АНОТАЦІЯ

Розробка розширеної версії тестових фреймворків для автоматизації сценарії використання на базі dotnet // Дипломна робота магістра // Хацюр Василь Валерійович // ТНТУ, Інженерія програмного забезпечення, група СПм-62 // Тернопіль, 2023 // с. – 55, рис. – 29, табл. – 0, додат. –1 , бібліогр. – 16.

Ключові слова: ФРЕЙМВОРК, ПЛАГІН, ТЕСТ-КЕЙС, ПРОГРАМА, РЕАЛІЗАЦІЯ, ПРОЕКТ.

Метою проекту є проведення досліджень принципів роботи фреймворка для автоматизації тестування, яке допоможе командам, які відповідаю за якість програмного продукту пришвидшити свою роботу та покращити кінцевий результат.

У першому розділі проведено аналіз поточного стану автоматизованого тестування, також досліджено та проаналізовано наявні фреймворки для автоматизації різноманітних програмних рішень. Крім того визначено головну проблему в поточному стані сфери верифікації якості програмних продуктів.

У другому розділі було спроектовано рішення, яке вирішує основну проблему. Наведено архітектуру програмного продукту, те як повинні працювати його окремі модулі та формалізації алгоритмів роботи основних складових проекту.

У третьому розділі було реалізовано програмний продукт, який надає змогу спростити роботу при тестуванні плагінів беручи за основу фунцкціональні вимоги, які були поставлені перед даним додатком, розроблено основні модулі системи та проведено верифікацію всіх компонентів фреймворку на наявність дефектів.

ABSTRACT

Development of an extended version of test frameworks for dotnet-based usage scenario automation // Master thesis // Khatsiur Vasyl // TNTU, Software Engineering, group SPm-62 // Ternopil, 2023 // p. - 55, fig. – 29, table. – 0, Add – 1, Ref. – 16.

Keywords: FRAMEWORK, PLUGIN, TEST CASE, PROGRAM, IMPLEMENTATION, PROJECT.

The goal of the project is to conduct research on the principles of the framework for testing automation, which will help the teams responsible for the quality of the software product to speed up their work and improve the final result.

The first chapter analyzes the current state of automated testing, and also researches and analyzes existing frameworks for automating various software solutions. In addition, the main problem in the current state of the field of quality verification of software products is defined.

In the second chapter, a solution was designed that solves the main problem. The architecture of the software product, how its individual modules should work, and the formalization of the work algorithms of the main components of the project are given.

In the third section, a software product was implemented that makes it possible to simplify the work when testing plugins based on the functional requirements that were set before this application, the main modules of the system were developed, and the verification of all components of the framework for the presence of defects was carried out.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ПД	Програмний додаток
ОП	Охорона праці
ПЗ	Програмне забезпечення
OCR	Оптичне розпізнавання символів
ООП	Об’єктно орієнтований підхід
ROI	Регіон інтересу

ВСТУП

Актуальність теми. За останні роки платформа dotnet виявила надзвичайну динаміку росту і стала однією з провідних технологій для розробки програмного забезпечення. Розширення тестових фреймворків на цій платформі відображає потребу в інструментах, які враховують специфіку dotnet.

У сучасному світі вимоги до якості програмного забезпечення ростуть з кожним днем. Автоматизація тестування виявляється надзвичайно важливою для забезпечення високого рівня стабільності і функціональності програмних продуктів.

З ростом функціональності програмних продуктів зростає і їхня складність. Завдяки розширеній версії тестових фреймворків можна більш ефективно впроваджувати та виконувати тестові сценарії в умовах високої складності програмних рішень.

В умовах швидкого розвитку технологій і високої конкуренції на ринку, швидкість розробки важлива. Розробка розширеної версії тестових фреймворків може значно зменшити час, необхідний для тестування програмних продуктів.

Однією з ключових переваг dotnet є його кросплатформенність. Розробка тестових фреймворків, які сприяють автоматизації тестування на різних операційних системах, стає особливо актуальною в умовах різноманітності екосистем.

В умовах широкого використання DevOps-процесів та Continuous Integration, надійність тестових сценаріїв стає ключовою. Розширення функціональності тестових фреймворків може відповісти цій потребі.

Індустрія постійно рухається в напрямку збільшення автоматизації у всіх аспектах розробки. Розширені тестові фреймворки дозволяють розробникам та тестувальникам зосередитися на більш значущих завданнях, залишаючи рутинні операції автоматизації фреймворку.

Розробка розширеної версії тестових фреймворків спрямована на підвищення якості тестування, що в свою чергу призводить до покращення якості

програмного продукту в цілому. Це особливо важливо в умовах високих вимог до користувацької задоволеності та надійності програм.

Мета дослідження: є дослідження методів і засобів розробки фреймворка для написання автотестів для зовнішніх додатків.

Завдання дослідження:

- Аналіз публікацій та найкращих практик для розробки фреймворків для тестування;
- Проектування модульної архітектури та основних компонентів фреймворку;
- Верифікація на коректність роботи розробленого програмного додатку.

Об’єкт дослідження: процес автоматизації верифікації плагінів для зовнішніх програмних додатків.

Предмет дослідження: методи і програмні засоби для написання тест-кейсів для зовнішніх програмних середовищ.

Методи дослідження. Для вирішення поставленої задачі було використано наступні методи: аналіз – для визначення потенційної області застосування та подальшого розвитку, а також потенційні способи для подолання проблем; проектування та розробка: архітектури модульного моноліта, впровадження даного підходу опираючись на поточні рекомендації для розробки подібниї систем; тестування – для верифікації роботи системи в різних сценаріях використання.

Наукова новизна одержаних результатів. Наукова новизна одержаних результатів:

- запропоновано процес автоматизованої верифікації плагінів, які підключаються до зовнішніх програмних систем виходячи з відсутності наявних рішень.
- Спроектовано фреймворк для написання тест-кейсів для автоматизації верифікації функціоналу, який був розроблений в рамках проекту по розробці плагіну для зовнішніх додатків.

Практичне значення одержаних результатів. Розроблено програмний продукт за допомогою: C#, DotNet, OpenCv, Tesseract для спрощення роботи команди, яка відповідає за якість програних продуктів.

Даний підхід до автоматизації допомагає відділу якості спрощувати перевірку розроблених додатків у випадках, коли немає безпосереднього доступу до програмного інтерфейсу зовнішньої системи.

Публікації. Результати дослідження апробовано на XI науково-технічна конференція «Інформаційні моделі, системи та технології» «Переваги використання архітектури модульного моноліта» (13-14 грудня 2023р.) Тернопільського національного технічного університету імені Івана Пулюя.

1 АНАЛІЗ СУЧАСНОГО СТАНУ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

1.1 Аналіз наявного стану тестування програмних продуктів

Протягом усього часу існування сфери інформаційних технологій дуже важливу частину займало і продовжує займати - тестування. Будь-який розроблений програмний продукт вимагає проведення етапу тестування незалежно від величини проекту, будь то маленький плагін, який використовує вузько спеціалізована частка людей, або ж великий продукт, який займає велику частину ринку і представлений в усьому світі.

Верифікація програмних систем - це обширний процес виявлення невідповідностей в системі визначеним стандартам ефективності та надійності. Цей етап включає виконання різноманітних тестів для виявлення помилок, дефектів або невідповідностей у функціональності програмного додатку високої якості програмного забезпечення перед його випуском або розгортанням у виробництві. Ключовий аспект якого допомагає уникнути проблем із надійністю та стабільністю продукту [1]. Тестування на ранніх стадіях розробки є важливим кроком у виявленні та усуненні помилок залежно від потреб проекту та специфіки програмного продукту, широке тестування може включати функціональне тестування, тестування продуктивності безпеки та тестування на відмовостійкість. Такий підхід допомагає переконатися, що всі аспекти плану ретельно перевірені перед реалізацією

Пропустивши даний етап під час розробки, або ж в процесі підтримки програмної системи - це може призвести до втрати фінансових показників, а в подальшому і до закриття продукту або ж банкротству компанії через відсутність довіри користувачі. Якщо доступ до додатку надається на платній моделі, то користувачі очікують певний рівень якості і досить легко можуть відмовитись від послуг, якщо це прямо впливатиме на їхню роботу.

Тому важливим є забезпечити якісне тестування функціоналу продукту, який потрапляє до кінцевого користувача.

Тестування програмного забезпечення може бути виконане як вручному режимі (мануально), так і за допомогою автоматизованих інструментів (автоматизовано). Обидва підходи мають свої переваги та недоліки, і їх використання залежить від конкретних умов проекту та завдань тестування.

Переваги ручного тестування:

1) ідеально підходить для тестування випадків, які важко автоматизувати;

2) легко адаптується до змін у вимогах та функціоналу;

3) дозволяє тестувати інтуїтивні аспекти користувацького інтерфейсу.

Серед недоліків можна виділити наступні:

1) займає багато часу та ресурсів для повторюваних тестувань;

2) може бути витратним у випадках великої кількості тестових кейсів;

3) може призводити до людських помилок.

Автоматизоване тестування використовує інструменти автоматизації для виконання тест-кейсів. Цей підхід дозволяє значно скоротити час, необхідний для проведення тестування, оскільки використовуються механізовані засоби [2]. З іншого боку, підтримка автоматизованих сценаріїв може зайняти більше часу.

Автоматизація особливо ефективна для великих проектів, де потрібно проводити багаторазові перевірки однієї і тієї ж області. Крім того, цей метод підйде для проектів, які пройшли початкову фазу мануального тестування. Основною метою автоматизації є виявлення помилок у простих операціях. Приклад автотестів, створених для верифікації API сервісів зображено на рис. 1.1.

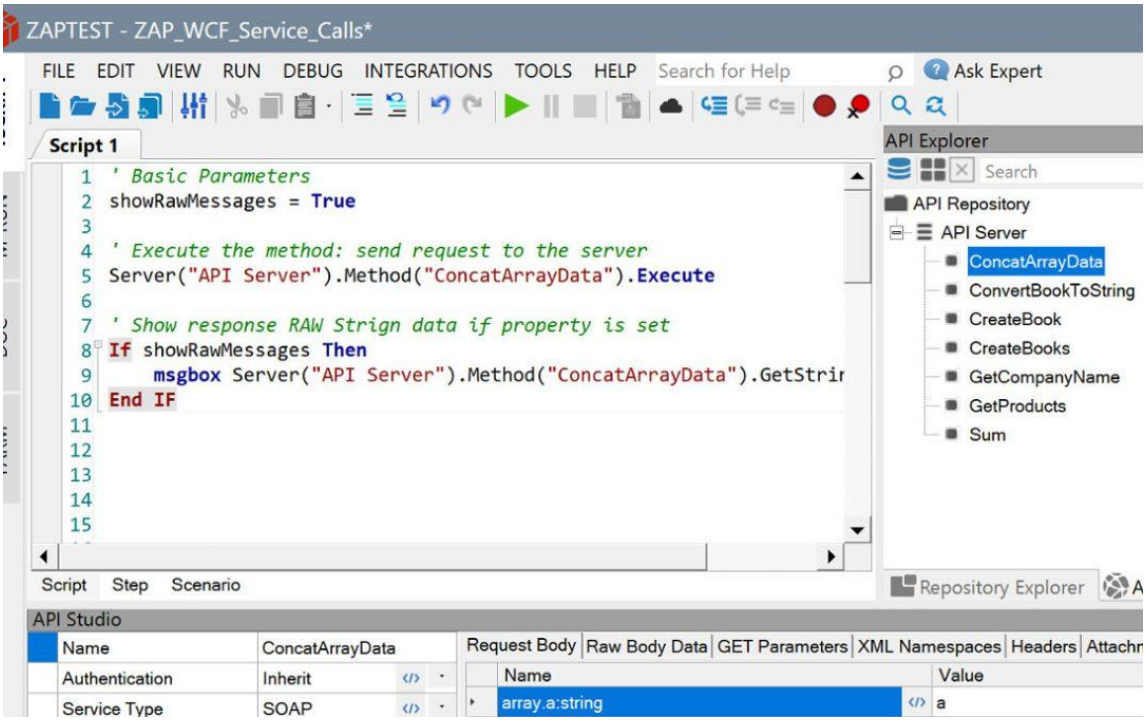

Рисунок 1.1 - Процес написання автотестів

Хоча автоматизоване тестування може бути важливим етапом для багатьох компаній великого масштабу, воно вимагає істотних вкладень для успішної реалізації.

Серед недоліків можна виділити наступні:

1) вимагає значних зусиль для розробки та підтримки автоматизованих сценаріїв;

2) не ефективне для тестування аспектів користувацького інтерфейсу, які важко виміряти автоматично;

3) не завжди ефективне для тестування нового функціоналу, який часто змінюється.

Обрання між мануальним та автоматизованим тестуванням залежить від конкретного контексту проекту, його обсягу, часових обмежень та вимог до якості. Багато організацій використовують комбінований підхід, використовуючи обидва типи тестування для максимізації вигод обох методів.

1.2 Класифікація фреймворків для автоматизованого тестування

Зараз фіксується значний ріст зацікавленості та збільшення кількості веб-додатків на ринку. Це призвело до зростання попиту на робочі місця, що вимагають програмувальних навичок, технічної експертизи та різноманітних навичок автоматизації. Внаслідок цього ручні тестувальники, щоб залишитися конкурентоспроможними, відчувають необхідність переорієнтувати свою діяльність з чистого ручного тестування на контроль якості (QA) і виявляють інтерес до автоматизованого тестування. Перехід від ручного тестування до автоматизованого визнається розумним рішенням, оскільки багато посад QA вимагають розширених технічних навичок, пов'язаних з написанням коду.

Розглянемо наступні фреймворки для автоматизації тестування програмних продуктів:

1) Robot Framework зображений на рис. 1.2.

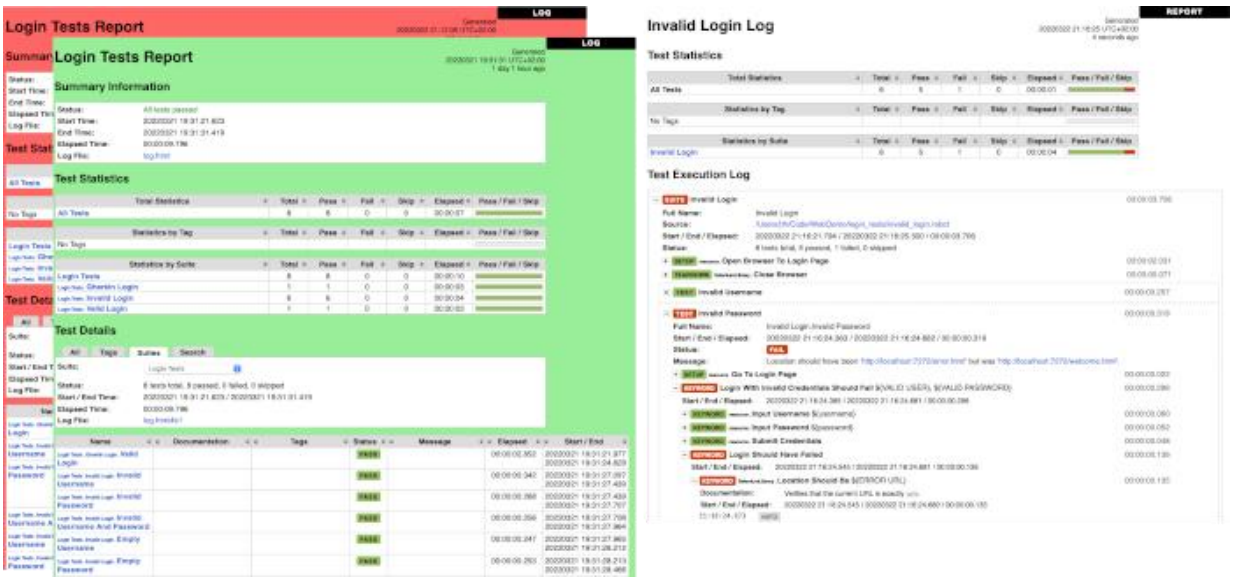

Рис. 1.2. Гайд користувача для Robot Framework

Robot Framework — популярна платформа автоматизації із відкритим кодом. Може використовуватися для здійснення автоматизації тестування та автоматизації роботизованих процесів [5].

Robot Framework Foundation підтримує Robot Framework. Також багато великих компаній використовують його для розробки програмного забезпечення.

Robot Framework, який є відкритим за своєю природою та підтримує можливість розширення функціоналу, може бути легко інтегрований з будь-яким іншим засобом для створення гнучких і автоматизованих рішень. Використання Robot Framework безкоштовне і не потребує витрат на ліцензування.

Синтаксис Robot Framework є простим і використовує зрозумілі людині ключові слова. Можливості цього фреймворку можна розширити за допомогою бібліотек, які реалізовані на Python, Java та інших мовах програмування. Багата екосистема Robot Framework включає в себе різноманітні бібліотеки і інструменти, які розробляються у вигляді окремих проектів.

2) Playwright зображений на рис. 1.3

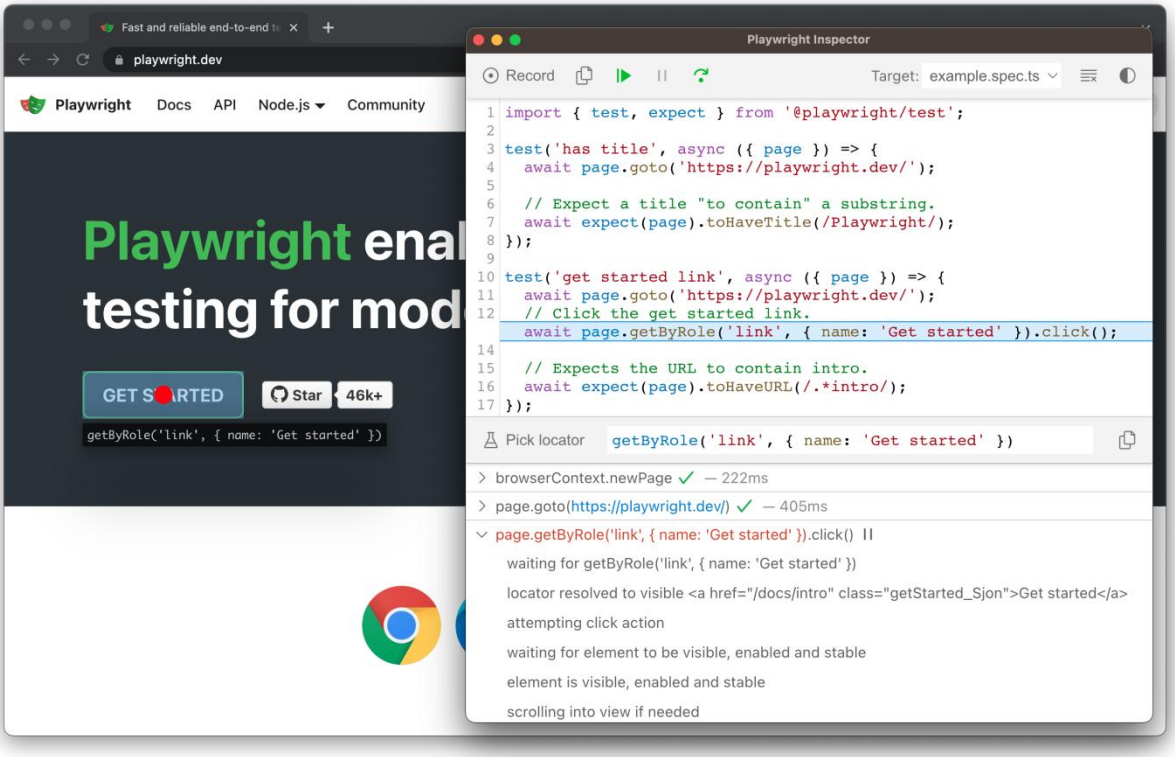

Рис. 1.3. Написання тестів за допомогою Playwright

Playwright — це бібліотека для автоматизації з відкритим вихідним кодом для тестування веб-переглядачів і веб-скрапінгу, розроблена Microsoft і запущена

31 січня 2020 року, яка з тих пір стала популярною серед програмістів і веб-розробників [6].

Playwright надає можливість автоматизувати завдання браузера в Chromium, Firefox і WebKit за допомогою єдиного API. Це дозволяє розробникам створювати надійні наскрізні тести, які можуть працювати як у повноцінному режимі, так і в безголовому режимі для автоматизації.

Playwright підтримує такі мови програмування, як C#, JavaScript, Python і Java, хоча його основний API спочатку був написаний на Node.js. Він підтримує всі сучасні веб-функції, включаючи перехоплення мережі та численні контексти браузера, а також забезпечує автоматичне очікування, що зменшує нестабільність тестів.

3) Selenium зображений на рис. 1.4.

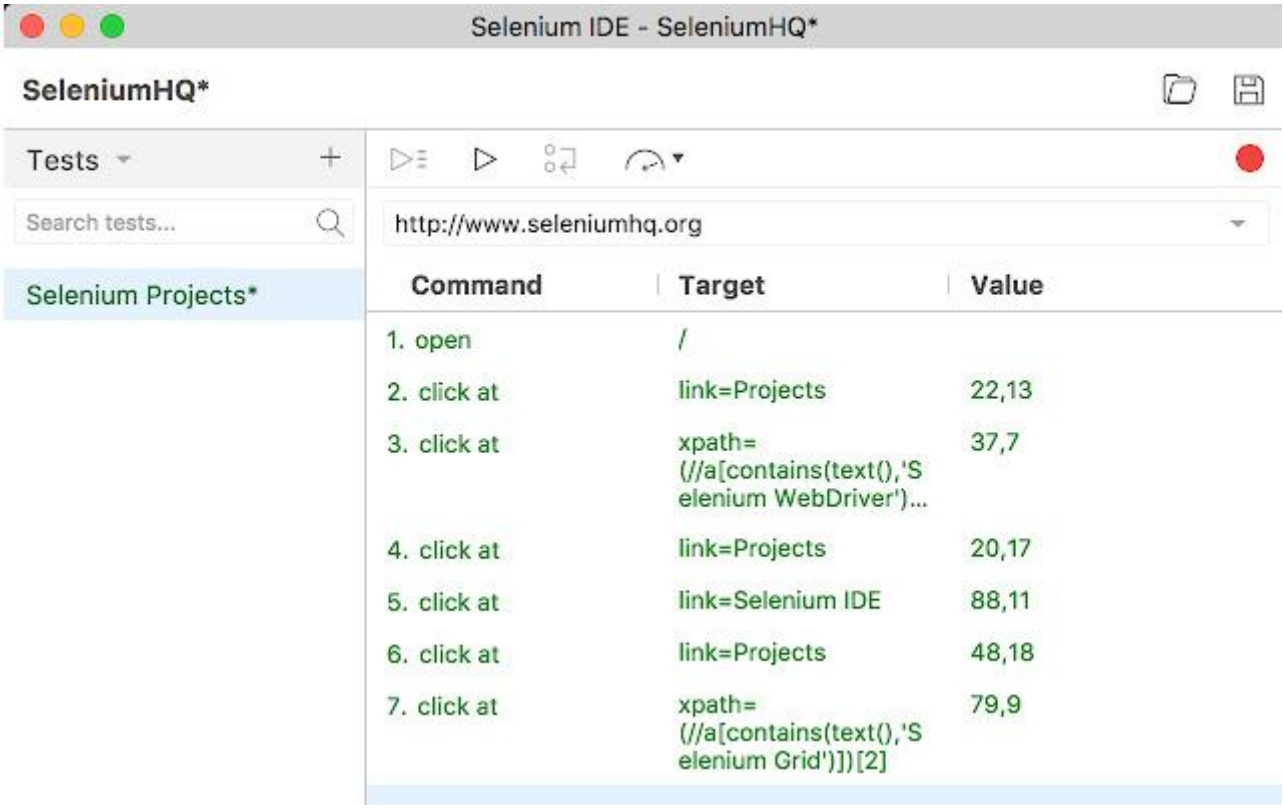

Рис. 1.4. Інтегроване середовище Selenium

Selenium — це проект із відкритим вихідним кодом для ряду інструментів і бібліотек, спрямованих на підтримку автоматизації браузера [7]. Він надає

інструмент відтворення для авторських функціональних тестів у більшості сучасних веб-браузерів без необхідності вивчати мову тестових сценаріїв. Він також надає тестову доменно-спеціальну мову (Selenese) для написання тестів на низці популярних мов програмування, включаючи JavaScript (Node.js), C#, Groovy, Java, Perl, PHP, Python, Ruby та Scala. Selenium працює на Windows, Linux і macOS. Це програмне забезпечення з відкритим кодом, випущене за ліцензією Apache 2.0.

1.3 Проблематика тестування плагінів для сторонніх додатків

У сучасному світі використання різноманітних плагінів для розширення функціоналу сторонніх додатків стало невід'ємною частиною інформаційно-технологічного середовища. Проте, разом із зростанням популярності цих додаткових модулів виникає складна проблематика тестування їхньої працездатності та взаємодії з основним програмним забезпеченням. Існуючі виклики включають в себе сумісність версій, безпеку, ефективність та надійність. Ця тема привертає увагу науковців та фахівців у сфері тестування, спрямованих на вдосконалення методик та інструментів для вирішення складних завдань, пов'язаних із тестуванням плагінів для сторонніх додатків.

В контексті тестування плагінів для сторонніх додатків, важливим аспектом є сумісність версій між самим плагіном і основним додатком [3]. По-перше, розробники повинні враховувати, що оновлення основного додатка можуть внести зміни в його різні частини, з якими взаємодіє плагін. Такі зміни можуть призвести до неправильної роботи плагінів, що робить важливим визначення сумісності плагінів із різними версіями основного додатка.

Другий аспект полягає в тому, що плагіни, які працюють коректно на певних версіях додатка, можуть виявити неправильну роботу або навіть відмовити при використанні інших версій. Це може виникнути через зміни в інтерфейсі користувача, архітектурі програми або у внутрішніх механізмах

обробки даних. Таким чином, для впевненості в правильній роботі плагінів, тестувальники повинні визначити сумісність кожного плагіна із всіма версіями основного додатка, а також виявити і усунути можливі неполадки, пов'язані зі змінами.

Тестування плагінів для сторонніх додатків, особливу увагу слід приділяти аспектам міграції даних, оскільки оновлення або зміни в основному додатку можуть вплинути на формат чи структуру даних, які використовують плагіни. Це може включати в себе не лише перетворення існуючих даних, але й адаптацію плагінів до нових способів зберігання чи обробки інформації. Такі зміни можуть вплинути на правильність роботи плагінів, спричинити втрату даних або навіть порушити їхню функціональність.

Крім того, важливо переконатися, що процес міграції даних відбувається безпечно та ефективно. Це означає перевірку збереження цілісності даних під час їх переміщення та відновлення після завершення процесу міграції. Тестування повинно визначити, чи плагіни можуть взаємодіяти з новою структурою даних, чи можуть ефективно обробляти великі обсяги інформації, а також як вони взаємодіють з іншими компонентами системи під час процесу міграції. Гарантування успішної міграції даних є ключовим аспектом для забезпечення надійності та безперебійності роботи плагінів у змінному середовищі.

Важливою проблемою в деяких випадках також залишається відсутність доступу до API розробника основного додатку, що унеможливлює автоматизацію тестування плагіну. Як результат цього потрібно збільшувати навантаження на команду, яка займається ручним тестування, яким доводиться проходитись по різних сценаріях, щоб перевірити коректність роботи плагіна.

1.4 Постановка задачі

Виходячи з поточного стану, який відображає наявний порядок речей в сфері автоматизації тестування. Також беручи до уваги, які рішення запропоновані на

ринку, для спрощення роботи з верифікації плагінів для зовнішніх систем і проблеми з якими через це стикаються команди з відділу якості програмного забезпечення вимагає створення фреймворку який б вирішував нагальні проблеми.

Проаналізувавши наявну інформацію, можна зробити висновок, що такий фреймворк повинен мати наступні можливості:

1) емулювати поведінку людини в плані вводу (натискання клавіш на клавіатурі, переміщення мишки);

2) запускати визначену програму в якій буде відбуватись тестування заданого функціоналу (в першу чергу до уваги беруть прикладні додатки);

3) можливість зчитувати візуальні маркери, для того щоб можна було переконатись, що ПД має коректний стан в процесі тестування;

4) можливість оновлювати ресурси (інформацію про візуальні маркери) в режимі реального часу, для швидкого реагування на дрібні зміни в інтерфейсі користувача основного додатку;

5) розробити рушій, який б зібрав весь наявний функціонал в одне ціле і запропонував команді відділу якості певний шаблон для написання тест кейсів.

Забезпечивши виконання пунктів, які вказані вище, можна якщо не розв’язати більшість наявних проблем, то як мінімум зменшити їх вплив на наявний стан речей. Крім того важливою складовою є низький поріг входу для даного рішення, що допоможе зменшити фінансове навантаження на компанію, яка займається розробкою плагінів і в той ж момент покращить якість рішень, які вони пропонують.

2 ПРОЕКТУВАННЯ ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦІЇ СЦЕНАРІЇВ ВИКОРИСТАННЯ

2.1 Визначення вимог

У реаліях сучасної розробки програмного забезпечення, важливим етапом є визначення вимог до програмного додатку, яке забезпечить ефективне та надійне автоматизоване тестування. В рамках цього завдання виокремлюються ключові аспекти, що охоплюють емуляцію поведінки людини в плані вводу, запуск тестових програм, зчитування візуальних маркерів та оновлення ресурсів в реальному часі. Розробка рушія та створення шаблонів для тестування визначаються як стратегічні кроки для систематизації та оптимізації тестувальних процесів. У цьому контексті, визначення вимог стає важливою ланкою в розробці програмних додатків, спрямованих на автоматизоване тестування, щоб забезпечити високий рівень якості та ефективність тестових сценаріїв.

Система емуляції вводу: розробка логіки для емуляції вводу є ключовим етапом для роботи автотестів, що емулює поведінку людини. Вона повинна надавати можливість точного відтворення натискань клавіш на клавіатурі та переміщень миші. Параметри емуляції, такі як швидкість та точність, мають бути налаштовані для максимального відповідності реальному користувацькому введенню.

Запуск тестової програми: успішне тестування вимагає автоматичного запуску визначених тестових програм з врахуванням конкретного функціоналу. Це означає створення механізму, який ініціює виконання програм та перевіряє їхню відповідність заданим тестовим кейсам, зокрема для прикладних додатків.

Зчитування візуальних маркерів: для надійного тестування необхідно вбудувати засоби розпізнавання візуальних маркерів. Це дозволить системі перевіряти правильність стану програми під час верифікації а також на ранніх етапах виявляти можливі аномалії чи некоректні відпрацювання логіки.

Оновлення ресурсів в реальному часі: реалізація можливості оновлення ресурсів, зокрема інформації про візуальні маркери, в режимі реального часу є ключовою для ефективного та швидкого реагування на зміни в інтерфейсі основного додатку. В свою чергу це дасть змогу підтримувати актуальність тест-кейсів у будь-який момент.

Розробка рушія та шаблонів для тестування: створення рушія, яке об'єднує весь функціонал та запропонує команді відділу якості шаблон для написання тест-кейсів, є важливим завданням. Це дозволить спростити написання та виконання тест-кейсів, а також уніфікувати підхід які використовуються під час тестування всього програмного продукту.

2.2 Вибір методології розробки

В сучасному інформаційному середовищі, де розробка програмного забезпечення стала необхідністю для різноманітних галузей та секторів, вибір відповідної методології розробки стає ключовим етапом в процесі створення програмних продуктів. Кожна методологія несе у собі свої унікальні підходи, принципи та практики, визначаючи шлях від ідеї до функціонуючого продукту. Цей вибір визначає не лише ефективність розробки, але й впливає на якість, складність управління проектом та його вартість. У такому контексті, вивчення різних методологій розробки є невід'ємною частиною стратегії успішного створення програмного продукту.

Scrum — це ітеративна та інкрементальна методологія розробки програмного забезпечення, яка покладає акцент на гнучкість, комунікацію та швидке адаптивне реагування на зміни вимог [8]. Основними складовими Scrum є короткі ітерації, відомі як спринти, що зазвичай тривають від двох до чотирьох тижнів, як зображено на рис. 2.1. Кожен спринт розпочинається плануванням, під час якого визначаються завдання, що будуть виконуватися протягом ітерації.

Протягом спринту команда розробників щоденно зустрічається на короткому засіданні, відомому як "щоденний Scrum", для оновлення проходження робіт та вирішення евентуальних проблем. По завершенні кожного спринту відбувається ретроспектива, під час якої команда аналізує свої досягнення та визначає шляхи для подальшого вдосконалення.

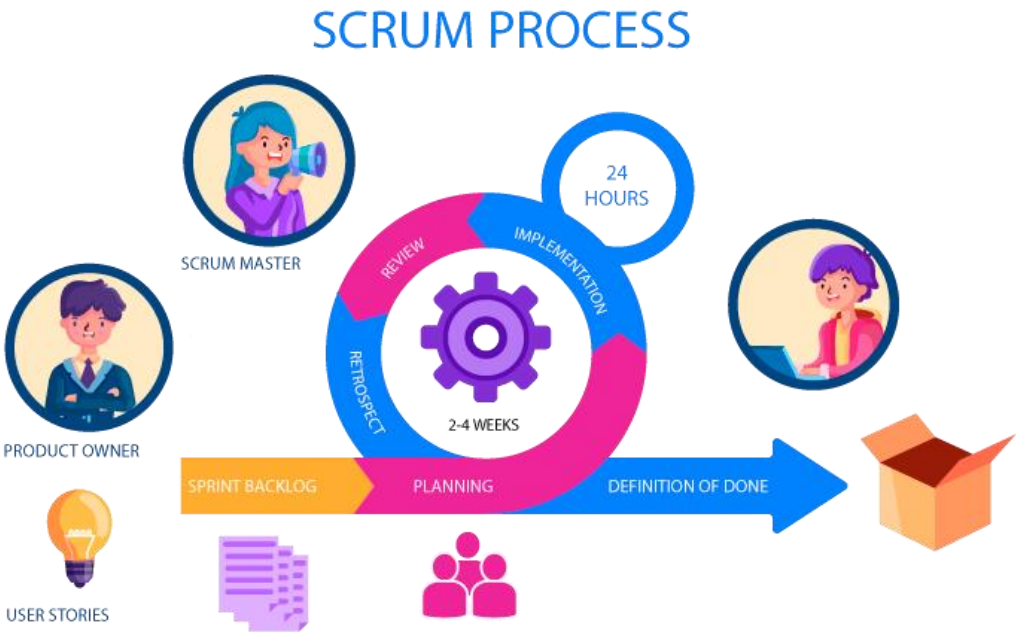

Рис. 2.1. Процеси Scrum методології

Дана методологія надає необхідний інструментарій для швидкого відгуку на зміни вимог, робить розробку прогнозованою та дозволяє гнучко реагувати на зміни пріоритетів. Крім того, вона стимулює комунікацію та співпрацю всіх учасників процесу, забезпечуючи ефективне управління проектом та розвиток продукту. Методологія Scrum також надає зручні інструменти для виявлення і вирішення проблем в реальному часі, зменшуючи ризики та підвищуючи якість розробки. Усі ці аспекти роблять Scrum ідеальним вибором для ефективної та динамічної розробки програмного забезпечення.

2.3 Проектування архітектура додатку

Проектування архітектури програмного додатку визначає фундаментальні структури та організацію системи, що лежать в основі його функціонування. Цей етап розробки відіграє визначальну роль у формуванні надійної, ефективної та легкозмінюваної платформи. Вирішення архітектурних питань передбачає виважену взаємодію між функціональністю, продуктивністю, масштабованістю та іншими ключовими аспектами, що визначають успіх програмного продукту. У цьому контексті важливо ретельно розглядати вибір архітектурних патернів, принципів та стратегій, щоб забезпечити високий рівень якості та гнучкість усього програмного забезпечення.

Архітектура модульного моноліта – це підхід до розробки програмного забезпечення, який поєднує в собі зручності монолітного додатку та можливості модульності [4]. У такій архітектурі програма розбивається на окремі модулі, кожен з яких відповідає за конкретну функціональність чи бізнес-логіку. Ці модулі взаємодіють між собою, утворюючи єдиний, але логічно розділений блок програмного забезпечення. Модульний моноліт сприяє покращенню керованості проекту та обслуговування коду, забезпечуючи легкість виправлення помилок та розширення функціоналу окремих модулів без великих змін в інших частинах системи. Візуальну різницю від традиційної монолітної архітектури зображено на рис. 2.2.

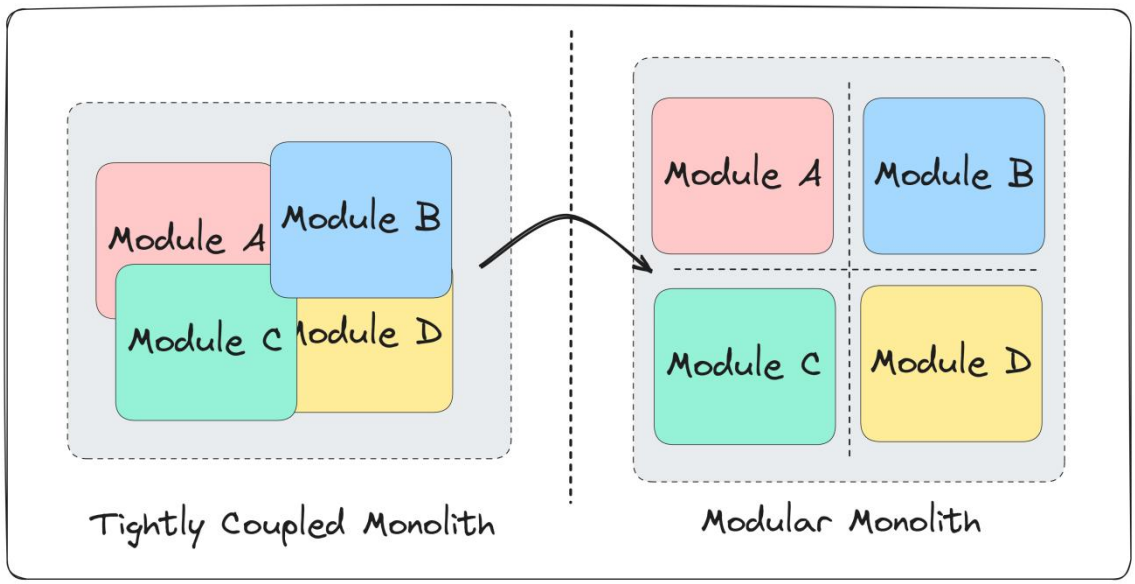

Рис. 2.2. Різниця між традиційними монолітом і модульним

Переваги архітектури модульного моноліта визначаються здатністю поєднувати зручність монолітного підходу та гнучкість модульної структури. По-перше, це спрощує розробку та обслуговування системи, оскільки розбиття на модулі робить код більш структурованим і зрозумілим для команди розробників. По-друге, модульний підхід дозволяє розширювати та змінювати окремі компоненти, не зачіпаючи весь код базового моноліта, що сприяє легкості масштабування та збереженню продуктивності. Крім того, ця архітектура покращує тестування, оскільки можливо проводити тести для окремих модулів, забезпечуючи швидку зміну та вдосконалення системи.

Отже, узявши до уваги всі за і проти, спроектуємо архітектуру даного програмного продукту, яку можна переглянути на рис. 2.3.

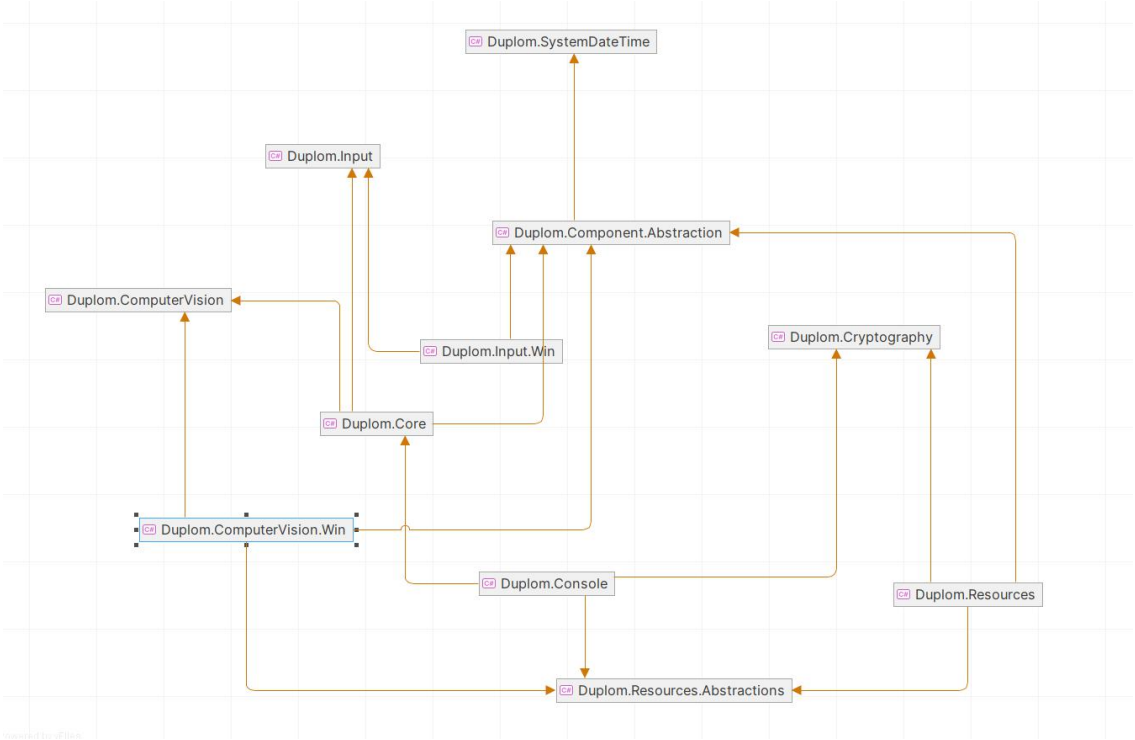

Рис. 2.3. Архітектура фреймворку

2.4 Алгоритм розпізнавання маркерів

Головною особливістю даного фреймворку є розпізнавання маркерів для побудови стану тест-кейса. Для цього використоується технологія для роботи з комп’ютерним зором - OpenCV.

OpenCV, або Open Source Computer Vision Library, є потужною бібліотекою для обробки зображень та комп'ютерного зору [9, 10]. Одним з основних аспектів роботи OpenCV є здатність взаємодіяти з зображеннями у числовому вигляді. Кожен піксель зображення може бути представлений значенням яскравості або кольору, що утворює матрицю чисел. OpenCV надає засоби для зчитування, обробки та аналізу цих матриць, що дозволяє виконувати різноманітні завдання, включаючи розпізнавання облич, об'єктів та інших елементів на зображеннях.

Розпізнавання елементів на фото в OpenCV може бути виконане за допомогою алгоритмів машинного навчання та комп'ютерного зору. Одним із

популярних методів є каскадні класифікатори, що використовуються, наприклад, для розпізнавання облич. Каскадні класифікатори визначають особливості об'єкта та використовують їх для подальшого визначення на зображенні. Для цього вони використовують концепцію Хаарових функцій та інтегральних зображень. Математично це може бути виражено наступним чином (2.1).

$$H(x,y) = \sum_i w_i * f_i(x,y), \tag{2.1}$$

де $H(x,y)$ - сума Хаарових функцій;

w_i - вага кожної функції;

$f_i(x,y)$- хаарова функція.

Каскадні класифікатори застосовують послідовне визначення різних етапів класифікації для поступового відсіювання негативних областей та збільшення точності розпізнавання.

2.5 Алгоритм розпізнавання тексту

Важливим аспектом також можна виділити алгоритм розпізнавання тексту, який використовує разом з алгоритмом розпізнавання маркерів для побудови стану, але в першу чергу для отримання даних з додатку для використання в різноманітних цілях (статистика, аналітика або для бізнес логіки).

Tesseract є відкритою бібліотекою оптичного розпізнавання символів (OCR), яка використовується для перетворення тексту зі зображень у редактований текст [10]. Алгоритм Tesseract базується на комбінації методів обробки зображень, статистичного аналізу та машинного навчання. Перший етап роботи Tesseract полягає в підготовці зображення, включаючи його бінаризацію та видалення шуму. Далі, Tesseract використовує нейромережі та статистичні моделі для

розпізнавання текстових символів на зображенні. Вагомий внесок у високу точність OCR бібліотеки Tesseract вносять його можливості адаптації до різних мов та складних умов зйомки.

Розпізнавання елементів на фото за допомогою Tesseract базується на використанні інструментів та функцій, які ця бібліотека надає. Одним із ключових методів є визначення областей інтересу (ROI) на зображенні, що містять текст. Після визначення ROI Tesseract використовує алгоритми OCR для розпізнавання тексту в кожній області. Сам процес OCR може бути математично виражений наступним чином (2.2).

$$T = OCR(I, P), \tag{2.2}$$

де I - вхідне зображенняй;

T - розпізнаний текст;

P- параметри OCR.

Ця формула відображає основний принцип функціонування OCR - використання вхідного зображення для отримання розпізнаного тексту з використанням певних параметрів та алгоритмів. Tesseract використовує контекстні моделі та словники для вдосконалення точності розпізнавання та підтримки різних мов.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦІЇ СЦЕНАРІЇВ ВИКОРИСТАННЯ

3.1 Вибір технологій розробки

Вибір технологій розробки є стратегічним етапом в життєвому циклі будь-якого програмного продукту, визначаючи його майбутню архітектуру, функціональні можливості та навіть успішність на ринку. Сучасний ландшафт розробки пропонує різноманітні технології, фреймворки та мови програмування, кожна з яких має свої унікальні переваги та обмеження. У цьому контексті, обдуманий та обґрунтований вибір технологічного стеку визначає майбутню успішність проекту, його здатність адаптуватися до змінних вимог ринку та швидкі темпи розвитку індустрії програмної розробки. В даному розділі детально розглядається процес вибору технологій, визначаючи оптимальні засоби для досягнення поставлених завдань та забезпечення якості та стійкості розроблюваного програмного продукту.

В контексті вибору технологій розробки, мова програмування виступає як критичний елемент, що визначає ефективність та продуктивність розробки. Однією з найбільш перспективних та популярних мов є C#. Розроблена компанією Microsoft, C# стала не тільки ключовою мовою для розробки програм на платформі .NET, а й завоювала широке визнання у галузі веб-розробки, ігор та корпоративних застосунків [11].

C# відзначається високим рівнем безпеки, ефективністю та зручністю використання. Її синтаксис є лаконічним та зрозумілим, що полегшує розробку та підтримку коду. Мова володіє об'єктно-орієнтованим підходом, гармонійно інтегрується з багатьма технологіями, такими як ASP.NET для веб-розробки та Unity для ігор.

Більш того, наявність великої та активної спільноти розробників та широкий вибір інструментів для розробки роблять C# привабливим вибором для проектів різного масштабу. Враховуючи ці фактори, докладне вивчення

можливостей мови програмування C# в рамках вибору технологій розробки може виявитися вирішальним для успішного втілення програмного проекту.

Під час вибору технологій для розробки програмного забезпечення, особливо у випадку візуального аналізу та обробки зображень, іншою ключовою складовою може бути використання бібліотек для комп'ютерного зору. Однією з таких потужних та ефективних бібліотек є EmguCV, яка є .NET обгорткою для відомої бібліотеки OpenCV.

EmguCV дозволяє легко інтегрувати функціонал комп'ютерного зору у проекти, розроблені на мові програмування C#. Вона надає доступ до широкого спектру алгоритмів обробки зображень, включаючи відокремлення об'єктів, розпізнавання облич, визначення руху та багато іншого [12, 13].

Використання EmguCV у сполученні з C# дозволяє зручно та ефективно впроваджувати відмінні можливості комп'ютерного зору в програмні продукти. Це може бути надзвичайно корисно в разі тестування візуальних маркерів, аналізу зображень чи інших завдань, пов'язаних з обробкою графічного контенту. Такий інструментарій сприяє покращенню точності та розширенню можливостей програмного додатку, роблячи його більш адаптованим до вимог сучасного ринку програмної розробки.

При виборі технологій для розробки програмного забезпечення, яке включає в себе оптичне розпізнавання символів (OCR), важливим елементом стає вибір потужного та надійного OCR движка. Тут TessaractOCR видається відмінним варіантом. TessaractOCR - це відкритий інструмент для OCR, розроблений Google. Він дозволяє впроваджувати функціонал розпізнавання тексту в зображеннях та документах безпосередньо у проекти, написані на мові програмування C#.

Завдяки TessaractOCR, розробники можуть легко інтегрувати потужні засоби OCR в свої програми, щоб автоматизувати процес розпізнавання тексту. Tessaract відрізняється високою точністю розпізнавання, а також можливістю працювати з текстом різної складності та структури. Використання TessaractOCR у поєднанні з мовою програмування C# дозволяє розробникам з легкістю

виконувати завдання з автоматичним розпізнаванням тексту, сприяючи ефективності та розширенню функціоналу програмного продукту.

3.2 Розробка програмного додатку

Першочергово потрібно впровадити архітектуру, яку було визначено на етапі проектування. А саме розробити основний інтерфейс для модулів (рис. 3.1).

```
public interface IComponent
{
    void Initialize(ContainerBuilder builder);

    Task ExecuteAsync(IContainer container);
}
```

Рис. 3.1. Програмний код основного інтерфейсу для модуля

Наступним кроком буде розробка модуля для роботи з ресурсами. А саме зберігання/отримання візуальних ресурсів, як конфігураційні файли та картинки. Даний компонент складається з сховища (ResourcesStorage), який представляє собою набір функцій для отримання доступу до ресурсів. В даному випадку підтримується три види сховищ, а саме:

1) Сховище відкритих ресурсів

2) Сховище захищених ресурсів

3) Віддалене сховище.

Сховище відкритих ресурсів відповідає за отримання доступу до файлів, які зберігаються безпосередньо на диску ПК і є у відкритому вигляді, будь-який користувач може їх відкрити за допомогою тестового редактора або переглядача фотографій вбудованого в операційній системі. Функцію яка відповідає за роботу з ресурсами представлено на рис. 3.2.

```
private static void LoadResourcePathToCache(
    string workingDirectory,
    Dictionary<string, string> cache)
{
    var directories = Directory.GetDirectories(workingDirectory);
    foreach (var directory in directories)
        LoadResourcePathToCache(
            Path.Combine(workingDirectory, directory),
            cache);
    var files = Directory.GetFiles(workingDirectory);
    foreach (var file in files)
    {
        var extension = Path.GetExtension(file);
        var fileName = Path.GetFileNameWithoutExtension(file);
        switch (extension)
        {
            case ".png":
                cache.Add(ResourceKeyBuilder.BuildByteKey(fileName), file);
                break;
            case ".json":
                cache.Add(ResourceKeyBuilder.BuildTextKey(fileName), file);
                break;
            default:
                break;
        }
    }
}
```

Рис. 3.2. Програмний код роботи з ресурсами у відкритому сховищі

Сховище захищених ресусрів має аналогічний функціонал до сховища відкритих ресурсів. Відмініть полягає в тому, що в даному випадку ресурси є зашифрованими з певним ключом і відкритого доступу до них немає. Функцію яка відповідає за роботу з ресурсами представлено на рис. 3.3.

```
private static List<string> GetResourcesNames(
    IResourcesContainer container,
    byte[] key,
    byte[] IV,
    string resourceNamesHash)
{
    var resourceBytes = container.GetResource(resourceNamesHash);
    var decryptedBytes =
        RijndaelWrapper.DecryptBytes(resourceBytes, key, IV);
    var json = Encoding.UTF8.GetString(decryptedBytes);
    return JsonSerializer.Deserialize<List<string>>(json);
}
```

Рис. 3.3. Програмний код роботи з ресурсами у захищеному сховищі

Віддалене сховище аналогічно до всіх інших відповідає тим самим правилам, що до функціоналу, а різниця полягає в зберіганні даних файлів. В даному випадку ресурси знаходяться на віддаленому сервері і будуть завантажуватись, коли будуть якісь оновлення на серверів і вже на ПК зберігається відповідно до того, як це було в сховищі захищених ресурсів. Функцію яка відповідає за роботу з ресурсами представлено на рис. 3.4.

```
private static Dictionary<string, byte[]> ReadResourcesFromZip(string name)
{
    var pathToFile = Path.Combine(
        Environment.GetEnvironmentVariable("REAL_TMP"),
        "Duplom",
        name);
    using var internalZip = new ZipFile(pathToFile)
    {
        Password = SHA256Wrapper.ComputeSha256Hash(INTERNAL_ZIP_PASSWORD),
        Encryption = EncryptionAlgorithm.WinZipAes256
    };
    var result = new Dictionary<string, byte[]>();
    foreach (var entry in internalZip)
    {
        using var memorystream = new MemoryStream();
        entry.Extract(memorystream);
        result.Add(entry.FileName, memorystream.ToArray());
    }
    return result;
}
```

Рис. 3.4. Програмний код роботи з ресурсами у віддаленому сховищі

Наступний компонент, який потрібно розробити - модуль роботи з вводом, а саме симуляція натискання розноманітніх клавіша клавіатури або з комп’ютерної миші. Алгоритм функції, яка відповідає за симуляцію натискання клавіш на клавіатурі представлено на рис. 3.5.

```
public Task PressKeyAsync(KeyboardKey key)
{
    var virtualKeyCode = KeyboardKeyMap.ToVirtualKeyCode(key);
    if (virtualKeyCode is null)
    {
        return Task.CompletedTask;
    }

    _keyboardSimulator.Keyboard.KeyPress(virtualKeyCode.Value);
    return Task.Delay(TimeSpan.FromSeconds(2));
}
```

Рис. 3.5. Програмний код роботи симуляторна клавіатури

Останній основний компонент є модуль комп'ютерного зору, який працює з маркерами і тексом на скріншотах робочого середовища операційної системи. Основою даного модуля є дві функці: пошук піксельного маркера та пошук маркера на основі картинки.

Функція пошуку піксельного маркера пробує порівняти піксель на скріншоті з інформацією, яка задається в конфігараційному файлі даного маркера, відповідно на основі цих даних видає результат. Роботу даної функції представлено на рис. 3.6.

```
private static bool FindPixel(Image<Bgr, byte> image, PixelMarkerInfo markerInfo)
{
    var delta = markerInfo.DifDelta.HasValue
        ? new Rgb(
            markerInfo.DifDelta.Value.Red,
            markerInfo.DifDelta.Value.Green,
            markerInfo.DifDelta.Value.Blue)
        : new Rgb(markerInfo.Delta,markerInfo.Delta,markerInfo.Delta);
    image.ROI = RectangleDrawing.Empty;
    var color = image[markerInfo.Position.Y, markerInfo.Position.X];
    var redCondation = Math.Abs(color.Red - markerInfo.Color.R) > delta.Red;
    var greenCondition = Math.Abs(color.Green - markerInfo.Color.G) > delta.Green;
    var blueCondition = Math.Abs(color.Blue - markerInfo.Color.B) > delta.Blue;
    return !(redCondation || greenCondition || blueCondition);
}
```

Рис. 3.6. Програмний код роботи функції для пошуку піксельного маркера

Функція пошуку маркера на основі картинки працює за аналогічним принципом, як і попередня. Тільки тепер шукається картинка на яку зсилається

конфігураційний файл і пробує найти співпадіння в заданій області. Результат буде точка в якій є співпадіння відносно вказаної точності. Алгоритм даної функції представлено на рис. 3.7.

```
private static Point FindMarker(Image<Bgr, byte> image, ImageMarkerInfo markerInfo)
{
    var searchArea = markerInfo.GetSearchArea();
    // TODO refactore this condition
    if (markerInfo.Image.Width > searchArea.Width
        || markerInfo.Image.Height > searchArea.Height)
        return Point.Empty;
    image.ROI = searchArea;
    using var result = markerInfo.Mask == null
        ? image.MatchTemplate(markerInfo.Image, TemplateMatchingType.CcoeffNormed)
        : image.MatchTemplate(markerInfo.Image, TemplateMatchingType.CcorrNormed, markerInfo.Mask);
    result.MinMax(out _, out var maxValues, out _, out var maxLocations);
    image.ROI = RectangleDrawing.Empty;
    if (maxValues.Length == 0 || maxValues[0] < markerInfo.Accuracy / 100)
        return Point.Empty;
    var location = maxLocations.Length != 0 ? maxLocations[0] : Point.Empty;
    return new Point(searchArea.X + location.X, searchArea.Y + location.Y);
}
```

Рис. 3.7. Програмний код функції для пошуку маркера на основі картинки

Важливим аспектом даного фреймворку також є рушій, який запускає дії, які описуються під час тест-кейсу. Даний рушій працює на основі сценаріїв. Сценарій - це певний набір дій, які потрібно зробити або можливо зробити (не обов’язково) для досягнення певного стану тест-кейсу. Приклад роботи з обов’язковими діями представлено на рис. 3.8.

```
private static async Task<(bool, bool)> ProcessingRegularActionAsync(IAction action)
{
    var canExecute = action.IsSkipCanExecute || await action.CanExecuteAsync();
    if (canExecute)
    {
        return (canExecute, await action.ExecuteAsync());
    }
    else
    {
        await Task.Delay(TimeSpan.FromMilliseconds(200));
        return (false, false);
    }
}
```

Рис. 3.8. Програмний код роботи функції обробки обов’язкових дій

3.3 Тестування програмного додатку

Проведемо верифікацію даного програмного продукту. В даному випадку буде використовуватись додаток “Калькулятор”, як система для якої будуть писатись тест-кейси. Для початку необхідно створити проект і підключити всі необхідні компоненти (рис. 3.9).

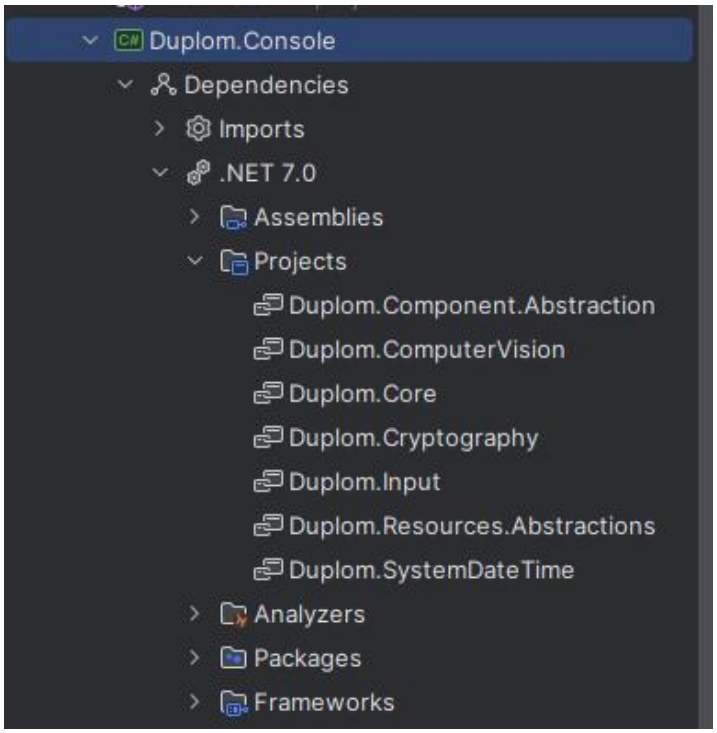

Рис. 3.9. Проект тестового скріпта

Далі необхідно створити тестовий скріпт, який буде містити сценарії для тест-кейсів (рис. 3.10).

```
var script = new TestScript();
script.Initialize();
await script.RunAsync(CancellationToken.None);
```

Рис. 3.10. Ініціалізація тестового скріпта

Подальші кроки включають створення дії, яка відповідає за введення наперед заданого числа. Алгоритм даної дії представлено на рис. 3.11.

```
public class EnterNumberAction : ActionBase
{
    public override string Name => nameof(EnterNumberAction);

    public EnterNumberAction(
        IKeyboardApi keyboardApi,
        IVisualAnalyzer va,
        ILogger logger)
        : base(keyboardApi, va, logger)
    {
    }

    public override Task<bool> CanExecuteAsync()
        => VA.HasMarkerAsync("CALCULATOR_IS_ACTIVE");

    public override async Task<bool> ExecuteAsync()
    {
        await KeyboardApi.PressKeyAsync(KeyboardKey.Tab);
        await KeyboardApi.PressKeyAsync(KeyboardKey.Num4);
        await KeyboardApi.PressKeyAsync(KeyboardKey.Num9);
        await KeyboardApi.PressKeyAsync(KeyboardKey.Num1);
        return true;
    }
}
```

Рис. 3.11. Програмний код дії для введення числа

Після цього потрібно створити дію, яка б розпізнала введене число і вивела його в консолі розробника. Алгоритм даної дії представено на рис. 3.12.

```
public class ShowCurrentValueAction : ActionBase
{
    private readonly IVisualParser _visualParser;

    public override string Name
        => nameof(ShowCurrentValueAction);

    public ShowCurrentValueAction(
        IKeyboardApi keyboardApi,
        IVisualAnalyzer va,
        IVisualParser visualParser,
        ILogger logger)
        : base(keyboardApi, va, logger)
    {
        _visualParser = visualParser;
    }

    public override Task<bool> CanExecuteAsync()
        => VA.HasMarkerAsync("CALCULATOR_IS_ACTIVE");

    public override async Task<bool> ExecuteAsync()
    {
        var value = await _visualParser.GetIntegerValueAsync("CURRENT_VALUE_IN_NUMBER_BOX");
        Logger.Information("Current value in number box is {value}", value);
        return true;
    }
}
```

Рис. 3.12. Програмний код дії для розпізнавання числа

З даних дій опишемо сценарій, який спочатку введе число, після чого розпізнає його і виведе його в консоль розробника. Алгорим даної дії представлено на рис. 3.13.

```
public override async Task<ScriptExecuteResult> RunAsync(
    CancellationToken cancellationToken)
{
    await _componentManager.ExecuteComponentsAsync(Container);

    await RunMacroScenarioAsync(builder =>
    {
        builder.Register<EnterNumberAction>()
            .Register<ShowCurrentValueAction>();
    }, cancellationToken);

    return ScriptExecuteResultFactory.Create(ExecuteStatus.Done);
}
```

Рис. 3.13. Програмний код тестового сценарію

Далі необхідно запустити даний скрипт і перевірити роботу всіх компонентів даного фрейморку. Після запуску скріпта можна побачити початковий екран додатку “Калькулятор” (рис. 3.14).

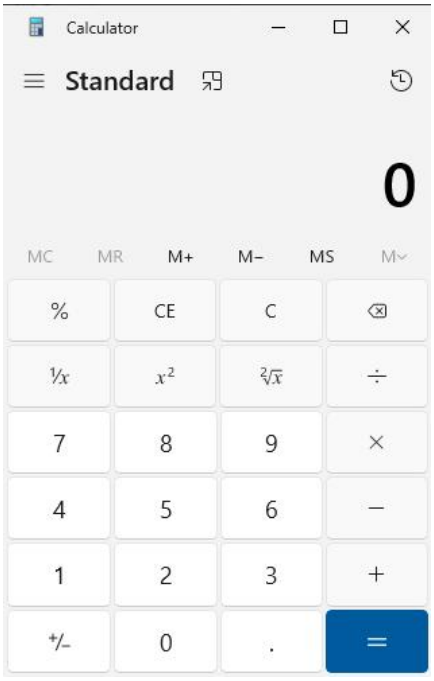

Рис. 3.14. Початковий екран програми “Калькулятор”

Тепер запускається сценарій, який спочатку вводить задане число, а саме - 491 (рис. 3.15)

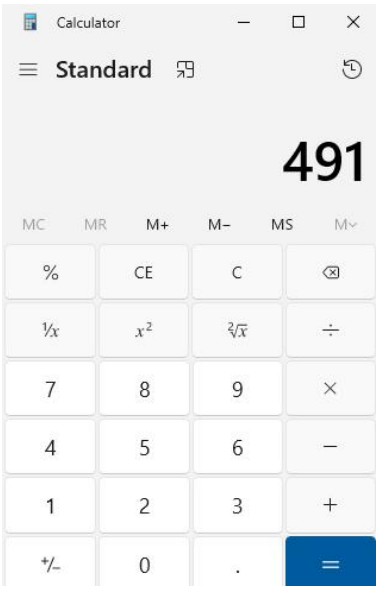

Рис. 3.15. Результат виконання дії введення числа

Наступна дія розпізнає дане число і результат роботи виведе в консоль розробника (рис. 3.16).

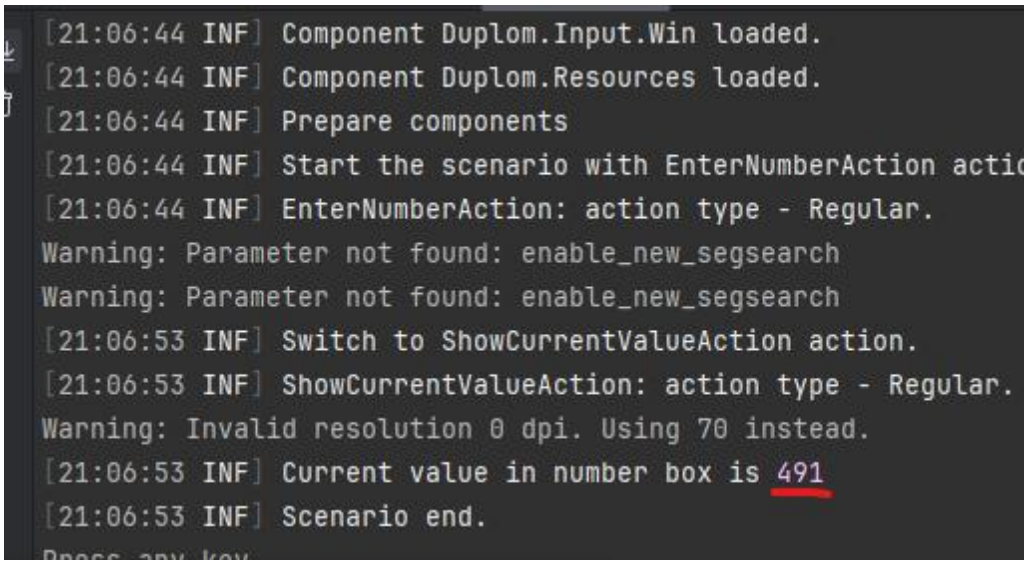

Рис. 3.16. Результат виконанняя дії розпізнавання числа

Створимо ще один сценарій, який буде запускати додаток “Postman” і створювати запит для заданої кінцевої точки.

Для початку потрібно створити дію, яка буде створювати новий запит, коли додаток “Postman” відкритий (рис. 3.17).

```
public class CreateEndpointAction : ActionBase
{
    public override string Name
        => nameof(CreateEndpointAction);

    public CreateEndpointAction(
        IKeyboardApi keyboardApi,
        IVisualAnalyzer va,
        ILogger logger) : base(keyboardApi, va, logger)
    {
    }

    public override Task<bool> CanExecuteAsync()
        => VA.HasMarkerAsync("POSTMAN_IS_ACTIVE");

    public override async Task<bool> ExecuteAsync()
    {
        await KeyboardApi.MoveMouseToAsync(new Point(124, 55));
        await KeyboardApi.MouseLeftClickAsync();
        await KeyboardApi.PressKeyAsync(KeyboardKey.Enter);
        return true;
    }
}
```

Рис. 3.17. Програмний код дії створення запита

Результат роботи даної дії наведено на рис. 3.18.

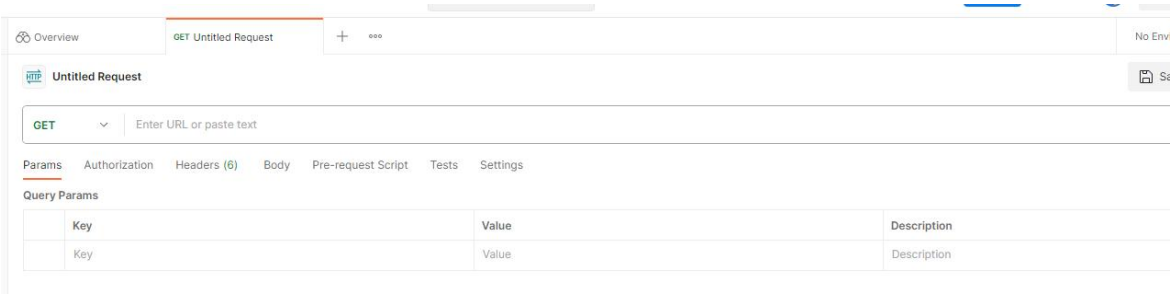

Рис. 3.18. Результат роботи дії створення запита

Наступним кроком буде створення дії, яка відповідає за введення адреси кінцевої точки на яку ми хочемо зробити запит (рис. 3.19). Потрібно задати в коді адресу кінцевої точки на яку буде здійснюватись запит, та передати його в модуль, який відповідає за роботу з операціями введення-виведення.

```
public class EnterUrlAction : ActionBase
{
    public override string Name
        => nameof(EnterUrlAction);

    public EnterUrlAction(
        IKeyboardApi keyboardApi,
        IVisualAnalyzer va,
        ILogger logger) : base(keyboardApi, va, logger)
    {
    }

    public override Task<bool> CanExecuteAsync()
        => VA.HasMarkerAsync("POSTMAN_HTTP_IS_ACTIVE");

    public override async Task<bool> ExecuteAsync()
    {
        await KeyboardApi.EnterTextAsync("https://www.google.com/");
        return true;
    }
}
```

Рис. 3.19. Програмний код дії введена кінцевої точки

Результат роботи даної дії наведено на рис. 3.20.

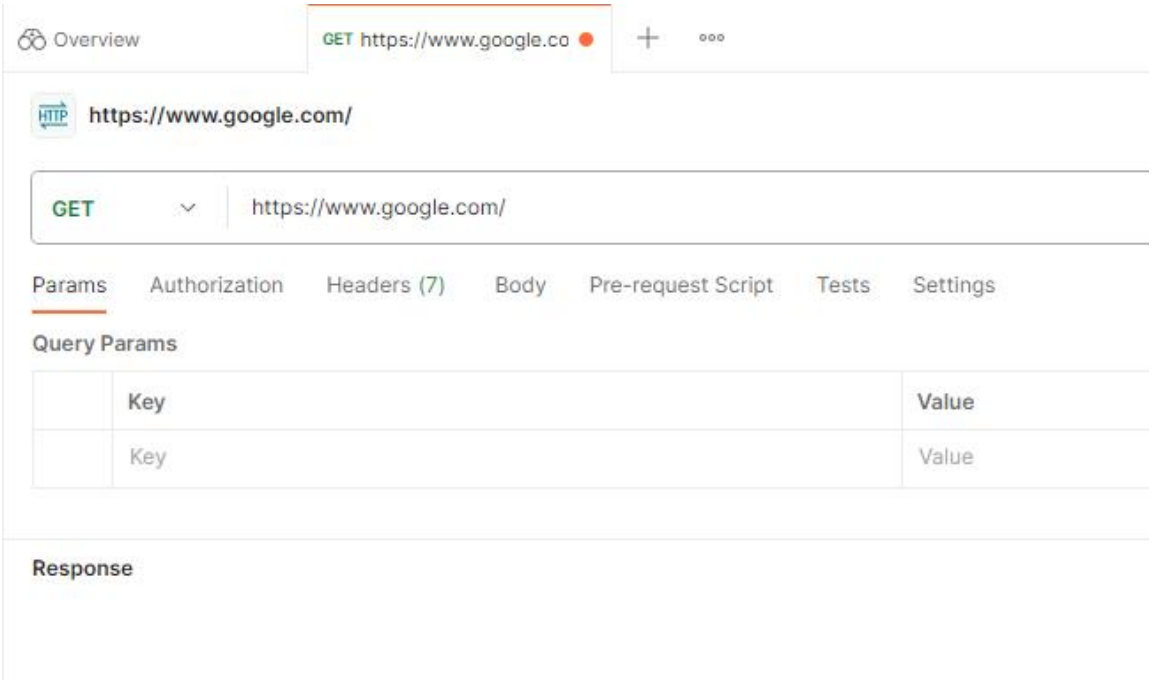

Рис. 3.20. Результат роботи дії введена кінцевої точки

Наступним кроком буде створення дії, яка відповідає за натискання клавіші виконати в додатку “Postman” (рис. 3.21).

```
public class ExecuteEndpointAction : ActionBase
{
    public override string Name
        => nameof(ExecuteEndpointAction);

    public ExecuteEndpointAction(
        IKeyboardApi keyboardApi,
        IVisualAnalyzer va,
        ILogger logger) : base(keyboardApi, va, logger)
    {
    }

    public override Task<bool> CanExecuteAsync()
        => VA.HasMarkerAsync("POSTMAN_HTTP_IS_ACTIVE");

    public override async Task<bool> ExecuteAsync()
    {
        await KeyboardApi.MoveMouseToAsync(new Point(1660, 122));
        await KeyboardApi.MouseLeftClickAsync();
        return true;
    }
}
```

Рис. 3.21. Програмний код дії виконання запита

Результат роботи даної дії наведено на рис. 3.22

Рис. 3.22. Результат роботи дії виконання запита

Отже, даний фреймворк працює коректно і відповідає поставленим вимогам та не вимагає додаткових виправлень.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

З розвитком науково-технічного прогресу важливу роль відіграє можливість безпечного виконання людьми своїх трудових обов'язків. У зв'язку з цим була створена і розвивається наука про безпеку праці.

Охорона праці - система законодавчих актів, соціально-економічних, організаційних, технічних, гігієнічних, лікувально профілактичних заходів, направлених на: забезпечення безпеки людини, збереження здоров'я та працездатності людини в процесі праці; розробку методів і засобів захисту шляхом зниження впливу шкідливих і небезпечних чинників до допустимих значень. Завдання ОП - звести до мінімуму вірогідність травмування або захворювання працюючого з одночасним забезпеченням комфорту для максимальної продуктивності праці [14].

На робочому місці, за яким співробітники компанії, які відповідають за якість продукту будуть використовувати дане програмне середовище повинні бути передбачені заходи захисту від можливої дії небезпечних і шкідливих чинників виробництва. Рівні цих чинників не повинні перевищувати граничних значень, обумовлених правовими, технічними і санітарно-технічними нормами. Ці нормативні документи зобов'язують до створення на робочому місці умов праці, при яких вплив небезпечних і шкідливих чинників на працюючих або усунений зовсім, або знаходиться в допустимих межах.

Найпоширенішими у процесі праці є поза сидяча і стояча. При організації робочого місця потрібно враховувати, що при виконанні роботи з фізичними навантаженнями бажаною є "стояча" поза, а при малих зусиллях "сидяча".

Під час роботи з комп'ютером найбільшому ризику піддаються зорова, опорно-рухова, нервово-психічна системи і репродуктивна функція у жінок (достовірно невідомо, що саме порушує її - випромінювання або постійна статична поза, але те, що вагітним жінкам слід уникати комп'ютера - безсумнівно).

Дисплей – головне джерело небезпеки. Він випускає випромінювання декількох видів: рентгенівське, ультрафіолетове, інфрачервоне, електромагнітне. Для кожного з цих випромінювань розроблені гранично допустимі норми, проте вони досить умовні й різняться у кожній країні. Норми передбачають, що опромінюється весь організм людини, тоді як на ділі впливу піддається лише верхня частина тулуба. Згадані норми встановлені з розрахунку на кожен вид опромінення в окремо, хоча реально всі поля діють одночасно, а їх комплексний вплив досі не досліджено.

Робота з монітором включає самі різні завдання, які об'єднуються такими загальними чинниками, як те, що робота проводиться в сидячому положенні і вимагає уважного, безперервного і тривалого спостереження.

Правильна установка робочого столу:

- при фіксованій висоті - краща висота - 72 см;
- повинен забезпечуватися необхідний простір для рук.

Правильна установка робочого стільця:

- висота повинна регулюватися;
- конструкція повинна бути такою, що обертається;
- правильна висота сидіння.

Правильна установка приладів: необхідно так встановити яскравість знаків і яскравість фону дисплея, щоб не існувало дуже великої відмінності в порівнянні з яскравістю навколишнього оточення, але щоб знаки чітко пізнавалися на відстані читання [15].

Не допускати:

- дуже велику яскравість (викликає мерехтіння);
- дуже слабку яскравість (сильне навантаження на очі);
- дуже чорну фонову яскравість дисплея (сильне навантаження на очі).

Правильне виконання робіт:

- положення тулуба пряме, ненапружене;
- положення голови пряме, вільне, зручне;
- положення рук - зігнуті трохи більше, ніж під прямим кутом;

- положення ніг - зігнуті трохи більше, ніж під прямим кутом;

- правильна відстань для зору, клавіатура і дисплей - приблизно на однаковій відстані для зору: при постійних роботах - близько 50 см, при випадкових роботах - до 70 см.

Правильне освітлення:

- освітлення по можливості з боку, зліва;

- по можливості - рівномірне освітлення всього робочого простору;

- прилади по можливості встановлювати в місцях, віддалених від вікон;

- вибирати непряме освітлення приміщення або вкривати корпуси світильників;

- світло, що поступає через вікна, пом'якшувати за допомогою штор;

- так організувати робоче місце, щоб напрям погляду йшов по можливості паралельно фронту вікон.

Правильне застосування допоміжних засобів: підлокітники використовувати, якщо клавіатура вище 1.5 см.; підставку для документів і опору для ніг.

Правильний метод роботи:

- передбачати по можливості зміну завдань і навантажень;

- дотримувати перерви в роботі: 5 хвилин через 1 годину роботи на дисплеї або 10 хвилин після 2-х годин роботи на дисплеї.

У створенні сприятливих умов для підвищення продуктивності і зменшення напруги значну роль грають чинники, що характеризують стан навколишнього середовища: мікроклімат приміщення, рівень шуму і освітлення. Рекомендована величина відносної вологості - 65-70%. Робоче місце повинне добре провітрюватися.

Раціональне освітлення приміщень - один з найбільш важливих чинників, від яких залежить ефективність трудової діяльності людини.

Хороше освітлення необхідне для виконання більшості завдань оператора. Для того, щоб спланувати раціональну систему освітлення, необхідно враховувати специфіку робочого завдання, для якого створюється система освітлення, швидкість і точність, з якою це робоче завдання повинне

виконуватися, тривалість його виконання і різні зміни в умовах виконання робочих операцій.

Вимоги до освітленості в приміщеннях, де встановлені комп'ютери, наступні: при виконанні зорових робіт високої точності загальна освітленість повинна складати 300 лк, а комбінована — 750 лк; аналогічні вимоги при виконанні робіт середньої точності — 200 і 300 лк відповідно.

Крім того, все поле зору повинне бути освітлене достатньо рівномірно – це основна гігієнічна вимога.

Сила струму. Зі зростанням сили струму небезпека ураження ним тіла людини зростає. Розрізняють порогові значення струму (при частоті 50 Гц):

- пороговий відчутний струм – 0,5-1,5 мА при змінному струмі і 5-7 мА при постійному струмі;

- пороговий невідпускний струм (струм, що викликає при проходженні через тіло людини нездоланні судомні скорочення м'язів руки, в котрій затиснений провідник) – 10-15 мА при змінному струмі і 50-80 мА при постійному струмі;

- пороговий фібриляційний струм (струм, що викликає при проходженні через організм фібриляцію серця) – 100 мА при змінному струмі і 300 мА при постійному струмі.

Тривалість проходження струму через організм істотно впливає на наслідок ураження: зі зростанням тривалості дії струму зростає ймовірність важкого або смертельного наслідку. Така залежність пояснюється тим, що зі зростанням часу впливу струму на живу тканину підвищується його значення, накопичуються наслідки впливу струму на організм. Зростає також імовірність співпадання моменту проходження струму через серце з уразливою фазою серцевого циклу (кардіоциклу).

Отже, небезпечні фактори погіршують умови праці надаючи шкідливу дію на організм людини. Наприклад, ті хто працюють в умовах тривалої шумової дії відчувають дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, пониження апетиту, болі у вухах і т.д. Такі порушення

в роботі ряду органів і систем організму людини можуть викликати негативні зміни в емоційному стані аж до стресових.

4.2 EMI-обстановка та її вплив на стійкість автоматизованих комп'ютерних систем в умовах надзвичайних ситуацій воєнного часу

Електромагнітний імпульс (EMI) став однією з ключових загроз для стійкості автоматизованих комп'ютерних систем в умовах надзвичайних ситуацій воєнного часу. У таких умовах, коли інтенсивність електромагнітних випромінювань може зростати експоненційно через використання великої кількості радіоелектронних засобів, автоматизовані системи стають вразливими перед потенційними EMI-впливами. Виникає необхідність вдосконалення заходів захисту та стійкості комп'ютерних систем.

Важливим аспектом є вивчення і розробка нових технологій, які дозволяють забезпечити стабільну роботу автоматизованих систем під час повітряних атак, використання радіочастотного засмічення або навіть ядерних вибухів. Стійкість до EMI-впливів має бути вбудована в архітектуру комп'ютерних систем від самого початку їх розробки, а також передбачати заходи захисту на рівні програмного забезпечення та жорсткого забезпечення.

Паралельно з цим, важливо розробляти стратегії відновлення та резервування, щоб забезпечити продовження функціонування систем в умовах надзвичайних ситуацій. Підвищення свідомості в галузі кібербезпеки та впровадження сучасних технологічних рішень може значно полегшити завдання забезпечення стійкості автоматизованих комп'ютерних систем в умовах електромагнітних загроз під час воєнних конфліктів.

Для забезпечення стійкості автоматизованих комп'ютерних систем в умовах надзвичайних ситуацій важливо враховувати не лише технічні аспекти, але й організаційні та методологічні [16]. Розвиток стандартів та нормативів з області

кібербезпеки дозволяє створити єдиний підхід до захисту інформаційних систем в умовах воєнного конфлікту. Крім того, навчання персоналу щодо протидії EMI-впливам та ефективного використання заходів захисту відіграє ключову роль у підвищенні загальної стійкості та резилієнтності комп'ютерних систем.

Застосування інноваційних технологій, таких як квантова криптографія та блокчейн, може сприяти забезпеченню конфіденційності та цілісності інформації під час військових дій. Також важливо розглядати можливість децентралізації систем, що дозволяє розподілити ризики та зменшити ймовірність впливу одного точкового впадку.

У кінцевому підсумку, вирішення проблеми стійкості автоматизованих комп'ютерних систем в умовах надзвичайних ситуацій воєнного часу потребує комплексного підходу, що враховує технічні, організаційні та соціально-економічні аспекти. Тільки такий підхід дозволить створити ефективні та надійні системи, здатні функціонувати в умовах великих викликів і загроз.

ВИСНОВКИ

Основні результати полягають у наступному:

1. На підставі детального аналізу електронних ресурсів виконано системний аналіз предметної області розробки розширеної версії тестового фреймворку для автоматизації сценаріїв використання на базі dotnet. Сформульовано та обґрунтовано мету проекту, визначено задачі, функції системи та структуру, а також встановлені вимоги до фреймворку.

2. Здійснено відбір ефективних технологій та інструментів для реалізації розширеної версії тестового фреймворку. Акцент був зроблений на використанні засобів dotnet, що дозволяє створювати ефективні та високопродуктивні рішення.

3. Описано математичне вирішення питань пов’язаних з розпізнавання тексту на скріншотах, а також алгоритм пошуку маркерів на скріншоті.

4. Виконано формалізацію процесу взаємодії та реакції тестових сценаріїв на дії користувачів..

5. Проведено докладний аналіз існуючих тестових фреймворків. Розглянуто приклади використання простих та ефективних тестових механізмів.

6. Здійснено успішну розробку та реалізацію розширеної версії тестового фреймворку. Проект позиціонується як інструмент для автоматизації сценаріїв використання на базі dotnet, спрямований на підвищення ефективності та швидкості тестування.

7. Як додаткові компоненти до програмного забезпечення було реалізовано сервер для автоматичного оновлення продуктів а також ланучер, який дозволяє програмам запускати додаток та підключатися до сервера для перевірки оновлень, завантаження та встановлення оновлення. Також є можливість отримати інформацію про додаток яку можна знайти на сайті розробника.

8. Створено тестові сценарії, які використовуються для розуміння того, як потрібно працювати з даним додатком.

9. Проведено верифікацію даного програмного продукту на основі описаних тестових скріптів.

ПЕРЕЛІК ПОСИЛАННЬ

1. Beginner’s Manual Software Testing Guide. URL: https://medium.com/techcompose/beginners-manual-software-testing-guide-8096bcd99a14

2. Automated Testing and Why You Should Use It. URL: https://betterprogramming.pub/automated-testing-why-you-should-use-it-28a1beda1bdd

3. Testing Plugins. URL: https://medium.com/pragmatic-programmers/testing-plugins-ced67e1274a

4. What is Modular Architecture? Benefits & Implementation Methods. URL: https://medium.com/geekculture/what-is-modular-architecture-benefits-implementation-methods-8c272ebc05eb

5. Robot Framework. URL: https://robotframework.org/

6. Fast and reliable end-to-end testing for modern web apps | Playwright. URL: https://playwright.dev/

7. Selenium. URL: https://www.selenium.dev/

8. A Short Introduction to the Scrum Framework. URL: https://medium.com/chingu/a-short-introduction-to-the-scrum-methodology-7a23431b9f17

9. Introduction to Computer Vision & OpenCV in Python. URL: https://medium.com/analytics-vidhya/introduction-to-computer-vision-opencv-in-python-fb722e805e8b

10. A comprehensive guide to OCR with Tesseract, OpenCV and Python. URL: https://medium.com/nanonets/a-comprehensive-guide-to-ocr-with-tesseract-opencv-and-python-fd42f69e8ca8

11. .Net Core Benefits: Why It is the most Desirable in 2022. URL: https://medium.com/geekculture/net-core-benefits-why-it-is-the-most-desirable-in-2022-73a90d7ab2a6

12. Emgu CV. URL: https://www.emgu.com/wiki/index.php/Main_Page

13. ID card Border detection using Emgu CV. URL: https://medium.com/peritos-solutions/id-card-border-detection-using-emgu-cv-59aa1114397d

14. Закон України «Про охорону праці». URL: http://zakon3.rada.gov.ua/laws/show/2694-12 (дата звернення: 18.12.2023).

15. Наказ України «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text (дата звернення: 18.12.2023).

16. Стручок В.С. Техноекологія та цивільна безпека. Частина "Цивільна безпека": Навчальний посібник; Автор-укладач: Стручок В.С.. — Тернопіль: ФОП Паляниця В.А., 2022. — 156 с. (дата звернення: 18.12.2023).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

УДК 004.9
В.В. Хацюр, Д.М. Михалик, канд. техн. наук, доцент
Тернопільський національний технічний університет імені Івана Пулюя

ПЕРЕВАГИ ВИКОРИСТАННЯ АРХІТЕКТУРИ МОДУЛЬНОГО МОНОЛІТА

V.V. Khatsiur, D.M. Mykhalyk PhD, Assoc. Prof.

BENEFITS OF USING MODULAR MONOLITH ARCHITECTURE

Сучасний світ програмування вимагає найефективніших та найбільш оптимальних рішень у сфері розробки програмного забезпечення. Одним із таких рішень є використання архітектури модульного моноліта, яка привнесла ряд переваг у порівнянні з іншими підходами.

Основна перевага модульного моноліта - це його здатність спростити процес розробки та управління кодом. Модулізація розвитку програми дозволяє розробникам працювати над конкретними частинами системи, не займаючись одночасно всією структурою [1]. Це полегшує взаємодію між командами та сприяє швидкому розвитку продукту.

Модульний моноліт дозволяє вирішувати проблему складності інтеграції різних компонентів, яка є актуальною для розподілених систем. Усі модулі об'єднуються в один цілий, що спрощує процес тестування та впровадження змін. Це дозволяє швидше реагувати на зміни вимог та впроваджувати новий функціонал.

Архітектура модульного моноліта сприяє оптимізації продуктивності, оскільки дозволяє краще управляти ресурсами системи. За рахунок однорідності та стандартизації внутрішніх компонентів, вдається досягти кращої ефективності роботи програми та зменшити ймовірність виникнення помилок.

Масштабування системи на основі модульного моноліта є більш ефективним порівняно з деякими іншими архітектурними підходами. Завдяки його структурі, де модулі легко замінюються та оновлюються, масштабування стає більш гнучким та зручним.

Використання модульного моноліта також полегшує управління версіями програмного забезпечення. При внесенні змін в один модуль необхідно тільки оновити частину програми, що відповідає за цей модуль, без необхідності вносити поправки у всю систему. Це дозволяє швидше випускати оновлення та реагувати на зміни вимог.

Архітектура модульного моноліта сприяє покращенню безпеки програмного забезпечення. Через стандартизацію та гомогенність внутрішніх модулів, можна ефективно впроваджувати та керувати заходами безпеки для всієї системи, забезпечуючи більш високий рівень захисту від потенційних загроз.

За всією своєю простотою, архітектура модульного моноліта виявляється потужним інструментом для розробки великих та складних програмних систем. Вона не лише сприяє швидкому розвитку, але і забезпечує ефективну управлінську модель, зменшуючи витрати часу та ресурсів. Таким чином, вибір модульного моноліта може бути ключовим кроком у побудові успішної та ефективної розробки програмного забезпечення.

Література:

1. What is Modular Monolith? [Електронний ресурс]. – Режим доступу: https://www.ahmetkucukoglu.com/en/what-is-modular-monolith