають доступними. Наприклад, корисними додатковими функціями є можливість записування відео із ефектами чи додавання ефекти для вибраних відео.

У майбутньому все залежить від відгуків користувачів, застосунок може продовжувати розвиватися та відповідати вимогам своїх користувачів, забезпечуючи свою актуальність і корисність, адже для розробки були використані новітні технології, що є легко підтримувані для модифікацій та додавання нового функціоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вейль А. Learn WPF MVVM – XAML, C# and the MVVM pattern: Lulu Press, 2016. 174 с.
2. Історія про створення відео. URL: <http://bit.ly/3JwmGoJ> (дата звернення 01.11.2022).
3. Натан А. Windows Presentation Foundation: Unleashed 1st Edition: Sams, 2007. 621 с.
4. Піхтер Дж. CLR via C#: Fourth Edition: Microsoft Press, 2012. 896 с.
5. Роденбург Дж. Code like a Pro in C#: Manning Publications, 2021. 385 с.
6. Соліс Д. Illustrated WPF (Expert's Voice in .Net): Apress, 2009. 507 с.
7. Формати відеофайлів. URL: <https://docs.fileformat.com/uk/video/> (дата звернення 05.11.2022).
8. EMGU CV Tutorial. URL: <https://emgu.com/wiki/index.php/Tutorial> (дата звернення 20.01.2023).
9. FFMpeg Documentation. URL: <https://bit.ly/3NhZCum> (дата звернення 13.02.2023) .
10. Generating a MSI Installer for a WPF Application. URL: <https://blog.jam-es.com/2019/12/generating-msi-installer-for-wpf.html> (дата звернення 06.03.2023).
11. MaterialDesign System Tutorial. URL: <http://materialdesigninxaml.net/> (дата звернення 13.03.2023).
12. NAudio Documentation. URL: <https://markheath.net/post/30-days-naudio-docs> (дата звернення 30.01.2023).
13. Windows Presentation Foundation документація. URL: <http://bit.ly/3ZzRieo> (дата звернення 07.11.2022).
14. WIX documentation. URL: <https://wixtoolset.org/docs/intro/> (дата звернення 06.03.2023).
15. WPF Apps With The Model-View-ViewModel Design Pattern. URL: <http://bit.ly/3T3MfjS> (дата звернення 07.03.2023).

ДОДАТКИ

Додаток А

**Приклад використання Attached Behavior для прив'язки значення
PasswordBox до змінної**

```
public static class PasswordBoxBehavior
{
    public static readonly DependencyProperty BoundPasswordProperty =
        DependencyProperty.RegisterAttached("BoundPassword", typeof(string),
            typeof>PasswordBoxBehavior, new PropertyMetadata(string.Empty,
                OnBoundPasswordChanged));

    public static string GetBoundPassword(DependencyObject obj)
    {
        return (string)obj.GetValue(BoundPasswordProperty);
    }

    public static void SetBoundPassword(DependencyObject obj, string value)
    {
        obj.SetValue(BoundPasswordProperty, value);
    }

    private static void OnBoundPasswordChanged(DependencyObject d,
        DependencyPropertyChangedEventArgs e)
    {
        var passwordBox = d as PasswordBox;

        if (passwordBox == null)
            return;

        passwordBox.PasswordChanged -= PasswordBox_PasswordChanged;

        if (e.NewValue != null)
        {
            passwordBox.Password = e.NewValue.ToString();
        }
        else
    }
```

```
{
    passwordBox.Password = string.Empty;
}

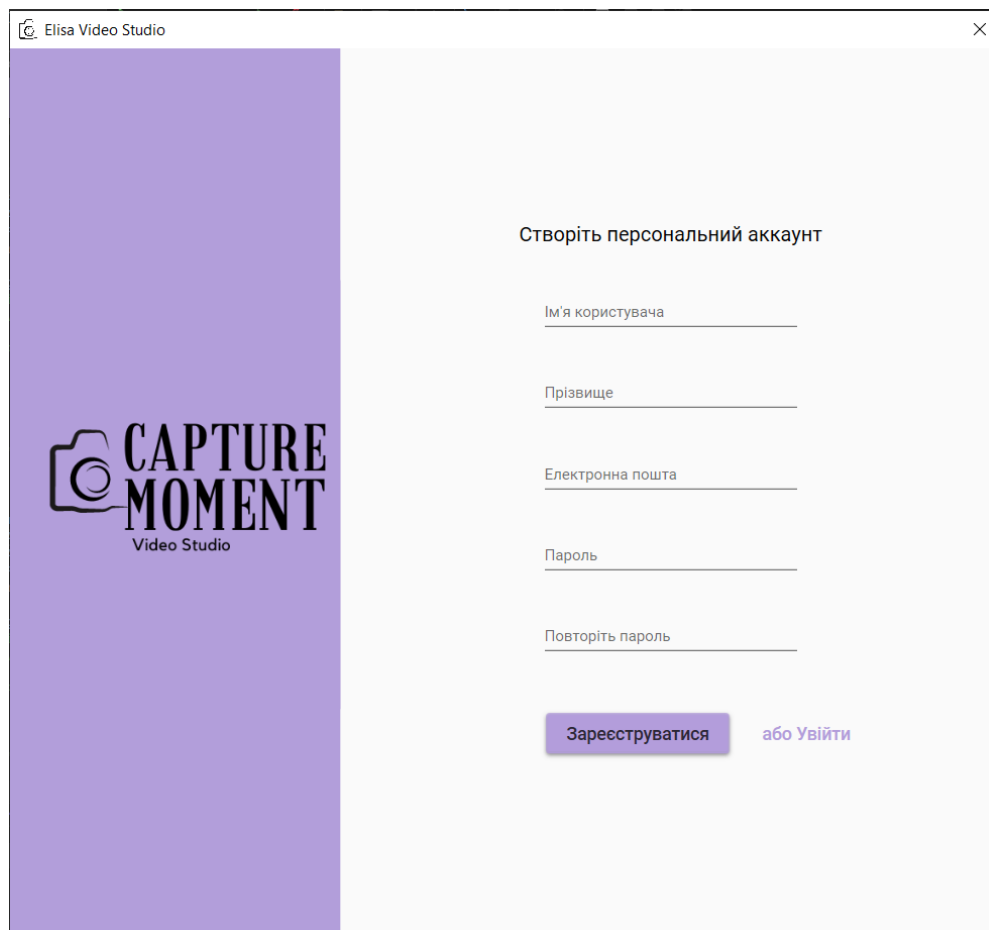
passwordBox.PasswordChanged += PasswordBox_PasswordChanged;
}

private static void PasswordBox_PasswordChanged(object sender,
RoutedEventArgs e)
{
    var passwordBox = sender as PasswordBox;

    if (passwordBox == null)
        return;

    SetBoundPassword(passwordBox, passwordBox.Password);
}
}
```

Створення XAML для сторінки реєстрації



```
<Page x:Class="DeviceProject.Views.Pages.RegistrationPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:local="clr-namespace:DeviceProject.Views"
  mc:Ignorable="d"
  xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
  Background="{DynamicResource MaterialDesignPaper}"
  FontFamily="{DynamicResource MaterialDesignFont}"
  d:DesignHeight="450" d:DesignWidth="800"
  Title="RegistrationPage">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="1*" />
```



```

    <ColumnDefinition Width="2*" />
</Grid.ColumnDefinitions>
<Grid Background="#B29EDA">
    <Grid.RowDefinitions>
        <RowDefinition />
    </Grid.RowDefinitions>
    <StackPanel VerticalAlignment="Center">
        <Image Source="/Resources/logo.png"
            Width="340"
            HorizontalAlignment="Center" />
    </StackPanel>
</Grid>
<StackPanel Grid.Column="1"
    VerticalAlignment="Center">
    <TextBlock Text="Створіть персональний аккаунт"
        HorizontalAlignment="Center"
        Style="{DynamicResource MaterialDesignTitleTextBlock}"
        Margin="0 0 0 20"
        FontSize="16" />
    <StackPanel HorizontalAlignment="Center"
        Margin="0 0 0 10">
        <TextBox Margin="10"
            Width="200"
            Name="textBoxName"
            materialDesign:HintAssist.Hint="Ім'я користувача"
            materialDesign:HintAssist.Foreground="#B29EDA"
            materialDesign:TextFieldAssist.UnderlineBrush="#B29EDA"
            Style="{StaticResource MaterialDesignFloatingHintTextBox}">
        <Binding Path="UserName"
            UpdateSourceTrigger="PropertyChanged">
        <Binding.ValidationRules>
            <domain:NotEmptyValidationRule
ValidatesOnTargetUpdated="True"
xmlns:domain="clr-namespace:Deviceject.Helpers" />
        </Binding.ValidationRules>
        </Binding>
    </TextBox>

```

```

</StackPanel>
<StackPanel HorizontalAlignment="Center"
    Margin="0 0 0 10">
    <TextBox Margin="10"
        Width="200"
        Name="textBoxLastName"
        materialDesign:HintAssist.Hint="Прізвище"
        materialDesign:HintAssist.Foreground="#B29EDA"
        materialDesign:TextFieldAssist.UnderlineBrush="#B29EDA"
        Style="{StaticResource MaterialDesignFloatingHintTextBox}">
    <Binding Path="UserName"
        UpdateSourceTrigger="PropertyChanged">
    <Binding.ValidationRules>
        <domain:NotEmptyValidationRule
ValidatesOnTargetUpdated="True"
                                xmlns:domain="clr-
namespace:DeviceProject.Helpers" />
    </Binding.ValidationRules>
    </Binding>
</TextBox>
</StackPanel>
<StackPanel HorizontalAlignment="Center"
    Margin="0 0 0 10">
    <TextBox Margin="10"
        Width="200"
        Name="textBoxEmail"
        materialDesign:HintAssist.Hint="Електронна пошта"
        materialDesign:HintAssist.Foreground="#B29EDA"
        materialDesign:TextFieldAssist.UnderlineBrush="#B29EDA"
        Style="{StaticResource MaterialDesignFloatingHintTextBox}">
    <Binding Path="UserEmail"
        UpdateSourceTrigger="PropertyChanged">
    <Binding.ValidationRules>
        <domain:NotEmptyValidationRule
ValidatesOnTargetUpdated="True"
                                xmlns:domain="clr-namespace:DeviceProject.Helpers" />
    </Binding.ValidationRules>

```

```

    </Binding>
  </TextBox>
</StackPanel>
<StackPanel HorizontalAlignment="Center"
  Margin="0 0 0 10">
  <PasswordBox Margin="10"
    Name="passwordBox"
    materialDesign:HintAssist.Hint="Пароль"
    materialDesign:HintAssist.Foreground="#B29EDA"
    materialDesign:TextFieldAssist.UnderlineBrush="#B29EDA"
    Width="200"
    materialDesign:PasswordBoxAssist.Password="{ Binding
Path=Password1Validated, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, ValidatesOnExceptions=True}"
    Style="{ StaticResource
MaterialDesignFloatingHintPasswordBox}" />
  </StackPanel>
<StackPanel HorizontalAlignment="Center"
  Margin="0 0 0 10">
  <PasswordBox Margin="10"
    Name="passwordBoxConfirm"
    materialDesign:HintAssist.Hint="Повторіть пароль"
    materialDesign:HintAssist.Foreground="#B29EDA"
    materialDesign:TextFieldAssist.UnderlineBrush="#B29EDA"
    Width="200"
    materialDesign:PasswordBoxAssist.Password="{ Binding
Path=Password1Validated, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged, ValidatesOnExceptions=True}"
    Style="{ StaticResource
MaterialDesignFloatingHintPasswordBox}" />
  <TextBlock Height="20"
    HorizontalAlignment="Left"
    Margin="67,0,0,0"
    x:Name="errorMessage"
    VerticalAlignment="Top"
    Width="247"
    OpacityMask="Crimson"

```

```

        Foreground="#FFE5572C" />
</StackPanel>
<Grid HorizontalAlignment="Center"
    Margin="50 0 0 0">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="2*"></ColumnDefinition>
        <ColumnDefinition Width="1*"></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <StackPanel Orientation="Horizontal"
        HorizontalAlignment="Center">
        <Button Margin="10"
            Content="Зареєструватися"
            Click="Registration_Click"
            Style="{StaticResource MaterialDesignRaisedLightButton}"
            ToolTip="Resource name: MaterialDesignRaisedLightButton" />
    </StackPanel>
    <StackPanel Orientation="Horizontal"
        Grid.Column="1"
        HorizontalAlignment="Center">
        <Button Content="або Увійти"
            Style="{StaticResource MaterialDesignFlatLightButton}"
            ToolTip="MaterialDesignFlatLightButton"
            Click="LoginButton_Click" />
    </StackPanel>
</Grid>
</StackPanel>
</Grid>
</Page>

```

Налаштування пакету EmguCV для запису відео за допомогою веб-камери

```

private void ImageGrabbed(object sender, EventArgs e)
{
    try
    {
        Mat frame = new Mat();
        Mat frame2 = new Mat();
        lock (lockObj)
        {
            if (capture != null)
            {
                if (!capture.Retrieve(frame))
                {
                    frame.Dispose();
                    return;
                }
                if (frame.IsEmpty)
                    return;

                CvInvoke.Resize(frame, frame2, sizeOfVideo, 0, 0);

                VideoFrame.Dispatcher.Invoke(new Action(() =>
                {
                    if (RecordingInProgress)
                    {
                        VideoFrame.Source =
BitmapSourceConvert.ToBitmapSource(frame2);
                        RecordFrame(videoWriter, frame2);
                    }
                    else
                    {
                        VideoFrame.Source = null;
                    }
                }));
            }
        }
    }
}

```

```
        frame2.Dispose();
    }
}

}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

private void RecordFrame(VideoWriter videoWriter, Mat frame)
{
    try
    {
        if (videoWriter.IsOpened)
        {
            using (frame)
            {
                videoWriter.Write(frame);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```