НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра інформаційних систем та технологій

Тема “Flexible automation tool”

Курсова робота
З дисципліни «Технології розроблення програмного забезпечення»

Керівник

доц. Амонс О.А.

«Допущений до захисту»

_____

(Особистий підпис керівника)

« » _____2025р.

Захищений з оцінкою

_____

(оцінка)

Члени комісії:

_____ _____

(особистий підпис) (розшифровка підпису)

_____ _____

(особистий підпис) (розшифровка підпису)

Виконавець

ст. Тунік О. І.

залікова книжка № ___ – _____

гр. ІА-34

_____

(особистий підпис виконавця)

« »_____2025р.

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
(назва навчального закладу)

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Дисципліна «Технології розроблення програмного забезпечення»

Курс 3 Група ІА-34 Семестр 1

ЗАВДАННЯ

на курсову роботу студента

Туніка Олександра Ігоровича

(прізвище, ім’я, по батькові)

1.Тема роботи Flexible Automation Tool

2. Строк здачі студентом закінченої роботи 8.12.2025

3. Вихідні дані до роботи: Інструмент автоматизації повинен забезпечувати найпростіші автоматичні дії для зручності користувача: завантаження нових фільмів / книг / файлів при випуску (наприклад, щоп'ятниці з'являються нові серії улюблених серіалів); встановити статуси в комунікаторах (skype – away при нульовій активності на тривалий час) і т.д. Автоматизація забезпечується шляхом введення правил (на зразок IFTTT.com сервісу), запису макросів (натискання клавіш, дії миші), планувальника завдань (о 5 ранку – початок роздачі торрент-файлів).

4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці)

_____

_____

_____

_____

Додатки:

_____

5.Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

_____

_____

_____

_____

6. Дата видачі завдання 17.09.2025

КАЛЕНДАРНИЙ ПЛАН

№, п/п	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Підписи або примітки
1.	Підбір та визначення літератури	30.09.2025	
2.	Проектування та написання розділу 1	31.10.2025	
3.	Розробка та написання розділу 2	20.11.2025	
4.	Подання курсової роботи на перевірку	25.11.2025	
5.	Захист курсової роботи	08.12.2025	

Студент _____ ____Олександр ТУНІК_____
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник_____ __Олександр АМОНС__
(підпис) (Ім’я ПРІЗВИЩЕ)

«____»_____ 2025 р.

ЗМІСТ

ВСТУП

Сучасне суспільство характеризується неймовірно високим рівнем цифровізації, значна частина повсякденних завдань виконується за допомогою комп'ютерних систем. Зі зростанням обсягу інформації та кількості рутинних дій виникає потреба у створенні програмних засобів, здатних автоматично виконувати повторювані операції без безпосередньої участі користувача. Автоматизація дає змогу підвищити ефективність роботи, зменшити кількість помилок та оптимізувати використання часу. Саме тому створення систем, які забезпечують гнучке керування автоматизованими діями, є актуальним завданням у галузі розроблення програмного забезпечення.

У сучасній практиці програмування існує велика кількість рішень, які дозволяють користувачам автоматизувати робочі процеси. Більшість таких систем базується на принципі зв'язку подій та дій – коли виконання певної умови ініціює заздалегідь визначену реакцію. Такі рішення дають змогу створювати ланцюги дій, що виконуються без участі користувача. Незважаючи на це, багато існуючих продуктів або є надмірно складними у налаштуванні, або не дозволяють користувачу достатньо гнучко керувати процесами, що обмежує їх практичне застосування.

У зв'язку з цим метою даної курсової роботи є розроблення системи Flexible Automation Tool – програмного засобу, що забезпечує створення та виконання користувацьких сценаріїв автоматизації дій за принципом «якщо – то». Передбачається, що користувач зможе самостійно формувати правила, що поєднують події (тригери) та реакції (дії), створюючи власні сценарії автоматизації. Такий підхід дозволить реалізувати широкий спектр можливостей – від запуску програм за розкладом до виконання дій при настанні певних умов, наприклад, отриманні повідомлення чи зміні стану системи.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

Розвиток систем автоматизації користувацьких дій тісно пов’язаний із пошуком способів спростити взаємодію людини з комп’ютером. Сьогодні на ринку представлено велику кількість інструментів, які дають змогу користувачам створювати сценарії автоматизації, поєднуючи різні сервіси, додатки та події. Серед найвідоміших рішень у цій сфері можна виділити такі системи, як IFTTT, Zapier, Microsoft Power Automate та Automate.io (придбана Notion в 2021 році), кожна з яких має власні переваги, але, звичайно, і обмеження, що знижують їх універсальність або зручність для певних типів користувачів.

Сервіс IFTTT (If This Then That) є одним із найвідоміших прикладів системи, що реалізує принцип умовно-логічної автоматизації “якщо – то”. Він дозволяє користувачу створювати або використовувати створені іншими користувачами прості сценарії («аплети»), де подія в одному сервісі викликає дію в іншому. Наприклад, автоматичне збереження фотографій із соціальних мереж у хмарне сховище або надсилання сповіщень при зміні погоди. Попри свою простоту, IFTTT має обмежену гнучкість – система підтримує лише заздалегідь визначений набір сервісів і не дає змоги створювати складні умови чи комбіновані послідовності дій.

Інший популярний сервіс, Zapier, орієнтований переважно на бізнес-користувачів і дозволяє будувати складніші автоматизовані робочі процеси («zap-и»), які можуть включати кілька кроків і умовних гілок. Zapier відзначається великою кількістю інтеграцій із хмарними платформами, проте більшість його функцій доступні лише у платній версії. Крім того, система вимагає постійного підключення до Інтернету та не підтримує автономну роботу локальних процесів користувача, що є істотним недоліком для персональних сценаріїв автоматизації.

Microsoft Power Automate є частиною екосистеми Microsoft і інтегрована з продуктами, такими як Office 365, Teams та SharePoint. Вона надає розширені можливості побудови автоматизованих потоків дій, включно з підтримкою бізнес-

процесів, доступом до API та можливістю використовувати штучний інтелект для аналізу даних. Водночас Power Automate також орієнтований насамперед на корпоративне використання і потребує високого рівня технічної підготовки для створення складних сценаріїв, що робить його менш придатним для пересічного користувача.

Ще одним прикладом є Automate.io, який дозволяв створювати багатокрокові сценарії та інтегруватися з популярними онлайн-платформами. Але починаючи з 2021, коли Notion придбала компанію, весь функціонал системи обмежений автоматіями у Notion.

Проведений огляд показує, що існуючі рішення поділяються на дві основні групи: хмарні платформи для інтеграції онлайн-сервісів і локальні утиліти для автоматизації дій користувача. Обидва підходи мають суттєві обмеження – перші не працюють без Інтернету, а другі не забезпечують зручної взаємодії із зовнішніми сервісами. У цьому контексті розроблення системи Flexible Automation Tool спрямоване на поєднання переваг обох типів інструментів: забезпечення локальної автоматизації із можливістю розширення через модулі, а також гнучке визначення користувацьких правил без необхідності програмування.

Таким чином, створення такої системи є доцільним, оскільки вона покликана усунути обмеження наявних рішень і надати користувачеві більш універсальний інструмент для персональної автоматизації дій у межах локального та мережевого середовищ.

1.2. Загальний опис проєкту

Проєкт Flexible Automation Tool (FAT) створюємо як універсальний інструмент, що дозволить користувачу створювати правила автоматизації у зручному форматі без програмування. Основна ідея полягає у використанні моделі “якщо подія – тоді дія” (основна ідея IFTTT), яка дасть змогу задавати тригери та відповідні реакції системи. Завдяки цьому користувач зможе створювати власні сценарії поведінки пристрою, комбінуючи різні умови та дії.

Система покликана спростити роботу користувача з цифровим середовищем, забезпечуючи не лише виконання завдань за розкладом, а й реакцію на події в режимі реального часу. Це зробить її гнучким рішенням для автоматизації широкого спектра задач – від персонального використання до професійного застосування у сфері моніторингу, адміністрування або тестування програмного забезпечення.

Проєкт передбачає розширювану архітектуру, що дозволить додавати нові типи подій і дій без зміни основної логіки роботи. Завдяки цьому користувач зможе налаштовувати систему під власні потреби або підключати сторонні модулі. Кожне правило автоматизації складатиметься з набору елементів: умови, яка ініціює виконання, дії, яку слід виконати, та додаткових параметрів, що визначатимуть контекст виконання. Наприклад, користувач може задати правило «о 5:00 ранку розпочати роздачу файлів», або «при отриманні нового повідомлення в месенджері змінити статус на ‘Busy’».

Проєктуватимемо систему з урахуванням принципів об’єктно-орієнтованого програмування (ООП), що дозволить ефективно моделювати різні види подій і дій через узагальнені класи та інтерфейси. Такий підхід забезпечить модульність, повторне використання коду та можливість інтеграції з іншими інструментами автоматизації. Крім того, використаємо відомі шаблонів проєктування – зокрема Strategy, Command, Abstract Factory, Facade та Interpreter, які допоможуть структурувати логіку виконання правил, відокремити процес прийняття рішень від виконання конкретних дій і спростити подальше розширення системи.

Головна мета проєкту – створити зручне, стабільне та гнучке середовище автоматизації, яке не потребує спеціальних технічних навичок, але дає широкі можливості для налаштування поведінки комп’ютера. В результаті користувач отримує інструмент, здатний самостійно виконувати регулярні завдання, реагувати на зміни в системі чи мережі та оптимізувати щоденну взаємодію з цифровими сервісами.

1.3. Вимоги до застосунків системи

У цьому розділі подано вимоги до системи Flexible Automation Tool, що визначають її функціональні можливості, поведінку та ключові якісні характеристики.

Вимоги сформульовані на основі аналізу очікуваної поведінки користувача та логіки роботи автоматизованих процесів. Для візуалізації взаємодії користувачів із системою створено діаграму варіантів використання (Use Case diagram), зображену на рисунку 1.

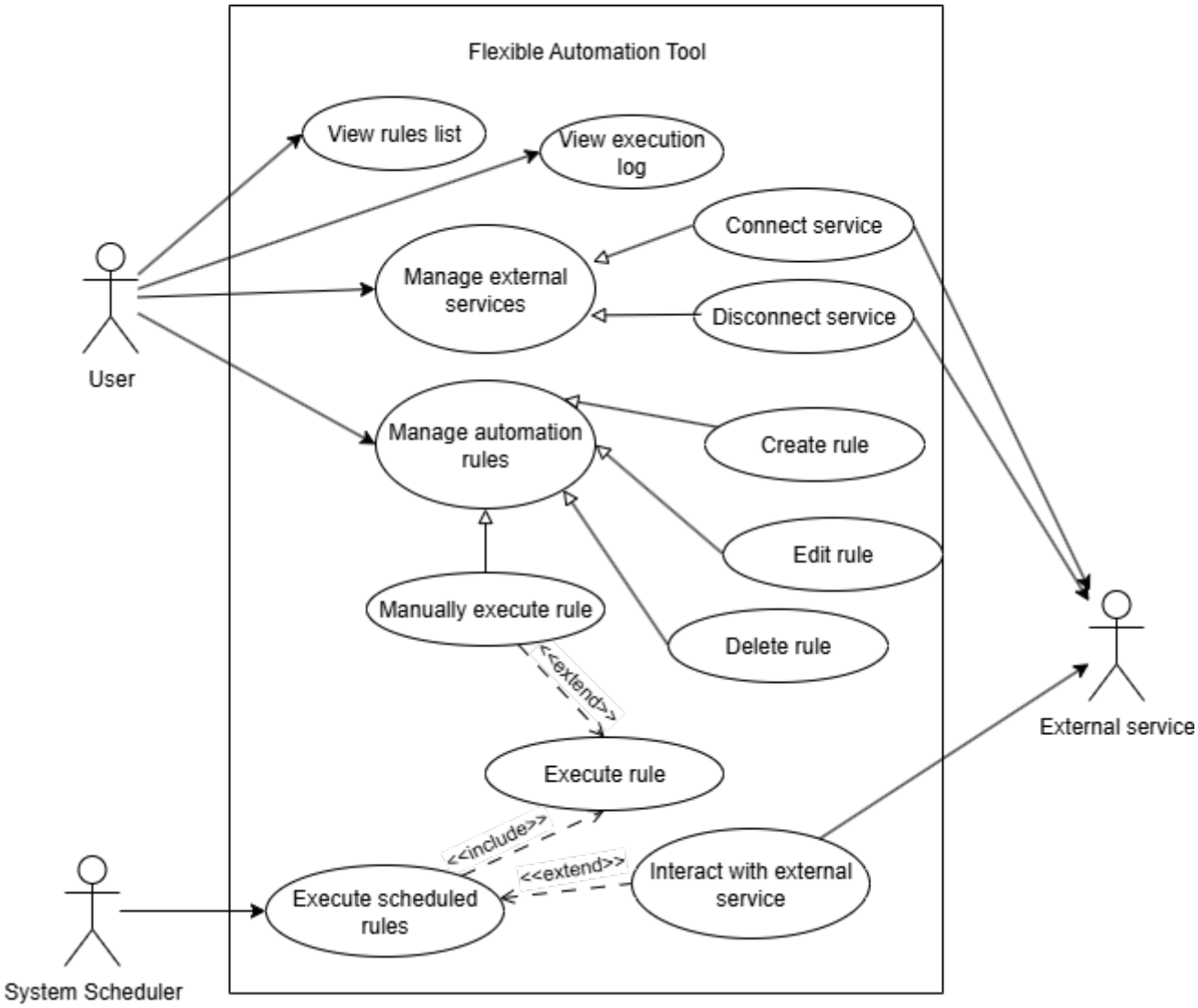

Рисунок 1. Діаграма варіантів використання системи Flexible Automation Tool

1.3.1. Функціональні вимоги до системи

Як показано на рисунку 1, у системі беруть участь два основні актори:

User (Користувач) – основний учасник, який створює та налаштовує правила автоматизації, а також керує зовнішніми сервісами;
і Scheduler (Планувальник) – системний процес, що відповідає за автоматичне виконання правил відповідно до встановленого розкладу.

Основним функціональним модулем системи є керування правилами автоматизації (Manage automation rules). Через нього користувач має змогу створювати нові правила (Create rule), змінювати існуючі (Edit rule) або видаляти непотрібні (Delete rule).

Окрім цього, передбачено можливість перегляду переліку створених правил (View rules list) та історії їх виконання (View execution log). Ці сценарії охоплюють основний цикл роботи користувача з системою, забезпечуючи інструменти для контролю та аналізу процесів автоматизації.

Окрема важлива функціональна вимога це керування зовнішніми сервісами (Manage external services). Система має вміти підключати (Connect service) або відключати (Disconnect service) зовнішні програми чи служби, з якими взаємодіють автоматизаційні сценарії. Такі сервіси зможуть виступати як джерела подій (тригери) або як об’єкти дій (наприклад, надсилання повідомлення, обробка файлів, керування системними процесами).

Ключовим функціональним елементом є процес виконання правил (Execute rule).

Він може відбуватися у двох формах: ручне виконання (Manually execute rule), коли користувач сам ініціює запуск певного правила, та автоматичне виконання запланованих правил (Execute scheduled rules) — коли це робить планувальник без прямої участі користувача. Автоматичне оброблення подій за розкладом передбачає виконання їх дій (include), тоді як взаємодія із зовнішніми сервісами під час оброблення є додатковою дією (extend).

Таким чином, функціональні вимоги системи включають повний набір дій, необхідних для створення, налаштування та виконання правил автоматизації, а також підтримку інтеграції із зовнішніми компонентами.

1.3.2. Нефункціональні вимоги до системи

Окрім основної функціональності, система має відповідати певним нефункціональним вимогам, які визначають її якість, надійність та зручність використання.

Найважливішою вимогою є зручність та інтуїтивність інтерфейсу. Система призначена для широкого кола користувачів, тому взаємодія з нею має бути зрозумілою без технічної підготовки. Інтерфейс повинен мати логічну структуру, стандартні елементи керування та просту навігацію між розділами.

Другою важливою вимогою поставили стабільність і надійність роботи. Система повинна забезпечувати коректне виконання завдань, навіть якщо під час роботи виникають неочікувані помилки або відсутні зовнішні ресурси. У таких випадках користувач повинен отримувати повідомлення про помилку без втрати цілісності даних.

Важливими аспектами також є масштабованість і розширюваність. Архітектура повинна дозволяти додавання нових типів дій, тригерів або сервісів без необхідності змінювати базовий код системи. Це забезпечуватися через використання принципів інтерфейсного програмування та шаблонів Strategy, Abstract Factory і Facade.

Система має підтримувати переносимість, тобто можливість роботи на різних версіях операційних систем Windows, без потреби в складному налаштуванні.

Ще однією вимогою є продуктивність. Виконання правил і реагування на події мають відбуватися з мінімальною затримкою, навіть якщо одночасно активовано кілька сценаріїв. Для задоволення цієї вимоги знадобиться оптимізація роботи з подіями та ефективне використання багатопоточності.

Окремо виділимо вимогу збереження даних і цілісності бази даних. Усі зміни правил мають фіксуватися без ризику втрати, навіть при аварійному завершенні роботи. Також система повинна забезпечувати коректне оновлення даних при внесенні змін користувачем.

Крім цього, система має бути безпечна у використанні – тобто не виконувати жодних дій, що можуть зашкодити даним користувача або його операційній системі.

Для цього реалізуємо базову перевірку безпеки виконуваних сценаріїв і обмеження на потенційно небезпечні команди.

1.4. Сценарії використання системи

У цьому розділі, продемонструємо реальні способи взаємодії користувача та системи у типових ситуаціях. Сценарії дозволять зрозуміти логіку роботи програмного комплексу не лише на рівні окремих компонентів, а й з позиції загальної поведінки системи під час виконання основних функцій. Вони відображають послідовність дій, обмін повідомленнями між елементами архітектури та реакції системи на користувацькі або внутрішні події, тим самим створюючи цілісну картину роботи Flexible Automation Tool у динаміці.

Перший сценарій, «Створення нового правила автоматизації», демонструє типовий процес взаємодії користувача із системою під час формування нової автоматичної дії. На діаграмі (рис. 2) відображено, як ініціатором процесу виступає користувач, який через графічний інтерфейс (GUI) відкриває форму створення нового правила. Система запитує у користувача базову інформацію: назву, короткий опис і тип тригера, що визначає умову запуску (наприклад, час, активність користувача або зовнішню подію). Після введення цієї інформації інтерфейс переходить до другого етапу – визначення дій (actions). Користувач обирає або формує дію, яку система повинна виконати при спрацюванні тригера, наприклад запуск програми, зміна статусу або завантаження файлу.

Коли всі параметри задані, AutomationEngine отримує з GUI зібрані дані та виконує перевірку їхньої коректності. На цьому етапі система аналізує синтаксис, унікальність назви, коректність параметрів часу чи умов. Якщо правило проходить валідацію, двигун автоматизації передає його до RuleRepository – сховища правил, що відповідає за їх збереження, оновлення та подальше використання. Після підтвердження успішного запису користувачу виводиться повідомлення про створення нового правила. У разі виявлення помилки (наприклад, відсутній тригер або некоректна дія) GUI відображає попередження із зазначенням проблемного

параметра. Такий процес забезпечує прозорість і контрольованість взаємодії, водночас зберігаючи гнучкість при розширенні типів тригерів і дій у майбутньому.

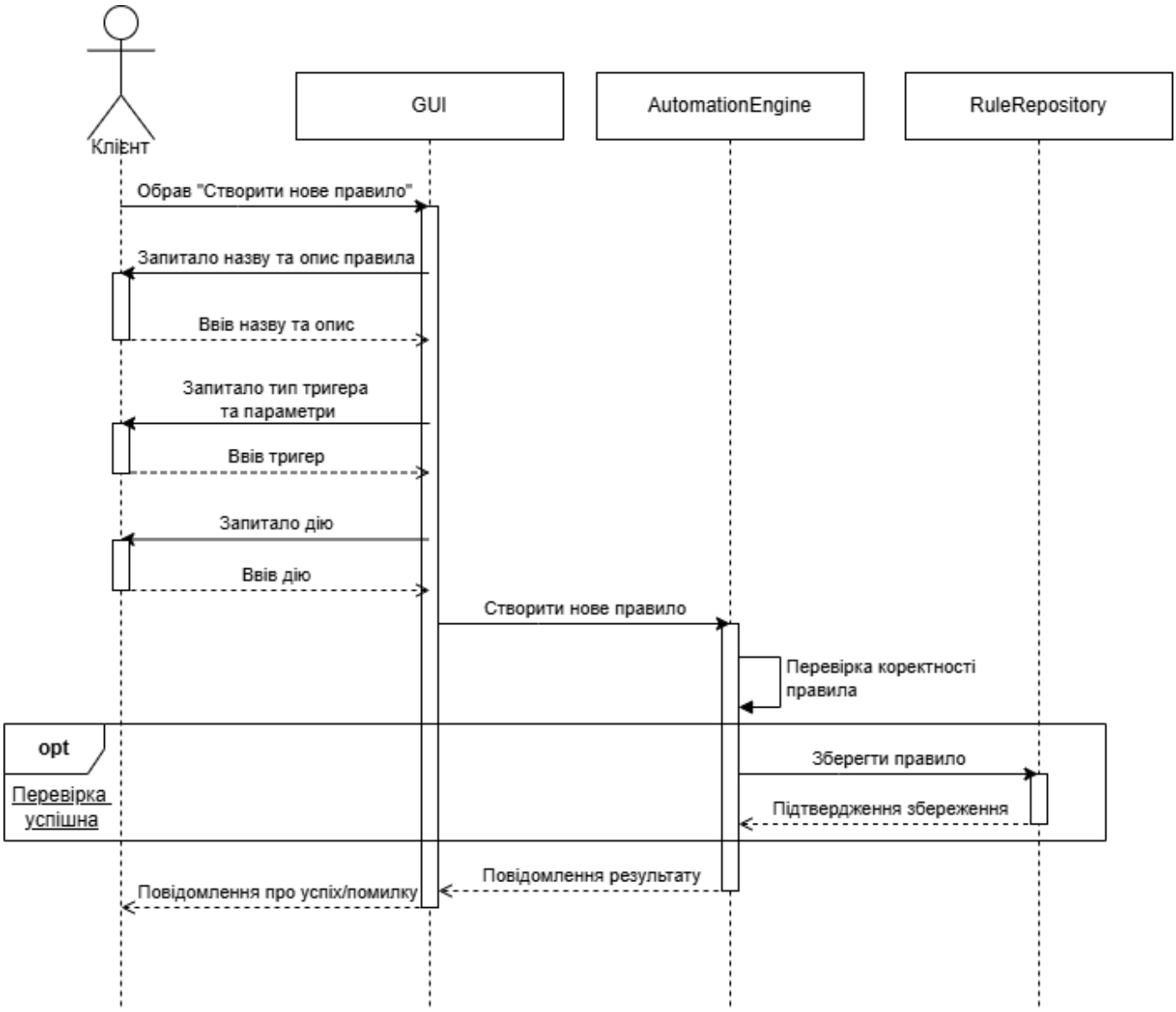

Рисунок 2. Діаграма послідовності процесу створення нового правила автоматизації

Другий сценарій – «Перегляд історії виконання правил» – демонструє, як користувач може отримати аналітичну інформацію про роботу системи. На діаграмі (рис. 3) зображено взаємодію між користувачем (Client), графічним інтерфейсом (GUI), ядром системи (AutomationEngine) та компонентом Logger, який веде облік усіх виконань правил. Користувач ініціює запит через інтерфейс, обираючи опцію «Переглянути історію». GUI формує запит до AutomationEngine, який виступає

посередником і звертається до Logger із запитом на отримання списку записів про виконання. Logger у відповідь шукає дані у своєму сховищі та повертає список об’єктів типу LogEntry, які містять такі поля, як дата та час запуску, назва правила, статус виконання, результат або повідомлення про помилку.

Після цього AutomationEngine отримує результати, структурує їх у зручному форматі й передає назад до GUI. Інтерфейс візуалізує історію у вигляді таблиці чи журналу, що дає користувачеві змогу швидко проаналізувати минулі події.

Ця функціональність є важливою частиною системи, оскільки дозволяє здійснювати зворотній зв’язок між користувачем та автоматизованими процесами — користувач може оцінювати стабільність роботи, виявляти невдалі виконання або налаштовувати правила ефективніше на основі попереднього досвіду. Взаємодія між компонентами при цьому залишається мінімалістичною, що дозволяє легко масштабувати систему або інтегрувати додаткові аналітичні модулі.

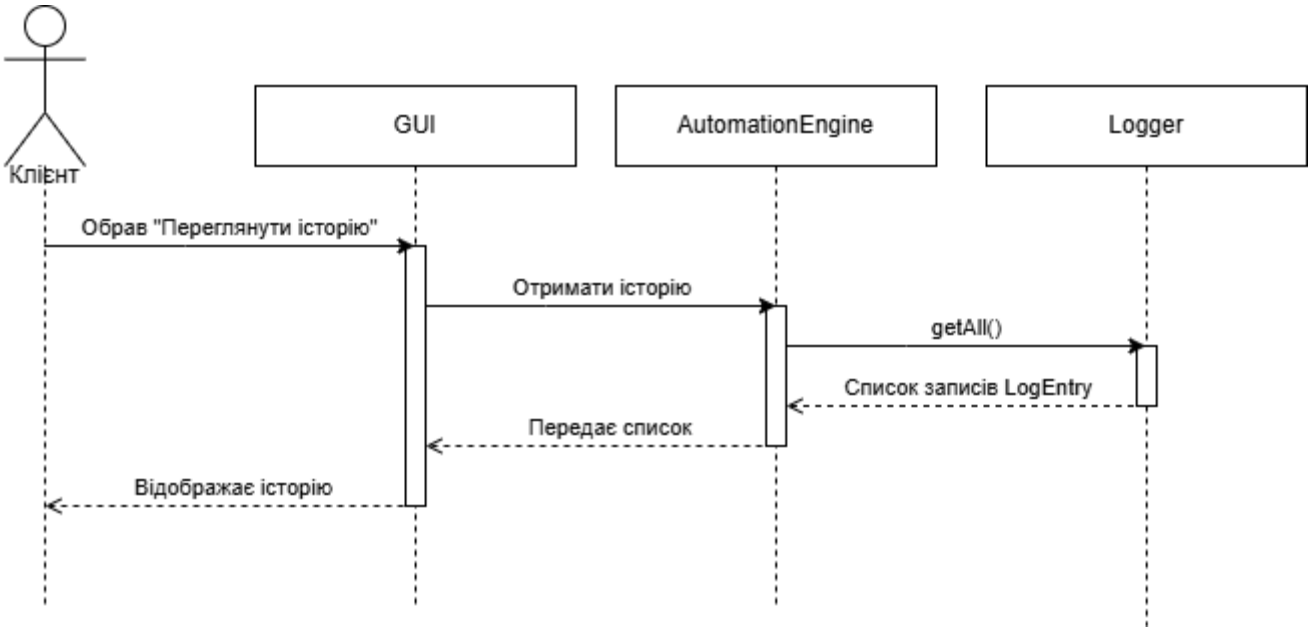

Рисунок 3. Діаграма послідовності процесу перегляду історії виконання правил

Третій описаний сценарій, «Автоматичне виконання правил за розкладом», ілюструє ключовий функціонал системи – повністю автономну поведінку системи, коли втручання користувача не потрібне. На діаграмі (рис. 4) показано, як компонент Scheduler у задані проміжки часу ініціює процес автоматичного опрацювання

активних правил. Scheduler звертається до RuleRepository із запитом на отримання списку правил, що мають активні тригери типу TimeTrigger. RuleRepository повертає колекцію об’єктів Rule, після чого планувальник перевіряє для кожного з них, чи настав час його виконання. Для цього Scheduler здійснює внутрішній виклик (self-call), що дозволяє перевіряти умови часу без участі зовнішніх компонентів. Якщо час збігається із заданим у правилі, Scheduler переходить до фази виконання.

У процесі виконання відповідне правило (Rule) активується, виконує задану дію (наприклад, змінює статус у додатку, запускає скрипт або надсилає запит до зовнішнього сервісу), після чого повертає планувальнику підтвердження про завершення. Паралельно Rule надсилає об’єкт LogEntry до Logger, який записує результат виконання у журнал історії. Таким чином, система забезпечує повний цикл автоматизації, від планування до фіксації результатів, без ручної взаємодії. Подібна модель дозволяє розширювати функціональність системи, додаючи нові типи тригерів (наприклад, події файлової системи або мережеві сигнали) без зміни базової архітектури.

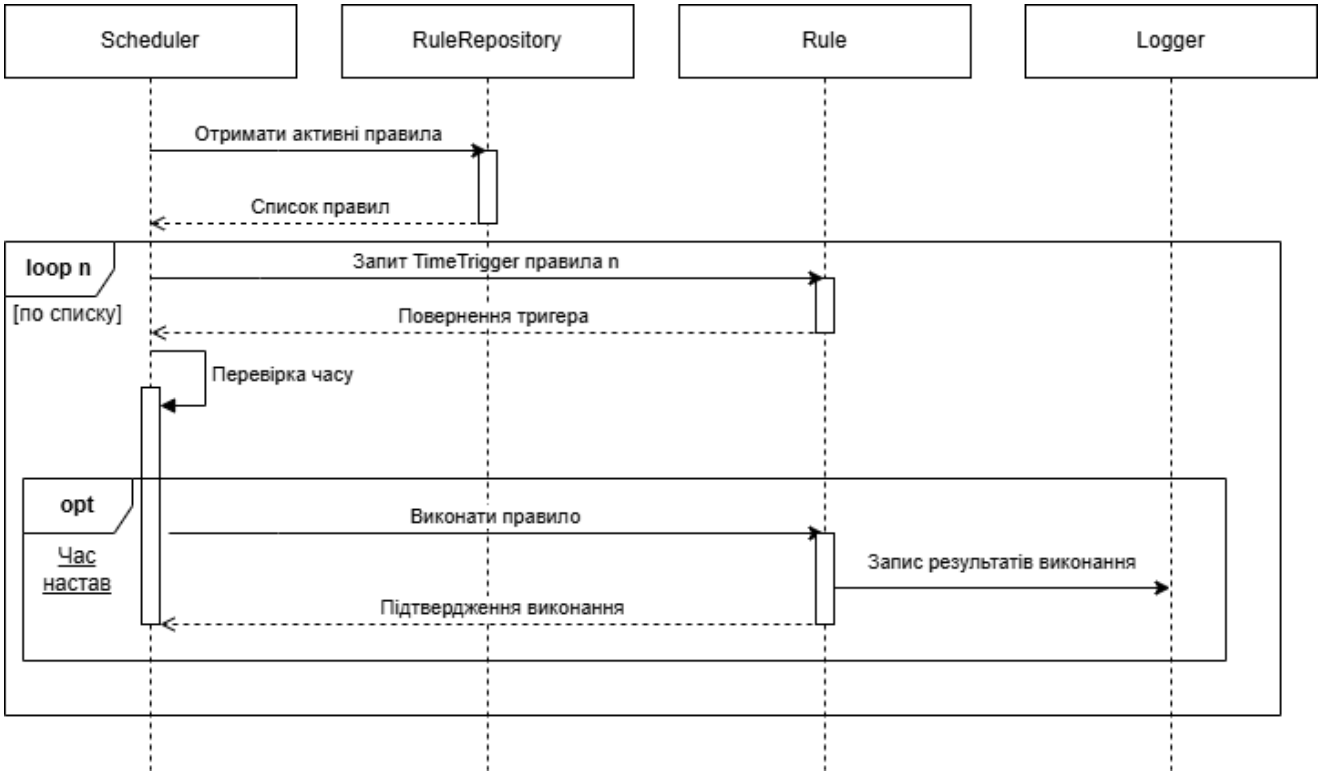

Рисунок 4. Діаграма послідовності процесу автоматичного виконання правил за розкладом

У сукупності ці три сценарії утворюють логічну основу роботи системи Flexible Automation Tool: користувач створює правила, планувальник виконує їх за заданими умовами, а журнал дій забезпечує відстежуваність усіх процесів. Такий підхід відображає принципи інкапсуляції, модульності та повторного використання, що є фундаментом об’єктно-орієнтованого проєктування.

1.5. Концептуальна модель системи

Концептуальна модель системи Flexible Automation Tool відображає основні сутності предметної області та зв’язки між ними, що формують логіку роботи інструмента автоматизації. На діаграмі класів (рис. 5) подано структуру взаємодії компонентів, які забезпечують створення, зберігання, виконання та керування правилами автоматизації. Модель описує як базові класи, так і відношення між ними, зокрема композицію, агрегацію, наслідування та асоціацію.

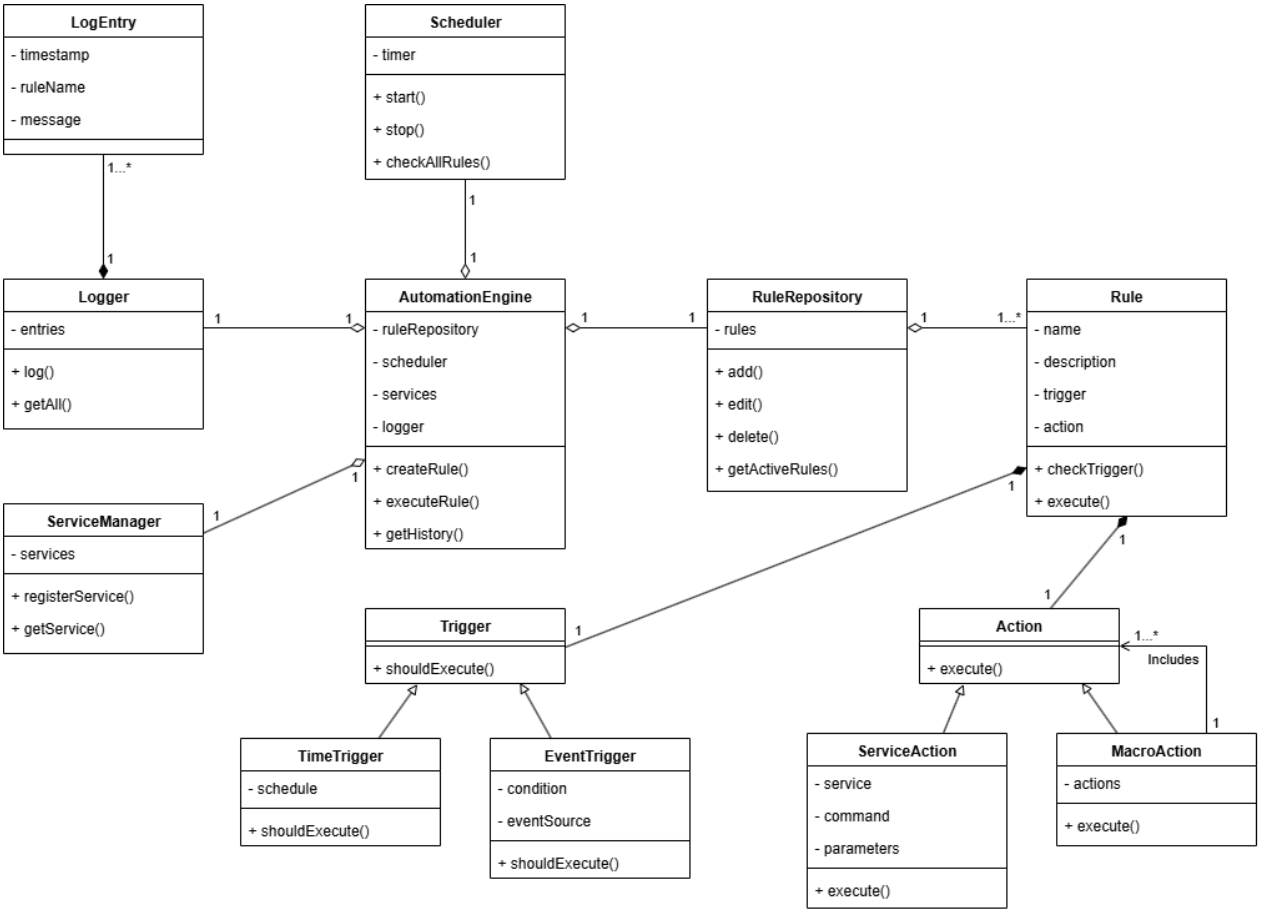

Рисунок 5. Діаграма класів системи

Центральним елементом моделі є клас AutomationEngine, який виконує роль ядра системи. Саме він координує роботу основних компонентів, таких як RuleRepository, Scheduler, ServiceManager та Logger. Взаємозв'язок між ними подано як агрегацію, що означає ці елементи є підконтрольними ядру, однак можуть існувати незалежно від нього. RuleRepository відповідає за управління життєвим циклом правил: створення, збереження, оновлення та видалення. Він композиційно містить множину об'єктів Rule, які становлять логічне ядро процесу автоматизації.

Кожне Rule описує окрему логіку автоматизації та складається з двох ключових частин: Trigger і Action. Зв'язок між ними та класом Rule є композиційним: тригери й дії створюються разом із правилом і не можуть існувати самостійно. Trigger є абстрактним класом, який визначає базові характеристики умов активації правила. Від нього наслідуються конкретні реалізації: TimeTrigger, що реагує на певний момент або інтервал часу, та EventTrigger, який активується у відповідь на подію чи зміну стану системи.

Аналогічно, Action є базовим класом, що описує виконувану дію у межах правила. Від нього наслідуються ServiceAction, яка взаємодіє з зовнішніми сервісами, та MacroAction, що дозволяє групувати кілька дій у складну послідовність. MacroAction пов'язаний асоціацією з множиною Action, утворюючи ієрархію вкладених дій, які виконуються як єдиний сценарій.

Завдяки діаграмі класів можемо продемонструвати гнучкість і масштабованість системи: будь-який компонент може бути розширений новими типами тригерів або дій без порушення загальної архітектури. Ця модель є основою для подальшого етапу проєктування – формування логіки реалізації компонентів та їхнього розгортання в системі.

1.6. Вибір бази даних

Для зберігання даних у системі Flexible Automation Tool обрали реляційну базу даних MySQL, оскільки вона забезпечує стабільність, масштабованість і високу

продуктивність при роботі зі структурованими даними. MySQL є відкритою та перевіреною часом технологією, що має розвинену екосистему інструментів і бібліотек для інтеграції з мовою програмування C# через ORM-рішення, такі як Entity Framework або Dapper. Це дозволяє ефективно реалізувати взаємозв'язки між сутностями, описаними в діаграмі класів, і гарантує зручність у подальшому розширенні системи.

Структура бази даних (рис. 6) побудована відповідно до логічної моделі системи та відображає основні сутності предметної області: Rules, Triggers, Actions, Services та ExecutionHistory. Таблиця Rules зберігає інформацію про всі правила автоматизації, включно з посиланнями на відповідні тригери та дії. Triggers описують умови запуску правил, тоді як Actions визначають конкретні дії, що виконуються після спрацювання тригера. У таблиці ExecutionHistory фіксуються результати виконання кожного правила, це забезпечує можливість моніторингу, аналізу та відлагодження роботи системи.

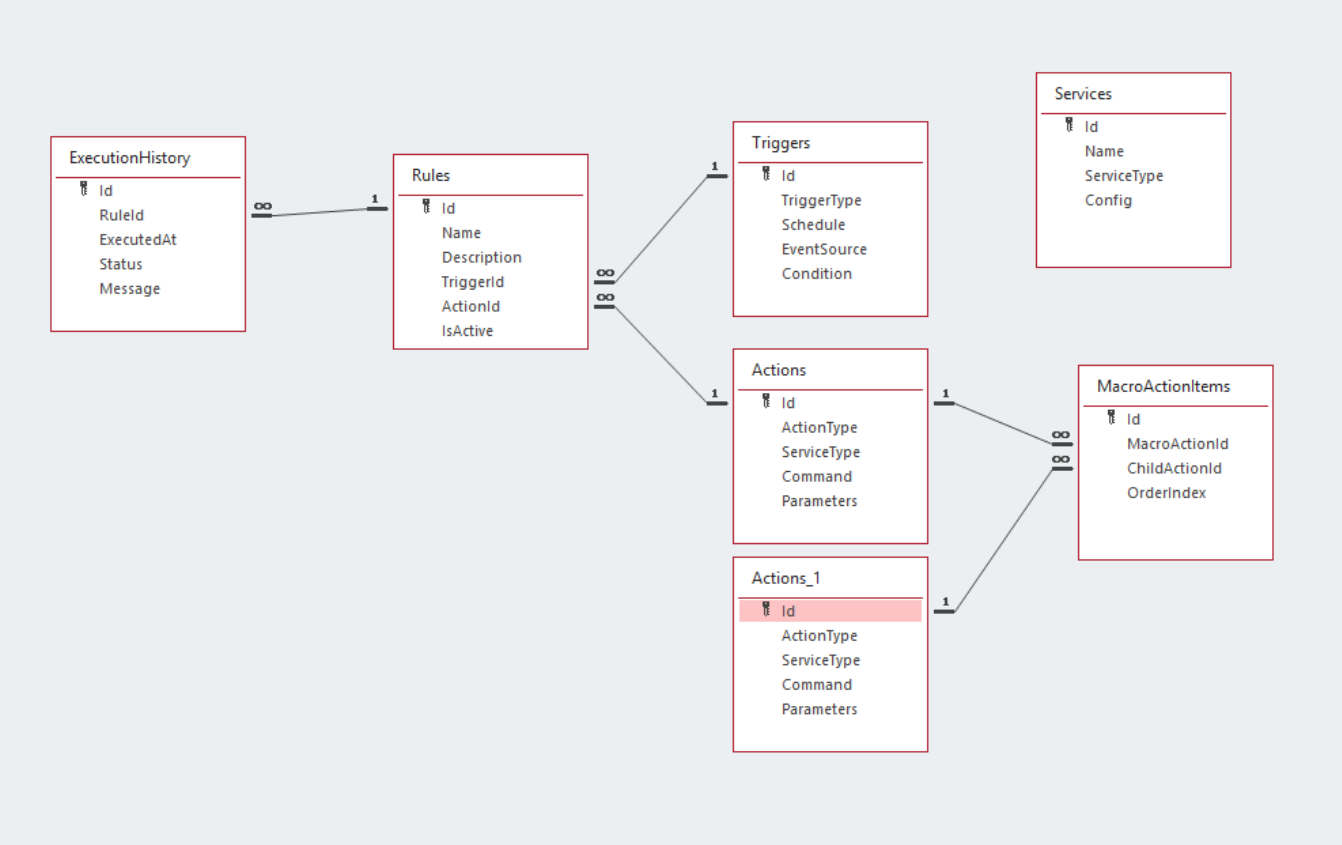

Рисунок 6. Зображення структури бази даних системи

Окремо реалізована таблиця MacroActionItems, яка відображає зв’язки між складеними (макро) діями та їхніми піддіями. Така структура дозволяє зберігати складні сценарії автоматизації без втрати гнучкості та підтримувати ієрархічну побудову дій. Вибір MySQL зумовлений також її сумісністю з різними платформами та легкістю розгортання, що робить її оптимальним рішенням для гібридних систем, орієнтованих на клієнтсько-серверну взаємодію та подальше масштабування.

1.7. Вибір мови програмування та середовища розробки

Для розроблення системи Flexible Automation Tool обрано мову програмування C# у середовищі .NET 8, що забезпечує високу продуктивність, кросплатформеність та підтримку сучасних архітектурних підходів. Розробка буде вестися у середовищі Microsoft Visual Studio, яке надає зручні інструменти для налагодження, тестування та візуального проєктування інтерфейсу.

Користувацький інтерфейс будемо реалізувати за допомогою Windows Forms, що дозволить швидко створювати інтерактивні десктопні застосунки з інтуїтивним графічним середовищем. У перспективі розглядається можливість переходу на Windows Presentation Foundation (WPF) для розширення візуальних можливостей та кращої підтримки шаблонів проєктування, таких як MVVM. Такий вибір технологій забезпечить баланс між швидкістю розробки, стабільністю роботи та гнучкістю у подальшій еволюції системи.

Вибір стеку технологій також зумовлений потребою у високій сумісності між різними компонентами системи. .NET 8 забезпечує ефективну взаємодію між бібліотеками, підтримує роботу з базами даних через ORM-технології, зокрема Entity Framework Core, що спростить роботу з MySQL і мінімізує кількість ручного SQL-коду. Крім того, C# має потужну типізацію та сучасні можливості об’єктно-орієнтованого програмування, що дає змогу ефективно реалізувати закладені в проєкті шаблони – Strategy, Command, Abstract Factory, Facade та Interpreter. Такий

технологічний підхід гарантує розширюваність, масштабованість і підтримуваність коду в довгостроковій перспективі.

1.8. Проєктування розгортання системи

Проєктування розгортання має важливе значення для забезпечення стабільності, масштабованості та доступності системи. На цьому етапі визначається конфігурація обладнання, необхідні операційні системи, програмні платформи та мережеві протоколи, що використовуються для комунікації між компонентами. Для системи Flexible Automation Tool важливою вимогою є можливість одночасної роботи декількох користувачів, тому структура розгортання має передбачати розподіл обчислювальних ресурсів між клієнтськими вузлами та сервером обробки даних.

Діаграма розгортання системи (рис. 7) відображає фізичну структуру компонентів програмного комплексу Flexible Automation Tool та їхню взаємодію під час роботи. На ній показано вузли, програмні середовища та канали комунікації між клієнтською та серверною частинами. Основна мета цієї діаграми – продемонструвати, як логічні компоненти системи реалізуються на фізичних пристроях та які умови необхідні для їхнього ефективного функціонування.

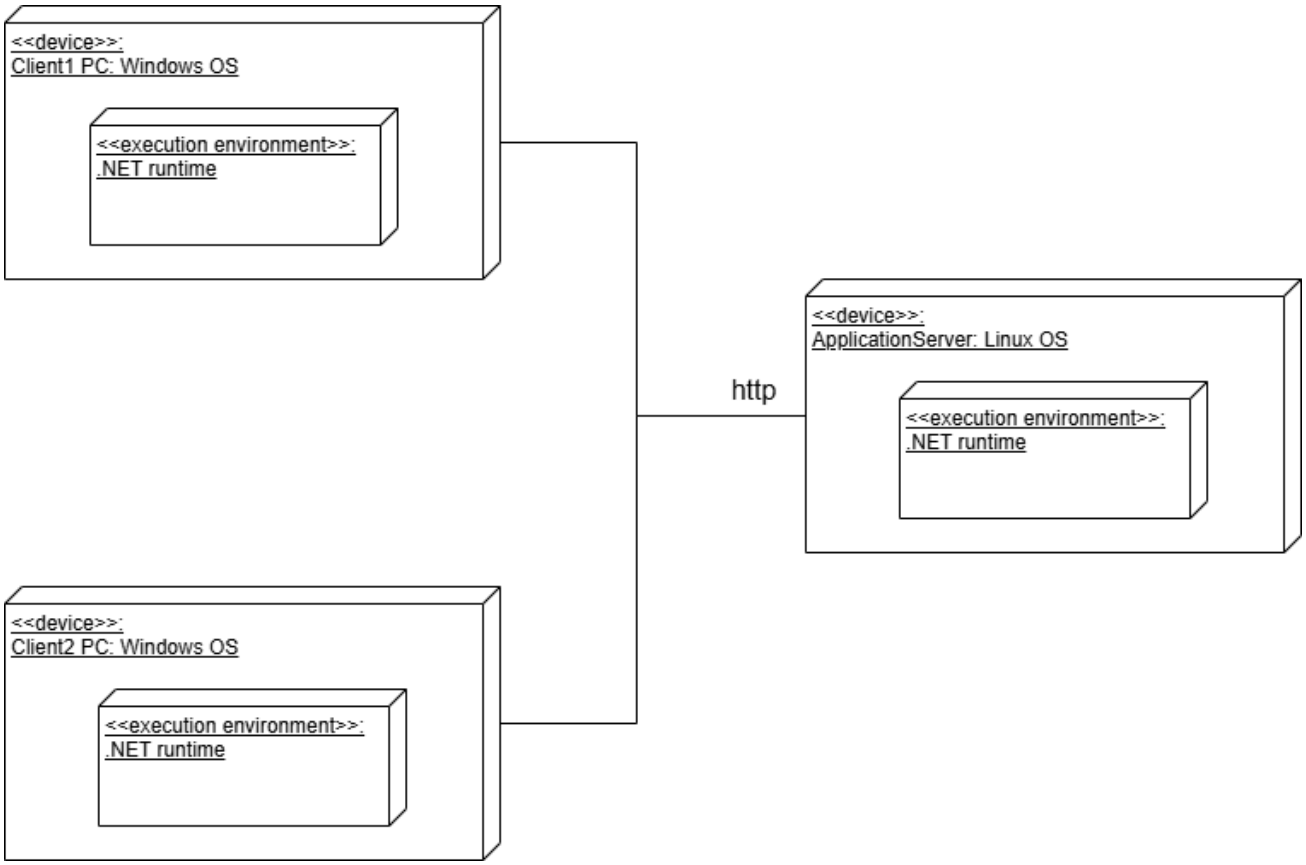

Рисунок 7. Діаграма розгортання системи

Клієнтські вузли представлені як персональні комп'ютери користувачів (Client1 PC, Client2 PC), на яких встановлена операційна система Windows. Саме тут запускається графічний інтерфейс користувача, який забезпечує взаємодію з ядром системи. Клієнтська частина дозволяє користувачам створювати, редагувати та відстежувати правила автоматизації, а також переглядати історію виконання. Всі клієнтські додатки працюють у середовищі .NET Runtime, що гарантує стабільність і сумісність з іншими компонентами системи.

Центральним вузлом є Application Server, який працюватиме під керуванням Linux OS. Цей сервер відповідає за обробку бізнес-логіки, зберігання правил, керування планувальником завдань та взаємодію із зовнішніми сервісами. Сервер також розгортається у середовищі .NET Runtime, що забезпечить єдину технологічну основу для всієї системи. Таким чином, клієнти й сервер мають спільне середовище

виконання, що значно спростить обмін даними, оновлення компонентів і підтримку системи.

Комунікація між клієнтами та сервером відьуватиметься за допомогою стандартного HTTP-протоколу, який забезпечить швидкий та безпечний обмін повідомленнями. Завдяки цьому реалізуємо класичну клієнт-серверну архітектуру, у якій клієнтські програми надсилають запити до сервера, а сервер відповідає, виконуючи заплановані або безпосередньо викликані дії.

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Структура бази даних

2.2. Архітектура системи

2.2.1. Специфікація системи

2.2.2. Вибір та обґрунтування патернів реалізації

2.3. Інструкція користувача

ВИСНОВКИ

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.

[2] I. Sommerville, *Software Engineering*, 10th ed. Harlow, U.K.: Pearson Education, 2016.

[3] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2020.

[4] Zapier Inc., "What is Zapier?", [Онлайн]. Доступно: https://help.zapier.com/hc/en-us/articles/37518970271245-What-is-Zapier. Доступ отримано: Жовтень 2025.

[5] E. Gamma, R. Helm, R. Johnson, та J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley, 1994.

ДОДАТКИ