ЗМІСТ

|  |  |
| --- | --- |
| ТЕХНІЧНА ЧАСТИНА………………………………………………………. | 6 |
| 1. ТЕХНІЧНЕ ЗАВДАННЯ…………………………………………………... | 7 |
| 1.1. Підстави для розробки…………………………………………………. | 7 |
| 1.2. Призначення розробки…………………………………………………. | 8 |
| 1.3. Аргументація теми та її актуальність………………………………… | 9 |
| 2. ВИМОГИ ДО РОБОТИ…………………………………………………….. | 11 |
| 2.1. Стадії та етапи розробки……………………………………………….. | 11 |
| 2.2. Аналіз вимог до програмного забезпечення………………………….. | 11 |
| 2.2.1. Функціональні вимоги……………………………………………... | 11 |
| 2.2.2. Вимоги до складу та параметрів технічних засобів……………… | 12 |
| 2.2.3. Вимоги до вхідних та вихідних даних……………………………. | 13 |
| 2.2.4. Вимоги до інтерфейсу……………………………………………… | 13 |
| 2.2.5. Вимоги до тестування програмного забезпечення……………….. | 14 |
| 2.3. Вимоги до програмної документації………………………………….. | 14 |
| ВИСНОВКИ ДО ТЕХНІЧНОЇ ЧАСТИНИ………………………………….. | 15 |
| ТЕОРЕТИЧНА ЧАСТИНА…………………………………………………… | 16 |
| 3. ОГЛЯД ТЕМИ, СУЧАСНІ ПІДХОДИ ДО РЕАЛІЗАЦІЇ, РОЗШИРЕНА |  |
| ПОСТАНОВКА ЗАДАЧІ……………………………………………………... | 17 |
| 3.1. Призначення та область застосування………………………………... | 17 |
| 3.2. Постановка задачі, огляд літератури, підходи до розв’язування……. | 17 |
| 3.3. Опис основних алгоритмів…………………………………………….. | 25 |
| 3.3.1 Алгоритм конвертування діалогових вікон...................................... | 25 |
| 3.3.2 Алгоритм експортування проектів.................................................... | 25 |
| 3.3.3 Алгоритм внутрішнього збереження проектів................................. | 25 |
| 3.3.4 Алгоритм побудови інсталятора....................................................... | 25 |
| 3.4. Опис інструментальних засобів……………………………………….. | 36 |
| 3.5. Загальна структура розробленого програмного продукту………….. | 18 |
| ВИСНОВКИ ДО ТЕОРЕТИЧНОЇ ЧАСТИНИ……………………………… | 37 |
| ПРАКТИЧНА ЧАСТИНА……………………………………………………. | 39 |
| 4. КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ……………... | 40 |
| 4.1. Загальна структура апаратно-програмного комплексу……………… | 40 |
| 4.2. Функціональні можливості системи…………………………………... | 41 |
| 4.3. Користувацький інтерфейс…………………………………………….. | 41 |
| 4.4. Опис класів та програмних модулів…………………………………... | 43 |
| 4.5. Методика та результати випробувань………………………………… | 44 |
| ВИСНОВКИ ДО ПРАКТИЧНОЇ ЧАСТИНИ……………………………….. | 48 |
| ЗАГАЛЬНІ ВИСНОВКИ ДО РОБОТИ……………………………………… | 49 |
| ВИКОРИСТАНА ЛІТЕРАТУРА……………………………………………... | 51 |
| ДОДАТКИ……………………………………………………………………... | 54 |
| Додаток А……………………………………………………………………. | 54 |
| Код програми………………………………………………………………... | 54 |

ТЕХНІЧНА ЧАСТИНА 1. ТЕХНІЧНЕ ЗАВДАННЯ

1.1. Підстави для розробки

Сучасні програмні продукти потребують постійного розповсюдження на численні користувацькі комп’ютери. Надати користувачу автономний файл для автоматичного розгортання та конфігурації програмного забезпечення – чимало не головний підхід, сформований досвідом, що використовують в корпоративному ринку ІТ. Для домашнього використання технологія інсталятора це також зручний засіб для забезпечення необхідним програмним забезпеченням, оновлення на новіші версії та видалення, оскільки уникаються проблеми ресурсних, а саме часових, затрат при альтернативному підході розгортання ПЗ вручну.

Для операційної системи Windows існує декілька вдубованих технологій для роботи з програмами інсталяції, зокрема MSI та MSIX. MSIX через підвищення рівня безпеки має платформні обмеження виконання програм та не дозволяє модифікувати хід встановлення – натомість налаштування програми переносяться на час першого запуску. Більш зріла технологія MSI (Microsoft Installer) або Windows Installer – надає більший набір можливостей та забезпечує ширший ряд підтримуваних версій операційної системи.

Технологія Windows Installer забезпечує єдиний до запуску та виконання програм встановлення, надає єдиний базовий візуальний інтерфейс користувача з можливістю модифікації а також дозволяє запуск окремих автономних модулів виконання, де в свою чергу можливе відображення будь-якого візуального інтерфейсу на замовлення. І хоч MSI налагоджує процеси автоматизованого вбудовування програм в систему, їх модифікацію, відновлення, оновлення на новіші версії та видалення, однак технологія не регулює повною мірою процеси створення таких програм інсталяції.

Початково робота по створенню інсталяторів MSI була організована шляхом ручного редагування таблиць реляційної бази файлу інсталяції. Пізніше спільнотою open source підхід еволюційно переріс у маніпуляції з XML файлами, а згодом і в керування за допомогою сутностей високорівневих мов програмування. Останній підхід поширюється повільно але, без сумнівів несе звання найбільш ефективного для створення базових програм інсталяції оскільки з усіх інших вимагає найменше зусиль до використання та найменше часу на ознайомлення, адже знання всіх тонкостей у роботі з технологією MSI тут не вимагається.

Для домашнього користувача досі актуальною є проблема швидкого створення та налаштування програм інсталяції без необхідності прямої роботи з програмнин кодом, при цьому щоб створена програма використовувала сучасні UI платформи під час виконання, адже відкриті рішення на ринку не надають подібних можливостей.

Для розробника програмного забезпечення є актуальною проблема створення програми встановлення у візуальному інтерфейсі, при цьому щоб можливість подальшої підтримки й розробки надавалася у формі застосунків високорівневих мов програмування, оскільки навіть наявні комерційні рішення не дозволяють редагування файлу проекту на сучасних високорівневих мовах програмуваня.

Розроблене програмне забезпечення та дослідження проведені у цій роботі вирішують наведені вище проблеми, та дозволяють користувачу створювати, редагувати й підтримувати проекти програм інсталяції засобами візуального інтерфейсу та високорівневої мови програмування C#, а створені програми можуть працювати з використанням сучасних UI платформ WPF та WinUI 3.

Розробка програмного забезпечення виконується на підставі рішення  
засідання кафедри МПУіК про затвердження тем курсових робіт та дипломних проєктів (протокол № *2* від “*07*” вересня 2020 року).

1.2. Призначення розробки

Програма-конструктор, розроблена в даній роботі, дозволяє як домашньому користувачу, так і профільному розробнику, будувати файли інсталяції ПЗ за технологією Windows Installer. На основі створених у конструкторі, або конвертованих наявних проектів інсталяції, побудовані інсталятори можуть запускатися на операційній системі Windows з метою розгортання необхідного програмного забезпечення або програмних сервісів. Підтримка проектів інсталяції за допомогою розробленого конструктора можлива на різних платформах (Windows, Linux, MacOS) завдяки використанню мови C# та платформи .NET 6.

1.3. Аргументація теми та її актуальність

Процес розробки інсталяторів потребує оптимізації та вдосконалення, оскільки більшість наявних рішень дотримуються застарілих підходів. Використовувана мова програмування або розмітки та візуальний інтерфейс користувача є ключовими аспектами для вдосконалення в роботі з технологією MSI.

Існуючі відкриті аналоги розробленої програми-конструктора дозволять будувати інсталятори, однак обмежені в можливості користувацьких діалогів до використання в черзі виконання програми інсталяції. Також наявні відкриті рішення не дають змоги використовувати сучасний користувацький інтерфейс і базуються на архаїчному MSI UI, що набув чимало недоліків за два десятиліття з часу впровадження в сучасних реаліях, а також немало застарілих більш не використовуваних можливостей. Відмінністю розробленої програми є можливість побудувати інсталятор на сучасній UI платформі WPF або WinUI 3. Завдяки використанню сучасного візуального інтерфейсу покращується досвід використання програми користувачем, а також вирішуються базові проблеми такі як масштабування вікон на дисплеях різноманітної роздільної здатності. Вагомою відмінністю розробленого ПЗ щодо відкритих аналогів також є можливість вибору діалогів у черзі виконання процесу інсталяції, а ще використання з наперед визначеного набору заготовлених діалогів що відсутні в базовому наборі MSI і можуть статися корисними відповідно до сучасних тенденцій.

На ринку комерційних аналогів, таких як Advanced Installer та InstallShield, вже почали надавати можливість використання діалогів на сучасній платформі як от WinUI. Ключовою перевагою розробленого конструктора тут виступає те, що окрім побудови сучасних діалогів з нуля є можливість імпортувати базові діалоги MSI конвертовані в сучасні за вибором користувача.

Головна перевага розробленого продукту – це можливість зберігати та підтримувати проект інсталятора у форматі проекту C# для Visual Studio. Саме ця особливість забезпечує оптимізація та ефективність розробки проектів програм розгортання з точки зору розробників програмного забезпечення. З точки зору домашнього користувача, побудувати базовий інсталятор за допомогою розробленого констурктора можна легко і швидко, в той же час зберігається доступ до глибокого налаштування через надані інструменти візуального інтерфейсу.

2. ВИМОГИ ДО РОБОТИ

2.1. Стадії та етапи розробки

Таблиця 2.1.1

|  |  |  |
| --- | --- | --- |
| № п/п | Назва етапів дипломного проєкту | Термін виконання етапів |
| 1 | Одержання технічного завдання | 17.09.20 р. |
| 2 | Аналіз поставленого завдання | 21.10.20 р. |
| 3 | Ознайомлення з літературою | 29.12.20 р. |
| 4 | Розробка загальної концепції ПЗ | 19.01.21 р. |
| 5 | Розробка основного алгоритму | 26.02.21 р. |
| 6 | Реалізація функціональної частини | 29.03.21 р. |
| 7 | Реалізація інтерфейсу | 05.04.21 р. |
| 8 | Реалізація допоміжного забезпечення | 10.08.21 р. |
| 9 | Тестування та налагодження програми | 10.09.21 р. |
| 10 | Оформлення програмної документації | 25.10.21 р. |
| 11 | Представлення готового проєкту | 01.12.21 р. |
| 12 | Захист дипломного проєкту | згідно з розкладом |

2.2. Аналіз вимог до програмного забезпечення

2.2.1. Функціональні вимоги

До програмного забезпечення висуваються наступні вимоги:

- можливість побудувати нову програму інсталяції шляхом використання візуального інтерфейсу;

- можливість зберегти проект програми інсталяції для подальшого редагування та підтримки через візуальний інтерфейс;

- можливість будувати користувацькі діалоги інсталятора на обраній платформі за вибором WPF або WinUI 3;

- здатність імпортувати існуючий проект програми інсталяції, початково розроблений засобами WiX Toolset;

- здатність імпортувати існуючий проект розгортання, початково розроблений засобами відкритої бібліотеки WixSharp;

- здатність створити проект програми встановлення на основі існуючого файлу .msi;

- застосування бібліотеки WixSharp для можливості експорту проекту в її форматі (збірки C# з відповідним скриптом побудови);

- можливість експорту проекту інсталятора у форматі .wixproj для подальшої підтримки засобами WiX Toolset;

- реалізації процесів конвертування базових діалогових вікон MSI UI з використанням універсальних абстрактних сутностей користувацьких маніпуляторів, з метою забезпечення подальших оновлень на інші UI платформи.

Програмний продукт, для базового використання, повинен бути зрозумілим користувачу у формі візуального інтерфейсу без вимоги знань технічних особливостей технології MSI та забезпечувати доступ до всіх необхідних конфігурацій розробнику у формі візуального інтерфейсу та програмного коду проекту мовою C#.

2.2.2. Вимоги до складу та параметрів технічних засобів

Мінімальні вимоги до апаратного забезпечення комп'ютера для коректної роботи розробленого програмного продукту:

Таблиця 2.2.2.1

|  |  |
| --- | --- |
| Параметр | Значення |
| Процесор | 2-ядерний Intel або AMD |
| ОЗП | 4 ГБ |
| Екран | 1024x768px |

Необхідною вимогою для коректної роботи розробленого програмного забезпечення є встановлені середовище розробки Microsoft Visual Studio 2022 Preview 4 з компонентами розробки C# застосунків, Windows SDK та WinUI 3, набір інструментів WiX Toolset v3.11.2 та платформа виконання .NET 6.

Розроблене програмне забезпечення може виконуватись на операційній  
системі Windows за умови, що встановлено все необхідне для роботи продукту програмне забезпечення. Виконання на операційних системах Linux та MacOS можливе лише в режимі редагування WixSharp з обмеженням на побудову програм інсталяцій через недоступність встановлення Windows SDK та WiX Toolset на цих операційних системах.

2.2.3. Вимоги до вхідних та вихідних даних

Вхідними даними для розробленого програмного продукту є проекти форматів .csproj (WixSharp), .wixproj (WiX Toolset), файли .msi та конфігурації, застосовані користувачем у візуальному інтерфейсі програми.

Вихідними даними є файли .json (конфігурації проектів), проекти форматів .csproj (WixSharp), .wixproj (WiX Toolset), і файли побудованих інсталяторів .msi та .exe.

2.2.4. Вимоги до інтерфейсу

Інтерфейс програмного продукту повинен бути декларативним.  
Користувач повинен мати можливість змінювати властивості проекту інсталяції, необхідні для налагодження кінцевого пакету розгортання.

Взаємодія із застосунком здійснюється через настільний віконний інтерфейс платформи WPF або WinUI 3.

Користувач має можливість переглядати властивості імпортованих інсталяторів, або проектів інсталювання та змінювати властивості з метою побудови нових .msi файлів.

Окрім цього, у користувача повинна бути можливість розширювати функціональність розробленого програмного продукту, шляхом експорту налаштованого проекту в формати WixSharp (.csproj) та WiX Toolset (.wixproj) з метою їх редагування і підтримки засобами відповідних інструментів.

Інтерфейс програми має забезпечити опцію швидкого перемикання бажаної UI платформи для побудови інсталятора, а також опції для зміни необхідних до використання діалогових вікон у черзі процесу інсталяції і додавання нових діалогових вікон. Кожне діалогове вікно повинно супроводжуватися формою попереднього перегляду відповідно до обраної UI платформи WPF чи WinUI 3, надаючи таким чином змогу користувачу оцінити візуальне представлення налаштованого інтферфейсу програми інсталювання.

Також, інтерфейс програмного забезпечення повинен супроводжуватися текстовими рядками статусу виконання побудови інсталятора чи проекту інсталятора, імпорту та експорту, оскільки ці дії потребують певного часу на виконання. Неодмінно, візуальний інтерфейс повинен бути постійно доступним до використання, за необхідності з функціональними обмеженнями в момент часу зумовленими поточним станом виконання, і не має заморожуватися під час обробки довготривалих операцій.

2.2.5. Вимоги до тестування програмного забезпечення

Для тестування програмного забезпечення необхідно виконати наступні  
дії:  
1. Використовуючи засоби інструменту Git, наприклад через застосунок SourceTree, клонувати репозиторій роботи на локальний диск комп’єтера.

2. Відкрити головний рішення (.sln) з директорії репозиторію в середовищі Microsoft Visual Studio 2022 Preview 4.

3. У списку вмісту рішення натиснути правою кнопкою миші на головній збірці проекту та встановити її по замовчуванню головною на запуск.

4. Натиснути клавішу F5 для запуску головного модуля програми (конструктора).

Розроблене програмне забезпечення буде вважатися таким, що працює коректно, якщо після виконання описаних кроків з’явиться головне вікно програми-конструктора інсталяторів.

2.3. Вимоги до програмної документації

Програмне забезпечення постачається разом із супроводжувальною  
документацією, до складу якої входить:

1. XML документація програмного коду.
2. Самодокументовані модульні тести.
3. Технічна специфікація інтеграційних тестів.
4. Приклади використання програми у формі окремих покрокових сесій.

ВИСНОВКИ ДО ТЕХНІЧНОЇ ЧАСТИНИ

Під час виконання технічної частини дипломної роботи було проведено дослідження актуальності технології Windows Installer. Проаналізовано наявні програмні рішення для створення пакетів інсталяції, окреслено їх переваги й недоліки. Обґрунтовано актуальність обраної теми роботи, та особливості що забезпечують вагомі відмінності розробленого програмного забезпечення від наявних комерційних та відкритих продуктів.

Виокремлено функціональні новації розробленої програми, що покращують процеси використання інструментів MSI та пропонують нові оптимальні підходи ефективної роботи в сфері створення програм інсталяції застосунків Windows. Описано вимоги до розробленого програмного забезпечення.

ТЕОРЕТИЧНА ЧАСТИНА

3. ОГЛЯД ТЕМИ, СУЧАСНІ ПІДХОДИ ДО РЕАЛІЗАЦІЇ,РОЗШИРЕНА ПОСТАНОВКА ЗАДАЧІ

3.1. Призначення та область застосування

Розроблений програмний продукт призначений для створення та побудови інсталяторів настільних додатків Windows на основі технології Microsoft Installer. Областю застосування є будь яка версія операційної системи сімейства Windows NT, оскільки платформа для запуску, виконання, і обслуговування MSI інсталяторів є вбудованою в операційну систему, а тому програми інсталяції не потребують допоміжного ПЗ для своєї роботи, а їх працездатність гарантована навіть після видалення власне файлу інсталяції з диску після встановлення програми в систему.

**3.2. Постановка задачі, огляд літератури, підходи до розв’язування**

Задача побудови інсталятора – одна із задач розповсюдження програмного забезпечення.

Подібні задачі досліджувались такими авторами як \_[x], \_[x], \_[x]. Зокрема Nick Ramirez дослідив використання методу BurnUI [x, с. 122] з інтрументарію WiX Toolset як точку входу для інсталяторів на базі власного користувацького інтерфейсу, зокрема WPF, що є важливою частиною в зробленій роботі, оскільки виконання власного інтерфейсу вікон використовує засіб BurnUI частково через WixSharp і частково безпосередньо.

Також Bob Baker досліджує особливості огранізації програм-конструкторів інсталяторів з пропрієтарним форматом проектів на прикладі InstallShield [x]. Розроблена програма частково слідує цим принципам у аспекті збереження конфігураційних файлів стану проекту, що підтримуються лише внутрішньо, і не призначені до прямого редагування чи експорту в загальні формати проектів інсталяції як WixSharp чи WiX Toolset.

Загалом доступні підходи по створенню інсталятора наступні: ручне редагування MSI таблиць, автоматизована розробка з використанням XML файлів та інструментів WiX Toolset та автоматизована розробка з використанням сутностей і високорівневої мови програмування.

На основі останнього підходу, а саме використанні високорівневої мови програмування C# та відкритої бібліотеки WixSharp, базується розроблене програмне забезпечення, оскільки такий підхід дозволяє отримати найбільш якісний і швидкий процес розробки і забезпечує зручність підтримки розроблених проектів.

3.3. Опис основних алгоритмів

3.3.1 Алгоритм конвертування діалогових вікон

Як зазначено з функціональних вимог, розроблена програма-конструктор дозволяє створювати пакети інсталяції з власним користувацьким інтерфейсом на платформах WPF або WinUI 3. Така можливість надається не лише з доступу до створення нового діалогово вікна на обраній платформі, а й використанням з ряду наявних базових діалогових вікон MSI UI.

Отримання аналогів діалогових вікон платформи MSI UI на інших платформах досягається шляхом аналізу загальних властивостей типових елементів користувацького інтерфейсу. Допоміжна утиліта (програмний модуль) проходячи крізь усі наявні вікна виокремлю самостійні елементи як кнопки чи текстові поля, аналізує їх властивості як загальні – висота, ширина, абсолютна позиція у вікні чи відносні розміри за прив’язками, так і специфічні до елементу керування – обробник натискання кнопки, фонове зображення чи максимальна кількість символів тексту в полі. На основі зібраних властивостей та відокремлення інтерфейсних компонентів будуються сутності елементів користувацького інтерфейсу. Функціональність таких сутностей не покриває всі можливі застосування сучасних UI платформ, однак покриває всі можливі випадки застосування у платформі MSI UI. Зібрані дані, завдяки свої концептуальній основі, є незалежними від використовуваної віконної платформи адже містять загальні абстракції та властивості, притаманні будь-якому сучасному UI фреймворку як базис. Отож, згенеровані сутності діалогових вікон можуть пізніше бути використані не лише для необхідних у роботі технологій WPF і WinUI 3 а й довільних інших.

На основі видобутих абстракцій представлення інтерфейсу заготовлюються автономні діалогові вікна під необхідні платформи. Обробка здійснюється допоміжною утилітою (програмним модулем) що переходом по сутностях всіх діалогах будує аналоги на потрібній платформі. Таким чином, наперед заготовлені конвертовані діалоги не будуть забирати додатковий час на обробку під час побудови інсталяторів, а конвертація за аналогічним алгоритомом під час виконання програми буде вимагатися лише для власних діалогів, що відсутні в ряді базових для середовища MSI UI.

3.3.2 Алгоритм експортування проектів

Під час виконання основного проектного модулю – програми конструктора, всі маніпуляції зроблені користувачем до відповідного аткивного проекту інсталювання зберігаються. Незважаючи на вихідне джерело отриманих даних, як то наявний .msi файл, файл проекту .wixproj чи файл проекту .csproj з скриптом побудови інсталятора всередині за допомогою бібліотеки WixSharp, вся інформація і властивості програми інсталяції будуть адаптовані до збереження у формі об’єктних моделей використовуваних розробленою програмою для оперування даними.

Більша частина моделей для конфігурації, наприклад клас представлення опціонального компонента інсталятора чи клас автоматизованого імпорту файлів інсталяції з вказаної директорії, взяті безпосередньо або модифіковані на основі наявних класів відкритої бібліотеки WixSharp. Інші абстракції, як класс представлення модифікації системного реєстру, визначені в проекті внутршньо і не використовуються для експорту в інші формати.

За кожним проектом програми встановлення конструктор закріплює набір даних, представлених такими об’єктами, що можуть бути збережені у файл як конфігурація цього проекту, або використовуються лише на час виконання програми з оперативної пам’яті комп’ютера.

Аналізатор побудови, представлений окремим програмним модулем у проекті, співставляє задані користувачем параметри проекту розгортання і їх основі будує проекті рішення вибраного формату експорту .csproj чи .wixproj з досягненням аналогічного вмісту на обраному типі проекту. Таким чином конфігурація проекту, збережена в конструкторі, буде будувати ідентичні рішення інсталятора на еспортованих проектах навіть за умови що вони збиратимуться у відповідному їм материнському інструментарії: WixSharp для .csproj та WiX Toolset для .wixproj.

Завдяки описаному підходу, розроблений конструктор не лише надає новий високий абстрактний рівень в процесі створення програм-інсталяторів, але й зберігає сумісність з навявними розпосвюдженими відкрити рішеннями з метою, при необхідності, перенести розробку проекту інсталювання на більш низький сумісний рівень абстракції з відповідним їй набором інструментів.

3.3.3 Алгоритм внутрішнього збереження проектів

Всі об’єктні моделі для збереження стану і конфігурації активного проекту в розробленому конструкторі спроектовані таким чином, щоб незалежно від точки входу в процес їх використання забезпечити можливість однозначного визначення стану. Відповідно, в будь-який момент часу роботи програми конструктора проект інсталяції підлягяє збереженню і може буде пізніше відновлений за цим станом.

Таким чином, завдяки використанню відкритої бібліотеки Newtonsoft.Json всі конфігураційні об’єкти активного проекту серіалізуються у файл-даних формату JSON. Цей файл зберігається по необхідності у потрібне місце за вибором користувача з розширенням .json, а також автоматично створює тимчасові резервні копії під час редагування проекту в папці програми з метою збереження стану розробки на випадок непередбачуваного завершення програми або з метою відновлення до попередніх конфігурацій. Описана функція автозбереження керується відповідною опцією на екрані налаштувань програми.

3.3.4 Алгоритм побудови інсталятора

Акумулюючи можливості бібліотеки WixSharp конструктор перетворює конфігураційні об’єкти ативного проекту інсталятора в скрипт побудови, що за допомогою відкритої бібліотеки WixSharp та попередньо встановлених на комп’ютері інструментів WiX Toolset v3.11 будує файл .msi, що і являє собою готовий пакет розгортання (програму інсталяції) цільового програмного забезпечення користувача.

Оскільки, побудова базується на використанні скрипта створення інсталятора класами бібліотеки WixSharp – модуль побудови підшукує аналоги класів для ін’єкції потрібних властивостей цільового інсталятора, або використовує класи створені в розробці програмного забезпечення цієї роботи.

3.4. Опис інструментальних засобів

Розробка програмного продукту здійснювалась на персональному  
комп’ютері наступної конфігурації:

1. Процесор – Intel Core i5 7200U

2. ОЗП – 16 ГБ DDR4

3. Відеоадаптери:

- інтегрований – Intel HD Graphics 620

- дискретний – NVIDIA GeForce 940 MX

4. Носій – SSD Intel M.2 NVMe 240 ГБ

5. DVD-RW – HL-DT-ST DVDRAM GUE0N

Комп’ютер працює під управлінням операційної системи Microsoft Windows 10 Pro.

**3.5. Загальна структура розробленого програмного продукту**

Розроблена програмна система має наступну структуру:

**-** головний модуль з інтерфейсом основної програми-конструктора на базі обох платформ WPF та WinUI 3 з можливістю гарячого перемикання між ними під час виконання програми;

- модуль з реалізацією побудови інсталяційних файлів;

- модулі автоматизованого виконання інтеграційних тестів на заготовлених .msi файлах та побудованих проектних результатах;

- модуль з прикладами використання бібліотечної частини розробленого програмного забезпечення;

- проектні модулі автоматизованого виконання модульних тестів коду основної програми, утиліт та окремих класів;

- модуль конвертування діалогів MSI UI в об’єкти абстрактних сутностей;

- модуль з реалізацією функцій імпорту даних .msi файлу;

- модуль зі згенерованими сутностями представлень компонентів візуального інтерфейсу;

- модуль з готовими діалоговими вікнами WinUI 3, аналогічних набору вікон MSI UI, попередньо наповнений шляхом використання допоміжного програмного модулю генерування діалогів цільової UI платформи із попередньо створених абстрактних сутностей представлення елементів користувацького інтерфейсу;

- модуль з реалізацією функції експорту проекту інсталяції в формат бібліотеки WixSharp (.csproj);

- модуль з реалізацією функції створення нового проекту конструктора на базі імпортованого існуючого проекту .csproj з використанням бібліотеки WixSharp;

- модуль виконання функцій обробки існуючого проекту WIX (.wixproj) та створення нового проекту конструктора на базі імпортованих даних XML файлів проекту WIX;

- модуль з реалізацією перетворень налаштованого проекту конструктора у формат готово до використання і побудови засобами WiX Toolset проекту й експорт у форматі .wixproj;

- модуль з готовими діалоговими вікнами WPF, аналогічних набору вікон MSI UI, попередньо створених допоміжним програмним модулем генерування з абстрактних сутностей цільових діалогів інтерфейсу, та зберігаючий заготовлені небазові діалоги для використання в реаліях сучасних тенденцій.

ВИСНОВКИ ДО ТЕОРЕТИЧНОЇ ЧАСТИНИ

Під час роботи над теоретичною частиною було сформульовано розширену задачу проекту та здійснено огляд літератури та підходів до реалізації задачі.

Описано призначення та область застосування розробленого програмного продукту для створення програм-інсталяторів настільних застосунків Windows.

Представлено основні алгоритми реалізовані в програмі. Описано використані інструментальні засоби та загальну структуру розробленого програмного рішення.

ПРАКТИЧНА ЧАСТИНА

4. КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Загальна структура апаратно-програмного комплексу

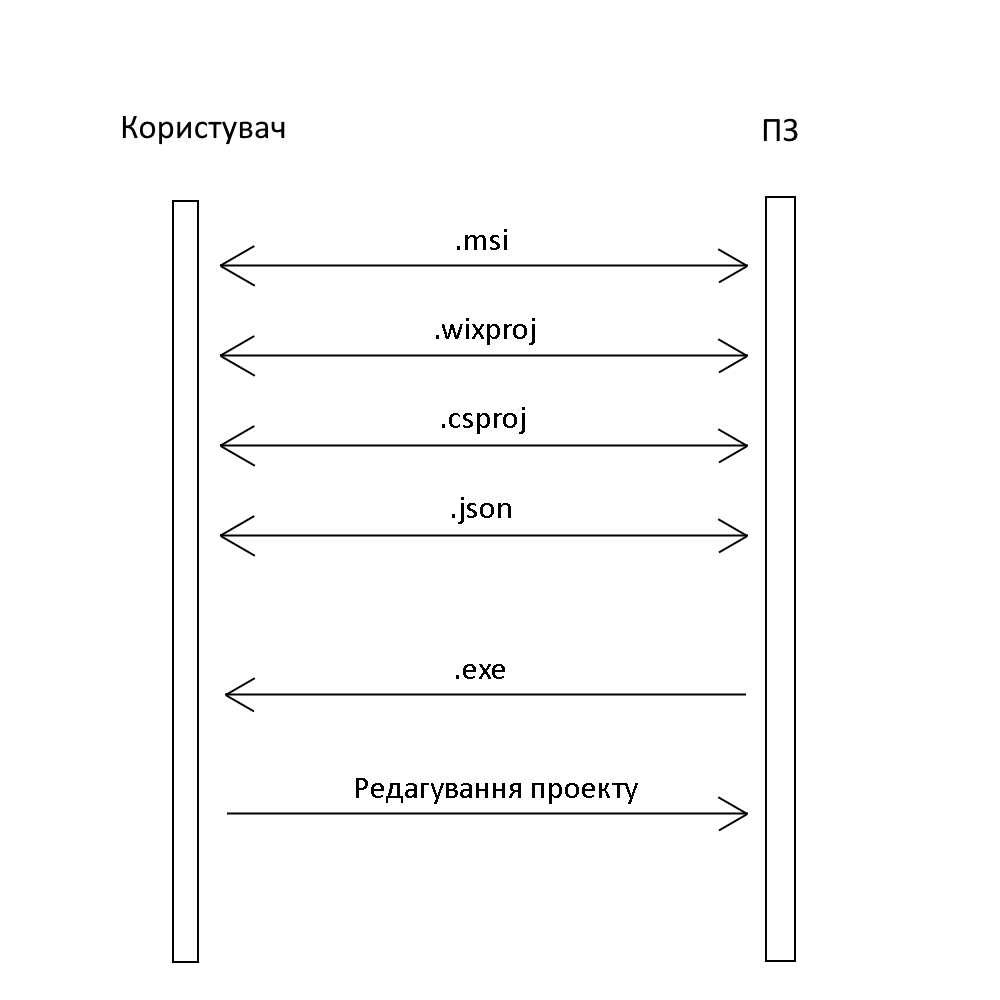
Для розв’язання задачі побудови інсталяторів настільних застосунків Windows розроблено програмне забезпечення мовою програмування C# з використанням засобів Windows Installer SDK, інструментів WiX Toolset та відкритої бібліотеки WixSharp.

Взаємодія користувача із продуктом здійснюється через візуальний інтерфейс програми у вигляді вікон WPF або WinUI 3. Користувач імпортує наявні проекти інсталяції .csproj, .wixproj або існуючий файл інсталяції .msi, здійснює редагування властивостей та виконує побудову та збереження і експорт у необхідний підтримуваний формат.

Розроблений продукт здійснює виведення результатів побудови у вигляді .msi або .exe файлів, а результати редагування проекту можуть бути збережені у форматі .json для внутрішнього використання. Для подальшої розробки, редагування й побудови проект у конструкторі може бути експортований до бажаного стороннього формату проекту .wixproj (WiX Toolset) або .csproj (WixSharp).

Схематично взаємодію користувача із програмою  
зображено на рис. 4.1.1.

Рис. 4.1.1. Взаємодія користувача з програмою



4.2. Функціональні можливості системи

Програмне забезпечення має такі функціональні можливості:

- створення нових проектів інсталяції у візауальному інтерфейсі програми-конструктора;

- імпорт існуючих .msi файлів для створення нових проектів інсталяції на основі отриманих даних;

- імпорт існуючих .wixproj проектів для створення нових проектів інсталяції шляхом аналізу, читання й перетворення отриманих даних;

- імпорт існуючих .csproj проктів, зі скриптами побудови інсталяторів засобами бібліотеки WixSharp всередині, для створення нових проектів інсталяції на основі адаптованих даних;

- експорт налаштованого проекту інсталяції у формати .wixproj та .csproj з метою їх подальшої підтримки засобами відповідного материнського інструментарію;

- збереження проекту інсталятора у власному форматі .json що містить дані серіалізованих об’єктів конфігурації проекту;

- побудова проекту інсталювання у файл .msi або .exe за бажанням користувача;

- автоматичне тимчасове збереження проміжних станів активного проекту з метою повернення до попередніх версій і навпаки.

4.3. Користувацький інтерфейс

Інтерфейсом програмного продукту є вікна WPF або WinUI 3. Користувач повинен мати можливість здійснювати розробку програм-інсталяторів через візуальний інтерфейс.

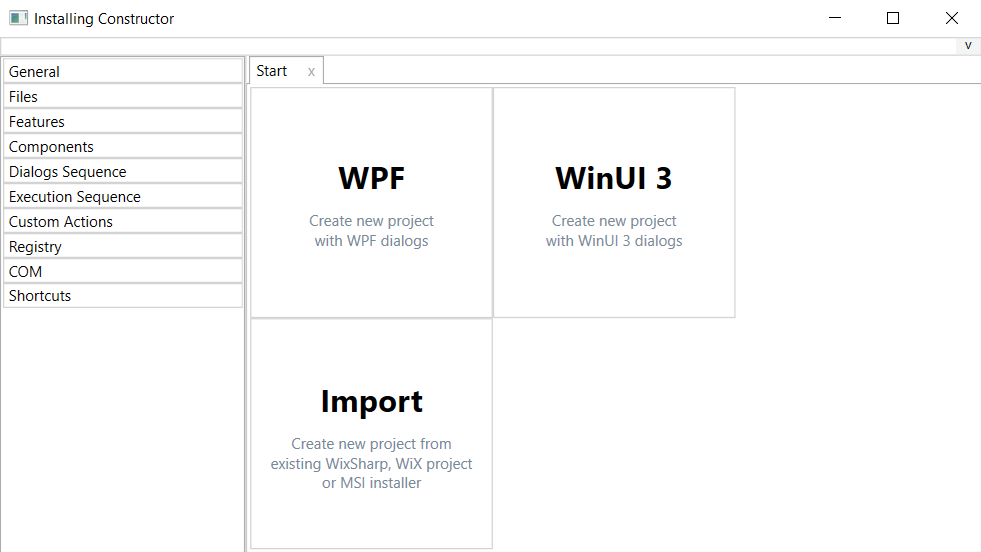


Рис. 4.3.1. Сторінка початку роботи

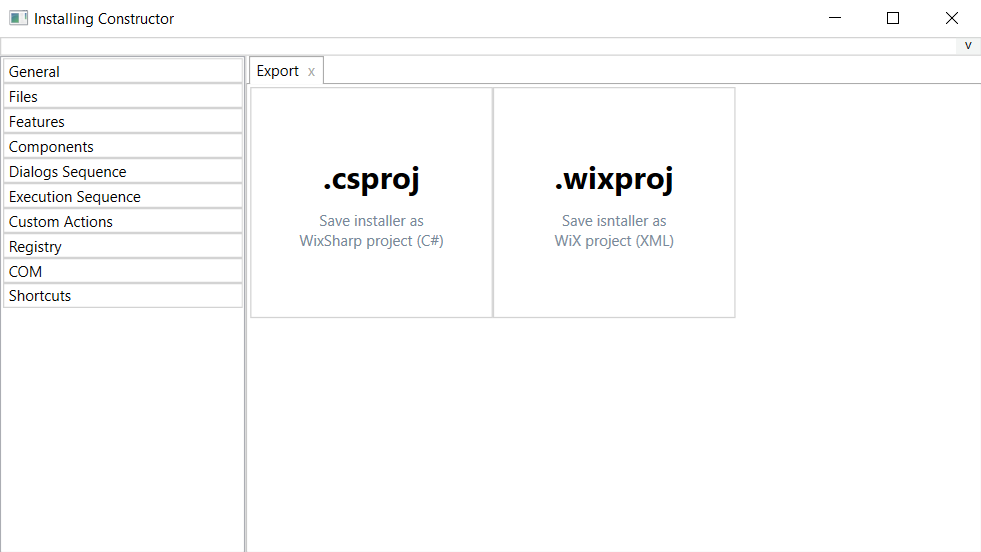


Рис. 4.3.2. Сторінка експорту проекту

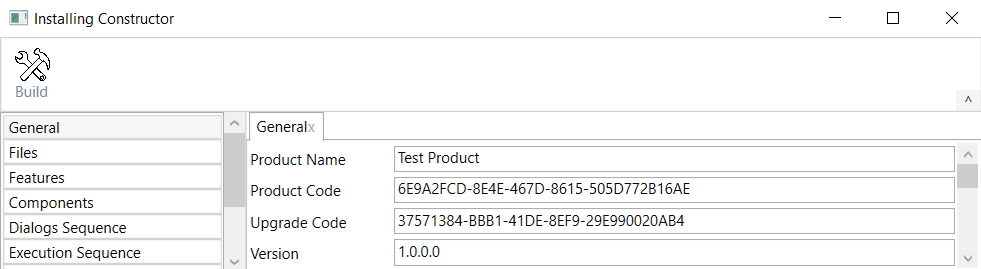


Рис. 4.3.3. Сторінка основних властивостей активного проекту

Взаємодія користувача із програмним продуктом здійснюється через головне вікно конструктора, що змінює вміст відповідно до поточної відкритої сторінки програми та відповідно оновлює набір доступних функцій згідно з поточним контекстом застосування загальних опцій програми чи налаштувань активного проекту.

Стартова сторінка, зображена на рис. 4.3.1, надає доступ до команд, що очікуються найбільш вживаними під час роботи – створення нового проекту на базі фреймворку WPF, створення нового проекту на базі фреймворку WinUI 3 та імпорт існуючих інсталяторів чи проектів інсталяторів підтримуваних відкритих форматів.

4.4. Опис класів та програмних модулів

Розроблене програмне забезпечення складається із таких модулів:  
**Спочатку просто перераховуються модулі, а далі робиться детальний опис кожного з них**

4.5. Методика та результати випробувань

**Описується методика тестування ПЗ і наводяться результати тестування на конкретних прикладах**

Таким чином, нами було проведене тестування роботи розробленого  
програмного продукту. За результатами випробувань програмне забезпечення показало свою працездатність.

ВИСНОВКИ ДО ПРАКТИЧНОЇ ЧАСТИНИ

У ході роботи над практичною частиною було розроблено програмне  
забезпечення для ….

У розробленому продукті реалізовані:

− …;

− …;

− ….

Програмний продукт розроблений мовою програмування … із  
використанням …. Продукт дозволяє….

Перевірка роботи програмного продукту здійснювалася ….

Розроблений програмний продукт відповідає поставленим вимогам та  
продемонстрував свою працездатність.

ЗАГАЛЬНІ ВИСНОВКИ ДО РОБОТИ

У цій роботі нами проводилося дослідження проблеми …

Для конструювання ПЗ нами були використані такі алгоритми:

1)…

2)…

3)…

У цілому, нами був розроблений програмний продукт для розв'язання  
задачі …, який за результатами випробувань показав свою працездатність.  
Розроблений програмний продукт можна використовувати для:

- …

-…

-…

Крім цього, програмне забезпечення можна використовувати і для більш  
загальних задач….

За результатами проведеного дослідження були подані тези на  
студентську наукову конференцію Чернівецького національного університету, а також зроблено доповідь на ….

ВИКОРИСТАНА ЛІТЕРАТУРА

**(літературні джерела оформлюються згідно з ДСТУ** 8302:2015**)**



ДОДАТКИ

Додаток АКод програми

import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
%matplotlib inline  
data = pd.read\_csv("../input/creditcard.csv")  
data.head()  
n(underSampleDf))