

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)

(особистий підпис)

Виконавець

ст. Тунік О. І.

залікова книжка № ____ – ____

гр. ІА-34

(особистий підпис виконавця)

« » _____ 2025р.

(розшифровка підпису)

(розшифровка підпису)

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

[illegible]

Студент _____
(підпис)

Олександр ТУНІК
(Ім'я ПРИЗВИЩЕ)

Керівник _____
(підпис)

Олександр АМОНС
(Ім'я ПРИЗВИЩЕ)

« » _____ 2025 p.

ЗМІСТ

ВСТУП	3
1 ПРОЄКТУВАННЯ СИСТЕМИ.....	4
1.1. Огляд існуючих рішень	4
1.2. Загальний опис проєкту.....	5
1.3. Вимоги до застосунків системи.....	6
1.3.1. Функціональні вимоги до системи.....	7
1.3.2. Нефункціональні вимоги до системи.....	9
1.4. Сценарії використання системи	10
1.5. Концептуальна модель системи	14
1.6. Вибір бази даних	16
1.7. Вибір мови програмування та середовища розробки.....	16
1.8. Проєктування розгортання системи.....	17
2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ	20
2.1. Структура бази даних	20
2.2. Архітектура системи.....	21
2.2.1. Специфікація системи	22
2.2.2. Вибір та обґрунтування патернів реалізації.....	23
2.3. Інструкція користувача.....	25
ВИСНОВКИ.....	31
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	33
ДОДАТКИ.....	34

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 («зар-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 автоматизаціями 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.

Проект 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'».

Проектуватимемо 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 проектування – 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 проекту – 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 1.1.

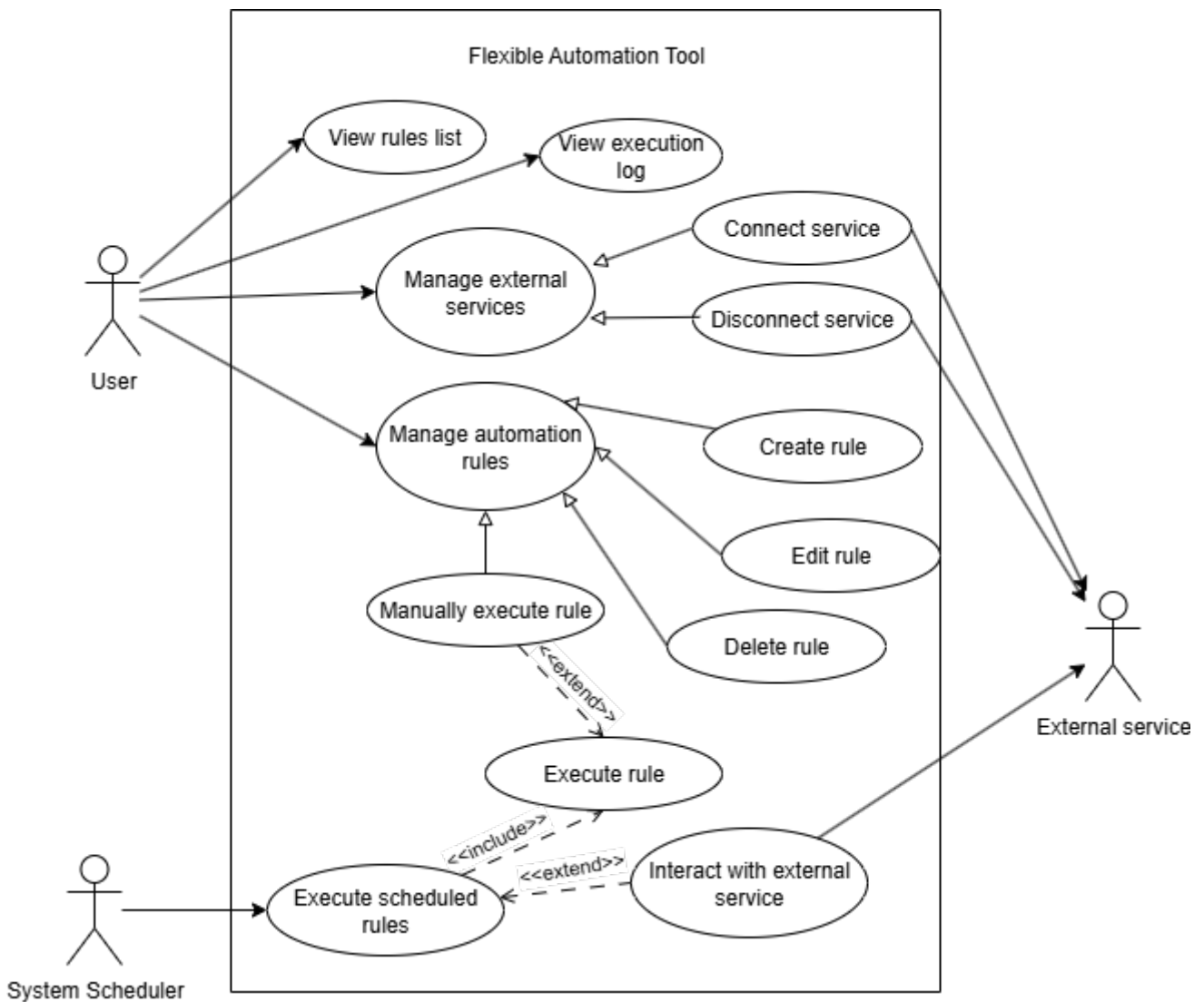


Рисунок 1.1. Діаграма варіантів використання системи Flexible Automation Tool

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;

i 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. 1.2) 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.

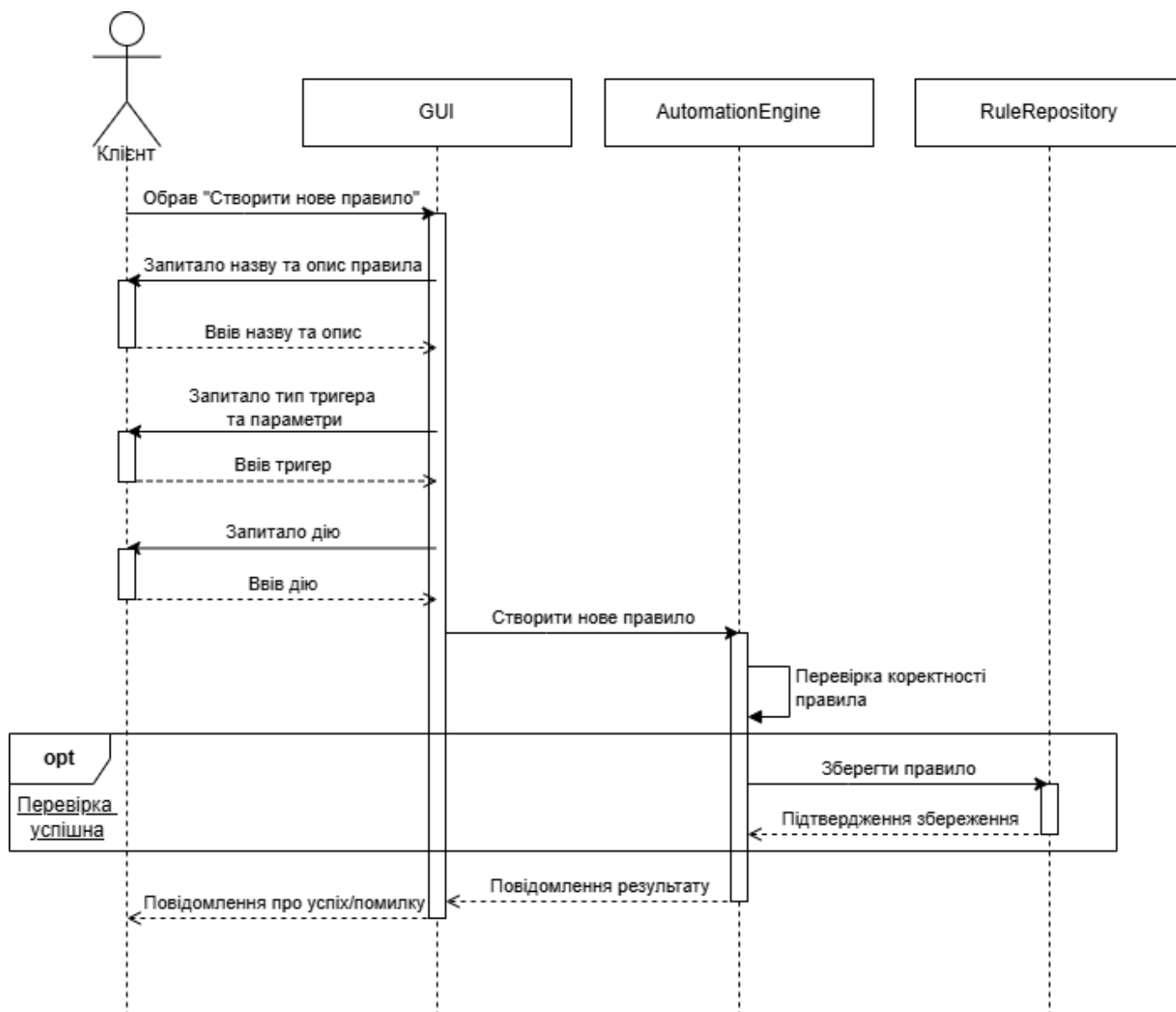


Рисунок 1.2. Діаграма послідовності процесу створення нового правила автоматизації

Другий сценарій – «Перегляд історії виконання правил» – демонструє, як користувач може отримати аналітичну інформацію про роботу системи. На діаграмі (рис. 1.3) 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.

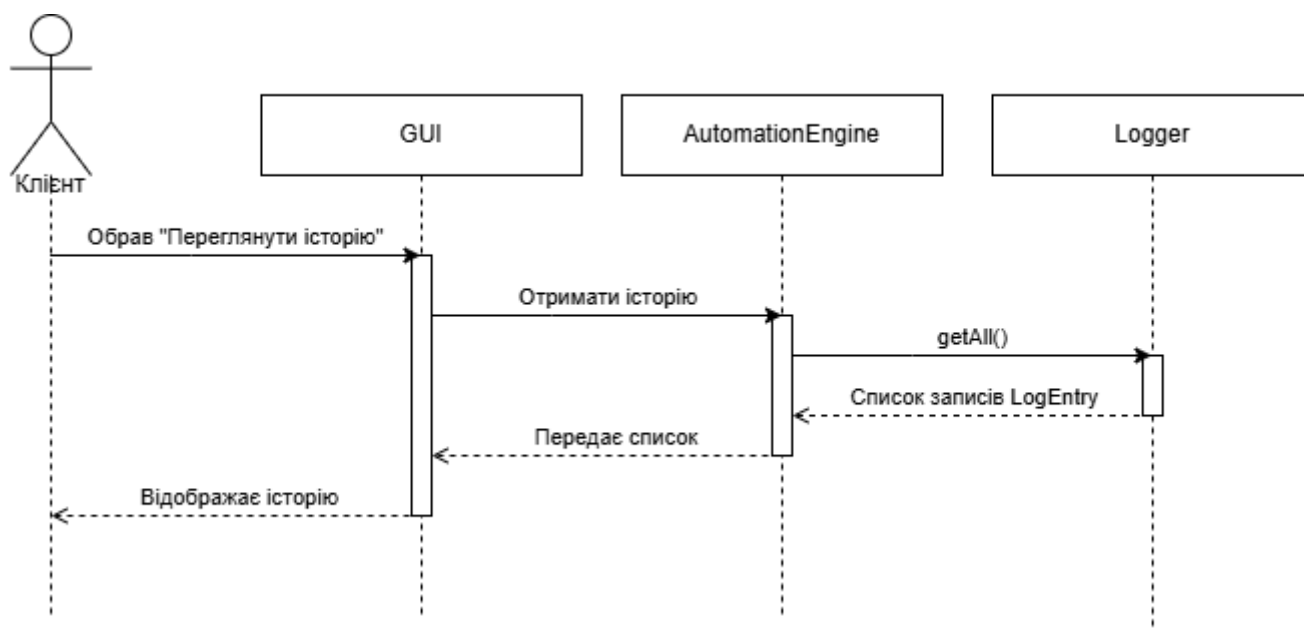


Рисунок 1.3. Діаграма послідовності процесу перегляду історії виконання правил

Третій описаний сценарій, «Автоматичне виконання правил за розкладом», ілюструє ключовий функціонал системи – повністю автономну поведінку системи, коли втручання користувача не потрібне. На діаграмі (рис. 1.4) 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.

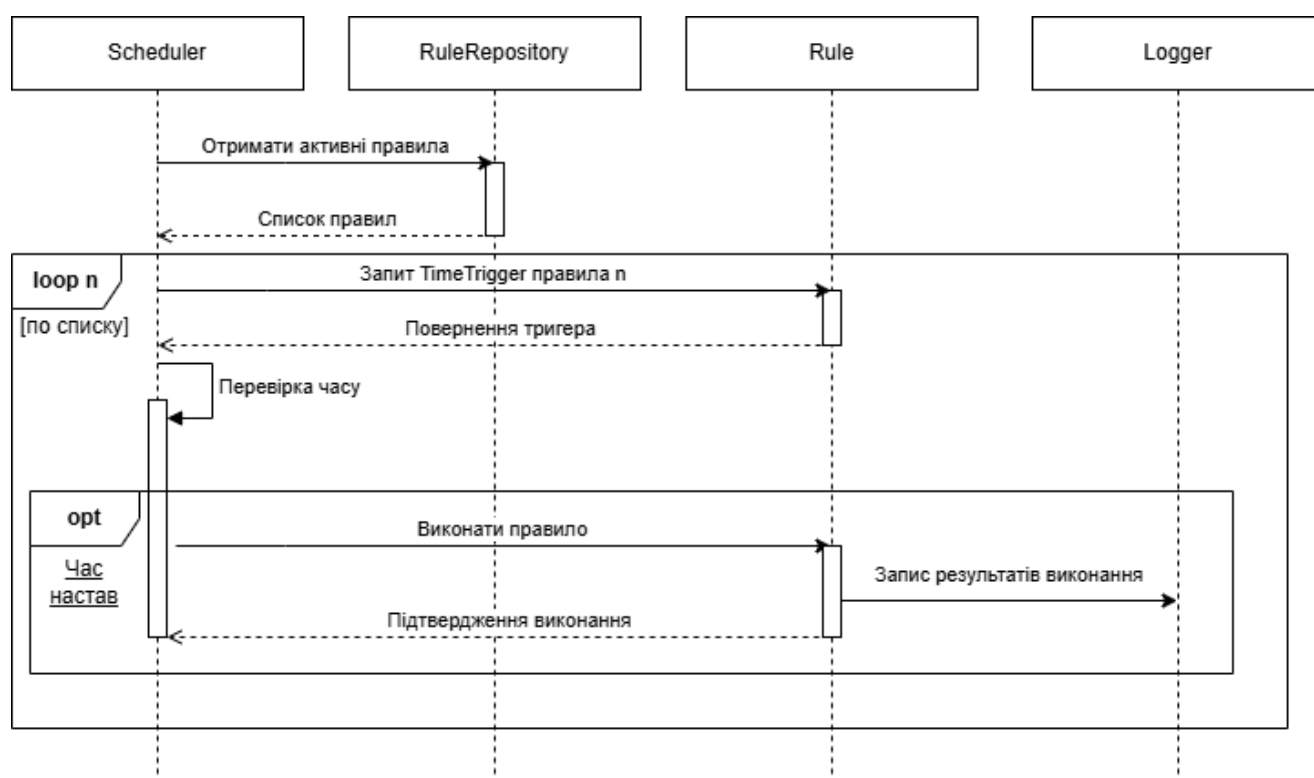


Рисунок 1.4. Діаграма послідовності процесу автоматичного виконання правил за розкладом

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. 1.5) 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.

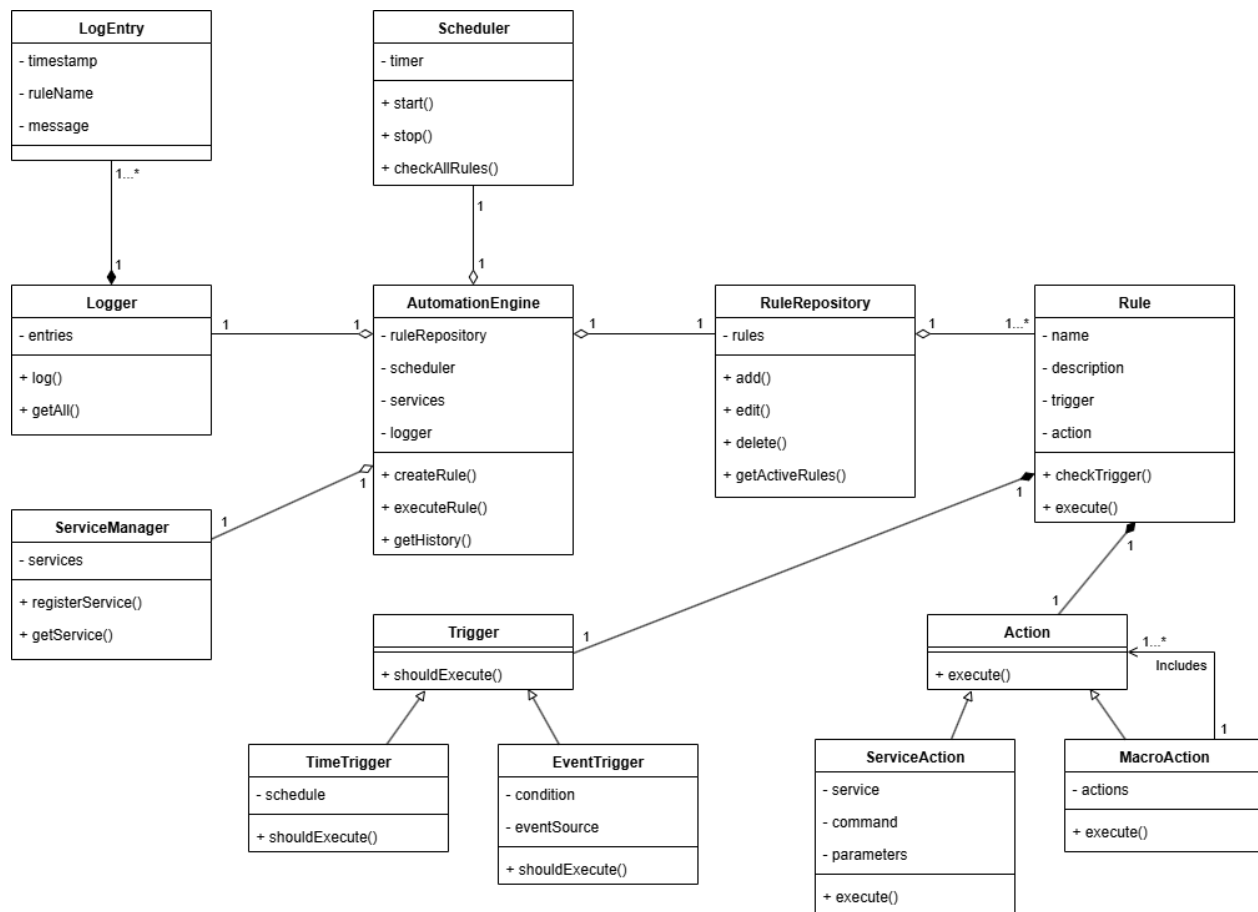


Рисунок 1.5. Діаграма класів системи

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 проектування – формування логіки реалізації компонентів та їхнього розгортання в системі.

1.6. Вибір бази даних

Для зберігання даних у системі `Flexible Automation Tool` обрано реляційну базу даних `SQLite`, оскільки вона є легкою, швидкою та працює у форматі вбудованої БД прямо всередині `.exe`-файлу. Це дозволяє запускати застосунок без окремого сервера чи додаткових служб, що значно спрощує розгортання та підвищує портативність.

Попри компактність, `SQLite` залишається надійною та стабільною технологією з повною підтримкою транзакцій і широкою екосистемою інструментів. Вона легко інтегрується з `C#` через ORM-рішення, такі як `Entity Framework Core` або `Dapper`, що дає можливість гнучко моделювати структуру системи, правильно реалізовувати зв'язки між сутностями та забезпечувати подальше масштабування або міграцію за потреби.

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 проектування 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 проектування, 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. Проектування розгортання системи

Проектування 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. 1.6) 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.

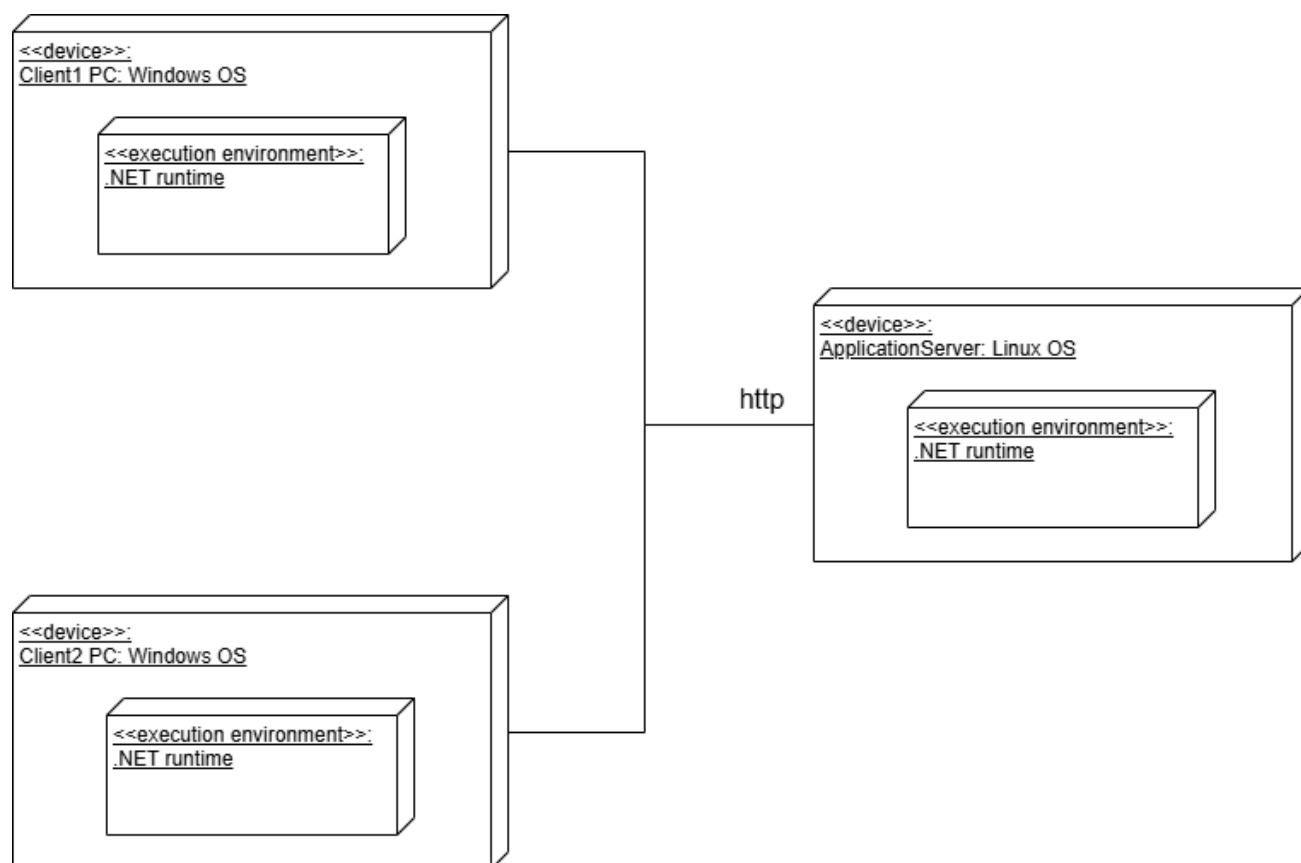


Рисунок 1.6. Діаграма розгортання системи

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

Структура бази даних (рис. 2.1) розроблена на основі логічної моделі системи та охоплює ключові сутності предметної області, що забезпечують повний цикл роботи інструменту автоматизації. Центральне місце в моделі займає таблиця Rules, у якій зберігаються всі правила автоматизації разом із базовими параметрами, такими як назва, опис та стан активності. Кожне правило містить посилання на відповідний тригер та одну або кілька дій, що дозволяє чітко визначати поведінку системи при виникненні певних умов.

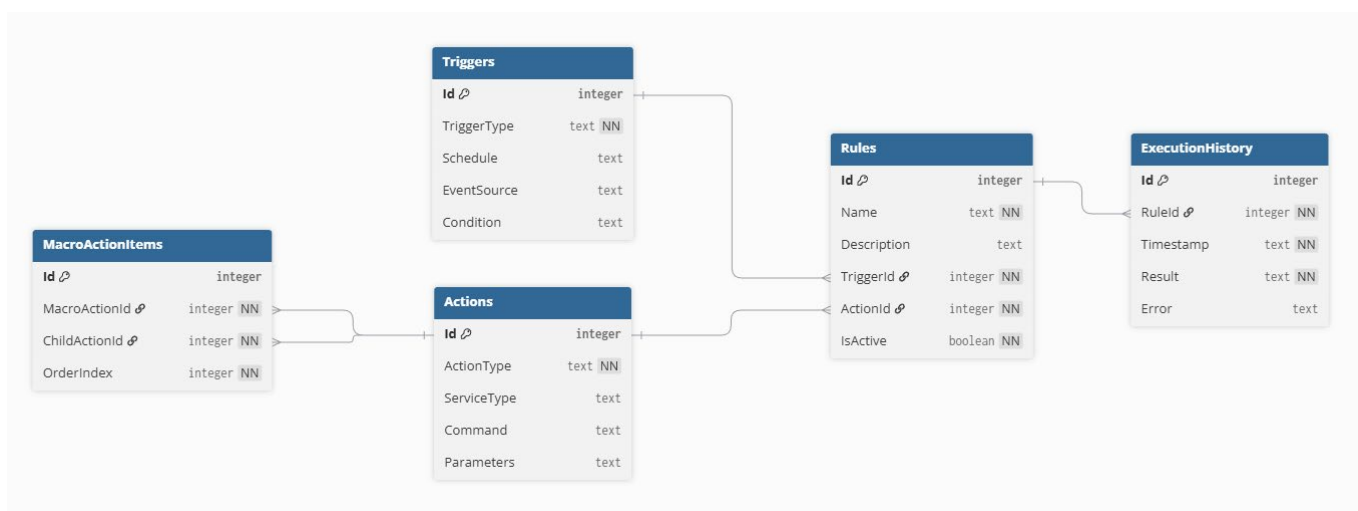


Рисунок 2.1. Структура бази даних

Таблиця Triggers описує умови, за яких правило повинно бути активоване. До тригера можуть належати такі параметри, як тип події, періодичність, порогові значення або інші специфічні критерії. Завдяки цьому система може реагувати на широкий спектр ситуацій — від часових подій і змін стану сервісів до користувацьких або зовнішніх сигналів. У свою чергу, таблиця Actions визначає конкретні операції, що виконуються після спрацювання тригера: виклики сервісів, запуск макродій, взаємодію з файловою системою чи відправку повідомлень. Це забезпечує модульність і можливість розширення системи новими типами дій.

Особливе значення має таблиця `ExecutionHistory`, яка фіксує інформацію про кожен запуск правила: час спрацювання, ідентифікатор правила, результат виконання та можливі помилки. Зберігання цього журналу дозволяє виконувати ретроспективний аналіз, відслідковувати стабільність роботи автоматизації та спрощує процес відлагодження. Наявність історії виконань також важлива для побудови звітів, оцінювання ефективності правил і забезпечення прозорості роботи системи.

Для реалізації складних сценаріїв у системі передбачена таблиця `MacroActionItems`, яка встановлює зв'язки між макродіями та їхніми піддіями. Макродія може складатися з набору послідовних або паралельних дій, що реалізує ієрархічну структуру сценарію. Така організація дозволяє будувати багатокрокові автоматизаційні процеси без втрати гнучкості та логічної цілісності. Завдяки цьому модель даних покриває як прості одноетапні правила, так і комплексні послідовності завдань.

Загалом, структура бази даних забезпечує чітку організацію інформації, підтримує модульність та розширюваність системи, а також створює основу для стабільної роботи механізму автоматизації, зберігаючи баланс між простотою реалізації та функціональною глибиною.

2.2. Архітектура системи

Архітектура системи побудована за принципами багаторівневого розподілу відповідальностей та слабого зв'язування між компонентами з метою забезпечення розширюваності, тестованості та простоти супроводу. На логічному рівні система складається з кількох основних підсистем: інтерфейс користувача (UI), прикладна логіка (`Engine / Facade`), механізми тригерингу і планування (`Scheduler + TriggerStrategy`), підсистема виконання команд (`CommandDispatcher + Action-об'єкти`), шар інтерпретації макросів (`Interpreter`), менеджер платформних сервісів (`PlatformFactory / ServiceManager`) та шар збереження стану (репозиторії + `SQLite`). Компоненти взаємодіють через чітко визначені інтерфейси, події і контракти, при

цьому залежності інтегровано за допомогою контейнера впровадження залежностей (DI). Така організація дозволяє змінювати реалізації окремих підсистем (наприклад, замінити сховище на інше або підключити новий тип платформи) без модифікації клієнтського коду.

У термінах розгортання, додаток складається з графічної оболонки, що працює в контексті desktop середовища (WinForms), та ядра, яке містить бізнес-логіку і може бути повторно використане в інших типах клієнтів. На старті програми контейнер DI створює екземпляри сінглтонів (Logger, Scheduler, SqliteDataContext тощо) і transient-об'єктів для діалогових форм. Scheduler періодично опитує репозиторій правил через стратегію тригерингу, виявляє готові для виконання правила і емулює події RuleReady, які підписані в Engine. Engine виконує фасадну роль: отримує правило, підготовлює його до виконання (інжектуює платформні сервіси у дії), делегує виконання CommandDispatcher-у, обробляє винятки та реєструє результати в історії виконань. Для персистентності використовуються нормалізовані таблиці Rules, Triggers, Actions, MacroActionItems, ExecutionHistory та Services; структура дозволяє зберігати як прості одноетапні дії, так і ієрархічні макродії. Журнал виконань зберігає часові мітки, статус і повідомлення, що дозволяє проводити ретроспективний аналіз та діагностику. Для інтеграції платформних можливостей (наприклад показ повідомлення, запуск програм) застосовано фабрику платформних сервісів, що інкапсулює особливості конкретної платформи й дозволяє підміняти реалізації під час тестування або для альтернативних середовищ.

2.2.1. Специфікація системи

Система призначена для організації автоматичного виконання правил, кожне з яких складається з тригера і пов'язаних дій. Вхідними елементами є визначення правил (назва, опис, активність), специфіка тригера (часовий розклад або подія) та опис дій (повідомлення, запуск програми, відкриття URL, запис у файл або послідовність дій — макродія). Система має забезпечувати створення, редагування, видалення правил через графічний інтерфейс, запуск і зупинку рушія автоматизації,

моніторинг і перегляд історії виконань з деталізацією часу, статусу і повідомлень/помилки.

Архітектурно система розділена на шар представлення (UI), шар прикладної логіки (Engine/Facade), механізм планування і виявлення готових правил (Scheduler + TriggerStrategy), підсистему виконання дій (CommandDispatcher + Action-об'єкти), шар персистентності (репозиторії над SQLite) і адаптери платформних сервісів (PlatformFactory, ServiceManager). Взаємодія між компонентами будується через інтерфейси та події: Scheduler опитує репозиторій правил і піднімає подію готовності, Engine реагує на події, готує і делегує виконання через диспетчер, Logger і репозиторій історії фіксують результати. Для макродій використовується інтерпретатор, який перетворює текстові визначення на набір об'єктів дій; MacroAction дозволяє композицію дій у ієрархічні сценарії.

Модель даних нормалізована: таблиці Triggers, Actions і Rules дозволяють зберігати різні типи тригерів і дій, MacroActionItems описує ієрархію макродій, ExecutionHistory фіксує кожен запуск з часовою міткою, статусом і повідомленням, а Services містить конфігурацію зовнішніх сервісів. CRUD-операції на рівні репозиторію виконуються транзакційно там, де потрібно (наприклад, додавання правила із залежними записами тригера й дії); це забезпечує узгодженість даних. Нефункціональні вимоги реалізовано через дизайн: UI працює у своєму потоці, довгі або потенційно блокуючі операції виконуються асинхронно або через диспетчер, для конкурентного доступу до таймерних задач Scheduler захищений від повторних викликів, логування і робота з БД мають обгортки, що запобігають краху UI при помилках з персистентністю. Для забезпечення переносимості абстраговано платформні сервісні інтеграції через фабрику сервісів, що дає можливість підмінювати реалізації без змін ядра.

2.2.2. Вибір та обґрунтування патернів реалізації

Вибір патернів проектування базується на вимогах розширюваності, тестованості, зрозумілості архітектури та контролю над персистентністю в

десктопному контексті. Інверсія залежностей і контейнер впровадження (Dependency Injection) застосовані як базовий механізм побудови зв'язків між компонентами. DI дозволяє визначати контракти через інтерфейси (IRuleRepository, IExecutionHistoryRepository, IMessageBoxService тощо) і підміняти реалізації на рівні конфігурації — це спрощує юніт-тестування, полегшує заміну сховища (SQLite ↔ InMemory) і дає контроль за життєвим циклом сервісів. Фасад (Facade) у вигляді IAutomationFacade/AutomationEngine об'єднує складну поведінку рушія (старт/стоп, створення правил, запуск роботи) за єдиним інтерфейсом, адже UI і зовнішні клієнти повинні взаємодіяти зі спрощеною абстракцією, а не з низкою дрібних сервісів; фасад також централізує логіку обробки помилок і запису історії, що підвищує відмовостійкість.

Репозиторій (Repository) виділено як шар, що інкапсулює доступ до даних: IRuleRepository і IExecutionHistoryRepository приховують реалізацію персистентності й дають змогу одним кодом працювати із різними бекендами. Це важливо для підтримки сценаріїв, де потрібно тимчасово працювати без бази (InMemory) або переселитися на інший механізм збереження. Стратегія (Strategy) застосована для політики виявлення готових правил (ITriggerStrategy / PollingTriggerStrategy); вона дає можливість впроваджувати альтернативні механізми (push-повідомлення, брокер подій), не змінюючи решту системи.

Командний патерн (Command) реалізований через ієрархію ActionBase і CommandDispatcher. Кожна дія — це об'єкт з інтерфейсом виконання і валідації; такий підхід дозволяє інкапсулювати поведінку, повторно використовувати і відкладати виконання, записувати/відтворювати дії. Composite-патерн у MacroAction створює ієрархію дій: макродія містить список піддій, що дозволяє будувати багатокрокові сценарії і зручно зберігати їх у БД через таблицю MacroActionItems. Інтерпретатор (Interpreter) використовується для трансформації текстових макросів у набір дій: це дає користувачу простий та гнучкий спосіб описувати послідовності без зміни коду; інтерпретатор відокремлює синтаксис макросу від внутрішньої моделі дій.

Фабрика (Factory) і Service Registry застосовані для інтеграції з платформними API: IPlatformFactory та ServiceManager інкапсулюють конструювання і реєстрацію реалізацій IMessageBoxService та інших platform-сервісів. Це дозволяє легко підміняти реалізації в тестах і при портуванні на іншу платформу. Набір подій і observer-механізм забезпечують слабке зв'язування між рушієм та UI: Engine піднімає події (RuleTriggered, RuleExecuted, RuleFailed), а AutomationEventHandler перетворює їх у виклики, придатні для UI-потoku; такий підхід покращує реактивність і спрощує масштабування поведінки сповіщень.

Обґрунтування вибору: поєднання DI + Repository дає тестованість і замінність; Facade спрощує інтеграцію UI; Command і Composite — природні для моделювання дій і макродій; Strategy дає конфігуруваність тригерингу; Factory забезпечує ізоляцію платформи. Технічні компроміси: замість повного ORM прийнято використовувати легкий SQL-шар для прозорості схеми і контролю міграцій у навчальному десктопному проєкті; це дає простішу залежність і явні транзакції, але збільшує кількість ручного коду при роботі з SQL. З іншого боку, у разі росту складності можна мігрувати на EF Core з мінімальними змінами інтерфейсів репозиторіїв.

2.3. Інструкція користувача

Застосунок знаходиться у папці App. Запустіть файл «FlexibleAutomationTool.UI.exe». Дайте застосунку пару хвилин, щоб запуснитися, особливо якщо це перший запуск.

Після запуску застосунку ви потрапите до головного вікна (рис. 2.2), яке служить центром управління автоматизацією: ліворуч відображається перелік збережених правил, праворуч — панель властивостей вибраного правила, а зверху розташовані керуючі кнопки для запуску і зупинки рушія, додавання, редагування, видалення правил і перегляду журналу виконань. Це вікно показує поточний стан рушія і виводить короткі повідомлення про останні події, тому при знайомстві з інструментом варто звернути увагу саме на його панель керування та список правил

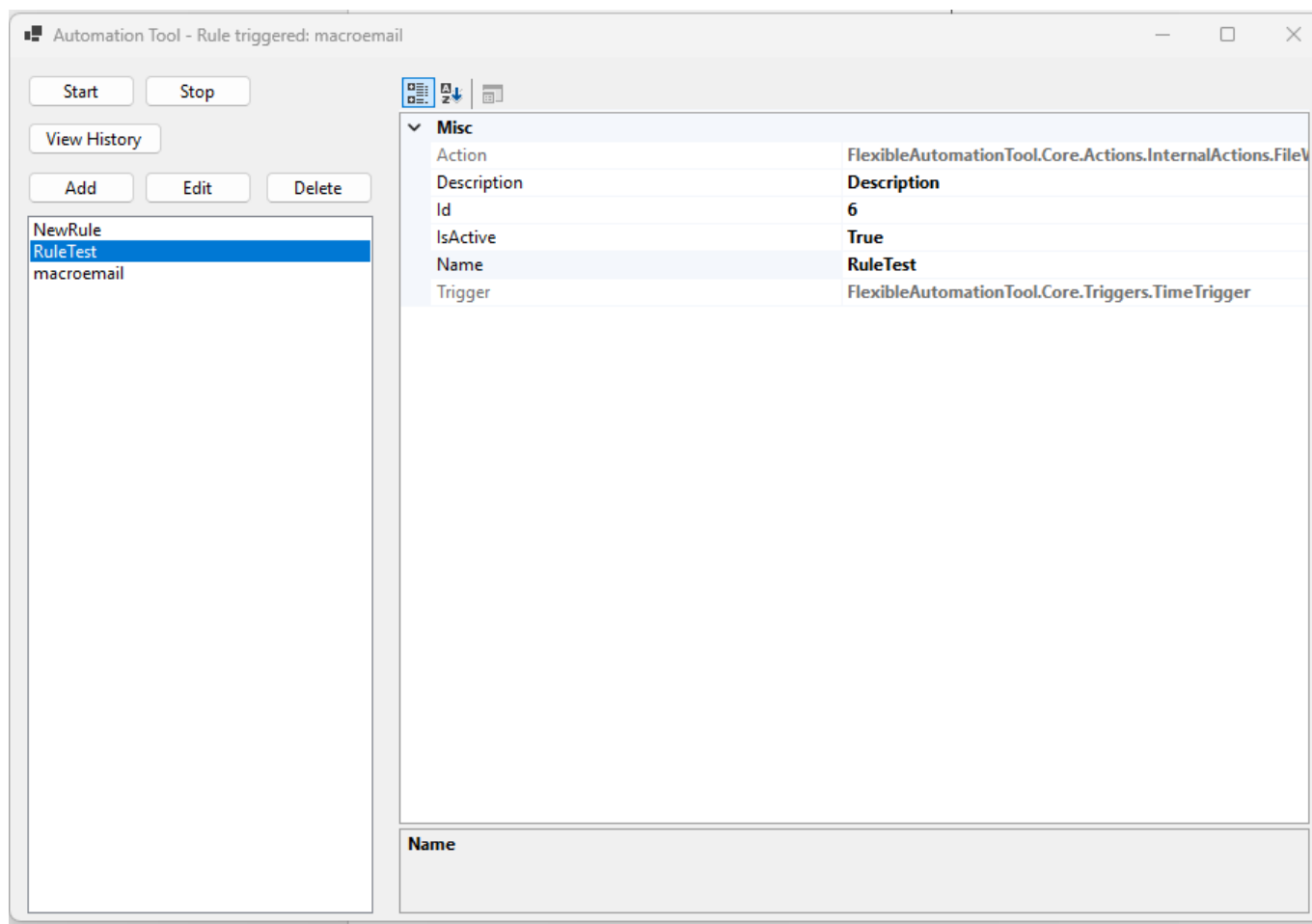


Рисунок 2.2. Головне вікно, список правил і панель керування

Щоб створити нове правило, натисніть кнопку для додавання: відкриється діалог, у якому в першому рядку задається назва і, за потреби, опис правила, потім обирається тип тригера і заповнюються відповідні поля, після цього обирається тип дії та вказуються її параметри. Діалог сконструйовано так, щоб показувати лише ті поля, які актуальні для обраних типів тригера і дії; наприклад, при виборі часового тригера з'являться поля для години і хвилини, а при виборі дії типу Message – поля для заголовка і тексту повідомлення. Після заповнення натисніть ОК, нове правило з'явиться у списку та, за замовчуванням, буде активоване.

The image shows a 'Create Rule' dialog box with the following fields and values:

- Name: RuleTest
- Description: Description
- Trigger Type: Time
- Hour: 0
- Minute: 51
- Action Type: FileWrite
- File Path: C:\Users\user\Documents\KPI\Software_dev_technologies\New Text Document.txt
- Content: Test write to file content

Buttons: OK, Cancel

Рисунок 2.3. Діалог створення правила

Часовий тригер задається просто: вкажіть годину та хвилину, коли правило має спрацювати; для подійного тригера ви можете вказати джерело події та опціональну умову у вигляді частини тексту або іншого простого критерію, що дозволяє відфільтровувати події. Вибрана конфігурація тригера зберігається разом з

правилом і використовується рушієм при періодичній перевірці активних правил; якщо правило тимчасово вимкнено, рушій його не опитує.

Дії можуть бути простими або складеними. Простими діями є показ повідомлення, запуск зовнішньої програми, відкриття URL або запис у файл – для кожного типу доступні поля, які пояснюють очікувані параметри. Складні сценарії створюються як макродії: введіть у полі Macro послідовність рядків, де кожен рядок описує одну піддію у простому форматі (наприклад: Message|Заголовок|Текст або Run|C:\Path\app.exe|--arg). Після збереження макродія з'явиться як одне правило, а піддії будуть виконуватися послідовно під час спрацьовування тригера.

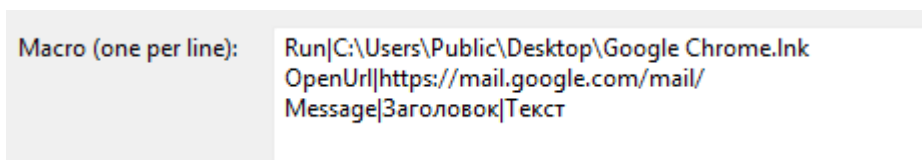


Рисунок 2.4. Приклад заповнення дій для макросу

Редагування правила відбувається через таку саму розкладку, що й створення: оберіть правило зі списку й натисніть Edit, внесіть необхідні зміни і підтвердіть їх натисканням ОК. Видалення правила виконується кнопкою Delete і супроводжується запитом підтвердження, а тимчасове відключення правила здійснюється через прапорець активності у панелі властивостей — це дозволяє зберегти налаштування, але призупинити перевірки рушієм. Всі зміни зберігаються у локальній базі SQLite, тож після перезапуску програми конфігурація залишиться незмінною.

Edit Rule

Name: RuleTest

Description: Description

Trigger: Time
 Hour: 0 Minute: 51

Event Source:

Condition:

Action: FileWrite

File Path: D:\Smegal\Documents\Universities\KPI\Software_dev_technologies\New T

Content: Test write to file content

OK Cancel

Рисунок 2.5. Приклад редагування дії

Запуск рушія здійснюється кнопкою **Start: Scheduler** починає періодично опитувати всі активні правила згідно з вибраною стратегією (за замовчуванням — опитування кожні 30 секунд) і у разі готовності передає правило в Engine для виконання. Engine виконує попередню підготовку (наприклад, інжектуює сервіс показу повідомлень у дії типу **MessageBox**), делегує виконання диспетчеру команд і фіксує результат у журналі виконань. Якщо необхідно зупинити автоматичні перевірки, натисніть **Stop: Scheduler** зупиниться і перевірки не відбуватимуться, поки ви знову не натиснете **Start**. У разі підозри на подвійне виконання правила перевірте, що рушій

не запущено кілька разів одночасно і що інтервал опитування налаштований коректно.

Журнал логів доступний через кнопку View History: у вікні журналу відображаються записи з часовими мітками, станом виконання і повідомленнями або текстом помилок. Ви можете фільтрувати записи за конкретним правилом, щоб швидко знайти дані про останні спроби його виконання або відфільтрувати глобальні повідомлення рушія. Якщо під час виконання правила виникла помилка, у цьому вікні буде показана додаткова інформація, що допоможе зрозуміти причину (наприклад, відсутня зовнішня програма або помилка виконання макросу).

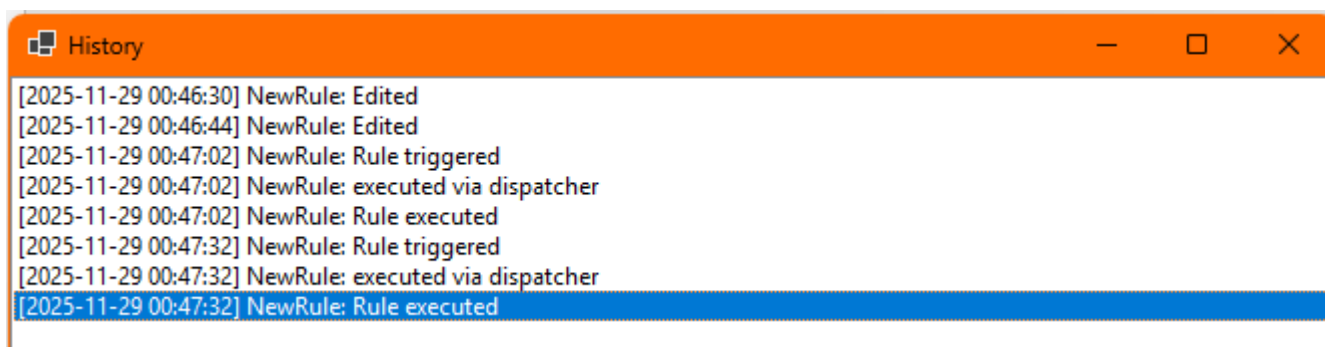


Рисунок 2.6: View History, журнал виконань

Система інформує користувача про критичні помилки в окремих модальних повідомленнях; якщо повідомлення про помилку стосується відсутнього сервісу (наприклад, IMessageBoxService), повідомлення покаже точну причину і поради щодо виправлення, одночасно записавши деталі до журналу виконань для подальшого аналізу. У випадку проблем із доступом до файлу бази або несумісності схеми програма надасть повідомлення і тимчасово переключиться в режим роботи в пам'яті, щоб інтерфейс залишався доступним – в такому режимі правила працюватимуть, але не збережуться після закриття програми.

ВИСНОВКИ

У ході виконання курсової роботи розробили модульну та розширювану систему автоматизації дій користувача, побудовану на принципах об'єктно-орієнтованого програмування та сучасних підходах до архітектурного проектування. Основною ідеєю проєкту було створення програмного засобу, який дає змогу визначати правила у форматі «тригер -> дія» та забезпечує їх автоматичне виконання згідно з часовими або подієвими умовами. Поставлена мета була успішно досягнута, що підтверджується функціональністю реалізованих компонентів і відповідністю системи сформульованим вимогам.

У процесі роботи проаналізували та застосували патерни проектування, зокрема Strategy, Command, Observer та Abstract Factory, які відіграли ключову роль у структуризації логіки системи та забезпеченні можливості подальшого розширення функціональності. Використання патернів дозволило створити гнучку архітектуру, яка легко адаптується до нових типів тригерів, дій, зовнішніх сервісів і UI-компонентів без необхідності змінювати існуючий код. Це підвищило підтримуваність системи та відповідність принципам SOLID.

Особливу увагу приділили розподілу відповідальностей між модулями. Створено окремі компоненти для керування правилами (RuleRepository), планування їх виконання (Scheduler), логування (Logger), керування зовнішніми службами (ServiceManager) та координації всієї системи (AutomationEngine). Такий підхід сприяє покращенню структурованості проєкту й дав можливість ізолювати логіку бізнес-рівня від UI та інфраструктури. Окремо реалізували механізм залежностей за допомогою концепції Dependency Injection, що дозволило спростити тестування та заміну реалізацій без втручання в роботу інших модулів.

Створили користувацький інтерфейс, який демонструє роботу внутрішніх компонентів та дозволяє виконувати базові дії, зокрема запуск автоматизацій та взаємодію з діями. UI та логіка були розділені, що відповідає сучасним практикам побудови програмних систем.

Розроблена система є масштабованою та може бути суттєво розширена в майбутньому. Подальший розвиток може включати впровадження збереження правил і логів у базу даних, інтеграцію зовнішніх API та сервісів, створення візуального конструктора правил, підтримку складніших тригерів (файлових, мережових, сенсорних), а також удосконалення механізму планування. Крім того, система може бути адаптована під архітектурний стиль SOA або мікросервісний підхід, що дозволить розподіляти окремі компоненти між процесами або навіть серверами.

У цілому, виконана курсова робота демонструє можливість застосування теоретичних знань із проєктування програмного забезпечення, архітектури, патернів і принципів інженерії на практиці. Створена система відповідає початковим цілям, є функціонально завершеною та закладає надійний фундамент для подальшого розвитку та вдосконалення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] E. Gamma, R. Helm, R. Johnson, та J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley, 1994.
- [2] G. Hohpe, B. Woolf, *Enterprise Integration Patterns*. Addison-Wesley, 2003.
- [3] Microsoft, “C# Documentation,” *Microsoft Learn*. [Онлайн]. Доступно: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. Доступ отримано: Листопад 2025.
- [4] Microsoft, “.NET Dependency Injection,” *Microsoft Learn*. [Онлайн]. Доступно: <https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>. Доступ отримано: Листопад 2025.
- [5] Microsoft, “Windows Forms Overview,” *Microsoft Learn*. [Онлайн]. Доступно: <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/>. Доступ отримано: Листопад 2025.
- [6] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2020.
- [7] M. Richards, N. Ford, *Fundamentals of Software Architecture*. O’Reilly Media, 2020.
- [8] I. Sommerville, *Software Engineering*, 10th ed. Harlow, U.K.: Pearson Education, 2016.
- [9] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2018.
- [10] Zapier Inc., “What is Zapier?”. [Онлайн]. Доступно: <https://help.zapier.com/hc/en-us/articles/37518970271245-What-is-Zapier>. Доступ отримано: Жовтень 2025.

ДОДАТКИ

```

namespace FlexibleAutomationTool.Core.Actions.InternalActions
{
    public class FileWriteAction : ActionBase
    {
        public string FilePath { get; set; } = "";
        public string Content { get; set; } = "";

        public override void Execute()
        {
            try
            {
                File.WriteAllText(FilePath, Content);
            }
            catch (Exception ex)
            {
                throw new InvalidOperationException($"Failed to write file '{FilePath}': {ex.Message}", ex);
            }
        }

        public override bool Validate() => !string.IsNullOrEmpty(FilePath);
    }
}

using FlexibleAutomationTool.Core.Interfaces;

namespace FlexibleAutomationTool.Core.Actions.InternalActions
{

```

```
public class MessageBoxAction : ActionBase
{
```

```
    public string Message { get; set; } = "";
```

```
    public string Title { get; set; } = "";
```

// Allows DI via property so interpreter can create actions without requiring the service at construction time

```
    public IMessageBoxService? MessageBoxService { get; set; }
```

```
    public MessageBoxAction()
```

```
    {
```

```
    }
```

// Backwards-compatible constructor used by UI code that provides the service at creation time

```
    public MessageBoxAction(IMessageBoxService service)
```

```
    {
```

```
        MessageBoxService = service;
```

```
    }
```

```
    public override void Execute()
```

```
    {
```

```
        try
```

```
        {
```

```
            if (MessageBoxService == null)
```

```
                throw new System.InvalidOperationException("No IMessageBoxService available to show message. Set MessageBoxService before executing this action.");
```

```
            MessageBoxService.Show(Message, Title);
```

```

    }
    catch (System.Exception ex)
    {
        throw new System.InvalidOperationException($"Failed to show message box:
{ex.Message}", ex);
    }
}

```

```

    public override bool Validate() => !string.IsNullOrEmpty(Message);
}
}

```

```

using System.Diagnostics;

```

```

namespace FlexibleAutomationTool.Core.Actions.InternalActions

```

```

{
    public class OpenUrlAction : ActionBase
    {
        public string Url { get; set; } = "";

        public override void Execute()
        {
            try
            {
                Process.Start(new ProcessStartInfo(Url) { UseShellExecute = true });
            }
            catch (System.Exception ex)
            {

```

```

        throw new System.InvalidOperationException($"Failed to open URL '{Url}':
{ex.Message}", ex);
    }
}

```

```

    public override bool Validate() => Url.StartsWith("http");
}
}

```

```

using System.Diagnostics;

```

```

namespace FlexibleAutomationTool.Core.Actions.InternalActions
{
    public class RunProgramAction : ActionBase
    {
        public string Path { get; set; } = "";
        public string? Arguments { get; set; }

        public override void Execute()
        {
            try
            {
                var psi = new ProcessStartInfo(Path)
                {
                    Arguments = Arguments ?? string.Empty,
                    UseShellExecute = true
                };
                Process.Start(psi);
            }
        }
    }
}

```

```

        catch (System.Exception ex)
        {
            throw new System.InvalidOperationException($"Failed to start process '{Path}':
{ex.Message}", ex);
        }
    }
}

```

```

        public override bool Validate() => !string.IsNullOrEmpty(Path);
    }
}

```

```

namespace FlexibleAutomationTool.Core.Actions
{
    public abstract class ActionBase
    {
        public int Id { get; set; }
        public abstract void Execute();
        public abstract bool Validate();
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace FlexibleAutomationTool.Core.Actions
{

```



```

public class MacroAction : ActionBase
{
    public List<ActionBase> Actions { get; set; } = new();

    public override void Execute()
    {
        List<Exception> errors = null;

        foreach (var a in Actions)
        {
            try
            {
                a.Execute();
            }
            catch (Exception ex)
            {
                // collect exceptions to report after attempting all actions
                errors ??= new List<Exception>();
                errors.Add(ex);
            }
        }

        if (errors != null && errors.Count > 0)
        {
            throw new AggregateException("One or more actions in the macro failed.",
errors);
        }
    }
}

```

```

        public override bool Validate() => Actions.Count > 0;
    }
}

namespace FlexibleAutomationTool.Core.Actions
{
    public class ServiceAction : ActionBase
    {
        public string ServiceType { get; set; } = string.Empty;
        public string Command { get; set; } = string.Empty;
        public string? Parameters { get; set; }

        public override void Execute()
        {
            Console.WriteLine($"[ServiceAction]           Service={ServiceType}
Command={Command} Params={Parameters}");
        }

        public override bool Validate() => !string.IsNullOrEmpty(ServiceType) &&
!string.IsNullOrEmpty(Command);
    }
}

using Microsoft.Data.Sqlite;

namespace FlexibleAutomationTool.Core.Data
{
    public class SqliteDataContext : IDisposable
    {

```

```
private readonly SqlConnection _connection;
public SqlConnection Connection => _connection;
```

```
public SqliteDataContext(string dbPath)
{
    _connection = new SqlConnection($"Data Source={dbPath};Cache=Shared");
    _connection.Open();
    EnsureSchema();
}
```

```
private void EnsureSchema()
{
    using var cmd = _connection.CreateCommand();
    cmd.CommandText = "PRAGMA foreign_keys = ON;";
    cmd.ExecuteNonQuery();
```

// Check if ExecutionHistory.RuleId column exists and is NOT NULL; if so, migrate to nullable

```
try
{
    using var pragma = _connection.CreateCommand();
    pragma.CommandText = "PRAGMA table_info('ExecutionHistory')";
    using var rdr = pragma.ExecuteReader();
    bool found = false;
    bool ruleIdNotNull = false;
    while (rdr.Read())
    {
        var colName = rdr.GetString(1); // name
        if (string.Equals(colName, "RuleId", StringComparison.OrdinalIgnoreCase))
```

```

{
    found = true;
    // notnull column is at index 3
    ruleIdNotNull = rdr.GetInt32(3) != 0;
    break;
}
}

```

```

if (found && ruleIdNotNull)

```

```

{
    // Migrate to nullable RuleId.
    // PRAGMA foreign_keys must be toggled outside of an explicit transaction,
    so turn it off first,
    // then perform the DDL within a transaction, and finally re-enable PRAGMA
    after commit.

```

```

    // Turn off foreign keys temporarily on the connection level
    using (var setOff = _connection.CreateCommand())
    {
        setOff.CommandText = "PRAGMA foreign_keys = OFF;";
        setOff.ExecuteNonQuery();
    }

```

```

    using var tx = _connection.BeginTransaction();

```

```

    using (var cmdRename = _connection.CreateCommand())
    {
        cmdRename.Transaction = tx;

```

```

        cmdRename.CommandText = "ALTER TABLE ExecutionHistory
RENAME TO ExecutionHistory_old;";
        cmdRename.ExecuteNonQuery();
    }

    using (var cmdCreate = _connection.CreateCommand())
    {
        cmdCreate.Transaction = tx;
        cmdCreate.CommandText = @"CREATE TABLE IF NOT EXISTS
ExecutionHistory (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    RuleId INTEGER,
    ExecutedAt TEXT NOT NULL,
    Status TEXT,
    Message TEXT,
    FOREIGN KEY (RuleId) REFERENCES Rules(Id) ON DELETE CASCADE
);";

        cmdCreate.ExecuteNonQuery();
    }

    using (var cmdCopy = _connection.CreateCommand())
    {
        cmdCopy.Transaction = tx;
        // convert invalid or zero RuleId to NULL to satisfy foreign key
        cmdCopy.CommandText = @"INSERT INTO ExecutionHistory (RuleId,
ExecutedAt, Status, Message)
SELECT CASE
    WHEN RuleId IS NULL THEN NULL
    WHEN RuleId = 0 THEN NULL

```

```

        WHEN NOT EXISTS(SELECT 1 FROM Rules WHERE Id =
ExecutionHistory_old.RuleId) THEN NULL
        ELSE RuleId END,
        ExecutedAt, Status, Message
FROM ExecutionHistory_old;";

        cmdCopy.ExecuteNonQuery();
    }

    using (var cmdDrop = _connection.CreateCommand())
    {
        cmdDrop.Transaction = tx;
        cmdDrop.CommandText = "DROP TABLE IF EXISTS
ExecutionHistory_old;";
        cmdDrop.ExecuteNonQuery();
    }

    tx.Commit();

    // Re-enable foreign keys on the connection AFTER the transaction has been
committed
    using (var setOn = _connection.CreateCommand())
    {
        setOn.CommandText = "PRAGMA foreign_keys = ON;";
        setOn.ExecuteNonQuery();
    }
}
}
catch
{

```

```
// ignore migration errors and continue ensuring schema below
}

using var cmd2 = _connection.CreateCommand();
cmd2.CommandText = @"-- Triggers
CREATE TABLE IF NOT EXISTS Triggers (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    TriggerType TEXT NOT NULL,
    Schedule TEXT,
    EventSource TEXT,
    Condition TEXT
);

-- Actions
CREATE TABLE IF NOT EXISTS Actions (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    ActionType TEXT NOT NULL,
    ServiceType TEXT,
    Command TEXT,
    Parameters TEXT
);

-- Rules
CREATE TABLE IF NOT EXISTS Rules (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    Description TEXT,
    TriggerId INTEGER NOT NULL,
    ActionId INTEGER NOT NULL,
```

```

IsActive INTEGER NOT NULL CHECK (IsActive IN (0, 1)),
FOREIGN KEY (TriggerId) REFERENCES Triggers(Id) ON DELETE CASCADE,
FOREIGN KEY (ActionId) REFERENCES Actions(Id) ON DELETE CASCADE
);

```

```

-- ExecutionHistory

```

```

CREATE TABLE IF NOT EXISTS ExecutionHistory (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    RuleId INTEGER,
    ExecutedAt TEXT NOT NULL,
    Status TEXT,
    Message TEXT,
    FOREIGN KEY (RuleId) REFERENCES Rules(Id) ON DELETE CASCADE
);

```

```

-- Services

```

```

CREATE TABLE IF NOT EXISTS Services (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    ServiceType TEXT NOT NULL,
    Config TEXT
);

```

```

-- MacroActionItems

```

```

CREATE TABLE IF NOT EXISTS MacroActionItems (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    MacroActionId INTEGER NOT NULL,
    ChildActionId INTEGER NOT NULL,
    OrderIndex INTEGER NOT NULL,

```



```
FOREIGN KEY (MacroActionId) REFERENCES Actions(Id) ON DELETE
CASCADE,
```

```
FOREIGN KEY (ChildActionId) REFERENCES Actions(Id) ON DELETE CASCADE
);";
```

```
cmd2.ExecuteNonQuery();
```

```
try
```

```
{
```

```
using var cmdCheck = _connection.CreateCommand();
```

```
cmdCheck.CommandText = "SELECT Id FROM Rules WHERE Name=$name
LIMIT 1";
```

```
cmdCheck.Parameters.AddWithValue("$name", "__System__");
```

```
var res = cmdCheck.ExecuteScalar();
```

```
if (res == null)
```

```
{
```

```
using var tx = _connection.BeginTransaction();
```

```
using (var cmdInsT = _connection.CreateCommand())
```

```
{
```

```
cmdInsT.Transaction = tx;
```

```
cmdInsT.CommandText = "INSERT INTO Triggers (TriggerType,
Schedule, EventSource, Condition) VALUES ($tt,$sch,$es,$cond);";
```

```
cmdInsT.Parameters.AddWithValue("$tt", "EventTrigger");
```

```
cmdInsT.Parameters.AddWithValue("$sch", (object)DBNull.Value);
```

```
cmdInsT.Parameters.AddWithValue("$es", (object)DBNull.Value);
```

```
cmdInsT.Parameters.AddWithValue("$cond", (object)DBNull.Value);
```

```
cmdInsT.ExecuteNonQuery();
```

```
}
```

```

long triggerId;
using (var cmdLast = _connection.CreateCommand())
{
    cmdLast.Transaction = tx;
    cmdLast.CommandText = "SELECT last_insert_rowid();";
    triggerId = (long)cmdLast.ExecuteScalar();
}

```

```

using (var cmdInsA = _connection.CreateCommand())
{
    cmdInsA.Transaction = tx;
    cmdInsA.CommandText = "INSERT INTO Actions (ActionType,
ServiceType, Command, Parameters) VALUES ($at,$st,$cm,$pm);";
    cmdInsA.Parameters.AddWithValue("$at", "MessageBox");
    cmdInsA.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdInsA.Parameters.AddWithValue("$cm", "System");
    cmdInsA.Parameters.AddWithValue("$pm", "System log placeholder");
    cmdInsA.ExecuteNonQuery();
}

```

```

long actionId;
using (var cmdLast2 = _connection.CreateCommand())
{
    cmdLast2.Transaction = tx;
    cmdLast2.CommandText = "SELECT last_insert_rowid();";
    actionId = (long)cmdLast2.ExecuteScalar();
}

```

```

using (var cmdInsR = _connection.CreateCommand())

```

```

        {
            cmdInsR.Transaction = tx;
            cmdInsR.CommandText = "INSERT INTO Rules (Name, Description,
TriggerId, ActionId, IsActive) VALUES ($name,$desc,$tid,$aid,$ia);";
            cmdInsR.Parameters.AddWithValue("$name", "__System__");
            cmdInsR.Parameters.AddWithValue("$desc", "System placeholder rule for
execution history");
            cmdInsR.Parameters.AddWithValue("$tid", triggerId);
            cmdInsR.Parameters.AddWithValue("$aid", actionId);
            cmdInsR.Parameters.AddWithValue("$ia", 0);
            cmdInsR.ExecuteNonQuery();
        }

        tx.Commit();
    }
}
catch
{
    // ignore failures creating system rule
}
}

public void Dispose()
{
    _connection?.Dispose();
}
}
}

```

```

using Microsoft.Data.Sqlite;
using System;
using System.Collections.Generic;
using System.Linq;
using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Repositories;
using System.Diagnostics;

namespace FlexibleAutomationTool.Core.Data
{
    public class SqliteLogger : IExecutionHistoryRepository, IDisposable
    {
        private readonly SqliteDataContext _context;
        private readonly object _lock = new();
        private readonly bool _ruleIdNotNull;
        private readonly int? _systemRuleId;

        public SqliteLogger(SqliteDataContext context)
        {
            _context = context ?? throw new ArgumentNullException(nameof(context));

            // detect if ExecutionHistory.RuleId column is NOT NULL in this DB
            try
            {
                using var cmd = _context.Connection.CreateCommand();
                cmd.CommandText = "PRAGMA table_info('ExecutionHistory');";
                using var rdr = cmd.ExecuteReader();
                while (rdr.Read())
                {

```

```

var colName = rdr.GetString(1);
if (string.Equals(colName, "RuleId", StringComparison.OrdinalIgnoreCase))
{
    _ruleIdNotNull = rdr.GetInt32(3) != 0; // notnull column
    break;
}
}
}
catch
{
    _ruleIdNotNull = false;
}

```

// If DB requires RuleId to be NOT NULL, try to locate the system placeholder rule created by the data context

```

if (_ruleIdNotNull)
{
    try
    {
        using var cmd = _context.Connection.CreateCommand();
        cmd.CommandText = "SELECT Id FROM Rules WHERE Name=$name
LIMIT 1";
        cmd.Parameters.AddWithValue("$name", "__System__");
        var res = cmd.ExecuteScalar();
        if (res != null && res != DBNull.Value)
        {
            // SQLite integers come back as long
            _systemRuleId = Convert.ToInt32(res);
        }
    }
}

```

```

    }
    catch
    {
        _systemRuleId = null;
    }
}
}

```

```

public void Add(int ruleId, DateTime executedAt, string status, string message)
{
    lock (_lock)
    {
        try
        {
            // Local copy of rule id we'll use for insertion
            var insertRuleId = ruleId;

            // If DB schema requires RuleId NOT NULL then map non-rule entries (ruleId
            <= 0)
            // to the system placeholder rule if available; otherwise skip.
            if (insertRuleId <= 0 && _ruleIdNotNull)
            {
                if (_systemRuleId.HasValue)
                {
                    insertRuleId = _systemRuleId.Value;
                }
                else
                {

```

```

        Debug.WriteLine($"SqliteLogger:  skipped  non-rule  history  entry
(ruleId={ruleId})  because  ExecutionHistory.RuleId  is  NOT  NULL  and  no  system
placeholder rule available.");

```

```

        return;
    }
}

```

```

using var cmd = _context.Connection.CreateCommand();
if (insertRuleId <= 0)
{
    // omit RuleId column when DB allows NULL
    cmd.CommandText = "INSERT INTO ExecutionHistory (ExecutedAt,
Status, Message) VALUES ($ts,$st,$msg);";
    cmd.Parameters.AddWithValue("$ts", executedAt.ToString("o"));
    cmd.Parameters.AddWithValue("$st", status ?? (object)DBNull.Value);
    cmd.Parameters.AddWithValue("$msg",          message          ??
(object)DBNull.Value);
}
else
{
    // Ensure referenced Rule exists to avoid FK violations
    using (var chk = _context.Connection.CreateCommand())
    {
        chk.CommandText = "SELECT 1 FROM Rules WHERE Id=$rid LIMIT
1";
        chk.Parameters.AddWithValue("$rid", insertRuleId);
        var exists = chk.ExecuteScalar();
        if (exists == null)
        {

```

```

        // If referenced rule doesn't exist, try to fallback to system placeholder
        (if DB requires NOT NULL)

        Debug.WriteLine($"SqliteLogger:      referenced      rule      with
Id={insertRuleId} was not found.");

        if (_ruleIdNotNull && _systemRuleId.HasValue)
        {
            insertRuleId = _systemRuleId.Value;
            Debug.WriteLine($"SqliteLogger: falling back to system placeholder
rule Id={insertRuleId} for history entry.");
        }
        else if (_ruleIdNotNull && !_systemRuleId.HasValue)
        {
            Debug.WriteLine($"SqliteLogger: cannot insert FK entry because
ExecutionHistory.RuleId is NOT NULL and no system placeholder exists; insert skipped
for ruleId={ruleId}.");
            return;
        }
        else
        {
            // DB allows NULL, insert without RuleId
            cmd.CommandText = "INSERT INTO ExecutionHistory
(ExecutedAt, Status, Message) VALUES ($ts,$st,$msg)";
            cmd.Parameters.AddWithValue("$ts", executedAt.ToString("o"));
            cmd.Parameters.AddWithValue("$st",          status          ??
(object)DBNull.Value);
            cmd.Parameters.AddWithValue("$msg",          message          ??
(object)DBNull.Value);
            cmd.ExecuteNonQuery();

```



```

        return;
    }
}

cmd.CommandText = "INSERT INTO ExecutionHistory (RuleId,
ExecutedAt, Status, Message) VALUES ($rid,$ts,$st,$msg)";
cmd.Parameters.AddWithValue("$rid", insertRuleId);
cmd.Parameters.AddWithValue("$ts", executedAt.ToString("o"));
cmd.Parameters.AddWithValue("$st", status ?? (object)DBNull.Value);
cmd.Parameters.AddWithValue("$msg", message ??
(object)DBNull.Value);
}

cmd.ExecuteNonQuery();
}
catch (SqliteException ex)
{
    // Swallow SQLite write errors to avoid breaking UI; log for diagnostics
    Debug.WriteLine($"SqliteLogger.Add failed: {ex.SqliteErrorCode} -
{ex.Message}");
}
catch (Exception ex)
{
    Debug.WriteLine($"SqliteLogger.Add unexpected error: {ex}");
}
}
}

```

```

public IEnumerable<LogEntry> GetAll()
{
    var list = new List<LogEntry>();
    lock (_lock)
    {
        using var cmd = _context.Connection.CreateCommand();
        cmd.CommandText = "SELECT Id, RuleId, ExecutedAt, Status, Message FROM
ExecutionHistory ORDER BY Id";
        using var rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
            var id = rdr.GetInt32(0);
            int? ruleId = rdr.IsDBNull(1) ? null : rdr.GetInt32(1);
            var ts = DateTime.Parse(rdr.GetString(2));
            var status = rdr.IsDBNull(3) ? null : rdr.GetString(3);
            var message = rdr.IsDBNull(4) ? null : rdr.GetString(4);
            list.Add(new LogEntry { Id = id, Timestamp = ts, RuleName =
ruleId?.ToString() ?? string.Empty, Message = message ?? status ?? string.Empty });
        }
    }
    return list;
}

public IEnumerable<LogEntry> GetByRuleId(int ruleId)
{
    var list = new List<LogEntry>();
    lock (_lock)
    {
        using var cmd = _context.Connection.CreateCommand();

```

```

        cmd.CommandText = "SELECT Id, RuleId, ExecutedAt, Status, Message FROM
ExecutionHistory WHERE RuleId=$rid ORDER BY Id";

        cmd.Parameters.AddWithValue("$rid", ruleId);
        using var rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
            var id = rdr.GetInt32(0);
            var ts = DateTime.Parse(rdr.GetString(2));
            var status = rdr.IsDBNull(3) ? null : rdr.GetString(3);
            var message = rdr.IsDBNull(4) ? null : rdr.GetString(4);
            list.Add(new LogEntry { Id = id, Timestamp = ts, RuleName =
ruleId.ToString(), Message = message ?? status ?? string.Empty });
        }
    }
    return list;
}

public void Dispose()
{
    // context is owned by DI, do not dispose here
}
}
}

using Microsoft.Data.Sqlite;
using System;
using System.Collections.Generic;
using System.Text;
using FlexibleAutomationTool.Core.Models;

```

```

using FlexibleAutomationTool.Core.Repositories;

namespace FlexibleAutomationTool.Core.Data
{
    public class SqliteRuleRepository : IRuleRepository, IDisposable
    {
        private readonly SqliteDataContext _context;
        private readonly object _lock = new();

        public SqliteRuleRepository(SqliteDataContext context)
        {
            _context = context ?? throw new ArgumentNullException(nameof(context));
        }

        public IEnumerable<Rule> GetAll()
        {
            var list = new List<Rule>();
            lock (_lock)
            {
                using var cmd = _context.Connection.CreateCommand();
                cmd.CommandText = @"SELECT r.Id, r.Name, r.Description, r.IsActive,
t.TriggerType, t.Schedule, t.EventSource, t.Condition,
a.ActionType, a.ServiceType, a.Command, a.Parameters, a.Id as ActionId
FROM Rules r
JOIN Triggers t ON r.TriggerId = t.Id
JOIN Actions a ON r.ActionId = a.Id
ORDER BY r.Id";
                using var rdr = cmd.ExecuteReader();
                while (rdr.Read())

```

```

{
    var id = rdr.GetInt32(0);
    var name = rdr.GetString(1);
    var desc = rdr.IsDBNull(2) ? null : rdr.GetString(2);
    var isActive = rdr.GetInt32(3) != 0;

    var triggerType = rdr.IsDBNull(4) ? null : rdr.GetString(4);
    var schedule = rdr.IsDBNull(5) ? null : rdr.GetString(5);
    var eventSource = rdr.IsDBNull(6) ? null : rdr.GetString(6);
    var condition = rdr.IsDBNull(7) ? null : rdr.GetString(7);

    var actionType = rdr.IsDBNull(8) ? null : rdr.GetString(8);
    var serviceType = rdr.IsDBNull(9) ? null : rdr.GetString(9);
    var command = rdr.IsDBNull(10) ? null : rdr.GetString(10);
    var parameters = rdr.IsDBNull(11) ? null : rdr.GetString(11);
    var actionId = rdr.IsDBNull(12) ? 0 : rdr.GetInt32(12);

    var rule = new Rule { Id = id, Name = name, Description = desc, IsActive =
isActive };

    // Reconstruct trigger
    if (string.Equals(triggerType, "TimeTrigger",
StringComparison.OrdinalIgnoreCase) && !string.IsNullOrEmpty(schedule))
    {
        var parts = schedule.Split(':');
        if (parts.Length == 2 && int.TryParse(parts[0], out var hour) &&
int.TryParse(parts[1], out var minute))
            rule.Trigger = new FlexibleAutomationTool.Core.Triggers.TimeTrigger {
Hour = hour, Minute = minute };
    }
}

```

```

        else
            rule.Trigger = new
FlexibleAutomationTool.Core.Triggers.EventTrigger(eventSource, null);
    }
    else if (string.Equals(triggerType, "EventTrigger",
StringComparison.OrdinalIgnoreCase))
    {
        rule.Trigger = new
FlexibleAutomationTool.Core.Triggers.EventTrigger(eventSource, null);
    }
    else
    {
        rule.Trigger = new
FlexibleAutomationTool.Core.Triggers.EventTrigger(null, null);
    }

    // Reconstruct action
    if (string.Equals(actionType, "MessageBox",
StringComparison.OrdinalIgnoreCase))
    {
        rule.Action = new
FlexibleAutomationTool.Core.Actions.InternalActions.MessageBoxAction { Title =
command ?? string.Empty, Message = parameters ?? string.Empty };
    }
    else if (string.Equals(actionType, "RunProgram",
StringComparison.OrdinalIgnoreCase))
    {

```

```

        rule.Action = new
FlexibleAutomationTool.Core.Actions.InternalActions.RunProgramAction { Path =
command ?? string.Empty, Arguments = parameters };
    }
    else if (string.Equals(actionType, "OpenUrl",
StringComparison.OrdinalIgnoreCase))
    {
        rule.Action = new
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction { Url = command
?? string.Empty };
    }
    else if (string.Equals(actionType, "FileWrite",
StringComparison.OrdinalIgnoreCase))
    {
        rule.Action = new
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction { FilePath =
command ?? string.Empty, Content = parameters ?? string.Empty };
    }
    else if (string.Equals(actionType, "Macro",
StringComparison.OrdinalIgnoreCase) && actionId != 0)
    {
        // Reconstruct macro by querying MacroActionItems and child Actions
        var macro = new FlexibleAutomationTool.Core.Actions.MacroAction();
        using (var cmdItems = _context.Connection.CreateCommand())
        {
            cmdItems.CommandText = "SELECT ChildActionId FROM
MacroActionItems WHERE MacroActionId=$mid ORDER BY OrderIndex";
            cmdItems.Parameters.AddWithValue("$mid", actionId);
            using var rdrItems = cmdItems.ExecuteReader();

```

```

while (rdrItems.Read())
{
    var childId = rdrItems.GetInt32(0);
    // load child action
    using var cmdChild = _context.Connection.CreateCommand();
    cmdChild.CommandText = "SELECT ActionType, ServiceType,
Command, Parameters FROM Actions WHERE Id=$id";
    cmdChild.Parameters.AddWithValue("$id", childId);
    using var rdrChild = cmdChild.ExecuteReader();
    if (rdrChild.Read())
    {
        var cType = rdrChild.IsDBNull(0) ? null : rdrChild.GetString(0);
        var cService = rdrChild.IsDBNull(1) ? null : rdrChild.GetString(1);
        var cCommand = rdrChild.IsDBNull(2) ? null :
rdrChild.GetString(2);
        var cParams = rdrChild.IsDBNull(3) ? null : rdrChild.GetString(3);

        if (string.Equals(cType, "MessageBox",
StringComparison.OrdinalIgnoreCase))
            macro.Actions.Add(new
FlexibleAutomationTool.Core.Actions.InternalActions.MessageBoxAction { Title =
cCommand ?? string.Empty, Message = cParams ?? string.Empty });
        else if (string.Equals(cType, "RunProgram",
StringComparison.OrdinalIgnoreCase))
            macro.Actions.Add(new
FlexibleAutomationTool.Core.Actions.InternalActions.RunProgramAction { Path =
cCommand ?? string.Empty, Arguments = cParams });
        else if (string.Equals(cType, "OpenUrl",
StringComparison.OrdinalIgnoreCase))

```



```

        macro.Actions.Add(new
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction { Url = cCommand
?? string.Empty });
        else if (string.Equals(cType, "FileWrite",
StringComparison.OrdinalIgnoreCase))
        macro.Actions.Add(new
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction { FilePath =
cCommand ?? string.Empty, Content = cParams ?? string.Empty });
        else
        macro.Actions.Add(new
FlexibleAutomationTool.Core.Actions.InternalActions.MessageBoxAction { Title =
string.Empty, Message = string.Empty });
    }
}
}

rule.Action = macro;
}
else
{
    // Default to MessageBoxAction placeholder to avoid nulls
    rule.Action = new
FlexibleAutomationTool.Core.Actions.InternalActions.MessageBoxAction { Title =
string.Empty, Message = string.Empty };
}

list.Add(rule);
}
}

```

```

    return list;
}

private long InsertTrigger(Rule rule, SqliteTransaction tx)
{
    using var cmdT = _context.Connection.CreateCommand();
    cmdT.Transaction = tx;
    cmdT.CommandText = "INSERT INTO Triggers (TriggerType, Schedule,
EventSource, Condition) VALUES ($tt,$sch,$es,$cond); SELECT last_insert_rowid();";
    if (rule.Trigger is FlexibleAutomationTool.Core.Triggers.TimeTrigger tt)
    {
        cmdT.Parameters.AddWithValueValue("$tt", "TimeTrigger");
        cmdT.Parameters.AddWithValueValue("$sch", $"{tt.Hour}:{tt.Minute}");
        cmdT.Parameters.AddWithValueValue("$es", (object)DBNull.Value);
        cmdT.Parameters.AddWithValueValue("$cond", (object)DBNull.Value);
    }
    else if (rule.Trigger is FlexibleAutomationTool.Core.Triggers.EventTrigger et)
    {
        cmdT.Parameters.AddWithValueValue("$tt", "EventTrigger");
        cmdT.Parameters.AddWithValueValue("$sch", (object)DBNull.Value);
        cmdT.Parameters.AddWithValueValue("$es", et.EventSource?.ToString() ??
(object)DBNull.Value);
        cmdT.Parameters.AddWithValueValue("$cond", et.Condition == null ?
(object)DBNull.Value : et.Condition.ToString());
    }
    else
    {

```

```

        cmdT.Parameters.AddWithValue("$tt", rule.Trigger?.GetType().Name ??
"Unknown");

        cmdT.Parameters.AddWithValue("$sch", (object)DBNull.Value);
        cmdT.Parameters.AddWithValue("$es", (object)DBNull.Value);
        cmdT.Parameters.AddWithValue("$cond", (object)DBNull.Value);
    }

    return (long)cmdT.ExecuteScalar();
}

private long InsertAction(Rule rule, SqliteTransaction tx)
{
    // Support MacroAction by inserting an action entry representing macro and child
actions
    if (rule.Action is FlexibleAutomationTool.Core.Actions.MacroAction mac)
    {
        // Insert placeholder Macro action record
        using var cmdA = _context.Connection.CreateCommand();
        cmdA.Transaction = tx;

        cmdA.CommandText = "INSERT INTO Actions (ActionType, ServiceType,
Command, Parameters) VALUES ($at,$st,$cm,$pm); SELECT last_insert_rowid();";
        cmdA.Parameters.AddWithValue("$at", "Macro");
        cmdA.Parameters.AddWithValue("$st", (object)DBNull.Value);
        cmdA.Parameters.AddWithValue("$cm", (object)DBNull.Value);
        cmdA.Parameters.AddWithValue("$pm", (object)DBNull.Value);
        var macroId = (long)cmdA.ExecuteScalar();

        // Insert child actions and MacroActionItems
        int order = 0;

```

```

foreach (var child in mac.Actions)
{
    // insert child action
    using var cmdChild = _context.Connection.CreateCommand();
    cmdChild.Transaction = tx;
    cmdChild.CommandText = "INSERT INTO Actions (ActionType,
ServiceType, Command, Parameters) VALUES ($at,$st,$cm,$pm); SELECT
last_insert_rowid()";

    if (child is
FlexibleAutomationTool.Core.Actions.InternalActions.MessageBoxAction cma)
    {
        cmdChild.Parameters.AddWithValueValue("$at", "MessageBox");
        cmdChild.Parameters.AddWithValueValue("$st", (object)DBNull.Value);
        cmdChild.Parameters.AddWithValueValue("$cm", cma.Title ?? string.Empty);
        cmdChild.Parameters.AddWithValueValue("$pm", cma.Message ??
string.Empty);
    }
    else if (child is
FlexibleAutomationTool.Core.Actions.InternalActions.RunProgramAction cra)
    {
        cmdChild.Parameters.AddWithValueValue("$at", "RunProgram");
        cmdChild.Parameters.AddWithValueValue("$st", (object)DBNull.Value);
        cmdChild.Parameters.AddWithValueValue("$cm", cra.Path ?? string.Empty);
        cmdChild.Parameters.AddWithValueValue("$pm", cra.Arguments ??
(object)DBNull.Value);
    }
    else if (child is
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction cua)

```

```

{
    cmdChild.Parameters.AddWithValue("$at", "OpenUrl");
    cmdChild.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdChild.Parameters.AddWithValue("$cm", cua.Url ?? string.Empty);
    cmdChild.Parameters.AddWithValue("$pm", (object)DBNull.Value);
}
else if (child is
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction cfa)
{
    cmdChild.Parameters.AddWithValue("$at", "FileWrite");
    cmdChild.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdChild.Parameters.AddWithValue("$cm", cfa.FilePath ?? string.Empty);
    cmdChild.Parameters.AddWithValue("$pm", cfa.Content ?? string.Empty);
}
else
{
    cmdChild.Parameters.AddWithValue("$at", child?.GetType().Name ??
"Unknown");
    cmdChild.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdChild.Parameters.AddWithValue("$cm", (object)DBNull.Value);
    cmdChild.Parameters.AddWithValue("$pm", (object)DBNull.Value);
}

var childId = (long)cmdChild.ExecuteScalar();

using var cmdItem = _context.Connection.CreateCommand();
cmdItem.Transaction = tx;
cmdItem.CommandText = "INSERT INTO MacroActionItems
(MacroActionId, ChildActionId, OrderIndex) VALUES ($mid,$cid,$ord)";

```

```

cmdItem.Parameters.AddWithValue("$mid", macroId);
cmdItem.Parameters.AddWithValue("$cid", childId);
cmdItem.Parameters.AddWithValue("$ord", order++);
cmdItem.ExecuteNonQuery();
}

```

```

return macroId;
}

```

```

using var cmdA2 = _context.Connection.CreateCommand();
cmdA2.Transaction = tx;
cmdA2.CommandText = "INSERT INTO Actions (ActionType, ServiceType,
Command, Parameters) VALUES ($at,$st,$cm,$pm); SELECT last_insert_rowid()";
if (rule.Action is FlexibleAutomationTool.Core.Actions.InternalActions.MessageBoxAction ma)
{
    cmdA2.Parameters.AddWithValue("$at", "MessageBox");
    cmdA2.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdA2.Parameters.AddWithValue("$cm", ma.Title ?? string.Empty);
    cmdA2.Parameters.AddWithValue("$pm", ma.Message ?? string.Empty);
}
else if (rule.Action is FlexibleAutomationTool.Core.Actions.InternalActions.RunProgramAction ra)
{
    cmdA2.Parameters.AddWithValue("$at", "RunProgram");
    cmdA2.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdA2.Parameters.AddWithValue("$cm", ra.Path ?? string.Empty);
    cmdA2.Parameters.AddWithValue("$pm", ra.Arguments ??
(object)DBNull.Value);
}

```

```

    }
    else if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction ua)
    {
        cmdA2.Parameters.AddWithValue("$at", "OpenUrl");
        cmdA2.Parameters.AddWithValue("$st", (object)DBNull.Value);
        cmdA2.Parameters.AddWithValue("$cm", ua.Url ?? string.Empty);
        cmdA2.Parameters.AddWithValue("$pm", (object)DBNull.Value);
    }
    else if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction fa)
    {
        cmdA2.Parameters.AddWithValue("$at", "FileWrite");
        cmdA2.Parameters.AddWithValue("$st", (object)DBNull.Value);
        cmdA2.Parameters.AddWithValue("$cm", fa.FilePath ?? string.Empty);
        cmdA2.Parameters.AddWithValue("$pm", fa.Content ?? string.Empty);
    }
    else
    {
        cmdA2.Parameters.AddWithValue("$at", rule.Action?.GetType().Name ??
"Unknown");
        cmdA2.Parameters.AddWithValue("$st", (object)DBNull.Value);
        cmdA2.Parameters.AddWithValue("$cm", (object)DBNull.Value);
        cmdA2.Parameters.AddWithValue("$pm", (object)DBNull.Value);
    }

    return (long)cmdA2.ExecuteScalar();
}

```

```

public void Add(Rule rule)
{
    lock (_lock)
    {
        using var tx = _context.Connection.BeginTransaction();

        var triggerId = InsertTrigger(rule, tx);
        var actionId = InsertAction(rule, tx);

        using var cmdR = _context.Connection.CreateCommand();
        cmdR.Transaction = tx;
        cmdR.CommandText = "INSERT INTO Rules (Name, Description, TriggerId,
ActionId, IsActive) VALUES ($name,$desc,$tid,$aid,$ia); SELECT last_insert_rowid();";
        cmdR.Parameters.AddWithValue("$name", rule.Name);
        cmdR.Parameters.AddWithValue("$desc", rule.Description ??
(object)DBNull.Value);
        cmdR.Parameters.AddWithValue("$tid", triggerId);
        cmdR.Parameters.AddWithValue("$aid", actionId);
        cmdR.Parameters.AddWithValue("$ia", rule.IsActive ? 1 : 0);

        var id = (long)cmdR.ExecuteScalar();
        rule.Id = (int)id;

        tx.Commit();
    }
}

public void Update(Rule rule)
{

```



```

lock (_lock)
{
    using var tx = _context.Connection.BeginTransaction();

    // Find triggerId and actionId for rule
    long triggerId = 0, actionId = 0;
    using (var cmdFind = _context.Connection.CreateCommand())
    {
        cmdFind.CommandText = "SELECT TriggerId, ActionId FROM Rules
WHERE Id=$id";

        cmdFind.Parameters.AddWithValue("$id", rule.Id);
        using var rdr = cmdFind.ExecuteReader();
        if (rdr.Read())
        {
            triggerId = rdr.IsDBNull(0) ? 0 : rdr.GetInt64(0);
            actionId = rdr.IsDBNull(1) ? 0 : rdr.GetInt64(1);
        }
    }

    if (triggerId != 0)
    {
        using var cmdT = _context.Connection.CreateCommand();
        cmdT.Transaction = tx;
        cmdT.CommandText = "UPDATE Triggers SET TriggerType=$tt,
Schedule=$sch, EventSource=$es, Condition=$cond WHERE Id=$id";
        if (rule.Trigger is FlexibleAutomationTool.Core.Triggers.TimeTrigger tt)
        {
            cmdT.Parameters.AddWithValue("$tt", "TimeTrigger");
            cmdT.Parameters.AddWithValue("$sch", $"{tt.Hour}:{tt.Minute}");

```

```

        cmdT.Parameters.AddWithValue("$es", (object)DBNull.Value);
        cmdT.Parameters.AddWithValue("$cond", (object)DBNull.Value);
    }
    else if (rule.Trigger is FlexibleAutomationTool.Core.Triggers.EventTrigger et)
    {
        cmdT.Parameters.AddWithValue("$tt", "EventTrigger");
        cmdT.Parameters.AddWithValue("$sch", (object)DBNull.Value);
        cmdT.Parameters.AddWithValue("$es", et.EventSource?.ToString() ??
(object)DBNull.Value);
        cmdT.Parameters.AddWithValue("$cond", et.Condition == null ?
(object)DBNull.Value : et.Condition.ToString());
    }
    else
    {
        cmdT.Parameters.AddWithValue("$tt", rule.Trigger?.GetType().Name ??
"Unknown");
        cmdT.Parameters.AddWithValue("$sch", (object)DBNull.Value);
        cmdT.Parameters.AddWithValue("$es", (object)DBNull.Value);
        cmdT.Parameters.AddWithValue("$cond", (object)DBNull.Value);
    }
    cmdT.Parameters.AddWithValue("$id", triggerId);
    cmdT.ExecuteNonQuery();
}

if (actionId != 0)
{
    // For simplicity, update Actions table for non-macro actions. Macro actions are
    not fully updated in-place here.
    using var cmdA = _context.Connection.CreateCommand();

```

```

cmdA.Transaction = tx;
cmdA.CommandText = "UPDATE Actions SET ActionType=$at,
ServiceType=$st, Command=$cm, Parameters=$pm WHERE Id=$id";
if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.MessageBoxAction ma)
{
    cmdA.Parameters.AddWithValue("$at", "MessageBox");
    cmdA.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdA.Parameters.AddWithValue("$cm", ma.Title ?? string.Empty);
    cmdA.Parameters.AddWithValue("$pm", ma.Message ?? string.Empty);
}
else if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.RunProgramAction ra)
{
    cmdA.Parameters.AddWithValue("$at", "RunProgram");
    cmdA.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdA.Parameters.AddWithValue("$cm", ra.Path ?? string.Empty);
    cmdA.Parameters.AddWithValue("$pm", ra.Arguments ??
(object)DBNull.Value);
}
else if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction ua)
{
    cmdA.Parameters.AddWithValue("$at", "OpenUrl");
    cmdA.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdA.Parameters.AddWithValue("$cm", ua.Url ?? string.Empty);
    cmdA.Parameters.AddWithValue("$pm", (object)DBNull.Value);
}

```

```

else if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction fa)
{
    cmdA.Parameters.AddWithValue("$at", "FileWrite");
    cmdA.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdA.Parameters.AddWithValue("$cm", fa.FilePath ?? string.Empty);
    cmdA.Parameters.AddWithValue("$pm", fa.Content ?? string.Empty);
}
else
{
    cmdA.Parameters.AddWithValue("$at", rule.Action?.GetType().Name ??
"Unknown");
    cmdA.Parameters.AddWithValue("$st", (object)DBNull.Value);
    cmdA.Parameters.AddWithValue("$cm", (object)DBNull.Value);
    cmdA.Parameters.AddWithValue("$pm", (object)DBNull.Value);
}
cmdA.Parameters.AddWithValue("$id", actionId);
cmdA.ExecuteNonQuery();
}

using var cmdR = _context.Connection.CreateCommand();
cmdR.Transaction = tx;
cmdR.CommandText = "UPDATE Rules SET Name=$name, Description=$desc,
IsActive=$ia WHERE Id=$id";
cmdR.Parameters.AddWithValue("$name", rule.Name);
cmdR.Parameters.AddWithValue("$desc", rule.Description ??
(object)DBNull.Value);
cmdR.Parameters.AddWithValue("$ia", rule.IsActive ? 1 : 0);
cmdR.Parameters.AddWithValue("$id", rule.Id);

```

```
cmdR.ExecuteNonQuery();
```

```
tx.Commit();
```

```
}
```

```
}
```

```
public void Delete(int id)
```

```
{
```

```
    using var cmd = _context.Connection.CreateCommand();
```

```
    cmd.CommandText = "DELETE FROM Rules WHERE Id=$id";
```

```
    cmd.Parameters.AddWithValue("$id", id);
```

```
    cmd.ExecuteNonQuery();
```

```
}
```

```
public void Dispose()
```

```
{
```

```
    _context.Dispose();
```

```
}
```

```
}
```

```
}
```

```
using FlexibleAutomationTool.Core.Models;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
namespace FlexibleAutomationTool.Core.Facades
```

```
{
```

```
    public interface IAutomationFacade
```

```
{
```

```

event Action<Rule>? RuleTriggered;
event Action<Rule>? RuleExecuted;
event Action<Rule, Exception>? RuleFailed;

```

```

void Start();
void Stop();

```

```

IEnumerable<LogEntry> GetHistory();

```

```

void CreateMacroRule(string name, string macroText);
void CreateRule(Rule rule);

```

```

}

```

```

}

```

```

using FlexibleAutomationTool.Core.Interfaces;

```

```

namespace FlexibleAutomationTool.Core.Factories

```

```

{

```

```

    public interface IPlatformFactory

```

```

    {

```

```

        IMessageBoxService CreateMessageBoxService();

```

```

    }

```

```

}

```

```

namespace FlexibleAutomationTool.Core.Interfaces

```

```

{

```

```

    public interface IMessageBoxService

```

```

    {
        void Show(string message, string? title = null);
    }
}

```

```

using System.Collections.Generic;
using FlexibleAutomationTool.Core.Actions;

```

```

namespace FlexibleAutomationTool.Core.Interpreter
{
    public interface IInterpreter
    {
        IEnumerable<ActionBase> ParseMacro(string macroText);
    }
}

```

```

using System;
using System.Collections.Generic;
using FlexibleAutomationTool.Core.Actions;
using FlexibleAutomationTool.Core.Actions.InternalActions;

```

```

namespace FlexibleAutomationTool.Core.Interpreter
{
    public class SimpleMacroInterpreter : IInterpreter
    {
        public IEnumerable<ActionBase> ParseMacro(string macroText)
        {

```

```

var list = new List<ActionBase>();
if (string.IsNullOrEmpty(macroText))
    return list;

var lines = macroText.Split(new[] { '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries);
foreach (var raw in lines)
{
    var line = raw.Trim();
    if (line.StartsWith("RUN ", StringComparison.OrdinalIgnoreCase))
    {
        var rest = line.Substring(4).Trim();
        var parts = rest.Split(' ', 2);
        var path = parts[0];
        var args = parts.Length > 1 ? parts[1] : null;
        list.Add(new RunProgramAction { Path = path, Arguments = args });
    }
    else if (line.StartsWith("MSG ", StringComparison.OrdinalIgnoreCase))
    {
        var msg = line.Substring(4).Trim();
        // Title not supported in shorthand
        list.Add(new MessageBoxAction { Message = msg, Title = "" });
    }
    // add other simple commands as needed
}

return list;
}

```



```

    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace FlexibleAutomationTool.Core.Models

```

```

{
    public class LogEntry
    {
        public int Id { get; set; }
        public DateTime Timestamp { get; set; } = DateTime.Now;
        public string LoggedBy { get; set; } = string.Empty;
        public string RuleName { get; set; } = string.Empty;
        public string Message { get; set; } = string.Empty;
        public string? Error { get; set; }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace FlexibleAutomationTool.Core.Models
{
    public class Rule
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        public string? Description { get; set; }
        public FlexibleAutomationTool.Core.Triggers.Trigger Trigger { get; set; } = null!;
        public FlexibleAutomationTool.Core.Actions.ActionBase Action { get; set; } = null!;
        public bool IsActive { get; set; } = true;

        public bool CheckTrigger()
        {
            return Trigger?.ShouldExecute() ?? false;
        }

        public void Execute()
        {
            Action.Execute();
        }

        public override string ToString()
        {
            return string.IsNullOrEmpty(Name) ? base.ToString() : Name;
        }
    }
}

```

```

using System.Collections.Generic;
using FlexibleAutomationTool.Core.Models;

namespace FlexibleAutomationTool.Core.Repositories
{
    public interface IExecutionHistoryRepository
    {
        void Add(int ruleId, System.DateTime executedAt, string status, string message);
        IEnumerable<LogEntry> GetAll();
        IEnumerable<LogEntry> GetByRuleId(int ruleId);
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FlexibleAutomationTool.Core.Models;

namespace FlexibleAutomationTool.Core.Repositories
{
    public class InMemoryRuleRepository : IRuleRepository
    {
        private readonly List<Rule> _rules = new();
        private int _nextId = 1;
    }
}

```

```
public IEnumerable<Rule> GetAll() => _rules.ToList();
```

```
public void Add(Rule rule)
```

```
{
    rule.Id = _nextId++;
    _rules.Add(rule);
}
```

```
public void Update(Rule rule)
```

```
{
    var idx = _rules.FindIndex(r => r.Id == rule.Id);
    if (idx >= 0) _rules[idx] = rule;
}
```

```
public void Delete(int id)
```

```
{
    var r = _rules.FirstOrDefault(x => x.Id == id);
    if (r != null) _rules.Remove(r);
}
}
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
using FlexibleAutomationTool.Core.Models;

namespace FlexibleAutomationTool.Core.Repositories
{
    public interface IRuleRepository
    {
        IEnumerable<Rule> GetAll();
        void Add(Rule rule);
        void Update(Rule rule);
        void Delete(int id);
    }
}
```

```
using FlexibleAutomationTool.Core.Actions;
using FlexibleAutomationTool.Core.Actions.InternalActions;
using FlexibleAutomationTool.Core.Facades;
using FlexibleAutomationTool.Core.Interpreter;
using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
```

```

namespace FlexibleAutomationTool.Core.Services
{
    public class AutomationEngine : IAutomationFacade
    {
        private readonly IRuleRepository _repo;
        private readonly Scheduler _scheduler;
        private readonly ServiceManager _serviceManager;
        private readonly Logger _logger;
        private readonly IInterpreter _interpreter;
        private readonly ICommandDispatcher _dispatcher;
        private readonly FlexibleAutomationTool.Core.Factories.IPlatformFactory _factory;
        private readonly IExecutionHistoryRepository? _historyRepo;

        public event Action<Rule>? RuleTriggered;
        public event Action<Rule>? RuleExecuted;
        public event Action<Rule, Exception>? RuleFailed;

        public AutomationEngine(IRuleRepository repo, Scheduler scheduler,
            ServiceManager svcManager, Logger logger, IInterpreter interpreter,
            ICommandDispatcher dispatcher,
            FlexibleAutomationTool.Core.Factories.IPlatformFactory factory,
            IExecutionHistoryRepository? historyRepo = null)
        {
            _repo = repo;
            _scheduler = scheduler;
            _serviceManager = svcManager;
            _logger = logger;
            _interpreter = interpreter;
            _dispatcher = dispatcher;
        }
    }
}

```

```

    _factory = factory;
    _historyRepo = historyRepo;

    _scheduler.RuleReady += OnRuleReady;
}

public void CreateRule(Rule rule)
{
    _repo.Add(rule);
    _logger.Log(rule.Name, "Created");
    _historyRepo?.Add(rule.Id, DateTime.UtcNow, "Created", "Rule created via UI");
}

public void CreateMacroRule(string name, string macroText)
{
    var actions = _interpreter.ParseMacro(macroText);
    var actionList = new List<ActionBase>(actions);

    // Inject platform services into actions that require them
    var msgService = _factory.CreateMessageBoxService();
    foreach (var a in actionList)
    {
        if (a is MessageBoxAction mba)
        {
            mba.MessageBoxService = msgService;
        }
    }
}

```

```

        var macro = new MacroAction { Actions = actionList };

        var rule = new Rule { Name = name, Action = macro, Trigger = new
Triggers.EventTrigger() };

        _repo.Add(rule);

        _logger.Log(name, "Created macro rule");

        _historyRepo?.Add(rule.Id, DateTime.UtcNow, "Created", "Macro rule created");
    }

    public void Start()
    {
        _scheduler.Start();

        _logger.Log("Automation Engine", "Started");

        _historyRepo?.Add(0, DateTime.UtcNow, "Started", "Automation Engine
started");
    }

    public void Stop()
    {
        _scheduler.Stop();

        _logger.Log("Automation Engine", "Stopped");

        _historyRepo?.Add(0, DateTime.UtcNow, "Stopped", "Automation Engine
stopped");
    }

    private void OnRuleReady(Rule rule)
    {
        try
        {
            RuleTriggered?.Invoke(rule);

            _logger.Log(rule.Name, "Rule triggered");
        }
    }

```



```

        _historyRepo?.Add(rule.Id, DateTime.UtcNow, "Triggered", "Rule triggered by scheduler");

```

```

        // Inject platform services into any actions in the rule before execution
        InjectPlatformServicesIntoAction(rule.Action);

```

```

        // Use dispatcher to execute the rule
        _dispatcher.Dispatch(rule);

```

```

        RuleExecuted?.Invoke(rule);
        _logger.Log(rule.Name, "Rule executed");
        _historyRepo?.Add(rule.Id, DateTime.UtcNow, "Executed", "Rule executed successfully");

```

```

    }

```

```

    catch (Exception ex)

```

```

    {

```

```

        RuleFailed?.Invoke(rule, ex);

```

```

        // Log full exception details so stack trace and inner exceptions are available in history

```

```

        _logger.Log(rule.Name, $"Error executing rule: {ex}");

```

```

        _historyRepo?.Add(rule.Id, DateTime.UtcNow, "Failed", ex.ToString());

```

```

    }

```

```

}

```

```

private void InjectPlatformServicesIntoAction(Actions.ActionBase action)

```

```

{

```

```

    if (action == null) return;

```

```

    // If action is a MessageBoxAction, set the service

```

```

if (action is MessageBoxAction mba)
{
    // Prefer service from factory
    var svc = _factory.CreateMessageBoxService();
    if (svc != null)
        mba.MessageBoxService = svc;
}

// If action is MacroAction, recurse into children
if (action is MacroAction mac)
{
    foreach (var child in mac.Actions)
    {
        InjectPlatformServicesIntoAction(child);
    }
}

public IEnumerable<LogEntry> GetHistory() => _logger.GetAll();
}
}

using FlexibleAutomationTool.Core.Models;
using System;

namespace FlexibleAutomationTool.Core.Services
{
    public class CommandDispatcher : ICommandDispatcher

```

```

{
    private readonly Logger _logger;

    public CommandDispatcher(Logger logger)
    {
        _logger = logger;
    }

    public void Dispatch(Rule rule)
    {
        try
        {
            rule.Execute();
            _logger.Log(rule.Name, "executed via dispatcher");
        }
        catch (Exception ex)
        {
            _logger.Log(rule.Name, $"dispatch failed: {ex}");
            throw;
        }
    }
}

```

```
using FlexibleAutomationTool.Core.Models;
```

```
namespace FlexibleAutomationTool.Core.Services
```

```
{
```

```

public interface ICommandDispatcher
{
    void Dispatch(Rule rule);
}

```

```

using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Repositories;
using System.Collections.Generic;

```

```

namespace FlexibleAutomationTool.Core.Services
{
    public interface ITriggerStrategy
    {
        IEnumerable<Rule> GetReadyRules(IRuleRepository repository);
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Data;
using FlexibleAutomationTool.Core.Repositories;

namespace FlexibleAutomationTool.Core.Services

```

```

{
    public class Logger
    {
        private readonly List<LogEntry> _entries = new();
        private readonly object _lock = new();
        public event Action<LogEntry>? LogAdded;

        private readonly IExecutionHistoryRepository? _historyRepo;

        public Logger(IExecutionHistoryRepository? historyRepo = null)
        {
            _historyRepo = historyRepo;
        }

        public void Log(string loggedBy, string message)
        {
            var entry = new LogEntry { LoggedBy = loggedBy, Message = message };
            lock (_lock)
            {
                _entries.Add(entry);
            }

            try
            {
                LogAdded?.Invoke(entry);
            }
            catch { }
        }
    }
}

```

```

        _historyRepo?.Add(0, DateTime.UtcNow, loggedBy, message);
    }

```

```

public IEnumerable<LogEntry> GetAll()
{
    lock (_lock)
    {
        return _entries.ToArray();
    }
}

```

```

using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Repositories;
using System.Collections.Generic;

```

```

namespace FlexibleAutomationTool.Core.Services
{
    public class PollingTriggerStrategy : ITriggerStrategy
    {
        public IEnumerable<Rule> GetReadyRules(IRuleRepository repository)
        {
            foreach (var r in repository.GetAll())
            {
                if (r.CheckTrigger())
                    yield return r;
            }
        }
    }
}

```

```

    }
}
}

using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace FlexibleAutomationTool.Core.Services
{
    public class Scheduler
    {
        private Timer? _timer;
        private readonly IRuleRepository _repository;
        private readonly ITriggerStrategy _strategy;

        // Подія для Engine
        public event Action<Rule>? RuleReady;

        // Prevent overlapping invocations
        private int _isRunning = 0;

        // Lock to make Start/Stop thread-safe and avoid creating multiple timers

```

```
private readonly object _startLock = new();
```

```
public Scheduler(IRuleRepository repository, ITriggerStrategy strategy)
```

```
{
    _repository = repository;
    _strategy = strategy;
}
```

```
public void Start()
```

```
{
    // Make start thread-safe: only one thread may create the timer
    lock (_startLock)
    {
        if (_timer != null)
            return;

```

```
        _timer = new Timer(CheckAllRules, null, TimeSpan.Zero,
TimeSpan.FromSeconds(30));
    }
}
```

```
public void Stop()
```

```
{
    // Stop can run concurrently; use Interlocked.Exchange to swap out the timer
reference
    var t = Interlocked.Exchange(ref _timer, null);
    t?.Dispose();
}
```



```

private void CheckAllRules(object? state)
{
    // Prevent reentrant execution if previous run still in progress
    if (Interlocked.Exchange(ref _isRunning, 1) == 1)
        return;

    try
    {
        foreach (var r in _strategy.GetReadyRules(_repository))
        {
            RuleReady?.Invoke(r);
        }
    }
    finally
    {
        Interlocked.Exchange(ref _isRunning, 0);
    }
}

using System;
using System.Collections.Concurrent;

namespace FlexibleAutomationTool.Core.Services
{

```

```

public class ServiceManager
{
    private readonly ConcurrentDictionary<string, object> _services = new();

    public void Register<T>(string name, T svc) => _services[name] = svc!;
    public T? Get<T>(string name) => _services.TryGetValue(name, out var s) && s is
T t ? t : default;
}
}

```

```

using System;

```

```

namespace FlexibleAutomationTool.Core.Triggers

```

```

{
    public class EventTrigger : Trigger
    {
        private volatile bool _eventRaised;
        private object? _lastEventData;
        private readonly object _lock = new();

        // Optional name for the trigger
        public string Name { get; set; } = string.Empty;

        // EventSource can be any object (publisher) or an identifier for the source
        public object? EventSource { get; }

        // Condition receives the last event data and returns true if the trigger should fire.
        // If null, any raised event will cause execution.
    }
}

```

```
public Func<object?, bool>? Condition { get; }
```

```
public EventTrigger(object? eventSource = null, Func<object?, bool>? condition =
null)
{
    EventSource = eventSource;
    Condition = condition;
}
```

```
// External code calls this to notify the trigger of an event occurrence.
```

```
public void RaiseEvent(object? eventData = null)
{
    lock (_lock)
    {
        _lastEventData = eventData;
        _eventRaised = true;
    }
}
```

```
public override bool ShouldExecute()
{
    if (!_eventRaised)
        return false;

    object? data;
    lock (_lock)
    {
        data = _lastEventData;
    }
}
```

```

        _eventRaised = false;
        _lastEventData = null;
    }

```

```

    try
    {
        return Condition == null || Condition(data);
    }
    catch
    {
        // If condition fails, do not execute the action.
        return false;
    }
}

```

```

    public override bool Validate() => true;
}
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace FlexibleAutomationTool.Core.Triggers
{
    public class TimeTrigger : Trigger

```

```

{
    // run at a specific hour/minute
    public int Hour { get; set; }
    public int Minute { get; set; }

    private DateTime _lastRun = DateTime.MinValue;

    public override bool ShouldExecute()
    {
        var now = DateTime.Now;
        if (now.Hour == Hour && now.Minute == Minute)
        {
            if (_lastRun.Date != now.Date || _lastRun.Hour != now.Hour || _lastRun.Minute
!= now.Minute)
            {
                _lastRun = now;
                return true;
            }
        }
        return false;
    }

    public override bool Validate() => Hour >= 0 && Hour < 24 && Minute >= 0 &&
Minute < 60;
}
}

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FlexibleAutomationTool.Core.Triggers
{
    public abstract class Trigger
    {
        public int Id { get; set; }

        public abstract bool ShouldExecute();

        public abstract bool Validate();
    }
}

```

```

using FlexibleAutomationTool.Core.Interfaces;
using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Services;

```

```

namespace FlexibleAutomationTool.UI.Services
{
    /// <summary>
    /// Handles AutomationEngine events and provides UI-friendly callbacks.
    /// Encapsulates translating engine events into UI notifications and selects relevant log
    entries.
    /// </summary>
    public class AutomationEventHandler : IDisposable
    {
        private readonly AutomationEngine _engine;
    }
}

```

```

private readonly Logger _logger;
private readonly IMessageBoxService _messageBoxService;
private readonly SynchronizationContext? _uiContext;

// UI-friendly events invoked on the UI thread
public event Action<string>? StatusUpdated;
public event Action<LogEntry>? LogEntryAdded;

// If true, a message box will be shown on rule failures. Default: true.
public bool ShowMessageBoxOnFailure { get; set; } = true;

public AutomationEventHandler(
    AutomationEngine engine,
    Logger logger,
    IMessageBoxService messageBoxService,
    SynchronizationContext? uiContext = null)
{
    _engine = engine ?? throw new ArgumentNullException(nameof(engine));
    _logger = logger ?? throw new ArgumentNullException(nameof(logger));
    _messageBoxService = messageBoxService ?? throw new
ArgumentNullException(nameof(messageBoxService));
    _uiContext = uiContext ?? SynchronizationContext.Current;

    // Subscribe to engine events
    _engine.RuleTriggered += OnRuleTriggered;
    _engine.RuleExecuted += OnRuleExecuted;
    _engine.RuleFailed += OnRuleFailed;
}

```

```

private void OnRuleTriggered(Rule rule)
{
    if (rule == null) return;

    var message = $"Rule triggered: {rule.Name}";

    PostToUI(() =>
    {
        StatusUpdated?.Invoke(message);

        var latestLog = GetLatestLogForRule(rule);
        if (latestLog != null)
            LogEntryAdded?.Invoke(latestLog);
    });
}

```

```

private void OnRuleExecuted(Rule rule)
{
    if (rule == null) return;

    var message = $"Rule executed: {rule.Name}";

    PostToUI(() =>
    {
        StatusUpdated?.Invoke(message);

        var latestLog = GetLatestLogForRule(rule);

```



```

        if (latestLog != null)
            LogEntryAdded?.Invoke(latestLog);
    });
}

```

```

private void OnRuleFailed(Rule rule, Exception ex)

```

```

{
    if (rule == null) return;

```

```

    var message = $"Rule failed: {rule.Name}";

```

```

    PostToUI(() =>

```

```

    {
        StatusUpdated?.Invoke(message);

```

```

        if (ShowMessageBoxOnFailure)

```

```

        {
            // Show a friendly message but include exception details if available

```

```

            var title = "Rule Execution Error";

```

```

            var body = ex == null ? $"Rule '{rule.Name}' failed." : $"Rule '{rule.Name}'
failed:\n{ex.Message}";

```

```

            _messageBoxService.Show(body, title);

```

```

        }

```

```

        var latestLog = GetLatestLogForRule(rule) ?? _logger.GetAll().LastOrDefault();

```

```

        if (latestLog != null)

```

```

            LogEntryAdded?.Invoke(latestLog);

```

```

    });

```

```
}
```

```
/// <summary>
```

```
/// Attempts to find the most relevant log entry for the provided rule.
```

```
/// Falls back to the last global entry if none found for the rule.
```

```
/// </summary>
```

```
private LogEntry? GetLatestLogForRule(Rule rule)
```

```
{
```

```
    try
```

```
    {
```

```
        // Prefer logs where LoggedBy matches the rule name
```

```
        var byName = _logger.GetAll()
```

```
            .Where(e => string.Equals(e.LoggedBy, rule.Name,
StringComparison.OrdinalIgnoreCase))
```

```
            .LastOrDefault();
```

```
        if (byName != null) return byName;
```

```
        // If no direct match, try to find a log that mentions the rule name in the message
```

```
        var byMessage = _logger.GetAll()
```

```
            .Where(e => e.Message != null && e.Message.Contains(rule.Name,
StringComparison.OrdinalIgnoreCase))
```

```
            .LastOrDefault();
```

```
        return byMessage;
```

```
    }
```

```
    catch
```

```
    {
```

```

        // If logger enumeration fails for any reason, return nothing
        return null;
    }
}

```

```

/// <summary>

```

```

/// Posts an action to UI synchronization context when available.

```

```

/// Falls back to using Application.OpenForms[0].BeginInvoke if no
SynchronizationContext was captured.

```

```

/// </summary>

```

```

private void PostToUI(Action action)

```

```

{
    if (action == null) return;

    if (_uiContext != null)
    {
        _uiContext.Post(_ => action(), null);
        return;
    }

```

```

// Fallback: try to use the first open form to marshal to UI thread

```

```

try
{
    if (Application.OpenForms != null && Application.OpenForms.Count > 0)
    {
        var form = Application.OpenForms[0];
        if (form != null)
        {

```

```

        if (form.InvokeRequired)
        {
            form.BeginInvoke((Action)(() => action()));
            return;
        }
        else
        {
            action();
            return;
        }
    }
}

```

```

catch

```

```

{
    // ignore and fallback to direct call
}

```

// Last resort: execute inline (may still cause cross-thread issues if UI access occurs)

```

    action();
}

```

```

public void Dispose()

```

```

{
    _engine.RuleTriggered -= OnRuleTriggered;
    _engine.RuleExecuted -= OnRuleExecuted;
    _engine.RuleFailed -= OnRuleFailed;
}

```

```

    }
}

```

```
using FlexibleAutomationTool.Core.Interfaces;
```

```
namespace FlexibleAutomationTool.UI.Services
```

```

{
    public class WinFormsMessageBoxService : IMessageBoxService
    {
        public void Show(string message, string? title = null)
        {
            MessageBox.Show(message, title ?? "Info");
        }
    }
}

```

```
using FlexibleAutomationTool.Core.Factories;
```

```
using FlexibleAutomationTool.Core.Interfaces;
```

```
using FlexibleAutomationTool.Core.Services;
```

```
namespace FlexibleAutomationTool.UI.Services
```

```

{
    public class WinFormsPlatformFactory : IPlatformFactory
    {
        private readonly IMessageBoxService _msgService;
        private readonly ServiceManager _svcManager;
    }
}

```

```

    public WinFormsPlatformFactory(IMessageBoxService msgService, ServiceManager
svcManager)
    {
        _msgService = msgService;
        _svcManager = svcManager;

        // register in service manager for UI discovery
        _svcManager.Register(nameof(IMessageBoxService), _msgService);
    }

    public IMessageBoxService CreateMessageBoxService() => _msgService;
}

using System;
using System.Windows.Forms;
using System.Drawing;

namespace FlexibleAutomationTool.UI
{
    partial class CreateRuleForm
    {
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()

```

```
{  
    pnlMain = new Panel();  
    lblName = new Label();  
    txtName = new TextBox();  
    lblDescription = new Label();  
    txtDescription = new TextBox();  
    lblTriggerType = new Label();  
    cmbTriggerType = new ComboBox();  
    lblHour = new Label();  
    nudHour = new NumericUpDown();  
    lblMinute = new Label();  
    nudMinute = new NumericUpDown();  
    lblEventSource = new Label();  
    txtEventSource = new TextBox();  
    lblCondition = new Label();  
    txtEventCondition = new TextBox();  
    sep = new Label();  
    lblActionType = new Label();  
    cmbActionType = new ComboBox();  
    lblMsgTitle = new Label();  
    txtMsgTitle = new TextBox();  
    lblMsgBody = new Label();  
    txtMsgBody = new TextBox();  
    lblProgramPath = new Label();  
    txtRunPath = new TextBox();  
    lblArguments = new Label();  
    txtRunArgs = new TextBox();  
    lblUrl = new Label();
```

```
txtUrl = new TextBox();
lblFilePath = new Label();
txtFilePath = new TextBox();
lblContent = new Label();
txtFileContent = new TextBox();
lblMacro = new Label();
txtMacroDef = new TextBox();
btnOk = new Button();
btnCancel = new Button();
pnlMain.SuspendLayout();
((System.ComponentModel.ISupportInitialize)nudHour).BeginInit();
((System.ComponentModel.ISupportInitialize)nudMinute).BeginInit();
SuspendLayout();
//
// pnlMain
//
pnlMain.AutoScroll = true;
pnlMain.Controls.Add(lblName);
pnlMain.Controls.Add(txtName);
pnlMain.Controls.Add(lblDescription);
pnlMain.Controls.Add(txtDescription);
pnlMain.Controls.Add(lblTriggerType);
pnlMain.Controls.Add(cmbTriggerType);
pnlMain.Controls.Add(lblHour);
pnlMain.Controls.Add(nudHour);
pnlMain.Controls.Add(lblMinute);
pnlMain.Controls.Add(nudMinute);
pnlMain.Controls.Add(lblEventSource);
```



```
pnlMain.Controls.Add(txtEventSource);
pnlMain.Controls.Add(lblCondition);
pnlMain.Controls.Add(txtEventCondition);
pnlMain.Controls.Add(sep);
pnlMain.Controls.Add(lblActionType);
pnlMain.Controls.Add(cmbActionType);
pnlMain.Controls.Add(lblMsgTitle);
pnlMain.Controls.Add(txtMsgTitle);
pnlMain.Controls.Add(lblMsgBody);
pnlMain.Controls.Add(txtMsgBody);
pnlMain.Controls.Add(lblProgramPath);
pnlMain.Controls.Add(txtRunPath);
pnlMain.Controls.Add(lblArguments);
pnlMain.Controls.Add(txtRunArgs);
pnlMain.Controls.Add(lblUrl);
pnlMain.Controls.Add(txtUrl);
pnlMain.Controls.Add(lblFilePath);
pnlMain.Controls.Add(txtFilePath);
pnlMain.Controls.Add(lblContent);
pnlMain.Controls.Add(txtFileContent);
pnlMain.Controls.Add(lblMacro);
pnlMain.Controls.Add(txtMacroDef);
pnlMain.Controls.Add(btnOk);
pnlMain.Controls.Add(btnCancel);
pnlMain.Dock = DockStyle.Fill;
pnlMain.Location = new Point(0, 0);
pnlMain.Name = "pnlMain";
pnlMain.Size = new Size(649, 713);
```

```
pnlMain.TabIndex = 0;
//
// lblName
//
lblName.Location = new Point(10, 13);
lblName.Name = "lblName";
lblName.Size = new Size(120, 20);
lblName.TabIndex = 0;
lblName.Text = "Name:";
//
// txtName
//
txtName.Location = new Point(140, 10);
txtName.Name = "txtName";
txtName.Size = new Size(480, 23);
txtName.TabIndex = 1;
//
// lblDescription
//
lblDescription.Location = new Point(10, 43);
lblDescription.Name = "lblDescription";
lblDescription.Size = new Size(120, 20);
lblDescription.TabIndex = 2;
lblDescription.Text = "Description:";
//
// txtDescription
//
txtDescription.Location = new Point(140, 40);
```

```

txtDescription.Name = "txtDescription";
txtDescription.Size = new Size(480, 23);
txtDescription.TabIndex = 3;
//
// lblTriggerType
//
lblTriggerType.Location = new Point(10, 73);
lblTriggerType.Name = "lblTriggerType";
lblTriggerType.Size = new Size(120, 20);
lblTriggerType.TabIndex = 4;
lblTriggerType.Text = "Trigger Type:";
//
// cmbTriggerType
//
cmbTriggerType.DropDownStyle = ComboBoxStyle.DropDownList;
cmbTriggerType.Items.AddRange(new object[] { "Time", "Event" });
cmbTriggerType.Location = new Point(140, 70);
cmbTriggerType.Name = "cmbTriggerType";
cmbTriggerType.Size = new Size(220, 23);
cmbTriggerType.TabIndex = 5;
cmbTriggerType.SelectedIndexChanged +=
CmbTriggerType_SelectedIndexChanged;
//
// lblHour
//
lblHour.Location = new Point(10, 103);
lblHour.Name = "lblHour";
lblHour.Size = new Size(120, 20);

```

```
lblHour.TabIndex = 6;
lblHour.Text = "Hour:";
//
// nudHour
//
nudHour.Location = new Point(140, 100);
nudHour.Maximum = new decimal(new int[] { 23, 0, 0, 0 });
nudHour.Name = "nudHour";
nudHour.Size = new Size(60, 23);
nudHour.TabIndex = 7;
//
// lblMinute
//
lblMinute.Location = new Point(220, 103);
lblMinute.Name = "lblMinute";
lblMinute.Size = new Size(60, 20);
lblMinute.TabIndex = 8;
lblMinute.Text = "Minute:";
//
// nudMinute
//
nudMinute.Location = new Point(281, 100);
nudMinute.Maximum = new decimal(new int[] { 59, 0, 0, 0 });
nudMinute.Name = "nudMinute";
nudMinute.Size = new Size(60, 23);
nudMinute.TabIndex = 9;
//
// lblEventSource
```

```
//  
lblEventSource.Location = new Point(10, 133);  
lblEventSource.Name = "lblEventSource";  
lblEventSource.Size = new Size(120, 20);  
lblEventSource.TabIndex = 10;  
lblEventSource.Text = "Event Source:";  
  
//  
// txtEventSource  
//  
txtEventSource.Location = new Point(140, 130);  
txtEventSource.Name = "txtEventSource";  
txtEventSource.Size = new Size(480, 23);  
txtEventSource.TabIndex = 11;  
  
//  
// lblCondition  
//  
lblCondition.Location = new Point(10, 163);  
lblCondition.Name = "lblCondition";  
lblCondition.Size = new Size(120, 20);  
lblCondition.TabIndex = 12;  
lblCondition.Text = "Condition:";  
  
//  
// txtEventCondition  
//  
txtEventCondition.Location = new Point(140, 160);  
txtEventCondition.Name = "txtEventCondition";  
txtEventCondition.Size = new Size(480, 23);  
txtEventCondition.TabIndex = 13;
```

```

//
// sep
//
sep.BorderStyle = BorderStyle.Fixed3D;
sep.Location = new Point(10, 190);
sep.Name = "sep";
sep.Size = new Size(600, 2);
sep.TabIndex = 14;
//
// lblActionType
//
lblActionType.Location = new Point(10, 205);
lblActionType.Name = "lblActionType";
lblActionType.Size = new Size(120, 20);
lblActionType.TabIndex = 15;
lblActionType.Text = "Action Type:";
//
// cmbActionType
//
cmbActionType.DropDownStyle = ComboBoxStyle.DropDownList;
cmbActionType.Items.AddRange(new object[] { "Message", "RunProgram",
"OpenUrl", "FileWrite", "Macro" });
cmbActionType.Location = new Point(140, 202);
cmbActionType.Name = "cmbActionType";
cmbActionType.Size = new Size(220, 23);
cmbActionType.TabIndex = 16;
cmbActionType.SelectedIndexChanged +=
CmbActionType_SelectedIndexChanged;

```

```
//  
// lblMsgTitle  
//  
lblMsgTitle.Location = new Point(10, 235);  
lblMsgTitle.Name = "lblMsgTitle";  
lblMsgTitle.Size = new Size(120, 20);  
lblMsgTitle.TabIndex = 17;  
lblMsgTitle.Text = "Title:";  
//  
// txtMsgTitle  
//  
txtMsgTitle.Location = new Point(140, 232);  
txtMsgTitle.Name = "txtMsgTitle";  
txtMsgTitle.Size = new Size(480, 23);  
txtMsgTitle.TabIndex = 18;  
//  
// lblMsgBody  
//  
lblMsgBody.Location = new Point(10, 265);  
lblMsgBody.Name = "lblMsgBody";  
lblMsgBody.Size = new Size(120, 20);  
lblMsgBody.TabIndex = 19;  
lblMsgBody.Text = "Message:";  
//  
// txtMsgBody  
//  
txtMsgBody.Location = new Point(140, 262);  
txtMsgBody.Multiline = true;
```

```
txtMsgBody.Name = "txtMsgBody";
txtMsgBody.ScrollBars = ScrollBars.Vertical;
txtMsgBody.Size = new Size(480, 80);
txtMsgBody.TabIndex = 20;
//
// lblProgramPath
//
lblProgramPath.Location = new Point(10, 345);
lblProgramPath.Name = "lblProgramPath";
lblProgramPath.Size = new Size(120, 20);
lblProgramPath.TabIndex = 21;
lblProgramPath.Text = "Program Path:";
//
// txtRunPath
//
txtRunPath.Location = new Point(140, 342);
txtRunPath.Name = "txtRunPath";
txtRunPath.Size = new Size(480, 23);
txtRunPath.TabIndex = 22;
//
// lblArguments
//
lblArguments.Location = new Point(10, 375);
lblArguments.Name = "lblArguments";
lblArguments.Size = new Size(120, 20);
lblArguments.TabIndex = 23;
lblArguments.Text = "Arguments:";
//
```



```
// txtRunArgs
//
txtRunArgs.Location = new Point(140, 372);
txtRunArgs.Name = "txtRunArgs";
txtRunArgs.Size = new Size(480, 23);
txtRunArgs.TabIndex = 24;
//
// lblUrl
//
lblUrl.Location = new Point(10, 405);
lblUrl.Name = "lblUrl";
lblUrl.Size = new Size(120, 20);
lblUrl.TabIndex = 25;
lblUrl.Text = "URL:";
//
// txtUrl
//
txtUrl.Location = new Point(140, 402);
txtUrl.Name = "txtUrl";
txtUrl.Size = new Size(480, 23);
txtUrl.TabIndex = 26;
//
// lblFilePath
//
lblFilePath.Location = new Point(10, 435);
lblFilePath.Name = "lblFilePath";
lblFilePath.Size = new Size(120, 20);
lblFilePath.TabIndex = 27;
```

```
lblFilePath.Text = "File Path:";
//
// txtFilePath
//
txtFilePath.Location = new Point(140, 432);
txtFilePath.Name = "txtFilePath";
txtFilePath.Size = new Size(480, 23);
txtFilePath.TabIndex = 28;
//
// lblContent
//
lblContent.Location = new Point(10, 465);
lblContent.Name = "lblContent";
lblContent.Size = new Size(120, 20);
lblContent.TabIndex = 29;
lblContent.Text = "Content:";
//
// txtFileContent
//
txtFileContent.Location = new Point(140, 462);
txtFileContent.Multiline = true;
txtFileContent.Name = "txtFileContent";
txtFileContent.ScrollBars = ScrollBars.Vertical;
txtFileContent.Size = new Size(480, 80);
txtFileContent.TabIndex = 30;
//
// lblMacro
//
```

```
lblMacro.Location = new Point(10, 545);
lblMacro.Name = "lblMacro";
lblMacro.Size = new Size(120, 20);
lblMacro.TabIndex = 31;
lblMacro.Text = "Macro (one per line):";
//
// txtMacroDef
//
txtMacroDef.Location = new Point(140, 542);
txtMacroDef.Multiline = true;
txtMacroDef.Name = "txtMacroDef";
txtMacroDef.ScrollBars = ScrollBars.Vertical;
txtMacroDef.Size = new Size(480, 120);
txtMacroDef.TabIndex = 32;
//
// btnOk
//
btnOk.DialogResult = DialogResult.OK;
btnOk.Location = new Point(406, 672);
btnOk.Name = "btnOk";
btnOk.Size = new Size(100, 25);
btnOk.TabIndex = 33;
btnOk.Text = "OK";
btnOk.Click += BtnOk_Click;
//
// btnCancel
//
btnCancel.DialogResult = DialogResult.Cancel;
```

```

    btnCancel.Location = new Point(520, 672);
    btnCancel.Name = "btnCancel";
    btnCancel.Size = new Size(100, 25);
    btnCancel.TabIndex = 34;
    btnCancel.Text = "Cancel";
    btnCancel.Click += BtnCancel_Click;
    //
    // CreateRuleForm
    //
    ClientSize = new Size(649, 713);
    Controls.Add(pnlMain);
    FormBorderStyle = FormBorderStyle.FixedDialog;
    MaximizeBox = false;
    MinimizeBox = false;
    Name = "CreateRuleForm";
    StartPosition = FormStartPosition.CenterParent;
    Text = "Create Rule";
    pnlMain.ResumeLayout(false);
    pnlMain.PerformLayout();
    ((System.ComponentModel.ISupportInitialize)nudHour).EndInit();
    ((System.ComponentModel.ISupportInitialize)nudMinute).EndInit();
    ResumeLayout(false);

    // Note: do not call user code here in designer file
}

// Designer fields
private System.Windows.Forms.Panel pnlMain;

```

```
private System.Windows.Forms.Label lblName;
private System.Windows.Forms.TextBox txtName;
private System.Windows.Forms.Label lblDescription;
private System.Windows.Forms.TextBox txtDescription;

private System.Windows.Forms.Label lblTriggerType;
private System.Windows.Forms.ComboBox cmbTriggerType;
private System.Windows.Forms.Label lblHour;
private System.Windows.Forms.NumericUpDown nudHour;
private System.Windows.Forms.Label lblMinute;
private System.Windows.Forms.NumericUpDown nudMinute;
private System.Windows.Forms.Label lblEventSource;
private System.Windows.Forms.TextBox txtEventSource;
private System.Windows.Forms.Label lblCondition;
private System.Windows.Forms.TextBox txtEventCondition;

private System.Windows.Forms.Label lblActionType;
private System.Windows.Forms.ComboBox cmbActionType;

private System.Windows.Forms.Label lblMsgTitle;
private System.Windows.Forms.TextBox txtMsgTitle;
private System.Windows.Forms.Label lblMsgBody;
private System.Windows.Forms.TextBox txtMsgBody;

private System.Windows.Forms.Label lblProgramPath;
private System.Windows.Forms.TextBox txtRunPath;
private System.Windows.Forms.Label lblArguments;
```

```
private System.Windows.Forms.TextBox txtRunArgs;
```

```
private System.Windows.Forms.Label lblUrl;
```

```
private System.Windows.Forms.TextBox txtUrl;
```

```
private System.Windows.Forms.Label lblFilePath;
```

```
private System.Windows.Forms.TextBox txtFilePath;
```

```
private System.Windows.Forms.Label lblContent;
```

```
private System.Windows.Forms.TextBox txtFileContent;
```

```
private System.Windows.Forms.Label lblMacro;
```

```
private System.Windows.Forms.TextBox txtMacroDef;
```

```
private System.Windows.Forms.Button btnOk;
```

```
private System.Windows.Forms.Button btnCancel;
```

```
private Label sep;
```

```
}
```

```
}
```

```
using FlexibleAutomationTool.Core.Actions.InternalActions;
```

```
using FlexibleAutomationTool.Core.Actions;
```

```
using FlexibleAutomationTool.Core.Models;
```

```
using FlexibleAutomationTool.Core.Triggers;
```

```
using FlexibleAutomationTool.Core.Interfaces;
```

```
using FlexibleAutomationTool.UI.Services;
```

```
using System;
```

```
using System.Linq;
```

```
using System.Windows.Forms;
```

```

namespace FlexibleAutomationTool.UI
{
    public partial class CreateRuleForm : Form
    {
        private readonly IMessageBoxService _messageBoxService;

        public Rule? CreatedRule { get; private set; }

        public CreateRuleForm(IMessageBoxService messageBoxService)
        {
            _messageBoxService = messageBoxService ?? throw new
ArgumentNullException(nameof(messageBoxService));

            InitializeComponent();

            // Ensure initial state
            UpdateTriggerControls();
            UpdateActionControls();
        }

        private void CmbTriggerType_SelectedIndexChanged(object? sender, EventArgs e)
=> UpdateTriggerControls();

        private void CmbActionType_SelectedIndexChanged(object? sender, EventArgs e)
=> UpdateActionControls();

        private void UpdateTriggerControls()
        {
            var isTime = cmbTriggerType.SelectedItem?.ToString() == "Time";

```

```

nudHour.Enabled = isTime;
nudMinute.Enabled = isTime;
txtEventSource.Enabled = !isTime;
txtEventCondition.Enabled = !isTime;

// Also adjust visibility of labels to keep UI clear
lblHour.Visible = isTime;
nudHour.Visible = isTime;
lblMinute.Visible = isTime;
nudMinute.Visible = isTime;

lblEventSource.Visible = !isTime;
txtEventSource.Visible = !isTime;
lblCondition.Visible = !isTime;
txtEventCondition.Visible = !isTime;
}

private void UpdateActionControls()
{
    var sel = cmbActionType.SelectedItem?.ToString();

    bool isMessage = sel == "Message";
    bool isRun = sel == "RunProgram";
    bool isUrl = sel == "OpenUrl";
    bool isFile = sel == "FileWrite";
    bool isMacro = sel == "Macro";

    if (isMacro)

```



```

{
    // Only macro textbox visible
    lblMsgTitle.Visible = txtMsgTitle.Visible = false;
    lblMsgBody.Visible = txtMsgBody.Visible = false;

    lblProgramPath.Visible = txtRunPath.Visible = false;
    lblArguments.Visible = txtRunArgs.Visible = false;

    lblUrl.Visible = txtUrl.Visible = false;

    lblFilePath.Visible = txtFilePath.Visible = false;
    lblContent.Visible = txtFileContent.Visible = false;

    lblMacro.Visible = txtMacroDef.Visible = true;
    return;
}

// Message
lblMsgTitle.Visible = txtMsgTitle.Visible = isMessage;
lblMsgBody.Visible = txtMsgBody.Visible = isMessage;

// Run
lblProgramPath.Visible = txtRunPath.Visible = isRun;
lblArguments.Visible = txtRunArgs.Visible = isRun;

// Url
lblUrl.Visible = txtUrl.Visible = isUrl;

```

```

// File
lblFilePath.Visible = txtFilePath.Visible = isFile;
lblContent.Visible = txtFileContent.Visible = isFile;

// Macro
lblMacro.Visible = txtMacroDef.Visible = false;
}

```

```

private void BtnOk_Click(object? sender, EventArgs e)

```

```

{
    // Basic validation
    if (string.IsNullOrEmpty(txtName.Text))
    {
        _messageBoxService.Show("Name is required.", "Validation");
        return;
    }

```

```

// Build trigger
Trigger trigger;
if (cmbTriggerType.SelectedItem?.ToString() == "Time")
{
    trigger = new TimeTrigger { Hour = (int)nudHour.Value, Minute =
(int)nudMinute.Value };
}
else
{
    var src = string.IsNullOrEmpty(txtEventSource.Text) ? null :
txtEventSource.Text;

```

```

        var condText = string.IsNullOrEmpty(txtEventCondition.Text) ? null :
txtEventCondition.Text;

        Func<object?, bool>? cond = null;
        if (!string.IsNullOrEmpty(condText))
        {
            cond = (obj) => obj?.ToString()?.IndexOf(condText,
StringComparison.OrdinalIgnoreCase) >= 0;
        }
        trigger = new EventTrigger(src, cond);
    }

    // Build action
    ActionBase action;
    var actType = cmbActionType.SelectedItem?.ToString();
    try
    {
        switch (actType)
        {
            case "Message":
                action = new MessageBoxAction(_messageBoxService) { Title =
txtMsgTitle.Text ?? "", Message = txtMsgBody.Text ?? "" };
                break;
            case "RunProgram":
                action = new RunProgramAction { Path = txtRunPath.Text ?? "",
Arguments = txtRunArgs.Text };
                break;
            case "OpenUrl":
                action = new OpenUrlAction { Url = txtUrl.Text ?? "" };
                break;

```

```

    case "FileWrite":
        action = new FileWriteAction { FilePath = txtFilePath.Text ?? "", Content
= txtFileContent.Text ?? "" };
        break;
    case "Macro":
        var macro = new FlexibleAutomationTool.Core.Actions.MacroAction();
        var lines = txtMacroDef.Text?.Split(new[] { '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries) ?? Array.Empty<string>();
        foreach (var line in lines)
        {
            var parts = line.Split('|');
            if (parts.Length == 0) continue;
            var kind = parts[0].Trim();
            if (string.Equals(kind, "Message", StringComparison.OrdinalIgnoreCase)
&& parts.Length >= 3)
            {
                var title = parts[1];
                var body = string.Join('|', parts.Skip(2));
                macro.Actions.Add(new MessageBoxAction(_messageBoxService) {
Title = title, Message = body });
            }
            else if (string.Equals(kind, "Run", StringComparison.OrdinalIgnoreCase)
&& parts.Length >= 2)
            {
                var path = parts[1];
                var args = parts.Length >= 3 ? string.Join('|', parts.Skip(2)) : null;
                macro.Actions.Add(new RunProgramAction { Path = path, Arguments
= args });
            }
        }
    }

```

```

        else if (string.Equals(kind, "OpenUrl",
StringComparison.OrdinalIgnoreCase) && parts.Length >= 2)
        {
            macro.Actions.Add(new OpenUrlAction { Url = parts[1] });
        }
        else if (string.Equals(kind, "FileWrite",
StringComparison.OrdinalIgnoreCase) && parts.Length >= 3)
        {
            var path = parts[1];
            var content = string.Join('|', parts.Skip(2));
            macro.Actions.Add(new FileWriteAction { FilePath = path, Content =
content });
        }
    }
    action = macro;
    break;
default:
    action = new MessageBoxAction(_messageBoxService) { Title = "Info",
Message = "No action configured" };
    break;
}
}
catch (Exception ex)
{
    _messageBoxService.Show("Failed to build action: " + ex.Message, "Error");
    return;
}

// Create rule

```

```

var rule = new Rule
{
    Name = txtName.Text.Trim(),
    Description = string.IsNullOrEmpty(txtDescription.Text) ? null :
txtDescription.Text.Trim(),
    Trigger = trigger,
    Action = action,
    IsActive = true
};

```

```

CreatedRule = rule;
this.DialogResult = DialogResult.OK;
this.Close();
}

```

// Moved from designer: handle Cancel button click

```

private void BtnCancel_Click(object? sender, EventArgs e)
{
    this.DialogResult = DialogResult.Cancel;
    this.Close();
}
}
}

```

namespace FlexibleAutomationTool.UI

```

{
    partial class EditRuleForm
    {

```

```

/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.txtName = new System.Windows.Forms.TextBox();

```

```
this.txtDescription = new System.Windows.Forms.TextBox();
this.cmbTriggerType = new System.Windows.Forms.ComboBox();
this.nudHour = new System.Windows.Forms.NumericUpDown();
this.nudMinute = new System.Windows.Forms.NumericUpDown();
this.txtEventSource = new System.Windows.Forms.TextBox();
this.txtEventCondition = new System.Windows.Forms.TextBox();
this.cmbActionType = new System.Windows.Forms.ComboBox();
this.txtMsgTitle = new System.Windows.Forms.TextBox();
this.txtMsgBody = new System.Windows.Forms.TextBox();
this.txtRunPath = new System.Windows.Forms.TextBox();
this.txtRunArgs = new System.Windows.Forms.TextBox();
this.txtUrl = new System.Windows.Forms.TextBox();
this.txtFilePath = new System.Windows.Forms.TextBox();
this.txtFileContent = new System.Windows.Forms.TextBox();
this.txtMacroDef = new System.Windows.Forms.TextBox();
this.btnOk = new System.Windows.Forms.Button();
this.btnCancel = new System.Windows.Forms.Button();
this.lblName = new System.Windows.Forms.Label();
this.lblDescription = new System.Windows.Forms.Label();
this.lblTriggerType = new System.Windows.Forms.Label();
this.lblHour = new System.Windows.Forms.Label();
this.lblMinute = new System.Windows.Forms.Label();
this.lblEventSource = new System.Windows.Forms.Label();
this.lblCondition = new System.Windows.Forms.Label();
this.lblActionType = new System.Windows.Forms.Label();
this.lblMsgTitle = new System.Windows.Forms.Label();
this.lblMsgBody = new System.Windows.Forms.Label();
this.lblProgramPath = new System.Windows.Forms.Label();
```



```

this.lblArguments = new System.Windows.Forms.Label();
this.lblUrl = new System.Windows.Forms.Label();
this.lblFilePath = new System.Windows.Forms.Label();
this.lblContent = new System.Windows.Forms.Label();
this.lblMacro = new System.Windows.Forms.Label();
((System.ComponentModel.ISupportInitialize)(this.nudHour)).BeginInit();
((System.ComponentModel.ISupportInitialize)(this.nudMinute)).BeginInit();
this.SuspendLayout();
//
// lblName
//
this.lblName.Location = new System.Drawing.Point(10, 10);
this.lblName.Name = "lblName";
this.lblName.Size = new System.Drawing.Size(100, 23);
this.lblName.Text = "Name:";
//
// txtName
//
this.txtName.Location = new System.Drawing.Point(120, 10);
this.txtName.Name = "txtName";
this.txtName.Size = new System.Drawing.Size(400, 23);
this.txtName.TabIndex = 0;
//
// lblDescription
//
this.lblDescription.Location = new System.Drawing.Point(10, 40);
this.lblDescription.Name = "lblDescription";
this.lblDescription.Size = new System.Drawing.Size(100, 23);

```

```

this.lblDescription.Text = "Description:";
//
// txtDescription
//
this.txtDescription.Location = new System.Drawing.Point(120, 40);
this.txtDescription.Name = "txtDescription";
this.txtDescription.Size = new System.Drawing.Size(400, 23);
this.txtDescription.TabIndex = 1;
//
// lblTriggerType
//
this.lblTriggerType.Location = new System.Drawing.Point(10, 70);
this.lblTriggerType.Name = "lblTriggerType";
this.lblTriggerType.Size = new System.Drawing.Size(100, 23);
this.lblTriggerType.Text = "Trigger:";
//
// cmbTriggerType
//
this.cmbTriggerType.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
this.cmbTriggerType.FormattingEnabled = true;
this.cmbTriggerType.Items.AddRange(new object[] {
    "Time",
    "Event"});
this.cmbTriggerType.Location = new System.Drawing.Point(120, 70);
this.cmbTriggerType.Name = "cmbTriggerType";
this.cmbTriggerType.Size = new System.Drawing.Size(200, 23);
this.cmbTriggerType.TabIndex = 2;

```

```
//  
// lblHour  
//  
this.lblHour.Location = new System.Drawing.Point(120, 100);  
this.lblHour.Name = "lblHour";  
this.lblHour.Size = new System.Drawing.Size(40, 23);  
this.lblHour.Text = "Hour:";  
//  
// nudHour  
//  
this.nudHour.Location = new System.Drawing.Point(160, 100);  
this.nudHour.Maximum = new decimal(new int[] {23,0,0,0});  
this.nudHour.Name = "nudHour";  
this.nudHour.Size = new System.Drawing.Size(60, 23);  
this.nudHour.TabIndex = 3;  
//  
// lblMinute  
//  
this.lblMinute.Location = new System.Drawing.Point(240, 100);  
this.lblMinute.Name = "lblMinute";  
this.lblMinute.Size = new System.Drawing.Size(50, 23);  
this.lblMinute.Text = "Minute:";  
//  
// nudMinute  
//  
this.nudMinute.Location = new System.Drawing.Point(290, 100);  
this.nudMinute.Maximum = new decimal(new int[] {59,0,0,0});  
this.nudMinute.Name = "nudMinute";
```

```
this.nudMinute.Size = new System.Drawing.Size(60, 23);
this.nudMinute.TabIndex = 4;
//
// lblEventSource
//
this.lblEventSource.Location = new System.Drawing.Point(10, 130);
this.lblEventSource.Name = "lblEventSource";
this.lblEventSource.Size = new System.Drawing.Size(100, 23);
this.lblEventSource.Text = "Event Source:";
//
// txtEventSource
//
this.txtEventSource.Location = new System.Drawing.Point(120, 130);
this.txtEventSource.Name = "txtEventSource";
this.txtEventSource.Size = new System.Drawing.Size(400, 23);
this.txtEventSource.TabIndex = 5;
//
// lblCondition
//
this.lblCondition.Location = new System.Drawing.Point(10, 160);
this.lblCondition.Name = "lblCondition";
this.lblCondition.Size = new System.Drawing.Size(100, 23);
this.lblCondition.Text = "Condition:";
//
// txtEventCondition
//
this.txtEventCondition.Location = new System.Drawing.Point(120, 160);
this.txtEventCondition.Name = "txtEventCondition";
```

```

this.txtEventCondition.Size = new System.Drawing.Size(400, 23);
this.txtEventCondition.TabIndex = 6;
//
// lblActionType
//
this.lblActionType.Location = new System.Drawing.Point(10, 190);
this.lblActionType.Name = "lblActionType";
this.lblActionType.Size = new System.Drawing.Size(100, 23);
this.lblActionType.Text = "Action:";
//
// cmbActionType
//
this.cmbActionType.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
this.cmbActionType.FormattingEnabled = true;
this.cmbActionType.Items.AddRange(new object[] {
    "Message",
    "RunProgram",
    "OpenUrl",
    "FileWrite",
    "Macro"});
this.cmbActionType.Location = new System.Drawing.Point(120, 190);
this.cmbActionType.Name = "cmbActionType";
this.cmbActionType.Size = new System.Drawing.Size(200, 23);
this.cmbActionType.TabIndex = 7;
//
// lblProgramPath
//

```

```
this.lblProgramPath.Location = new System.Drawing.Point(10, 220);
this.lblProgramPath.Name = "lblProgramPath";
this.lblProgramPath.Size = new System.Drawing.Size(100, 23);
this.lblProgramPath.Text = "Program Path:";
//
// txtRunPath
//
this.txtRunPath.Location = new System.Drawing.Point(120, 220);
this.txtRunPath.Name = "txtRunPath";
this.txtRunPath.Size = new System.Drawing.Size(400, 23);
this.txtRunPath.TabIndex = 8;
//
// lblArguments
//
this.lblArguments.Location = new System.Drawing.Point(10, 250);
this.lblArguments.Name = "lblArguments";
this.lblArguments.Size = new System.Drawing.Size(100, 23);
this.lblArguments.Text = "Arguments:";
//
// txtRunArgs
//
this.txtRunArgs.Location = new System.Drawing.Point(120, 250);
this.txtRunArgs.Name = "txtRunArgs";
this.txtRunArgs.Size = new System.Drawing.Size(400, 23);
this.txtRunArgs.TabIndex = 9;
//
// lblUrl
//
```

```
this.lblUrl.Location = new System.Drawing.Point(10, 280);
this.lblUrl.Name = "lblUrl";
this.lblUrl.Size = new System.Drawing.Size(100, 23);
this.lblUrl.Text = "URL:";
//
// txtUrl
//
this.txtUrl.Location = new System.Drawing.Point(120, 280);
this.txtUrl.Name = "txtUrl";
this.txtUrl.Size = new System.Drawing.Size(400, 23);
this.txtUrl.TabIndex = 10;
//
// lblFilePath
//
this.lblFilePath.Location = new System.Drawing.Point(10, 310);
this.lblFilePath.Name = "lblFilePath";
this.lblFilePath.Size = new System.Drawing.Size(100, 23);
this.lblFilePath.Text = "File Path:";
//
// txtFilePath
//
this.txtFilePath.Location = new System.Drawing.Point(120, 310);
this.txtFilePath.Name = "txtFilePath";
this.txtFilePath.Size = new System.Drawing.Size(400, 23);
this.txtFilePath.TabIndex = 11;
//
// lblContent
//
```

```
this.lblContent.Location = new System.Drawing.Point(10, 340);
this.lblContent.Name = "lblContent";
this.lblContent.Size = new System.Drawing.Size(100, 23);
this.lblContent.Text = "Content:";
//
// txtFileContent
//
this.txtFileContent.Location = new System.Drawing.Point(120, 340);
this.txtFileContent.Multiline = true;
this.txtFileContent.Name = "txtFileContent";
this.txtFileContent.Size = new System.Drawing.Size(400, 60);
this.txtFileContent.TabIndex = 12;
//
// lblMacro
//
this.lblMacro.Location = new System.Drawing.Point(10, 410);
this.lblMacro.Name = "lblMacro";
this.lblMacro.Size = new System.Drawing.Size(100, 23);
this.lblMacro.Text = "Macro (one per line)";
//
// txtMacroDef
//
this.txtMacroDef.Location = new System.Drawing.Point(120, 410);
this.txtMacroDef.Multiline = true;
this.txtMacroDef.Name = "txtMacroDef";
this.txtMacroDef.Size = new System.Drawing.Size(400, 80);
this.txtMacroDef.TabIndex = 13;
//
```



```
// lblMsgTitle
//
this.lblMsgTitle.Location = new System.Drawing.Point(10, 220);
this.lblMsgTitle.Name = "lblMsgTitle";
this.lblMsgTitle.Size = new System.Drawing.Size(100, 23);
this.lblMsgTitle.Text = "Msg Title:";
//
// txtMsgTitle
//
this.txtMsgTitle.Location = new System.Drawing.Point(120, 220);
this.txtMsgTitle.Name = "txtMsgTitle";
this.txtMsgTitle.Size = new System.Drawing.Size(400, 23);
this.txtMsgTitle.TabIndex = 14;
//
// lblMsgBody
//
this.lblMsgBody.Location = new System.Drawing.Point(10, 250);
this.lblMsgBody.Name = "lblMsgBody";
this.lblMsgBody.Size = new System.Drawing.Size(100, 23);
this.lblMsgBody.Text = "Msg Body:";
//
// txtMsgBody
//
this.txtMsgBody.Location = new System.Drawing.Point(120, 250);
this.txtMsgBody.Multiline = true;
this.txtMsgBody.Name = "txtMsgBody";
this.txtMsgBody.Size = new System.Drawing.Size(400, 60);
this.txtMsgBody.TabIndex = 15;
```

```

//
// btnOk
//
this.btnOk.Location = new System.Drawing.Point(340, 500);
this.btnOk.Name = "btnOk";
this.btnOk.Size = new System.Drawing.Size(80, 25);
this.btnOk.TabIndex = 16;
this.btnOk.Text = "OK";
this.btnOk.UseVisualStyleBackColor = true;
//
// btnCancel
//
this.btnCancel.Location = new System.Drawing.Point(440, 500);
this.btnCancel.Name = "btnCancel";
this.btnCancel.Size = new System.Drawing.Size(80, 25);
this.btnCancel.TabIndex = 17;
this.btnCancel.Text = "Cancel";
this.btnCancel.UseVisualStyleBackColor = true;
//
// EditRuleForm
//
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 15F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(560, 540);
this.Controls.Add(this.lblName);
this.Controls.Add(this.txtName);
this.Controls.Add(this.lblDescription);
this.Controls.Add(this.txtDescription);

```

```
this.Controls.Add(this.lblTriggerType);
this.Controls.Add(this.cmbTriggerType);
this.Controls.Add(this.lblHour);
this.Controls.Add(this.nudHour);
this.Controls.Add(this.lblMinute);
this.Controls.Add(this.nudMinute);
this.Controls.Add(this.lblEventSource);
this.Controls.Add(this.txtEventSource);
this.Controls.Add(this.lblCondition);
this.Controls.Add(this.txtEventCondition);
this.Controls.Add(this.lblActionType);
this.Controls.Add(this.cmbActionType);
this.Controls.Add(this.lblProgramPath);
this.Controls.Add(this.txtRunPath);
this.Controls.Add(this.lblArguments);
this.Controls.Add(this.txtRunArgs);
this.Controls.Add(this.lblUrl);
this.Controls.Add(this.txtUrl);
this.Controls.Add(this.lblFilePath);
this.Controls.Add(this.txtFilePath);
this.Controls.Add(this.lblContent);
this.Controls.Add(this.txtFileContent);
this.Controls.Add(this.lblMacro);
this.Controls.Add(this.txtMacroDef);
this.Controls.Add(this.lblMsgTitle);
this.Controls.Add(this.txtMsgTitle);
this.Controls.Add(this.lblMsgBody);
this.Controls.Add(this.txtMsgBody);
```

```

    this.Controls.Add(this.btnOk);
    this.Controls.Add(this.btnCancel);
    this.Name = "EditRuleForm";
    this.Text = "Edit Rule";
    ((System.ComponentModel.ISupportInitialize)(this.nudHour)).EndInit();
    ((System.ComponentModel.ISupportInitialize)(this.nudMinute)).EndInit();
    this.ResumeLayout(false);
    this.PerformLayout();

}

```

```

#endregion

```

```

private System.Windows.Forms.TextBox txtName;
private System.Windows.Forms.TextBox txtDescription;
private System.Windows.Forms.ComboBox cmbTriggerType;
private System.Windows.Forms.NumericUpDown nudHour;
private System.Windows.Forms.NumericUpDown nudMinute;
private System.Windows.Forms.TextBox txtEventSource;
private System.Windows.Forms.TextBox txtEventCondition;
private System.Windows.Forms.ComboBox cmbActionType;
private System.Windows.Forms.TextBox txtMsgTitle;
private System.Windows.Forms.TextBox txtMsgBody;
private System.Windows.Forms.TextBox txtRunPath;
private System.Windows.Forms.TextBox txtRunArgs;
private System.Windows.Forms.TextBox txtUrl;
private System.Windows.Forms.TextBox txtFilePath;
private System.Windows.Forms.TextBox txtFileContent;

```

```

private System.Windows.Forms.TextBox txtMacroDef;
private System.Windows.Forms.Button btnOk;
private System.Windows.Forms.Button btnCancel;
private System.Windows.Forms.Label lblName;
private System.Windows.Forms.Label lblDescription;
private System.Windows.Forms.Label lblTriggerType;
private System.Windows.Forms.Label lblHour;
private System.Windows.Forms.Label lblMinute;
private System.Windows.Forms.Label lblEventSource;
private System.Windows.Forms.Label lblCondition;
private System.Windows.Forms.Label lblActionType;
private System.Windows.Forms.Label lblMsgTitle;
private System.Windows.Forms.Label lblMsgBody;
private System.Windows.Forms.Label lblProgramPath;
private System.Windows.Forms.Label lblArguments;
private System.Windows.Forms.Label lblUrl;
private System.Windows.Forms.Label lblFilePath;
private System.Windows.Forms.Label lblContent;
private System.Windows.Forms.Label lblMacro;
    }
}

using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Triggers;
using FlexibleAutomationTool.Core.Actions;
using FlexibleAutomationTool.Core.Interfaces;
using FlexibleAutomationTool.UI.Services;
using FlexibleAutomationTool.Core.Actions.InternalActions;

```

```

using System;
using System.Linq;
using System.Windows.Forms;

namespace FlexibleAutomationTool.UI
{
    public partial class EditRuleForm : Form
    {
        private readonly IMessageBoxService _mb;

        public Rule EditedRule { get; private set; }

        public EditRuleForm(Rule rule, IMessageBoxService mb)
        {
            _mb = mb;
            EditedRule = rule;

            InitializeComponent();

            // populate designer fields from existing rule
            txtName.Text = rule.Name;
            txtDescription.Text = rule.Description;

            if (rule.Trigger is TimeTrigger tt)
            {
                cmbTriggerType.SelectedItem = "Time";
                nudHour.Value = tt.Hour;
                nudMinute.Value = tt.Minute;
            }
        }
    }
}

```

```

    }
    else if (rule.Trigger is EventTrigger et)
    {
        cmbTriggerType.SelectedItem = "Event";
        txtEventSource.Text = et.EventSource?.ToString();
        // cannot restore condition expression reliably
    }

    // Populate action-specific fields
    if (rule.Action is MessageBoxAction ma)
    {
        cmbActionType.SelectedItem = "Message";
        txtMsgTitle.Text = ma.Title;
        txtMsgBody.Text = ma.Message;
    }
    else if (rule.Action is RunProgramAction ra)
    {
        cmbActionType.SelectedItem = "RunProgram";
        txtRunPath.Text = ra.Path;
        txtRunArgs.Text = ra.Arguments;
    }
    else if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction ua)
    {
        cmbActionType.SelectedItem = "OpenUrl";
        txtUrl.Text = ua.Url;
    }

```

```

    else if (rule.Action is
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction fa)
    {
        cmbActionType.SelectedItem = "FileWrite";
        txtFilePath.Text = fa.FilePath;
        txtFileContent.Text = fa.Content;
    }
    else if (rule.Action is FlexibleAutomationTool.Core.Actions.MacroAction mac)
    {
        cmbActionType.SelectedItem = "Macro";
        // build macro text lines
        var lines = mac.Actions.Select(a =>
        {
            switch (a)
            {
                case MessageBoxAction m: return $"Message|{m.Title}|{m.Message}";
                case RunProgramAction r: return $"Run|{r.Path}|{r.Arguments}";
                case FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction
u: return $"OpenUrl|{u.Url}";
                case
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction f: return
$"FileWrite|{f.FilePath}|{f.Content}";
                default: return null;
            }
        }).Where(x => x != null);

        txtMacroDef.Text = string.Join(Environment.NewLine, lines);
    }

```



```

// Ensure controls visibility reflects current action selection
UpdateActionControls();

btnOk.Click += BtnOk_Click;
btnCancel.Click += (s, e) => { DialogResult = DialogResult.Cancel; Close(); };

cmbTriggerType.SelectedIndexChanged += (s, e) => UpdateTriggerControls();
cmbActionType.SelectedIndexChanged += (s, e) => UpdateActionControls();
}

private void UpdateTriggerControls()
{
    var isTime = cmbTriggerType.SelectedItem?.ToString() == "Time";
    nudHour.Enabled = isTime;
    nudMinute.Enabled = isTime;
    txtEventSource.Enabled = !isTime;
    txtEventCondition.Enabled = !isTime;

    lblHour.Visible = isTime;
    nudHour.Visible = isTime;
    lblMinute.Visible = isTime;
    nudMinute.Visible = isTime;

    lblEventSource.Visible = !isTime;
    txtEventSource.Visible = !isTime;
    lblCondition.Visible = !isTime;
    txtEventCondition.Visible = !isTime;
}

```

```
private void UpdateActionControls()
{
    var sel = cmbActionType.SelectedItem?.ToString();

    bool isMessage = sel == "Message";
    bool isRun = sel == "RunProgram";
    bool isUrl = sel == "OpenUrl";
    bool isFile = sel == "FileWrite";
    bool isMacro = sel == "Macro";

    if (isMacro)
    {
        // Only macro textbox visible
        lblMsgTitle.Visible = txtMsgTitle.Visible = false;
        lblMsgBody.Visible = txtMsgBody.Visible = false;

        lblProgramPath.Visible = txtRunPath.Visible = false;
        lblArguments.Visible = txtRunArgs.Visible = false;

        lblUrl.Visible = txtUrl.Visible = false;

        lblFilePath.Visible = txtFilePath.Visible = false;
        lblContent.Visible = txtFileContent.Visible = false;

        lblMacro.Visible = txtMacroDef.Visible = true;
        return;
    }
}
```

```

// Message
lblMsgTitle.Visible = txtMsgTitle.Visible = isMessage;
lblMsgBody.Visible = txtMsgBody.Visible = isMessage;

// Run
lblProgramPath.Visible = txtRunPath.Visible = isRun;
lblArguments.Visible = txtRunArgs.Visible = isRun;

// Url
lblUrl.Visible = txtUrl.Visible = isUrl;

// File
lblFilePath.Visible = txtFilePath.Visible = isFile;
lblContent.Visible = txtFileContent.Visible = isFile;

// Macro
lblMacro.Visible = txtMacroDef.Visible = false;
}

private void BtnOk_Click(object? sender, EventArgs e)
{
    if (string.IsNullOrEmpty(txtName.Text))
    {
        _mb.Show("Name is required.", "Validation");
        return;
    }
}

```

```

    EditedRule.Name = txtName.Text.Trim();

    EditedRule.Description = string.IsNullOrEmpty(txtDescription.Text) ? null :
txtDescription.Text.Trim();

    // build trigger
    if (cmbTriggerType.SelectedItem?.ToString() == "Time")
    {
        EditedRule.Trigger = new TimeTrigger { Hour = (int)nudHour.Value, Minute =
(int)nudMinute.Value };
    }
    else
    {
        Func<object?, bool>? cond = null;
        if (!string.IsNullOrEmpty(txtEventCondition.Text))
        {
            var condText = txtEventCondition.Text.Trim();
            cond = (obj) => obj?.ToString()?.IndexOf(condText,
StringComparison.OrdinalIgnoreCase) >= 0;
        }

        EditedRule.Trigger = new
EventTrigger(string.IsNullOrEmpty(txtEventSource.Text) ? null :
txtEventSource.Text, cond);
    }

    // build action
    var act = cmbActionType.SelectedItem?.ToString();
    try
    {
        switch (act)

```

```

{
    case "Message":
        EditedRule.Action = new MessageBoxAction(_mb) { Title =
txtMsgTitle.Text ?? "", Message = txtMsgBody.Text ?? "" };
        break;
    case "RunProgram":
        EditedRule.Action = new RunProgramAction { Path = txtRunPath.Text ??
"", Arguments = txtRunArgs.Text };
        break;
    case "OpenUrl":
        EditedRule.Action = new
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction { Url = txtUrl.Text
?? "" };
        break;
    case "FileWrite":
        EditedRule.Action = new
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction { FilePath =
txtFilePath.Text ?? "", Content = txtFileContent.Text ?? "" };
        break;
    case "Macro":
        var macro = new FlexibleAutomationTool.Core.Actions.MacroAction();
        var lines = txtMacroDef.Text?.Split(new[] { '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries) ?? Array.Empty<string>();
        foreach (var line in lines)
        {
            var parts = line.Split('|');
            if (parts.Length == 0) continue;
            var kind = parts[0].Trim();
            if (string.Equals(kind, "Message", StringComparison.OrdinalIgnoreCase)
&& parts.Length >= 3)

```

```

        {
            var title = parts[1];
            var body = string.Join('|', parts.Skip(2));
            macro.Actions.Add(new MessageBoxAction(_mb) { Title = title,
Message = body });
        }
        else if (string.Equals(kind, "Run", StringComparison.OrdinalIgnoreCase)
&& parts.Length >= 2)
        {
            var path = parts[1];
            var args = parts.Length >= 3 ? string.Join('|', parts.Skip(2)) : null;
            macro.Actions.Add(new RunProgramAction { Path = path, Arguments
= args });
        }
        else if (string.Equals(kind, "OpenUrl",
StringComparison.OrdinalIgnoreCase) && parts.Length >= 2)
        {
            macro.Actions.Add(new
FlexibleAutomationTool.Core.Actions.InternalActions.OpenUrlAction { Url = parts[1] });
        }
        else if (string.Equals(kind, "FileWrite",
StringComparison.OrdinalIgnoreCase) && parts.Length >= 3)
        {
            var path = parts[1];
            var content = string.Join('|', parts.Skip(2));
            macro.Actions.Add(new
FlexibleAutomationTool.Core.Actions.InternalActions.FileWriteAction { FilePath = path,
Content = content });
        }
    }
}

```

```

        EditedRule.Action = macro;
        break;
    default:
        EditedRule.Action = new MessageBoxAction(_mb) { Title = "Info",
Message = "No action configured" };
        break;
    }
}
catch (Exception ex)
{
    _mb.Show("Failed to build action: " + ex.Message, "Error");
    return;
}

DialogResult = DialogResult.OK;
Close();
}
}
}

```

```

namespace FlexibleAutomationTool.UI
{
    partial class MainForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
    }
}

```

```

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>

```

```

protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

```

#region Windows Form Designer generated code

```

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    btnStart = new Button();
    btnStop = new Button();
    btnAddRule = new Button();
    btnDeleteRule = new Button();
    btnViewHistory = new Button();
}

```



```
btnEditRule = new Button();
listBoxRules = new ListBox();
propertyGridRule = new PropertyGrid();
SuspendLayout();
//
// btnStart
//
btnStart.Location = new Point(12, 12);
btnStart.Name = "btnStart";
btnStart.Size = new Size(75, 23);
btnStart.TabIndex = 1;
btnStart.Text = "Start";
btnStart.UseVisualStyleBackColor = true;
btnStart.Click += btnStart_Click;
//
// btnStop
//
btnStop.Location = new Point(93, 12);
btnStop.Name = "btnStop";
btnStop.Size = new Size(75, 23);
btnStop.TabIndex = 2;
btnStop.Text = "Stop";
btnStop.UseVisualStyleBackColor = true;
btnStop.Click += btnStop_Click;
//
// btnAddRule
//
btnAddRule.Location = new Point(12, 79);
```

```

btnAddRule.Name = "btnAddRule";
btnAddRule.Size = new Size(75, 23);
btnAddRule.TabIndex = 3;
btnAddRule.Text = "Add";
btnAddRule.UseVisualStyleBackColor = true;
btnAddRule.Click += btnAddRule_Click;
//
// btnDeleteRule
//
btnDeleteRule.Location = new Point(177, 79);
btnDeleteRule.Name = "btnDeleteRule";
btnDeleteRule.Size = new Size(75, 23);
btnDeleteRule.TabIndex = 4;
btnDeleteRule.Text = "Delete";
btnDeleteRule.UseVisualStyleBackColor = true;
btnDeleteRule.Click += btnDeleteRule_Click;
//
// btnViewHistory
//
btnViewHistory.Location = new Point(12, 45);
btnViewHistory.Name = "btnViewHistory";
btnViewHistory.Size = new Size(94, 23);
btnViewHistory.TabIndex = 7;
btnViewHistory.Text = "View History";
btnViewHistory.UseVisualStyleBackColor = true;
btnViewHistory.Click += btnViewHistory_Click;
//
// btnEditRule

```

```

//
btnEditRule.Location = new Point(93, 79);
btnEditRule.Name = "btnEditRule";
btnEditRule.Size = new Size(75, 23);
btnEditRule.TabIndex = 8;
btnEditRule.Text = "Edit";
btnEditRule.UseVisualStyleBackColor = true;
btnEditRule.Click += btnEditRule_Click;
//
// listBoxRules
//
listBoxRules.FormattingEnabled = true;
listBoxRules.ItemHeight = 15;
listBoxRules.Location = new Point(12, 110);
listBoxRules.Name = "listBoxRules";
listBoxRules.Size = new Size(240, 484);
listBoxRules.TabIndex = 5;
listBoxRules.SelectedIndexChanged += listBoxRules_SelectedIndexChanged;
//
// propertyGridRule
//
propertyGridRule.Location = new Point(270, 12);
propertyGridRule.Name = "propertyGridRule";
propertyGridRule.Size = new Size(632, 582);
propertyGridRule.TabIndex = 9;
//
// MainForm
//

```

```

    AutoScaleDimensions = new.SizeF(7F, 15F);
    AutoScaleMode = AutoScaleMode.Font;
    ClientSize = new.Size(914, 599);
    Controls.Add(propertyGridRule);
    Controls.Add(listBoxRules);
    Controls.Add(btnEditRule);
    Controls.Add(btnViewHistory);
    Controls.Add(btnDeleteRule);
    Controls.Add(btnAddRule);
    Controls.Add(btnStop);
    Controls.Add(btnStart);
    Margin = new.Padding(2);
    Name = "MainForm";
    Text = "MainForm";
    FormClosing += MainForm_FormClosing;
    ResumeLayout(false);
}

```

```

#endregion

```

```

private Button btnStart;
private Button btnStop;
private Button btnAddRule;
private Button btnDeleteRule;
private ListBox listBoxRules;
private Button btnViewHistory;
private Button btnEditRule;
private PropertyGrid propertyGridRule;
}

```

```
}
```

```
using FlexibleAutomationTool.Core.Actions.InternalActions;
using FlexibleAutomationTool.Core.Interfaces;
using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Repositories;
using FlexibleAutomationTool.Core.Services;
using FlexibleAutomationTool.UI.Services;
using Microsoft.Extensions.DependencyInjection;
using FlexibleAutomationTool.Core.Facades;
using System.Diagnostics;
```

```
namespace FlexibleAutomationTool.UI
```

```
{
```

```
    public partial class MainForm : Form
```

```
    {
```

```
        private readonly IRuleRepository _repo;
        private readonly Logger _logger;
        private readonly ServiceManager _svcManager;
        private readonly IAutomationFacade _facade;
        private readonly IMessageBoxService _messageBoxService;
        private readonly MessageBoxAction _messageBoxAction;
        private readonly AutomationEventHandler _eventHandler;
        private readonly IServiceProvider _serviceProvider;
```

```
        // Constructor updated to receive dependencies from DI
```

```
        public MainForm(
```

```
            IRuleRepository repo,
```

```

    Logger logger,
    ServiceManager svcManager,
    IAutomationFacade facade,
    IMessageBoxService messageBoxService,
    MessageBoxAction messageBoxAction,
    AutomationEventHandler eventHandler,
    IServiceProvider serviceProvider)
{
    InitializeComponent();

    // Ensure form is visible and centered (diagnostic for missing window)
    try
    {
        this.StartPosition = FormStartPosition.CenterScreen;
        this.WindowState = FormWindowState.Normal;
        this.ShowInTaskbar = true;
        this.Opacity = 1.0;
        Debug.WriteLine("MainForm ctor: initialized and set visibility properties");
        this.Shown += (s, e) => Debug.WriteLine("MainForm Shown event fired");
    }
    catch (Exception ex)
    {
        Debug.WriteLine("MainForm visibility setup failed: " + ex);
    }

    _repo = repo ?? throw new ArgumentNullException(nameof(repo));
    _logger = logger ?? throw new ArgumentNullException(nameof(logger));

```

```

        _svcManager = svcManager ?? throw new
ArgumentNullException(nameof(svcManager));

        _facade = facade ?? throw new ArgumentNullException(nameof(facade));

        _messageBoxService = messageBoxService ?? throw new
ArgumentNullException(nameof(messageBoxService));

        _messageBoxAction = messageBoxAction ?? throw new
ArgumentNullException(nameof(messageBoxAction));

        _eventHandler = eventHandler ?? throw new
ArgumentNullException(nameof(eventHandler));

        _serviceProvider = serviceProvider ?? throw new
ArgumentNullException(nameof(serviceProvider));

        // Initialization that may throw should not prevent form from showing
        try
        {
            // Підписка на UI івенти (status only)
            _eventHandler.StatusUpdated += OnStatusUpdated;

            RefreshRulesList();

            _facade.Start();
        }
        catch (Exception ex)
        {
            Debug.WriteLine("MainForm initialization error: " + ex);

            try { MessageBox.Show($"Initialization error:\n{ex.Message}", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error); } catch { }
        }
    }
}

```

```
// Прості UI callback методи
```

```
private void OnStatusUpdated(string message)
```

```
{
    this.Text = $"Automation Tool - {message}";
    // statusLabel.Text = message;
}
```

```
private void button1_Click(object sender, EventArgs e)
```

```
{
    _messageBoxAction.Title = "Greeting";
    _messageBoxAction.Message = "Hello from DI!";
    _messageBoxAction.Execute();
}
```

```
// Ensure engine and scheduler are stopped and subscriptions removed when form
closes
```

```
private void MainForm_FormClosing(object? sender, FormClosingEventArgs e)
```

```
{
    // Dispose event handler first to unsubscribe from engine events
    _eventHandler?.Dispose();
    // Stop engine and scheduler to ensure background work ends
    _facade?.Stop();
}
```

```
// Start / Stop
```

```
private void btnStart_Click(object sender, EventArgs e) => _facade.Start();
```

```
private void btnStop_Click(object sender, EventArgs e) => _facade.Stop();
```



```
// Add / Delete
```

```
private void btnAddRule_Click(object sender, EventArgs e)
```

```
{
```

```
    // Resolve CreateRuleForm from DI so it receives IMessageBoxService  
    automatically
```

```
    using var dlg = _serviceProvider.GetRequiredService<CreateRuleForm>();
```

```
    var res = dlg.ShowDialog(this);
```

```
    if (res == DialogResult.OK && dlg.CreatedRule != null)
```

```
    {
```

```
        _facade.CreateRule(dlg.CreatedRule);
```

```
        RefreshRulesList();
```

```
    }
```

```
}
```

```
private void btnDeleteRule_Click(object sender, EventArgs e)
```

```
{
```

```
    if (listBoxRules.SelectedItem is Rule selected)
```

```
    {
```

```
        _repo.Delete(selected.Id);
```

```
        _logger.Log(selected.Name, "Deleted");
```

```
        RefreshRulesList();
```

```
    }
```

```
}
```

```
private Rule BuildRuleFromForm()
```

```
{
```

```

        // TODO: replace with real CreateRuleForm dialog. For now build a simple rule
        using a MessageBoxAction

        var r = new Rule
        {
            Name = "HelloRule",
            Description = "Sample rule created from UI",
            Trigger = new FlexibleAutomationTool.Core.Triggers.EventTrigger(null),
            Action = new
Core.Actions.InternalActions.MessageBoxAction(_messageBoxService) { Title = "Rule",
Message = "Rule executed" },
            IsActive = true
        };

        return r;
    }

    private void RefreshRulesList()
    {
        listBoxRules.Items.Clear();

        // Only display active user rules; hide internal '__System__' placeholder
        foreach (var r in _repo.GetAll().Where(x => x.IsActive &&
!string.Equals(x.Name, "__System__", StringComparison.OrdinalIgnoreCase)))
        {
            listBoxRules.Items.Add(r);
        }
    }

    private void listBoxRules_SelectedIndexChanged(object sender, EventArgs e)
    {

```

```

if (listBoxRules.SelectedItem is Rule selected)
{
    try
    {
        propertyGridRule.SelectedObject = selected;
    }
    catch
    {
    }
}
else
{
    try { propertyGridRule.SelectedObject = null; } catch { }
}
}

```

// New: View History for selected rule or global

```

private void btnViewHistory_Click(object? sender, EventArgs e)
{
    IEnumerable<FlexibleAutomationTool.Core.Models.LogEntry> entries;
    string? filterBy = null;
    if (listBoxRules.SelectedItem is Rule selected)
    {
        // Show history filtered by rule name
        filterBy = selected.Name;

        entries = _facade.GetHistory().Where(le => string.Equals(le.LoggedBy,
selected.Name, StringComparison.OrdinalIgnoreCase));
    }
}

```

```

else
{
    entries = _facade.GetHistory();
}

var dlg = new ViewHistoryForm(entries, _eventHandler, filterBy, _logger);
// show modeless so user can continue interacting with main form
dlg.Show(this);
}

```

// New: Edit rule using CreateRuleForm by repopulating fields - CreateRuleForm does not support editing, so use EditRuleForm

```

private void btnEditRule_Click(object? sender, EventArgs e)
{
    if (listBoxRules.SelectedItem is Rule selected)
    {
        using var dlg = new EditRuleForm(selected, _messageBoxService);
        var res = dlg.ShowDialog(this);
        if (res == DialogResult.OK)
        {
            _repo.Update(dlg.EditedRule);
            _logger.Log(dlg.EditedRule.Name, "Edited");
            RefreshRulesList();
        }
    }
}

```

```

using FlexibleAutomationTool.Core.Actions.InternalActions;
using FlexibleAutomationTool.Core.Interfaces;
using FlexibleAutomationTool.Core.Repositories;
using FlexibleAutomationTool.Core.Services;
using FlexibleAutomationTool.UI.Services;
using Microsoft.Extensions.DependencyInjection;
using FlexibleAutomationTool.Core.Interpreter;
using FlexibleAutomationTool.Core.Factories;
using FlexibleAutomationTool.Core.Facades;
using FlexibleAutomationTool.Core.Data;

```

```

namespace FlexibleAutomationTool.UI
{
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            var services = new ServiceCollection();

            ConfigureServices(services);

            ServiceProvider provider;

            try

```

```

    {
        provider = services.BuildServiceProvider();
    }
    catch (Exception ex)
    {
        try
        {
            MessageBox.Show($"Failed to build services:\n{ex}", "Startup error",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch { }
        throw;
    }

    ApplicationConfiguration.Initialize();

    try
    {
        // Resolve main form and run UI inside try/catch to show any startup exceptions
        var mainForm = provider.GetRequiredService<MainForm>();
        Application.Run(mainForm);
    }
    catch (Exception ex)
    {
        // Show exception so user can see why UI failed to start
        try
        {

```

```
        MessageBox.Show($"Failed to start UI:\n{ex}", "Startup error",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
    }
```

```
    catch
```

```
    {
```

```
        // fallback to console if MessageBox can't be shown
```

```
        Console.Error.WriteLine(ex.ToString());
```

```
    }
```

```
    throw;
```

```
}
```

```
}
```

```
private static void ConfigureServices(ServiceCollection services)
```

```
{
```

```
    var dbPath =
```

```
    Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData),
    "FlexibleAutomationTool", "data.db");
```

```
    Directory.CreateDirectory(Path.GetDirectoryName(dbPath)!);
```

```
    // Try to initialize SQLite-based persistence. If it fails, fall back to in-memory repo
    and logger so UI can start.
```

```
    SqliteDataContext? ctx = null;
```

```
    IExecutionHistoryRepository? historyRepo = null;
```

```
    bool sqliteOk = false;
```

```
    try
```

```
    {
```

```
        ctx = new SqliteDataContext(dbPath);
```

```
        // register context and sqlite-backed repositories
```

```
        services.AddSingleton(ctx);
```

```

        services.AddSingleton<SqliteRuleRepository>();

        services.AddSingleton<IExecutionHistoryRepository, SqliteLogger>(sp => new
SqliteLogger(sp.GetRequiredService<SqliteDataContext>()));

        services.AddSingleton<IRuleRepository>(sp =>
sp.GetRequiredService<SqliteRuleRepository>());

        services.AddSingleton<Logger>(sp => new
Logger(sp.GetRequiredService<IExecutionHistoryRepository>()));

        sqliteOk = true;
    }
    catch (Exception ex)
    {
        try
        {
            MessageBox.Show($"Failed to initialize database, running in memory
mode:\n{ex.Message}", "Database error", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }
        catch { }

        // fallback registrations

        services.AddSingleton<IRuleRepository, InMemoryRuleRepository>();
        services.AddSingleton<Logger>();
    }

    // Always register ServiceManager
    services.AddSingleton<ServiceManager>();

    // Strategy

```



```
services.AddSingleton<FlexibleAutomationTool.Core.Services.ITriggerStrategy,
PollingTriggerStrategy>();
```

```
// Scheduler now needs strategy and repository
```

```
services.AddSingleton<Scheduler>(sp => new
Scheduler(sp.GetRequiredService<IRuleRepository>(),
sp.GetRequiredService<FlexibleAutomationTool.Core.Services.ITriggerStrategy>()));
```

```
// Command dispatcher
```

```
services.AddSingleton<ICommandDispatcher, CommandDispatcher>();
```

```
// Interpreter
```

```
services.AddSingleton<IInterpreter, SimpleMacroInterpreter>();
```

```
// Реєстрація сервісів UI
```

```
services.AddSingleton<IMessageBoxService, WinFormsMessageBoxService>();
```

```
// Platform factory
```

```
services.AddSingleton<IPlatformFactory, WinFormsPlatformFactory>(sp => new
WinFormsPlatformFactory(sp.GetRequiredService<IMessageBoxService>(),
sp.GetRequiredService<ServiceManager>()));
```

```
// Engine (now serves as facade) depends on scheduler and other services
```

```
services.AddSingleton<IAutomationFacade>(sp => new
AutomationEngine(sp.GetRequiredService<IRuleRepository>(),
sp.GetRequiredService<Scheduler>(), sp.GetRequiredService<ServiceManager>(),
sp.GetRequiredService<Logger>(), sp.GetRequiredService<IInterpreter>(),
sp.GetRequiredService<ICommandDispatcher>(),
sp.GetRequiredService<IPlatformFactory>(),
sp.GetService<IExecutionHistoryRepository>()));
```

```

// Facade convenience registration
services.AddSingleton<AutomationEngine>(sp =>
(AutomationEngine)sp.GetRequiredService<IAutomationFacade>());

// Реєстрація Actions
services.AddTransient<MessageBoxAction>();

// Реєстрація форм
services.AddTransient<CreateRuleForm>();
services.AddTransient<MainForm>();

// AutomationEventHandler needs SynchronizationContext.Current from the UI
thread. Use factory to capture it.
services.AddTransient<AutomationEventHandler>(sp =>
    new AutomationEventHandler(
        sp.GetRequiredService<AutomationEngine>(),
        sp.GetRequiredService<Logger>(),
        sp.GetRequiredService<IMessageBoxService>(),
        SynchronizationContext.Current
    )
);
}
}
}

namespace FlexibleAutomationTool.UI
{
    partial class ViewHistoryForm

```

```

{
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be disposed;
    otherwise, false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {

```

```

this.listBoxHistory = new System.Windows.Forms.ListBox();
this.SuspendLayout();
//
// listBoxHistory
//
this.listBoxHistory.Dock = System.Windows.Forms.DockStyle.Fill;
this.listBoxHistory.FormattingEnabled = true;
this.listBoxHistory.ItemHeight = 15;
this.listBoxHistory.Location = new System.Drawing.Point(0, 0);
this.listBoxHistory.Name = "listBoxHistory";
this.listBoxHistory.Size = new System.Drawing.Size(800, 450);
this.listBoxHistory.TabIndex = 0;
//
// ViewHistoryForm
//
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 15F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(800, 450);
this.Controls.Add(this.listBoxHistory);
this.Name = "ViewHistoryForm";
this.Text = "History";
this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.ListBox listBoxHistory;

```

```

    }
}

```

```

using FlexibleAutomationTool.Core.Models;
using FlexibleAutomationTool.Core.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

```

```

namespace FlexibleAutomationTool.UI

```

```

{
    public partial class ViewHistoryForm : Form
    {
        private readonly string? _filterLoggedBy;
        private readonly Logger? _logger;

```

```

        public ViewHistoryForm(IEnumerable<LogEntry> entries,
Services.AutomationEventHandler? events = null, string? filterLoggedBy = null, Logger?
logger = null)

```

```

        {
            InitializeComponent();

            _filterLoggedBy = filterLoggedBy;
            _logger = logger;

            if (entries != null)
            {

```

```

        foreach (var e in entries.OrderBy(x => x.Timestamp))
        {
            listBoxHistory.Items.Add($"[{e.Timestamp:yyyy-MM-dd HH:mm:ss}]
{e.LoggedBy}: {e.Message}");
        }
    }

    if (_logger != null)
    {
        _logger.LogAdded += OnLogAdded;
        this.FormClosed += (s, e) => _logger.LogAdded -= OnLogAdded;
    }
    else if (events != null)
    {
        events.LogEntryAdded += OnLogEntryAdded;
        this.FormClosed += (s, e) => events.LogEntryAdded -= OnLogEntryAdded;
    }
}

private void OnLogEntryAdded(LogEntry entry)
{
    if (entry == null) return;

    if (!string.IsNullOrEmpty(_filterLoggedBy) &&
!string.Equals(entry.LoggedBy, _filterLoggedBy, StringComparison.OrdinalIgnoreCase))
        return;

    if (listBoxHistory.InvokeRequired)

```

```

    {
        listBoxHistory.BeginInvoke((Action)(() =>
listBoxHistory.Items.Add($"[{entry.Timestamp:yyyy-MM-dd HH:mm:ss}]
{entry.LoggedBy}: {entry.Message}")));
    }
    else
    {
        listBoxHistory.Items.Add($"[{entry.Timestamp:yyyy-MM-dd HH:mm:ss}]
{entry.LoggedBy}: {entry.Message}");
    }
}

private void OnLogAdded(LogEntry entry) => OnLogEntryAdded(entry);
}
}

```