МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет економіки, підприємництва та інформаційних технологій

Кафедра інформаційних технологій

Допущено до захисту

"_____" _____ 2026 р.

Зав. кафедри, к.т.н., доцент

_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОГО ГЕНЕРУВАННЯ ІНСТАЛЯЦІЙНИХ ПАКЕТІВ

Виконав – студент 4 курсу, групи 41-кн
освітньої програми "Комп'ютерні науки"
спеціальності 122 Комп'ютерні науки
Покуц Максим Дмитрович

Науковий керівник – к.т.н., доцент
Ліщук Роман Ігорович

Умань – 2026

Зміст

ВСТУП 3

РОЗДІЛ 1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 6

1.1. Аналіз стану пакетних менеджерів 6

1.2. Порівняльний аналіз аналогів 6

1.3. Аналіз методів розв’язання поставленої задачі 10

1.4. Постановка задачі 12

РОЗДІЛ 2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ГЕНЕРУВАННЯ ІНСТАЛЯЦІЙНИХ ПАКЕТІВ 13

2.1. Вибір мови програмування 13

2.2. Аналіз даних 14

2.3. Розробка графічної схеми інтерфейсу програмного додатку 16

2.4. Розробка веб серверу для оновлення даних додатку 18

2.5. Розробка алгоритмів роботи додатку 24

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОГО ГЕНЕРУВАННЯ ІНСТАЛЯЦІЙНИХ ПАКЕТІВ 32

3.1. Реалізація інтерфейсу користувача 32

3.2. Реалізація бази даних для зберігання відомостей про програмне забезпечення 35

3.3. Розробка серверної частини для забезпечення доступу до даних .. 38

3.4. Тестування розробленого програмного забезпечення 41

ВИСНОВКИ 57

СПИСОК ЛІТЕРАТУРИ 59

ДОДАТКИ 62

ВСТУП

Стрімкий розвиток інформаційних технологій зумовлює підвищену потребу у надійному програмному забезпеченні та ефективних засобах його контролю. Забезпечення корпоративного та державного секторів безпечними робочими станціями, захищеними від шкідливого коду, стало фундаментальним чинником еволюції пакетних менеджерів, що функціонують на основі інсталяції верифікованого програмного забезпечення з довірених джерел.

Сучасні пакетні менеджери виконують низку критичних функцій: – підключення розширених бібліотек для забезпечення програмних залежностей шляхом їх завантаження та динамічного зв'язування; – конфігурування, що передбачає встановлення та збереження параметрів функціонування програмних застосунків; – взаємодія з репозиторієм для завантаження компонентів зі спеціалізованих віддалених сховищ; – оновлення шляхом моніторингу та розгортання актуальних версій програмних продуктів; – деінсталяція програмного забезпечення та асоційованих залежностей з метою оптимізації дискового простору.

Підтримка стабільності, надійності та високої ефективності експлуатації операційних систем вимагає своєчасної актуалізації програмних засобів. Ігнорування цього процесу неминуче призводить до зниження продуктивності операторів, а також до деградації показників стабільності та безвідмовності функціонування програмних застосунків і середовища загалом.

Існуючі методи та засоби системного менеджменту характеризуються значним споживанням обчислювальних ресурсів та операційного часу, що генерує затримки в процесах розгортання та оновлення новітнього програмного забезпечення. Зазначені недоліки негативно корелюють зі стабільністю та безвідмовністю, знижуючи загальну ефективність роботи програмних застосунків і операційних систем.

З огляду на вищезазначене, актуальним науково-практичним завданням є розробка системи автоматизованого генерування інсталяційних пакетів з використанням сценарних підходів, що дозволить підвищити стабільність, безвідмовність та загальну ефективність функціонування програмного забезпечення та операційної системи.

Метою кваліфікаційної роботи є покращення стабільності, безвідмовності та підвищення ефективності функціонування програмних застосунків і операційної системи шляхом розробки системи автоматизованого генерування інсталяційних пакетів на базі шаблонів.

Для досягнення поставленої мети визначено такі головні завдання дослідження: – здійснити критичний аналіз існуючих менеджерів інсталяційних пакетів; – запропонувати метод використання вебсервера для зберігання метаданих програмного забезпечення без платформної залежності від операційної системи; – розробити метод інсталяції програмного забезпечення за допомогою сценаріїв; – спроєктувати кросплатформний графічний інтерфейс користувача; – розробити серверну частину за архітектурою REST API; – провести експериментальне дослідження розроблених засобів розгортання програмного забезпечення.

Об’єкт дослідження – процес розробки системи автоматизованого генерування інсталяційних пакетів.

Предмет дослідження – методи та засоби розробки системи автоматизованого генерування інсталяційних пакетів.

Методи дослідження. У процесі виконання роботи застосовано комплекс загальнонаукових та спеціальних методів: аналізу та синтезу для формалізації ключових моделей пакетного менеджера; прикладної теорії інформації та теорії алгоритмів для проєктування алгоритмічного і програмного забезпечення; методи тестування програмних продуктів для верифікації працездатності розробленої системи; методи управління даними для організації їх зберігання у реляційній базі.

Практична цінність отриманих результатів. Набуті в ході дослідження результати склали основу для практичної реалізації системи автоматизованого генерування інсталяційних пакетів з використанням сценаріїв, впровадження якої дозволяє суттєво покращити показники стабільності та безвідмовності функціонування операційної системи.

РОЗДІЛ 1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1. Аналіз стану пакетних менеджерів

Еволюція інформаційних технологій актуалізує проблематику захисту інформації, гарантування безпеки програмного забезпечення та верифікації джерел його постачання. З метою задоволення зазначених потреб розробляються пакетні менеджери, які забезпечують коректне розгортання програмного забезпечення на цільових пристроях користувачів із мінімізацією потенційних загроз безпеці [1].

Для оптимізації процесів дистрибуції розробники формують власні програмні репозиторії, ресурси яких кінцевий користувач здатний інтегрувати до локальної бази даних застосовуваного пакетного менеджера.

До вагомих переваг експлуатації існуючих пакетних менеджерів належать: – застосування стандартизованих форматів пакетів; – наявність розгалуженої інфраструктури діючих репозиторіїв.

Незважаючи на вказані переваги, виділено низку неврегульованих проблем, що потребують вирішення: – відсутність графічного інтерфейсу користувача; – жорстка залежність від архітектури конкретної операційної системи; – відсутність систематизації програмного забезпечення за категоріями.

На основі аналізу вищезазначених недоліків встановлено, що використання доступних пакетних менеджерів супроводжується експлуатаційними ускладненнями та характеризується платформозалежністю, що унеможливлює забезпечення уніфікованого підходу до взаємодії з користувачем під час розгортання та автоматизованого генерування інсталяційних пакетів.

1.2. Порівняльний аналіз аналогів

Сучасний етап розвитку інформаційних систем характеризується наявністю широкого спектра інструментів для управління програмним забезпеченням. До найбільш поширених рішень у цьому сегменті належать: – Apt; – Pacman; – Pamac; – Winget.

Інструмент Apt (рис. 1.1) являє собою спеціалізований пакетний менеджер, що базово застосовується у дистрибутивах сімейства Debian та Ubuntu і здійснює обробку файлів формату deb [2]. Концептуальною особливістю екосистеми даного інструменту є політика комплектування офіційних репозиторіїв, яка обмежується виключно вільним програмним забезпеченням з відкритим вихідним кодом. При цьому, з огляду на пріоритетність гарантування високого рівня стабільності операційної системи, механізми дистрибуції орієнтовані на постачання перевірених, надійних, проте здебільшого не найновіших релізів програмних продуктів.

```
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

Інструмент Pacman, представлений на рисунку 1.2, являє собою пакетний менеджер, базово інтегрований в екосистему дистрибутива Arch Linux, що спеціалізується на обробці бінарних пакетних файлів. Специфіка його репозиторіїв полягає в акумулюванні значного масиву програмного забезпечення, супровід якого переважно здійснюється силами спільноти незалежних розробників. Незважаючи на високий рівень затребуваності даного рішення, низка програмних продуктів не забезпечується офіційною підтримкою з боку їх авторів. Проте концептуальною перевагою такої моделі є гарантована дистрибуція найактуальніших релізів програмного забезпечення.

Програмна реалізація пакетного менеджера Pacman виконана мовою програмування C із застосуванням функціональних можливостей утиліти bsdtar [3].

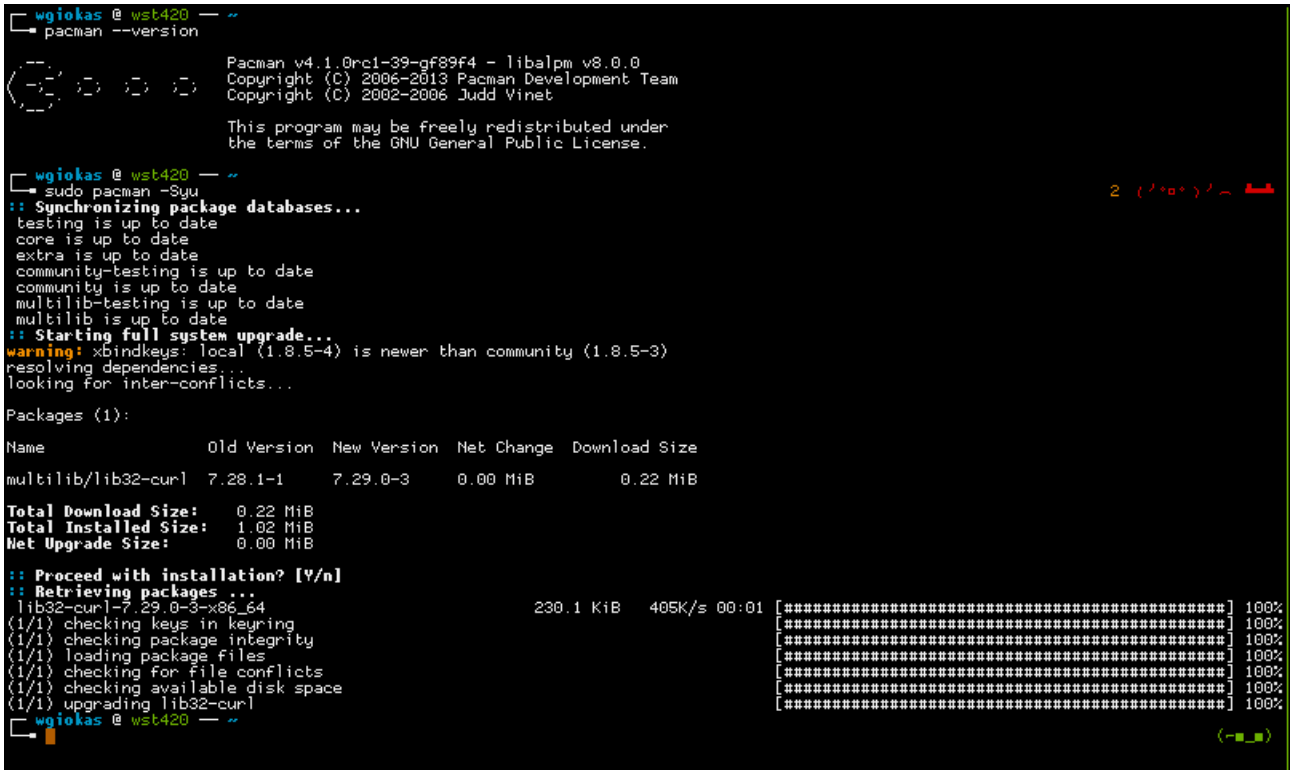


Рисунок 1.2 – Приклад команди пакетного менеджера Pacman

Інструмент Pamac, наведений на рисунку 1.3, являє собою пакетний менеджер, що функціонує в операційному середовищі дистрибутива Manjaro Linux і спеціалізується на обробці бінарних пакетних файлів [4]. Зазначене рішення оснащене графічним інтерфейсом користувача, який спроєктовано на засадах візуального мінімалізму. Архітектурною та функціональною особливістю цього засобу є відсутність механізмів підтримки інтеграції зі сторонніми або альтернативними репозиторіями.

Комплекс можливостей пакетного менеджера Pamac охоплює виконання операцій інсталяції, актуалізації версій та деінсталяції пакунків. Застосування стандартизованих пакетів як альтернативи процедурам безпосередньої компіляції та ручного встановлення програмного забезпечення забезпечує суттєву раціоналізацію процесів адміністрування операційної системи.

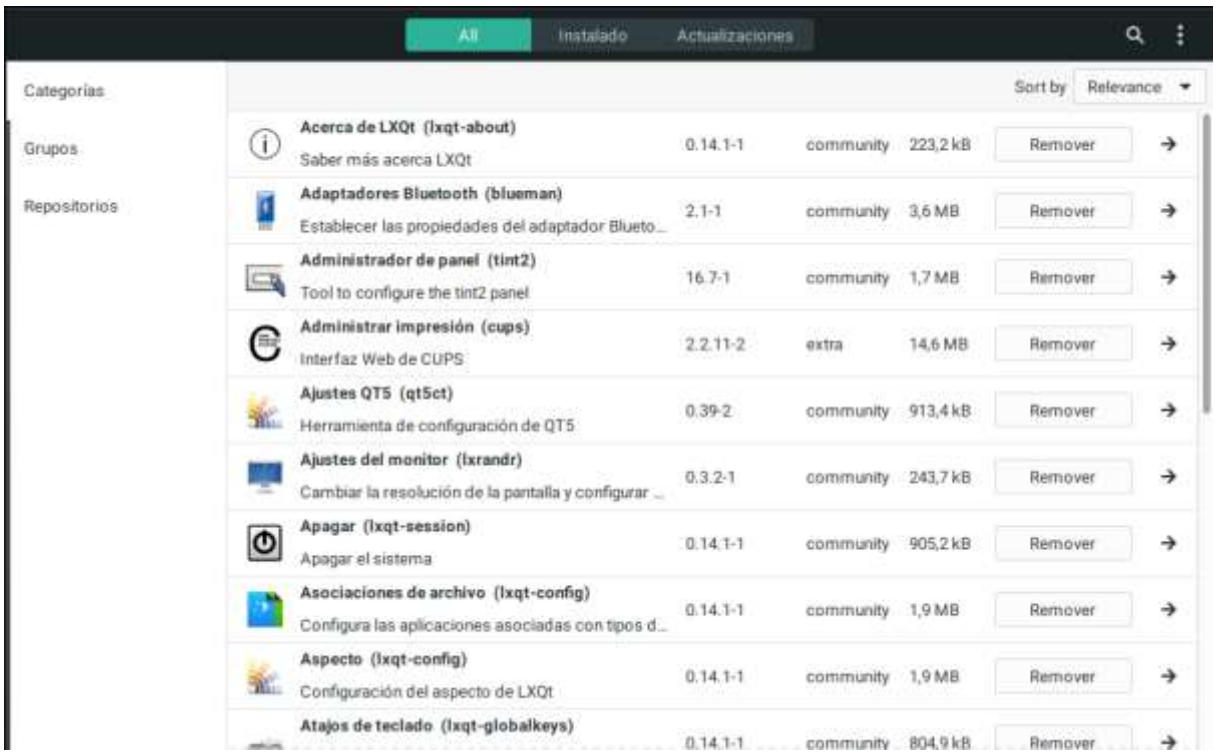


Рисунок 1.3 – Приклад інтерфейсу пакетного менеджера Pamac

Програмне рішення Winget, наведене на рисунку 1.4, є спеціалізованим пакетним менеджером, спроєктованим корпорацією Microsoft для інтеграції з операційними системами Windows 10 та 11 [5]. Технологічна специфіка даного засобу полягає в обов'язковій вимозі щодо сумісності застосунків із форматом UWP. Зазначена архітектурна обмеженість виступає критичним фактором, який істотно звужує загальний обсяг та варіативність доступного для розгортання програмного забезпечення.

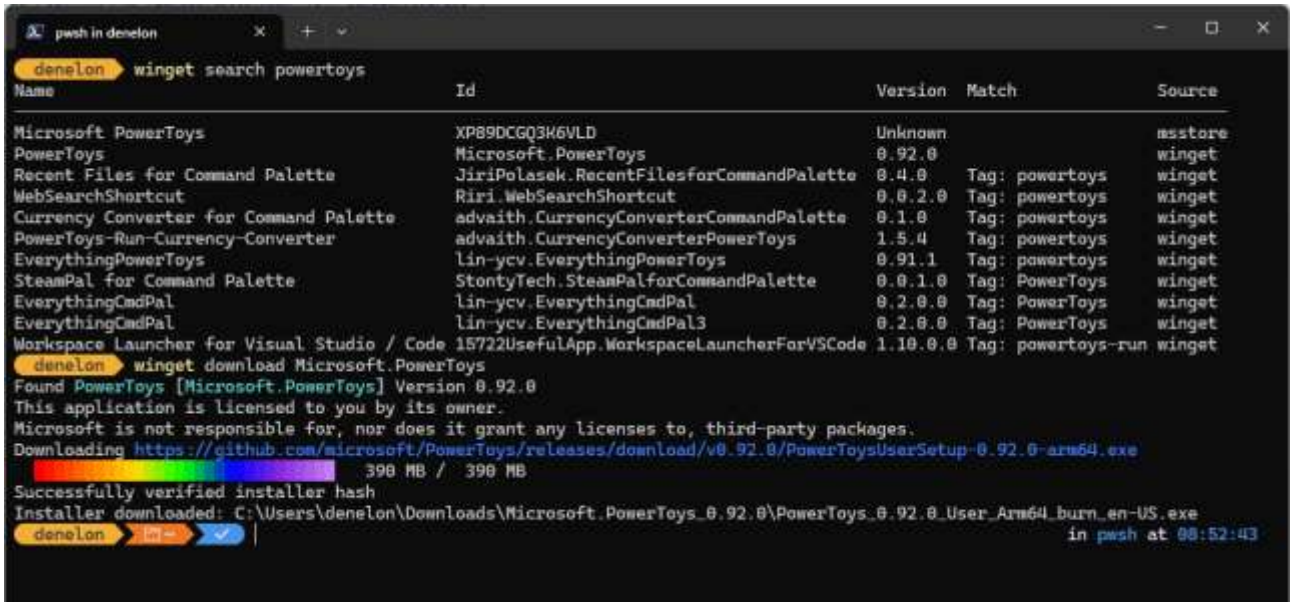


Рисунок 1.4 – Приклад команди пакетного менеджера Winget

Узагальнені результати порівняльного аналізу існуючих аналогів представлено в таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристики пакетних менеджерів

Критерії порівняння	Apt	Pacman	Pamac	Winget
Цільова платформа	Debian, Ubuntu	Arch Linux	Manjaro Linux	Windows
Тип інтерфейсу	Командний рядок	Командний рядок	Графічний та командний рядок	Командний рядок
Стратегія оновлень	Фокус на стабільність	Безперервні оновлення	Безперервні оновлення	За запитом
Підтримка зовнішніх джерел	Потребує ручного налаштування	Широка підтримка спільноти	Інтегрована	Базова
Робота з графічною оболонкою	Відсутня базова	Відсутня базова	Наявна	Відсутня базова

Здійснений аналіз підтверджує, що кожне з розглянутих рішень має вузьку спеціалізацію, переважно позбавлене вбудованого графічного інтерфейсу та має жорстку прив'язку до конкретної операційної системи. Зазначені фактори обґрунтовують практичну необхідність проєктування універсальної системи автоматизованого генерування інсталяційних пакетів, яка б нівелювала виявлені недоліки та забезпечила кросплатформний підхід.

1.3. Аналіз методів розв’язання поставленої задачі

Вирішення сформульованої науково-практичної проблеми передбачає розгляд та оцінку альтернативних концептуальних підходів. Перший підхід базується на проєктуванні нативного системного компонента, який характеризується жорсткою архітектурною прив'язкою до специфіки конкретної операційної системи та передбачає розробку унікальної візуальної оболонки для кожної окремої платформи. Зазначена стратегія забезпечує

можливість глибокої системної інтеграції та реалізації низькорівневих параметрів конфігурації. Однак критичним недоліком такого методу є надмірна трудомісткість процесу розробки, що зумовлена необхідністю паралельного створення кількох відокремлених гілок коду. Крім того, нативний підхід вимагає імплементації багатоетапних та складних процедур верифікації і тестування, що суттєво збільшує загальні часові витрати на життєвий цикл розробки програмного забезпечення.

Альтернативним та значно більш раціональним методом є застосування кросплатформних технологій для реалізації універсального графічного інтерфейсу користувача. Цей підхід дозволяє експлуатувати єдину кодову базу візуальної оболонки у різнорідних операційних середовищах. При цьому процеси розгортання та автоматизованого генерування інсталяційних пакетів пропонується реалізувати на базі спеціалізованих сценаріїв. Відповідні алгоритмічні інструкції динамічно адаптуються під архітектурні особливості цільової платформи користувача, забезпечуючи високий рівень гнучкості та абстрагування від жорстких системних залежностей. Зазначена архітектурна парадигма гарантує прискорення темпів розробки, спрощує подальший технічний супровід програмного продукту та оптимізує ресурсоємність процесів дистрибуції.

На основі комплексного порівняльного аналізу сильних та слабких сторін розглянутих архітектурних рішень, для подальшого проєктування було обрано комбінований метод. Визначений вектор дослідження полягає у синтезі незалежного від платформи графічного інтерфейсу та сценарного підходу до управління інсталяційними процесами. Такий вибір є науково та технічно обґрунтованим, оскільки він забезпечує оптимальний баланс між високою продуктивністю функціонування розробленого програмного забезпечення, універсальністю його застосування на широкому спектрі користувацьких пристроїв та загальною мінімізацією витрат на етапах системного проєктування і тестування.

1.4. Постановка задачі

На основі комплексного аналізу переваг та експлуатаційних недоліків існуючих пакетних менеджерів було сформульовано такі завдання кваліфікаційної роботи:

– здійснити критичний огляд наявних менеджерів інсталяційних пакетів;

– розробити метод використання веб-сервера для зберігання даних про програмне забезпечення із дотриманням принципу платформонезалежності;

– розробити метод інсталяції програмного забезпечення на основі сценарного підходу;

– спроєктувати кросплатформний графічний інтерфейс користувача для забезпечення стабільного функціонування у різних операційних системах;

– розробити серверну інфраструктуру на базі архітектурного стилю REST API;

– провести експериментальну апробацію розроблених засобів розгортання програмного забезпечення.

Підсумовуючи, у розділі досліджено поточний стан інструментальних засобів управління пакунками на прикладі Apt, Pacman, Pamac та Winget. За результатами проведеного аналізу ідентифіковано низку концептуальних недоліків аналогів, серед яких: високий поріг входження та необхідність спеціалізованого навчання користувачів, архітектурна неможливість експлуатації єдиного пакетного менеджера в гетерогенних операційних системах, а також недосконалі механізми пошуку новітнього програмного забезпечення.

З огляду на виявлені проблеми, науково та практично доведено доцільність розробки універсальної системи автоматизованого генерування інсталяційних пакетів. Відповідно до визначеної мети визначено вичерпний перелік завдань, розв'язання яких є необхідною умовою для успішного проєктування та програмної реалізації розроблюваної системи автоматизованого генерування інсталяційних пакетів.

РОЗДІЛ 2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ГЕНЕРУВАННЯ ІНСТАЛЯЦІЙНИХ ПАКЕТІВ

2.1. Вибір мови програмування

Вибір технологічного стека для реалізації розроблюваного програмного забезпечення є критично важливим етапом проєктування, який детермінує подальший хід розробки, архітектурну гнучкість, масштабованість та зручність технічного супроводу. Для розв'язання задачі розробки системи автоматизованого генерування інсталяційних пакетів виникає об'єктивна необхідність у застосуванні спеціалізованого інструментарію, здатного забезпечити кросплатформність клієнтської частини та високу продуктивність серверної інфраструктури.

Для реалізації графічного інтерфейсу користувача було обрано високорівневу мову програмування Python у комбінації з фреймворком PyQt та середовищем візуального проєктування QtCreator. Застосування мови Python обґрунтовується її мультипарадигмальністю, зокрема повноцінною підтримкою об’єктно-орієнтованого, структурного та функціонального підходів до написання коду. Використання динамічної типізації, механізмів пізнього зв'язування та автоматизованого управління пам'яттю на основі алгоритмів збирання сміття суттєво пришвидшує процеси розробки програмних рішень. Архітектурна та ідеологічна філософія даної мови базується на фундаментальних принципах максимальної читабельності, експліцитності, лаконічності та переваги простоти над невиправданою складністю програмного коду [9].

Інтеграція спеціалізованої бібліотеки PyQt, розробленої компанією Riverbank Computing [10], забезпечує створення повноцінного графічного інтерфейсу, що гарантує ідентичність візуального представлення та стабільність функціонування у гетерогенних середовищах, включаючи Microsoft Windows, Linux та MacOS. З метою оптимізації процесу створення візуальних макетів застосовано інтегроване середовище QtCreator. Цей

інструмент дозволяє генерувати розмітку компонентів у форматі розширюваної мови розмітки без необхідності прямого програмування інтерфейсу на рівні вихідного коду. Крім того, наявність вбудованих засобів налагодження та конструювання форм значно підвищує загальну ефективність роботи над клієнтським застосунком.

Для розробки серверної частини комплексу, яка відповідає за обробку запитів, бізнес-логіку взаємодії та безпечне управління даними, обрано об'єктно-орієнтовану мову програмування C# у поєднанні з серверним фреймворком ASP.NET. Зазначена мова вирізняється строгою статичною типізацією та оптимізованим синтаксисом. Застосування платформи .NET є науково та практично обґрунтованим вибором для проєктування складних розподілених архітектур, оскільки це забезпечує високий рівень продуктивності, експлуатаційної надійності, а також закладає фундаментальні можливості для подальшого безперешкодного масштабування та модифікації розроблюваного програмного забезпечення.

2.2. Аналіз даних

У процесі функціонування програмне забезпечення здійснює безперервну взаємодію з інформаційними потоками. Фундаментальним завданням будь-якого програмного забезпечення є рецепція та обробка даних, що адаптується відповідно до специфічних потреб та кваліфікаційного рівня поточного користувача. Традиційно алгоритм роботи передбачає прийом вхідних даних у строго детермінованому форматі, їх подальшу алгоритмічну обробку та, за необхідності, генерацію вихідної інформації у формі, яка відповідає функціональному призначенню розв'язуваної задачі.

Одним із найбільш поширених методів організації клієнт-серверної взаємодії є використання архітектурного стилю REST API. Зазначений підхід розроблений з метою стандартизації проєктування та імплементації архітектури мережевих розподілених застосунків. Парадигма REST регламентує комплекс суворих обмежень щодо функціонування системної архітектури.

Концептуальною особливістю даного стилю є принцип відсутності збереження стану клієнта на серверному боці. Відповідно, кожен окремий запит до вебсервера є самодостатнім та містить усю необхідну інформацію для його повноцінної обробки. Взаємодія суб'єктів архітектури реалізується за допомогою трансляції HTTP-повідомлень, переважно із застосуванням формату обміну даними JSON.

Обмін інформацією між клієнтським та серверним вузлами відбувається за допомогою протоколу HTTP. Транзакції в межах даного протоколу поділяються на дві базові категорії: запити, які генеруються клієнтом з метою ініціації відповідних операцій, та відповіді, що формуються сервером за результатами обробки отриманого звернення.

Специфікація протоколу HTTP визначає множину стандартизованих методів, серед яких ключовими є GET, POST, PUT, DELETE, HEAD, CONNECT, OPTIONS, TRACE та PATCH [6].

Функціональне призначення основних HTTP-методів зводиться до наступного: – GET призначений для запиту репрезентації цільового ресурсу виключно з метою зчитування інформації без внесення змін на сервері, при цьому передача параметрів здійснюється через маршрутизатор або рядок запиту без використання тіла повідомлення; – POST застосовується для передачі сутностей до визначеного ресурсу, що переважно ініціює зміну стану системи або викликає побічні ефекти на сервері, на відміну від ідемпотентних методів; – PUT ініціює повну заміну поточного стану цільового ресурсу даними, що містяться в запиті, або створює новий ресурс у разі його відсутності з відповідною генерацією статус-коду; – DELETE виконує операцію знищення ідентифікованого ресурсу і характеризується властивістю ідемпотентності; – HEAD є аналогом методу GET, проте вимагає від сервера формування відповіді без тіла повідомлення, виключно з передачею заголовків, що доцільно застосовувати для попередньої оцінки характеристик ресурсу; – CONNECT використовується для встановлення тунельного з'єднання із сервером, переважно для реалізації протоколу HTTPS; – TRACE виконує функцію луна-

запиту до сервера для діагностики та налагодження маршрутів передачі даних; – PATCH призначений для застосування часткових модифікацій до вже існуючого ресурсу.

Формат JSON являє собою універсальний, незалежний від конкретної мови програмування текстовий стандарт обміну даними. Структура формату оптимізована для візуального сприйняття людиною та машинного синтаксичного аналізу, що зумовило його домінуюче становище у сфері сучасного електронного інформаційного обміну. Незважаючи на первинне походження від синтаксичних конструкцій мови JavaScript, інструментарій для генерації та обробки JSON-структур нині базово інтегровано у більшість сучасних технологічних стеків.

2.3. Розробка графічної схеми інтерфейсу програмного додатку

Під час проєктування графічного інтерфейсу користувача застосовано концепцію візуального мінімалізму, що передбачає інтеграцію строго обмеженої кількості графічних компонентів та базових елементів управління. Таке дизайнерське та архітектурне рішення є цілком обґрунтованим з огляду на специфіку розв'язуваної проблеми. Головний принцип побудови інтерфейсу системи автоматизованого генерування інсталяційних пакетів полягає у забезпеченні максимальної концентрації уваги оператора безпосередньо на процесах пошуку, аналізу та вибору необхідного програмного забезпечення, що вимагає усунення будь-яких відволікаючих візуальних чинників.

Схематичну структуру головного вікна розробленого програмного забезпечення наведено на рисунку 2.1. Відповідно до сформованих вимог щодо ергономіки та функціональності, базова робоча область інтерфейсу спроєктована таким чином, щоб комплексно об'єднувати ключові модулі взаємодії. Структура вікна обов'язково включає деталізований перелік доступних програмних продуктів, інтерактивні елементи керування життєвим циклом програмного забезпечення, модуль ієрархічної класифікації та вибору

категорій, а також інструментарій для здійснення швидкого контекстного пошуку цільових застосунків.

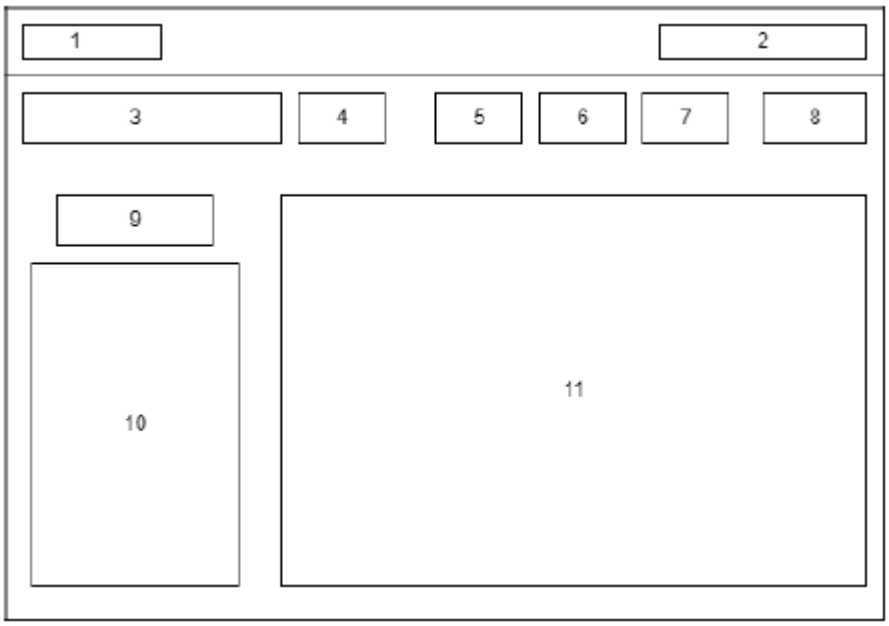


Рисунок 2.1 – Графічна схема головного вікна програмного забезпечення

Структурні елементи графічного інтерфейсу користувача включають:

1. ідентифікатор назви програмного забезпечення;
2. базові елементи управління вікном програмного забезпечення;
3. текстове поле для введення пошукових запитів;
4. елемент ініціалізації алгоритму пошуку програмного забезпечення;
5. інструмент активації процесу оновлення обраного програмного забезпечення;
6. інструмент запуску процедури інсталяції обраного програмного забезпечення;
7. інструмент ініціалізації процесу деінсталяції обраного програмного забезпечення;
8. елемент актуалізації загального переліку компонентів;
9. інструмент застосування фільтрів для відображення програмного забезпечення за визначеною категорією;
10. структурований класифікатор категорій програмного забезпечення;
11. візуалізований реєстр доступного програмного забезпечення.

Зважаючи на критичну вимогу щодо забезпечення кросплатформного функціонування розроблюваного рішення у різноманітних операційних системах, етап проєктування візуальної оболонки вимагає залучення спеціалізованого інструментарію. Відповідні технологічні засоби повинні гарантувати формування універсального графічного інтерфейсу користувача, що дозволить уникнути додаткових витрат ресурсів та часу на складні процеси адаптації і портування коду розроблюваного програмного забезпечення між різними системними архітектурами.

2.4. Розробка веб серверу для оновлення даних додатку

З метою забезпечення користувачів актуальними реєстрами програмного забезпечення без потреби постійного оновлення клієнтських застосунків, обґрунтовано доцільність проєктування та розгортання спеціалізованого вебсервера. Зазначене архітектурне рішення надає можливість динамічного отримання найновішої інформації щодо наявного програмного забезпечення, а також актуальних алгоритмічних інструкцій для його розгортання на кінцевих пристроях.

Програмну реалізацію серверної частини комплексу здійснено в інтегрованому середовищі розробки Visual Studio 2022 із застосуванням високорівневої мови програмування C# та інфраструктурного фреймворку ASP.NET.

Мова C# є об'єктно-орієнтованою мовою програмування загального призначення, що функціонує на базі платформи .NET, характеризується строгою статичною типізацією та використовує механізми динамічної компіляції під час виконання. Технологія стандартизована міжнародними консорціумами відповідно до специфікацій ECMA-334 та ISO/IEC 23270, які декларують такі фундаментальні принципи розробки [7]:

– мова повинна відповідати критеріям лаконічності, сучасності та повної підтримки об'єктно-орієнтованої парадигми;

– обов'язкова наявність строгої типізації, інтегрованих механізмів автоматичного збирання сміття та запобігання доступу до неініціалізованих або вивільнених ділянок пам'яті;

– пріоритетність забезпечення високої продуктивності процесу розробки та гарантування експлуатаційної надійності програмного забезпечення;

– забезпечення архітектурної кросплатформності та портативності виконуваного коду.

Технологія ASP.NET становить собою відкритий інфраструктурний фреймворк, розроблений корпорацією Microsoft для проєктування мережевих застосунків. Платформа підтримує множину парадигм програмування та надає інструментарій для створення масштабованих мережевих рішень, що відзначаються високим рівнем стабільності та зручністю технічного супроводу [8].

Зазначений фреймворк забезпечує підтримку таких архітектурних моделей:

– ASP.NET Web Forms – технологія конструювання серверної частини для динамічної генерації та обробки сторінок;

– ASP.NET MVC – архітектурний підхід до проєктування веб-застосунків на основі патерна "модель-вигляд-контролер";

– ASP.NET Web Pages – механізм інтеграції серверного виконуваного коду безпосередньо у статичну розмітку HTML;

– ASP.NET Web API – спеціалізований інструментарій для розробки серверних служб на основі архітектурного стилю REST API.

У контексті реалізації серверної архітектури розроблюваної системи було обрано модель ASP.NET Web API. Початковим етапом проєктування є конструювання маршрутизаторів, за допомогою яких клієнтські запити ініціюватимуть отримання класифікованого переліку категорій програмного забезпечення, що ілюструє рисунок 2.2.

```
[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

Для реалізації даного функціоналу застосовано HTTP-метод GET, специфікація якого передбачає виключно вилучення інформаційних масивів із серверного середовища без внесення змін до його поточного стану.

Наступним етапом проєктування маршрутизації є алгоритм отримання реєстрів програмного забезпечення із супровідними метаданими, що динамічно формуються відповідно до обраної класифікаційної категорії, як це проілюстровано на рисунку 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 2.3 – Метод отримання списку програмного забезпечення

Для програмної реалізації вказаного функціоналу застосовано HTTP-метод GET. Відповідний запит ініціюється з передачею аргументу у формі числового ідентифікатора категорії, що слугує критерієм для цільової вибірки програмного забезпечення.

Подальша алгоритмічна організація клієнт-серверної взаємодії передбачає проєктування маршрутизаторів, за допомогою яких клієнтський застосунок отримуватиме доступ до сценаріїв інсталяції, що представлено на рисунку 2.4, оновлення, відповідно до рисунку 2.5, та деінсталяції програмного забезпечення, що ілюструє рисунок 2.6.

Синтез методу отримання сценарних алгоритмів розгортання також базується на застосуванні HTTP-методу GET. Згідно зі стандартизованою специфікацією протоколу, конструкція такого запиту формується без включення тіла повідомлення та орієнтована виключно на вилучення інформаційних масивів із сервера.

За результатами успішної обробки запиту серверна частина генерує відповідь у форматі рядкового типу даних, контент якого становить безпосередній код інсталяційного сценарію.

Вхідна сигнатура зазначеного методу вимагає передачі двох ключових параметрів:

– osTypeId – числовий ідентифікатор, що детермінує архітектурний тип операційної системи на апаратному забезпеченні користувача;

– softwareId – числовий ідентифікатор цільового програмного забезпечення, яке підлягає розгортанню у клієнтському середовищі.

```
[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

Програмну реалізацію методу отримання сценарних алгоритмів оновлення здійснено із застосуванням HTTP-методу GET. Відповідно до стандартної специфікації протоколу, такий запит формується без використання тіла повідомлення та спеціалізується виключно на процесах вилучення інформаційних ресурсів із серверної інфраструктури.

Результатом успішної обробки запиту є генерація серверною частиною відповіді у форматі рядкового типу даних, контент якого становить безпосередній виконуваний код запитуваного сценарію.

Запропонований метод містить інтегровані механізми обробки виняткових ситуацій та забезпечення стійкості до програмних відмов. Організація процесу взаємодії з клієнтським застосунком, який функціонує на базі графічного інтерфейсу користувача, реалізується шляхом трансляції стандартизованих статус-кодів протоколу HTTP.

У випадку виникнення помилок під час зчитування інформації з бази даних, що спричинено некоректністю або невалідністю надісланих вхідних

параметрів, алгоритм вебсервера ініціює формування відповіді з ідентифікатором статусу Bad Request.

У разі реєстрації непередбачених критичних збоїв або неідентифікованих винятків у процесі виконання серверної обробки запиту, програмний комплекс генерує відповідь із статус-кодом Internal Server Error, що сигналізує про виникнення внутрішньої помилки сервера.

```
[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

Візуалізацію алгоритмічної структури методу, що відповідає за формування запитів на отримання сценаріїв оновлення, наведено на рисунку 2.5. Зазначений функціонал відіграє критичну роль у підтримці актуальності розгорнутого програмного забезпечення. Механізм обробки функціонує за аналогією до попередньо розглянутих маршрутів, забезпечуючи трансляцію ідентифікаторів операційної системи та конкретного застосунку для завантаження точного алгоритму. Отриманий сценарний код здійснює безпечний перехід до новітньої версії цільового продукту із забезпеченням цілісності попередньо збережених конфігурацій.

```
[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

Графічне представлення архітектури методу запиту сценарних інструкцій для деінсталяції проілюстровано на рисунку 2.6. Процедура коректного видалення є невід'ємною складовою повного життєвого циклу управління програмним забезпеченням. Реалізація даного методу дозволяє системі автоматизованого генерування інсталяційних пакетів динамічно ініціювати завантаження оптимізованих скриптів. Відповідні алгоритми гарантують комплексне очищення цільового пристрою від залишкових файлів, конфігураційних записів та асоційованих залежностей, що оптимізує використання дискового простору та запобігає деградації показників стабільності операційної системи.

2.5. Розробка алгоритмів роботи додатку

Узагальнений алгоритм функціонування системи автоматизованого генерування інсталяційних пакетів передбачає послідовну реалізацію комплексу взаємопов'язаних етапів. Початкова стадія процесу полягає в ініціалізації та рецепції масиву вхідних даних із подальшою їхньою алгоритмічною обробкою. На основі результатів аналізу ініціюється виконання цільових процедур розгортання, деінсталяції або актуалізації програмного

забезпечення, вибір яких строго детермінується ідентифікованими намірами та керуючими впливами користувача. На завершальному етапі циклу обробки програмне забезпечення здійснює формування та виведення підсумкової інформації, що відображає кінцевий статус реалізації попередньо ініційованих операцій (рис. 2.7.).

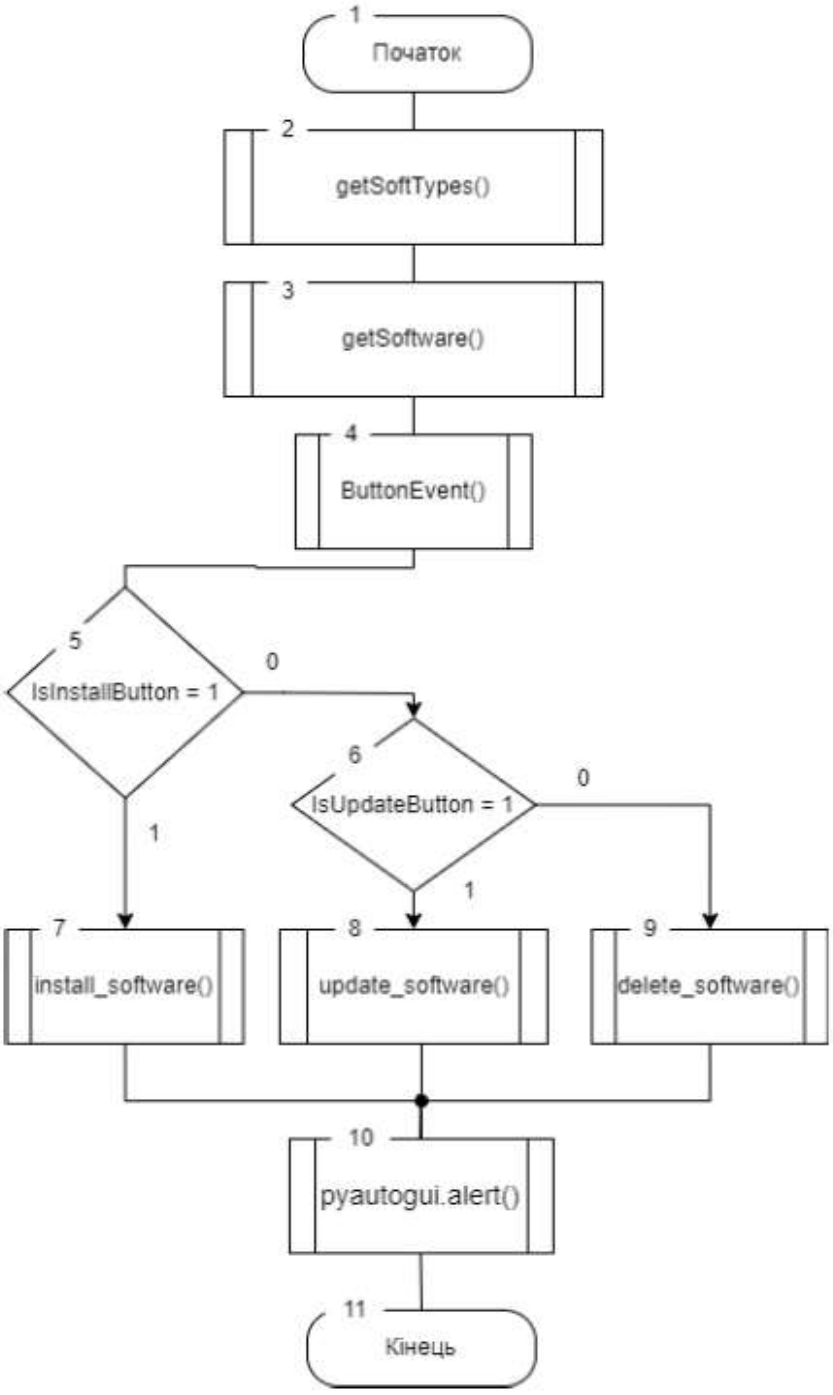


Рисунок 2.7. - Блок-схема узагальненого алгоритму функціонування програмного забезпечення

Подальшим етапом проєктування є розробка алгоритму отримання реєстру категорій. Зазначений алгоритмічний механізм призначений для динамічного вилучення актуальної інформації з віддаленого вебсервера. Передача масивів даних здійснюється у стандартизованому текстовому форматі JSON, що забезпечує клієнтську частину програмного забезпечення вичерпним набором відомостей, необхідних для коректної та повноцінної взаємодії користувача із системою (рис. 2.8.).

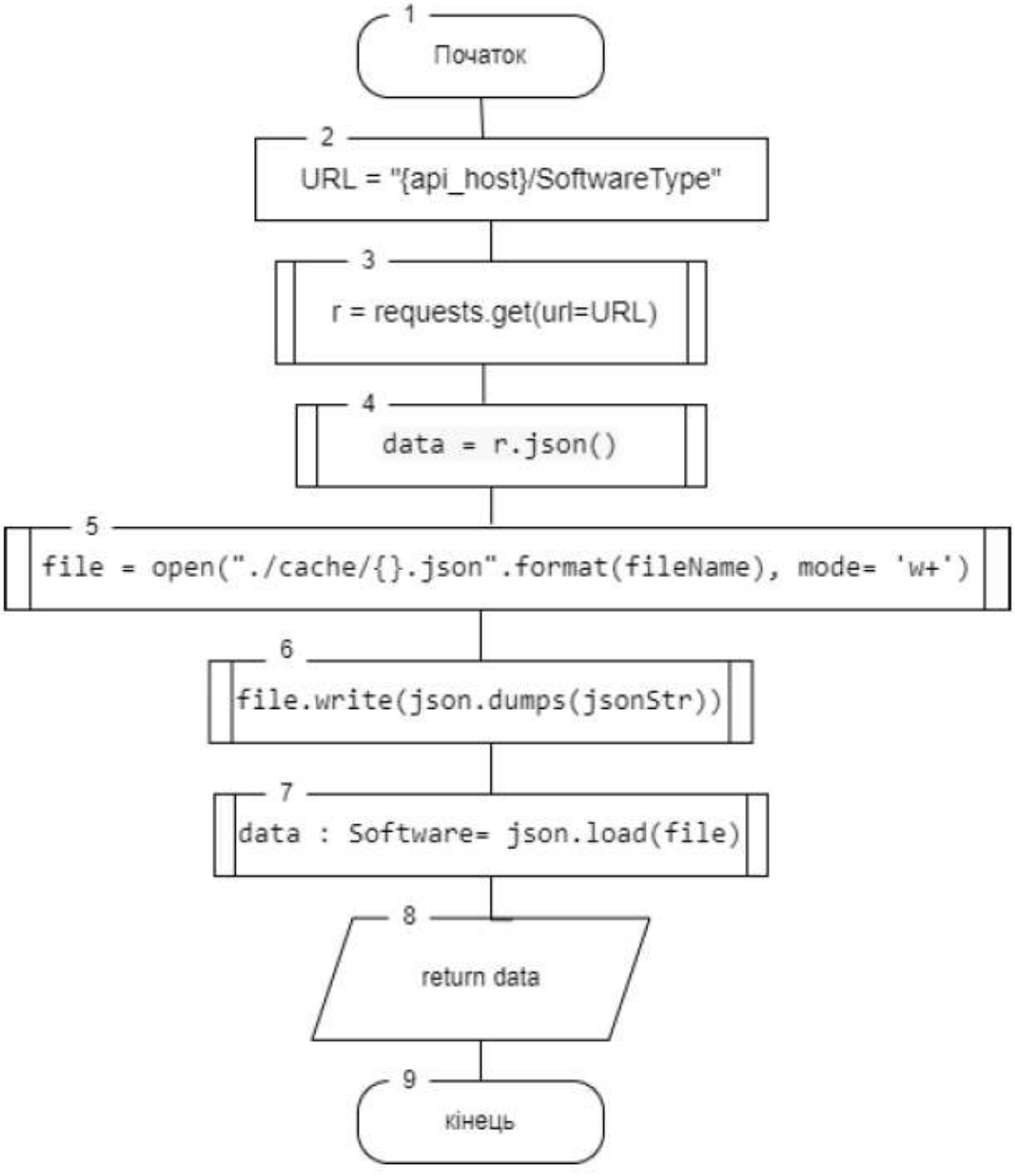


Рисунок 2.8 –Алгоритм оновлення списку категорій

Поряд із реалізацією базового функціоналу класифікації, невід'ємним етапом проєктування є забезпечення механізму безперервної актуалізації реєстрів програмного забезпечення. Зазначений процес здійснюється шляхом організації динамічної мережевої взаємодії із відповідними маршрутизаторами віддаленого веб-сервера. Інтеграція такого алгоритмічного рішення гарантує надання кінцевому користувачеві оперативного доступу до вичерпного та найбільш достовірного переліку доступних програмних продуктів, що цілком нівелює ризики оперування неактуальною інформацією під час експлуатації розробленої системи.

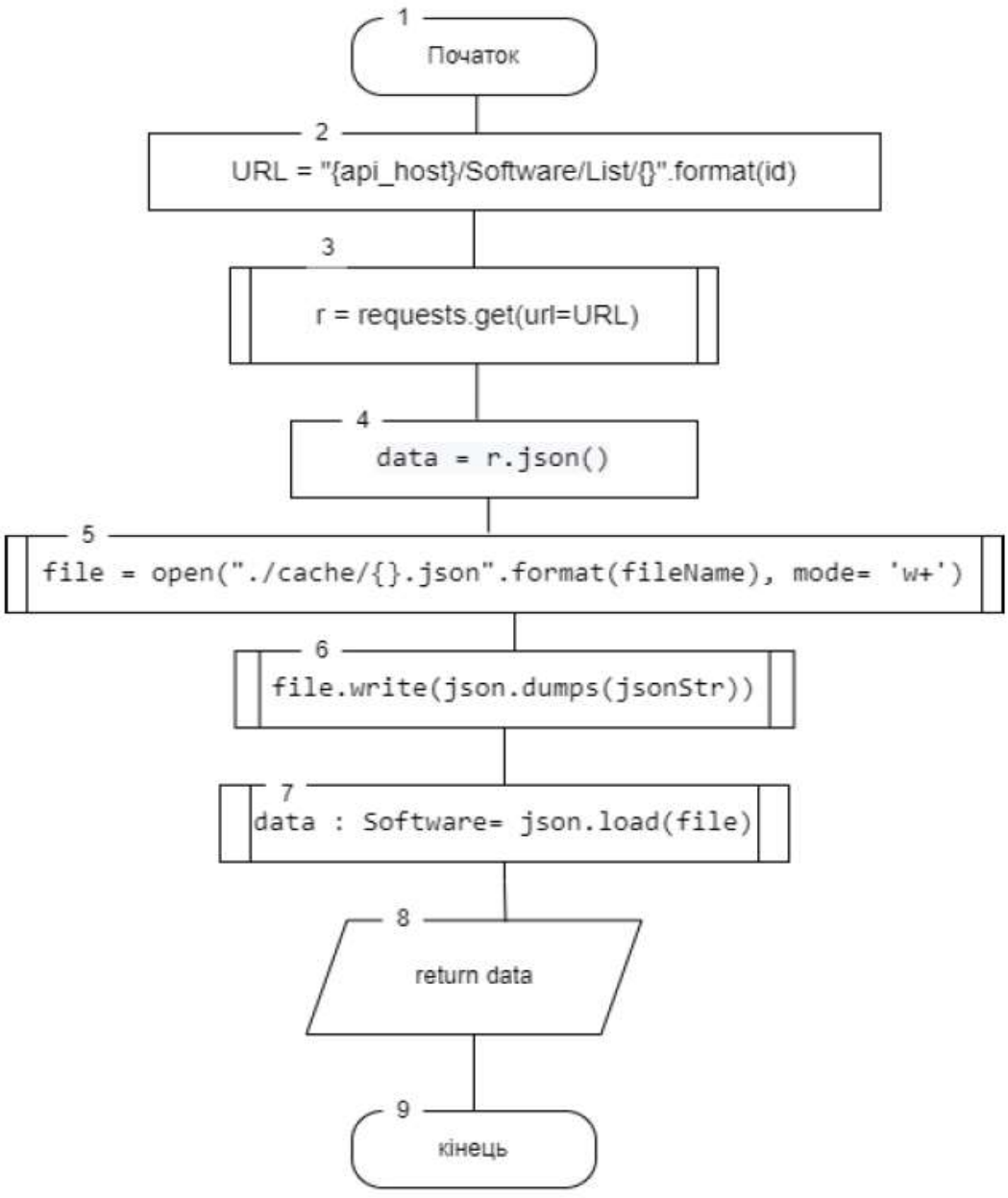


Рисунок 2.9 – Алгоритм оновлення списків програмного забезпечення

Завершальним етапом проєктування є синтез алгоритмічного забезпечення, призначеного для реалізації безпосередніх процесів інсталяції, актуалізації та деінсталяції програмного забезпечення. Структурно-логічні схеми зазначених алгоритмів розгортання, оновлення та видалення детально проілюстровано на рисунках 2.10, 2.11 та 2.12 відповідно.

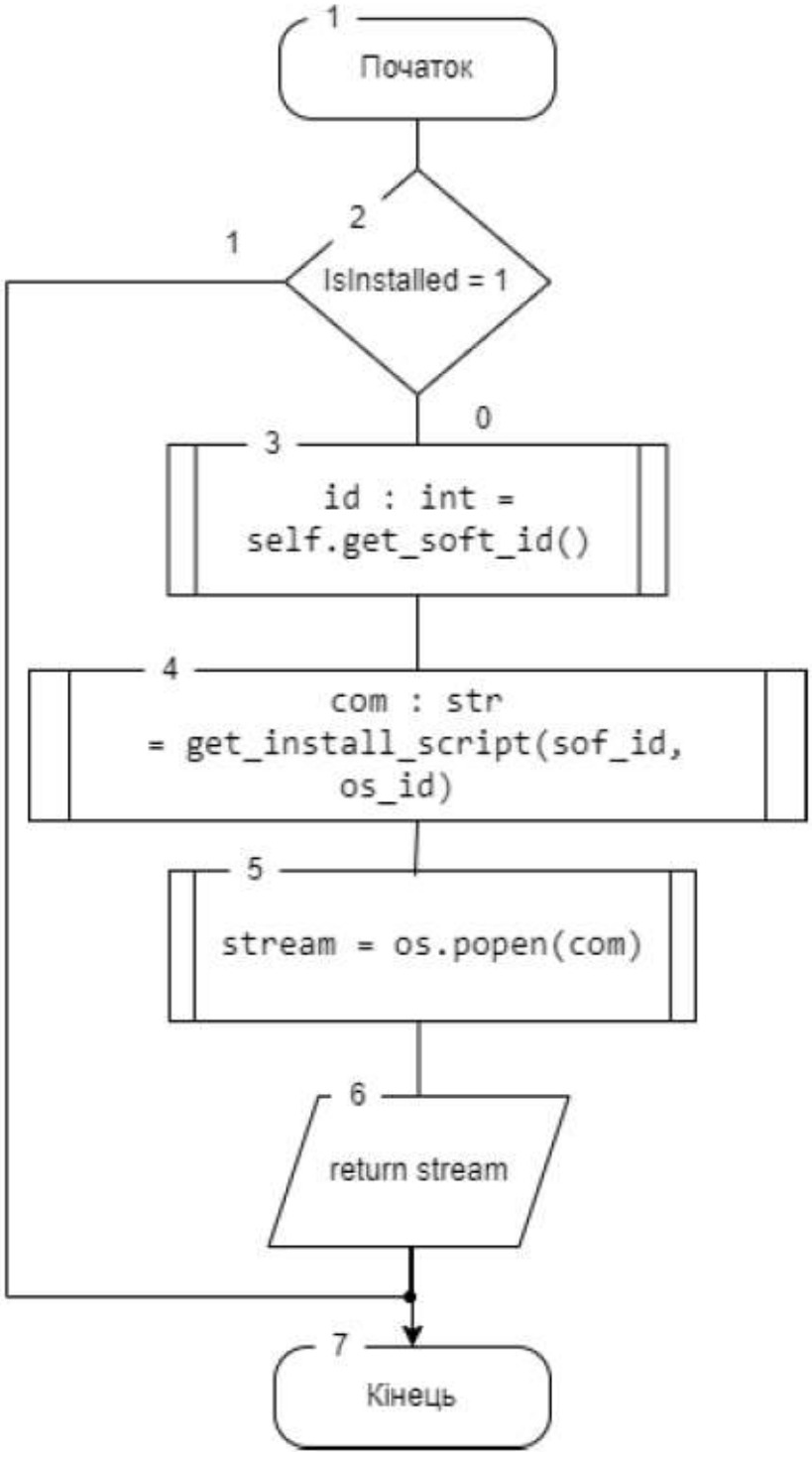


Рисунок 2.10 – Блок-схема алгоритму інсталяції програмного забезпечення

Представлений алгоритм ініціює первинну перевірку статусу обраного програмного забезпечення щодо його попереднього розгортання в операційній системі. На основі результатів верифікації клієнтський застосунок формує та надсилає цільовий запит до віддаленого веб-сервера з метою отримання актуальних інсталяційних сценаріїв. Після успішного завантаження необхідних алгоритмічних інструкцій система автоматично виконує процедуру встановлення обраного програмного продукту на пристрої користувача.

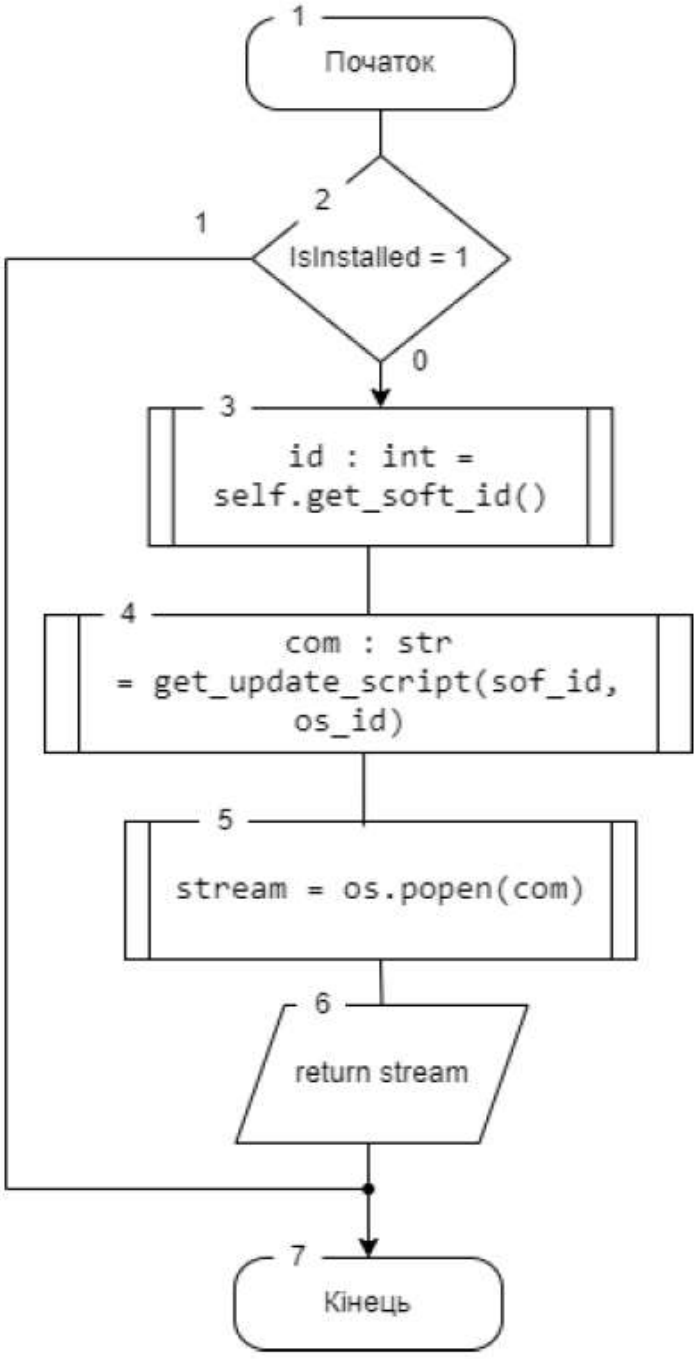


Рисунок 2.11 – Алгоритм оновлення програмного забезпечення

Запропонований алгоритмічний механізм ініціює попередню верифікацію факту наявності розгорнутого програмного забезпечення на цільовому апаратному пристрої користувача. На основі результатів зазначеної системної перевірки клієнтський застосунок формує та спрямовує відповідний мережевий запит до віддаленого веб-сервера. Головною метою ініційованої транзакції є завантаження актуальних сценарних інструкцій, які регламентують та забезпечують коректне виконання процедури оновлення програмного забезпечення.

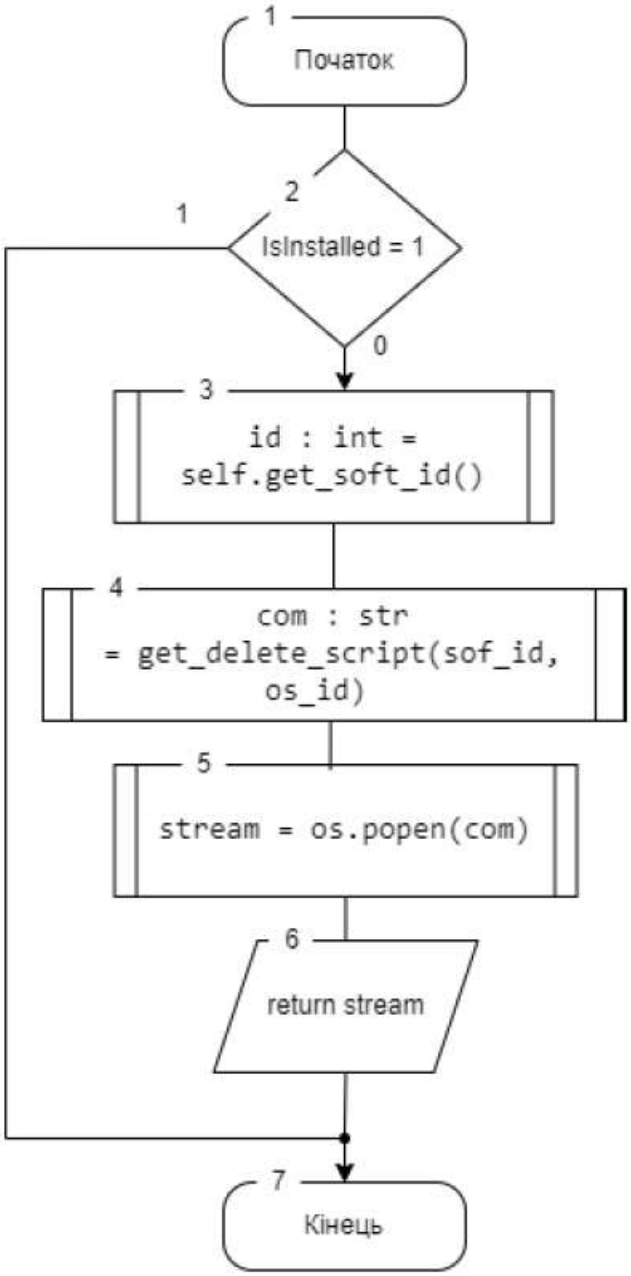


Рисунок 2.13 – Алгоритм деінсталяції програмного забезпечення

Представлений алгоритмічний механізм здійснює первинну верифікацію наявності встановленого програмного забезпечення на цільовому апаратному пристрої користувача. За результатами цієї системної перевірки ініціюється формування та спрямування цільового мережевого запиту до віддаленого веб-сервера з метою завантаження специфічних інструкцій для процедури деінсталяції. Отримані алгоритми динамічно адаптуються відповідно до архітектурних особливостей операційної системи клієнтського середовища, що гарантує коректне виконання операції.

Підсумовуючи результати даного етапу проєктування, здійснено комплексний аналіз структури вхідних інформаційних потоків та науково обґрунтовано оптимальні методи інтерактивної взаємодії користувача з програмним забезпеченням. Спроєктовано кросплатформний графічний інтерфейс користувача та синтезовано ключові алгоритми функціонування розроблюваного рішення. У рамках побудови архітектури реалізовано базу даних, призначену для централізованого та безпечного зберігання інсталяційних сценаріїв, а також розгорнуто відповідну серверну інфраструктуру.

Для забезпечення повноцінної експлуатації розроблюваної системи автоматизованого генерування інсталяційних пакетів формалізовано базовий алгоритм функціонування програмного забезпечення, що доповнюється спеціалізованими алгоритмічними механізмами для безперервної актуалізації класифікаційних списків, розгортання, оновлення та повного видалення цільових програмних продуктів.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОГО ГЕНЕРУВАННЯ ІНСТАЛЯЦІЙНИХ ПАКЕТІВ

3.1. Реалізація інтерфейсу користувача

Програмну реалізацію здійснено із застосуванням високорівневої мови програмування Python та спеціалізованої бібліотеки PyQt. Вибір вказаного технологічного інструментарію обґрунтовується необхідністю оптимізації часових витрат на проєктування клієнтської частини та забезпеченням повноцінної кросплатформності інтерфейсу користувача. Такий підхід дає змогу нівелювати додаткові трудовитрати, пов'язані з портуванням розробленого програмного забезпечення на альтернативні архітектури та операційні системи.

Початковий етап візуального проєктування передбачає формування конфігураційного файлу у стандарті розширюваної мови розмітки XML із використанням інтегрованого середовища QtCreator. Зазначений файл виконує функцію декларативного опису графічного інтерфейсу, архітектура якого безпосередньо конструюється за допомогою вбудованого візуального редактора форм.

Першочерговим завданням у процесі побудови клієнтської оболонки є ініціалізація та програмне конструювання об'єкта головного вікна, візуальне представлення якого ілюструє рисунок 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 програмного забезпечення

Наступним етапом конструювання графічного інтерфейсу є інтеграція спеціалізованого структурного компонента, функціональне призначення якого полягає у централізованому зберіганні та візуальному відображенні систематизованого переліку категорій. Зазначений об'єкт архітектурно виконує роль базового навігаційного інструмента, що забезпечує користувачеві ергономічний доступ до класифікованих груп програмного забезпечення та суттєво оптимізує процеси взаємодії з інформаційними масивами розробленої системи. Графічне представлення доданого візуального компонента та його просторове розташування у межах робочої області детально проілюстровано на рисунку 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 етапом проєктування графічного інтерфейсу є ініціалізація та конфігурація структурного компонента, функціональне призначення якого полягає у формуванні та візуальному відображенні реєстру доступного програмного забезпечення, призначеного для безпосередньої взаємодії та здійснення цільового вибору кінцевим користувачем. Графічне представлення зазначеного об'єкта детально проілюстровано на рисунку 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

З метою розширення функціональних можливостей графічного інтерфейсу та забезпечення повноцінної інтерактивної взаємодії, архітектура візуальної оболонки передбачає інтеграцію спеціалізованих елементів управління обраним програмним забезпеченням. Візуальне подання компонентів, призначених для ініціалізації процесів актуалізації версій, безпосереднього розгортання та повної деінсталяції, детально проілюстровано на рисунку 3.4.

```
<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 елементів управління програмним забезпеченням

Підсумковий візуальний результат інтеграції та просторового компонування всіх розроблених структурних компонентів графічного

інтерфейсу користувача в межах єдиного робочого середовища детально проілюстровано на рисунку 3.5.

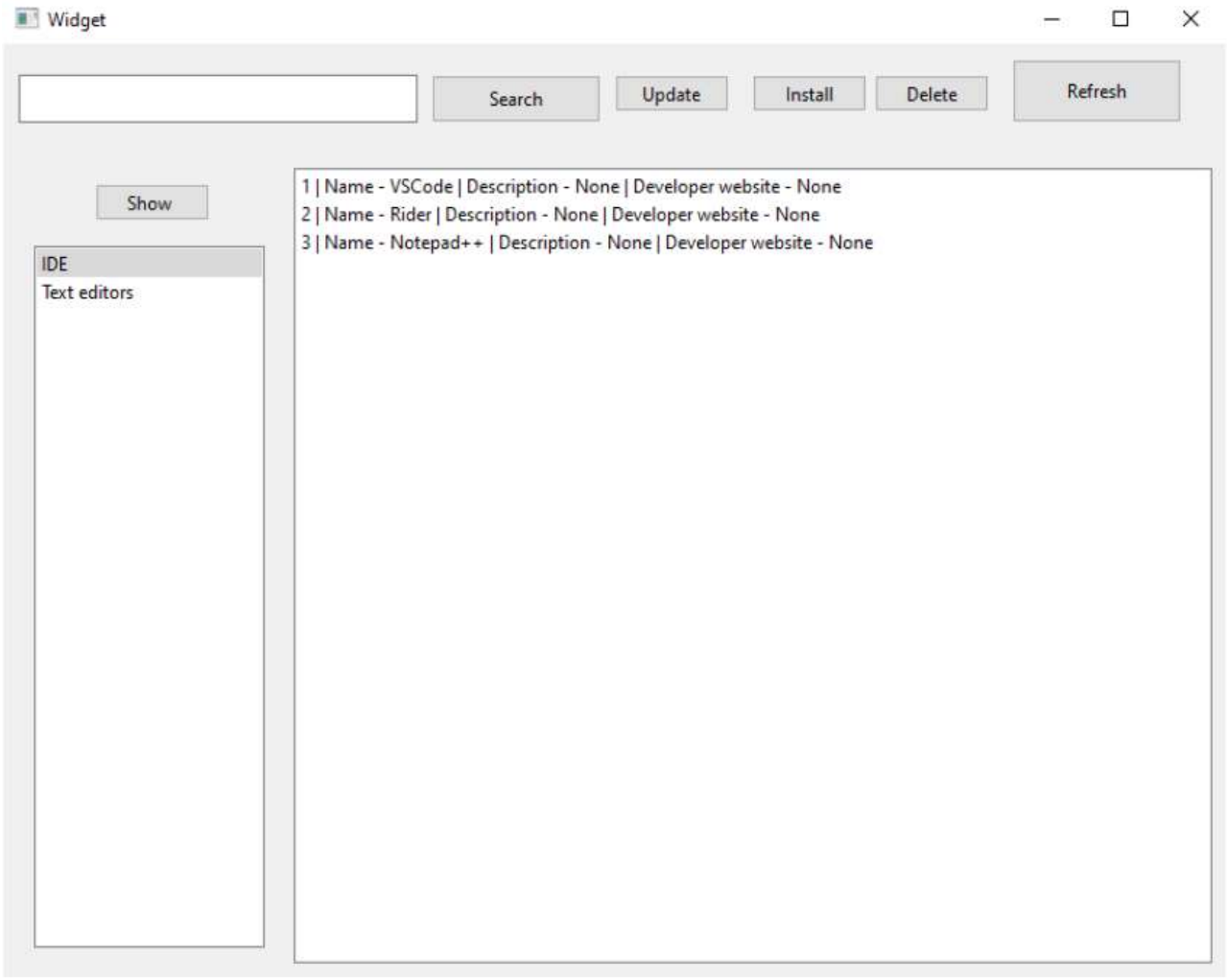


Рисунок 3.5 – Робоче середовище програмного забезпечення

3.2. Реалізація бази даних для зберігання відомостей про програмне забезпечення

Для реалізації інформаційного сховища застосовано вільно поширювану об'єктно-реляційну систему керування базами даних PostgreSQL. Зазначена програмна платформа повною мірою задовольняє вимоги транзакційності, гарантуючи атомарність, узгодженість, ізольованість та довговічність збереження інформаційних масивів. Архітектура даної системи управління базами даних орієнтована на забезпечення високого рівня масштабованості та вирізняється кросплатформністю, що уможливлює її безперешкодне розгортання в гетерогенних операційних середовищах.

З метою ініціалізації логічної структури бази даних розроблено спеціалізований SQL-сценарій. Застосування такого підходу гарантує можливість оперативної міграції інформаційної моделі між різними апаратними та системними вузлами без необхідності залучення додаткових інструментальних засобів. Візуальне представлення програмного коду зазначеного сценарію, що регламентує генерування системи реляційних таблиць, наведено 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

З метою встановлення логічних реляційних зв'язків між структурованими масивами інформації та гарантування посилальної цілісності даних виникає об'єктивна необхідність визначення відповідних ключових обмежень. Програмний код сценарію, який реалізує імплементацію зазначених структурних елементів та обмежень у систему бази даних, детально проілюстровано 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 додавання ключових обмежень

У результаті послідовної імплементації наведених конфігураційних скриптів здійснюється остаточне формування реляційної структури інформаційної бази. Відповідну концептуальну модель даних, візуалізовану у вигляді ER-діаграми, детально проілюстровано на рисунку 3.9.

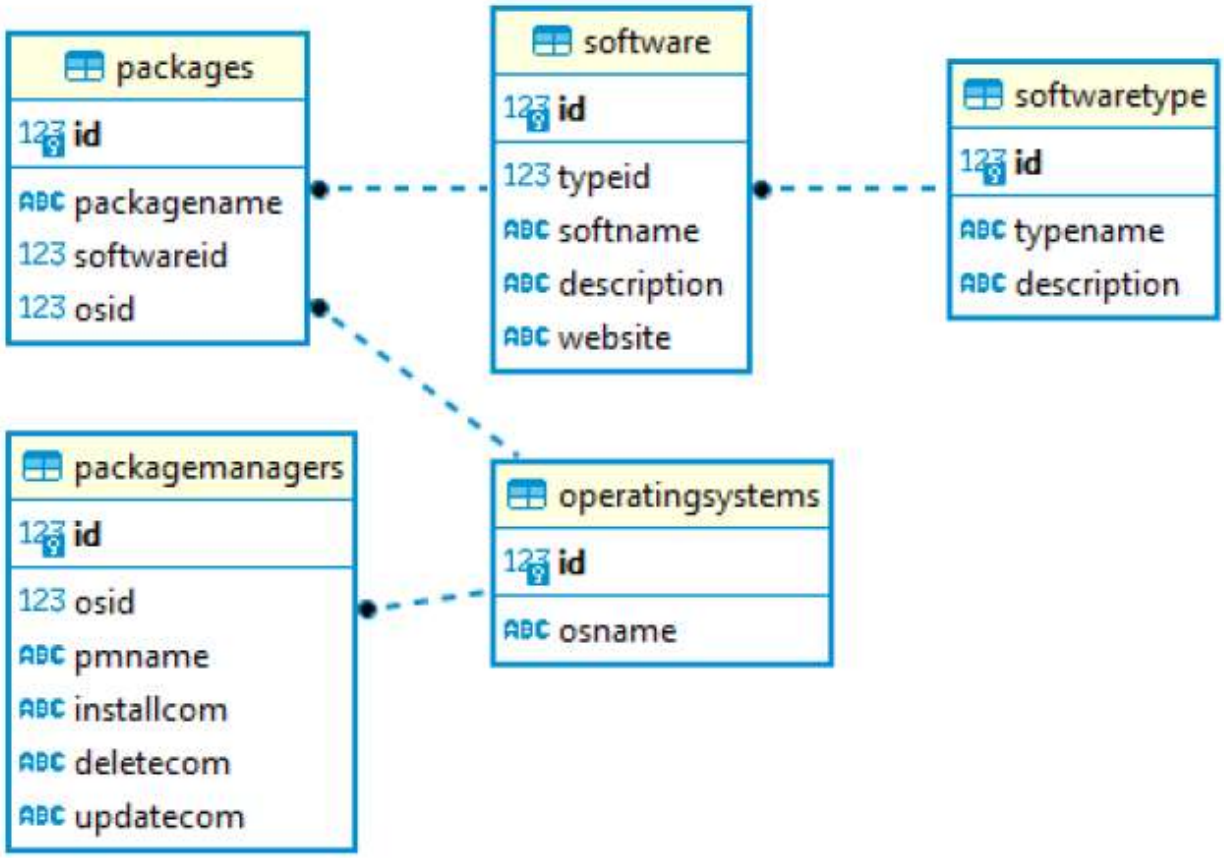


Рисунок 3.9 – Концептуальна модель реляційної бази даних у вигляді ER-діаграми

3.3. Розробка серверної частини для забезпечення доступу до даних

З метою реалізації механізмів вилучення інформації з реляційної бази даних через визначені маршрутизатори вебсервера застосовано високорівневу мову програмування C# у комплексі з інфраструктурним фреймворком ASP.NET та мікро-об'єктно-реляційним перетворювачем Dapper.

Першочерговим завданням у межах побудови серверної логіки є алгоритмічна імплементація процедури отримання класифікаційних типів програмного забезпечення, програмна структура якої ініціюється без передачі жодних вхідних параметрів. Подальшим етапом є розробка механізму динамічного формування реєстру цільового програмного забезпечення. Сигнатура цього запиту вимагає обов'язкової передачі числового

ідентифікатора відповідної категорії, що слугує базовим критерієм для здійснення інформаційної вибірки з бази даних.

Програмну реалізацію зазначених методів обробки клієнтських запитів детально проілюстровано на рисунках 3.10 та 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 – Програмна реалізація методів зчитування інформації щодо категорій та доступного програмного забезпечення

На рисунку 3.11 наведено програмну реалізацію асинхронного методу, функціональне призначення якого полягає у динамічному генеруванні цільових скриптових інструкцій для управління програмним забезпеченням. Вхідна сигнатура зазначеного методу передбачає передачу трьох ключових параметрів: ідентифікатора типу необхідного сценарію, унікального числового ідентифікатора програмного забезпечення та ідентифікатора цільової операційної системи клієнта.

```
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 – Програмна реалізація методу формування сценарного алгоритму для користувача

Алгоритмічна послідовність виконання методу розпочинається з ініціалізації та відкриття безпечного асинхронного з'єднання з реляційною базою даних. На наступному етапі, із застосуванням логічної конструкції перевірки відповідності шаблонам, здійснюється детермінація цільового стовпця бази даних залежно від запитуваної системної дії. Відповідно до переданого типу сценарію алгоритм обирає атрибут, що містить базову команду для інсталяції, оновлення або деінсталяції. У разі передачі непередбаченого типу сценарію генерується виняткова ситуація про неприпустимість операції.

Подальша обробка передбачає формування та асинхронне виконання двох послідовних структурованих запитів до інформаційного сховища. Перший запит спрямований на вилучення синтаксичної основи команди пакетного менеджера, яка відповідає ідентифікованій операційній системі користувача. Другий мережевий запит здійснює пошук та вилучення точного

номенклатурного імені інсталяційного пакета, що суворо асоційований із визначеним програмним продуктом та архітектурою операційної системи.

Отримані текстові компоненти конкатенуються за допомогою спеціалізованого об'єкта для побудови рядків, утворюючи цілісний та синтаксично коректний виконуваний сценарій. З метою гарантування експлуатаційної надійності серверної інфраструктури вся описана бізнес-логіка інкапсульована у блок перехоплення та обробки виняткових ситуацій. У випадку виникнення критичних збоїв під час підключення або зчитування інформації метод коректно перериває виконання операції та повертає порожнє значення, тим самим запобігаючи аварійному завершенню роботи серверного застосунку.

3.4. Тестування розробленого програмного забезпечення

Процес тестування програмного забезпечення являє собою комплекс заходів, спрямованих на верифікацію поведінкових характеристик застосунку в гетерогенних умовах, що імітують реальні сценарії взаємодії користувача із системою. Проведення тестових випробувань забезпечує формування об'єктивної та незалежної оцінки розробленого рішення, дозволяючи ідентифікувати його експлуатаційні переваги та функціональні недоліки [11].

У сучасній інженерії програмного забезпечення виокремлюють низку базових технік тестування: – інспекція внутрішньої архітектури та загального проєктування програмного продукту; – безпосереднє виконання програмного коду із супутнім аналізом його поведінкових патернів; – залучення спеціалізованих інструментальних засобів для моніторингу функціонування застосунку; – аудит апаратно-програмної інфраструктури, необхідної для розгортання та експлуатації програмного забезпечення.

Відповідно до загальноприйнятих стандартів, життєвий цикл розробки передбачає класифікацію процесу тестування за ієрархічними рівнями. До найбільш розповсюджених належать: – модульне тестування, об'єктом якого є верифікація ізольованих фрагментів вихідного коду; – інтеграційне тестування,

орієнтоване на перевірку коректності взаємодії між модулями та загальної структурної цілісності; – системне тестування, що забезпечує комплексний аналіз відповідності програмного забезпечення заявленим функціональним та нефункціональним вимогам.

Методологічний апарат перевірки якості програмного забезпечення базується на кількох фундаментальних підходах.

Статичне тестування охоплює процеси експертного огляду та інспекції вихідного коду без його безпосереднього виконання. До цієї категорії також належать інтегровані аналітичні інструменти середовищ розробки, які здійснюють безперервний синтаксичний та семантичний контроль.

Динамічне тестування реалізується через виконання наперед визначених тестових сценаріїв, що формалізують можливі варіанти взаємодії користувача з програмним забезпеченням.

Дослідницьке тестування являє собою евристичний підхід, який поєднує процеси вивчення продукту, проєктування тестових випадків та їх синхронне виконання. Цей метод сфокусований на емпіричному аналізі реакцій системи на нетривіальні сценарії експлуатації.

Тестування білої скриньки передбачає глибинний аналіз внутрішньої логічної структури та архітектурної імплементації програмного коду.

Тестування чорної скриньки фокусується виключно на верифікації зовнішньої поведінки та функціональності програмного забезпечення, абстрагуючись від його внутрішньої реалізації.

Тестування на основі порівняння результатів полягає у систематичному співставленні фактичних вихідних даних, згенерованих системою, з еталонними очікуваними значеннями.

У контексті апробації розробленої системи автоматизованого генерування інсталяційних пакетів було обґрунтовано доцільність застосування методів тестування чорної скриньки та стратегії порівняння вихідних результатів.

Верифікація графічного інтерфейсу користувача становить спеціалізований етап тестування, спрямований на підтвердження відповідності

візуальної оболонки ергономічним та функціональним специфікаціям. Реалізація цього етапу здійснюється шляхом моделювання та імітації різноманітних експлуатаційних сценаріїв, що відображають визначені вимоги до системи.

Під час ініціалізації розробленого програмного забезпечення на головному екрані формується візуальний простір, що містить ієрархічний класифікатор категорій, інструментальну панель управління обраними елементами та рядок контекстного пошуку, призначений для оперативної ідентифікації необхідних компонентів. Графічне подання структури головного вікна детально проілюстровано на рисунку 3.12.

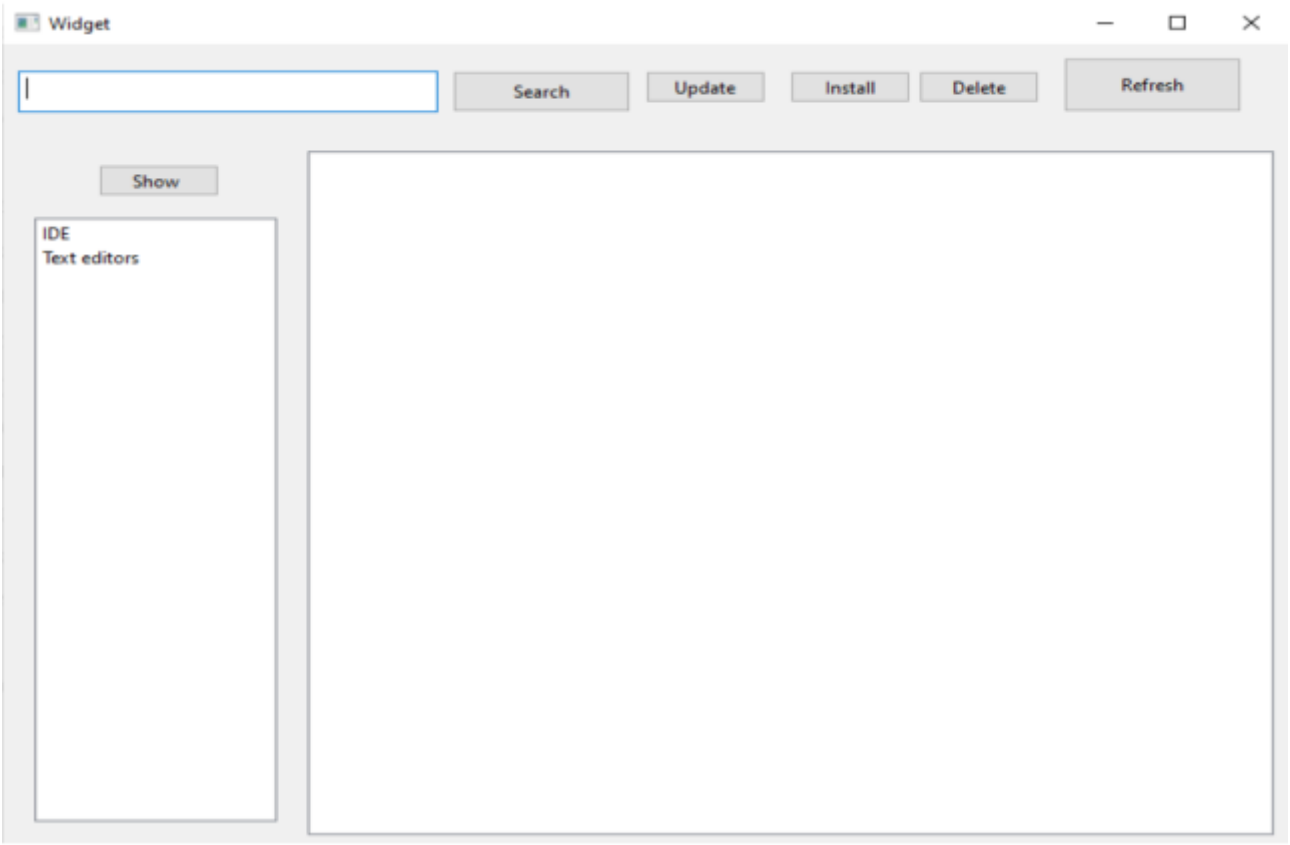


Рисунок 3.12 – Ініціалізаційна форма головного вікна програмного забезпечення

З метою візуалізації переліку доступних застосунків користувач повинен здійснити вибір відповідної класифікаційної категорії із запропонованого ієрархічного реєстру. Після ідентифікації цільової групи виникає необхідність ініціювати алгоритм відображення даних шляхом активації графічного елемента управління "Show". Виконання зазначеної дії генерує системний

запит, результати обробки якого автоматично виводяться у спеціалізованому візуальному компоненті у вигляді структурованого списку програмного забезпечення. Стан графічного інтерфейсу після успішного виконання описаної операції детально проілюстровано на рисунку 3.13.

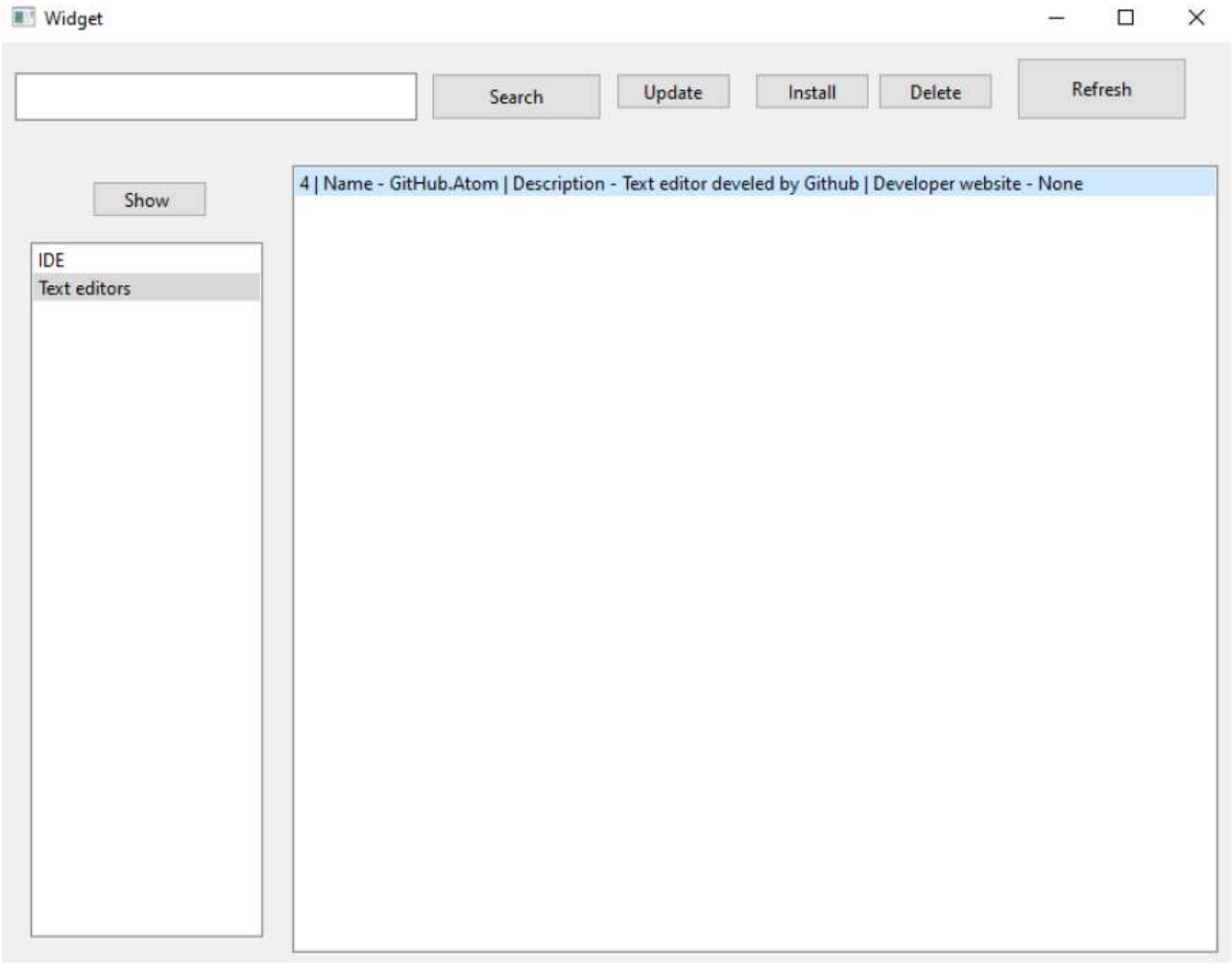


Рисунок 3.13 – Результат візуалізації реєстру програмного забезпечення

Для ініціалізації процедури розгортання програмного забезпечення користувачу необхідно здійснити вибір відповідного цільового елемента зі згенерованого реєстру та активувати графічний інструмент управління «Install». Наслідком виконання зазначеної транзакції є автоматизоване встановлення обраного програмного продукту на локальний апаратний пристрій, що супроводжується відповідним системним сповіщенням на рівні операційної системи. З метою емпіричної верифікації коректності функціонування розроблених алгоритмів було проведено тестове розгортання утиліти Notepad++, яка класифікується як розширений текстовий редактор. По

завершенню інсталяційного процесу система генерує візуальне повідомлення, що підтверджує успішне виконання операції, результат якої детально проілюстровано на рисунку 3.14.



Рисунок 3.14 – Системне повідомлення про успішну інсталяцію програмного забезпечення

Для ініціалізації процесу деінсталяції користувач має ідентифікувати та обрати відповідний цільовий об'єкт із запропонованого реєстру, після чого активувати графічний інструмент управління «Delete». З метою емпіричної верифікації коректності функціонування алгоритму видалення було застосовано раніше розгорнутий програмний продукт Notepad++. По завершенню процедури деінсталяції система формує відповідне візуальне сповіщення щодо успішного вилучення програмного забезпечення з локального апаратного пристрою. Зафіксований результат виконання зазначеної транзакції детально проілюстровано на рисунку 3.15.

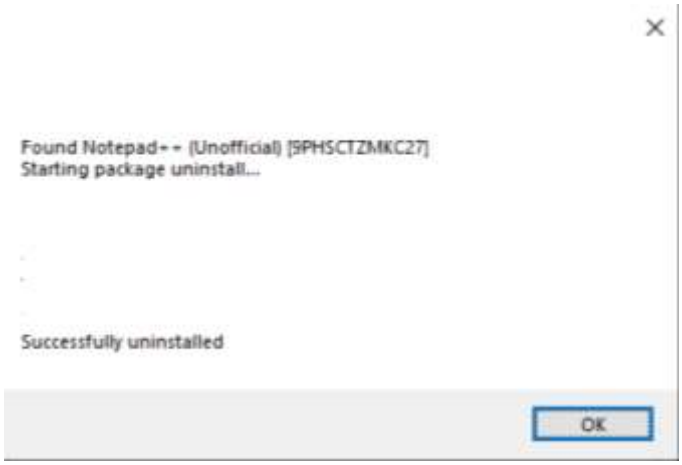


Рисунок 3.15 – Системне повідомлення про деінсталяцію програмного забезпечення

З метою розширення навігаційних можливостей графічного інтерфейсу реалізовано функціонал цільового пошуку програмного забезпечення за допомогою спеціалізованого текстового рядка. Для ініціалізації пошукового запиту користувачеві необхідно ввести номенклатурну назву цільового програмного продукту та активувати відповідний графічний елемент управління «Search».

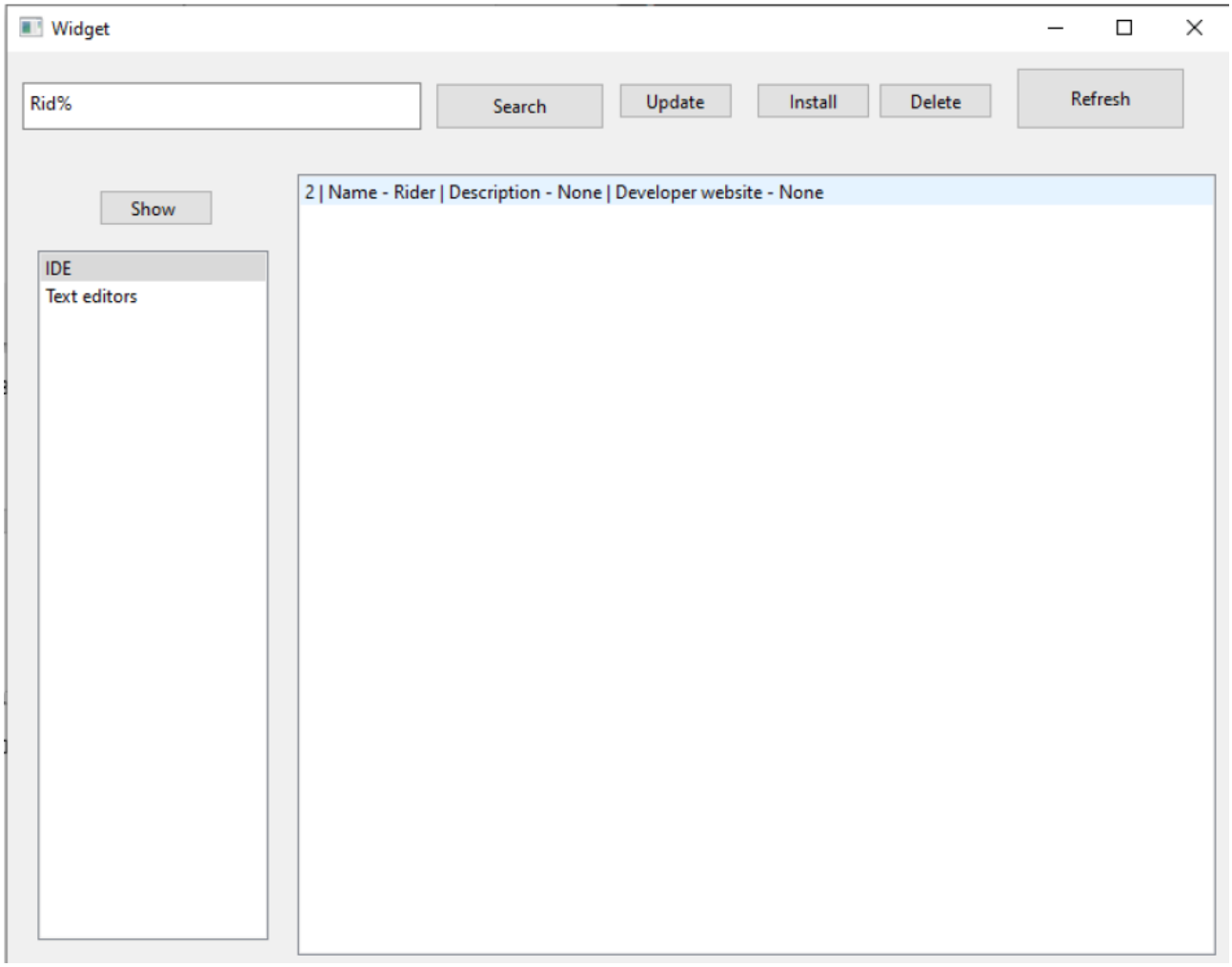


Рисунок 3.16 – Результати виконання пошукового запиту щодо програмного забезпечення

З метою ініціалізації процедури актуалізації версії програмного забезпечення користувачеві необхідно здійснити вибір цільового об'єкта із сформованого реєстру та активувати графічний інструмент управління «Update». Для емпіричної верифікації працездатності запропонованого алгоритму оновлення було застосовано програмний продукт «MS PowerToys»,

поточна інстальована версія якого на локальному апаратному пристрої становила 0.59.0. Візуальне відображення стану системи під час ініціалізації зазначеної транзакції детально проілюстровано на рисунку 3.17.

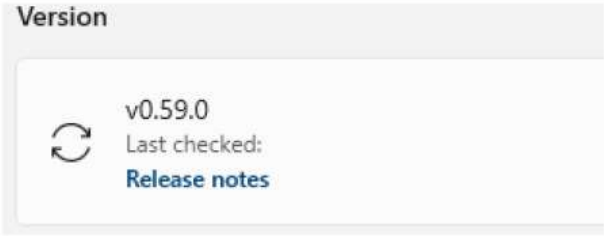


Рисунок 3.17 – Версія програмного забезпечення до ініціалізації процесу оновлення

Під час взаємодії з графічним інтерфейсом користувача здійснюється перехід до класифікаційної категорії «System Utility», де ідентифікується цільове програмне забезпечення «MS PowerToys» для подальшої ініціалізації процедури актуалізації. По завершенню процесів мережевого завантаження та локального розгортання оновлених компонентів система автоматизованого генерування інсталяційних пакетів автоматично формує інформаційне сповіщення щодо статусу виконання системної транзакції. Візуальне подання зазначеного повідомлення наведено на рисунку 3.18.



Рисунок 3.18 – Системне сповіщення про успішне оновлення програмного забезпечення

З метою верифікації результатів проведеної операції здійснюється безпосередній запуск цільового програмного забезпечення із подальшим контролем його поточної версії. Цей етап перевірки повністю підтверджує коректність та безпомилковість функціонування розробленого алгоритму актуалізації, що наочно продемонстровано на рисунку 3.19.

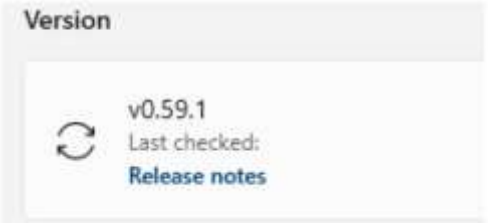


Рисунок 3.19 – Версія програмного забезпечення після завершення процедури актуалізації

За результатами проведеного комплексу тестування графічного інтерфейсу користувача підтверджено безпомилковість виконання всіх закладених функціональних можливостей та повну експлуатаційну працездатність розробленого програмного забезпечення.

З метою верифікації працездатності серверної інфраструктури застосовано спеціалізований інструментарій Swagger UI. Зазначений інтерфейс забезпечує візуалізацію повної специфікації прикладного програмного інтерфейсу та надає механізми для безпосереднього тестування всіх наявних мережевих маршрутизаторів вебсервера. На початковому етапі перевірки здійснюється ініціалізація стартової сторінки Swagger UI з подальшим системним аудитом наявності та коректності конфігурації всіх спроєктованих кінцевих точок інтеграції. Візуальне подання графічного інтерфейсу інструментарію тестування детально проілюстровано на рисунку 3.20.

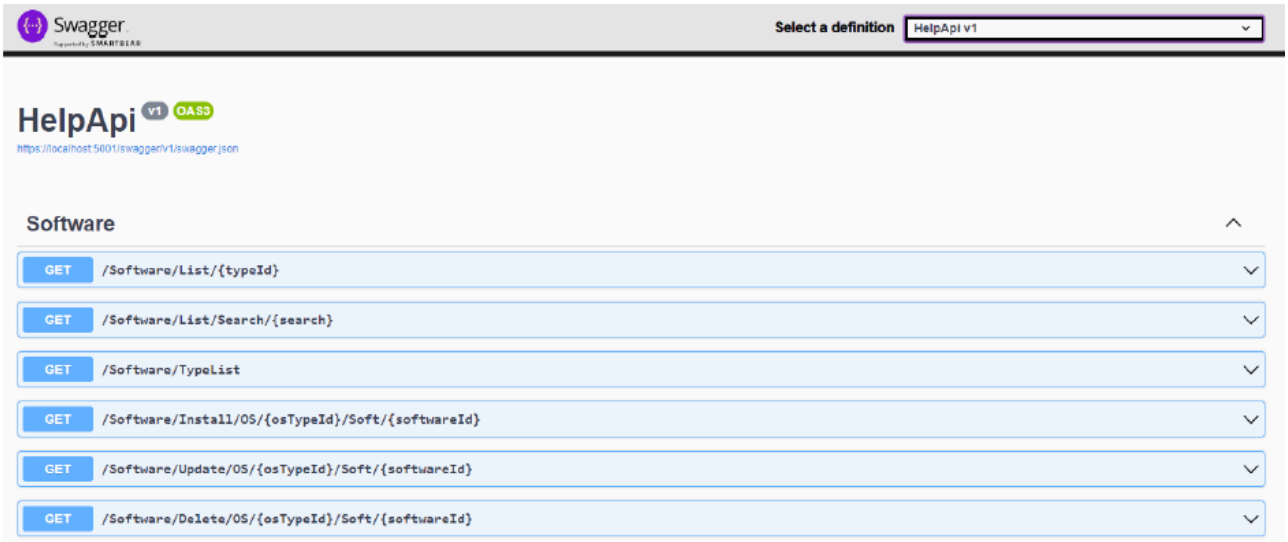


Рисунок 3.21 – Головна сторінка веб-інтерфейсу Swagger UI

Зважаючи на успішне підтвердження наявності всіх передбачених архітектурою маршрутизаторів, подальшим етапом тестування є ініціалізація запиту до віддаленого вебсервера для перевірки коректності вилучення та повернення реєстру доступних категорій програмного забезпечення. Фактичний результат обробки ініційованого мережевого запиту детально проілюстровано на рисунку 3.22.



Рисунок 3.22 – Відображення реєстру категорій, отриманого від вебсервера

З метою верифікації достовірності отриманих результатів здійснюється ініціалізація ідентичного прямого запиту безпосередньо до реляційної бази даних із подальшим аналізом сформованої вибірки. Фактичний результат виконання безпосереднього транзакційного звернення до інформаційного сховища детально проілюстровано на рисунку 3.23.

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

Рисунок 3.23 – Контрольна вибірка реєстру категорій

Результати сформованої контрольної вибірки демонструють абсолютну відповідність масиву інформації, отриманому внаслідок обробки попереднього мережевого запиту віддаленим вебсервером.

З метою подальшої системної верифікації ініційовано перевірку коректності вилучення вебсервером переліку цільового програмного забезпечення, що належить до класифікаційної категорії «IDE». Фактичні результати виконання зазначеного тестового запиту детально проілюстровано на рисунку 3.24.

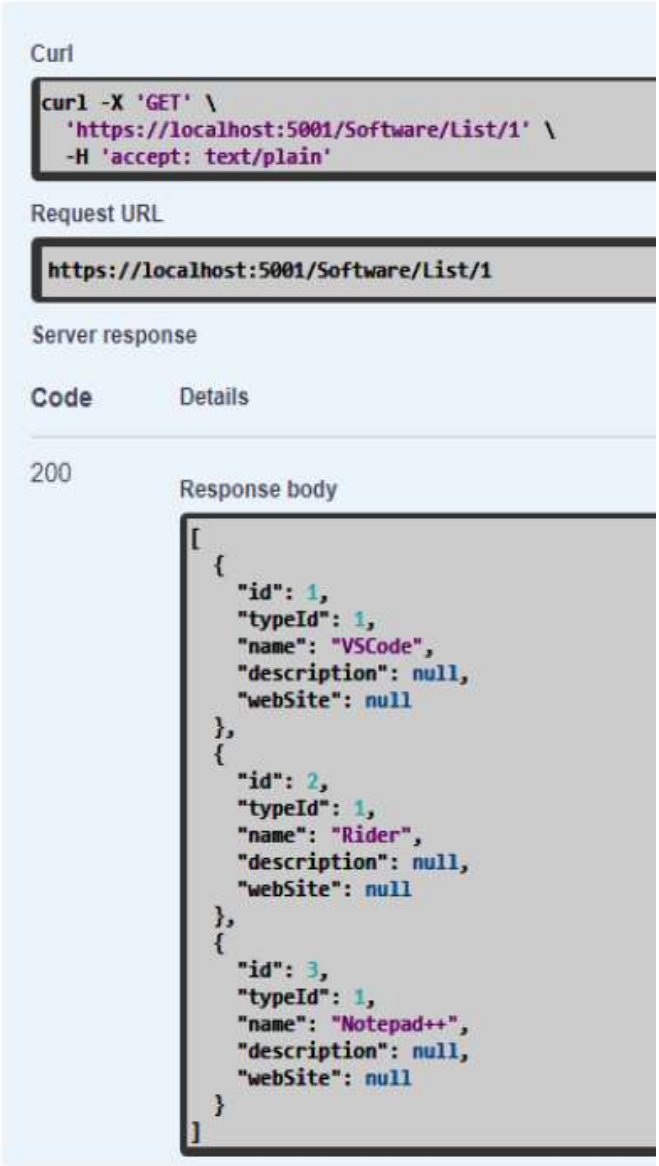


Рисунок 3.24 – Відображення списку програмного забезпечення, отриманого від веб-сервера

З метою верифікації достовірності згенерованих даних було виконано аналогічний прямий запит безпосередньо до реляційної бази даних. Фактичні результати цієї транзакції для подальшого порівняльного аналізу детально проілюстровано на рисунку 3.25.

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

Рисунок 3.25 – Контрольна вибірка реєстру програмного забезпечення

Дані контрольної вибірки, сформовані шляхом безпосереднього транзакційного звернення до інформаційного сховища, демонструють абсолютну тотожність із масивом інформації, отриманим унаслідок обробки мережевого запиту віддаленим вебсервером.

Подальший етап системної верифікації передбачає перевірку коректності алгоритмів формування та повернення цільового керуючого сценарію для ідентифікованого програмного забезпечення. З цією метою ініційовано тестовий запит на отримання скрипта інсталяції для програмного продукту «Notepad++» у середовищі функціонування операційної системи Windows. Фактичні результати виконання зазначеної процедури детально проілюстровано на рисунку 3.26.

Curl

```
curl -X 'GET' \
  'https://localhost:5001/Software/Install/OS/1/Soft/3' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:5001/Software/Install/OS/1/Soft/3
```

Server response

Code	Details
200	Response body

```
winget install --accept-package-agreements 9PHSCTZMKC27
```

Рисунок 3.26 – Сценарій інсталяції програмного забезпечення Notepad++

З метою верифікації синтаксичної коректності згенерованої команди розгортання першочергово виникає необхідність формування контрольної вибірки для базового сегмента керуючої інструкції. Фактичний результат виконання зазначеного етапу перевірки детально проілюстровано на рисунку 3.27.



Рисунок 3.26 – Сценарій інсталяції програмного забезпечення Notepad++

З метою верифікації синтаксичної коректності згенерованої команди розгортання першочергово виникає необхідність формування контрольної вибірки для базового сегмента керуючої інструкції. Фактичний результат виконання зазначеного етапу перевірки детально проілюстровано на рисунку 3.27.

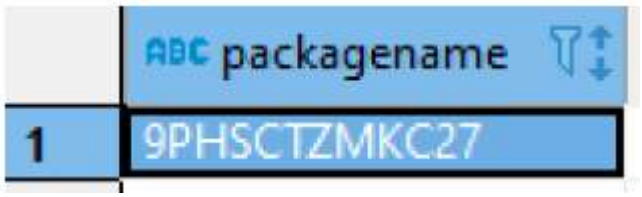


Рисунок 3.28 – Контрольна вибірка з номенклатурною назвою програмного забезпечення

Отримані результати контрольної вибірки демонструють повну тотожність із масивом даних, згенерованим віддаленим вебсервером. Зазначений факт об'єктивно підтверджує функціональну працездатність та алгоритмічну коректність розробленого методу формування інсталяційного сценарію.

У межах подальшого етапу системної апробації ініційовано процедуру отримання керуючого скрипта деінсталяції цільового програмного забезпечення «Atom». З метою верифікації здійснено генерацію відповідної системної інструкції, адаптованої для середовища функціонування операційної системи Windows. Фактичний результат виконання зазначеного тестового запиту детально проілюстровано на рисунку 3.29.

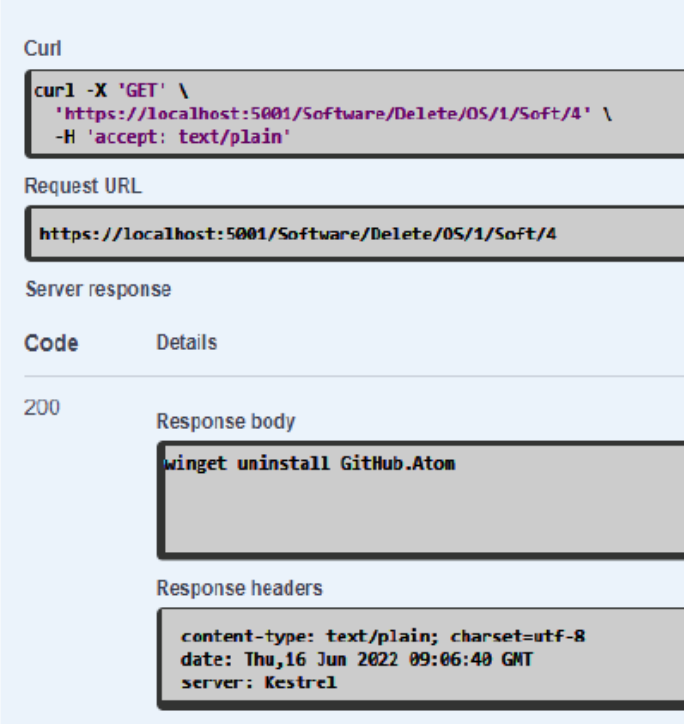


Рисунок 3.30 – Сценарій деінсталяції програмного забезпечення «Atom»

З метою верифікації коректності згенерованого результату ініційовано виконання двох транзакційних запитів до бази даних. Зазначена процедура спрямована на перевірку базового сегмента команди, який регламентує виклик системної інструкції, та її другої частини, що містить унікальну номенклатурну назву програмного забезпечення, адаптовану для відповідного середовища операційної системи. Фактичні результати виконання описаних контрольних запитів детально проілюстровано на рисунку 3.31.



Рисунок 3.31 – Контрольний результат виконання команди деінсталяції

Наступним етапом верифікації є перевірка унікальної номенклатурної назви програмного забезпечення, адаптованої для зазначеного середовища операційної системи. Відповідний процес та його фактичні результати детально проілюстровано на рисунку 3.32.

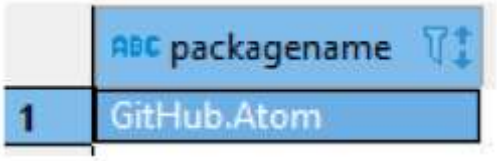


Рисунок 3.32 – Контрольна вибірка номенклатурної назви програмного забезпечення

Аналіз результатів верифікації свідчить, що масив інформації, згенерований віддаленим вебсервером для ініціалізації процесу деінсталяції програмного забезпечення «Atom» у середовищі операційної системи Windows, є абсолютно коректним. Зазначений факт слугує об'єктивним доказом експлуатаційної працездатності та алгоритмічної точності розробленого методу формування керуючого сценарію видалення.

Подальшим етапом системної апробації є верифікація керуючої інструкції, призначеної для актуалізації версії програмного забезпечення. З метою емпіричного підтвердження функціональності алгоритму вилучення сценаріїв оновлення ініційовано перевірку відповідного масиву даних, адаптованого для архітектури операційної системи ArchLinux та цільового програмного продукту «Visual Studio Code». Фактичні результати виконання зазначеного тестового запиту детально проілюстровано на рисунку 3.33.



Рисунок 3.33 – Результат генерації сценарію актуалізації програмного забезпечення

З метою верифікації експлуатаційної працездатності методу формування керуючого скрипта оновлення ініційовано виконання двох транзакційних запитів до бази даних. Зазначена процедура спрямована на перевірку базового сегмента команди, який регламентує виклик системної інструкції, та її другої частини, що містить унікальну номенклатурну назву програмного забезпечення, адаптовану для відповідного середовища операційної системи. Фактичні результати виконання описаних контрольних запитів детально проілюстровано на рисунку 3.34.



Рисунок 3.34 – Контрольний результат виконання команди оновлення

Подальшим етапом верифікації є ідентифікація та перевірка коректності номенклатурної назви цільового програмного забезпечення, що безпосередньо адаптована для зазначеного середовища операційної системи. Відповідну процедуру та її фактичні результати детально проілюстровано на рисунку 3.35.

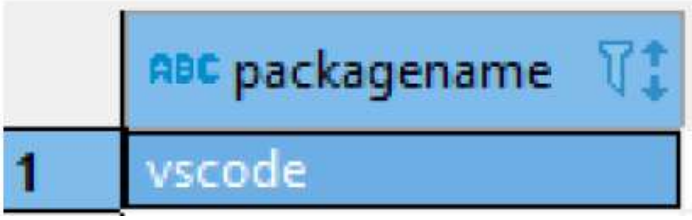


Рисунок 3.36 – Контрольна вибірка номенклатурної назви програмного забезпечення

У межах даного етапу дослідження здійснено аналітичний огляд сучасних методів і методологій тестування програмного забезпечення, за результатами якого обґрунтовано доцільність застосування обраних підходів для системної верифікації веб-сервера та графічного інтерфейсу користувача.

У ході експериментальної апробації реалізовано комплексне тестування серверної інфраструктури щодо коректності вилучення структурованої інформації з бази даних та взаємодії з клієнтським застосунком. Емпірично підтверджено безперебійну працездатність усіх програмних методів веб-

сервера, а також експлуатаційну надійність та стабільність графічного інтерфейсу. Окрему увагу приділено системній перевірці валідності інформаційних потоків, що гарантує точну відповідність згенерованих серверних відповідей на ініційовані дії та транзакції в процесі роботи розробленого програмного забезпечення для автоматизованого генерування інсталяційних пакетів.

ВИСНОВКИ

У межах кваліфікаційної роботи бакалавра спроєктовано та реалізовано систему автоматизованого генерування інсталяційних пакетів на основі сценарних алгоритмів. Зазначене рішення спрямоване на підвищення рівня експлуатаційної стабільності, забезпечення безвідмовності функціонування, а також загальну оптимізацію продуктивності роботи програмного забезпечення та операційного середовища.

Здійснено ґрунтовне дослідження предметної області з подальшою декомпозицією завдань проєктування. На основі порівняльного аналізу існуючих систем управління пакетами сформовано базовий комплекс критеріальних вимог, яким має відповідати розроблене програмне забезпечення.

Виконано аналітичний огляд інструментальних засобів та методологій розробки. За результатами оцінювання обґрунтовано доцільність застосування високорівневої мови програмування Python для імплементації графічного інтерфейсу користувача, що гарантує кросплатформність та сумісність програмного забезпечення із широким спектром сучасних операційних систем. Окрім цього, наведено результати дослідження структури інформаційних потоків, що виникають у процесі обміну даними під час клієнт-серверної взаємодії.

Практична частина роботи охоплює проєктування візуальної оболонки користувача, розробку реляційної бази даних для структурованого зберігання відомостей про програмне забезпечення, а також створення серверної інфраструктури для забезпечення безперебійного доступу до інформаційного сховища.

Здійснено систематизацію методів та інструментів тестування програмного забезпечення з подальшим вибором оптимальних стратегій системної верифікації. За результатами проведеної експериментальної апробації емпірично підтверджено експлуатаційну надійність графічного інтерфейсу програмного забезпечення та абсолютну валідність інформації, що постачається

віддаленим веб-сервером. Як логічний підсумок, розроблено вичерпну експлуатаційну документацію, що регламентує порядок взаємодії користувача зі створеним продуктом.

СПИСОК ЛІТЕРАТУРИ

1. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley Professional, 2021. 448 p.
2. Beazley D., Jones B. K. Python Cookbook. 3rd ed. O'Reilly Media, 2013. 706 p.
3. Chacon S., Straub B. Pro Git. 2nd ed. Apress, 2014. 456 p.
4. Crispin L., Gregory J. Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley Professional, 2009. 576 p.
5. Docker Documentation. URL: https://docs.docker.com/
6. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Pearson, 2015. 1280 p.
7. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003. 560 p.
8. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 560 p.
9. Freeman E., Robson E. Head First Design Patterns. 2nd ed. O'Reilly Media, 2020. 672 p.
10. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. O'Reilly Media, 2018. 316 p.
11. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010. 512 p.
12. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press, 2016. 480 p.
13. McKinney W. Python for Data Analysis. 3rd ed. O'Reilly Media, 2022. 576 p.
14. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. John Wiley & Sons, 2011. 256 p.
15. Newman S. Building Microservices. 2nd ed. O'Reilly Media, 2021. 614 p.
16. OpenAPI Specification. Swagger. URL: https://swagger.io/specification/

17. PEP 8 – Style Guide for Python Code. Python.org. URL: https://peps.python.org/pep-0008/
18. Percival H. Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript. 2nd ed. O'Reilly Media, 2017. 614 p.
19. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill Education, 2019. 608 p.
20. PyQt5 Reference Guide. Riverbank Computing. URL: https://www.riverbankcomputing.com/static/Docs/PyQt5/
21. Python 3 Documentation. Python Software Foundation. URL: https://docs.python.org/3/
22. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming. 2nd ed. O'Reilly Media, 2022. 1014 p.
23. Richardson L., Ruby S. RESTful Web Services. O'Reilly Media, 2007. 448 p.
24. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. McGraw-Hill Education, 2019. 1376 p.
25. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 816 p.
26. SQLite Official Documentation. URL: https://www.sqlite.org/docs.html
27. Swagger UI Tutorial. URL: https://swagger.io/tools/swagger-ui/
28. Windows Installer Documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/windows/win32/msi/windows-installer-portal
29. Баран С. В. Розробка програмного забезпечення з використанням патернів проєктування : навч. посіб. Кривий Ріг : ДУЕТ, 2023. 203 с.
30. Бородкіна І. Л., Бородкін Г. О. Інженерія програмного забезпечення : навч. посіб. Київ : Центр учбової літератури, 2020. 204 с.
31. Грицюк Ю. І., Рак Т. Є. Програмування мовою Python : навч. посіб. Львів : Вид-во Львівської політехніки, 2020. 268 с.
32. Коротєєва Т. О. Алгоритми та структури даних : навч. посіб. Львів : Видавництво Львівської політехніки, 2014. 280 с.
33. Макконнелл С. Досконалий код. Майстер-клас / пер. з англ. Харків : Фабула, 2019. 912 с.

34. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення / пер. з англ. О. Логвиненко. Харків : Фабула, 2020. 368 с.
35. Мартін Р. Чистий код. Створення, аналіз і рефакторинг / пер. з англ. О. Логвиненко. Харків : Фабула, 2019. 464 с.
36. Мельник А. О. Архітектура комп'ютерних систем : підручник. Луцьк : Вежа-Друк, 2018. 340 с.
37. Пасічник В. В., Резніченко В. А. Організація баз даних та знань : підручник. Львів : Магнолія 2006, 2018. 384 с.
38. Трофименко О. Г., Прокоп Ю. В., Шинкарьов О. Г., Логінова Н. І. Python. Створення програм : навч. посіб. Одеса : ОНАЗ ім. О. С. Попова, 2019. 248 с.
39. Федорченко В. М. Основи інженерії програмного забезпечення : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 195 с.
40. Шаховська Н. Б., Голощук Р. О. Алгоритми та структури даних : навч. посіб. Львів : Магнолія 2006, 2024. 216 с.

ДОДАТКИ