Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка програмної системи створення інсталяційних пакетів

Виконав: студент ___4___ курсу
групи ___2ПІ-18б____
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Войтенко Д. О.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Рейда О. М
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф КН Белзецький Р. С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
_____

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Войтенка Денис Олександровича

1. Тема роботи – «Розробка програмної системи створення інсталяційних пакетів»

Керівник роботи: Рейда Олександр Миколайович , к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24.03.22р №66

2. Строк подання студентом роботи 13.06.22

3. Вихідні дані до роботи: Середовище розробки – QtCreator та Visual Studio 2022, Мови розробки – Python та C#, Операційна система – Windows 10.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; розробка архітектури та алгоритмів програмного додатка; розробка програмного додатку; тестування додатку, висновки; список використаних джерел, додатки, графічна частина.

5. Перелік графічного матеріалу: актуальність розробки, головні задачі дослідження, новизна отриманих результатів, практична цінність, діаграма класів, блок-схема алгоритму виклику функцій, обробка ключових слів, ініціалізація даних, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О.М., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.2022 – 26.04.2022	Вик.
2	Розробка архітектури та алгоритмів роботи системи	27.04.2022 – 08.05.2022	Вик.
3	Вибір середовища та мови розробки	09.05.2022 – 11.05.2022	Вик.
4	Розробка програмного продукту	12.05.2022 – 23.05.2022	Вик.
5	Тестування роботи системи	24.05.2022 – 01.06.2022	Вик.
6	Оформлення матеріалів до захисту БДР	02.06.2022 – 10.06.2022	Вик.

Студент _____ **Войтенко Д.О.**

(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Рейда О.М.**

(підпис) (прізвище та ініціали)

Анотація

Бакалаврська дипломна робота складається з 50 сторінок формату А4, на яких є 46 рисунків, 4 таблиць, список використаних джерел містить 12 найменувань.

У бакалаврській дипломній роботі проведено аналіз роботи пакетних менеджерів, які використовуються в різних операційних системах, розроблено алгоритми функціонування та графічний інтерфейс користувача. Мета досліджень полягає у покращені стабільності і безвідмовності, підвищені ефективності роботи програмних застосунків і операційної системи за рахунок розробки системи створення інсталяційних пакетів з використанням шаблонів.

Запропоновано метод використання веб серверу для зберігання даних про програмне забезпечення без прив’язки до операційної системи; метод встановлення програмного забезпечення за допомогою сценаріїв. Розроблено веб сервер для отримання актуальних даних та базу даних.

Отримані в бакалаврській дипломній роботі результати можна використати для створення пакетного менеджера.

Ключові слова: пакетний менеджер, операційні системи, програмне забезпечення.

Annotation

The bachelor's thesis consists of 50 A4 pages with 46 figures, 4 tables, the list of used sources contains 12 titles.

In the bachelor's thesis a detailed analysis of the work of batch managers used in various operating systems, developed algorithms and graphical user interface. The purpose of the research is to improve the stability and reliability, increase the efficiency of software applications and operating system by developing a system for creating installation packages using templates.

The method of using a web server to store software data without binding to the operating system is proposed; script installation method. A web server for obtaining up-to-date data and a database have been developed.

The results obtained in the bachelor's thesis can be used to create a package manager.

Keywords: package manager, operating systems, software.

ЗМІСТ

ВСТУП

З розвитком технологій зростає потреба в надійному програмному забезпеченні та засобах контролю за ним. Підприємство чи державна установа потребують робочі станції, які будуть надійними та забезпечать потреби без небезпеки отримання шкідливого програмного забезпечення, що стало основним рушієм для створення пакетних менеджерів, які надають змогу встановлювати перевірене програмне забезпечення з надійних джерел.

Пакетні менеджери призначені для:

– підключення розширених бібліотек - призначений для програм, що вимагають іншого програмного забезпечення, завантажує необхідне програмне забезпечення за допомогою динамічного зв’язування;

– конфігурування - призначений для налаштовування і зберігання налаштувань програми;

– підключення до репозиторію – призначений для завантаження програми зі спеціального сховища;

– оновлення – призначений для пошуку нових версій програм та їх оновлення;

– видалення – призначений для видалення програм та всіх залежних бібліотек від неї, для економії місця на диску користувача.

Для забезпечення стабільності, надійності роботи операційної системи і підвищення ефективності її використання необхідно вчасно проводити оновлення програмних засобів, інакше, відбувається зниження ефективності і якості роботи операторів, погіршується стабільність і безвідмовність роботи програмних застосунків і операційної системи.

Існуючі методи і засоби менеджменту системи такі, як пакетні менеджери, характеризуються суттєвим використанням ресурсів і операційного часу, що в значній мірі впливає на затримки у встановлені і оновлені новітніх програмних засобів. Такі недоліки призводять до зменшення стабільність і безвідмовність, зниження ефективності роботи програмних застосунків і операційної системи.

Тому актуальними є питання розробки системи створення інсталяційних пакетів з використанням сценаріїв, що дозволяє покращити стабільність і безвідмовність, підвищити ефективність роботи програмних застосунків і операційної системи.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Мета бакалаврської дипломної роботи полягає у покращені стабільності і безвідмовності, підвищені ефективності роботи програмних застосунків і операційної системи за рахунок розробки системи створення інсталяційних пакетів з використанням шаблонів.

Головні задачі дослідження є:

– провести аналіз існуючих менеджерів інсталяційних пакетів;

– запропонувати нові:

- метод використання веб серверу для зберігання даних про програмне забезпечення без прив'язки до операційної системи;
- метод встановлення програмного забезпечення за допомогою сценаріїв;

– розробити графічний інтерфейс користувача для різних операційних систем;

– розробити веб сервер за архітектурою Rest Api;

– провести експериментальні дослідження розроблених засобів встановлення програмного забезпечення.

Об'єкт дослідження – процес розробки системи створення інсталяційних пакетів.

Предмет дослідження є методи та засоби розробки системи створення інсталяційних пакетів.

Методи дослідження. Під час досліджень використовувалися такі методи дослідження: методи аналізу та синтезу для побудови ключових моделей пакетного менеджеру, методи прикладної теорії інформації і теорії алгоритмів для розробки

алгоритмів і програмного забезпечення, методи тестування для перевірки працездатності системи; методи зберігання даних у реляційній базі даних.

Наукова новизна отриманих результатів:

- уперше запропоновано метод використання сценаріїв у системах створення інсталяційних пакетів, який на відміну від існуючих, використовує набір умов і правил, що дозволяє підвищити ефективність оновлення програмних застосунків і операційної системи;
- уперше запропоновано метод використання сценаріїв, що зберігаються на веб сервері, який на відміну від існуючих, класифікує їх на уніфіковані і авторські, що дозволяє підвищити якість виконання процесу управління користувачем.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі розроблено систему створення інсталяційних пакетів з використанням сценаріїв, що дозволяє покращити стабільність і безвідмовність роботи операційної системи.

Особистий внесок здобувача. Всі результати, які викладенні у бакалаврській дипломній роботів були отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: аналіз пакетних менеджерів

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на: Науково-технічна конференція підрозділів Вінницького національного технічного університету (Вінниця 2022) [12].

Публікації. Основні результати досліджень опубліковано у матеріалах конференції.

Аналіз. У пояснювальній записці до бакалаврської дипломної роботи було розглянуто 4 розділи та було використане 11 літературних джерел.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану пакетних менеджерів

З розвитком технологій все більше постає питання в захисті інформації та безпечному програмному забезпеченні та надійних джерелах його постачання. Для таких потреб створюють пакетні менеджери, які вірно встановлять програмне забезпечення на пристрій користувача, без загроз для нього [1].

Також для полегшення поширення свого програмного забезпечення розробники створюють свої репозиторії, в якому знаходиться їхнє програмне забезпечення, а користувач може добавити його у базу свого пакетного менеджера.

З переваг використання наявних пакетних менеджерів:

- використання типового формату пакетів;
- наявність існуючих репозиторіїв.

Попри ці переваги є ряд проблем, які вимагають рішення:

- Відсутність графічного інтерфейсу;
- залежність від операційної системи;
- відсутність категорій програмного забезпечення.

Виходячи з описаних вище проблем можна зрозуміти, що використання доступних пакетних менеджерів несуть з собою складнощі у використанні, та прив’язані до певної операційної системи, без змоги надати уніфікований спосіб взаємодії з користувачем.

1.2 Порівняльний аналіз аналогів

На даний момент існує велика кількість різних пакетних менеджерів. Найбільш відомими з них є:

- Apt;
- Pacman;
- Pamac;
- Winget.

Apt - Advanced package tool(просунутий пакетний менеджер) (див. рис. 1.1) – - пакетний менеджер, який використовується в дистрибутивах Ubuntu та Debian, та працює з файлами у форматі deb-пакетів [2]. У його репозиторіях містяться тільки те програмне забезпечення, яке повністю відкрите і вільне до використання. Зазвичай програмне забезпечення не самих останніх версій.

```
wikipedia@wikimedia.org:~$ sudo apt-get install mediawiki
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libkipi7 libkexiv2-8 python-urwid python-iniparse python-wicd libtelepathy-qt4-0
  libkdcraw8
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  libdbd-mysql-perl libdbi-perl libhtml-template-perl libnet-daemon-perl libplrpc-perl
  mysql-client-5.1 mysql-server mysql-server-5.1 php5
Suggested packages:
  libipc-sharedcache-perl mediawiki-math memcached clamav tinyca mailx
The following NEW packages will be installed:
  libdbd-mysql-perl libdbi-perl libhtml-template-perl libnet-daemon-perl libplrpc-perl
  mediawiki mysql-client-5.1 mysql-server mysql-server-5.1 php5
0 upgraded, 10 newly installed, 0 to remove and 223 not upgraded.
Need to get 27.8MB of archives.
After this operation, 84.6MB of additional disk space will be used.
Do you want to continue [Y/n]? 
```

Рисунок 1.1 – Приклад команди пакетного менеджера Apt

Pacman (див. рис. 1.2) – пакетний менеджер, який використовується у дистрибутиві ArchLinux та працює з бінарними пакетними файлами. У його репозиторії велика кількість програмного забезпечення, але яке підтримується ентузіастами. Попри популярність не підтримується багатьма розробниками офіційно. Проте завжди програмне забезпечення найновіших версій.

Pacman написаний мовою програмування C і використовує bsdtar [3]

```
wgiokas @ wst420 — ~
pacman --version

                    Pacman v4.1.0rc1-39-gf89f4 - libalpm v8.0.0
                    Copyright (C) 2006-2013 Pacman Development Team
                    Copyright (C) 2002-2006 Judd Vinet

                    This program may be freely redistributed under
                    the terms of the GNU General Public License.

wgiokas @ wst420 — ~
sudo pacman -Syu
:: Synchronizing package databases...
 testing is up to date
 core is up to date
 extra is up to date
 community-testing is up to date
 community is up to date
 multilib-testing is up to date
 multilib is up to date
:: Starting full system upgrade...
warning: xbindkeys: local (1.8.5-4) is newer than community (1.8.5-3)
resolving dependencies...
looking for inter-conflicts...

Packages (1):

Name                  Old Version  New Version  Net Change  Download Size

multilib/lib32-curl   7.28.1-1     7.29.0-3     0.00 MiB         0.22 MiB

Total Download Size:    0.22 MiB
Total Installed Size:   1.02 MiB
Net Upgrade Size:       0.00 MiB

:: Proceed with installation? [Y/n]
:: Retrieving packages ...
 lib32-curl-7.29.0-3-x86_64                      230.1 KiB   405K/s 00:01
(1/1) checking keys in keyring
(1/1) checking package integrity
(1/1) loading package files
(1/1) checking for file conflicts
(1/1) checking available disk space
(1/1) upgrading lib32-curl
wgiokas @ wst420 — ~
```

Рисунок 1.2 – Приклад команди пакетного менеджера Pacman

Pamac (див. рис. 1.3) – пакетний менеджер, який використовується у дистрибутиві Manjaro Linux та працює з бінарними пакетними файлами [4]. Має інтерфейс графічного користувача з мінімалістичним виглядом. Не підтримує інші репозиторії.

Менеджер пакунків Pamac може встановлювати, оновлювати та видаляти пакунки. Використання пакетів замість компіляції та встановлення програм

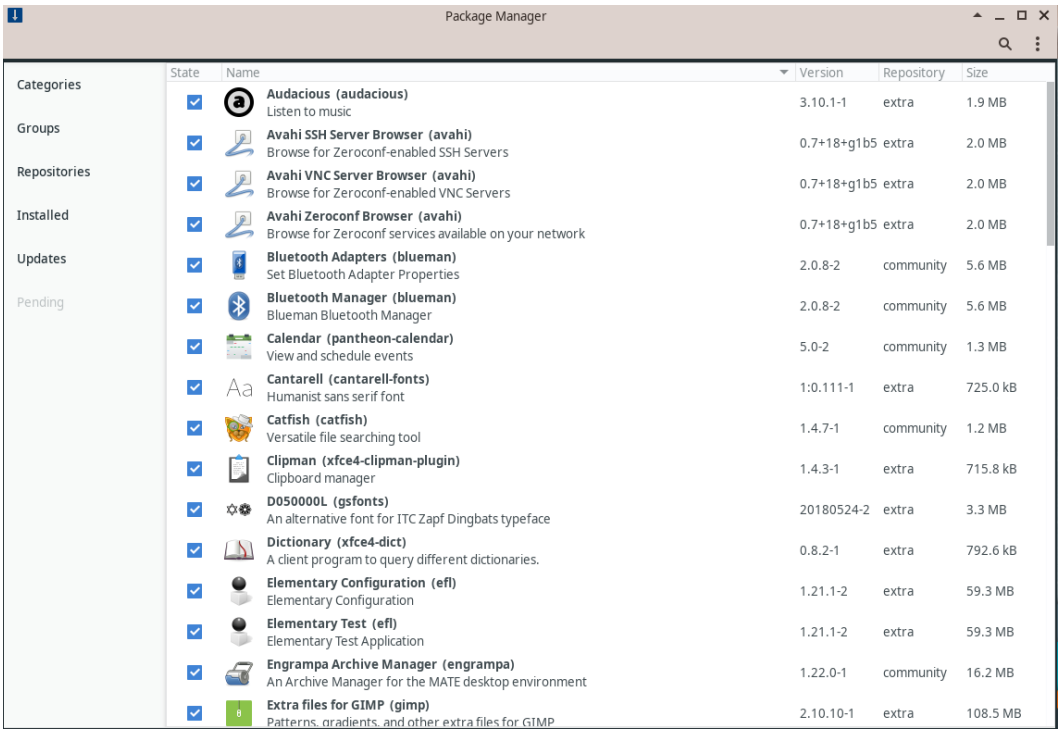

Рисунок 1.3 – Приклад інтерфейсу Pamac

Winget (див. рис. 1.4) – пакетний менеджер, який був розроблений компанією Microsoft для Windows 10 і 11 [5]. Вимагає від програмного забезпечення бути у форматі UWP, чим значно зменшує кількість доступного забезпечення.

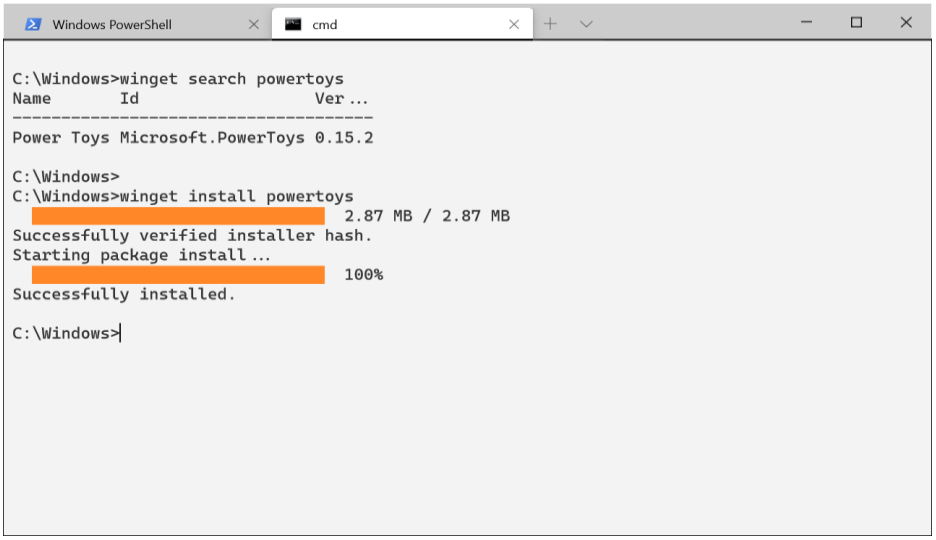

Рисунок 1.4 – Приклад команди Winget

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльні характеристики пакетних менеджерів

Критерії	Apt	Pacman	Pamac	Winget	Gacman
Графічна оболонка	0	0	1	0	1
Сторонні репозиторії	1	1	0	0	1
Категорії програмного забезпечення	0	0	1	0	1
Універсальність	0	0	0	0	1
Рейтинги	1	0	1	0	0
Загалом	2	1	3	0	4

Відповідно до таблиці з порівняльними характеристиками розробка власного пакетного менеджера є доцільною. Отриманий пакетний менеджер може позбавити користувача від недоліків, які мають наявні рішення та полегшити пошук необхідного програмного забезпечення.

1.3 Аналіз методів розв’язання поставленої задачі

Для розв’язання поставленої задачі існує декілька рішень. Створення пакетного менеджера, який буде специфічним для ножної операційної системи та відповідного графічного інтерфейсу. Це дозволить більш детальні налаштування. Недоліком є складність даного процесу та необхідність налагодження складних етапів тестування та додаткових витрат часу.

Кращим і швидшим буде створення інтерфейсу користувача за допомогою інструментів, які дозволять використовувати його на різних операційних системах, а встановлення програм виконувати за допомогою сценаріїв, які залежатимуть від платформи на якій працює користувач.

Врахувавши переваги й недоліки цих методів було вирішено використати метод розробки графічного інтерфейсу користувача та встановлення програмного забезпечення за допомогою сценаріїв.

1.4 Постановка задач для пакетного менеджера

Проаналізувавши переваги та недоліки існуючих пакетних менеджерів було сформульовані такі задачі бакалаврської дипломної роботи:

- провести аналіз існуючих менеджерів інсталяційних пакетів;
- розробити метод використання веб серверу для зберігання даних про програмне забезпечення без прив’язки до операційної системи;
- розробити метод встановлення програмного забезпечення за допомогою сценаріїв;
- розробити графічний інтерфейс користувача, що працюватиме на різних операційних системах;
- розробити веб сервер за архітектурою Rest Api;
- провести експериментальні дослідження розроблених засобів встановлення програмного забезпечення.

1.5 Висновки

У першому розділі розглянуто стан пакетних менеджерів, таких як: Apt, Pacman, Pamac та Winget. В результаті проведеного аналізу виявлено такі недоліки: необхідність навчання для роботи із пакетним менеджером, не можливість використовувати пакетний менеджер для різних операційних систем, складність пошуку нового програмного забезпечення. Отже, доведено доцільність розробки системи створення інсталяційних пакетів. Визначено перелік задач, які необхідно розв’язати для розробки системи створення інсталяційних пакетів.

2. РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Вибір мови програмування

Для розробки системи створення інсталяційних пакетів необхідно обрати мову програмування, що дозволить написати графічний інтерфейс користувача, які будуть працювати на багатьох операційних системах.

Провівши аналіз мов програмування було встановлено, що мова Python має найбільшу кількість переваг для написання графічного інтерфейсу користувача, що розглянуто в табл. 2.1.

Таблиця 2.1 – Порівняльний аналіз мов програмування

	Python	C++	Rust	C#
Крос-платформовість	1	0	1	0
Простота	1	0	0	1
Збирач сміття	1	0	0	1
Асинхронність	1	1	0	1
Сумарний коефіцієнт	4	1	1	3

Використання мови програмування Python дозволить за допомогою бібліотеки PyQt створити графічний інтерфейс користувача, що працюватиме не залежно від операційної системи користувача та збільшить швидкість розробки через свою простоту.

Для створення розмітки інтерфейсу доцільно використовувати QtCreator, який дозволить у графічному режимі створити інтерфейс користувача.

Отже для написання графічного інтерфейсу користувача буде використано мову програмування Python, та середовище розробки QtCreator.

2.2 Аналіз даних

Програмне забезпечення взаємодіє з різними даними під час своєї роботи. Головне завдання будь-якого програмного додатку є сприйняття цих даних й їх обробка в залежності від потреб і кваліфікації особи, що працює з програмою в даний момент часу. Зазвичай програма приймає введення даних у певному визначеному форматі, оброблює їх та за необхідності виводить для користувача у зрозумілій для нього формі у повній відповідності до поставленої на неї задачі.

Одним із найпопулярніших шляхів клієнт-серверної взаємодії є Rest Api - це архітектурний стиль програмного забезпечення, який був створений для керівництва проектуванням і розробкою архітектури для інтернету. REST визначає набір обмежень для того, як повинна вести себе архітектура системи. Головна особливість даної архітектури полягає у тому, що веб сервер не зберігає данні про стан користувача. Кожний запит до веб серверу є достатнім і самостійним.

Він визначає, що спілкування має проходити шляхом Http повідомлень у форматі Json.

HTTP повідомлення - це обмін даними між сервером та клієнтом. Є два типи повідомлень: запити, що надсилаються клієнтом, ініціювати реакцію з боку сервера, і відповіді від сервера.

Серед методів Http повідомлень є GET, POST, PUT, DELETE, HEAD, CONNECT, OPTIONS, TRACE, PATCH [6].

GET - запитує певний ресурс. Запити, які виконують за допомогою цього методу призначені лише для зчитування даних з серверу. Також цей метод може використовувати route та query для отримання даних. Не може мати тіла запиту.

POST – використовується для надсилання сутностей до певного ресурсу. Часто викликає зміну стану або якісь побічні ефекти на сервері. Відрізняється від методу PUT тим, що виклик його має різний ефекти на відміну від PUT, виклик якого завжди має однаковий ефект. Зазвичай надсилається за допомогою HTML form.

PUT – замінює всі поточні уявлення ресурсу даними запиту. Якщо існуючого ресурсу не існує, то його буде створено, а веб сервер повідомить про це за допомогою спеціального коду.

DELETE – видаляє вказаний ресурс. При виклику цього методу його результат завжди буде однаковим, що робить його схожим з методом PUT.

HEAD – метод для отримання заголовків, які були б отримані, якщо замість методу HEAD був використаний метод GET. Зазвичай використовується для прогнозування отриманого результат.

CONNECT – використовується для з'єднання з сайтами, що використовують HTTPS з'єднання.

TRACE – метод, що використовується для перевірки з'єднання з ресурсу і надає інформацію для налагодження.

PATCH – метод, що застосовується для часткового оновлення даних, що представлені на сервері.

JSON – це незалежний від мови формат даних. Орієнтований на зручність читання людиною. Є найбільш поширеним способом передачі даних у електронному обміну даними. Він був отриманий на основі JavaScript, але багато сучасних мов програмування включають код для генерації та аналізу даних у форматі JSON.

2.3 Розробка графічної схеми інтерфейсу програмного додатку

При розробці графічного інтерфейсу користувача було використано невелику кількість графічних елементів, та елементів керування. Для вирішення поставленої задачі необхідно не відволікати користувача від вибору програмного забезпечення, тож таке рішення є цілком доцільним.

Створимо схематичне зображення головного вікна програми (див. рис. 2.1), яке мусить містити список з програмним забезпечення, елементи керування програмним забезпеченням, вибір категорії та пошук програмного забезпечення.

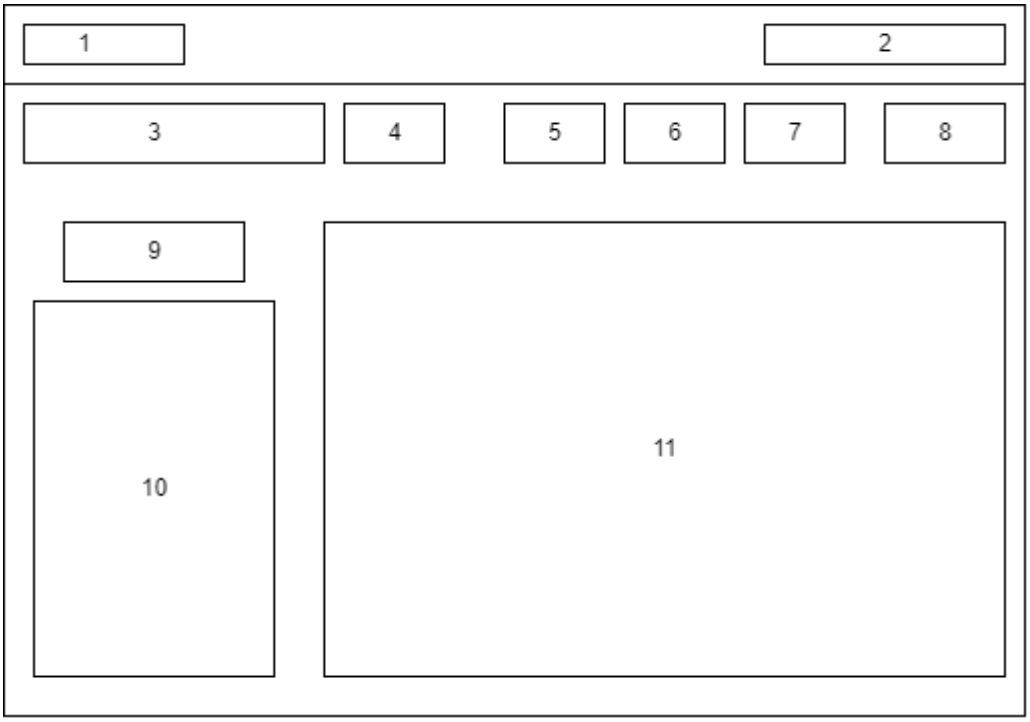

Рисунок 2.1 – Графічна схема головного вікна додатку

Елементи інтерфейсу:

1) назва програмного додатку;
2) кнопки керування програмою;
3) поле вводу для пошуку програмного забезпечення;
4) кнопка для пошуку програмного забезпечення;
5) кнопка оновлення вибраного програмного забезпечення;
6) кнопка встановлення вибраного програмного забезпечення;
7) кнопка видалення вибраного програмного забезпечення;
8) кнопка оновлення списку;
9) кнопка показу програмного забезпечення у вибраній категорії;
10) список категорій програмного забезпечення;
11) список програмного забезпечення.

Оскільки програмний продукт має працювати на багатьох операційних система для розробки інтерфейсу необхідно вибрати засоби, що дозволять створити графічний інтерфейс без необхідності додаткових зусиль по переносу програмного продукту.

2.4 Розробка веб серверу для оновлення даних додатку

Для того, щоби користувач міг отримувати актуальні списки програмного забезпечення без необхідності випуску нової версії програми, яка міститиме ці списки було прийнято рішення про розробку веб серверу, яке дає змогу отримувати найсвіжішу інформацію про програмне забезпечення та інструкції по встановленню програмного забезпечення на пристрої користувача.

Розробка веб серверу була проведена у середовищі розробки Visual Studio 2022 за допомогою мови C# та бібліотеки Asp.Net.

C# - це мова програмування загального призначення, яка використовує платформу .Net та є мовою з суворою типізацію та компіляцією під час виконання. Мова отримала міжнародний стандарт від ECMA, а саме ECMA-334, а також ISO/IEC 23270, що встановили наступні принципи [7]:

- мова має бути простою, сучасною та об’єктно орієнтованою;
- вона має бути суворо типізованою, підтримувати збирач сміття, не давати використовувати видаленні змінні;
- ефективність програміста і надійність програмного забезпечення є важливими цілями;
- можливість виконання на різних платформах.

Asp.Net - це відкритий фреймворк для розробки веб додатків, що було розроблено компанією Microsoft. Підтримує велику кількість програмних моделей та дозволяє створювати веб додатки, які будуть легко масштабовані та легкі у використанні та супроводженні [8].

Підтримує такі моделі програмування:

– Asp.Net Web Form – фреймворк для розробки веб серверу, який оброблятиме сторінки сайту;

– Asp.Net MVC – дозволяє будувати веб сторінки за допомогою model-view-controller патерна проєктування;

– Asp.Net Web Pages – синтаксис для додавання динамічного коду у розмітці HTML;

– Asp.Net Web API – фреймворк для створення веб серверу для основі RestAPI.

Для розробки веб серверу було використано Asp.Net Web API.

Спершу приступи до шляхів по яким користувач буде отримувати список категорій програмного забезпечення (див. рис. 2.3).

```
[HttpGet("TypeList")]
[ProducesResponseType(typeof(List<TypeDTO>), 200)]
public async Task<IActionResult> TypeList()
{
    try
    {
        var result = await _manager.GetTypeListAsync();
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}
```

Рисунок 2.2 – Метод отримання категорій програмного забезпечення

Було використано Http метод Get, який призначений лише для отримання даних від веб-серверу.

Наступним шляхом є отримання списків програмного забезпечення з необхідною інформацією в залежності від вибраної категорії (див. рис. 2.4).

```
[HttpGet("TypeList")]
[ProducesResponseType(typeof(List<TypeDTO>), 200)]
public async Task<IActionResult> TypeList()
{
    try
    {
        var result = await _manager.GetTypeListAsync();
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}
```

Рисунок 2.3 – Метод отримання списку програмного забезпечення

Було використано Http метод Get, який приймає аргумент у вигляді номера категорії в якій необхідно отримати програмне забезпечення.

Наступними кроками необхідно добавити шляхи по яким клієнтська програма буде отримувати сценарії для встановлення (див. рис. 2.4), оновлення (див. рис. 2.5) та видалення (див. рис. 2.6) програмного забезпечення.

Для розробки методу отримання сценаріїв встановлення було використано Http метод GET, який не містить тіла запиту та призначений для отримання даних у сервері.

Метод повертає данні у форматі строки, яка відповідає сценарію встановлення.

Метод приймає два аргументи:

– osTypeId – число, що вказує на операційну систему, яку використовує користувач на своєму пристрої;

– softwareId – число, що вказує на програмне забезпечення, яке користувач хоче встановити на свій пристрій.

```
[HttpGet("Install/OS/{osTypeId}/Soft/{softwareId}")]
[ProducesResponseType(typeof(string), 200)]
public async Task<IActionResult> InstallScript(
    [FromRoute] int osTypeId, [FromRoute] int softwareId)
{
    try
    {
        string result = await _manager.GetScript(
            ScriptType.Install, softwareId, osTypeId);
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}
```

Рисунок 2.4 – Метод отримання сценаріїв встановлення

Для розробки методу отримання сценаріїв оновлення було використано Http метод GET, який не містить тіла запиту та призначений для отримання даних у сервері.

Метод повертає данні у форматі строки, яка відповідає сценарію встановлення.

Також він має механізми захисту від помилок та взаємодію з клієнтським додатком, що являє собою графічний інтерфейс користувача за допомогою кодів відповідей Http.

При помилці зчитування з бази даних, через некоректність надісланих даних веб сервер поверне відповідь з кодом «Bad Request».

При не визначеній помилці, веб сервер поверне відповідь з кодом «nternal Server Error».

```
[HttpGet("Update/OS/{osTypeId}/Soft/{softwareId}")]
[ProducesResponseType(typeof(string), 200)]
public async Task<IActionResult> UpdateScript(
    [FromRoute] int osTypeId, [FromRoute] int softwareId)
{
    try
    {
        string result = await _manager.GetScript(
            ScriptType.Update, softwareId, osTypeId);
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}
```

Рисунок 2.5 – Метод отримання сценаріїв оновлення

```
[HttpGet("Delete/OS/{osTypeId}/Soft/{softwareId}")]
[ProducesResponseType(typeof(string), 200)]
public async Task<IActionResult> DeleteScript(
    [FromRoute] int osTypeId, [FromRoute] int softwareId)
{
    try
    {
        string result = await _manager.GetScript(
            ScriptType.Delete, softwareId, osTypeId);
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}
```

Рисунок 2.6 – Метод отримання сценаріїв видалення

2.5 Розробка алгоритмів роботи додатку

Загальний алгоритм роботи додатку має такі кроки: отримання вхідних даних, програма оброблює введені данні, а потім починає процес встановлення, видалення або оновлення програмного забезпечення в залежності від дій користувача. В кінці програма в залежності від результату попередніх дій виводить результат. Блок-схема алгоритму зображена на рисунку 2.8.

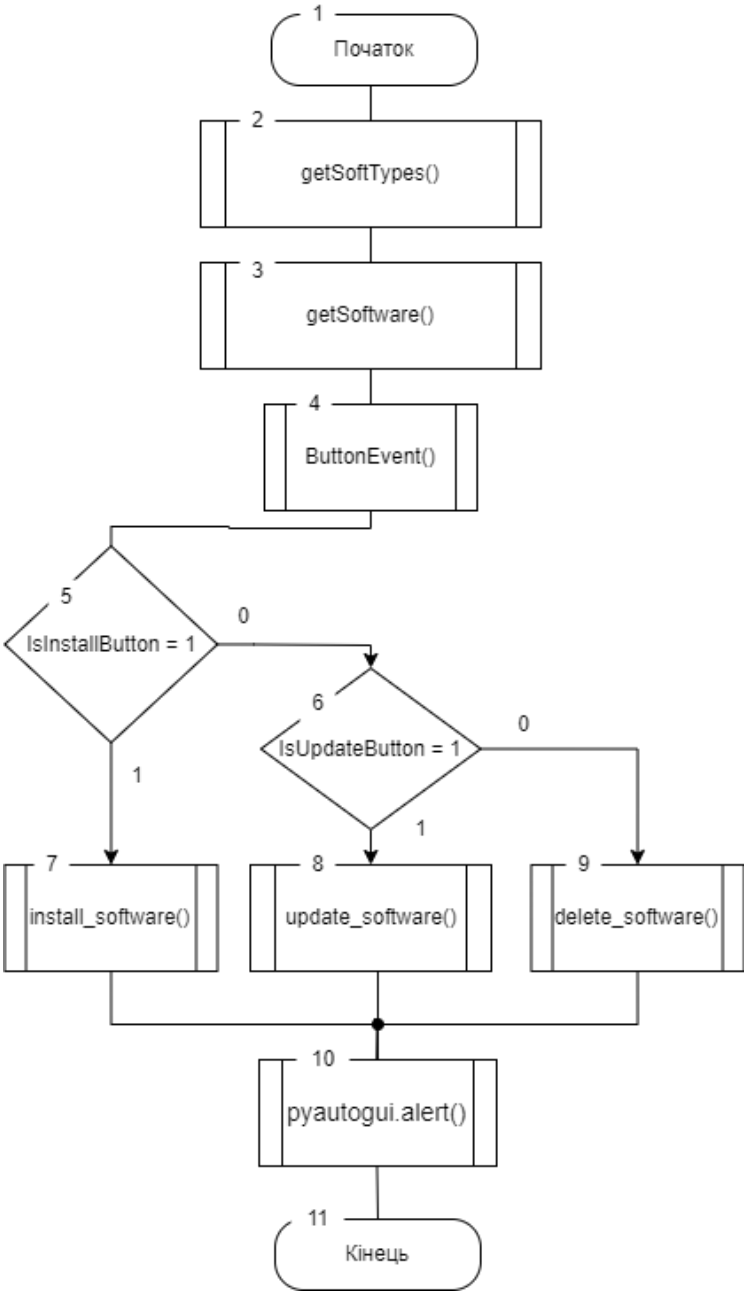

Рисунок 2.8 – Блок-схема загального алгоритму роботи додатку

Наступним кроком є розробка алгоритму отримання списку категорій, який буде отримувати актуальні данні з веб серверу у форматі json файла з усіма необхідними даними для користувача.

Рисунок 2.10 – Блок-схема оновлення списку категорій

Також потрібно забезпечити оновлення списків програмного забезпечення за допомогою шляхів з веб-серверу для того, щоби користувач міг переглянути актуальний список програмного забезпечення, яке доступне йому.

Рисунок 2.9 – Блок схема оновлення списків програмного забезпечення

Фінальним етапом є розробка алгоритмів по встановленню (див. рис. 2.11), оновленню (див. рис. 2.12) та видаленню (див. рис. 2.13) програмного забезпечення.

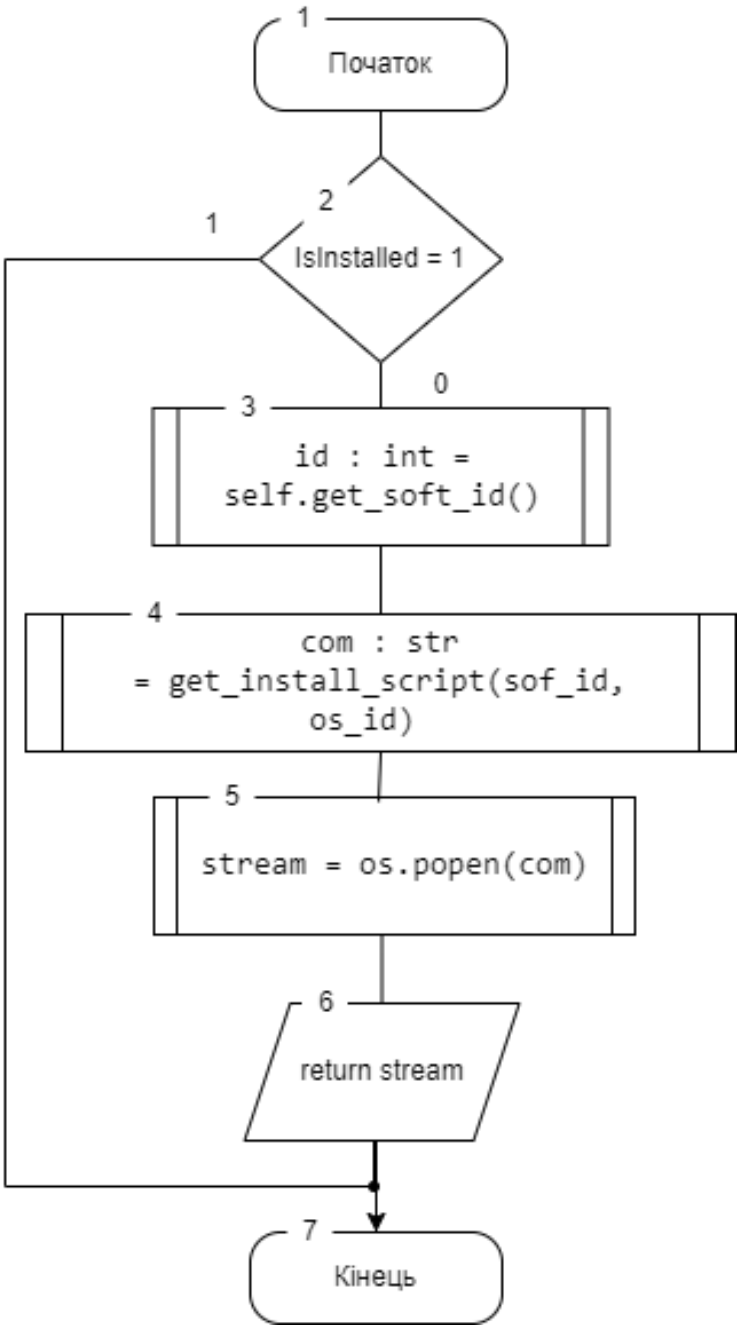

Рисунок 2.11 – Блок схема встановлення програмного забезпечення

Даний алгоритм (див. рис. 2.11) перевіряє чи вибране програм не забезпечення встановлене і робить запит до веб серверу, який повертає інструкції для встановлення, а клієнтська програма встановлює програмне забезпечення.

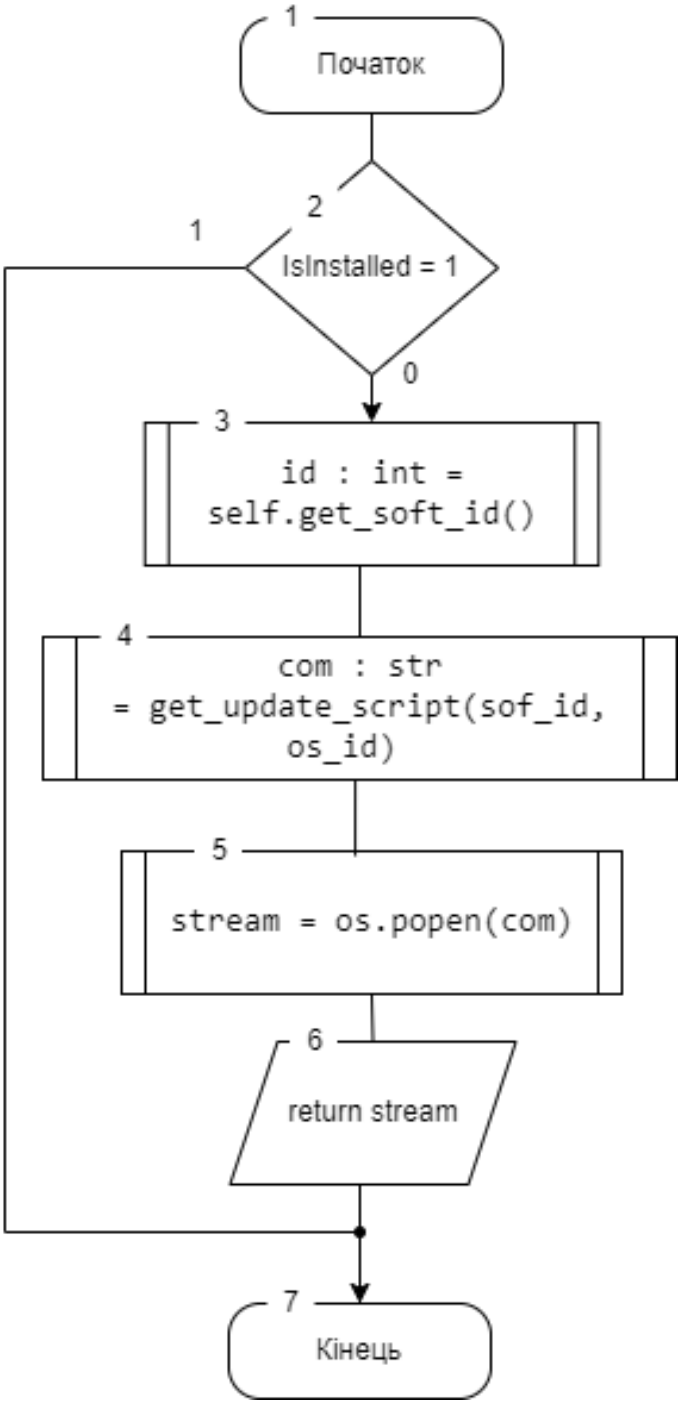

Рисунок 2.12 – Блок схема оновлення програмного забезпечення

Наступний алгоритм (див. рис. 2.12) перевіряє чи встановлене програмне забезпечення на пристрої користувача. Після цього клієнтська програма робить запрос до веб серверу, щоби отримати інструкції для оновлення програмного забезпечення.

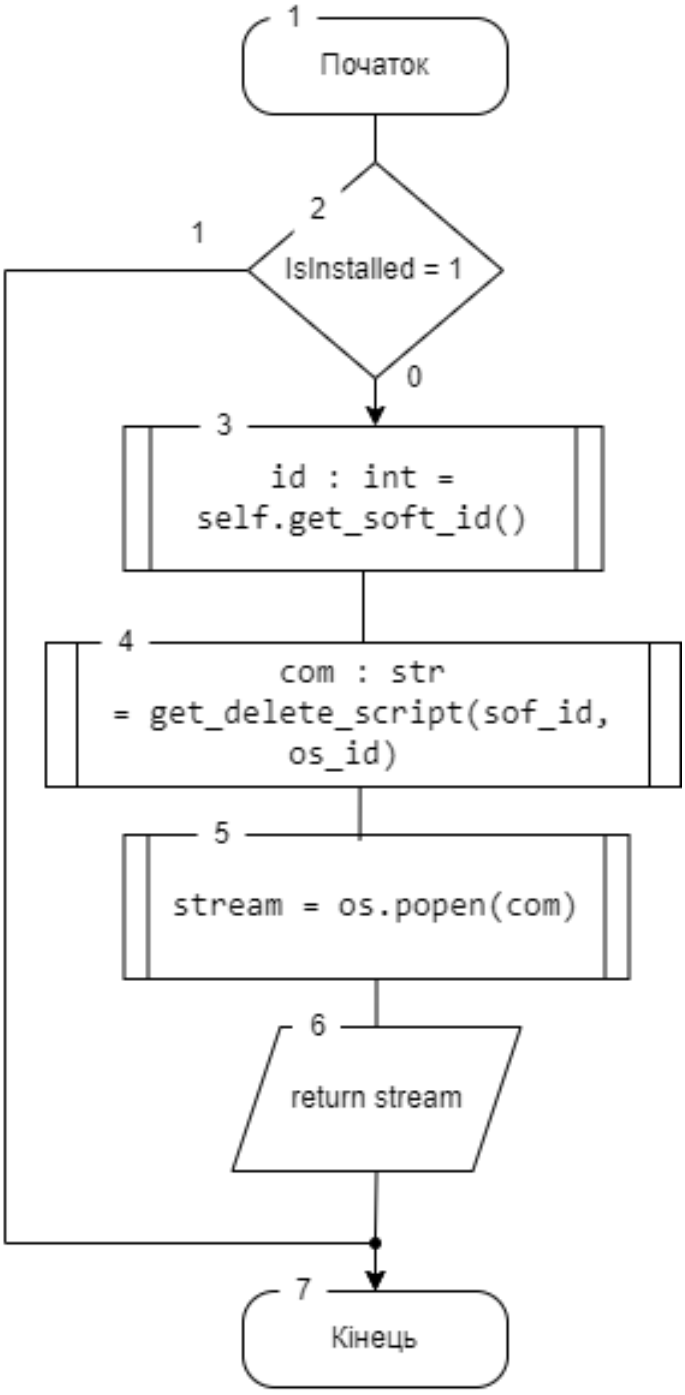

Рисунок 2.13 – Блок схема видалення програмного забезпечення

Зображений алгоритм (див. рис. 2.13) перевіряє чи встановлене програмне забезпечення на пристрої користувача і надсилає запит до веб серверу для отримання інструкцій по видаленню програмного забезпечення в залежності від операційної системи користувача.

2.6 Висновки

У другому розділі проведено аналіз вхідних даних та методи взаємодії користувача з програмним застосунком. Розроблено користувацький інтерфейс та алгоритми роботи додатку . Було розроблено базу даних для зберігання сценаріїв та веб сервер. Розроблено основний алгоритм програми та алгоритми оновлення списку категорій, встановлення, оновлення та видалення програмного забезпечення.

3 РОЗРОБКА СИСТЕМИ СТВОРЕННЯ ІНСТАЛЯЦІЙНИХ ПАКЕТІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Вибір правильної технології для реалізації програмного продукту є важливим етапом, який дозволяє визначити майбутній процес розробки програмного продукту та полегшити певні його етапи, або навпаки ускладнити. Необхідно виділити важливі частини програмного продукту та вибрати інструменти, які допоможуть найкраще справитися з виставленими цілями.

Для розробки інтерфейсу користувача необхідно вибрати інструменти, які дозволять створити інтерфейс, який працюватиме на багатьох платформах. Також для розробки веб серверу необхідно мова, що дозволить написати складну систему, яка буде легко масштабованою та дозволятиме написати систему, яка буде легко у супроводженні та бути здатною до модифікацій.

Для написання інтерфейсу було вибрано мову Python з бібліотекою PyQt.

Python – це мова програмування, що підтримує різні парадигми програмування. Об’єктно-орієнтоване та структурне з функціональним програмуванням. Інші парадигми підтримується за допомогою різних розширень.

Мова використовує динамічну типізацію та використовує збирач сміття для керування пам’яттю, який визначає цикл для керування пам’яттю. Також вона використовує пізнє прив’язування, що зв’язує імена методів та змінних лише під час виконання програми.

Його основна філософія узагальнена в документі The Zen of Python, який включає такі афоризми [9], як:

- красиве краще, ніж потворне;
- явне краще, ніж неявне;
- просте краще, ніж складне;
- складне краще, ніж складне;
- читабельність має значення.

PyQt – це безкоштовна бібліотека розробки графічного інтерфейсу користувача, яка дозволяє створювати графічний інтерфейс, який буде працювати на Microsoft Windows, а також на різних варіантах UNIX, включаючи Linux і MacOS. Було розроблена британською фірмою Riverbank Computing [10] та доступна за умовами, які схожі на Qt версії 4.5, що включає і комерційну ліцензію.

Для розмітки інтерфейсу використаємо QtCreator, що дозволить нам створити інтерфейс без необхідності описувати його у коді програми. Файл інтерфейсу представляє собою xml файл з розміткою усіх елементів. Також він включає в себе налагоджувач, конструктор форм та працює не залежно від операційної системи.

Для написання веб серверу було вибрано мову програмування C# з фреймворком Asp.Net.

C# - це одна із найпопулярніших мов програмування, що дотримується об’єктно-орієнтованої парадигми програмування, та чудово підходить до написання систем широкого спектру. Мова має строгу статичну типізацію з синтаксисом близьким до C++ та Java. Дозволяє написати додатки, які легко масштабувати та забезпечує легкість розробки разом з його ефективністю.

3.2 Розробка інтерфейсу користувача

Для розробки було використано мову програмування Python та бібліотеку PyQt, що дозволять зменшити час написання інтерфейсу користувача та розробки крос-платформового інтерфейсу користувача без витрачання значних зусиль на перенос додатку на інші операційні системи.

Для початку за допомогою QtCreator створимо файл у форматі XML, що слугуватиме розміткою інтерфейсу, який буде створений за допомогою редактору форм.

Першим кроком буде створення об’єкта головного вікна (див. рис. 3.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Widget</class>
 <widget class="QWidget" name="Widget">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
```

Рисунок 3.1 – Розмітка головного вікна додатку

Далі необхідно добавити об’єкт, який буде зберігати список категорій програмного забезпечення (див. рис. 3.2).

```
<widget class="QListWidget" name="listWidget">
 <property name="geometry">
  <rect>
   <x>20</x>
   <y>130</y>
   <width>151</width>
   <height>451</height>
  </rect>
 </property>
</widget>
```

Рисунок 3.2 – Розмітка списку категорій

Наступним кроком є створення об’єкта, який буде зберігати список програмного забезпечення, з якого має вибирати користувач (див. рис. 3.3).

```
 <widget class="QListWidget" name="softwareList">
  <property name="geometry">
   <rect>
    <x>190</x>
    <y>80</y>
    <width>601</width>
    <height>511</height>
   </rect>
  </property>
 </widget>
</widget>
<resources/>
<connections/>
```

Рисунок 3.3 – Розмітка списку програмного забезпечення

Також необхідно добавити кнопки керування програмного забезпечення, яке користувач вибере зі списку. На рисунку 3.4 представлені кнопки оновлення, встановлення та видалення відповідно.

```
<widget class="QPushButton" name="updateButton">
 <property name="geometry">
  <rect>
   <x>400</x>
   <y>20</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Update</string>
 </property>
</widget>
<widget class="QPushButton" name="installButton">
 <property name="geometry">
  <rect>
   <x>490</x>
   <y>20</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Install</string>
 </property>
</widget>
<widget class="QPushButton" name="deleteButton">
 <property name="geometry">
  <rect>
   <x>570</x>
   <y>20</y>
   <width>75</width>
   <height>24</height>
  </rect>
 </property>
 <property name="text">
  <string>Delete</string>
 </property>
</widget>
```

Рисунок 3.4 – Розмітка кнопок керування програмним забезпеченням

Результат виконання можна переглянути на рисунку 3.5.

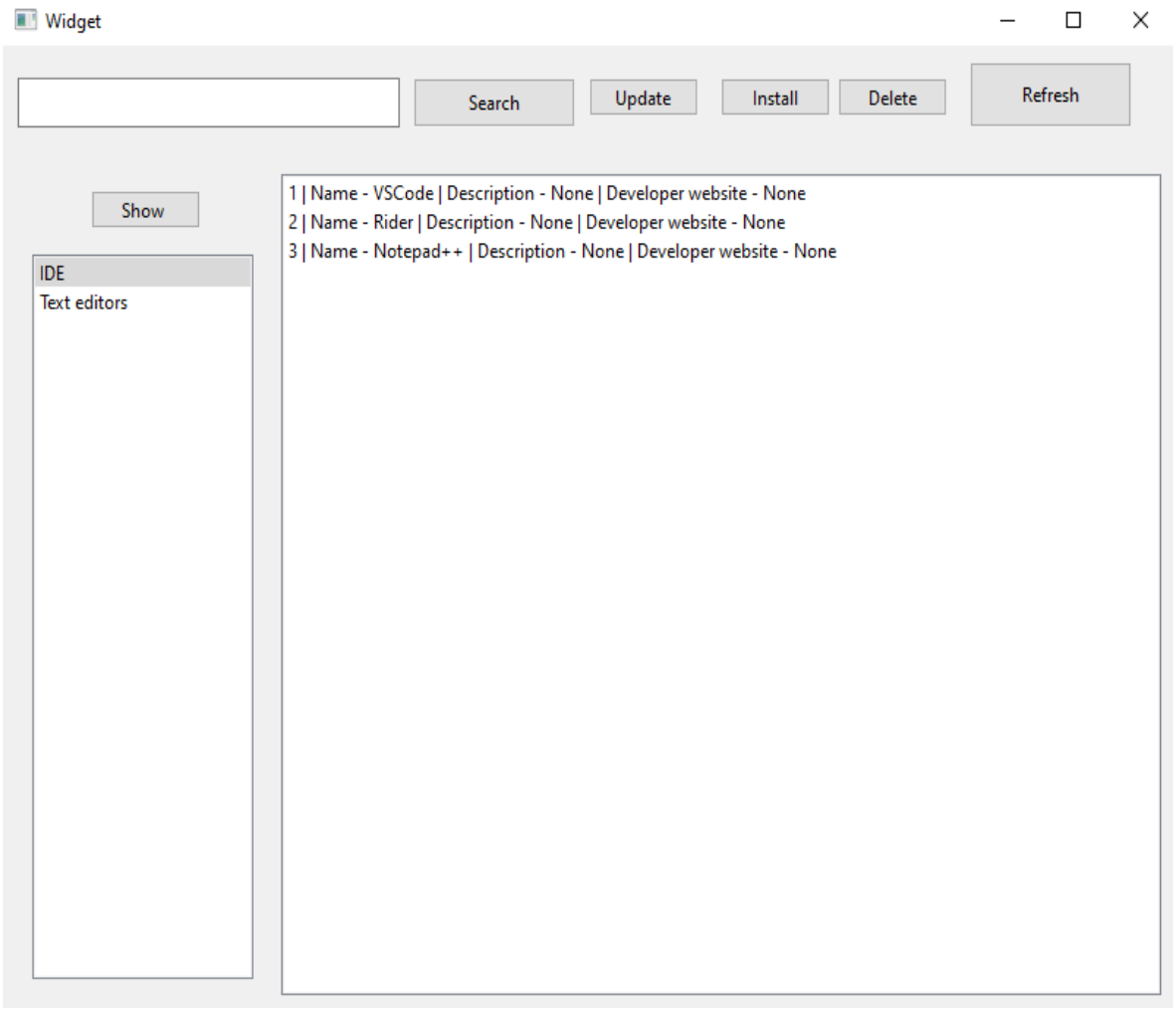

Рисунок 3.5 – Графічний інтерфейс додатку

3.3 Розробка бази даних для зберігання даних про програмне забезпечення

Для розробки бази даних була використана безкоштовна, реляційна база даних під назвою PostgreSQL. Ця база даних забезпечує транзакції, атомарність, стабільність, ізоляцію та цілісність даних. Вона розроблена для використання широким спектром користувачів та може працювати на багатьох операційних системах.

Для створення бази даних було написано Sql сценарій, який дозволяє провести міграцію бази даних на різні системи без використання додаткових засобів. Сценарій створення таблиць наведений на рисунку 3.7.

```
create table if not exists RequestType(
id serial not null unique   primary key,
name text not null,
description text not null
);

create table if not exists users(
id serial not null unique primary key,
email text not null unique,
password text not null,
phoneNumber text unique
);

create table if not exists UserRequests(
id serial not null unique primary key,
requesttypeId int4 not null,
userId int4 not null,
requestDescription text not null
);

create table if not exists softwareType(
id serial not null unique primary key,
typeName text not null unique,
description text
);

create table if not exists software(
id serial not null unique primary key,
typeId int4 not null,
softName text not null,
description text,
webSite text
);
```

Рисунок 3.7 – Sql сценарій створення бази даних

Також необхідно добавити ключі, які будуть зв’язувати данні у базі даних, та слугувати обмеженням, що забезпечать їх цілісність. Даний сценарій наведений на рисунку 3.8.

```
alter table userrequests add CONSTRAINT userrequest_userId
foreign key (userid) REFERENCES  users (id) match full;

alter table userrequests add CONSTRAINT userrequest_requestId
foreign key (requesttypeid) REFERENCES  requesttype (id) match full;

alter table software  add CONSTRAINT software_softwareTypeId
foreign key (typeid) REFERENCES  softwareType (id) match full;


create table if not exists operatingsystems(
id serial not null unique primary key,
osName text not null
);

create table if not exists packages(
id serial not null unique primary key,
packageName text not null,
softwareId int4 not null,
osId int4 not null
);

create table if not exists packageManagers(
id serial not null unique primary key,
osid int4 not null unique,
pmName text not null,
installCom text,
deleteCom text,
updateCom text
);

alter table packageManagers add constraint os_packageManager_id
foreign key (osid) references operatingsystems (id) match full;

alter table packages add constraint software_package_idw
foreign key (softwareId) references software (id) match full;

alter table packages add constraint package_os_id
foreign key (osId) references operatingsystems (id) match full;
```

Рисунок 3.8 – Sql сценарій добавлення ключів

Після виконання даних сценаріїв буде створена база даних, ER – діаграма якої наведена на рисунку 3.9.

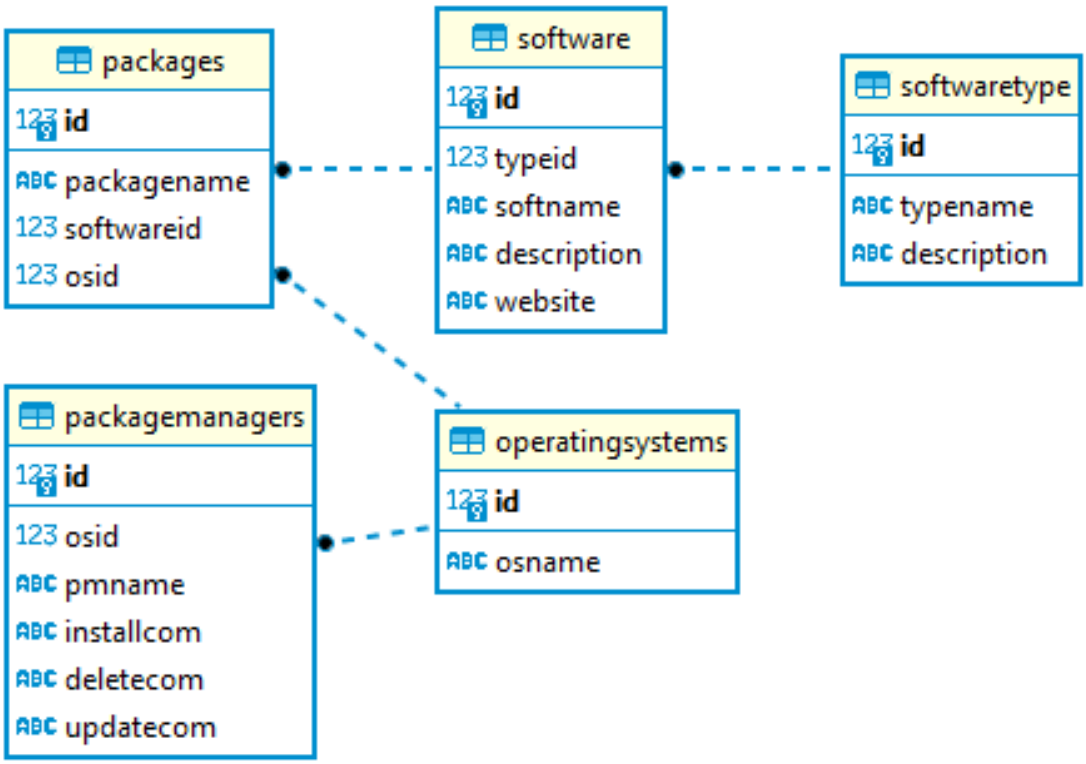

Рисунок 3.9 – ER діаграма бази даних

3.4 Розробка веб серверу для отримання актуальних даних

Для зчитування даних з бази даних за визначеними шляхами за допомогою веб серверу була використана мова програмування C# та фреймворком Asp.Net й бібліотекою Dapper.

Спершу необхідно реалізувати зчитування типів програмного забезпечення, яке не приймає жодних аргументів та зчитування списку програмного забезпечення, яке приймає в якості аргументу номер категорії до якої має належати програмне забезпечення. Дані методи можна переглянути на рисунку 3.10 та рисунку 3.11 відповідно.

```
public async Task<List<SoftwareDTO>> GetListAsync(int typeId)
{
    try
    {
        using var connection = new NpgsqlConnection(_connectionString);
        await connection.OpenAsync();

        string query = $"select id, typeid ,softname as name, description , website from software s where s.\"typeid\" = {typeId};";

        var result = await connection.QueryAsync<SoftwareDTO>(query);

        return result.AsList();
    }
    catch
    {
        return null;
    }
}

public async Task<List<TypeDTO>> GetTypeListAsync()
{
    try
    {
        using var connection = new NpgsqlConnection(_connectionString);
        await connection.OpenAsync();

        string query = "select id as id, typename as name, description  from softwaretype s ;";

        var result = await connection.QueryAsync<TypeDTO>(query);

        return result.AsList();
    }
    catch
    {
        return null;
    }
}
```

Рисунок 3.10 – Зчитування даних категорій і програмного забезпечення

Наступним кроком є реалізація метода, який повертатиме необхідний сценарій необхідний для маніпуляції з програмним забезпечення на пристрої користувача.

```
public async Task<string> GetScript(ScriptType type, int softwareId, int osId)
{
    try
    {
        using var connection = new NpgsqlConnection(_connectionString);
        await connection.OpenAsync();

        var builder = new StringBuilder();

        string searchField = type switch
        {
            ScriptType.Install => "installcom",
            ScriptType.Update => "updatecom",
            ScriptType.Delete => "deletecom",
            _=> throw new InvalidOperationException()
        };

        string queryCommand = $"select {searchField} from packagemanagers o where o.\"osid\" = {osId};";

        string command = await connection.QueryFirstOrDefaultAsync<string>(queryCommand);

        builder.Append(command);

        string queryPackage = $"select packagename from packages p where p.\"osid\" = {osId} and p.\"softwareid\" = {softwareId};";

        string packageName = await connection.QueryFirstOrDefaultAsync<string>(queryPackage);

        builder.Append(packageName);

        return builder.ToString();
    }
    catch
    {
        return null;
    }
}
```

Рисунок 3.11 – Метод повернення сценарія для користувача

3.5 Висновки

У третьому розділі обґрунтовано вибір мов програмування та технологій, які будуть використовуватися при розробці програмного продукту та наведено основні їх переваги. У результаті аналізу обрано мову програмування C# та технологію ASP.NET для написання веб серверу. Для зберігання даних було вибрано PostgreSQL. Для розробки графічного інтерфейсу користувача було обрано мову програмування Python та бібліотеку PyQt.

4 ТЕСТУВАННЯ ДОДАТКУ

4.1 Засоби і методи для тестування

Тестування програмного продукту це процес під час якого перевіряється поведінка програми при різних ситуаціях, які симулюють роботу користувача. Тестування дозволяє отримати об’єктивний й незалежний погляд на додаток, щоби зрозуміти переваги і недоліки його використання [11]. Є декілька технік для тестування:

- перевірка структури і загального дизайну продукту;
- виконання програми з аналізом її поведінки;
- використання інструментів для моніторингу додатку;
- перегляд інфраструктури необхідної, або використовуваної додатком.

Також розрізняють рівні тестування програмного додатку. Найбільш поширеними є:

- блочне тестування, яке перевіряє лише невелику частину коду;
- інтеграційне тестування, перевіряє цілісність системи;
- системне тестування, що перевіряє систему на виконання всіх вимог.

Можливо виділити й підходи до тестування програмного забезпечення.

Статичне тестування – перегляду коду, інспекції зазвичай відносять до статичного тестування. Також до нього відносять різні аналітичні засоби у IDE, які постійно перевіряють код на помилки.

Динамічне тестування – тестування за допомогою test cases, які описують дії, що може виконувати користувач при роботі з програмним додатком.

Дослідницьке тестування – підхід до тестування, яке описують як одночасне навчання, розробка тестів й їх виконання. Воно зосереджено на дослідженні як буде працювати програмний додаток для з’ясовування, як він буде працювати у різних ситуаціях.

Тестування білої коробки – підхід тестування, коли тестується внутрішня структура додатку.

Тестування чорної коробки – підхід до тестування, коли тестування, коли тестується функціонал програмного додатку, без необхідності знання того, як саме цей додаток виконує те чи інше завдання.

Тестування з порівнянням виводу – тестування при якому порівнюється очікуваний вивід додатку з отриманим.

Для проведення тестування було прийнято рішення використати тестування порівнянням виходу та тестування чорної коробки.

4.2 Тестування інтерфейсу користувача

Тестування графічного інтерфейсу – це процес перевірки інтерфейсу користувача у програмному додатку, щоби перевірити, що він задовольняє постановленим вимогам. Зазвичай це виконується через планування різних ситуацій, які відповідатимуть поставленим вимогам.

При відкритті програмного додатку користувач має бачити список категорій програмного забезпечення разом з кнопками керування вибраним програмним забезпеченням і текстовим полем, яке дозволяє користувача шукати необхідне йому програмне забезпечення. Головне вікно додатку можна побачити на рисунку 4.1.

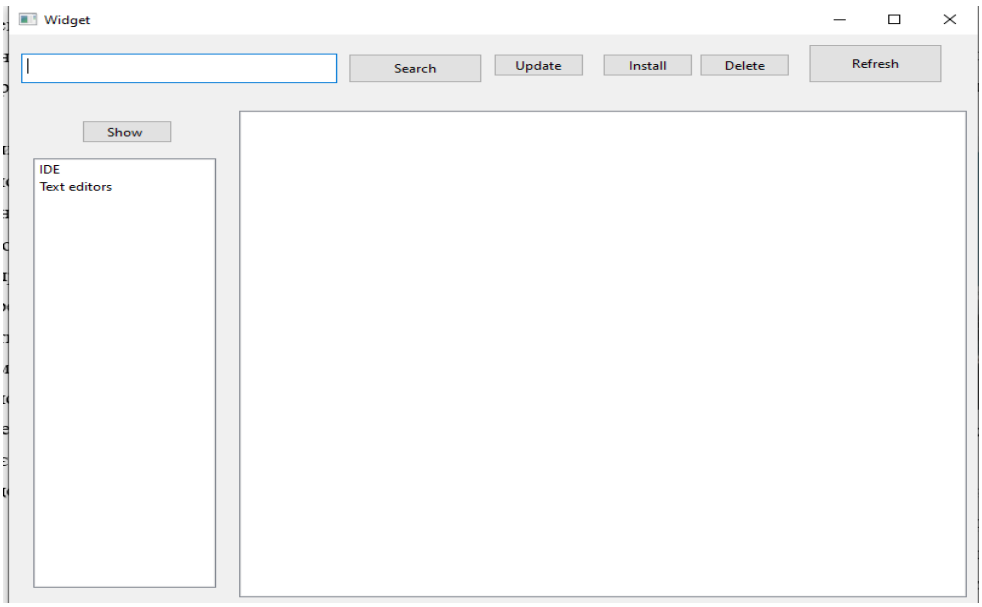

Рисунок 4.1 – Ініціалізаційна форма головного вікна додатку

Для показу програмного забезпечення користувачу необхідно вибрати категорію до якої має належати програмне забезпечення та натиснути кнопку «Show», для відображення в списку програмного забезпечення. Результат виконання наведений на рисунку 4.2.

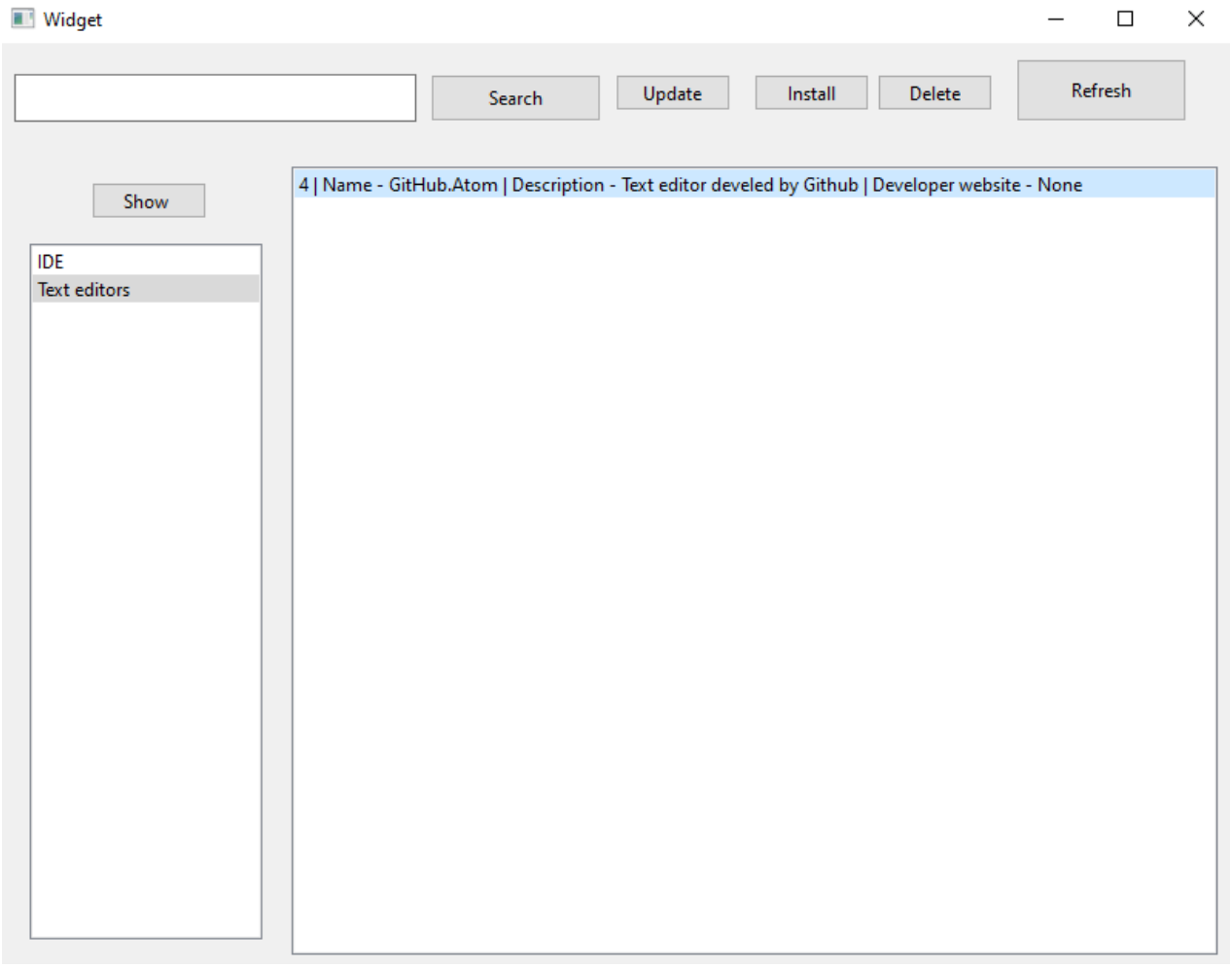

Рисунок 4.2 – Результат відображення списку програмного забезпечення

Для того, щоби встановити програмне забезпечення користувач має вибрати програмне забезпечення зі списку та натиснути кнопку керування «Install». В результаті вибране програмне забезпечення має бути встановлене на комп’ютер користувача, а операційна система повідомити користувача про це. Для перевірки встановимо додаток Notepad++, що являє собою текстовий редактор. Після певного часу користувач бачить повідомлення про успішне встановлення програмного додатку (див. рис. 4.3).

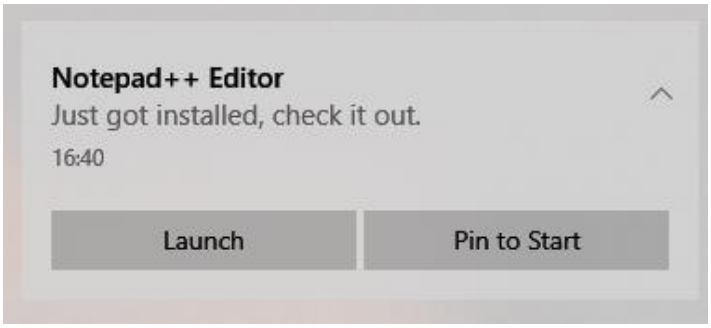

Рисунок 4.3 – Повідомлення про встановлення додатку

Для того, щоби видалити програмне забезпечення користувачу потрібно вибрати його зі списку та натиснути на кнопку «Delete». Для перевірки використаємо додаток Notepad++. Після певного часу користувач побачить повідомлення про видалення програмного додатку (див. рис. 4.4).

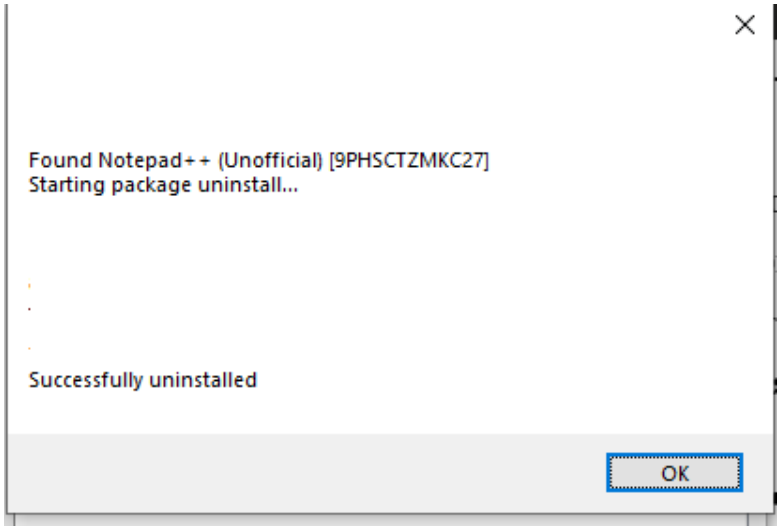

Рисунок 4.4 – Повідомлення про видалення додатку

Також користувач має змогу шукати цікаве йому програмне забезпечення за допомогою строки пошуку. Для цього йому потрібно ввести назву програмного забезпечення на натиснути кнопку «Search».

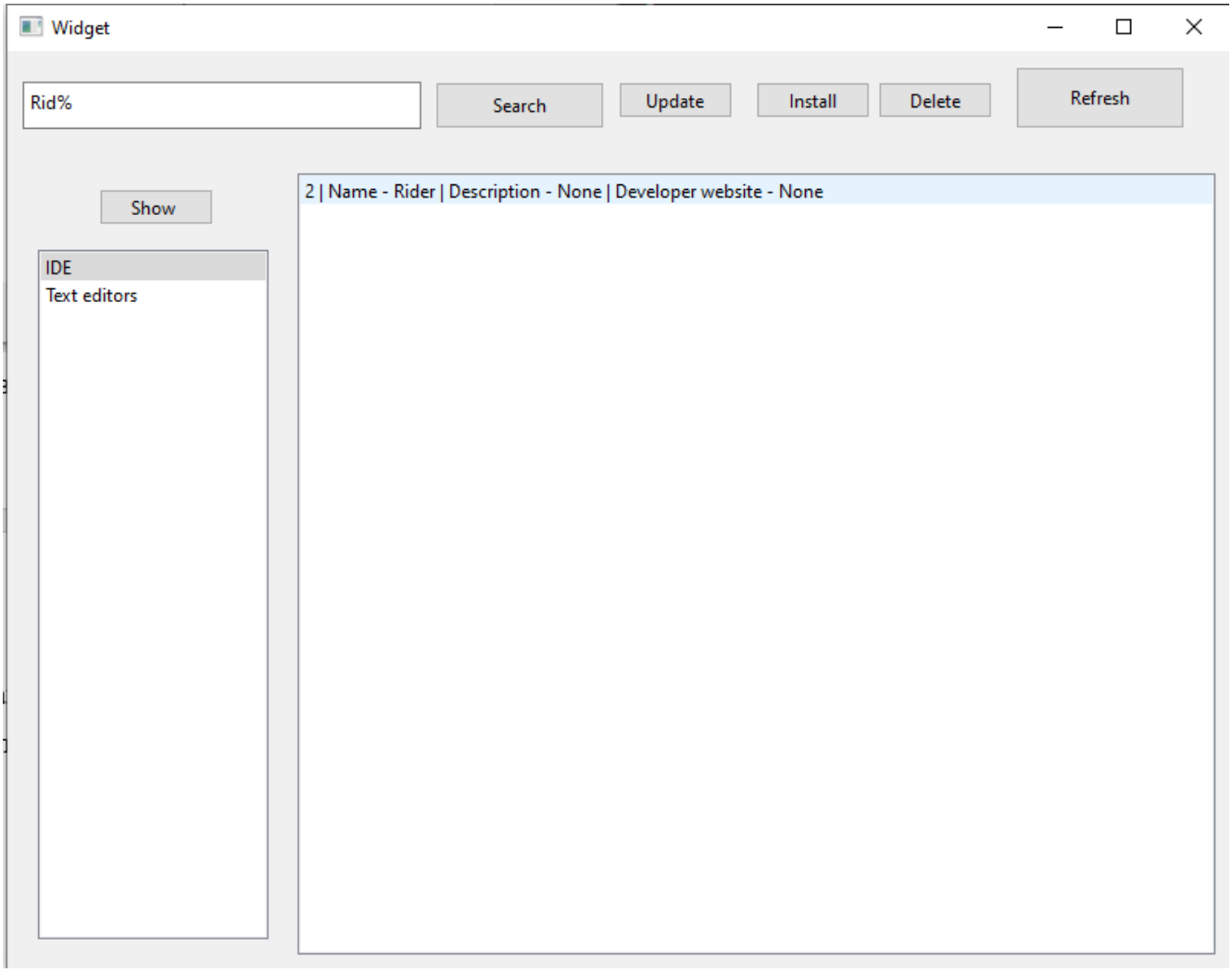

Рисунок 4.5 – Результат пошуку програмного забезпечення

Для оновлення програмного забезпечення необхідно вибрати програмне забезпечення зі списку на натиснути кнопку «Update». Для тесту використаємо додаток «MS PowerToys», який має версію 0.59.0 на пристрої (див. рис. 4.6).

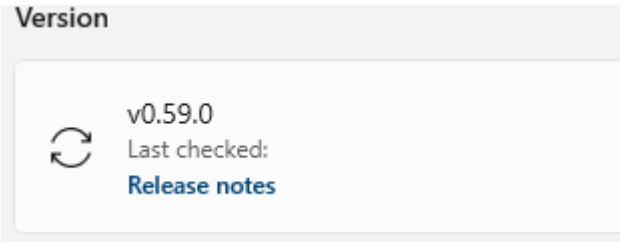

Рисунок 4.6 – Версія додатку до оновлення

При відкритті додатку виберемо категорію «System Utility» та виберемо додаток «MS PowerToys» й оновимо додаток. Після очікування на завантаження програма повідомить користувачу про результат виконання (див. рис. 4.7).

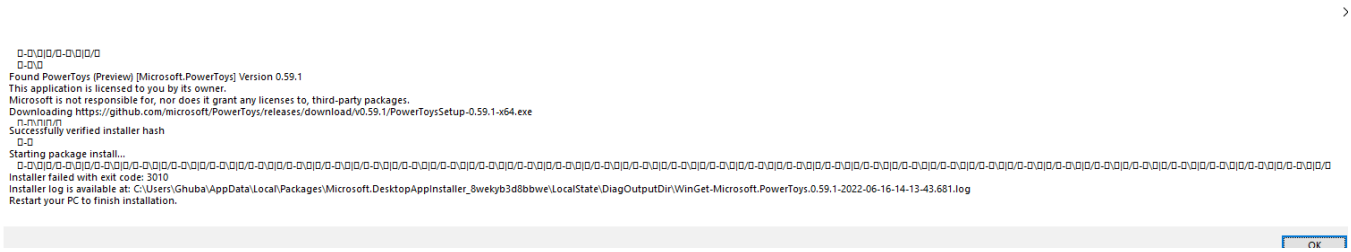

Рисунок 4.7 – Повідомлення про оновлення додатку

Перевіримо версію додатку відкривши його і переконаємося в успішному оновленні додатку (див. рис. 4.8).

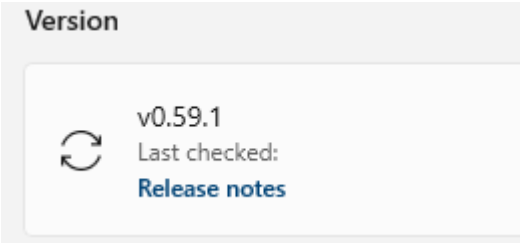

Рисунок 4.8 – Версія додатку після оновлення

Після проведених тестів графічний інтерфейс користувача було встановлено, що додаток виконує всі свої функції і є повністю робочим.

4.3 Тестування веб серверу

Для тестування веб серверу буде використано інтерфейс SwaggerUI, який відображає всі шляхи, які наявні у веб сервері та дозволяє протестувати їх. Спершу необхідно відкрити сторінку SwaggerUI (див. рис. 4.9) та перевірити чи наявні всі шляхи, які розроблені у веб сервері.

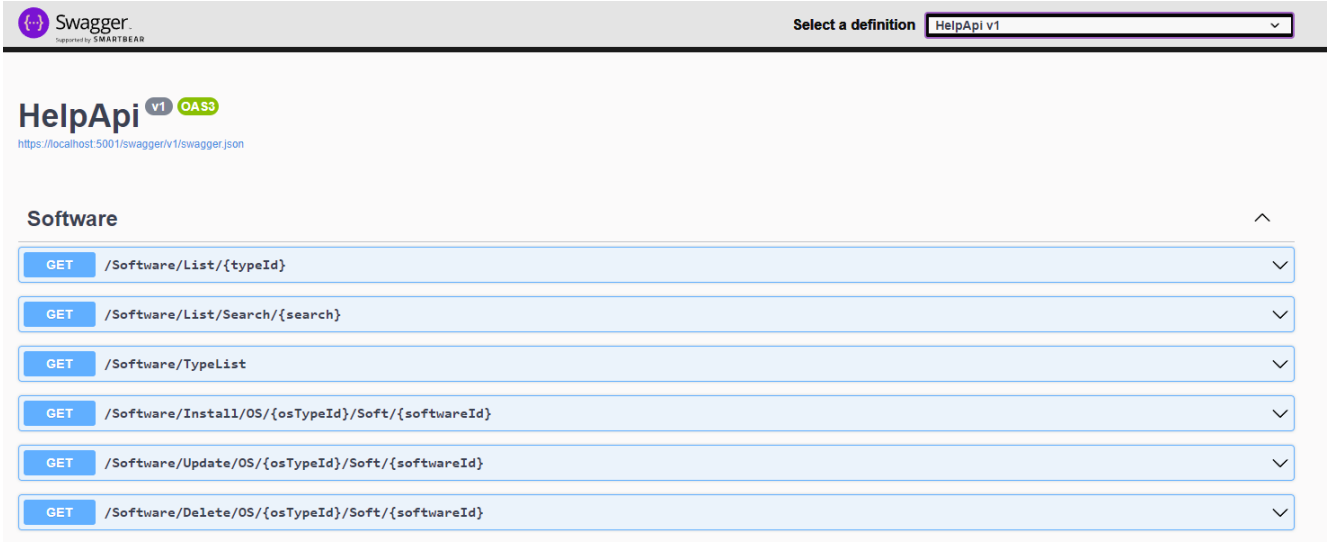

Рисунок 4.9 – Сторінка SwaggerUI

Оскільки всі шляхи наявні, то можна перевірити чи веб сервер поверне наявні категорії програмного забезпечення. Результат виконання можна переглянути на рисунку 4.10.

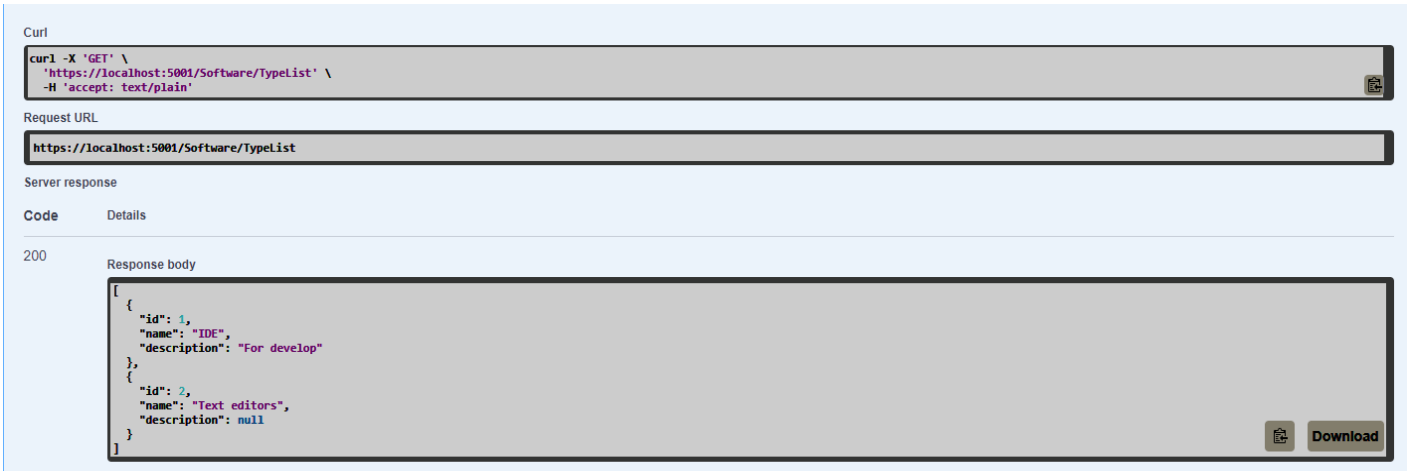

Рисунок 4.10 – Відображення списку категорій від веб серверу

Для того, щоби перевірити вірність результату виконає аналогічний запит до бази даних напряму, та перевіримо його результат (див. рис. 4.11).

	id	name	description
1	1	IDE	For develop
2	2	Text editors	[NULL]

Рисунок 4.11 – Контрольний результат списку категорій

Контрольний результат повністю відповідає отриманому запиту з веб серверу.

Також варто перевірити чи повертає веб сервер програмне забезпечення, що належить до категорії «IDE». Результат виконання можна переглянути на рисунку 4.12.

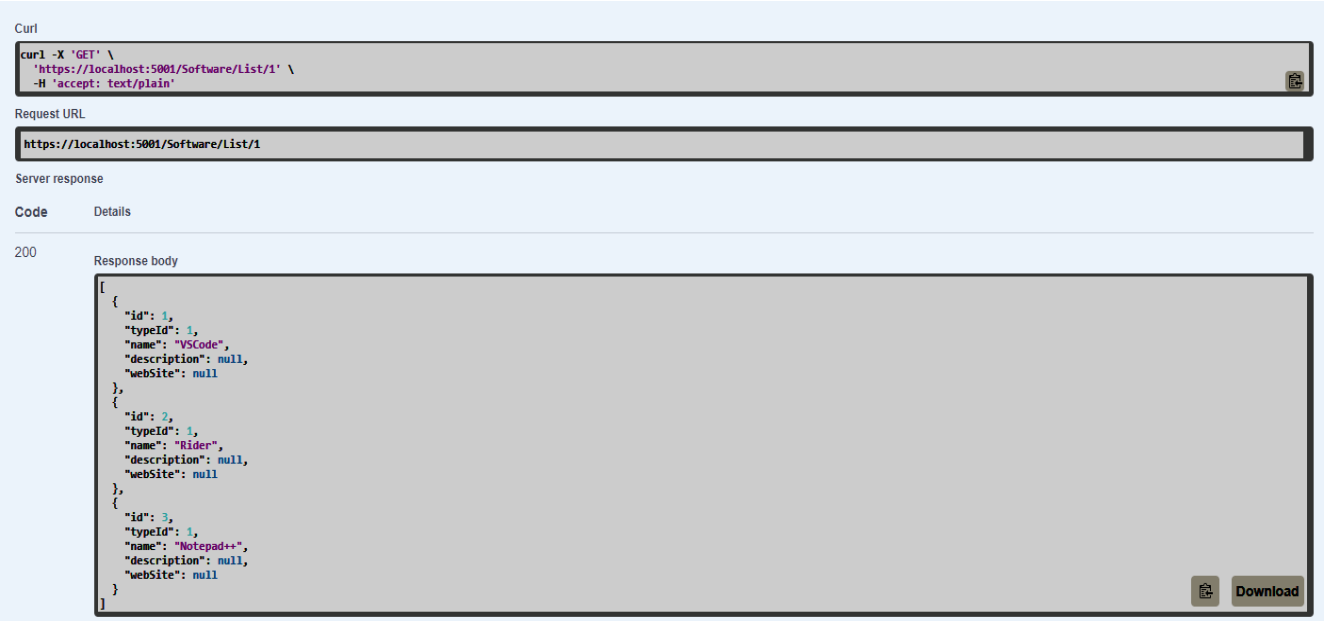

Рисунок 4.12 - Відображення списку програмного забезпечення

Для того, щоби перевірити вірність результату виконає аналогічний запит до бази даних напряму, та перевіримо його результат (див. рис. 4.13).

	id	typeid	name	description	website
1	1	1	VSCode	[NULL]	[NULL]
2	2	1	Rider	[NULL]	[NULL]
3	3	1	Notepad++	[NULL]	[NULL]

Рисунок 4.13 – Контрольний результат списку програмного забезпечення

Контрольний результат, який був отриманий з бази даних повністю відповідає отриманому результату з веб серверу.

Наступним етапом є перевірка повернення відповідного сценарію для вибраного програмного забезпечення. Для цього спробуємо отримати сценарій встановлення для додатку «Notepad++» на операційній системі «Windows». Результат виконання можна переглянути на рисунку 4.14.

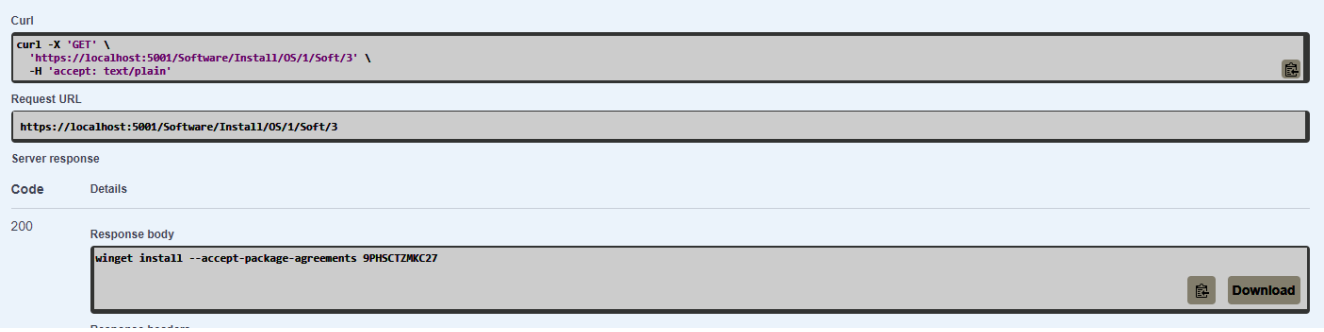

Рисунок 4.14 – Сценарій встановлення для додатку «Notepad++»

Для перевірки команди встановлення спершу необхідно отримати контрольний результат для першої половини команди (див. рис. 4.15).

	installcom
1	winget install --accept-package-agreements

Рисунок 4.15 – Контрольний результат першої половини команди

Як видно контрольний результат і фактичний співпадають, тому необхідно перевірити наступну половину команди, яка відповідає за назву програмного забезпечення (див. рис. 4.16).

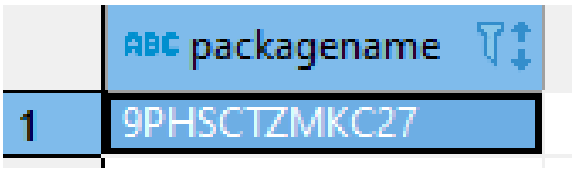

Рисунок 4.16 – Контрольний результат з назвою додатку

Контрольні результати і дані отримані з веб серверу співпадають і тому працездатність роботи метода для отримання сценарію встановлення доведена.

Для тесту отримаємо сценарій видалення програмного забезпечення «Atom». Для перевірки спробуємо отримати команду для операційної системи «Windows» (див. рис. 4.17).

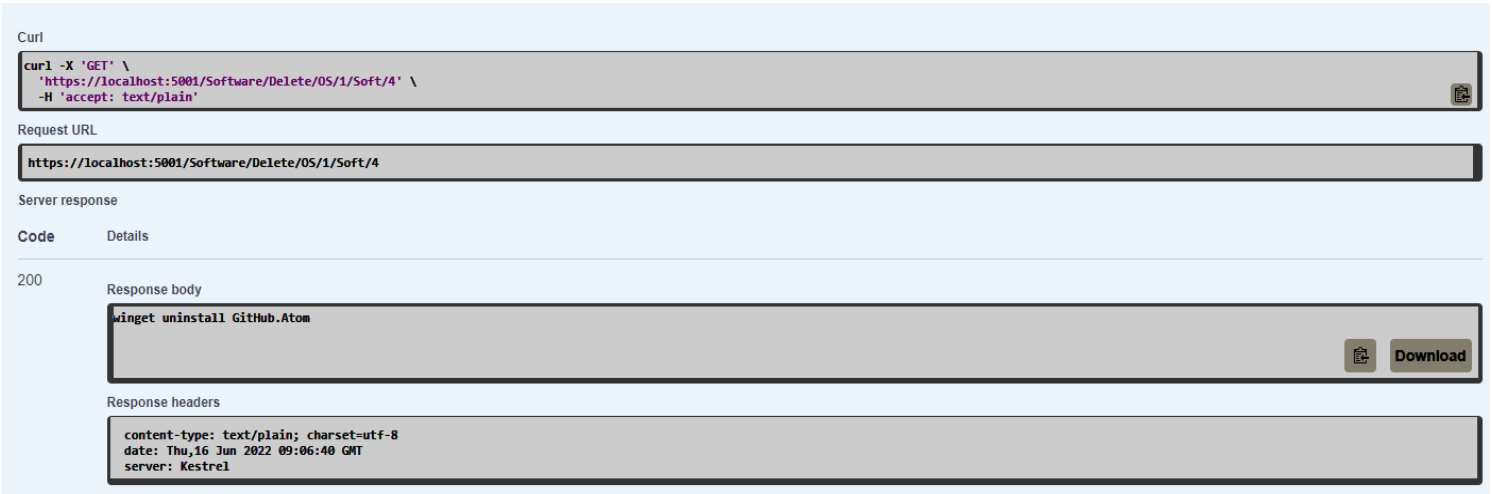

Рисунок 4.17 – Сценарій видалення додатку «Atom»

Для перевірки коректності отриманого результату виконаємо два запити до бази даних, щоби перевірити першу половину команди, що відповідає за інструкцію по виклику команди та другу половину, що відповідає назві програмного забезпечення для цієї операційної системи (див. рис. 4.18).

Рисунок 4.18 – Контрольний результат команди видалення

Наступним кроком є перевірка назви програмного забезпечення для цієї операційної системи (див. рис. 4.19).

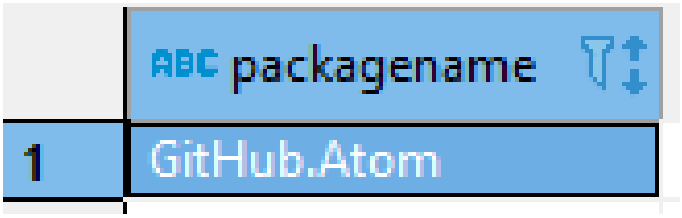

Рисунок 4.19 – Контрольний результат назви додатку

Як видно з результатів перевірки дані, які отримані з веб серверу для видалення програмного забезпечення під назвою «Atom» на операційній системі Windows є коректними тому працездатність методу по отриманню сценарію видалення є доведеною.

Наступним кроком є перевірка команди оновлення програмного забезпечення.

Для підтвердження працездатності методу по отриманню сценаріїв оновлення перевіримо дані для операційної системи «ArchLinux» та програмного забезпечення «Visual Studio Code» (див. рис. 4.20).

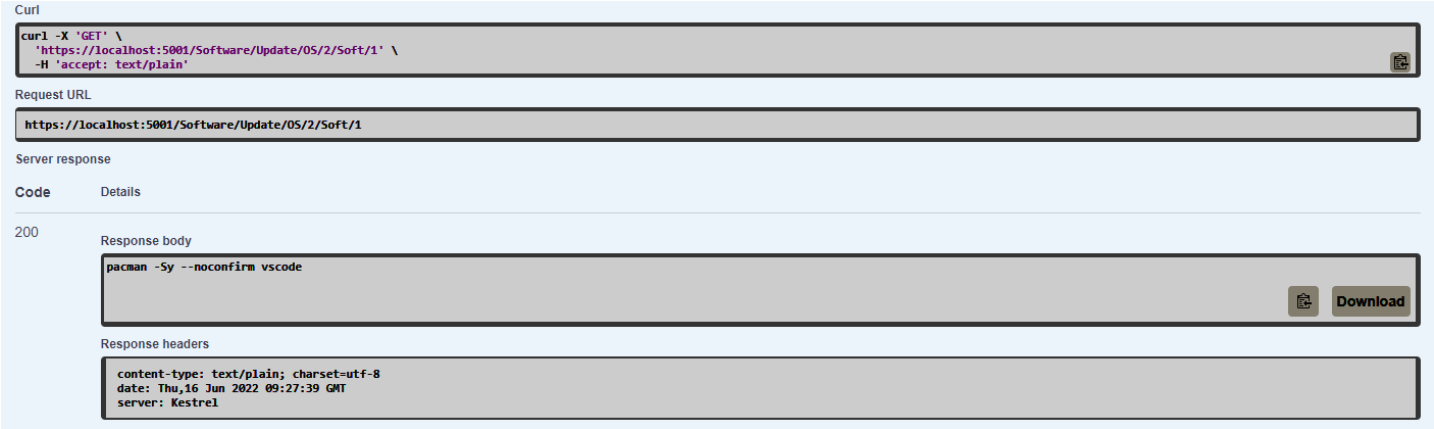

Рисунок 4.20- – Результат отримання сценарію оновлення

Для перевірки працездатності методу по отриманню сценарію оновлення виконаємо два запити до бази даних, щоби перевірити першу половину команди, що відповідає за інструкцію по виклику команди та другу половину, що відповідає назві програмного забезпечення для цієї операційної системи (див. рис. 4.21).

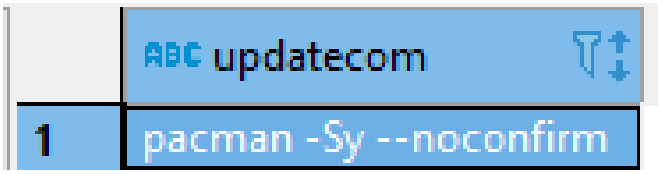

Рисунок 4.21 – Контрольний результат команди оновлення

Наступним кроком є перевірка назви програмного забезпечення для цієї операційної системи (див. рис. 4.22).

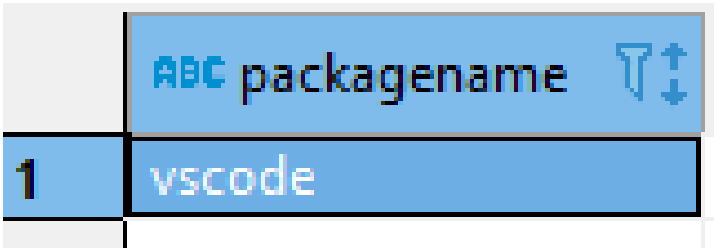

Рисунок 4.22 – Контрольний результат назви додатку

4.3 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації для запуску додатку представлено в табл. 4.1 та табл. 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 1 ГГц
Об’єм оперативної пам’яті	2 ГБ
Місце на жорсткому диску	10 ГБ
Операційна система	Windows 10, або GNU/Linux

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	32-розрядний (x86) або 64-розрядний (x64) процесор з тактовою частотою 2 ГГц
Об’єм оперативної пам’яті	2 ГБ
Місце на жорсткому диску	10 ГБ
Операційна система	Windows 10, або GNU/Linux

Для запуску веб серверу необхідно встановити .Net Runtime, а саме .Net Core 3.1, що слугує платформою для виконання програм, та дозволяє працювати додаткам без необхідності їх переносу на інші операційні системи.

Після цього користувачу необхідно скористатися інтерфейсом командного рядка і виконати команду «dotnet HelpApi.dll». Тепер веб сервер буде активним та готовим приймати запити.

4.5 Висновки

Розглянуто методи та методики тестування програмного забезпечення, обрано такі, що доцільно використати для тестування веб серверу та графічного інтерфейсу користувача.

Було проведено тестування веб серверу для отримання даних з бази даних та графічного інтерфейсу користувача. Також провірено працездатність виконання всіх методів веб серверу, коректність роботи графічного користувача інтерфейсу. Було перевірено валідність даних в залежності від дій користувача та відповідей веб серверу.

Розроблено інструкцію користувача для запуску веб серверу, що буде обслуговати клієнтський інтерфейс користувача.

ВИСНОВКИ

У бакалаврській дипломній роботі розроблено систему інсталяційних пакетів з використанням сценаріїв, що дозволяє покращити стабільність і безвідмовність, підвищити ефективність роботи програмних застосунків і операційної системи.

В першому розділі проведено аналіз предметної області, визначено задачі розробки, проведено аналіз аналогів пакетних менеджерів та визначено основні критерії, по яким повинна відповідати розробка.

У другому розділі проведено аналіз методів і засобів розробки. Проведений аналіз показав пріоритетне використання мови програмування Python для розробки графічного інтерфейсу користувача, що дозволить працювати додатку на більшості наявних операційних системах. У розділі приведено результати аналізу даних, які виникають під час клієнт-серверної взаємодії.

У третьому розділ представлено розробку графічного інтерфейсу додатку, розробку бази даних для зберігання інформації про програмне забезпечення й розробка веб серверу для отримання інформації з бази даних.

У четвертому розділі проведено аналіз методів та засобів тестування програмного забезпечення та вибір актуальних варіантів для перевірки роботи. Проведено тестування, що підтверджує працездатність графічного інтерфейсу додатку та підтверджено вірність отриманих даних з веб серверу. Розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Optimal Package Install/Uninstall Manager [Електронний ресурс] – Режим доступу до ресурсу: https://cseweb.ucsd.edu//~lerner/papers/opium.pdf .
2. Apt Package Manager [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/APT_(software) .
3. Pacman [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.archlinux.org/title/pacman .
4. Pamac [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.manjaro.org/index.php/Pamac .
5. Winget [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/microsoft/winget-cli .
6. HTTP methods [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
7. Standard ECMA-334 навч. посіб. Anders Hejlsberg, Scott Wiltamuth, and Peter Gold – ECMA 2002 – 490 ст.
8. Create a web API with ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу:https://docs.microsoft.com/en-gb/aspnet/core/tutorials/first-web-api?view=aspnetcore-6.0&tabs=visual-studio
9. PEP 20 – The Zen of Python [Електронний ресурс] – Режим доступу до ресурсу: https://peps.python.org/pep-0020/
10. Beginning PyQt: A Hands-on Approach to GUI Programming with PyQt6 навч. посіб. Joshua Willman – Apressб 2022 – 568 ст.
11. Software Testing [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_testing
12. Войтенко Д. О. «Система створення інсталяційний пакетів «Gacman»» / Електронні інформаційні ресурси: створення, використання, доступ. Пам’яті Олексія Петровича Стахова. Збірник матеріалів Міжнародної науково-практичної Інтернет конференції 9-10 листопада 2021 р. – Суми/Вінниця: НІКО/ВНТУ, 2021

ДОДАТКИ

ДОДАТОК А.

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"_31_"___03___2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмної системи створення інсталяційних пакетів» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ д.т.н., доц. О. М. Рейда
"_31_"_____03_____2022 р.
Виконав:
_____студент гр.2ПІ-18б Д. О. Войтенко
"_31_"_____03_____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка системи створення інсталяційних пакетів».

Галузь застосування – операційні системи.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ від 24.03.22р №66 ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є створення системи створення інсталяційних, що дозволить створити пакетний менеджер, який працюватиме не залежно від операційної системи користувача.

Призначення роботи – Розробка системи для створення інсталяційних пакетів

4. Технічні вимоги

Розробка графічного інтерфейсу користувача, що дозволяє переглядати списки програмного забезпечення у залежності від категорій; можливість встановлювати програмне забезпечення; можливість видалення програмного забезпечення; можливість оновлення програмного забезпечення; розробка веб серверу для отримання даних про програмне забезпечення; розробка бази даних для зберігання інформації про програмне забезпечення.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БДР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.2022 – 26.04.2022
2	Розробка архітектури та алгоритмів роботи системи	27.04.2022 – 08.05.2022
3	Вибір середовища та мови розробки	09.05.2022 – 11.05.2022
4	Розробка програмного продукту	12.05.2022 – 23.05.2022
5	Тестування роботи системи	24.05.2022 – 01.06.2022
6	Оформлення матеріалів до захисту БДР	02.06.2022 – 10.06.2022

9. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

ДОДАТОК Б.

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник:

Оригінальність	92%
Схожість	8%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Войтенко Д. О.

Керівник роботи _____ Рейда О. М.

ДОДАТОК В. Лістинг додатку

Додаток В.1 – Лістинг файлу Software.cs

```
using HelpApi.DTOs;
using HelpApi.Enums;
using HelpApi.Helpers;
using HelpApi.Managers;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace HelpApi.Controllers
{
    [Route("Software")]
    public class SoftwareController : Controller
    {
        private Manager _manager;
        public SoftwareController()
        {
            _manager = new Manager();
        }

        [HttpGet("List/{typeId}")]
        [ProducesResponseType(typeof(List<SoftwareDTO>), 200)]
        public async Task<IActionResult> List([FromRoute] int typeId)
        {
            try
            {
```

```
        var result = await _manager.GetListAsync(typeId);
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}

[HttpGet("List/Search/{search}")]
[ProducesResponseType(typeof(List<SoftwareDTO>), 200)]
public async Task<IActionResult> List([FromRoute] string search)
{
    try
    {
        var result = await _manager.SearchSoftware(search);
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}
```

```
[HttpGet("TypeList")]
[ProducesResponseType(typeof(List<TypeDTO>), 200)]
public async Task<IActionResult> TypeList()
{
    try
    {
        var result = await _manager.GetTypeListAsync();
        if (result is null)
            return this.Error(ErrorCodes.DataBaseError);

        return this.Ok(result);
    }
    catch
    {
        return this.Error(ErrorCodes.UnknowError);
    }
}

[HttpGet("Install/OS/{osTypeId}/Soft/{softwareId}")]
[ProducesResponseType(typeof(string), 200)]
public async Task<IActionResult> InstallScript([FromRoute] int osTypeId,
[FromRoute] int softwareId)
{
    try
    {
        string result = await _manager.GetScript(ScriptType.Install, softwareId,
osTypeId);
        if (result is null)
```

```
                return this.Error(ErrorCodes.DataBaseError);


            return this.Ok(result);
        }
        catch
        {
            return this.Error(ErrorCodes.UnknowError);
        }
    }


    [HttpGet("Update/OS/{osTypeId}/Soft/{softwareId}")]
    [ProducesResponseType(typeof(string), 200)]
    public async Task<IActionResult> UpdateScript([FromRoute] int osTypeId,
[FromRoute] int softwareId)
    {
        try
        {
            string result = await _manager.GetScript(ScriptType.Update, softwareId,
osTypeId);
            if (result is null)
                return this.Error(ErrorCodes.DataBaseError);


            return this.Ok(result);
        }
        catch
        {
            return this.Error(ErrorCodes.UnknowError);
        }
    }
```

```
        [HttpGet("Delete/OS/{osTypeId}/Soft/{softwareId}")]
        [ProducesResponseType(typeof(string), 200)]
        public async Task<IActionResult> DeleteScript([FromRoute] int osTypeId,
[FromRoute] int softwareId)
        {
            try
            {
                string result = await _manager.GetScript(ScriptType.Delete, softwareId,
osTypeId);
                if (result is null)
                    return this.Error(ErrorCodes.DataBaseError);

                return this.Ok(result);
            }
            catch
            {
                return this.Error(ErrorCodes.UnknowError);
            }
        }

    }
}
```

Додаток В.2 – Лістинг файлу Manager.cs

```
using Dapper;
using HelpApi.DTOs;
using HelpApi.Enums;
```

```
using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace HelpApi.Managers
{
    public class Manager
    {
        private readonly string _connectionString;
        public Manager()
        {
            _connectionString = "Server=localhost;Port=5432;Database=Dyplom;User
Id=postgres;Password=gutenmorgen123;";
        }

        public async Task<int> CreateUserAsync(string email, string password)
        {
            try
            {
                using var connection = new NpgsqlConnection(_connectionString);
                await connection.OpenAsync();

                string query = $"insert into users (\"email\", \"password\") values (\'{email}\',
\'{password}\' ) returning \"id\";";

                int result = await connection.QuerySingleAsync<int>(query);
```

```
            return result;
        }
        catch
        {
            return 0;
        }
    }

    public async Task<bool> LoginUserAsync(string email, string password)
    {
        try
        {
            using var connection = new NpgsqlConnection(_connectionString);
            await connection.OpenAsync();

            string query = $"select id from users u where u.\"email\"  =  \'{email}\' and u.\"password\" = '{password}'";

            int result = await connection.QueryFirstOrDefaultAsync<int>(query);
            if(result<=0)
                return false;

            return true;
        }
        catch
        {
            return false;
        }
    }
```

```
public async Task<List<RequestTypeDTO>> RequestTypesAsync()
{
    try
    {
        using var connection = new NpgsqlConnection(_connectionString);
        await connection.OpenAsync();

        string query = "select * from  requesttype r ;";

        var result = await connection.QueryAsync<RequestTypeDTO>(query);

        return result.AsList();
    }
    catch
    {
        return null;
    }
}

public async Task<bool> CreateRequestAsync(int userId, int type, string
description)
{
    try
    {
        using var connection = new NpgsqlConnection(_connectionString);
        await connection.OpenAsync();
```

```
            string query = $"insert into userrequests (\"requesttypeid\", \"userid\", \"requestdescription\") values ({type}, {userId}, \'{description}\') returning \"id\";";

            var result = await connection.QueryFirstOrDefaultAsync<int>(query);
            if (result <= 0)
                return false;

            return true;
        }
        catch
        {
            return false;
        }
    }

    public async Task<List<SoftwareDTO>> GetListAsync(int typeId)
    {
        try
        {
            using var connection = new NpgsqlConnection(_connectionString);
            await connection.OpenAsync();

            string query = $"select id, typeid ,softname as name, description , website from software s where s.\"typeid\" = {typeId};";

            var result = await connection.QueryAsync<SoftwareDTO>(query);

            return result.AsList();
        }
```

```
        catch
        {
            return null;
        }
    }

    public async Task<List<TypeDTO>> GetTypeListAsync()
    {
        try
        {
            using var connection = new NpgsqlConnection(_connectionString);
            await connection.OpenAsync();

            string query = "select id as id, typename as name, description  from softwaretype s ;";

            var result = await connection.QueryAsync<TypeDTO>(query);

            return result.AsList();
        }
        catch
        {
            return null;
        }
    }

    public async Task<List<SoftwareDTO>> SearchSoftware(string search)
    {
        try
```

```
        {
            using var connection = new NpgsqlConnection(_connectionString);
            await connection.OpenAsync();

            string query = $"select id, typeid, softname as name, description , website
from software s where s.softname like '{search}';";

            var result = await connection.QueryAsync<SoftwareDTO>(query);

            return result.AsList();
        }
        catch
        {
            return null;
        }
    }

    public async Task<string> GetScript(ScriptType type, int softwareId, int osId)
    {
        try
        {
            using var connection = new NpgsqlConnection(_connectionString);
            await connection.OpenAsync();

            var builder = new StringBuilder();

            //if (osId != 1)
            //    builder.Append("#!/bin/sh \n");
```

```
            string searchField = type switch
            {
                ScriptType.Install => "installcom",
                ScriptType.Update => "updatecom",
                ScriptType.Delete => "deletecom",
                _=> throw new InvalidOperationException()
            };

            string queryCommand = $"select {searchField} from packagemanagers o
where o.\"osid\" = {osId};";

            string command = await
connection.QueryFirstOrDefaultAsync<string>(queryCommand);

            builder.Append(command);

            string queryPackage = $"select packagename from packages p where
p.\"osid\" = {osId} and p.\"softwareid\" = {softwareId};";

            string packageName = await
connection.QueryFirstOrDefaultAsync<string>(queryPackage);

            builder.Append(packageName);

            return builder.ToString();
        }
        catch
        {
            return null;
        }
```

```
        }
    }
}
```

Додаток В.3 – Лістинг файлу main.py

```
# This Python file uses the following encoding: utf-8
from PyQt6 import QtCore, QtGui, QtWidgets

from apiReceiver import getsoftTypes;
from logic import refresh_db_list, refresh_db_types

from ui_form import Ui_Form



if __name__ == "__main__":

    list = refresh_db_types()
    refresh_db_list(int(list[0]['id']))

    import sys
    app = QtWidgets.QApplication(sys.argv)
    Form = QtWidgets.QWidget()
    ui = Ui_Form()
    ui.setupUi(Form)
    Form.show()
    sys.exit(app.exec())
```

Додаток В.4 – Лістинг файлу ui_form.py

```
from typing import List
from apiReceiver import get_install_script


from jsonReader import *
import logic

from PyQt6 import QtCore, QtGui, QtWidgets

from PyQt5.QtWidgets import  QListWidgetItem

OS_TYPE: int = 1

class Ui_Form(object):

    def loadTypes(self):
        print("\loadTypes - ui_form.py\n")
        listTypes : List[SoftType] = logic.refresh_db_types()

        self.listWidget.clear()
        for type in listTypes:
            self.listWidget.addItem(str(type['name']))

    def set_soft_list(self, softList: List[Software]):
        self.softwareList.clear()
```

```
        for soft in softList:
            self.softwareList.addItem(str("{} | Name - {} | Description - {} | Developer website - {}"
            .format(soft['id'],soft['name'],soft['description'],soft['webSite'])))

    def loadList(self):
        print("\loadList - ui_form.py\n")

        currentItem = self.listWidget.currentItem()
        if(currentItem is None):
            print("Select type")
            return

        currentItem = currentItem.text()

        softList : List[Software] = []
        listTypes : List[SoftType] = logic.refresh_db_types()
        for type in listTypes:
            if type['name'] == currentItem:
                softList = logic.refresh_db_list(type['id'])



        self.set_soft_list(softList= softList)



    def btn_search(self):
        text = self.textEdit.toPlainText()
```

```
    list = logic.search_list(text)
    if(len(list) == 0):
        return

    self.set_soft_list( list)

def get_soft_id(self) -> int:
    currentItem = self.softwareList.currentItem()

    if(currentItem is None):
        print("Select software")
        return

    currentItem = currentItem.text()

    return int(currentItem[0])

def btn_install(self):
    print("\btn_install - ui_form.py\n")

    id : int = self.get_soft_id()

    logic.install_software(OS_TYPE, id)

def btn_delete(self):
    print("\btn_delete - ui_form.py\n")

    id : int = self.get_soft_id()
```

```
    logic.delete_software(OS_TYPE, id)

def btn_update(self):
    print("\btn_update - ui_form.py\n")

    id : int = self.get_soft_id()

    logic.update_software(OS_TYPE, id)



def setupUi(self, Widget):
    Widget.setObjectName("Widget")
    Widget.resize(800, 600)
    self.textEdit = QtWidgets.QTextEdit(Widget)
    self.textEdit.setGeometry(QtCore.QRect(10, 20, 261, 31))
    self.textEdit.setObjectName("textEdit")

    #SEARCH BUTTON

    self.searchButton = QtWidgets.QPushButton(Widget)
    self.searchButton.setGeometry(QtCore.QRect(280, 20, 111, 31))
    self.searchButton.setObjectName("searchButton")

    self.searchButton.clicked.connect(self.btn_search)

    #Refresh button

    self.refreshDbButton = QtWidgets.QPushButton(Widget)
    self.refreshDbButton.setGeometry(QtCore.QRect(660, 10, 111, 41))
```

```
self.refreshDbButton.setObjectName("refreshDbButton")

#UPDATE BUTTON

self.updateButton = QtWidgets.QPushButton(Widget)
self.updateButton.setGeometry(QtCore.QRect(400, 20, 75, 24))
self.updateButton.setObjectName("updateButton")

self.updateButton.clicked.connect(self.btn_update)

#INSTALL BUTTON

self.installButton = QtWidgets.QPushButton(Widget)
self.installButton.setGeometry(QtCore.QRect(490, 20, 75, 24))
self.installButton.setObjectName("installButton")

self.installButton.clicked.connect(self.btn_install)

#DELETE BUTTON

self.deleteButton = QtWidgets.QPushButton(Widget)
self.deleteButton.setGeometry(QtCore.QRect(570, 20, 75, 24))
self.deleteButton.setObjectName("deleteButton")

self.deleteButton.clicked.connect(self.btn_delete)

#SHOW BUTTON
self.showButton = QtWidgets.QPushButton(Widget)
self.showButton.setGeometry(QtCore.QRect(60, 90, 75, 24))
```

```
        self.showButton.setObjectName("showButton")

        self.showButton.clicked.connect(self.loadList)

        #LIST WIDGET
        self.listWidget = QtWidgets.QListWidget(Widget)
        self.listWidget.setGeometry(QtCore.QRect(20, 130, 151, 451))
        self.listWidget.setObjectName("listWidget")

        #SOFTWARE LIST

        self.softwareList = QtWidgets.QListWidget(Widget)
        self.softwareList.setGeometry(QtCore.QRect(190, 80, 601, 511))
        self.softwareList.setObjectName("softwareList")

        self.retranslateUi(Widget)
        QtCore.QMetaObject.connectSlotsByName(Widget)

        self.loadTypes()

    def retranslateUi(self, Widget):
        _translate = QtCore.QCoreApplication.translate
        Widget.setWindowTitle(_translate("Widget", "Widget"))
        self.searchButton.setText(_translate("Widget", "Search"))
        self.refreshDbButton.setText(_translate("Widget", "Refresh"))
        self.updateButton.setText(_translate("Widget", "Update"))
        self.installButton.setText(_translate("Widget", "Install"))
        self.deleteButton.setText(_translate("Widget", "Delete"))
        self.showButton.setText(_translate("Widget", "Show "))
```

```
    def debug(self):
        print("PRESS")
```

Додаток В.5 – Лістинг файлу api_receiver.py

```
# importing the requests library
import json
import string
from typing import List
import requests

from jsonReader import Software



def getsoftTypes() -> string:
    # api-endpoint
    URL = "http://localhost:5000/Software/TypeList"

    # defining a params dict for the parameters to be sent to the API
#    PARAMS = {'address':location}

    # sending get request and saving the response as response object
    r = requests.get(url=URL)

    # extracting data in json format
    data = r.json()
```

```
    print(data)

    return data


def getSoftware(id: int) -> string:
    # api-endpoint
    URL = "http://localhost:5000/Software/List/{}".format(id)

    # defining a params dict for the parameters to be sent to the API
    PARAMS = {'typeId ': id}

    r = requests.get(url = URL)

    # extracting data in json format
    data = r.json()

    print(data)

    return data;

def search_software(search: str) -> string:
    # api-endpoint
    URL = "http://localhost:5000/Software/List/Search/{}".format(search)

    # defining a params dict for the parameters to be sent to the API
    # PARAMS = {'typeId ': id}

    r = requests.get(url = URL)
```

```
    # extracting data in json format
    data = r.json()

    print(data)

    return data

def get_install_script(softwareId: int, osId: int) -> str:
    URL = "http://localhost:5000/Software/Install/OS/{}/Soft/{}".format(osId,
softwareId)

    r = requests.get(url = URL)

    # extracting data
    data = r.text

    print(data)

    return data

def get_update_script(softwareId: int, osId: int)-> str:
    URL = "http://localhost:5000/Software/Update/OS/{}/Soft/{}".format(osId,
softwareId)

    r = requests.get(url = URL)

    # extracting data
    data = r.text
```

```
    print(data)

    return data

def get_delete_script(softwareId: int, osId: int)-> str:
    URL = "http://localhost:5000/Software/Delete/OS/{}/Soft/{}".format(osId, softwareId)

    r = requests.get(url = URL)

    # extracting data
    data = r.text

    print(data)

    return data
```

Додаток В.6 – Лістинг файлу migr.sql

```
create table if not exists RequestType(
id serial not null unique   primary key,
name text not null,
description text not null
);

create table if not exists users(
id serial not null unique primary key,
email text not null unique,
password text not null,
phoneNumber text unique
);

create table if not exists UserRequests(
id serial not null unique primary key,
requesttypeId int4 not null,
userId int4 not null,
requestDescription text not null
```

```
);

create table if not exists softwareType(
id serial not null unique primary key,
typeName text not null unique,
description text
);

create table if not exists software(
id serial not null unique primary key,
typeId int4 not null,
softName text not null,
description text,
webSite text
);

alter table userrequests add CONSTRAINT userrequest_userId
foreign key (userid) REFERENCES  users (id) match full;

alter table userrequests add CONSTRAINT userrequest_requestId
foreign key (requesttypeid) REFERENCES  requesttype (id) match full;

alter table software  add CONSTRAINT software_softwareTypeId
foreign key (typeid) REFERENCES  softwareType (id) match full;


create table if not exists operatingsystems(
id serial not null unique primary key,
osName text not null
);

create table if not exists packages(
id serial not null unique primary key,
packageName text not null,
softwareId int4 not null,
osId int4 not null
);

create table if not exists packageManagers(
id serial not null unique primary key,
osid int4 not null unique,
pmName text not null,
installCom text,
deleteCom text,
updateCom text
);

alter table packageManagers add constraint os_packageManager_id
foreign key (osid) references operatingsystems (id) match full;

alter table packages add constraint software_package_idw
foreign key (softwareId) references software (id) match full;

alter table packages add constraint package_os_id
foreign key (osId) references operatingsystems (id) match full;
```

ДОДАТОК Г.

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА СИСТЕМИ СТВОРЕННЯ ІНСТАЛЯЦІЙНИХ ПАКЕТІВ

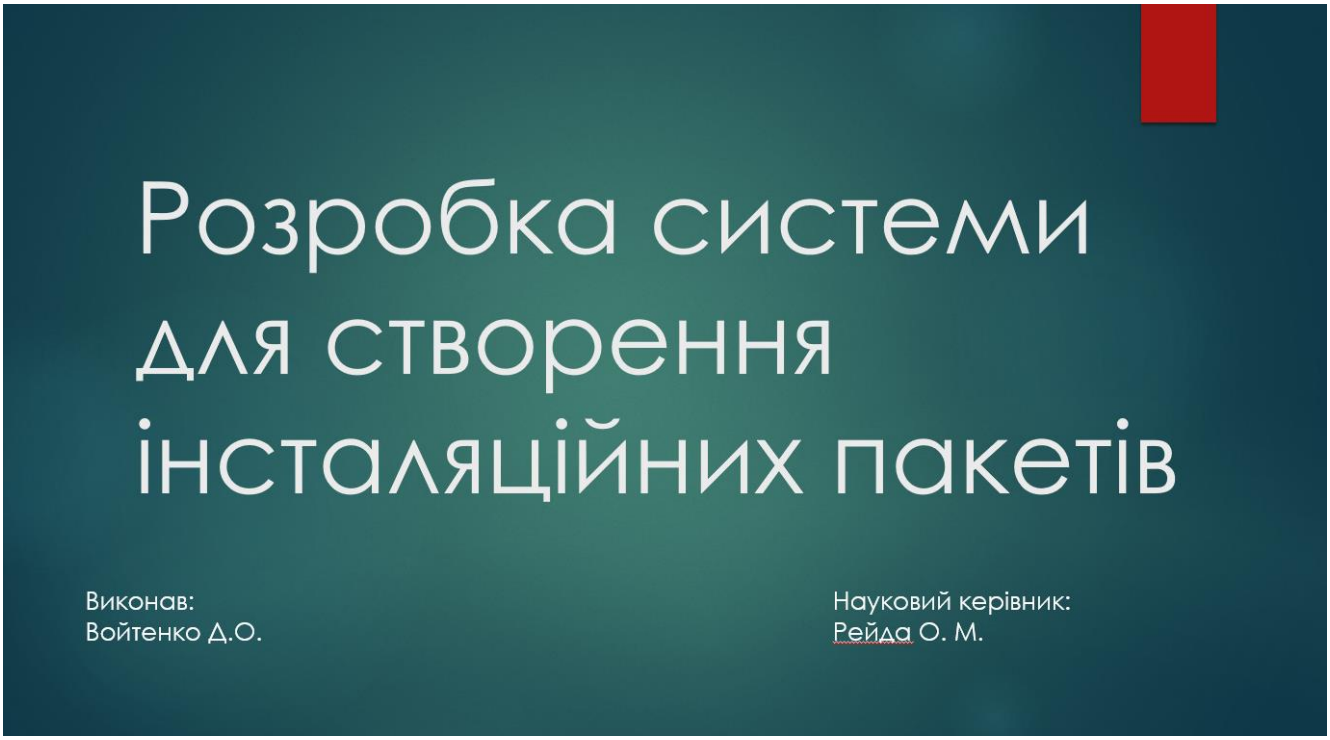

Рисунок Г.1 – Заголовок роботи

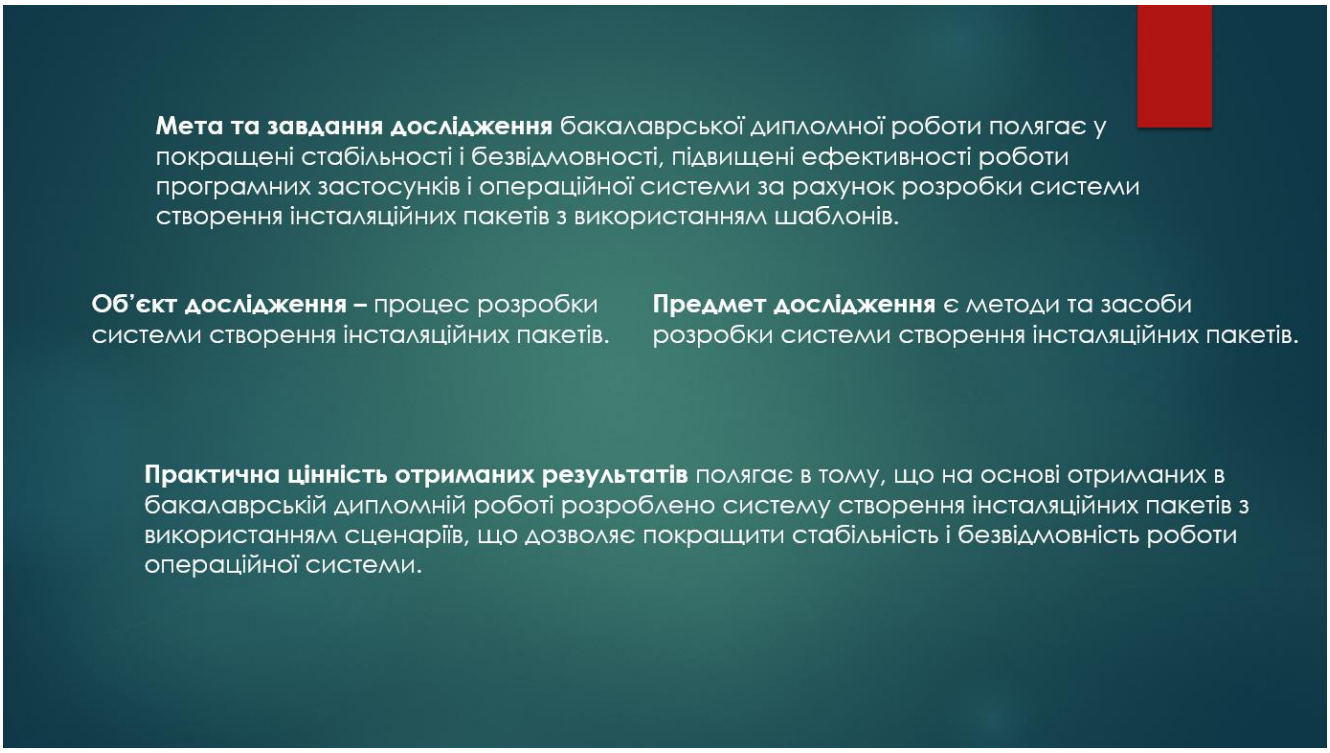

Рисунок Г.2 - Мета, об'єкт і предмет дослідження

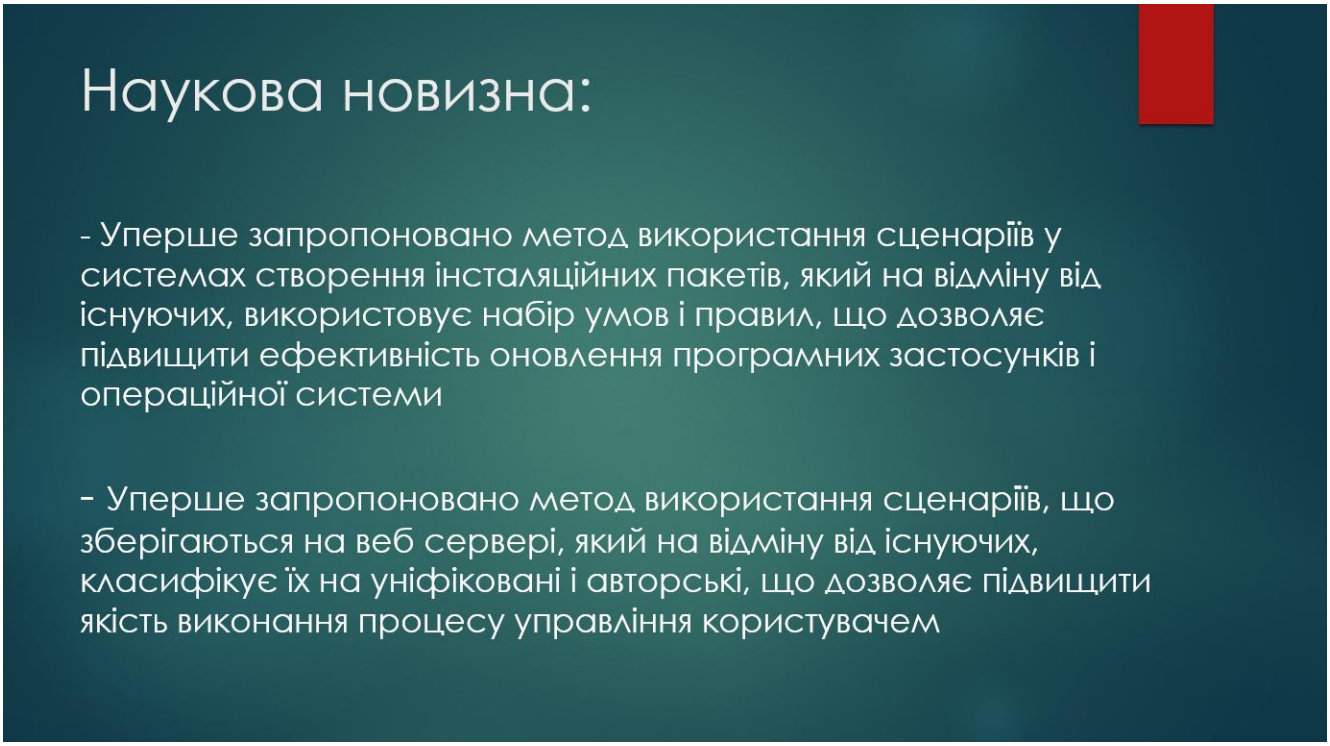

Рисунок Г.3 – Наукова новизна

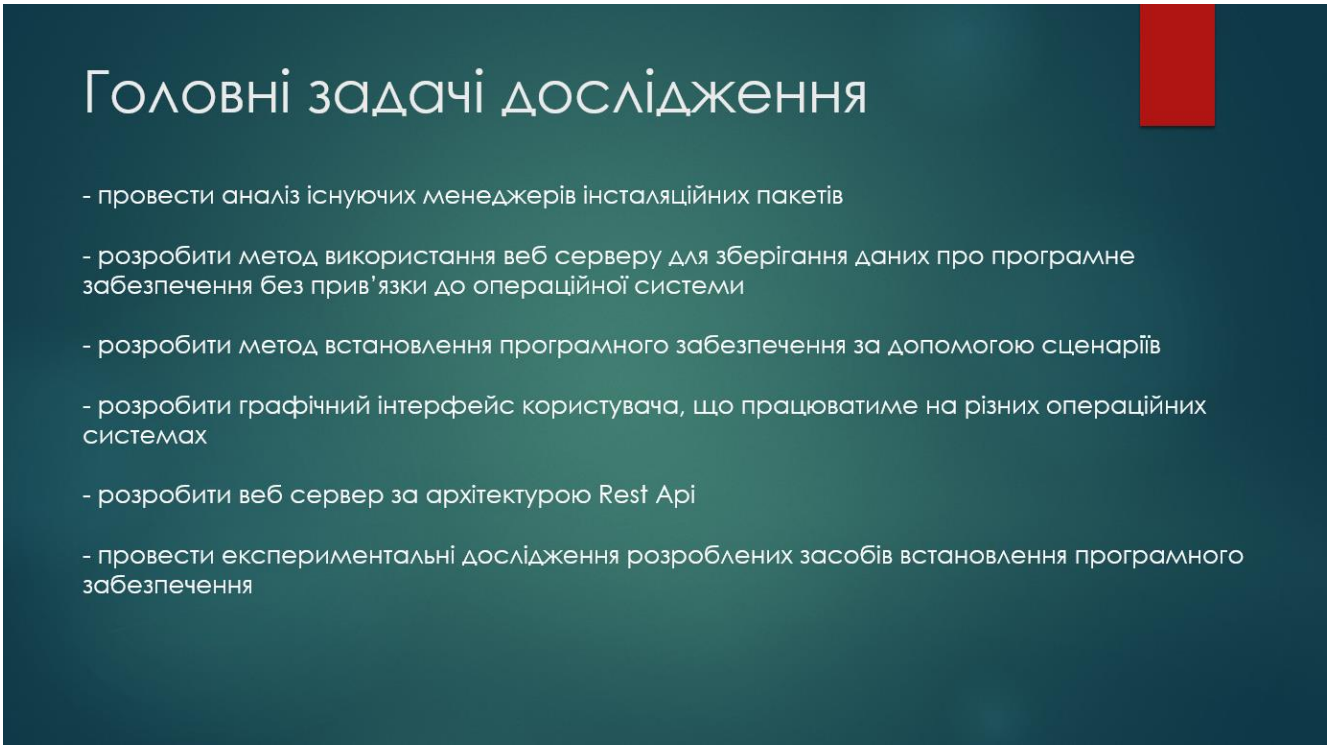

Рисунок Г.4 - Головні задачі дослідження

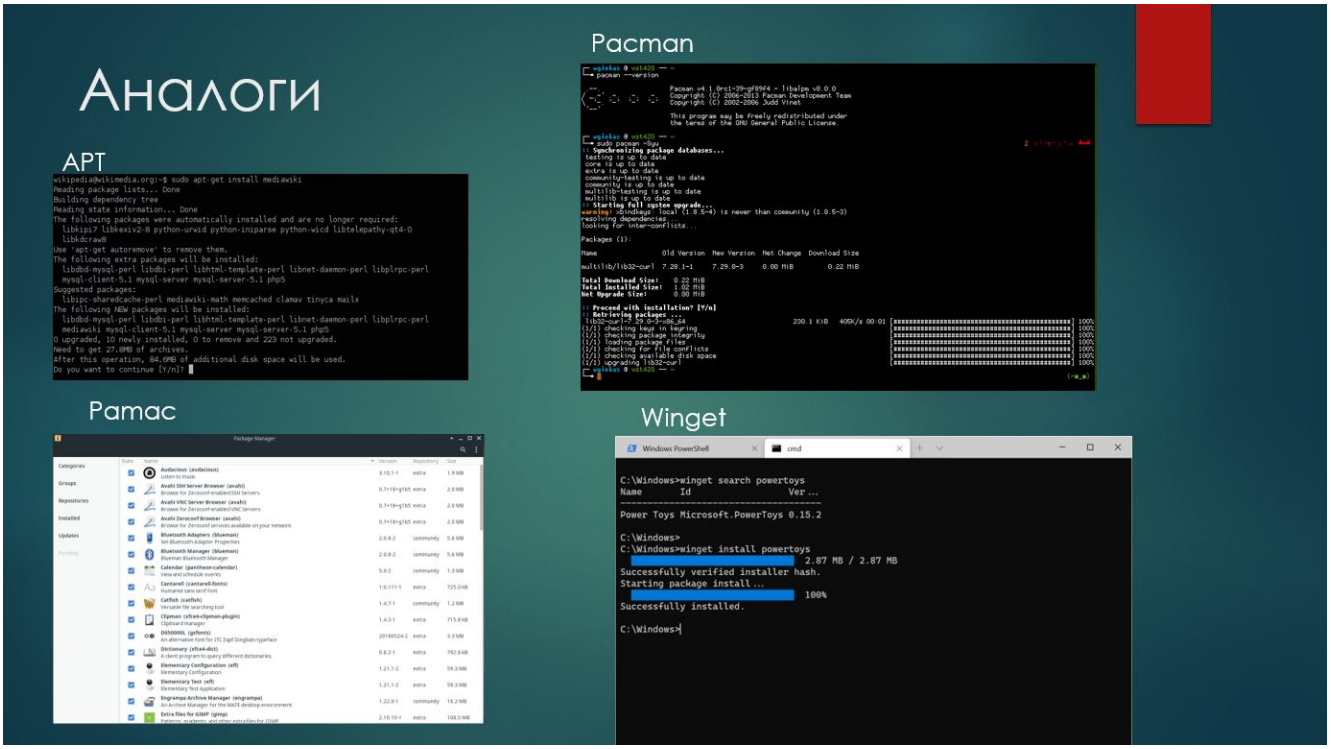

Рисунок Г.5 – Аналоги

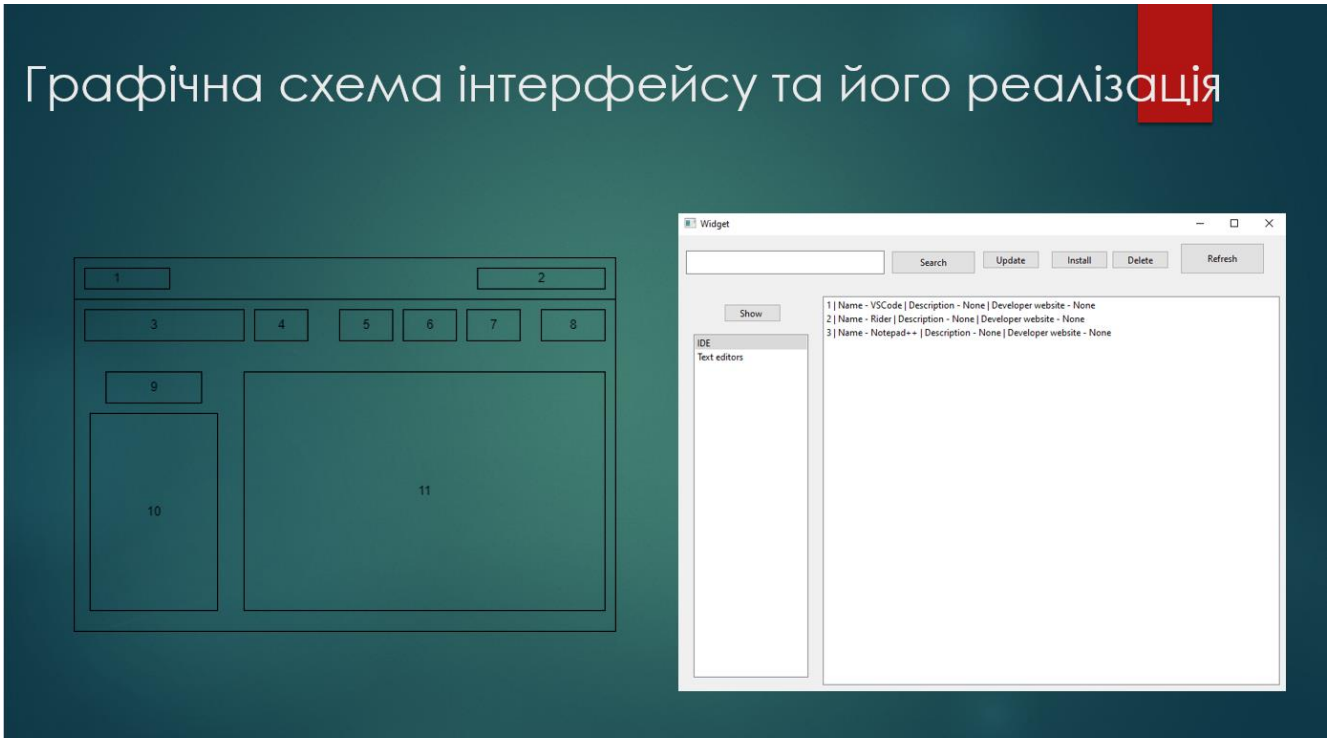

Рисунок Г.6 – Графічна схема інтерфейсу та його реалізація

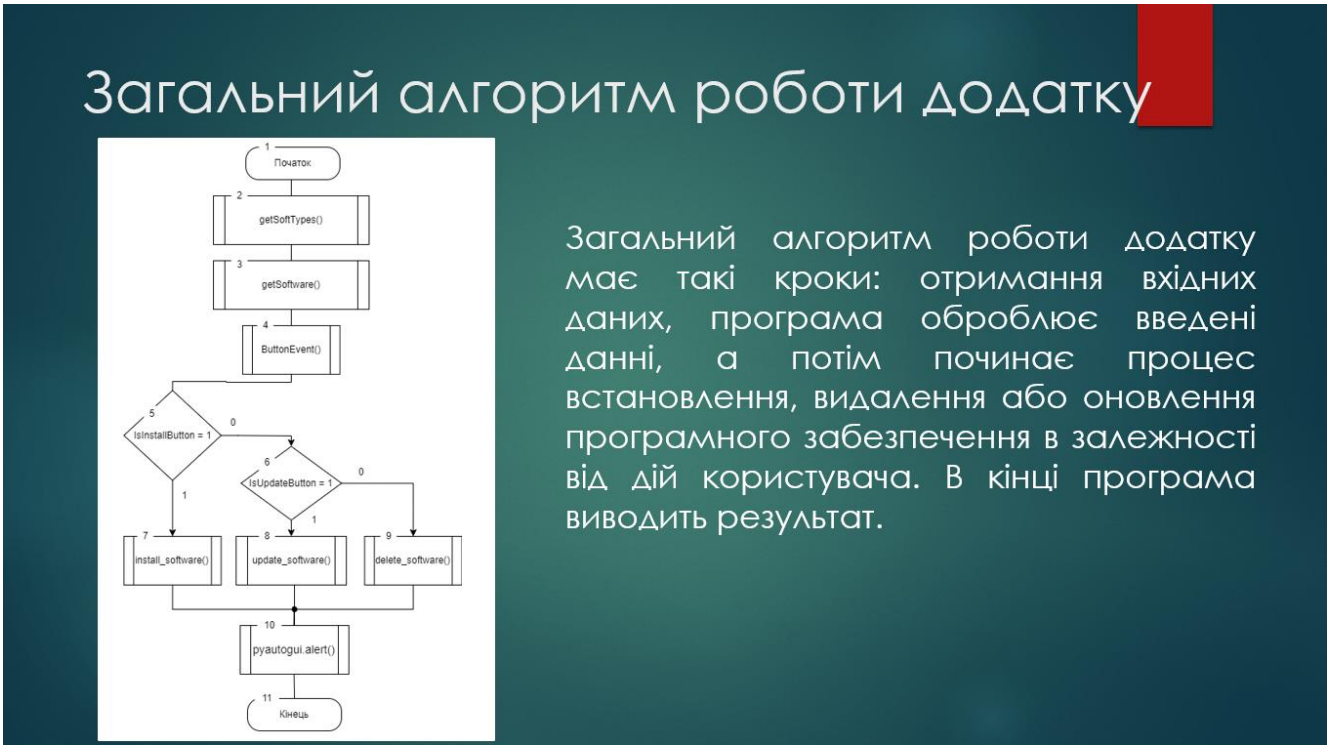

Рисунок Г.7 – Загальний алгоритм роботи додатку

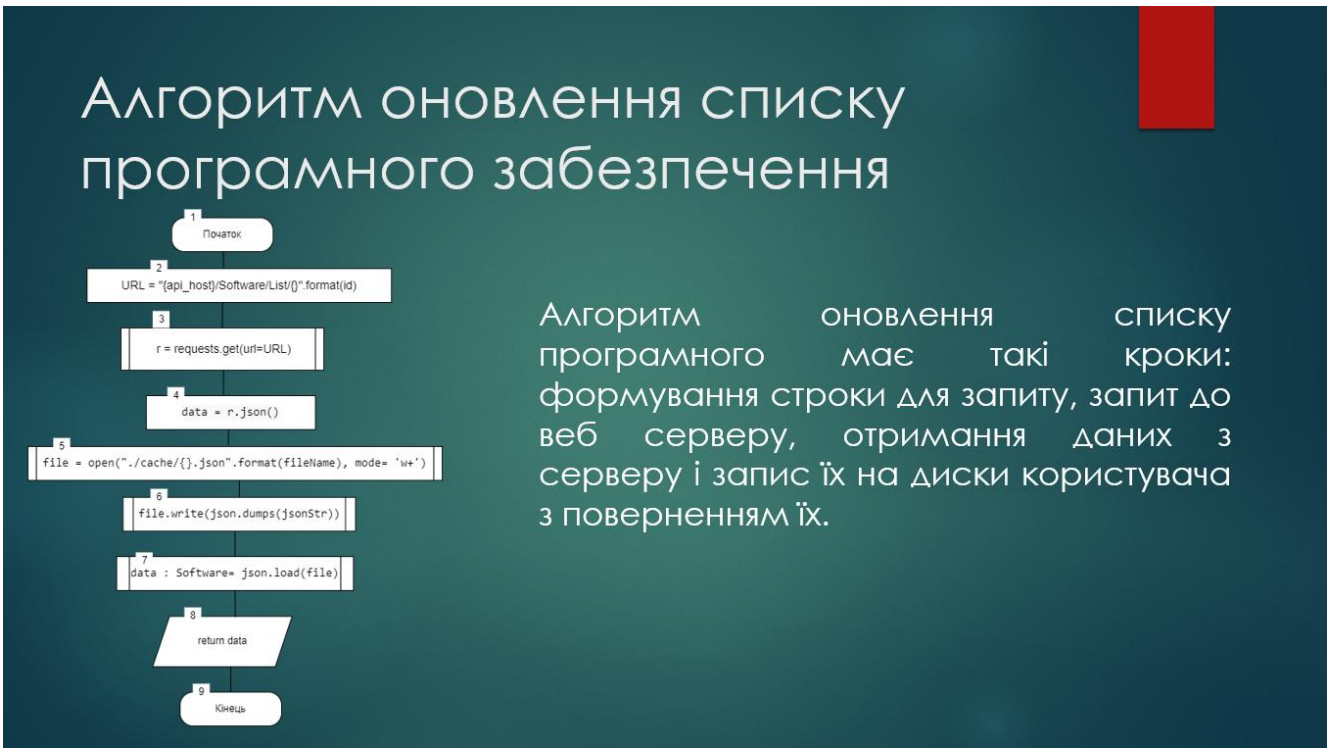

Рисунок Г.8 – Алгоритм оновлення списку програмного забезпечення

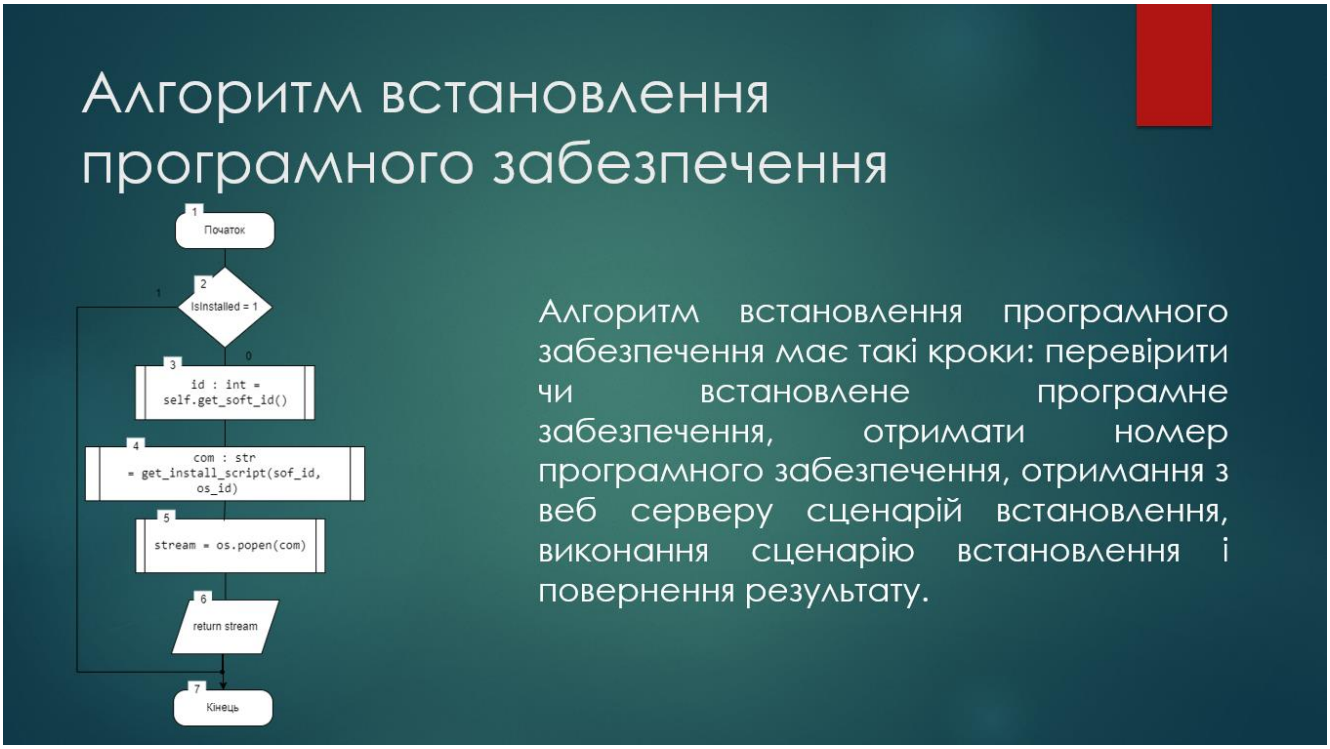

Рисунок Г.9 – Алгоритм встановлення програмного забезпечення

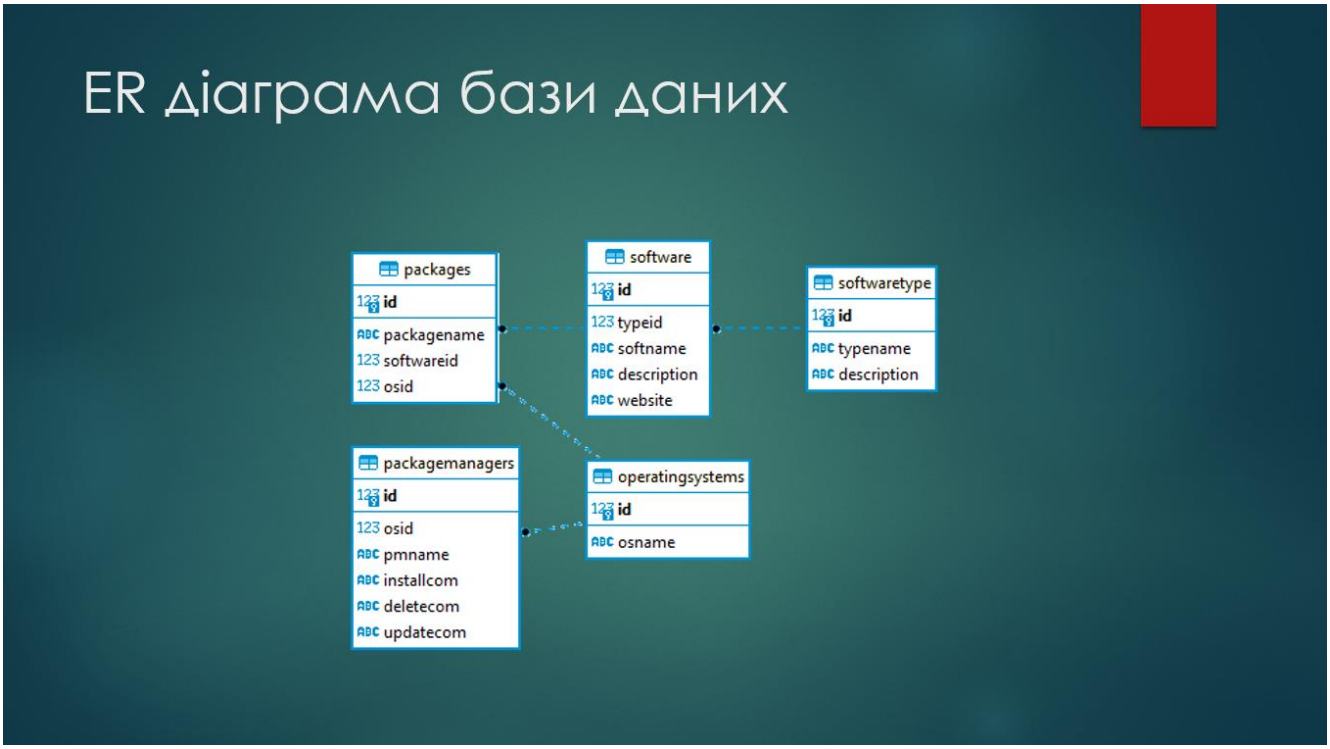

Рисунок Г.10 – ER діаграма бази даних

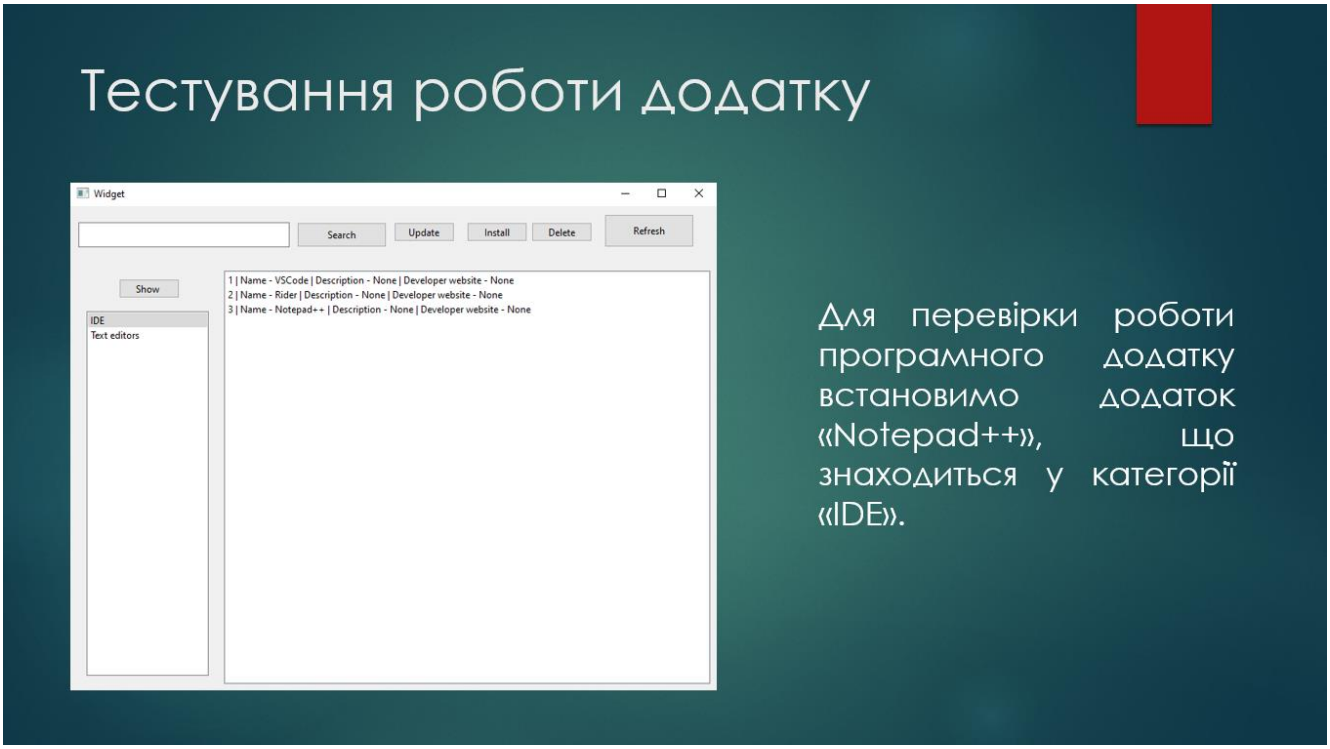

Рисунок Г.11 – Вибір програмного забезпечення

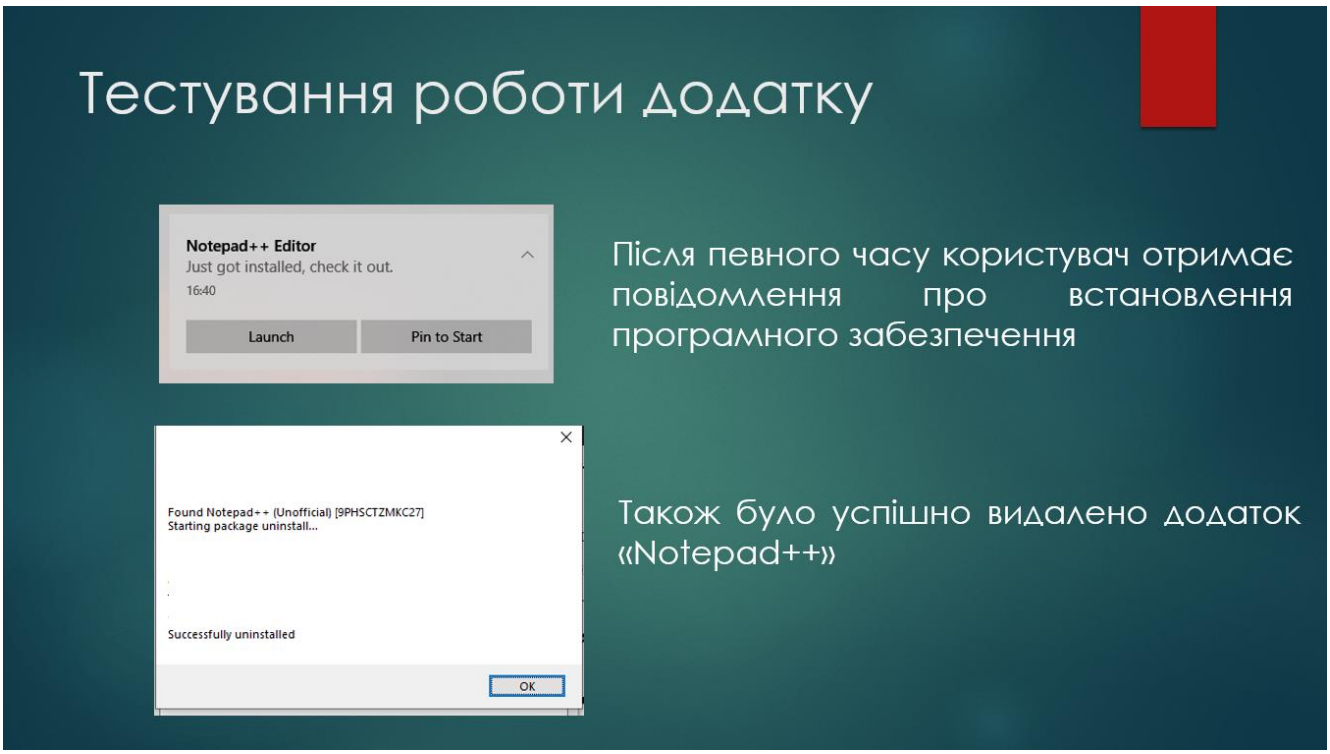

Рисунок Г.12 – Встановлення і видалення додатку «Notapad++»

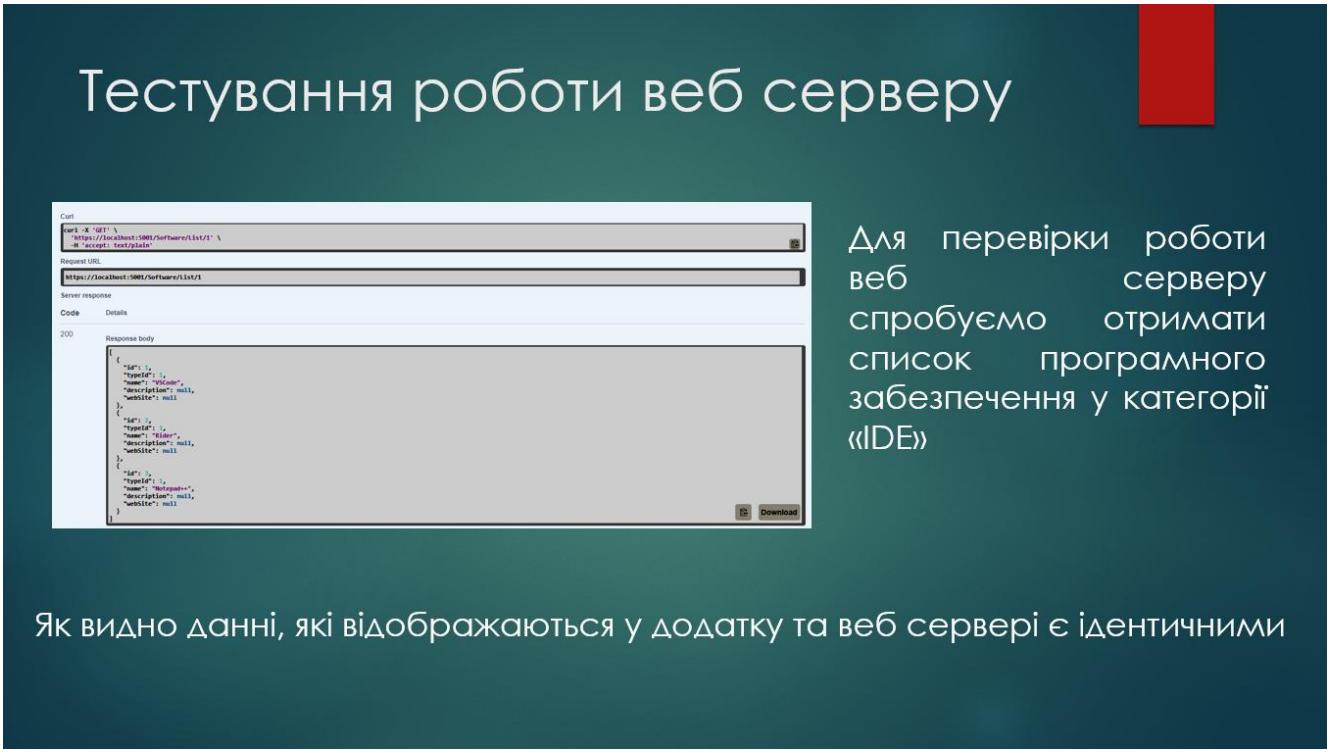

Рисунок Г.13 – Данні з веб серверу

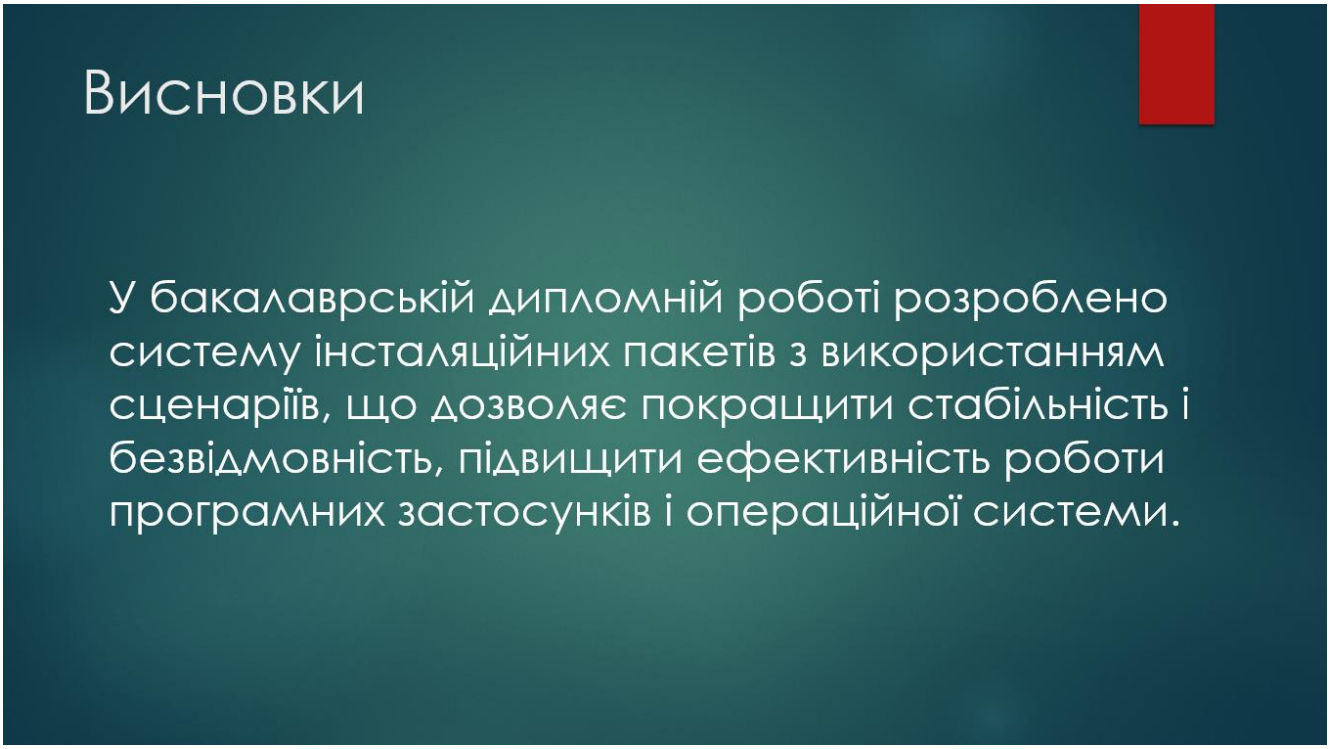

Рисунок Г.14 – Висновки

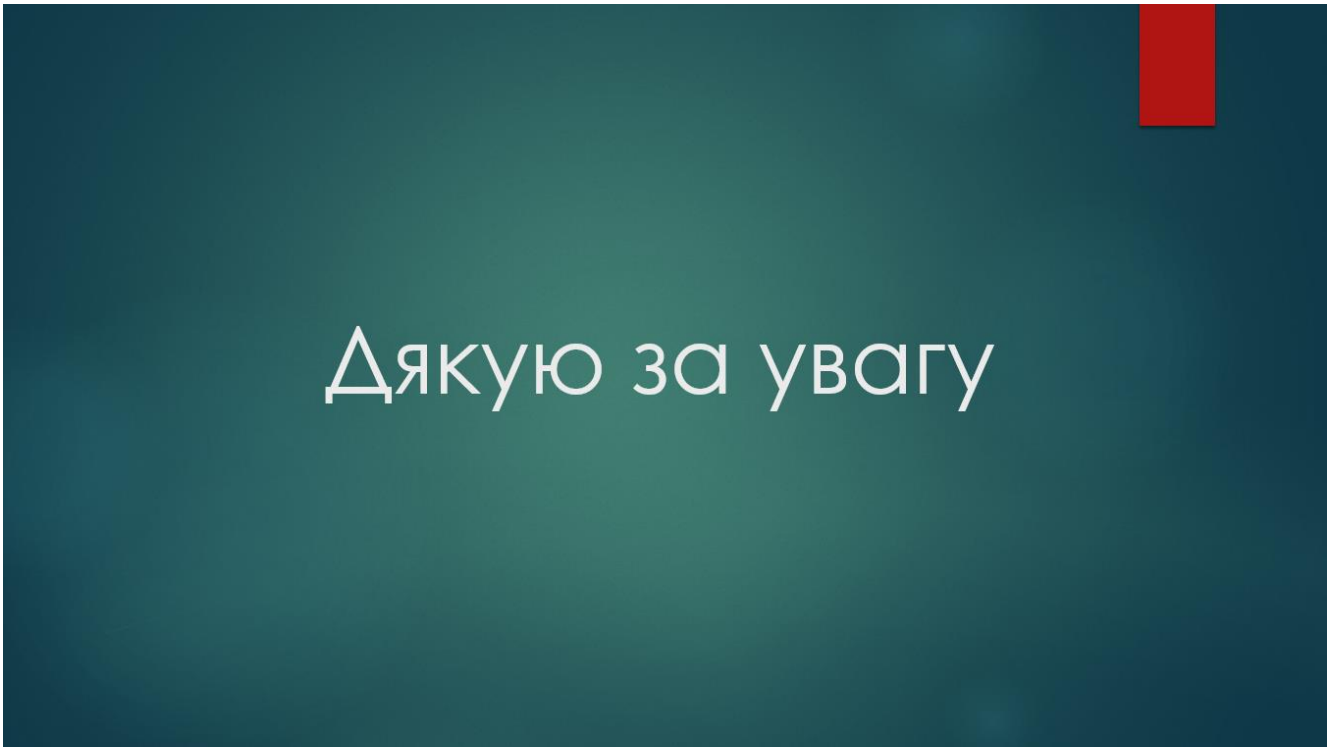

Рисунок Г.15 – Кінцевий слайд