ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи
на ступінь вищої освіти бакалавр
на тему: «**РОЗРОБКА ДЕСКТОП ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ НА МОВІ ПРОГРАМУВАННЯ JS**»

Виконав: студент 4 курсу, групи ПД– 41
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Слободяник В.В.
(прізвище та ініціали)
Керівник Негоденко О.В.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)
Нормоконтроль _____
(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
О.В. Негоденко

“_____” _____2021 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Слободянику Віталію Валерійовичу
(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка десктоп додатку для автоматизації тестування на мові програмування JS»

Керівник роботи зав. кафедри ІПЗ Негоденко О.В. ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “12” березня 2021 року №65.

2. Строк подання студентом роботи 01.06.2021.
3. Вихідні дані до роботи:
   - 3.1. Положення побудови систем автоматизації тестування ПЗ;
   - 3.2. Методи побудови систем автоматизації тестування ПЗ;
   - 3.3. Існуючі інструменти автоматизації тестування ПЗ;
   - 3.4. Науково-технічна література;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   - 4.1. Огляд архітектури існуючих інструментів автоматизації тестування
   - 4.2. Розробка структури додатку для автоматизації тестування ПЗ на JS

4.3. Програмна реалізація додатку
4.4. Приклади використання та тестування системи
4.5. Висновки

5. Перелік графічного матеріалу.
   5.1. Титульний слайд
   5.2. Мета, об’єкт та предмет дослідження
   5.3. Актуальність роботи
   5.4. Аналоги
   5.5. Порівняння з аналогами
   5.6. Технічне завдання
   5.7. Програмні засоби реалізації
   5.8. Інструменти використані для реалізації
   5.9. Архітектура браузерного розширення
   5.10. Архітектура плеєра тестових сценаріїв
   5.11. Апробація результатів дослідження
   5.12. Висновки

6. Дата видачі завдання: 19.04.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04.2021	Виконано
2	Дослідження існуючих інструментів для автоматизації тестування ПЗ	20.04.2021	Виконано
3	Проектування архітектури системи	21.04.2021 – 22.04.2021	Виконано
4	Розробка системи для автоматизації тестування ПЗ	23.04.2021 – 27.04.2021	Виконано
5	Висновки, оформлення роботи	27.04.2021 – 30.04.2021	Виконано
6	Розробка демонстраційних матеріалів	30.04.2021	Виконано
7	Попередній захист роботи	15.05.2021	
8	Здача роботи	01.06.2021	

Студент _____ Слободяник В.В.
(підпис) (прізвище та ініціали)

Керівник роботи_____ Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 64 с., 43 рис., 18 джерел.

CHROME EXTENSIONS, JAVASCRIPT, NODE JS, ELECTRON JS, SELENIUM WEB DRIVER , HTML, CSS

Об'єкт дослідження – автоматизація процесів ручного тестування програмного забезпечення для підвищення ефективності роботи тестувальника програмного забезпечення та збільшення його продуктивності в сфері веб–тестування .

Предмет дослідження – технологія розробки програмного забезпечення для запису та відтворення дій користувача на веб–сторінці.

Мета роботи – розробка програмного забезпечення для забезпечення контролю якості веб–сервісів та веб–додатків.

Методи дослідження – методи теорії інформації, методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування, валідації та верифікації програмного забезпечення.

Наукова новизна даної роботи полягає в наступному:

1. Розроблено та реалізовано унікальний алгоритм зчитування дій користувача з веб–сторінки.
2. Розроблено протокол взаємодії браузерного розширення з десктопним додатком.
3. Розроблено систему запису подій, які відбулись на веб сторінці та їх розшифрування.
4. Інтегровано інструмент автоматизації тестування SELENIUM WEB DRIVER у клієнтську частину.

В роботі виконано аналіз задач автоматизації тестування програмного забезпечення; аналіз і моделювання архітектури взаємодії браузерного розширення та десктоп додатку з інтеграцією SELENIUM WEB DRIVER. Проаналізовано можливості платформ CHROME EXTENSIONS, ELECTRON JS, засобів розробки і відлагодження програм на JS. Розроблено структуру бази

даних системи, програмне забезпечення для запису та відтворення тестових сценаріїв на веб–сторінках, виконано налагодження та випробування розробленої системи.

Галузь використання – контроль та автоматизація забезпечення якості програмного забезпечення.

ЗМІСТ

ВСТУП

Актуальність теми

Забезпечення якості – це дуже відповідальний та трудомісткий процес, без якого зараз не можливо уявити цикл розробки програмного забезпечення. Та завжди існувала проблема вибору підходу до тестування продукту. Два основні напрями в сфері тестування є мануальне та автоматизоване тестування. Кожен тип тестування має свої недоліки та переваги. Наприклад, мануальне тестування при порівнянні з автоматизованим є в середньому до 4– х разів повільніше, та не гарантує 100% якості, оскільки роботу виконують люди, що дає шанс виникнення помилки, але дозволяє протестувати складні бізнес процеси продукту. Автоматизоване ж тестування дозволяє швидше пройти повторне тестування – регресивні тести, покрити певні нефункціональні тести продукту – такі як продуктивність, навантажувальне тестування, стрес тести. Але при частих змінах функціоналу, написані скрипти також постійно потрібно модифікувати, що тягне за собою надлишкові фінансові та часові витрати у проекті.

Індустрія тестування програмного забезпечення завжди потребувала інструменту, який дозволив б швидко та ефективно згенерувати автоматизовані тестові сценарії за допомогою запису дій користувача в додатку. Це дозволить економити час на написанні скриптів, з легкістю їх переписувати та використовувати у подальших потребах спеціаліста – автоматизатора. Другий інструмент, який потрібен для мануальних спеціалістів – являється інструментом відтворення тестових сценаріїв, що дозволить не витрачати велику кількість часу на виконання тестів, мінімізує вплив людського фактору на якість проведеного контролю, та дасть можливість швидко змінювати потрібні сценарії. Ще одним важливим фактором є авто аналіз виконаного тесту – система повинна показувати статистику виконаних тестів, їх статус та аналітику процесу. Тому можемо стверджувати, що розробка інструменту для запису та відтворення тестових сценаріїв є дуже актуальною в наш час, оскільки це дозволить вирішити

низку проблем, з якими стикаються як тестувальники–автоматизатори так і мануальні спеціалісти.

Об'єкт дослідження – технології розробки інструментів запису та відтворення дій користувача на веб–сторінці.

Предмет дослідження – технологія розробки програмного забезпечення для запису та відтворення дій користувача на веб–сторінці.

Мета роботи – розробка програмного забезпечення для забезпечення якості веб–сервісів та веб–додатків.

У відповідності до поставленої мети визначено основні завдання дослідження:

– проаналізувати поточний стан задач для автоматизації тестування програмного забезпечення.

– дослідити можливості використання платформи Chrome Extensions, бібліотеки Selenium Web Driver засоби розробки і відлагодження програм на Java Script;

– встановити відповідність вимогам об'єктно-орієнтованого аналізу для розробки системи запису та відтворення тестових сценаріїв;

– розробити структури бази даних системи;

– розробити архітектуру для запису тестових сценаріїв з веб сторінки та їх подальшої обробки.

– розробити програмне забезпечення для запису та відтворення тестових сценаріїв.

Методи дослідження – методи теорії інформації, методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування, валідації та верифікації програмного забезпечення.

Наукова новизна даної роботи полягає в наступному:

1. Розроблено та реалізовано унікальний алгоритм зчитування дій користувача з веб–сторінки.
2. Розроблено протокол взаємодії браузерного розширення з десктопним додатком.

3. Розроблено систему запису подій, які відбулись на веб сторінці та їх розшифрування.
4. Інтегровано інструмент автоматизації тестування SELENIUM WEB DRIVER у клієнтську частину.

Практична значущість результатів дослідження полягає у вирішенні важливої практичної задачі – розробки програмного забезпечення для запису та відтворення тестових сценаріїв, що дозволяє використовувати нестандартний підхід до тестування якості програмного забезпечення, економити час витрачений на створення та виконання тестових сценаріїв.

Результати дослідження бакалаврської роботи апробовані на всеукраїнських науково-технічних конференціях: «Застосування програмного забезпечення в інфокомунікаційних технологіях» та «Сучасний стан та перспективи розвитку IOT».

1. ОГЛЯД АРХІТЕКТУРИ ІСНУЮЧИХ ІНСТРУМЕНТІВ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ

1.1. Переваги засобів автоматизації тестування над ручним тестуванням

На початку розвитку індустрії тестування програмного забезпечення було повністю мануальним. Повторюваний характер тестування, однотипність завдань та час, необхідний для тестування, показали що ринок потребує інструментів, які б могли виконувати тести замість спеціаліста.

Проблеми регресії – найболючіші проблеми. Ми люди. І ми не можемо робити одне і те ж з однаковою енергією, швидкістю та точністю щодня. Це те, що властиво машинам. Для цього необхідна автоматизація, щоб повторювати ті самі дії з однаковою швидкістю, точністю та ефективністю.

Автоматизація тестів допомагає спеціалістам зосереджуватись на тестуванні нових функціональних можливостей, одночасно покриваючи регресивне тестування системи. Згідно до досліджень ISTQB, автоматизоване тестування скорочує час виконання тестів до 90%.

Сценарій заповнить усі поля та покаже результат запуску разом із скріншотами. У разі відмови він може точно визначити місце, де тест не вдався, і таким чином допоможе вам легко його відтворити.

Автоматизація – економічно ефективний метод регресійного тестування. Спочатку витрати на автоматизацію дійсно вищі. Вони включають вартість інструменту, вартість персоналу для автоматизації тестування та його навчання.

Але коли сценарії готові, їх можна виконувати багаторазово з однаковою точністю і швидкістю. Це заощаджує багато робочого часу спеціалістів. Тому вартість впровадження автоматизації поступово зменшується, і в кінцевому рахунку це стає економічно ефективним методом регресійного тестування.

Завдяки автоматизації тестувальники можуть автоматизувати свої повторювані завдання та зосередитись на інших тестових завданнях, таких як

вибір правильних тестових сценаріїв для тестового запуску та тестування нових функцій.

Перевагою автоматизованого тестування було зменшення часу на тестування однакових сценаріїв тестування, тоді як недоліком є те, що автоматизація тестування передбачає використання скриптів. Спеціалістам потрібно вивчати підтримувані мови для автоматизації, або наймати нові ресурси, які мають навички програмування та знають необхідні інструменти.

Поширені проблеми під час використання автоматизованого тестування на основі сценаріїв:

- Потрібно вивчити підтримувані мови сценаріїв або найняти нові ресурси, які вже знають цю мову;
- Великі початкові інвестиції з точки зору налаштування засобів автоматизованого тестування та призначення ресурсів для автоматизації;
- Обслуговування автоматизованих тест– кейсів також стає високооб'ємним завданням;

Для вирішення вищезазначених проблем зараз на ринку існує кілька засобів автоматизованого тестування, які не потребують сценаріїв. І якщо вірити прогнозам, все більше компаній збираються застосовувати автоматичні засоби тестування без скриптів.

Одним з найбільш популярних підходів до безскриптового автоматизованого тестування являються Інструменти «запису та відтворення» або, так звані «Record–playback tools», тому що вони не вимагають навичок програмування.

Багато підприємств зосередили свою увагу на безперервному тестуванні, використовуючи автоматизоване тестування, щоб уникнути критичних бізнес–проблем. Крім того, для досягнення конкурентних переваг їм потрібно створювати свій продукт / додаток у постійному стані готовності до випуску.

При розробці додатків або в життєвому циклі розробки програмного

забезпечення існує безліч сценаріїв та тригерних точок, які вимагають застосування автоматизації тестів для наскрізного тестування.
Наприклад, рішення робототехніки та ботів надзвичайно важливо протестувати якісно, щоб переконатися, що продуктивність відповідає очікуванням. Або ж це може призвести до руйнівного провалу у використанні цієї технології.

Автоматизація тестів дозволяє повторно використовувати сценарії тестів та максимально використовувати покриття тесту, щоб пришвидшити процес тестування з точністю та надійністю. Це зменшує зусилля людини і одночасно гарантує, що додаток працюватиме належним чином, як очікувалося на стадії передачі кінцевому користувачу.

Доступні різні інструменти автоматичного запису та відтворення тестування, завдяки яким полегшується створення тесту. Це розширення браузера / плагіни, що використовується для запису заздалегідь визначених дій, які виконують тестувальники під час взаємодії з веб– додатком. Тоді як функція відтворення автоматично генерує попередньо записані дії, щоб врешті порівняти результати з очікуваним результатом.

Цей інструмент реєструє кожну дію тестера, наприклад, наведення миші, знімки екрана та натискання клавіш під час виконання тест– кейса вручну. Коли тест записаний, його можливо запускати заново, коли потрібно зробити регресійне тестування системи. Простіше кажучи, цей інструмент дозволяє виконувати одні й ті самі сценарії кілька разів.

Ось причини, чому слід розглянути можливість записування та відтворення для автоматичного тестування:

Інструменти запису і відтворення – це безпроблемний спосіб автоматизувати процеси ручного тестування в найпростішій формі. Вони не вимагають знань програмування. Ви можете виконати ручне тестування без необхідності писати довгі сценарії, і функція запису буде реєструвати ці дії.

Цей інструмент дуже ефективний для тестувальників, які перебувають на етапі навчання; без необхідності боротися зі складними фреймворками та командами.

Автоматизація запису та відтворення дозволяє створювати, редагувати та імпортувати записані сценарії, а також відстежувати процеси тестування. Легко перезаписати або відредагувати частину записаних сценаріїв, де б ви не виявили помилку під час процесу запису.

Користувачі інструменту запису та відтворення можуть мати доступ до ряду конфігурацій браузера для тестування в різних браузерах. Незалежно від того, чи це найновіший випуск Chrome або IE8, у вас є готові ресурси для перевірки функціональності, коли це потрібно.

Інструменти запису та відтворення корисні в процесі початкового тестування програми. Однак це розмиває роль спеціальної команди з контролю якості. Організаціям більше не потрібно покладатися на спеціальну команду тестувальників, щоб протестувати невеликі області програми.

Раніше автоматизацію тестування проводили лише інженери автоматизації, але тепер кожен може виконати початкове тестування за допомогою даних інструментів, будь то розробник або менеджер; вся команда може взяти участь у процесі випробувань. Це зменшує навантаження спеціальної команди з тестування та дозволяє зосередитись на інших важливих тестових заходах.

Ось чому інструменти запису і відтворення в тестовій автоматизації мають вирішальне значення для розробки додатків та циклу розробки програмного забезпечення. Це прискорює процес тестування програмного забезпечення, а також робить його більш ефективним та орієнтованим на результат.

1.2. Огляд існуючих інструментів автоматизації тестування ПЗ

– Інструменти запису та відтворення;

Ці інструменти корисні для сценаріїв, які не змінюються протягом тривалого часу, але якщо тестована програма нестабільна та підвержена частим змінам, запуск тестів може бути неуспішнм.

Таким чином, основним недоліком використання засобів тестування записів та відтворення є те, що вони строго визначені у тому, що вони записують і відтворюють. Отже, можливість налаштування записаного тестового випадку обмежена. Деякі інструменти, такі як Selenium IDE та Katalon IDE, надають можливість перетворити записані тестові кейси на скрипти на бажаних мовах, але для редагування цих тестів потрібні навички програмування.

– NLP;

Такі інструменти, як Testsigma, використовують NLP (Natural Language Processing) для створення тестових сценаріїв. Ці інструменти мають низький рівень входження, оскільки автоматизовані тести написані простою природною мовою. Користувачеві просто необхідно знати правильну граматику, яка використовується для розробки тестів, і використовувати її для автоматизації простих або складних сценаріїв тестування, залежно від конкретного випадку.

– Тестування на основі моделі;

TOSCA – це популярний інструмент, який використовує тестування на основі моделі. Цей інструмент автоматично розпізнає всі об’єкти веб– сайту. Логіка тестового сценарію, дані тесту та тестові об'єкти зберігаються окремо і об'єднуються під час виконання тесту. Центральна модель, що має об'єкти, оновлюється сама, якщо будь– які зміни зустрічаються в будь– якому елементі програми.

– Інструменти запису знімків екрану;

Існують інструменти, які використовують підхід запису знімків екрану та використовують ШІ для оновлення тестових сценаріїв, коли будь– які зміни трапляються в інтерфейсі тестованої програми.

Вони записують сценарій у вигляді знімків екрану, зроблених кожного разу, коли користувач виконує дію. Тест вважається неуспішним, якщо в будь– який час тестовий кейс не відповідає зробленим знімкам екрана. Інструмент спрямований на підтримку тестових випадків, використовуючи ШІ для розпізнавання будь– яких змін у розташуванні, назві чи зовнішньому вигляді елемента веб– сайту.

– Тестування на основі об'єктного зіставлення;

Деякі інструменти пропонують автоматичну автоматизацію тестування, створюючи об'єкти шляхом безпосереднього копіювання з веб– сайту. Потім ці об’єкти можна використовувати повторно скрізь, де це потрібно для створення тестових скриптів.

– Тестування на основі ключових слів;

Ці інструменти використовують ключові слова як дії, які слід виконувати над тестовими об’єктами. Це робить створення тестових сценаріїв простим та ефективним без скриптів.

– Візуальне тестування за допомогою блок– схем;

Codefuse надає користувачам можливість створювати тести у вигляді блок– схем, кожен компонент на блок– схемі відповідає дії на веб– сайті, а підхід до тестування не має сценаріїв.

– Запис тестів як сценаріїв;

Це традиційний метод тестування, який використовується Selenium IDE, і Katalon IDE. Тестовий сценарій записує всі дії користувача у вигляді скрипта, який містить у собі всі дії у вигляді команд та деталей селектора. Потім ці тестові приклади можна експортувати вибраною мовою та, якщо потрібно, розширити цією мовою.

– AI– боти для автоматизації тестів;

Вони усувають необхідність будь– яких сценаріїв для автоматизації тестів. На ринку доступно кілька інструментів, які навчають ботів поводитися за зразком наданих вхідних даних. Це навчання є безперервним процесом, але також не передбачає жодних складних навичок у програмуванні.

1.3. Огляд технології Chrome Extensions

Браузер Google Chrome має багатопроцесну архітектуру, це означає, що кожна відкрита вкладка – це інший процес. У нас є процес браузера, з яким

взаємодіє кожен процес вкладки. Щоразу, коли вкладка виходить з ладу, решта вкладки браузера залишаються незмінними.

Основні компоненти браузера Chrome:

– Ядро;
– Рендерер сторінок;
– Плагіни;

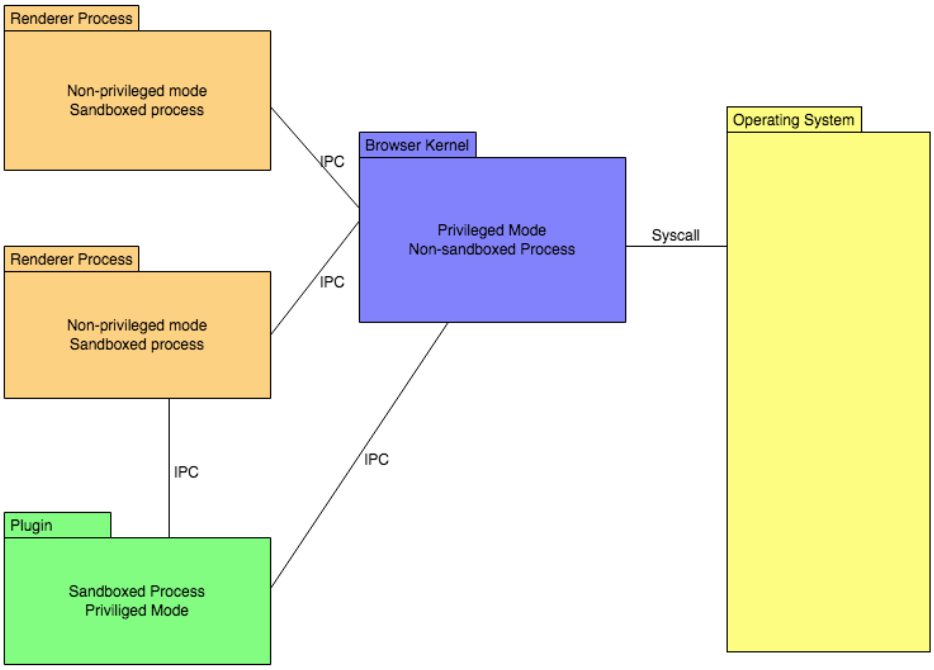


Рис. 1.1. – Структура браузера веб–сторінок

Ядро браузера – це основний процес, який координує процеси візуалізації, які в основному є процесами вкладки. Отже, існує один процес ядра браузера та кілька процесів візуалізації, кожен з яких відповідає вкладці. Кожен процес Renderer "ізольований", таким чином, вони мають обмежені привілеї, такі як обмежений доступ до системних викликів ОС, файлової системи тощо.

Процес візуалізації відповідає за аналіз HTML– CSS, побудову DOM, екземпляра V8 (виконання JavaScript) для вкладки. Він не працює з привілеєм користувача. Це запобігає доступу компрометованого процесу візуалізації до файлової системи користувача. Він обробляє веб–сторінку у растрові зображення (деякий проміжний формат) і відправляє її в ядро, яке відображає веб–сторінку на

дисплей (Рис 1.1.).

Ядро браузера відповідає за всі привілейовані операції, такі як мережа, файлова система, системні виклики ОС тощо. Щоб вкладка (Процес візуалізації) виконувала будь–яку привілейовану операцію користувача, наприклад, завантаження файлів (доступ до файлової системи), вона делегує операцію до ядра браузера через IPC (взаємодія між процесами). Ядро браузера задовольняє привілейовані потреби та абстрагує ОС для процесу візуалізації. Він також відповідає за підтримку постійних об'єктів зберігання, таких як зберігання файлів cookie, локальне сховище, indexedDB тощо.

Плагіни не слід сприймати як розширення Chrome. Наприклад, Adobe Flash – це плагін. Читач pdf – це плагін. В основному це нові технології, такі як Flash Player, розроблені сторонніми розробниками, які можна вважати об’єднаними в ядро браузера. Вони є сторонніми програмами, і вони працюють в іншому процесі з привілеями користувача. Така архітектура реалізована для захисту ядра від пошкодженого чи зловмисного плагіна.

Ядро браузера надає API для процесів візуалізації в ізольованому середовищі для виконання привілейованих операцій. Воно управляє життєвим циклом процесів візуалізації.

Процес візуалізації бере файли HTML, CSS, JS і готує їх для ядра для їх відображення.

Розширення – це додаткова функція, призначена для підключення до вашого веб–браузера та функціонування так, ніби це вбудована частина програмного забезпечення браузера. Провідні браузери відрізняються тим, як вони використовують термінологію щодо розширень:

Firefox називає їх розширеннями або плагінами залежно від їх ролі в браузері.

Internet Explorer називає їх доповненнями, які вони поділяють на п’ять типів доповнень: панелі інструментів, розширення, постачальники пошуку, прискорювачі та захист відстеження.

Chrome просто називає їх розширеннями, теми та програми класифікуються

окремо.

У Chrome, як і в інших браузерах, розширення працюють поряд із вбудованими функціями браузера, щоб покращити продуктивність. Оскільки кожне розширення має певне призначення, можливо, ви виконуєте певне завдання в браузері, перш ніж побачите розширення в дії. Наприклад, якщо ви використовуєте розширення Shopping Assistant, рядок порівняння цін, який здійснює пошук того самого товару на інших веб– сайтах, не відображатиметься внизу вікна браузера, якщо ви не купуєте товар у магазині конкурента.

Хоча розширення часто можна вважати подібними в браузерах, Chrome має унікальну опцію, яку інші браузери ще не підтримують: програми. Раніше ми говорили про одну з основних цілей Google для Chrome: Оптимізувати браузер для запуску веб–додатків. Сьогодні Chrome використовує власне віртуальне обчислювальне середовище для веб– програм, і додатки Chrome користуються цією обробною потужністю. Програми Chrome встановлюються та працюють у Chrome так, ніби ви встановлюєте програмне забезпечення на свій комп’ютер або смартфон.

На кожній сторінці нової вкладки Chrome відображає ярлики всіх встановлених вами програм.

Google Chrome став лідером за кількістю користувачів, і багато з нас вже використовували певне розширення, розроблене для нього. Від простої теми до розширення Netflix.

Розширення Chrome працюють із трьома окремими областями, які взаємодіють один з одним: фоном, вмістом та внутрішніми сценаріями. Кожен із них має певну функцію на цій сторінці.

Скрипт вмісту діє в області, яку відвідує користувач, і може служити для передачі інформації про поточний стан сторінки. З нього можна отримати будь–яку інформацію та надіслати її на розширення, щоб змінити його поведінку.

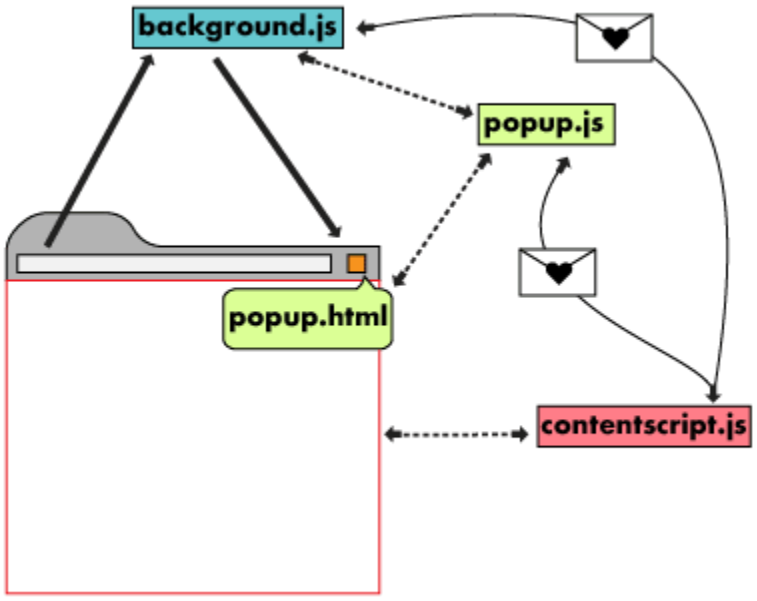


Рис. 1.2. – Схема роботи браузерного розширення

Фоновий скрипт діє в області браузера, він є обробником подій розширення. Тут зберігаються всі прослуховувачі подій, важливі для розширення. Він залишається неактивним, доки подія не запускається і не виконує логіку, яка їй була призначена.

Нарешті, внутрішній скрипт (Рис. 1.2.) відповідає за візуальні та функціональні можливості розширення, взаємодіючи безпосередньо з HTML розширення і разом із фоновим скриптом може виконувати, наприклад, виклики до API, доступ до користувацьких даних, тощо.

Основний файл в розширенні називатися manifest.json. Його структура повинна бути такою:

```
{
  "name": "Creating my first extension",
  "version": "1.0",
  "description": "Build an Extension!",
  "background": {
    "scripts": ["scripts/background.js"],
    "persistent": false
  },
  "content_scripts": {
```

```
    "js": ["scripts/content.js"]
  },
  "permissions": ["storage"],
  "browser_action": {
    "default_icon": "favicon.png",
    "default_popup": "./src/extension/popup.html"
  }
  "manifest_version": 2
}
```

Давайте переглянемо розділи цього файлу та зрозуміємо кожен з них. На додаток до основної інформації, такої як назва та версія розширення, ми також бачимо два сценарії, фоновий та вмісту, позначені відповідними шляхами.

За допомогою розширень ми також можемо попросити Chrome дозволи на збір різної інформації браузера. Для JSON, який ми створюємо, потрібні лише дозволи на зберігання, але ми також можемо додати нові функції, такі як мікрофони, камери, локації тощо

Процес браузера відповідає за візуальні функції, такі як основний файл, який буде служити спливаючим вікном для вашого розширення, та його піктограма, яка з'явиться, коли користувач встановить його у браузері.

1.4. Розробка десктоп додатків за допомогою мови JS

Створення настільних додатків не так давно вважалося важким, оскільки вам довелося вивчити спеціалізовану мову програмування, таку як Java або C++.

На щастя, тепер веб–розробники можуть створювати чудові настільні програми, використовуючи інструменти, які перетворюють ваш код JavaScript у діючі програми.

Щоб створювати настільні програми як веб– розробник, вам слід зрозуміти основні, найбільш доступні рішення, щоб не витрачати час на

інструменти, які не можуть забезпечити вас необхідним. Ось розбивка 3 найкращих фреймворків для створення додатків для настільних ПК, щоб дати вам уявлення про них:

ELECTRON JS – цей інструмент ми будемо використовувати, оскільки він найпопулярніший і найпотужніший. Він існує роками, його використовують багато компаній, включаючи слабину, атом, розбрат, інтуїцію та багато інших.

Він був створений командою GitHub, тому він має сильну репутацію та код якості. Ви можете бути впевнені, що ця структура буде продовжувати рости та домінувати над більшою кількістю проектів у майбутньому.

Вони використовують хромований механізм, що означає, що ваш додаток буде працювати надзвичайно швидко незалежно від використовуваного комп'ютера.

За допомогою нього ви можете швидко і легко створювати продукт, оскільки він використовує JavaScript, HTML і CSS, що чудово підходить для багатьох веб–розробників, які хочуть розширити свій набір навичок, маючи можливість розробляти для настільних платформ.

NW.JS – ще один чудовий продукт, але з набагато меншою популярністю. Це дозволяє створювати настільні програми за допомогою node.js. Він був відомий як движок Node–webkit, оскільки використовує ту саму технологію, яку використовують такі браузери, як Chrome та Safari.

Для тих, хто знайомий з React та React Native, Proton Native є чудовим вибором, оскільки він використовує фреймворк React, дозволяючи вам використовувати всі наявні там бібліотеки реагування.

Він має навіть кращу продуктивність, ніж електрон, при вирішенні конкретних завдань, оскільки вони реалізують власний двигун, щоб ви могли очікувати майже рідної продуктивності.

Чудово підходить для тих, хто знайомий з реакцією, яка хоче отримати більший контроль за рахунок тривалого часу розробки.

Тепер, коли ви знаєте найкращі варіанти створення настільних додатків як веб– розробник, давайте почнемо відразу з Electron, оскільки це найкращий

варіант для початківців та для більшості випадків використання.

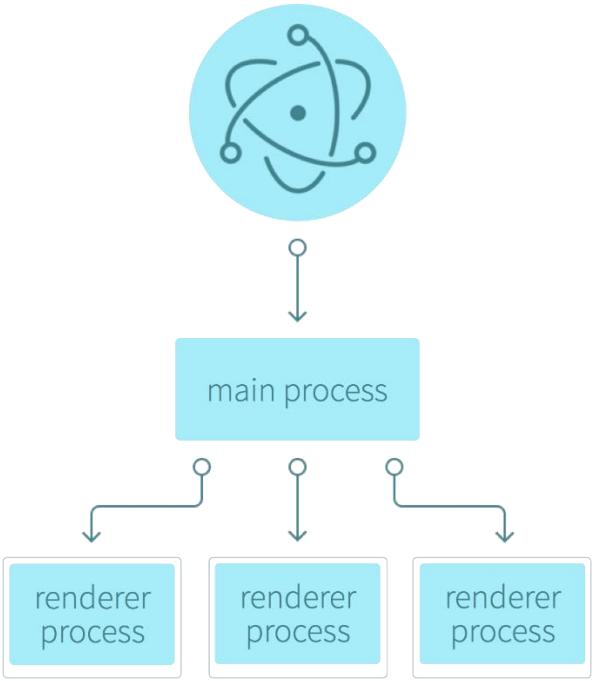


Рис. 1.3. – Архітектура бібліотеки ELECTRON JS

З тих пір, як браузери та мобільні пристрої стали достатньо потужними, спостерігається постійне падіння популярності настільних програм, які замінюються мобільними та веб– програмами. Тим не менш, є багато переваг для написання настільних програм – вони завжди присутні в системі, в основному краще взаємодіють з операційною системою (із ярликами, сповіщеннями, тощо), ніж веб– програми.

Основна ідея розробки настільних додатків з JavaScript полягає в тому, що ви створюєте одну базу коду та упаковуєте її для кожної операційної системи окремо. Це абстрагує знання, необхідні для створення власних настільних додатків, і полегшує їх обслуговування. В наш час розробка настільного додатка з JavaScript можлива за допомогою бібліотек Electron або NW.js. Хоча обидва інструменти пропонують більш– менш однакові функції, я обрав Electron, оскільки він має деякі переваги, які я знайшов важливими (Рис 1.3.).

Electron забезпечує рішення для створення настільних додатків із чистим JavaScript. Принцип роботи – Electron бере основний файл, визначений у файлі

package.json, і виконує його. Потім цей основний файл (як правило, називається main.js) створює вікна програм, які містять відтворені веб– сторінки з доданим функціоналом взаємодії та власним графічним інтерфейсом (графічним інтерфейсом користувача) вашої операційної системи.

Докладніше, після запуску програми за допомогою Electron створюється основний процес. Цей основний процес відповідає за взаємодію з власним графічним інтерфейсом вашої операційної системи та створює графічний інтерфейс вашої програми (вікна вашого додатка).

Просто запуск основного процесу не дає користувачам вашої програми доступу до її графічного інтерфейсу. Вони створюються основним процесом у головному файлі за допомогою модуля BrowserWindow. Потім кожне вікно браузера запускає власний процес візуалізації. Цей процес візуалізації бере веб– сторінку (файл HTML, який посилається на звичайні файли CSS, файли JavaScript, зображення тощо) і відображає її у вікні. Веб– сторінки відображаються за допомогою ядра Chromium, тому гарантується дуже високий рівень сумісності зі стандартами (Рис 1.3).

Наприклад, якщо у вас була лише програма для калькулятора, ваш основний процес створив би вікно із веб– сторінкою, де знаходиться ваша фактична веб– сторінка (калькулятор).

Основний процес може отримати доступ до власного графічного інтерфейсу через серію модулів, доступних безпосередньо в Electron. Ваша настільна програма може отримати доступ до всіх модулів NodeJS, на якому саме і працює бібліотека, таких як модуль нотифікацій, щоб показати системні сповіщення, веб– модуль для того щоб робити HTTP– запити тощо.

1.5. Огляд інструменту автоматизації тестування SELENIUM

Для розробки плеєра відтворення тестових сценаріїв використана бібліотека Selenium WebDriver.

Selenium WebDriver – це набір API з відкритим кодом, який надає можливості взаємодії з будь–яким із сучасних веб–браузерів, а потім, в свою чергу, для автоматизації дій користувача з цим браузером. Це важливий компонент сімейства Selenium який являється набором інструментів для автоматизації тестування.

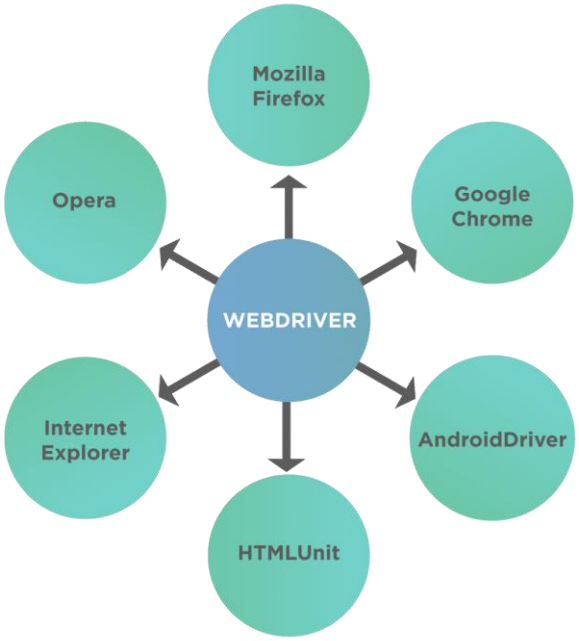


Рис.1.4. – Підтримувані драйвери браузерів Selenium Web– Driver

WebDriver охоплює деякі унікальні характеристики, що додає його популярності як інструменту веб– автоматизації. Деякі з цих характеристик:

– Сумісність із декількома браузерами – Однією з головних причин популярності Selenium та WebDriver є підтримка між браузерами з використанням того самого коду сценаріїв. Це дає можливість запускати певний фрагмент коду, що імітує реального користувача, використовуючи власну підтримку браузера, щоб здійснювати прямі виклики API без необхідності будь–якого програмного забезпечення або пристрою проміжного програмного забезпечення. На Рис.1.4. наведено список підтримуваних браузерів.

– Багатомовна підтримка – Не всі тестувальники добре знають певну мову програмування. Оскільки Selenium забезпечує підтримку багатьох мов, спеціаліст може використовувати JS, Python, Java, C#, Ruby. Це дає свободу писати код тією мовою, яка є більш зручною для користувача так і використовується на проекті загалом.
– Швидке виконання – на відміну від Selenium RC, WebDriver не залежить від сервера проміжного програмного забезпечення для взаємодії з браузером. WebDriver комунікує з браузерами за допомогою визначеного протоколу (JSON Wire), що дозволяє йому спілкуватися швидше, ніж більшість інструментів Selenium. Крім того, оскільки JSON Wire сам використовує JSON, який є дуже простим, обсяг передачі даних за виклик мінімальний. На Рис.1.7.2 показано, як WebDriver взаємодіє з браузером:

Рис.1.5. – Принцип взаємодії Selenium з браузером

– Локація веб–елементів – для того, щоб виконувати такі дії, як натискання, друк, drag&drop, нам спочатку потрібно визначити, над яким веб–елементом (кнопка, прапорець, спадне меню, текстове поле) нам потрібно виконати дію. Щоб полегшити це, WebDriver реалізовує методи ідентифікації веб–елементів за допомогою різних атрибутів HTML – таких як ідентифікатор, ім'я, клас, CSS, ім'я тегу, XPath, текст посилання тощо.
– Обробка динамічних веб–елементів – Бувають випадки, коли на сторінці є веб– елементи, які змінюються при кожному перезавантаженні сторінки. Оскільки атрибути HTML змінюються, виявлення цих елементів стає проблемою. Selenium пропонує безліч методів вирішення

цих ситуацій: Абсолютний Xpath – він містить повний XML– шлях відповідного елемента, Contains() – за допомогою цих функціональних елементів можна здійснювати пошук за частковим або повним текстом і використовувати для обробки динамічних елементів. StartsWith() – ця функція заснована на пошуку елементів за допомогою початкового тексту атрибута, про який йдеться.

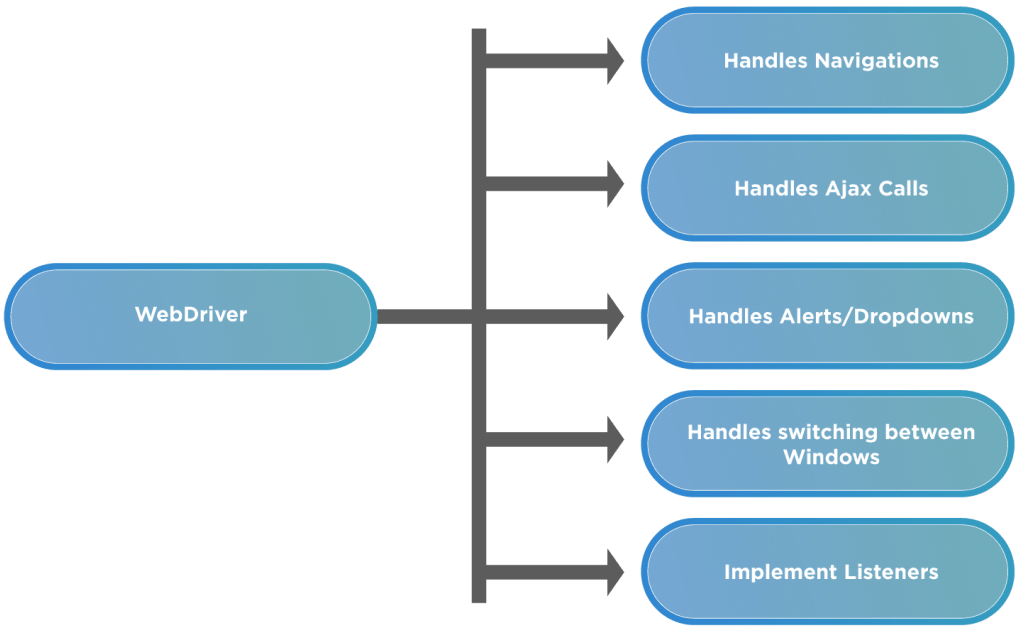


Рис.1.6. – Основні функції WebDriver

– Обробка очікування елементів – Не всі сторінки мають однакову структуру. Деякі з них легкі, а інші мають значну кількість обробки даних або викликів AJAX. Високонавантажені веб– елементи потребують деякого часу для повного завантаження. Для обробки таких ситуацій WebDriver забезпечив безліч механізмів очікування, які можна використовувати для призупинення виконання сценарію на необхідний проміжок часу на основі певних умов, а потім продовження, як тільки умова буде виконана. На Рис.1.7.3 наведено список, який показує можливості WebDriver, що допомагають обробляти динамічну поведінку веб– сторінок.

Будучи частиною загальної системи компонентів, ми дійшли висновку, що Selenium WebDriver не є самостійним інструментом тестування. Він містить різні компоненти, необхідні для запуску тестів які є архітектурною складовою Selenium.

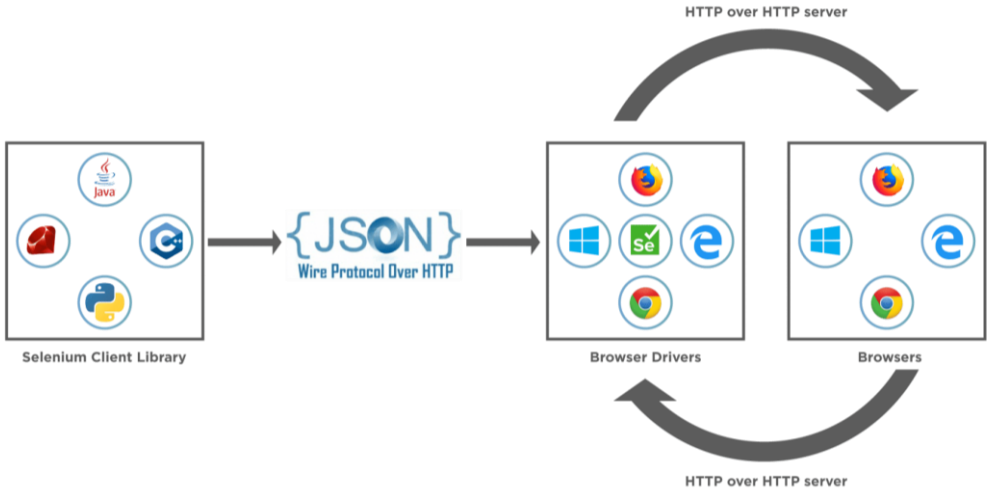


Рис.1.7. – Модель архітектури Selenium

JSON WIRE PROTOCOL – Відповідно до архітектури Selenium (Рис.1.6.), JSON Wire Protocol полегшує весь зв'язок, що відбувається в Selenium між браузером і кодом. Це ядро селену. Провідний протокол JSON забезпечує передачу даних за допомогою RESTful API (Репрезентативна передача стану), що являє собою транспортний механізм та визначає веб– службу RESTful за допомогою JSON через HTTP.

Драйвери браузера – Оскільки існують різні браузери, які підтримуються Selenium, кожен браузер має власну реалізацію стандарту W3C. Протокол JSONWire встановлює зв'язок між бінарними файлами браузера та клієнтськими бібліотеками.

Selenium може запускати тести в браузерах, лише якщо вони встановлені локально, або на локальній машині, або на серверних машинах. Тож установка браузера необхідна.

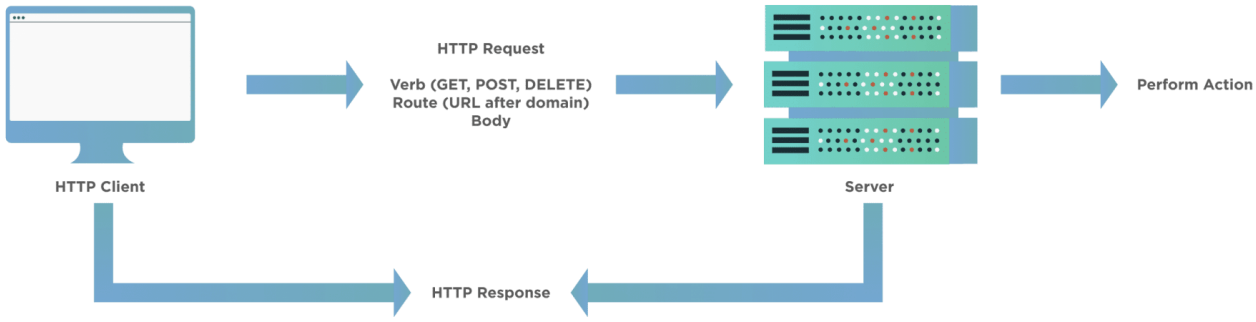


Рис.1.8. – Архітектура внутрішнього сервера Selenium

Коли користувач пише код WebDriver в Selenium і виконує його, у фоновому режимі відбуваються такі дії (Рис.1.7.5):

- Створюється запит HTTP, який надходить до відповідного драйвера браузера (Chrome, IE, Firefox). Для кожної команди Selenium існує індивідуальний запит.
- Драйвер браузера отримує запит через HTTP– сервер.
- Сервер HTTP оприділяє, які дії / інструкції потрібно виконати у браузері.
- Браузер виконує інструкції / кроки.
- Сервер HTTP отримує статус виконання, а потім відправляє статус до сценарію автоматизації, який відображає результат (позитивний, виняток або помилка виконання).

1.6. Архітектура сучасних JS додатків

1.6.1. Використання NodeJS для написання серверної частини додатку

Node.js – це серверна платформа, побудована на JavaScript Engine Google Chrome (V8 Engine). Node.js був розроблений Райаном Далом у 2009 році, і його остання версія – v0.14.36. Визначення Node.js, подане в офіційній документації, є таким:

Node.js – це платформа, побудована на середовищі виконання Chrome для того, щоб легко створювати швидкі та масштабовані мережеві програми. Node.js

використовує керовану подіями неблоковану модель вводу– виводу, що робить її простою та ефективною, що ідеально підходить для додатків які повинні обробляти інформацію у режимі реального часу, на розподілених пристроях.

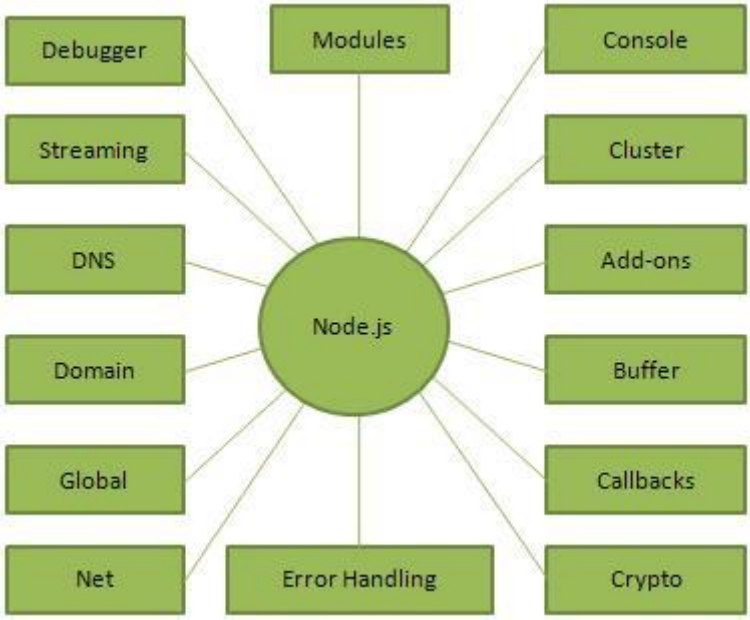


Рис. 1.9. – Основні модулі платформи Node JS

Node.js – це міжплатформене середовище виконання з відкритим кодом для розробки серверних та мережевих додатків. Додатки Node.js написані на JavaScript і можуть запускатися в середовищі виконання Node.js в операційних системах Mac OS, Microsoft Windows та Linux.

Node.js також надає багату бібліотеку різноманітних модулів JavaScript, що значною мірою спрощує розробку веб– додатків з його допомогою(Рис. 1.9.).

Node.js = Середовище виконання + Бібліотека JavaScript

Нижче наведено деякі важливі функції, які роблять Node.js найкращим вибором архітекторів програмного забезпечення.

Асинхронний та керований подіями – всі API бібліотеки Node.js є асинхронними, тобто не блокуючими. По суті, це означає, що сервер на основі Node.js ніколи не чекає, поки API поверне дані. Сервер переходить до наступного API після його виклику, і механізм сповіщення про події Node.js

допомагає серверу отримати відповідь від попереднього виклику API.

Швидкість роботи – побудований на JavaScript Engine V8 від Google Chrome, бібліотека Node.js виконує код на рівні з такими мовами програмування як C# та Java.

Однопоточний, але масштабований – Node.js використовує однопоточну модель із циклічними подіями. Механізм подій допомагає серверу реагувати неблокуючим способом і робить сервер дуже масштабованим на відміну від традиційних серверів, які створюють обмежені потоки для обробки запитів. Node.js використовує одно потокову програму, і ця сама програма може надавати послуги набагато більшій кількості запитів, ніж традиційні сервери, такі як Apache HTTP Server.

Без буферизації – програми Node.js ніколи не буферизують будь– які дані. Ці програми просто виводять дані шматками.

Ліцензія – Node.js випускається за ліцензією MIT.

NodeJS використовують такі команії, як eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo ! та Yammer.

В яких типах програм використовується Node.js:

- програми з введенням / виведенням;
- програми потокового передавання даних;
- інтенсивні програми в режимі реального часу (DIRT);
- додатки на основі JSON API;
- односторінкові програми;

Недоцільно використовувати Node.js для побудови додатків, які вимагають інтесивне навантаження на процесор – обробка фото та відео, системи обчислень.

1.6.2 Протоколи взаємодії HTTP\HTTPS

HTTP – це протокол клієнт–сервер, який дозволяє клієнтам запитувати веб–сторінки з веб–серверів. Це протокол рівня додатку, який широко

використовується в Інтернеті. Клієнтами, як правило, є веб–браузери. Коли користувач хоче отримати доступ до веб–сторінки, браузер надсилає повідомлення із запитом HTTP на веб– сервер. Сервер відповідає запитаною веб–сторінкою. За замовчуванням веб– сервери використовують порт TCP 80.

Клієнти та веб– сервери використовують метод відповіді на запит для взаємодії між собою, при цьому клієнти надсилають запити HTTP, а сервери відповідають відповідями HTTP. Клієнти зазвичай надсилають свої запити методами GET або POST, наприклад GET /homepage.html. Веб– сервери відповідають повідомленням про стан (200, якщо запит був успішним) і надсилає запитаний ресурс.

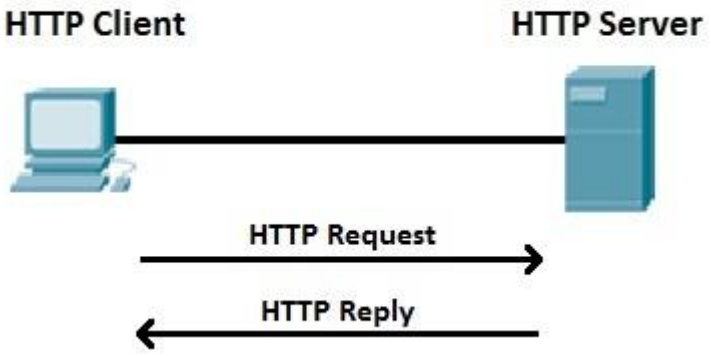


Рис. 1.10. Схема роботи протоколу HTTP

Наприклад, клієнт хоче отримати доступ до http://google.com і спрямовує свій браузер на URL–адресу http://google.com (це приклад повідомлення із запитом HTTP). Веб–сервер, що розміщує http://google.com, отримує запит і відповідає вмістом веб–сторінки (повідомлення відповіді HTTP)(Рис. 1.10.).

Веб– сервери зазвичай використовують добре відомий TCP– порт 80. Якщо порт не вказаний у URL– адресі, браузери використовуватимуть цей порт під час надсилання HTTP– запиту. Наприклад, ви отримаєте однаковий результат при запиті http://google.com та http://google.com:80.

Найчастіше використовується версія HTTP сьогодні – HTTP / 1.1. Більшість браузерів доступна і підтримує нову версію, HTTP / 2.

HTTPS (Hyper Text Transfer Protocol Secure) – це захищена версія HTTP. Цей протокол забезпечує безпечний зв'язок між клієнтом (наприклад, веб–браузером) та сервером (наприклад, веб–сервером) за допомогою шифрування. Для шифрування HTTPS використовує протокол TLS (Transport Layer Security) або попередній рівень Secure Sockets Layer (SSL).

HTTPS зазвичай використовується для створення захищеного каналу через якусь небезпечну мережу, наприклад Інтернет. Багато трафіку в Інтернеті є незашифрованим і легкодоступним для хакерських атак. HTTPS шифрує конфіденційну інформацію, що робить з'єднання безпечним.

URL– адреси HTTPS починаються з https замість http. Ви можете відразу визнати, що веб– сайт використовує HTTPS, оскільки праворуч від адресного рядка з'являється блокування:

HTTPS використовує добре відомий порт TCP 443. Якщо порт не вказаний у URL– адресі, браузери використовуватимуть цей порт під час надсилання запиту HTTPS. Наприклад, ви отримаєте однаковий результат при запиті https://gmail.com та https://gmail.com:443 .

1.6.3. Формат даних JSON

JSON або JavaScript Object Notation – це мінімальний, читабельний формат для структурування даних. Він використовується в основному для передачі даних між сервером та веб–додатком, як альтернатива XML. Squarespace використовує JSON для зберігання та упорядкування вмісту сайту, створеного за допомогою CMS.

Дві основні частини, що складають JSON, – це ключі та значення. Разом вони складають пару ключ / значення.

Ключ: Ключ – це завжди рядок, укладений у лапки.

Значення: Значення може бути рядком, числом, логічним виразом, масивом або об'єктом.

Пара ключ / значення: Пара значень ключа слідує певному синтаксису, за

ключем слідує двокрапка і значення. Пари ключ / значення розділяються комами.

Візьмемо один рядок JSON та визначимо кожну частину коду.

"foo": "bar"

Цей приклад – пара ключ / значення. Ключ – "foo", а значення – "bar".

Типи даних, які підтримує JSON

- масиви: Асоціативний масив значень;
- логічні значення: Істинно чи хибно;
- числа: цілі та числа з плаваючим знаком розміром до 64біт;
- об’єкти: Асоціативний масив пар ключ / значення;
- рядки: Кілька символів простого тексту, які зазвичай утворюють слово;

1.6.4. Restful APIs

Веб–архітектура RESTful надає інформацію про себе у вигляді інформації про свої ресурси. Це також дозволяє клієнту здійснювати дії з цими ресурсами, наприклад, створювати нові ресурси або змінювати існуючі ресурси.

Щоб API мали змогу підтримувати RESTful, слід дотримуватися набору обмежень при проектуванні.

REST означає «**RE**presentational **S**tate **T**ransfer».

Це означає, що при виклику RESTful API сервер передає клієнтові подання стану запитуваного ресурсу.

Наприклад, коли розробник викликає Instagram API для отримання конкретного користувача (ресурсу), API повертає стан цього користувача, включаючи його ім'я, кількість публікацій, які користувач розміщував до цього часу, скільки підписників у них є , і більше.

Представлення стану може бути у форматі JSON, і, мабуть, для більшості API це справді так. Також є підтримка форматів XML або HTML, але на сьогоднішній день вони є малопопулярними.

Те, що робить сервер, коли клієнт, викликаєтє один із його API, залежить

від двох речей, які потрібно надати серверу:

Ідентифікатор ресурсу – це URL– адреса ресурсу, також відома як кінцева точка(endpoint). Фактично, URL означає Uniform Resource Locator.

Операція, яку ви хочете, щоб сервер виконував на цьому ресурсі, у формі методу HTTP. Поширеними методами HTTP є GET, POST, PUT та DELETE.

Наприклад, для отримання конкретного користувача Twitter за допомогою RESTful API Twitter потрібна URL– адреса, що ідентифікує цього користувача та метод HTTP GET.

Інший приклад, ця URL–адреса: www.twitter.com/jk_rowling має унікальний ідентифікатор користувача Дж. К. Роулінг у Twitter, яким є її ім’я користувача, jk_rowling. Twitter використовує ім'я користувача як ідентифікатор, і справді імена користувачів Twitter унікальні – немає 2 користувачів Twitter з однаковим іменем користувача.

Метод HTTP GET вказує на те, що ми хочемо отримати стан цього користувача.

1.6.5. NoSQL MongoDB як альтернатива реляційним БД

NoSQL – загальний термін, що описує будь– яку альтернативну систему традиційним базам даних SQL.

Бази даних NoSQL досить сильно відрізняються від баз даних SQL. Всі вони використовують модель даних, яка має іншу структуру, ніж традиційна модель таблиці рядків і стовпців, що використовується з реляційними системами управління базами даних (СУБД).

Але бази даних NoSQL також досить сильно відрізняються одна від одної. NoSQL були розроблені в епоху Інтернету у відповідь на нездатність баз даних SQL задовольняти потреби веб– програм, що обробляють величезні обсяги даних і трафіку.

Компанії можуть застосовувати технологію NoSQL до широкого списку задач, заощаджуючи ресурси порівняно з розробкою реляційної бази даних. Бази

даних NoSQL містять гнучку модель схеми і розроблені для горизонтального масштабування на багатьох серверах, що робить їх привабливими для великих обсягів даних або навантажених додатків, які використовують більше одного сервера.

Популярність NoSQL обумовлена такими причинами:

- Темпи конструювання баз даних NoSQL є швидшими, порівняно із базами даних SQL.
- Структура багатьох різних форм даних простіше обробляється та зберігається за допомогою бази даних NoSQL.
- Обсяг даних у багатьох програмах не може ефективно обслуговуватися базою даних SQL.
- SQL не може впоратися з масштабами трафіку та мінімальним часом відповіді.

Бази даних документів зберігають дані у документах JSON, BSON або XML (звичайно, не в документах Word або документах Google). У базі даних документів документи можуть бути вкладеними. Окремі елементи можна проіндексувати для швидшого запитування.

Документи можна зберігати та отримувати у формі, яка набагато ближче до об'єктів даних, що використовуються в додатках, а це означає, що для використання даних у програмі потрібно менше коду. Дані SQL часто потрібно збирати та розбирати при переміщенні між програмами та сховищем.

Бази даних документів популярні серед розробників, оскільки вони мають можливість переробляти свої структури документів відповідно до їх потреб, формуючи специфічні структури даних, оскільки вимоги програм змінюються з часом. Ця гнучкість прискорює розробку, оскільки фактично дані стають схожими на код і перебувають під контролем розробників. У базах даних SQL може знадобитися втручання адміністраторів баз даних для зміни структури.

Найбільш широко прийняті бази даних документів зазвичай реалізуються з масштабованою архітектурою, що забезпечує чіткий шлях до масштабованості

як обсягів даних, так і трафіку.

Варіанти використання включають платформи електронної комерції, торгові платформи та розробку мобільних додатків у різних галузях.

MongoDB – це орієнтована на документи база даних NoSQL, яка використовується для великого обсягу зберігання даних. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключ–значення, які є основною одиницею даних у MongoDB. Колекції містять набори документів та функції, що є еквівалентом реляційних таблиць баз даних.

1.7. Постановка завдань дослідження

Під час розробки концепції системи було визначено, що перш ніж проектувати систему, та вибирати конкретні інструменти та засоби реалізації, потрібно дослідити сферу автоматизації тестування програмного забезпечення та вивчити системи, які мають подібний функціонал, до нашої майбутньої.

Тому для ефективності побудови додатку, були поставлені завдання на дослідження певних технологій та інструментів, що дозволять реалізувати розроблену концепцію, а саме:

- Дослідити засоби запису дій користувача з веб-сторінок;
- Дослідити інструменти побудови десктопних додатків за допомогою мови програмування JS;
- Дослідити методи передачі інформації між додатками які розташовані на одному ПК;
- Дослідити структуру даних майбутньої системи, та підібрати БД, яка задовольнятиме архітектуру та вимоги до системи.
- Дослідити інструменти відтворення тестових сценаріїв за допомогою мови програмування JS.

2. РОЗРОБКА СТРУКТУРИ ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ПЗ НА JS

2.1. Завдання додатку для автоматизації тестування ПЗ на JS

Мануальне тестування завжди являло собою дуже довгий, ресурсозатратний та вимагаючий особливої уваги і терпіння процес. Дуже часто спеціалісти, які працюють на посадах мануального тестувальника стикаються з такими проблемами як, недостатній рівень концентрації, зниження робочої продуктивності через великий об’єм монотонної та нудної роботи. З іншої сторони використання автоматизоаних тестів якби і вирішує всі вищеперелічені проблеми, але тягне за собою нові, такі як:

- підготовка спеціалістів та їх тренування;
- витрати часу на розробку автоматизованих сценаріїв;
- постійна підтримка тестів при змінах функціоналу програми;

Ці фактори дуже часто являються причиною відмови компаній від використання автоматизованого тестування. Єдиним виходом є інструменти безскриптової автоматизації тестів, такі як інструменти захоплення та відтворення дій.

Основна перевага цих інструментів, це можливість без спеціальної підготовки та за відносно короткий час побудувати автоматизований набір тестових сценаріїв. Що дозволяє спеціалістам, які не мають спеціальних навичок, бути такими ж ефективними як і спеціалісти– автоматизатори.

Проектована система дозволить виконувати такі задачі, пов’язані з автоматизацією тестування програмного забезпечення:

- обробка дій користувача на веб– сторінці;
- відображення стану захоплення тестового сценарію;
- передача, збереження та обробка тестового сценарію;
- конвертація записаного сценарію у скрипт на мові програмування (JS,

Java, C#, Python);

- імпорт вже записаних сценаріїв в систему та їх обробка;
- виконання записаних сценаріїв;
- відображення статусу запуску тестів;
- збереження метрик виконання тестових сценаріїв для проведення аналізу тестування;

Мета роботи – розробка програмного забезпечення для автоматизації запису та відтворення тестових сценаріїв на веб–сторінках на мові програмування JavaScript.

2.2. Моделювання об’єкту проектування

2.2.1. Діаграма прецедентів системи

Основна мета створення будь–якої програмної системи – створення такого програмного продукту, який допомагає користувачеві виконувати свої повсякденні завдання. Для створення таких програм в першу чергу визначаються вимоги, яким повинна задовольняти система. Однак, якщо дати користувачам написати ці вимоги на папері, то часто можна отримати список функцій, за яким важко судити чи майбутня система виконувати своє призначення і чи зможе вона полегшити користувачеві виконання його роботи взагалі. Незрозуміло які з виконуваних функцій важливіші і для кого.

Для того щоб більш точно зрозуміти як повинна працювати система, все частіше використовується опис функціональності системи через варіанти використання (Use Case або прецеденти). Варіанти використання це – опис послідовності дій, які може здійснювати система у відповідь на зовнішні впливи користувачів або інших програмних систем. Варіанти використання відображають функціональність системи з точки зору отримання відчутного результату для користувача, тому вони точніше дозволяють ранжувати функції за значимістю одержуваного результату.

Варіанти використання призначені переважно для визначення функціональних вимог до системи і керують усім процесом розробки. Всі основні види діяльності, такі як аналіз, проектування, тестування виконуються на основі варіантів використання. Під час аналізу і проектування варіанти використання дозволяють зрозуміти як результати, які хоче отримати користувач впливають на архітектуру системи і як повинні поводитися компоненти системи, для того щоб реалізувати потрібну для користувача функціональність.

В процесі тестування, описані раніше варіанти використання, дозволяють простіше оцінити точність реалізації вимог користувачів і дозволяють провести покрокову перевірку цих вимог.

Стратегія використання прецедентів при визначенні вимог визначає необхідність додатково до питання "що користувачі чекають від системи?" задавати питання "що система повинна зробити для учасників форуму?". Такий підхід дозволяє шукати функції, які потрібні багатьом користувачам, і виключати ті можливості, які не можуть допомогти користувачам виконувати свої повсякденні завдання.

Діаграма варіантів використання складається з акторів, для яких система виробляє дію і власне дії Use Case, яке описує те, що актор хоче отримати від системи. Актор позначається чоловічка, а Use Case – овалом.

Метою даного проекту є створення системи, яка буде зберігати дані про клієнтів, інформацію про угоди з ними. Також основним завданням цієї системи є автоматичне формування рахунків, які згодом менеджер відправляє клієнту.

Користувачами даної системи будуть менеджери відділу продажів середнього підприємства. На рисунку 2.1 показані основні дії, які будуть доступні користувачеві системи керування малим і середнім бізнесом.

Діаграма класів (class diagram) – діаграма мови UML, на якій представлена сукупність декларативних або статичних елементів моделі, таких як класи з атрибутами і операціями, а також зв'язують їх відносини.

Діаграма класів призначена для представлення статичної структури моделі

системи в термінології класів об'єктно програмування. При цьому діаграма класів може містити інтерфейси, пакети, відносини і навіть окремі екземпляри класифікаторів, такі як об'єкти і зв'язку. Коли говорять про даної діаграмі, мають на увазі статичну структурну модель проектованої системи, тобто графічне представлення таких структурних взаємозв'язків логічної моделі системи, які не залежать від часу.

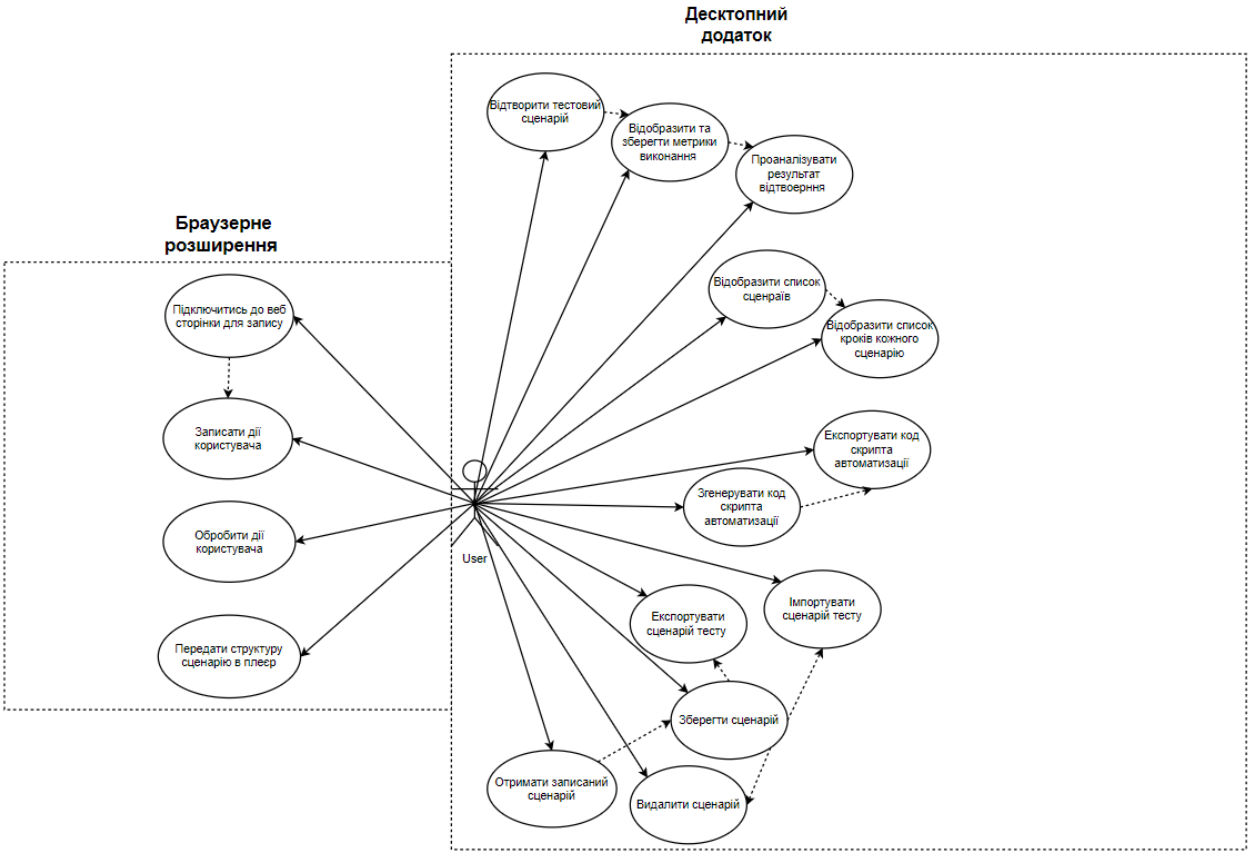


Рисунок 2.1 – UML Діаграма прецедентів системи

Клас (class) – абстрактне опис безлічі однорідних об'єктів, що мають однакові атрибути, операції і відносини з об'єктами інших класів. Графічно клас в нотації мови UML зображується у вигляді прямокутника, який додатково може бути розділений горизонтальними лініями на розділи або секції. У цих секціях можуть зазначатися ім'я класу, атрибути і операції класу.

На початкових етапах розробки діаграми окремі класи можуть позначатися простим прямокутником, в якому повинно бути зазначено ім'я відповідного

класу. У міру опрацювання окремих компонентів діаграми опис класів доповнюється атрибутами і операціями.

За допомогою методів класи взаємодіють один з одним. Сукупність усіх методів класів утворює інтерфейс системи.

2.2.2. Структура даних в системі

Оскільки ми використовуємо документо–орієнтовану БД MongoDB, структура нашої бази даних складатиметься з документів, які матимуть певний набір властивостей. Чому я вибрав саме документо–орієнтовану базу даних? Відповідь лежить в архітектурі самого додатку. Оскільки записаний сценарій складається з набору дій згенерованих у форматі JSON, то немає сенсу використовувати реляційну БД, вона тільки збільшить навантаження на роботу системи, тому що потрібно буде конвертувати записаний сценарій, записувати в різні таблиці, та при отриманні знову назад групувати дані у один тест. За допомогою NoSQL бази можна записувати однотипні документи, які зразу ж надходять в додаток.

Розглянемо приклади структур даних, які використовуються у системі:

```
  action: {
         locatorType: 'xpath',
         locator: "id('rso')/div[@class='hlcw0c']/div[@class='g']/div[@class='tF2Cx
c']/div[@class='yuRUbf']/a[1]/h3[@class='LC20lb DKV0Md']/span[1]",
         actionType: 'typeForm'
         text: 'lalalla'
     }
```

Найменшою одиницею і документом в структурі додатку є Action в якій зберігаються записані дані про 1 дію користувача, про елемент над яким була виконана ця дія, його положення на веб сторінці у форматі Xpath, тип дії та текст, який було передано елементу, якщо він існує (Наприклад, заповнення текстового поля).

```
 scenario: {
       url: 'https://www.google.com/search?q=Vusokyi+Zamok&oq=Vusokyi+Zamok&aqs=chr
ome..69i57j0i13l2j0i13i30l2.1142j0j7&sourceid=chrome&ie=UTF- 8',
       actions:  action[]
    }
```

Далі існує документ сценарію, він використовується для передачі вже записаних сценаріїв з браузерного розширення у декстопний додаток.

При отриманні сценарію, система його ідентифікує, записує ім’я сценарію.

```
executionData:{
          date: '3/27/2021T14:20:35',
          status: true
      }
```

Ця структура існує для того, щоб зберігати інформацію про запуски сценаріїв та результат їх використання, що потрібно для відображення метрик в системі.

Структура даних, в якій вся інформація групується, являється

```
 testScenario : {
          name: 'Test',
          uuid: 'jsaui213wwqhdjhqjyiqyiiiue23i4234i2u'
          scenario: scenario{},
          stats: executionData[],
      }
```

В ній кожен сценарій має ім’я, унікальний ідентифікатор, сам сценарій, та статистику виконання.

Ну і фінальна структура даних, за допомогою якої вже і відбувається запис в БД, являється набором тестових сценаріїв:

```
testSuite :{
        name: 'Test',
        uuid: 'saoidqowna218ksajd21381saldsaoid1ods'
        testScenarios: testScenario[]
    }
```

2.2.3. Робота з даними за допомогою Mongo DB

Для того щоб використовувати MongoDB в додатках, керованих Node JS, потрібно підключити драйвер бази даних за допомогою пакету mongodb, який можна завантажити за допомогою пакетного менеджера NPM.

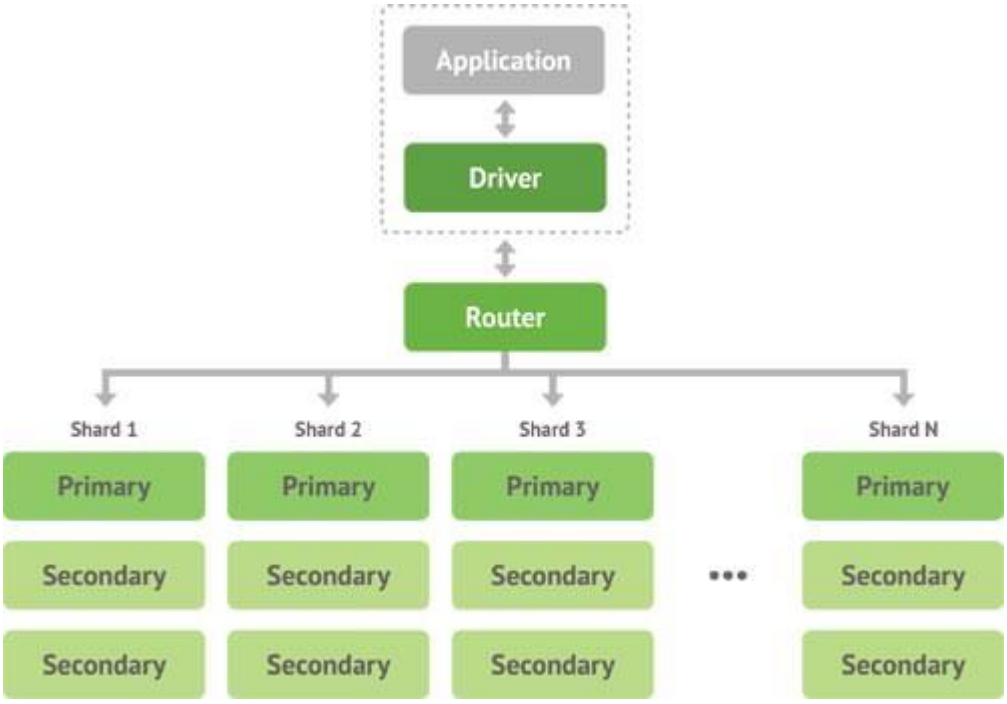


Рис. 2.2. – Схема роботи з даними MongoDB

Для підключення до БД достатньо лише вказати розташування сервера:

```
const MongoClient = require("mongodb").MongoClient;
const url = "mongodb://localhost:27017/";
const mongoClient = new MongoClient(url, { useUnifiedTopology: true });
```

Варто зауважити що всі методи роботи з даними в MongoDB засновані на

принципі callback функцій. Тобто метод роботи приймає 2 параметра: дані, які потрібно знайти чи додати, та функцію обробник, яка буде викликана одразу після успішного проведення операції.

MongoDB Driver реалізує зручний та зрозумілий інтерфейс для роботи з даними, всі документи в базі даних записуються у форматі JSON.

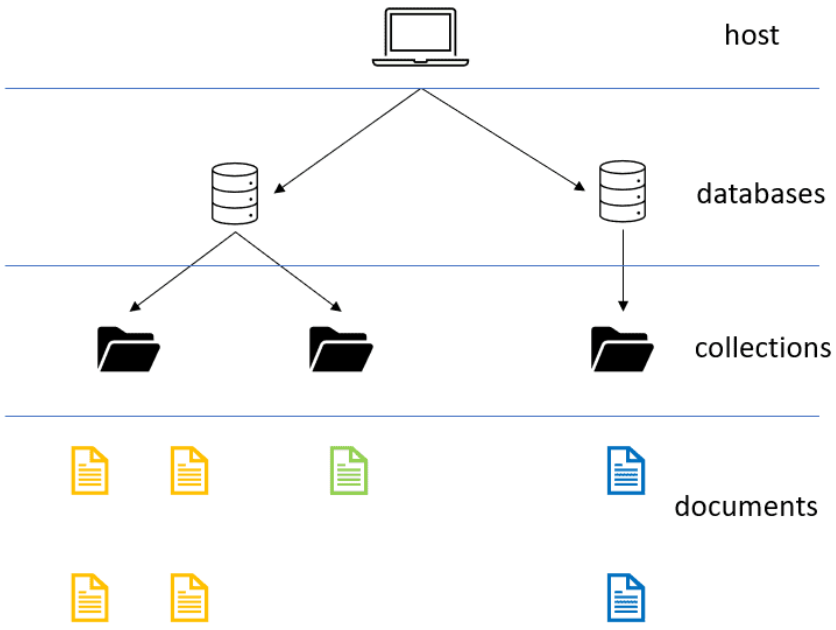


Рис. 2.3. – Модель файлової структури MongoDB

Проводити маніпуляцію з даними можливо за допомогою методів класу MongoClient:

- Connect(dataBaseUrl, callback);
- CreateDatabase(dataBaseUrl, callback);
- CreateCollection(dataBaseUrl, collectionName, callback);
- InsertOne(data, callback);
- InsertMany(data, callback);
- FindOne\FindMay(query, callback);
- UpdateOne\UpdateMany(query, newData, callback);
- DeleteOne\DeleteMany(query, callback);

2.3. Структура системи

2.3.1. Структура браузерного розширення.

Як вже згадувалось у розділі 1.3 ми використовуємо платформу Chrome Extensions для того щоб збудувати додаток для захоплення дій користувача.

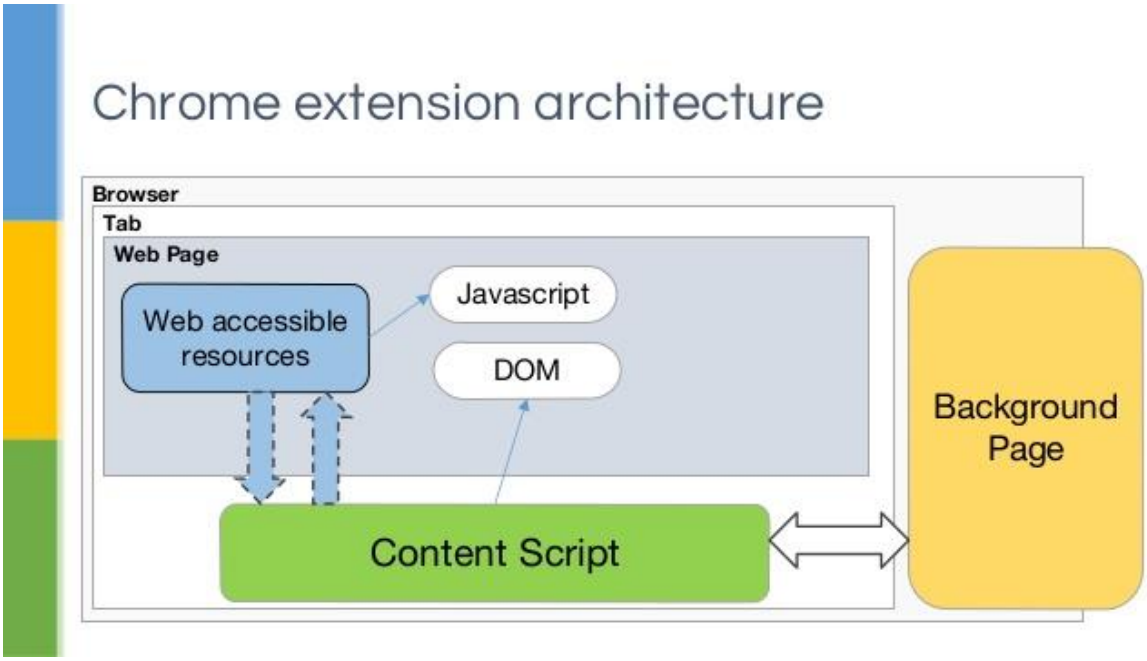


Рис 2.4. – Стандартна структура розширення

Початково структура типічного розширення складається з таких частин (Рис. 2.4):

- Manifest файлу в якому лежать всі налаштування, дозволи та підключені всі скрипти що потрібні для запуску додатку.
- HTML файлу, в якому лежить код сторінки випливаючого вікна, яке ми будемо показувати при запуску розширення. Також разом з цим файлом йдуть додаткові JS скрипти які відповідають за поведінку та логіку роботи випливаючого вікна, CSS стилі, що дозволяють стилізувати розширення.
- Content Scripts – призначені для того, щоб отримувати інформацію з браузера чи впливати на неї – отримання даних про події на веб–сторінці, захоплення відео з камери, зміна розміру вікна.

– Background Scripts – слугують для обробки дій користувача на сторінці, додавання нового функціоналу до браузера – розширення контекстних меню. Такий тип скриптів являється асинхронним, тобто виконується браузером в окремому процесі та не може обробляти події які були додані за допомогою цього ж скрипта. Тому частіше за все обробка всіх подій та відповідні зміни до поведінки браузера реалізовані в Content скрипті, які передаються туди за допомогою спеціального Send Message API.

Дуже важливим інструментом при побудові розширень являється інструмент Extensions APIs – що являє собою набір бібліотек, які реалізують доступ до різних API браузера. Це дозволяє отримувати інформацію про веб сторінку, про події на ній, доступ до заліза комп'ютера (веб– камера, мікрофон, пам'ять, активні вкладки у браузері, історія відвідування та ін.).

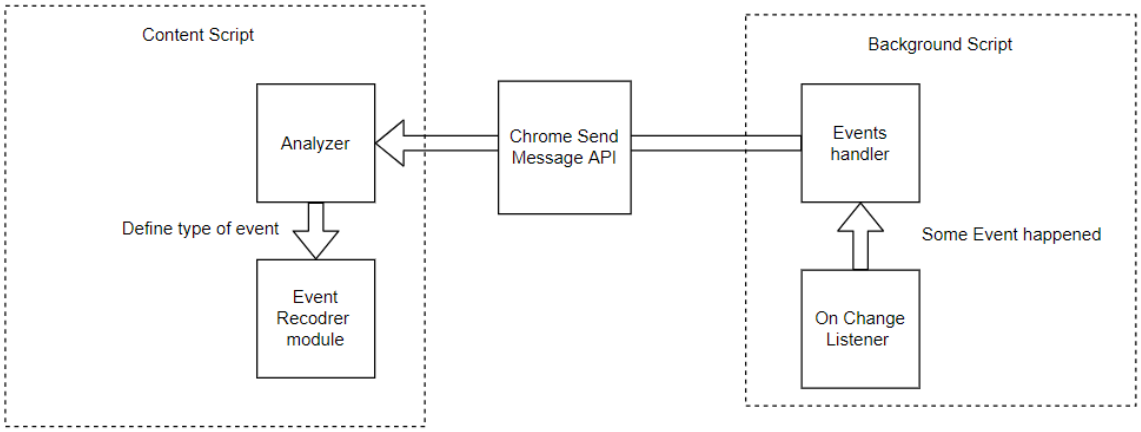


Рис. 2.5. – Модель взаємодії модулів розширення

Як відбувається процес захоплення події та його передачі на обробку в розширенні, основна задача якого являється записувати дії користувача:

Слугує фоновий скрипт, який використовує Chrome Event Listener, що зав'язаний на подію Change. Тобто коли відбувається хоч якась подія на

сторінці, то обробник спрацьовує, та відправляє дані про подію за допомогою Chrome Send Message API до контентного скрипта, де вже спеціальний модуль оприділяє що це за подія, над яким елементом вона відбулась, та які дані були передані. Також цей модуль вичисляє xpath шлях до елемента на сторінці. Ця вся інформація передається, до модуля що записує та генерує структуру тестового сценарію (Рис. 2.5.).

2.3.2. Структура десктопного додатку

Десктопний додаток побудований за допомогою мікросервісної архітектури Node JS та включає в себе мікросервіс на основі бібліотеки Electron JS який відповідає за відображення графічного інтерфейсу для користування, та включає в собі ще декілька додаткових мікросервісів, які відповідають за отримання скрипта від браузерного розширення– записувача , його збереження та відтворення.

Загалом, архітектуру десктопного додатку можна відобразити за допомогою діаграми:

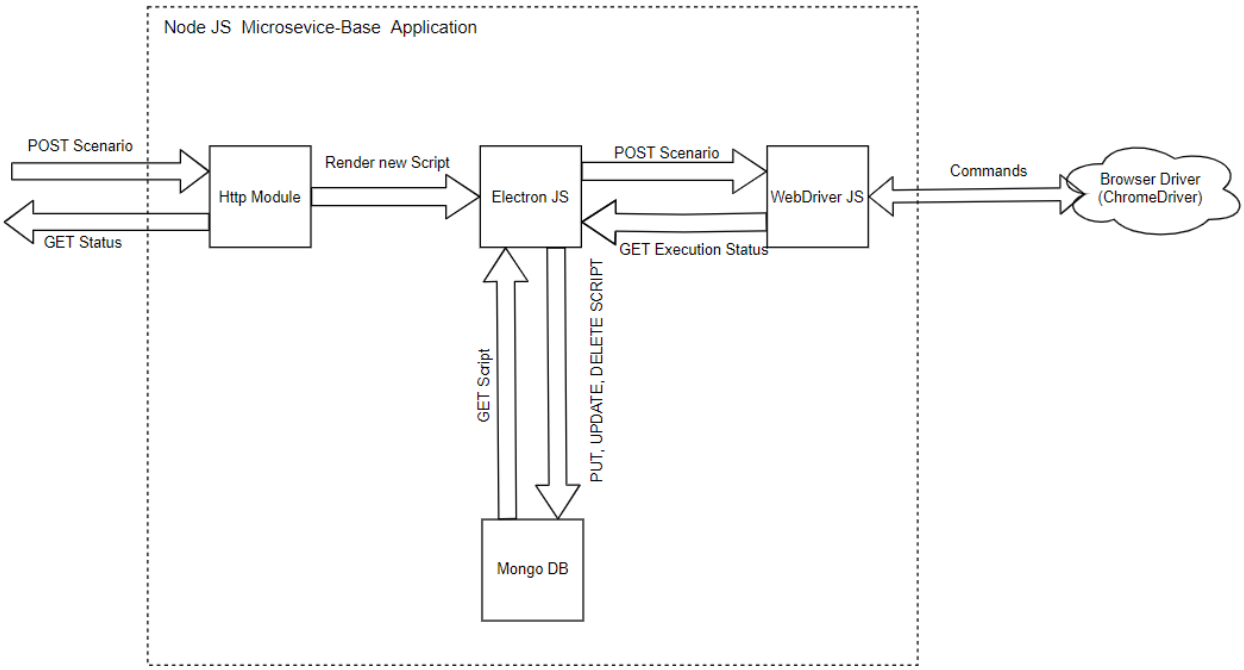


Рис. 2.6. – Структура мікросервісів десктопного додатку– плеєра

Потік даних в десктопному додатку– плеєрі можна описати так: Коли користувач записав тестовий сценарій в браузерному розширенні та натискає кнопку «Відіслати Сценарій», формується HTTP запит, який обробляє HTTP модуль додатку, він оприділяє чи було надіслані валідні дані та викликає Electron метод для відображення нового скрипта. Mongo DB Data Service слугує для того, щоб зберігати нові сценарії, отримувати вже записані, оновлювати їх чи видаляти. Webdriver JS відповідає безпосередньо за основну функцію додатку – відтворення тестових сценаріїв та отримання статистики про виконання. Для цього був реалізований спеціальний модуль, який обробляє всі дії в скрипті, оприділяє їх тип, локатор та вже за допомогою драйвера браузера (в нашому випадку це Google Chrome) надсилає покроково дії на виконання (Рис. 2.6.).

Також, в алгоритм відтворення було інтегровано методи, як стежать за статусом відтворення кожного кроку, що дозволяє зупинити процес роботи алгоритму, якщо виконання невдале, та прозвітувати що саме і в якому місці пішло не так.

Загалом, потрібно зауважити, що частина алгоритму, яка відповідає за відтворення тестових сценаріїв, є гнучкою завдяки використанню Webdriver JS, оскільки спілкування з драйвером браузера є поліморфним, що означає те, що розробники привели алгоритм комунікації до одного формату для всіх існуючих та підтримуваних драйверів на ринку – Google Chrome, Opera, Mozilla Firefox, MS Edge, IE 11. Це дає можливість у майбутньому розробити функціонал, який дозволятиме користувачу вибрати браузер для виконання тесту, що дозволить системі покривати потреби кросбраузерного тестування.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Оцінка та планування розробки проекту

Коли було визначено основний набір вимог, та архітектуру системи, потрібно було сформувати список задач, які потрібно реалізувати для того, щоб отримати MVP (Minimum Viable Product). Чому я обрав саме цей підхід до реалізації? Насправді, відповідь лежить у тому, що для реалізації продукту є досить обмежені часові рамки, які потрібно використати з максимальною ефективністю, та задовольнити найбільшу частину вимог продукту. Тому естімація(оцінка) розробки була проведена з акцентом на принцип KFF(Key Features First), що являється одним з ключових положень підходу до розробки MVP.

№	Вимога
1	Запис користувацьких дій
2	Відстеження статусу запису
3	Генерація тестових скриптів мовою JS
4	Генерація тестових скриптів мовою C#
5	Передача тестових сценаріїв
6	Збереження тестових сценаріїв
7	Відтворення тестових сценаріїв
8	Відстеження статусу відтворення теста
9	Відстеження статусу відтворення кожного кроку теста
10	Редагування теста
11	Формування груп тестових сценаріїв

Рис. 3.1. – Функціональні вимоги системи

Варто зазначити, що оцінка часу потрібного на розробку була проведена за допомогою Experience Based техніки, яка базується на минулому досвіду розробника, та його рівню знань (Рис. 3.1).

Коли був сформований список задач, та виявилось, що загальна кількість часу потрібного на розробку їх всіх становить близько 170 годин, а кількість

робочого часу, що залишилась становить всього 100 годин, було прийнято рішення застосувати техніку ABC аналізу, щоб залишити тільки найбільш приоритетні задачі для реалізації.

№	Вимога	Критерій необхідності (1-10)	Оцінка часу на розробку (год.)	Важливість	Відсоток важливості	Категорія
1	Запис користувацьких дій	10	20	200	17%	A
2	Відстеження статусу запису	9	10	90	8%	B
3	Генерація тестових скриптів мовою JS	9	10	90	8%	B
4	Генерація тестових скриптів мовою C#	3	10	30	3%	C
5	Передача тестових сценаріїв	10	20	200	17%	A
6	Збереження тестових сценаріїв	10	10	100	8%	B
7	Відтворення тестових сценаріїв	10	20	200	17%	A
8	Відстеження статусу відтворення теста	10	10	100	8%	B
9	Відстеження статусу відтворення кожного кроку теста	3	20	60	5%	C
10	Редагування теста	4	20	80	7%	B
11	Формування груп тестових сценаріїв	2	20	40	3%	C
		Сума:	170	1190	100%	

Рис. 3.2. – Використання ABC аналізу для приоритезації вимог

Завдяки даній техніці було виділено категорію кожної вимоги та було прийнято рішення відкинути від розробки початкової версії програми всі вимоги категорії C, та вимогу під номером 10 з категорії B, так як вона також поміщалась по оцінці часу, та являє собою вимогу з найменшою важливістю з категорії B (Рис. 3.2.).

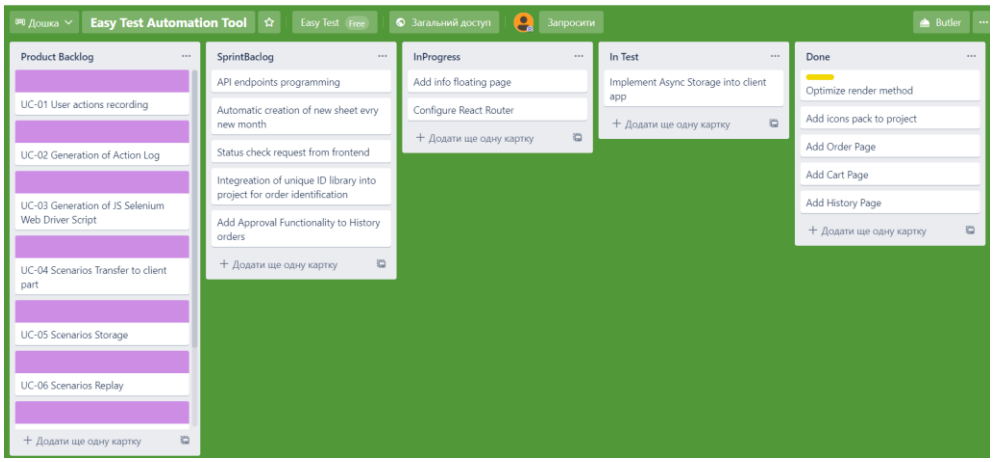


Рис. 3.3. – Відстеження статусу розробки продукту за допомогою інструмента Trello

Всі задачі, що були описані, були поміщені у систему відстежування задач Trello, для кращої простежуваності статусу їх виконання та труднощів, які можуть виникати з ними (Рис. 3.3).

Кожна вимога була описана в стилі use–case (Сценарію використання), що дозволяє приміняти BDD(Behavioral Driven Development) підхід, який базується на розробці функціоналу, орієнтуючись на сценарії використання та поведінки системи.

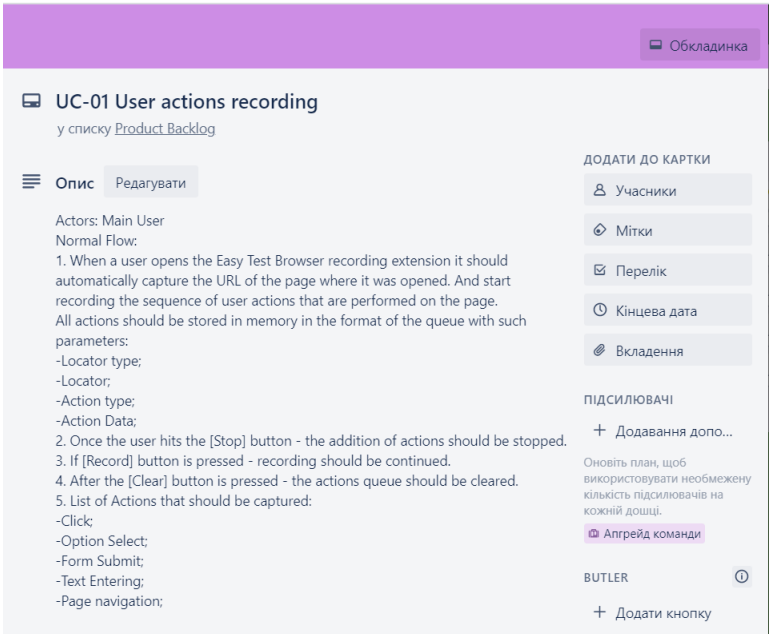


Рис 3.3. – Приклад описання вимоги у форматі use–case

При безпосередньому переміщенні задачі в спринт, вона розбивається вже на низку технічних підзадач, які оцінюються окремо та поступово реалізуються.

3.2. Набір інструментів використаних для розробки

В якості інтегрованого середовища розробника було вибрано Visual Studio Code, оскільки вона першочергово була розроблена для написання коду програм з використанням веб технологій та мов JS, HTML, CSS та має ряд вагомих переваг над іншими IDE, а саме (Рис 3.4.):

– Зручний користувацький інтерфейс;
– Підтримка багатьох мов програмування з коробки – JS, Python, TypeScript, C#;
– Великий набір розширень та плагінів;
– Інтеграція з системою контроля версій GIT;

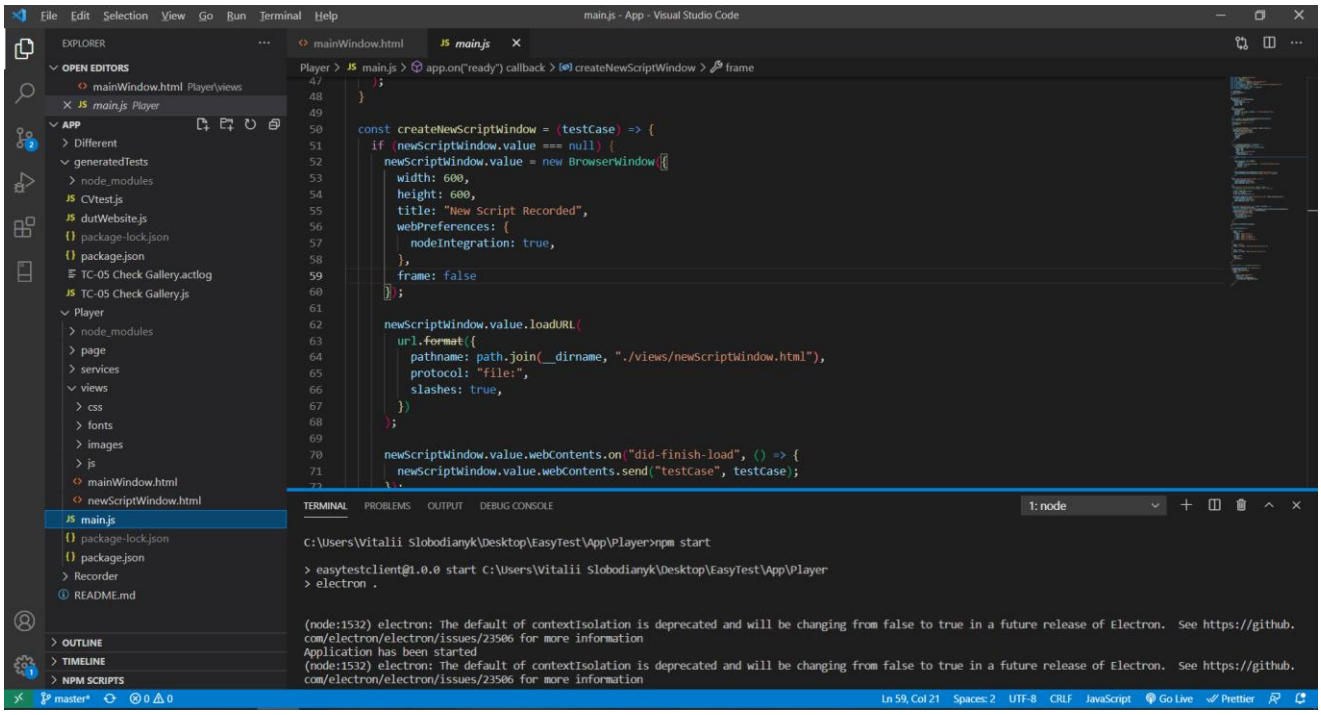


Рис 3.4. – Visual Studio Code

Для перегляду даних в БД, та їх тестування було використано MongoDB Compass – утиліта яка дозволяє підключатись до MongoDB та переглядати, шукати, додавати та видаляти дані. Даний інструмент нативно підтримує БД MongoDB, що робить процес підключення та пошук даних максимально простим. Достатньо всього лиш вказати значення певного параметра, або почати пошук за ключовим словом. Також даний інструмент підтримує підключення до декількох серверів БД, що робить його зручним при тестуванні системи з розподіленими БД.

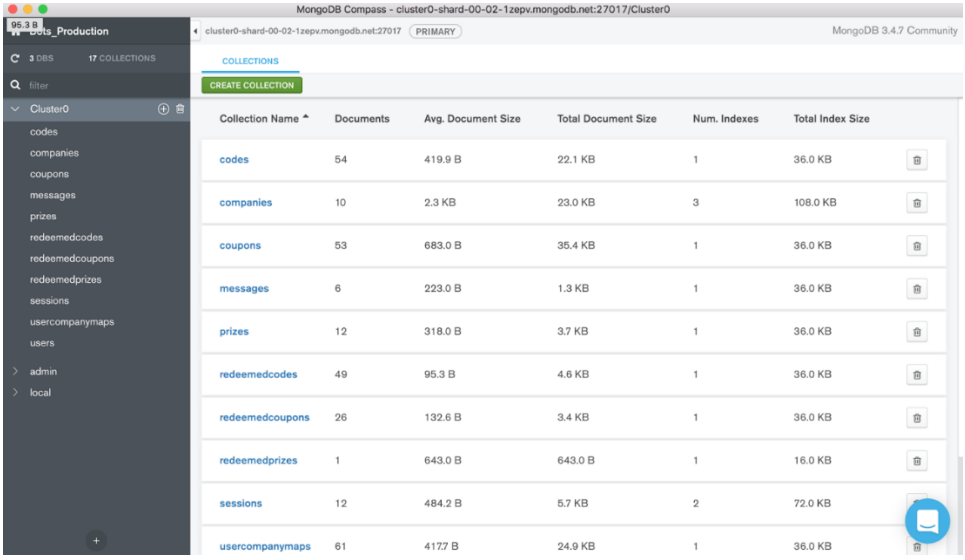


Рис 3.5. – Mongo DB Compass

Також за допомогою цього інструмента можна тестувати розроблену структуру БД. Основна перевага даного інструменту – спрощення використання БД в порівнянні зі стандартними інструментами, які використовують командний рядок для взаємодії з БД та її даними.

При чому, для маніпуляції з даними зовсім не потрібно знати якусь особливу мову запитів, всі дії проводяться через інтерфейс користувача(Рис 3.5.).

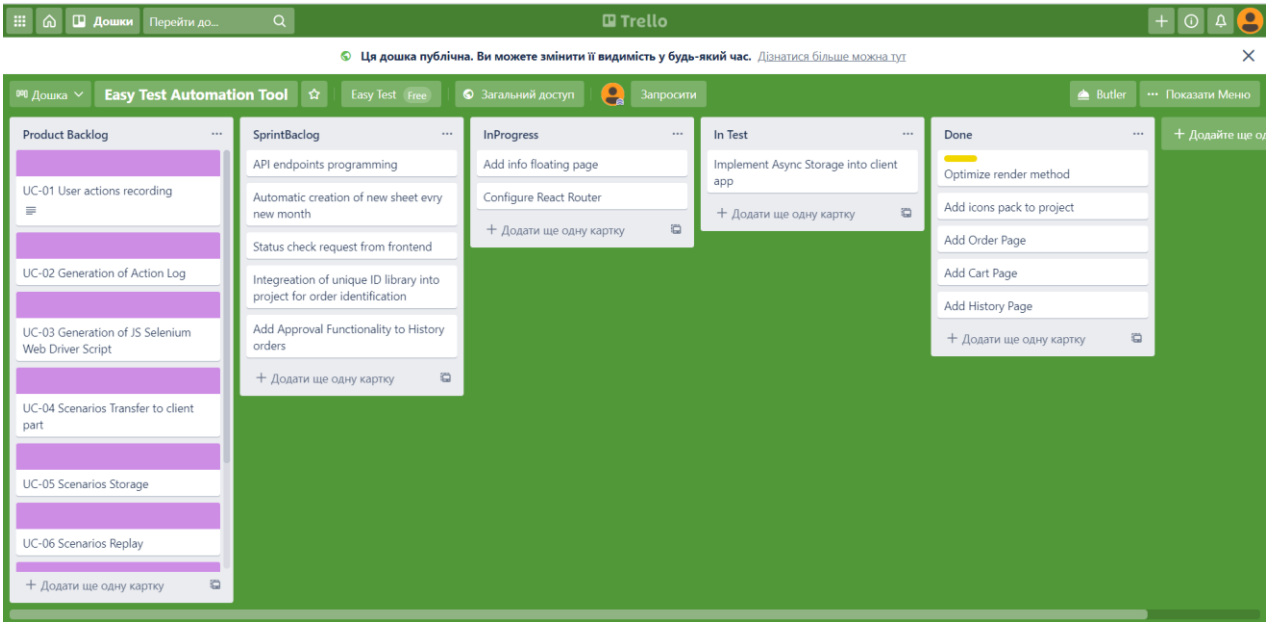


Рис 3.6. – Trello

Для відстеження статусу виконання задач, потрібних для успішної розробки проекту, планування наступних дій і документування процесу було використано інструмент Trello, який являється дуже простим та зручним, а головне безплатним інструментом менеджменту проектів. Варто зазначити, що дана утиліта ідеально підходить для менеджменту невеликих проектів в складі яких є 1– 5 персон (Рис 3.6.).

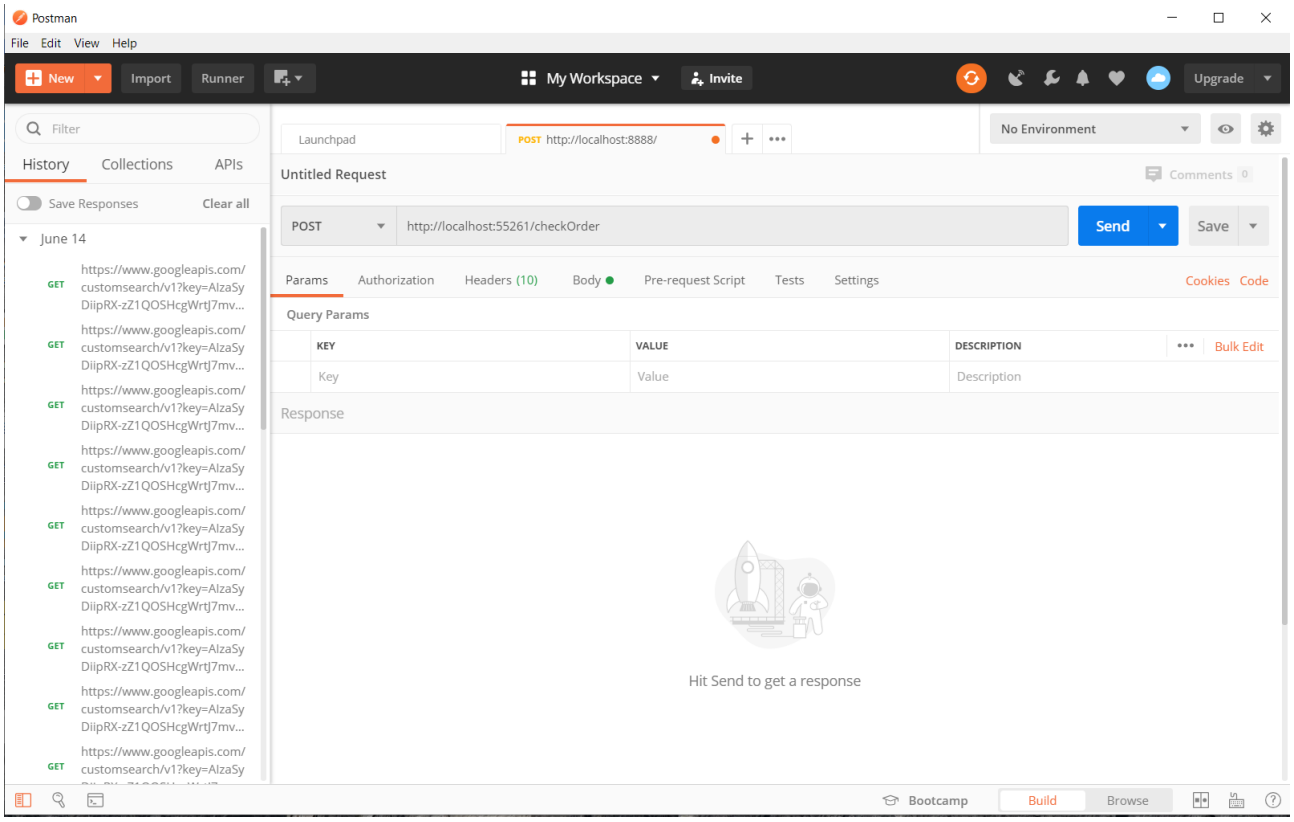


Рис 3.7. – Postman

Для тестування взаємодії додатка записувача сценаріїв та десктопного плеєра, було використано утиліту Postman, яка дозволяє просто, швидко та зручно протестувати реалізовані API за допомогою формування та відправки HTTP запитів до нього, та простежування результату їх виконання і відповіді (Рис. 3.7.).

3.3. Функціонал браузерного розширення та його модулі

Як вже відомо з вимог до системи, функціонал розширення поділений на такі категорії:

- Запис дій користувача (Перехоплення, Обробка, Запис);
- Відображення всього процесу запису на графічному інтерфейсі;
- Генератор скриптів з записаного сценарію на мовах JS, Python, C#, Java;
- Передача записаного сценарію;

За кожен функціонал відповідає один або декілька модулів, які обробляють дані(Рис. 3.8.):

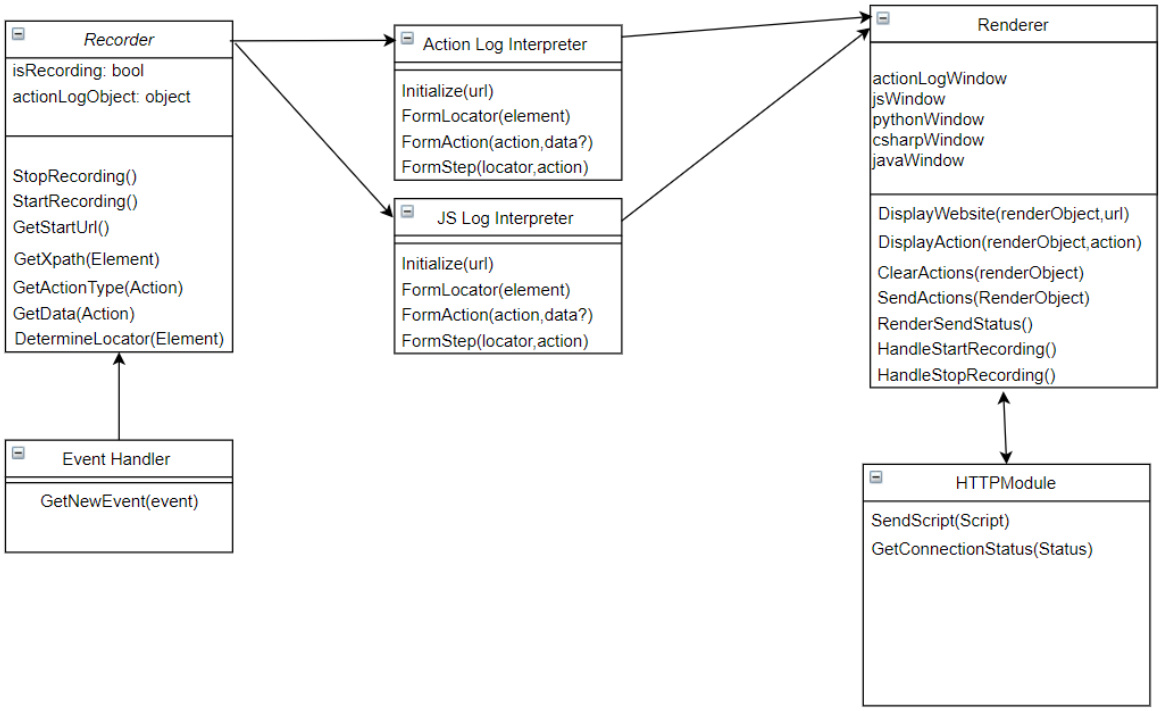


Рис. 3.8. – Схема модулів браузерного розширення системи

Розглянемо модуль EventHandler, його основна відповідальність – обробляти дії користувача які поступають через Chrome SendMessage API на подію onChange. Він містить функцію GetNewEvent(Event) – так званий «Зчитувач» користувацьких дій.

Суть його роботи полягає в тому, що в сторінку додається сторонній код, який зчитує структуру DOM дерева сторінки, та додає у кожен її елемент слухач подій, залежно від типу, тобто якщо елемент являється кнопкою, то розроблений алгоритм додає подію onClick для даного вузла сторінки, для поля введення onKeyPress і тд. (Рис 3.9.).

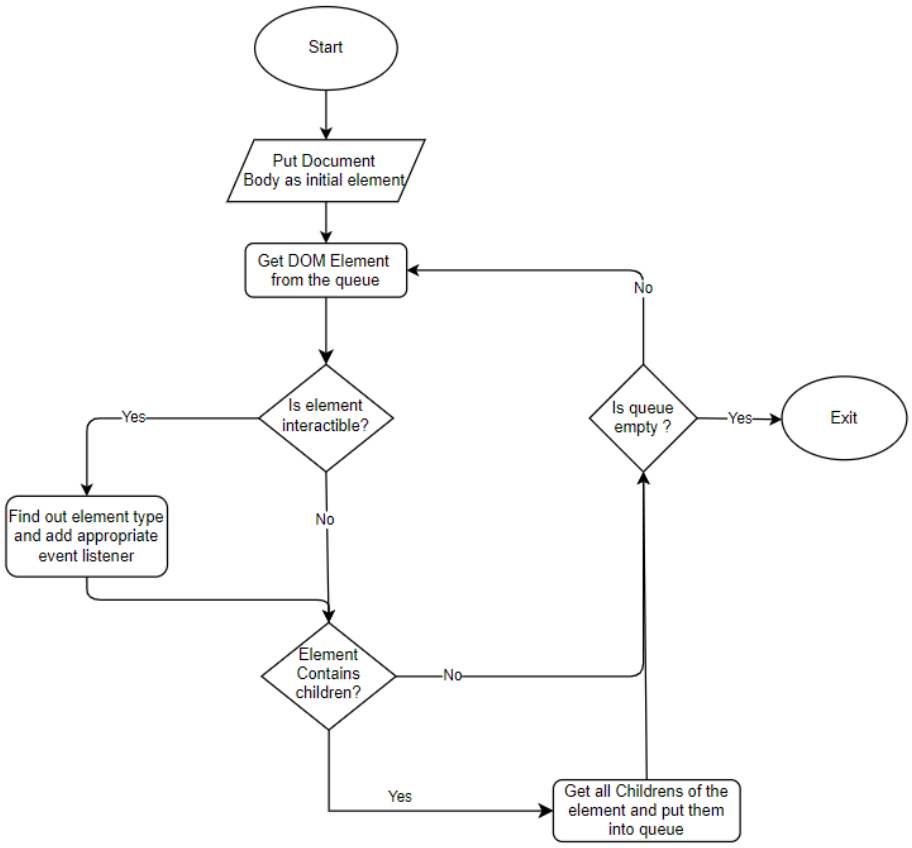


Рис. 3.9. – Блок–схема алгоритму зчитування користувацьких дій зі сторінки

Далі зчитану подію обробляє модуль Recorder, він формує об’єкт з послідовністю користувацьких дій, приклад його структури:

```
{
“startPage”: “https://dut.dl.edu.ua”,
“steps”: [
{ “type”: “click”,
        “locatorType”: “id”,
  “locator”: “#start– btn”
},
```

```
{
 “type”: “sendKeys”,
 “locatorType”: “class”,
 “locator”: “.login– field”,
        “data”: “slobodyanyck@gmail.com”}]}
```

Але для того, щоб правильно записати дію в об’єкт, потрібно спочатку оприділити як знайти елемент на сторінці, та які дії над ним виконати.

Функція DetermineLocator(Element) перевіряє чи об’єкт містить у собі якісь специфічні локатори, по типу ідентифікатора чи класу об’єкта. Загалом пріоритетність локаторів можна описати послідовністю:

1) ID.
2) Name.
3) Class.

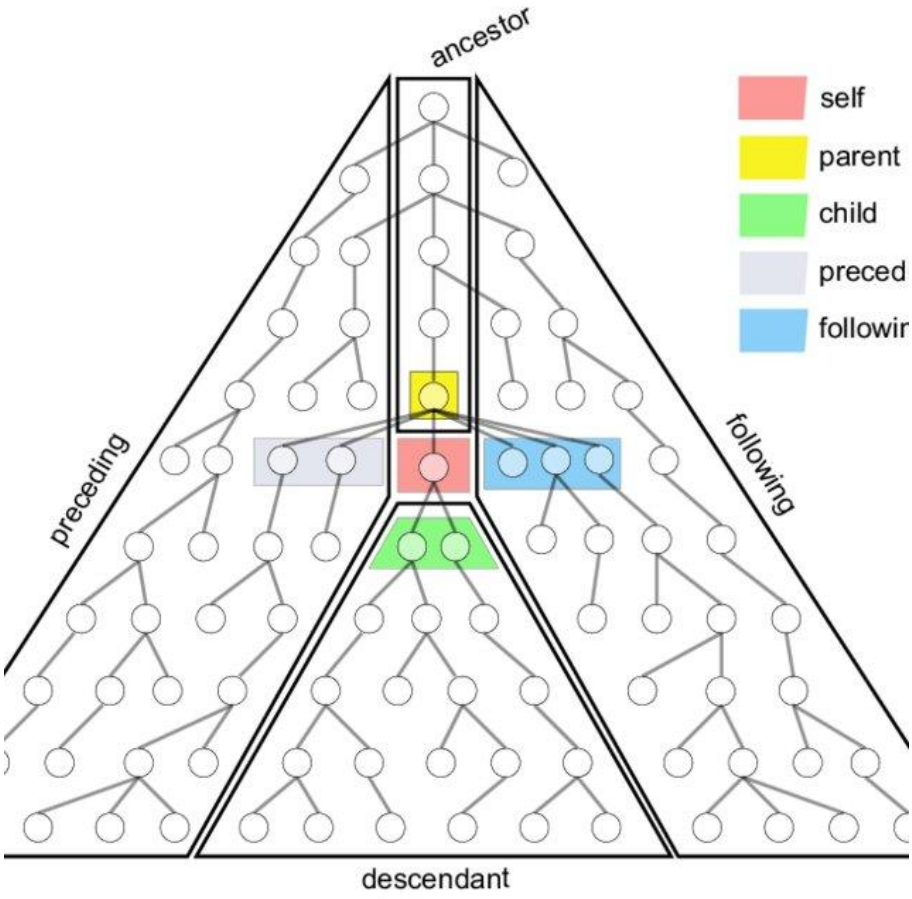


Рис. 3.10. – Ієрархія елементів сторінки у структурі Xpath

Якщо ж у об’єкта немає ніяких специфічних ідентифікаторів, то буде викликана функція GetXpath(Element), яка вирахує шлях до вузла на сторінці:

Xpath буває двох типів:

1) Повний – вираховується відносно 1 вузла на сторінці, тобто тега <html>
   Приклад:
   /html/body/section[6]/div/div[2]/div[4]/div
2) Короткий – вираховується відносно 1 елемента на сторінці, у якого є унікальний ідентифікатор.
   Приклад: //*[@id="services"]/div/div[2]/div[4]/div

Як працює алгоритм оприділення Xpath? Коли спрацьовує подія надходження нової користувацької дії, ми в якості параметра отримуємо в ній посилання на вузол у якому відбулась дія, а завдяки стандарту DOM у кожного вузла є посилання на батьківський, сусідні та дочірні елементи, тому алгоритм рухається по дереву вверх, із реєстрацією пройдених вузлів, поки не зустріне елемент з назвою <html> або елемент з унікальним ідентифікатором, якщо потрібно знайти вкорочений xpath (Рис. 3.10.).

GetActionType(Action) – визначає тип події, яку було зроблено над елементом, це може бути просто клік, вибір опції, введення тексту.

Знайшовши всю необхідну інформацію про подію, яку зробив користувач в браузері ми можемо передати її в ActionLogInterpreter чи JSLogInterpreter. Суть роботи цих модулів – конвертувати події в код на мові програмування JS з використанням бібліотеки WebDriverJS, який можна потім використати для побудови автоматизованого сценарію для тестування чи у випадку JSLogInterpreter просто записувати хронологію користувацьких подій на сторінці.

Модуль Renderer містить у собі сукупність функцій які відповідають за графічну частину розширення – відображення вкладок зі згенерованим кодом, обробка дій користувача та ін.

HTTP Модуль відповідає за зв’язок з додатком– плеєром тестових сценаріїв, а саме отримання статусу з’єднання та безпосередньо передачу

записаних тестових сценаріїв.

3.4. Функціонал десктопного додатку та його модулі

Функціонал десктопної частини додатку– плеєра тестових сценаріїв поділений на такі модулі (Рис. 3.11.):

- Отримання скрипта (HTTP Module);
- Графічне відображення скрипта (Renderer);
- Збереження, оновлення, видалення скрипта (DataWorker);
- Відтворення скрипта (Player);
- Експорт сценарію (Exporter);

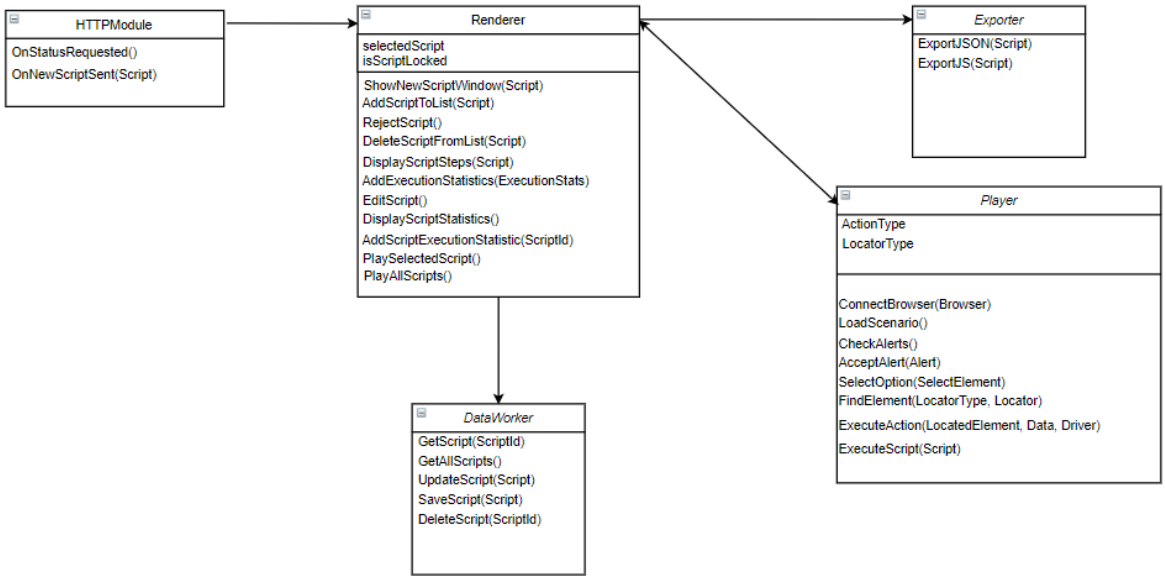


Рис. 3.11. – Схема модулів плеєра тестових сценаріїв

Модуль HTTP відповідає за підтримку з'єднання з додатком– записувачем тестових сценаріїв, отримання записаних скриптів, та їхнє декодування з тексту в JSON об'єкт.

Далі викликається вікно за допомогою модуля Renderer, в якому

відображається новоприбулий скрипт, який користувач має змогу як відхилити так і зберегти. Вціолому Renderer відповідає за відображення списку тестових сценаріїв, їх кроків, та запуску різних дій над ними, таких як редагування, видалення, відтворення. Модуль використовує архітектуру Electron JS, тобто додаток має набір HTML сторінок, які керуються обробниками подій у JS скриптах, що вимагає використання патерна MVC (Model – View – Controller) для належної побудови додатку.

Модуль DataWorker слугує для роботи з даними, та містить у собі інтерфейс підключення до MongoDB, та CRUD(Create Read Update Delete) операції, які потрібні для належної роботи з інформацією.

Найважливішим модулем в архітектурі додатку– плеєра являється модуль самого ж плеєра (Player), в ньому містяться функції які інтерпретують команди з записаного сценарію в інструкції для браузера, керованого WebDriver JS. Варто зауважити що всі методи даної бібліотеки являються асинхронними, тобто коли вони виконується, то основний потік програми за замовчуванням не очікує моменту їхнього завершення, тому що дії над браузером можуть виконуватись різний проміжок часу. Це враховано при побудові структури плеєра, та додано async/await інструкції для коректного функціонування програми.

```
const selectOption = async function (selectElement, itemToSelectText) {
  await selectElement.click();
  let selectOptions = await selectElement.findElements(By.tagName("option"));
  selectOptions.forEach(async (selectOption) => {
    let optionText = await selectOption.getText();
    console.log(optionText);
    if (optionText === itemToSelectText) {
      selectOption.click();
    }
  });
};
```

Якщо ж говорити про алгоритм відтворення сценарію, то він спочатку завантажує всі дії в чергу, в циклі бере кожну дію , знаходить локатор елемента

над яким вона відбулась, оприділяє тип дії, який потрібно виконати над елементом, виконує і так поступово поки не будуть оброблені всі дії в черзі.

Задля безпеки та оптимізації алгоритму (зменшення кількості лишніх дій), всі логічні кроки алгоритму було огорнуто в try\catch конструкції, та додано прапорці які ідентифікують статус виконання дії, якщо ж хоч один крок був провалений, то виконування сценарію припиняється та виконання вважається неуспішним (Рис. 3.12.)

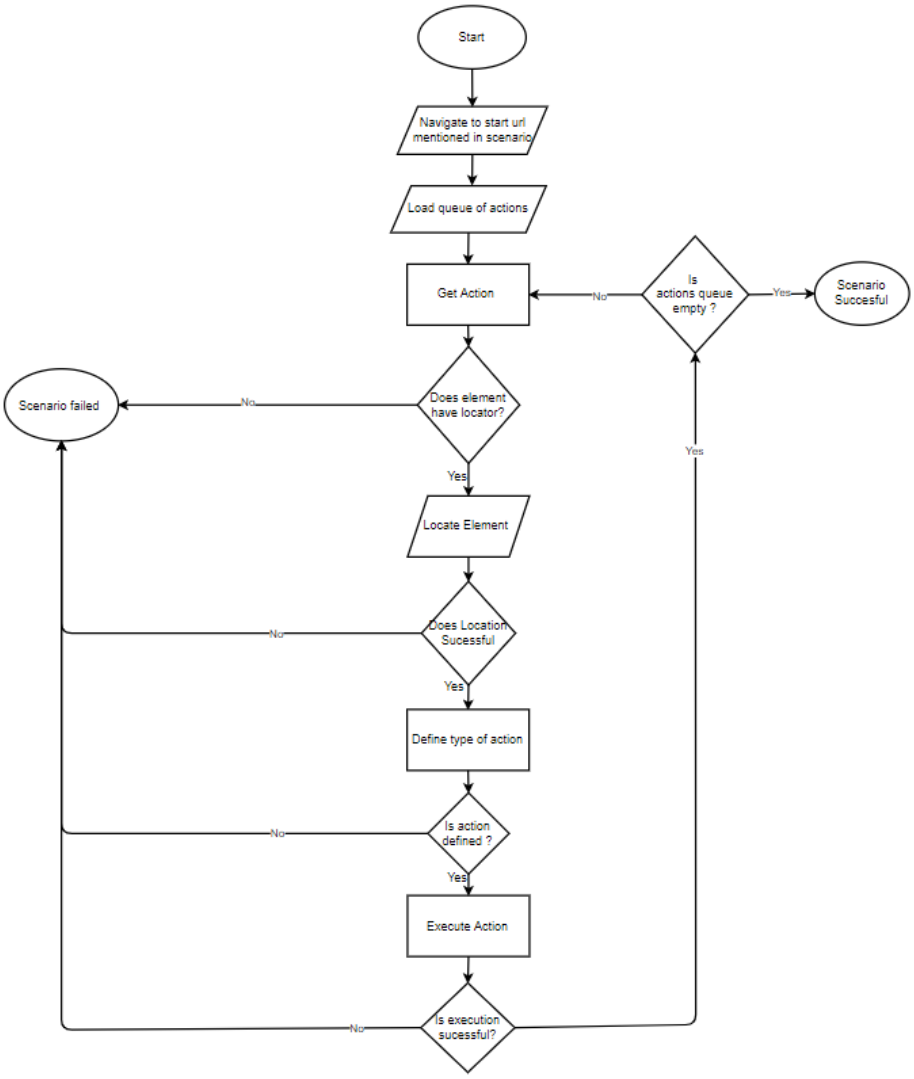


Рис. 3.12. – Блок схема алгоритму відтворення тестових сценаріїв

Модуль Exporter слугує для формування файлу тестового сценарію, який можна зберегти на диск комп’ютера для подальшого використання з метою модифікації сценарію, чи інтеграції в автоматизований тест– сет.

4. ПРИКЛАДИ ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1. Опис роботи системи при автоматизації тестування веб–сайту

4.1.1. Приклад використання Записувача сценаріїв

Як вже було зазначено вище, запис дій користувача на веб сторінці відбувається за допомогою браузерного розширення, яке потрібно додати в браузер, це можна зробити на сторінці «Розширення» (Рис. 4.1.).

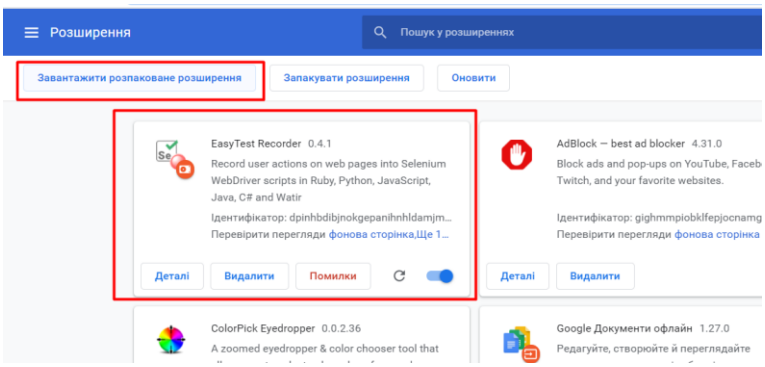


Рис. 4.1. – Встановлення розширення

Після того як розширення було успішно встановлено, достатньо всього скористатись контекстним меню «Розширення» та викликати встановлений записувач тестових сценаріїв (Рис. 4.2.).

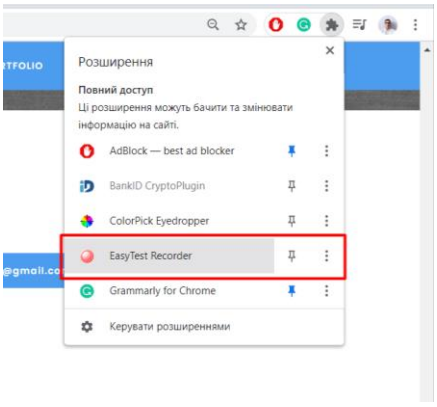


Рис. 4.2. – Запуск розширення

Після того як користувач запустив розширення, в окремому вікні з’явиться програма записувача.

Як бачимо, графічний інтерфейс складається з декількох вкладок зі згенерованими сценаріями. В адресному рядку розширення відображається веб– сайт з якого відбувається запис. За замовчуванням додаток завантажується в режимі «активного записування», зроблено це для більшої зручності користування, та зменшення надлишкових дій.

Кожна дія користувача на сторінці відображається у вікні Action Log, з локатором елемента над яким відбулась дія, тип дії, та дані які були використані для дії (Рис. 4.3.).

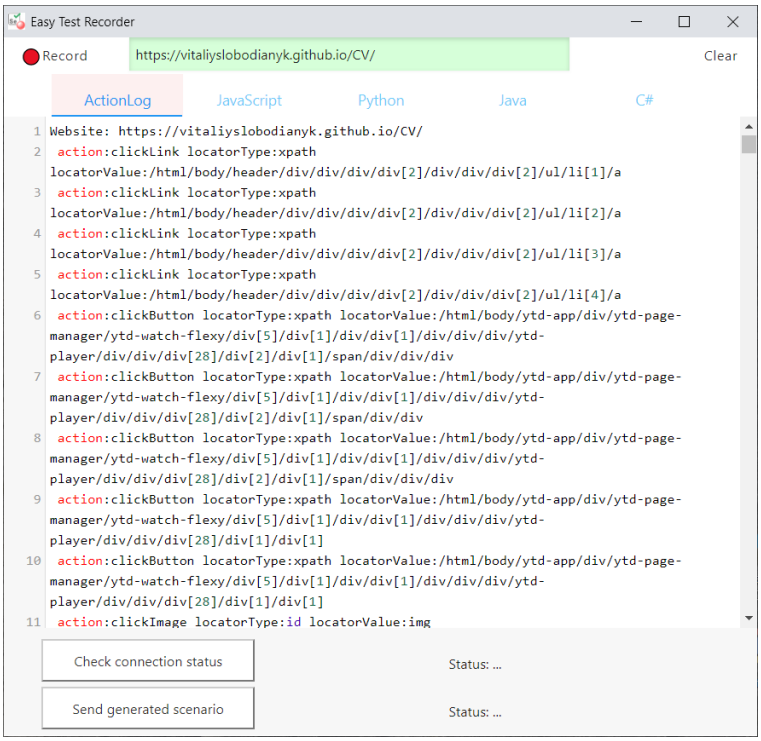


Рис. 4.3. – Приклад роботи розширення для запису в процесі створення тестового сценарію

За допомогою кнопок «Record\Stop» можна зупиняти та активувати запис дій на сторінці. Також можна очистити всі записані кроки за допомогою «Clear». Також якщо переключитись на інші вкладки, наприклад «» то можна побачити що паралельно із записом користувацьких дій відбувається генерація

автоматизованого тестового сценарію, за допомогою розробленого інтерпретатора, який конвертує записані дії в рядки сценарію на вибраній мові програмування.

За допомогою кнопки «Copy» можна скопіювати згенерований авто– тест та використати у своїх цілях в середовищі розробки автоматизованих сценаріїв.

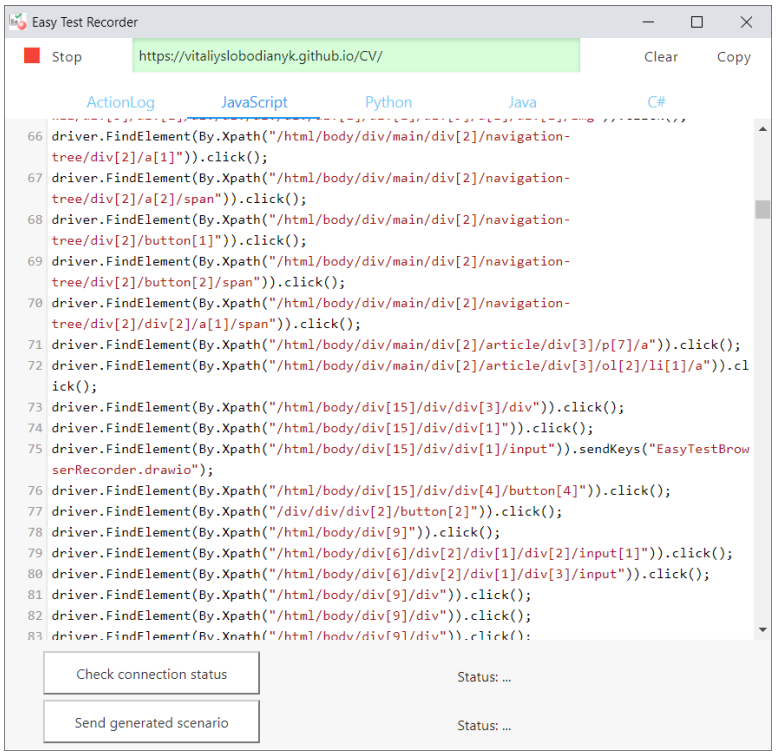


Рис. 4.4. – Вкладка зі згенерованим скриптом тесту мовою JS

На нижній панелі додатку відображені кнопки взаємодії з десктопним–додатком плеєром тестових сценаріїв.

«Check connection status» – ця функція слугує для того, щоб додаток встановив підключення з десктопною частиною, у випадку ж якщо плеєр не запущений, то буде відображена помилка з проханням його запустити.

«Send generated scenario» – спрацює, якщо підключення пройшло успішно. Та слугує для передачі записаного тестового сценарію в плеєр (Рис. 4.4.).

4.1.2 Приклад використання Плеєра тестових сценаріїв

Після того, як користувач відправив записаний сценарій, десктопний

відтворювач повідомляє користувача, що був записаний новий скрипт:

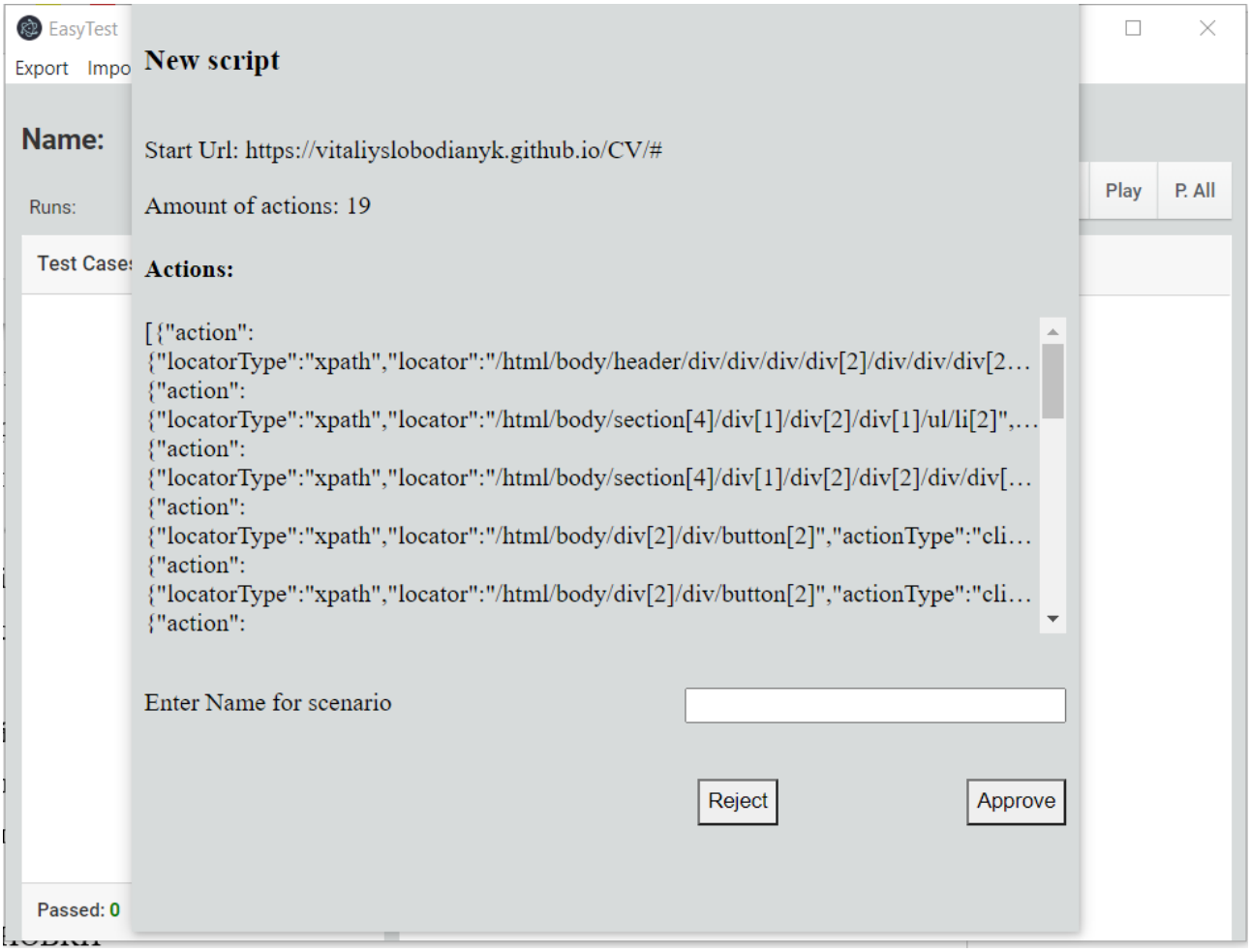


Рис. 4.5. – Вікно збереження нового сценарію

Відобразиться вікно з даними сценарію, веб– сторінка на якій він був записаний, кількість кроків для відтворення, та коротка інформація про кожен крок, щоб у користувача була можливість зрозуміти що це за сценарій.
На даному етапі можливо ввести ім’я сценарію, та зберегти його у список тест кейсів, тоді він запишеться в базу даних та буде доступним для відтворення та перегляду статистики, або ж відмовитись від збереження, якщо записано непотрібний чи невірний тестовий сценарій.

Коли користувач зберігає новий сценарій, він відображається в загальному списку сценаріїв:

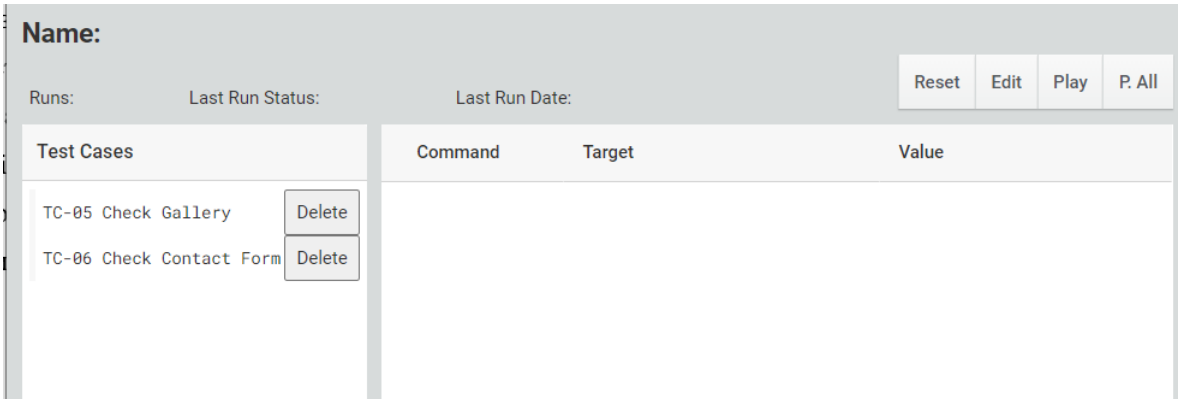


Рис. 4.6. – Список тестових сценаріїв

Кожен сценарій можна видалити, відтворити, скинути метадані виконання та відредагувати назву (Рис. 4.6.).

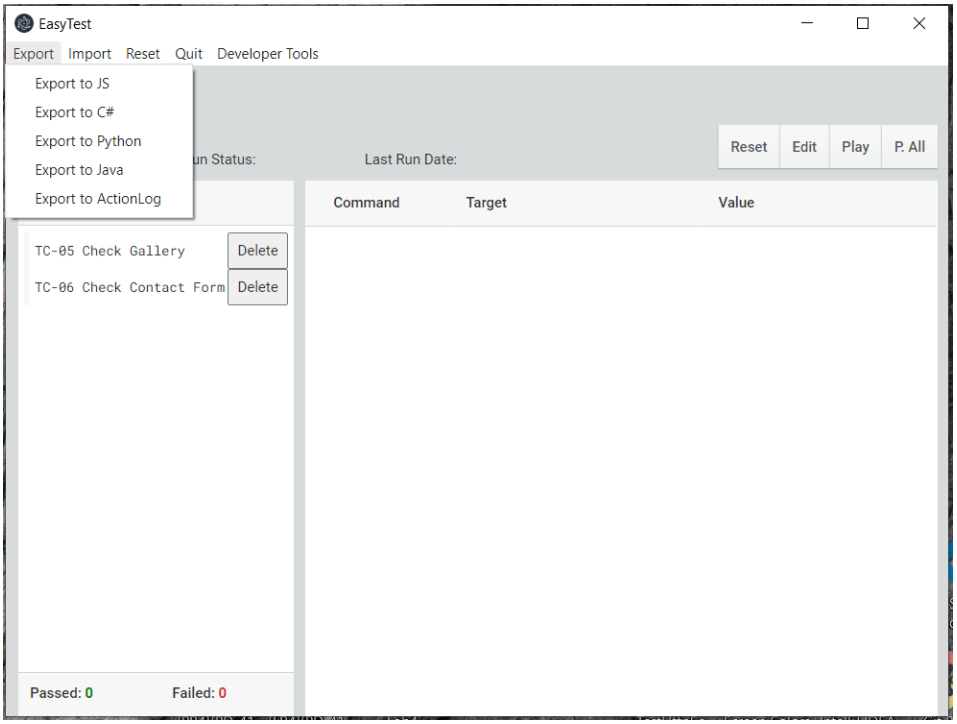


Рис. 4.7. – Додаткові функції системи

Також з додаткового функціоналу, його можна експортувати на одній з представлених мов програмування: JS, C#, Python, Java та Action Log – по суті являє собою JSON документ, яким скрипт записується в базу даних (Рис. 4.7.):
Як виглядає експортований сценарій в JS:

```
JS TC-05 Check Gallery.js U ×
generatedTests > JS TC-05 Check Gallery.js > ...
1  var webdriver = require('selenium-webdriver'), By = webdriver.By;NaN
2  var driver = new webdriver.Builder().forBrowser('chrome').build();
3  driver.get("https://vitaliyslobodianyk.github.io/CV/#");
4  driver.FindElement(By.Xpath("/html/body/header/div/div/div/div[2]/div/div/div[2]/ul/li[3]/a")).click();
5  driver.FindElement(By.Xpath("/html/body/section[4]/div[1]/div[2]/div[1]/ul/li[2]")).click();
6  driver.FindElement(By.Xpath("/html/body/section[4]/div[1]/div[2]/div[2]/div/div[3]/div/a/img")).click();
7  driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
8  driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
9  driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
10 driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
11 driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
12 driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
13 driver.FindElement(By.Xpath("/html/body/div[2]/div/div[1]/div/button")).click();
14 driver.FindElement(By.Xpath("/html/body/section[4]/div[1]/div[2]/div[1]/ul/li[3]")).click();
15 driver.FindElement(By.Xpath("/html/body/section[4]/div[1]/div[2]/div[2]/div/div[2]/div/a/img")).click();
16 driver.FindElement(By.Xpath("/html/body/div[2]/div/div[1]/div/button")).click();
17 driver.FindElement(By.Xpath("/html/body/section[4]/div[1]/div[2]/div[1]/ul/li[4]")).click();
18 driver.FindElement(By.Xpath("/html/body/section[4]/div[1]/div[2]/div[2]/div/div[1]/div/a")).click();
19 driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
20 driver.FindElement(By.Xpath("/html/body/div[2]/div/button[2]")).click();
21 driver.FindElement(By.Xpath("/html/body/div[2]/div/div[1]/div/button")).click();
22 driver.FindElement(By.Xpath("/html/body/section[7]/div/div/div[2]/div/a")).click();
23 driver.FindElement(By.Xpath("/html/body/div[1]/header[1]/div[3]/div[2]/ul/li[2]/a")).click();
```

Рис. 4.8. – Приклад експортованого сценарію на мові програмування JS

Та експортований сценарій у форматі Action Log(JSON):

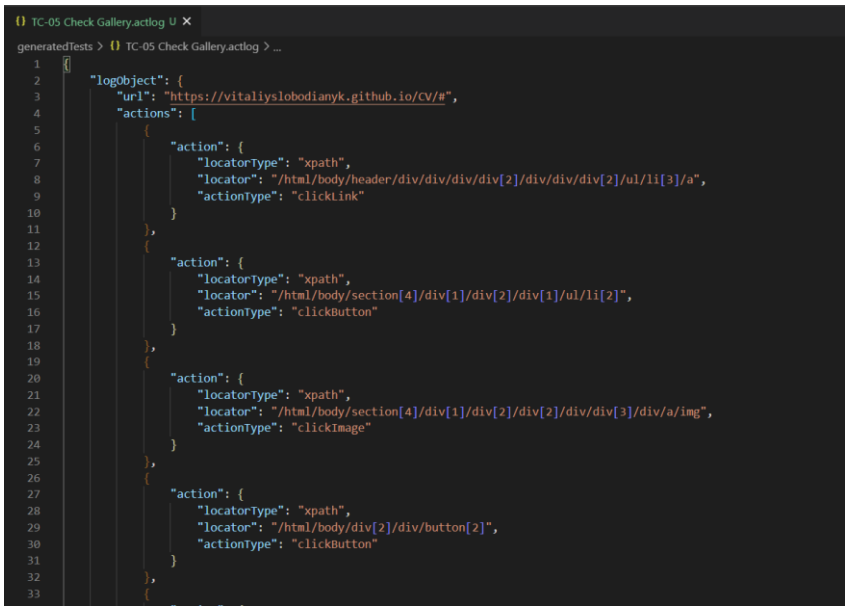


Рис. 4.9. – Приклад експортованого сценарію в форматі JSON

Коли користувач вибирає тестовий сценарій, який він хоче виконати, в головній секції програми відображаються кроки тестового сценарію, кожен крок має такі атрибути як: тип виконуваної дії, шлях до елемента, та дані, які передаються елементу в тестовому кроці. Також в головній секції сценарію

відображаються такі важливі метадані тестового сценарію як назва, кількість запусків, дата останнього запуску, та його статус(пройдено\не пройдено) (Рис. 4.10.):

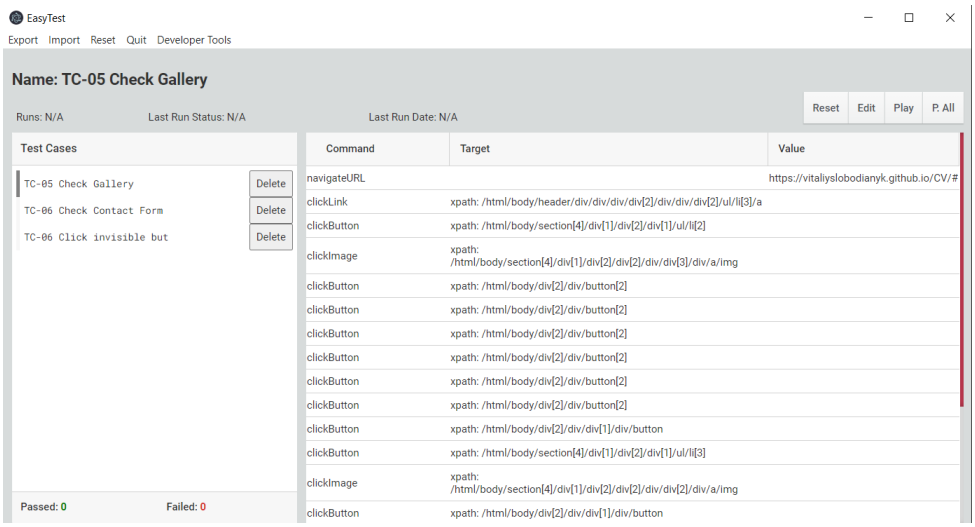


Рис. 4.10. – Перегляд кроків та метаданих сценарію

Після того як користувач запустив виконання сценарію, відкривається окреме вікно браузера у якому система відтворює кроки тесту, кожен успішно пройдений крок підсвічується зеленим в головному списку кроків, відповідно якщо якийсь крок не вдасться успішно виконати він буде підсвічений червоним (Рис. 4.11.).

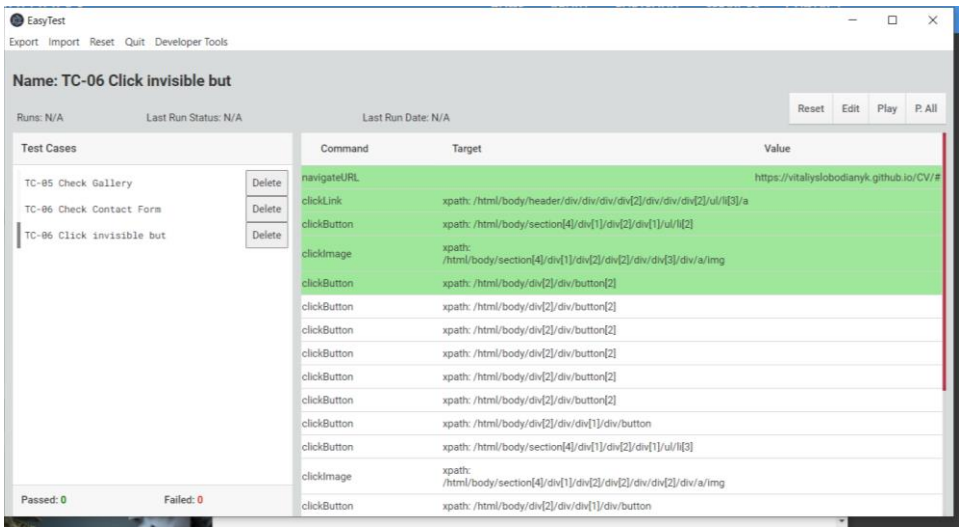


Рис. 4.11. – Відображення статусу виконання кроків сценарію

Така ж сама логіка спостерігається для списку тест кейсів, ті які пройдені успішно, відмічені зеленим кольором, ті які провалені – червоним, які ще не були запущені – мають нейтральний колір.

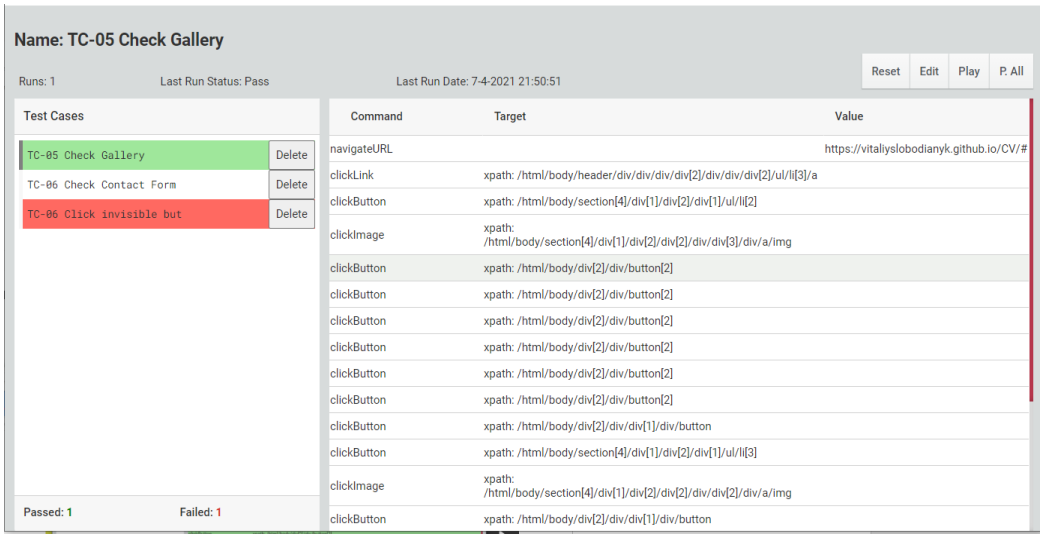


Рис. 4.12. – Відображення статусу та статистики виконання сценаріїв

Також, для простежуваності статистики тестування, відображено такі метрики як кількість успішно пройдених тест–кейсів та кількість провалених, що в подальшому допомагає звітування (Рис. 4.12.).

4.2. Набір тестових сценаріїв для забезпечення якості продукту

Для забезпечення належного рівня якості розробленого програмного продукту, та високого рівня надійності розробленої системи, було вирішено підійти з особливою увагою до її тестування.

ID	Summary	Roles	Preconditions	Steps	Expected Result	Status
TC-01	[Recorder] Record actions from the web-site	Recorder	User has opened recorder app on the web site.	1. Press [Record] button; 2. Click on some elemen\button on the web-page; 3. Type in some data the input field. 4. Repeat click on differen ui elements; 5. Navigate to another page from the tested page; 6. Repeat Steps 2-4; 7. Press [Stop] button;	1. Application has started a recording of actions; Start url has been added to the log; 2. Appropriate action of click has been recorded and displayed in the log; 3. Appropriate action of input has been recorded and displayed in the log; 4. All click events are recorded into the log; 5. Navigation action has been recorded; 6.All events are recorded; 7. Script has been recorded;	Pass
TC-02	[Recorder] Testing of recording logic	Recorder	User has opened recorder app on the web site.	1. Click on some element\button on the web-page; 2.Press [Record] button; 3. Repeat Step1; 4. Press [Stop] button; 5. Repeat Step1;	1. Any actions should be recorded; 2. Recording of action has been started; 3. All actions are recording into the log; 4. Recording of actions should be stopped; 5. Any actions should be recorded;	Fail
TC-03	Send Recorded Actions	Recorder, Player	User has recorded some test scenario. User has opened recorder app on the web site. Player app is opened;	1. Click [Check Connection Status] button; 2.Press [Send Generatd Scenario]button; 3. Verify that amount of steps correspond to amount of recorded steps; 4. Verify recorded steps;	1. System shoud show "Connected" status; 2.Recorder app shoud show window with new scenario; 3. Amount of steps should be the same; 4.Steps should match with steps from recorder;	Pass
TC-04	Accept Recorded Actions	Player	User has recorded and send some scenario to player app; "New Script" window with sent script is displayed;	1.Enter Script Name; 2. Press [Accept] button; 3.Resend recorded script from recorder; 4.Enter Script Name; 5. Press [Reject] button;	1. Name is entered; 2. Script is accepted and displayed in lists of scenarios with given name; 3. "New Script" window is displayed; 4. Name is entered; 5. Script is rejected and not displayed in list of scenarios;	Pass

Рис 4.13. – Приклад функціональних тест кейсів для тестування системи

Було розроблено тест– сет для проведення функціонального тестування системи, з акцентом на негативні та конфліктні сценарії та їх обробку.

Для його розробки використовувались такі техніки тест дизайну, як передбачення помилок на основі попереднього досвіду, діаграми станів та переходів, тестування засноване на тест сценаріях юз– кейсів. Також було проведено багато статичного тестування (аналізу коду програми) по типу тестування білого ящика, в якому основним процесом являється статичний перегляд коду та тестування рішень і тверджень в коді (Рис. 4.13.).

4.3. Результати апробації та подальший розвиток проекту

Апробація даного продукту була проведена на моєму поточному місці роботи, та у випробуваннях розробленого інструменту взяли участь 5

висококваліфікованих співробітників у сфері тестування програмного забезпечення з компанії Infopulse. Протягом тестів було виявлено низку сильних сторін програми, так і деякі недоліки.

Основними перевагами виявились:

– Простота використання;
– Стабільність роботи та відтворення тестових сценаріїв;
– Продуманий користувацький інтерфейс;
– Можливість генерації автоматизованих тестових сценаріїв;

Так і знайшлось декілька недоліків, таких як:

- Неточність локації елементів на динамічних сайтах;
- Неможливість згрупувати тестові сценарії по секціях;

Так як розроблений продукт, показав досить хороші якості використання під час апробації, та показав що може покривати базові вимоги тестувальника програмного забезпечення, та полекшувати його роботу, було сформовано план подальшого розвитку продукту та розробки розширень і покращень для системи:

- Покращення алгоритму локації елементів на динамічних сайтах;
- Можливість групування тестових сценаріїв;
- Можливість редагування кроків тестових сценаріїв за допомогою конструктора;
- Розширення набору метрик та діаграм по результатах проведеного тестування;

Даний план включає як і пункти, які націлені на усунення теперішніх недоліків продукту, так і додаткові покращення, які дозволять зробити використання програми більш приємним та ефективним для цільової аудиторії.

ВИСНОВКИ

Дана робота була спрямована на проектування та реалізацію інструменту для автоматизації тестування веб–сайтів шляхом запису та відтворення тестових сценаріїв.

1. Було обгрунтовано актуальність розробленої системи та її наукову новизну шляхом аналізу використання інструментів автоматизації типу «Запису-Відтворення». Було досліджено типові задачі робітника сфери забезпечення якості ПЗ та виявлено, що сучасна індустрія потребує інструментів, які можуть зменшувати час витрачений на виконання тестів, одночасно збільшуючи ефективність роботи тестувальника. Задля розуміння концепції майбутньої системи було проведено аналіз існуючих інструментів автоматизації тестування.
2. Було розроблено розділену архітектуру додатку з використанням браузерного розширення для запису сценаріїв та десктопного відтворювача тестів, оскільки це обумовлено функціональною специфікою програми.

Було вибрано інструменти для побудови системи, такі як Chrome Extensions, Node JS, Mongo DB, Electron JS, Selenium Web Driver, обгрунтовано їх перевагу та ефективність над іншими аналогами.

Для того щоб розробити успішний продукт, було проаналізовано типічні завдання тестувальника програмного забезпечення та створено набір вимог у вигляді UML діаграми.

В ході розробки системи було спроектовано та реалізовано унікальний алгоритм перехоплення користувацьких дій на веб-сторінці та їх передачі за допомогою технології HTTP між 2 частинами системи. Було організовано спосіб передачі та збереження тестових сценаріїв зі створенням специфічних документів формату JSON та використання Mongo DB для їх збереження. Також було розроблено, не маючий аналогів метод відтворення згенерованих тестових сценаріїв на основі Selenium Web Driver.

3. Описано програмні засоби та інструменти, котрі було застосовано для розробки програмного забезпечення. Виявлено що середовище розробки VS

Code є найбільш зручним для виконання поставлених задач. Підібрано допоміжні інструменти розробки, такі як Postman для тестування розроблених API,та MongoDB Compass для тестування БД. Менеджмент та відстеження статусу виконання проекту було здійснено за допомогою інструменту Trello.

4. Високу якість та відповідність системи визначеним вимогам забезпечує розроблений набір тестових сценаріїв, який покриває всі функціональні вимоги системи, з урахуванням граничних випадків, та перевірки їх обробки системою.

5. Розроблена система цілеспрямована на допомогу спеціалістам із забезпечення якості програмного забезпечення, в сфері тестування веб–застосунків. Було з’ясовано що найкраще інструмент підходить для побудови тестових сценаріїв регресійного типу та також для генерації початкових скриптів автоматизації тестування розробленого функціоналу. Визначено, що інструмент може використовуватись як і в навчальних цілях для спеціалістів-початківців, так і в реальних робочих умовах, для виконання поставлених задач тестування.

6. Роботу було апробовано серед висококваліфікованих спеціалістів в сфері тестування програмного забезпечення компанії Infopulse, у апробації взяли участь 5 осіб з рівнем знань та досвідом від Middle до Senior. Було проаналізовано та виявлено ряд переваг та недоліків системи. Також було розроблено план покращень та розширень для системи.Виявлено, що основними перевагами програми є:

- Простота використання;
- Стабільність роботи та відтворення тестових сценаріїв;
- Продуманий користувацький інтерфейс;
- Можливість генерації автоматизованих тестових сценаріїв;

Результати дослідження бакалаврської роботи апробовані на всеукраїнських науково-технічних конференціях: "Застосування програмного забезпечення в інфокомунікаційних технологіях" та «Сучасний стан та перспективи розвитку IOT»

ПЕРЕЛІК ПОСИЛАНЬ

1. Elfriede Dustin, Jeff Rashka, John Paul. Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance./- 2007 – с. 277–295.
2. Roy Sutton. Enyo: Up and Running: Build Native-Quality Cross-Platform JavaScript Apps. /- “O'Reilly Media”, 2015 – с. 87.
3. Google Chrome API Reference [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://developer.chrome.com/docs/extensions/reference/ Дата звернення: 10.04.2021
4. Simon Bisson. Easy cross-platform app dev with GitHub's Electron. -/ “InfoWorld”, 2015 – с. 200–210.
5. Technical Differences Between Electron and NW.js (formerly node-webkit). [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://github.com/electron/electron/blob/master/docs/development/atom-shell-vs-node-webkit.md Дата звернення: 11.04.2021.
6. Fielding Roy. Architectural Styles and the Design of Network-based Software Architectures./- 2000 – с. 180 – 185.
7. Douglas Crockford. JSON: The Fat-Free Alternative to XML, 2009 – 55 p.
8. Mike Richardson, Ruby Leonard, Sam Amundsen. RESTful Web APIs./- “O'Reilly Media”, 2013 – с. 357 – 363.
9. Eric Elliott. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Moderns JS Libraries. -/ “O’Reilly Media”, 2013 – с. 134–177.
10. Leavitt Neal. Will NoSQL Databases Live Up to Their Promise./- “IEEE Computer”, 2010 – с. 43 – 45.
11. Офіційний веб-сайт Mongo DB. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.mongodb.com/– Дата звернення: 14.04.2021.

12. D. Armstrong. Learn XPath Fast: A beginner-friendly, exercise-based course for people who want to use XPath in Selenium, SQL Server, XQuery or anywhere else /- 2020 – с. 110–132.
13. Unmesh Gundecha, Satya Avasarala. Selenium WebDriver 3 Practical Guide: End-to-end Automation Testing for Web and Mobile Browsers with Selenium WebDriver, 2nd Edition. /- 2010 – с. 423–455.
14. Офіційний веб-сайт Selenim. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.selenium.dev/documentation/en/ – Дата звернення: 14.04.2021.
15. Laiza Krispin, Janette Gregory. Agile Testing: A Practical Guide for Testers and Agile Teams. /- 2010 – с. 215–247.
16. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестування програмного забезпечення. Фундаментальні концепції менеджменту бізнес-додатків./- 2001 – с. 344–375.
17. W Hetzel. Complete Guide to Software Testing (2e)./- “QED Information Sciences: Wellesley MA”, 1993 – с. 115–121.
18. A. Spillner, T. Linz, H. Schaefer. Software Testing Foundations (4e)./- “Rocky Nook: San Rafael CA”, 2014 – с. 223–229.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ДЕСКТОП ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ НА МОВІ ПРОГРАМУВАННЯ JS

Виконав студент 4 курсу
Групи ПД-41
СЛОБОДЯНИК В.В.
Керівник роботи
Негоденко О.В.

Київ – 2021

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

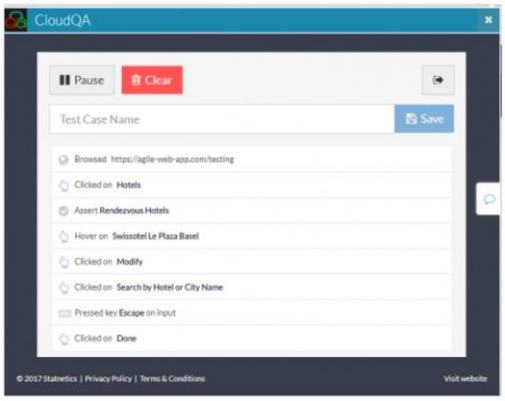
- **Мета роботи** – розробка програмного забезпечення для забезпечення контролю якості веб–сервісів та веб–додатків.
- **Об'єкт дослідження** – автоматизація процесів ручного тестування програмного забезпечення для підвищення ефективності роботи тестувальника програмного забезпечення та збільшення його продуктивності в сфері веб–тестування .
- **Предмет дослідження** – технологія розробки програмного забезпечення для запису та відтворення дій користувача на веб–сторінці.

АКТУАЛЬНІСТЬ РОБОТИ

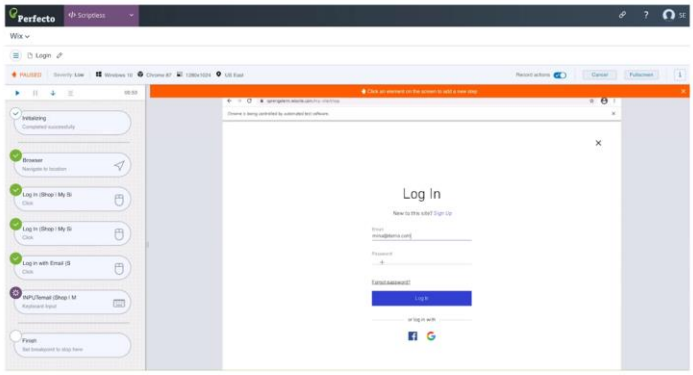
- Актуальність даної роботи заключається в тому, що розроблений інструмент допомагає ефективніше працювати спеціалістам із тестування ПЗ та автоматизовувати частину їхнього робочого процесу без знань мов програмування та написання коду автоматичних тестів.

- Розроблена система цілеспрямована на допомогу спеціалістам із забезпечення якості програмного забезпечення, в сфері тестування веб–застосунків. Було з'ясовано що найкраще інструмент підходить для побудови тестових сценаріїв регресійного типу та також для генерації початкових скриптів автоматизації тестування розробленого функціоналу. Визначено, що інструмент може використовуватись як і в навчальних цілях для спеціалістів-початківців, так і в реальних робочих умовах, для виконання поставлених задач тестування.

АНАЛОГИ



- CloudQA



- Perfecto

ПОРІВНЯННЯ З АНАЛОГАМИ

Назва продукту	Переваги	Недоліки
Cloud QA	1) Алгоритм локації елементів заснований на основі реєстрації подій на сторінці. 2) Присутні особисті кабінети користувача.	1) Підписка на платній онові 2) Відсутність можливості відредагувати крок тесту 3) Немає змоги експортувати записані сценарії у скрипт.
Perfecto	1) Просунуті засоби звітування виконання тестів. 2) Інструмент повністю інтегрований у браузер.	1) Архаїчний спосіб реєстрації дій за допомогою координат на сторінці. 2) Немає змоги експортувати записані сценарії у скрипт. 3) Підписка на платній онові.

ТЕХНІЧНЕ ЗАВДАННЯ

1. Дослідити можливості використання платформи Chrome Extensions, бібліотеки Selenium Web Driver засоби розробки і відлагодження програм на Java Script;
2. Розробити алгоритм захоплення користувацьких дій на веб-сторінці.
3. Розробити браузерне розширення для запису тестових сценаріїв з веб сторінки та їх подальшої обробки.
4. Розробити структури бази даних системи;
5. Розробити бібліотеку для генерації тестових сценаріїв для інструменту Selenium Web Driver на мовах програмування JS, C#, Python, Java.
6. Розробити десктопне програмне забезпечення відтворення тестових сценаріїв та їх аналізу.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

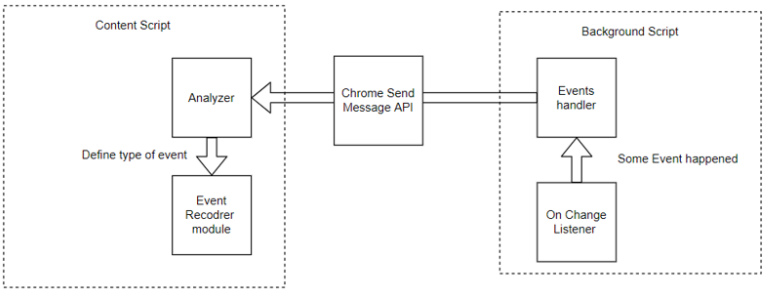
- Chrome Extensions – було використано власне для побудови браузерного розширення, що захоплює дії користувача;
- Node JS – використано для розробки API, що відповідає за зв'язок десктопної частини з розширенням, для розробки сервісу, що відтворює;
- Electron JS – використано для побудови користувацького інтерфейсу системи;
- Selenium WEB Driver – використано для розробки сервісу, що відтворює записані тестові сценарії на основі Node JS;
- Mongo DB – слугує для збереження записаних сценаріїв, та статистики їх виконання;



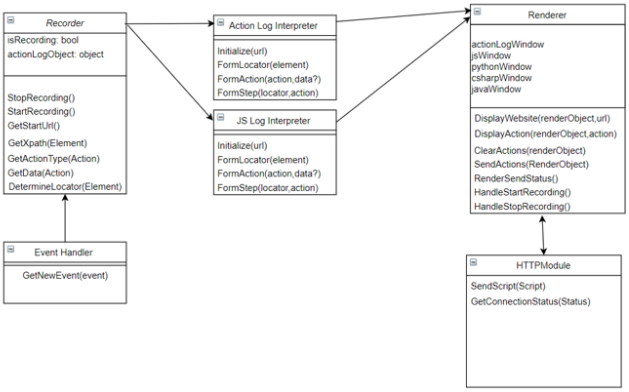
ІНСТРУМЕНТИ ВИКОРИСТАНІ ДЛЯ РЕАЛІЗАЦІЇ

- VS code – інтегроване середовище розробника, яке було використано для написання коду системи.
- Postman – Інструмент, який було використано для тестування розробленого API для комунікації
- Mongo DB Compass – інструмент для роботи з базою даних Mongo DB.
- Google Chrome Driver – драйвер відомого браузера Google Chrome, який саме використовується основним браузером
- Git – система контроля версій використовувалась для полекшення процесу розробки, та запобіганню ризиків пошкодити систему змінами.

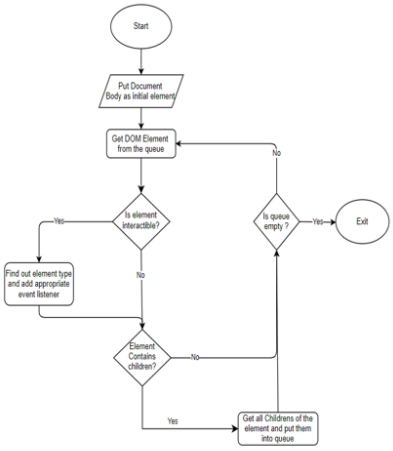

АРХІТЕКТУРА БРАУЗЕРНОГО РОЗШИРЕННЯ



МОДУЛІ БРАУЗЕРНОГО РОЗШИРЕННЯ

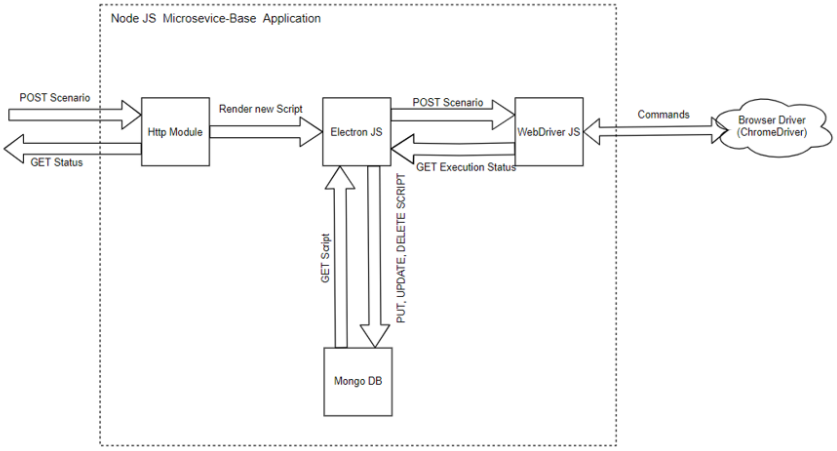


Структура модулів



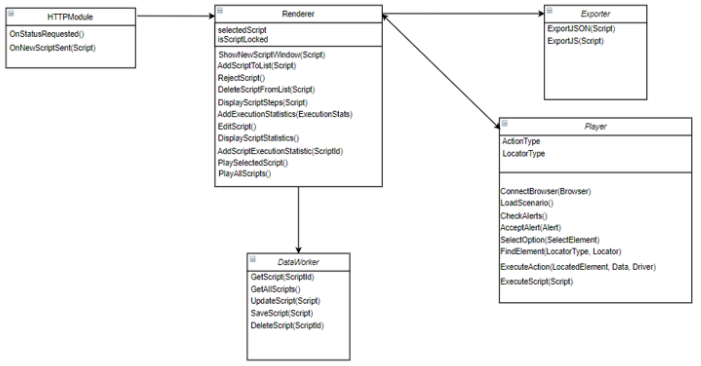
Алгоритм зчитування подій

АРХІТЕКТУРА ПЛЕЄРА ТЕСТОВИХ СЦЕНАРІЇВ

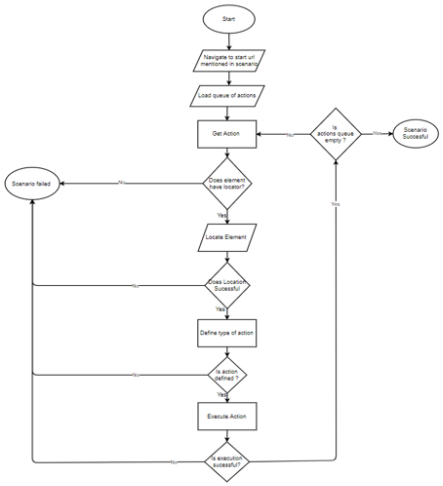


11

МОДУЛІ ПЛЕЄРА ТЕСТОВИХ СЦЕНАРІЇВ



Структура модулів



Алгоритм відтворення сценаріїв

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Слободяник В.В . Архітектура додатку для автоматизації запису та відтворення тестових сценаріїв веб-сторінок за допомогою JavaScript та Selenium Webdriver : Матеріали всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інфокомунікаційних технологіях». Збірник тез.\- 10.02.2021, ДУТ, м. Київ — К.: ДУТ, 2021. — С. 24 — 26.
2. Слободяник В.В. Побудова архітектури додатку для автоматизації запису та відтворення тестових сценаріїв веб-сторінок за допомогою JavaScript та Selenium Webdriver : Матеріали всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IOT». Збірник тез.\- 09.04.2021, ДУТ, м. Київ — К.: ДУТ, 2021. — С. 26 — 29.

Також розроблена система була апробована спеціалістами із забезпечення якості програмного забезпечення компанії Infopulse.

13

ВИСНОВКИ

1. Було обгрунтовано актуальність розробленої системи та її наукову новизну шляхом аналізу використання інструментів автоматизації типу «Запису-Відтворення».
2. Було розроблено розділену архітектуру додатку з використанням браузерного розширення для запису сценаріїв та десктопного відтворювача тестів.
3. Описано програмні засоби та інструменти, котрі було застосовано для розробки програмного забезпечення.
4. Було протестовано розроблену систему за допомогою набору функцыональних тестових сценарыъв та прийомочного тестування.
5. Було доведено, що розроблена система підходить для практичного використання спеціалістами забезпечення якості ПЗ та покриває їхні основні робочі задачі.

Перспективи подальших досліджень та розвитку даної роботи полягають в наступному:
1. Покращення алгоритму локації елементів на динамічних сайтах;
2. Можливість групування тестових сценаріїв;
3. Можливість редагування кроків тестових сценаріїв за допомогою конструктора;
4. Розширення набору метрик та діаграм по результатах проведеного тестування;
5. Підтримка виконання тестів на браузерах MS Edge, Mozilla, Opera, IE;

14

ДЯКУЮ ЗА УВАГУ!