Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Проектування та розробка модулів тестування для валідації подій у json форматі»

(тема роботи)

Виконав: студент 2-го курсу групи 1ІСТ-22м
(шифр групи)
спеціальності 126 – Інформаційні системи та технології
(шифр та назва спеціальності)
Олексій АБДУЛЛАЄВ
(ПІБ студента)
Керівник: к.т.н., доц. каф. АІІТ
Ілона БОГАЧ
(науковий ступінь, вчене звання / посада, ПІБ керівника)
«4» грудня 2023 р.
Опонент: к.т.н., доц. каф. АІІТ
Людмила КРИЛИК
(науковий ступінь, вчене звання / посада, ПІБ опонента)
«7» грудня 2023 р.

Допущено до захисту
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«11» 12 2023 р.

Вінниця ВНТУ – 2023 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

Рівень вищої освіти – ІІ-ий (магістерський)

Галузь знань – 12 – Інформаційні технології

Спеціальність – 126 Інформаційні системи та технології

Освітньо-професійна програма – Інформаційні технології аналізу даних та зображень

ЗАТВЕРДЖУЮ

Завідувач кафедри АІІТ

д.т.н., проф. Олег БІСІКАЛО

«20» 09 2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Олексію АБДУЛЛАЄВУ

1. Тема роботи: Проектування та розробка модулів тестування для валідації подій у json форматі

   Керівник роботи: к.т.н., доцент каф. АІІТ Ілона БОГАЧ

   Затверджені наказом ВНТУ від «18» 09 2023 року №247.
2. Строк подання студентом роботи до «20» листопада 2023 року.
3. Вихідні дані до роботи: операційна система Windows 10; мова програмування Python, веб-фреймворк Flask; програмне середовище PyCharm.
4. Зміст текстової частини: вступ; аналіз існуючих засобів тестування та аналогів систем тестування; огляд існуючих програмних засобів для реалізації поставленої задачі; розробка програмного забезпечення для валідації подій у json форматі; тестування програмного забезпечення; економічне обґрунтування; висновки; список використаних джерел.
5. Перелік ілюстративного (або графічного) матеріалу: UML діаграма модулю валідації; UML діаграма модулю перевірки події; діаграма класів модулю валідації подій; UML-class diagram модулю валідації подій; UML-object diagram модулю валідації подій; UML-deployment diagram модулю валідації подій; UML-use case diagram модулю валідації подій; відображення результатів валідації подій

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Ілона БОГАЧ, к.т.н., доцент каф. АІІТ		
5	Володимир КОЗЛОВСЬКИЙ, к.е.н., доцент каф. ЕПтаВМ		

7. Дата видачі завдання «20» 09 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз існуючих засобів тестування та аналогів систем тестування	20.09.23 - 02.10.23	виконано
2	Огляд існуючих програмних засобів для реалізації поставленої задачі	02.10.23 - 16.10.23	виконано
3	Розробка програмного забезпечення для валідації подій у json форматі	17.10.23 - 08.11.23	виконано
4	Тестування програмного забезпечення	09.11.23 - 14.11.23	виконано
5	Економічний розділ	15.11.23 - 17.11.23	виконано
6	Оформлення пояснювальної записки і графічного матеріалу	17.11.23 - 20.11.23	виконано
7	Попередній захист роботи	21.11.23	виконано
8	Остаточний захист роботи	11.12.23 - 22.12.23	виконано

Студент _____ (підпис) Олексій АБДУЛЛАЄВ (прізвище та ініціали)

Керівник роботи _____ (підпис) Ілона БОГАЧ (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.457

Олексій АБДУЛЛАЄВ Проектування та розробка модулів тестування для валідації подій у json форматі. Магістерська кваліфікаційна робота зі спеціальності 126 – Інформаційні системи та технології, освітньо-професійна програма – Інформаційні технології аналізу даних та зображень. Вінниця: ВНТУ, 2023. 122 с.

На укр. мові. Бібліогр.: 33 назв; рис.: 27; табл.: 8.

У магістерській кваліфікаційній роботі було досліджено підходи тестування правильності передачі подій користувачів та розроблено модуль тестування для валідації подій у json форматі з гнучким налаштуванням. Зручність системи полягає у налаштуванні списку потрібних для валідації подій, їх можливих значень. Розроблений модуль є гнучким у конфігурації, , а також кроссплатформеним.

Ключові слова: валідація, модуль тестування, події, алгоритм, програмне забезпечення.

ABSTRACT

Oleksii ABDULLAIEV Design and development of testing modules for events validation in json format. Master’s qualification paper of a specialty 126 – Informational systems and technologies, educational program – Informational technologies of data and image analysis. Vinnytsia: VNTU, 2023. 122 p.

In Ukrainian language. Bibliography: 33 titles; fig.: 27; tabl.: 8.

In the master's thesis, was studied approaches to testing the correctness of the transfer of user events and developed a testing module for event validation in json format with flexible configuration. The convenience of the system lies in the customization of the list of events required for validation, their possible values, and cross-platform compatibility. The developed module is flexible in configuration.

Keywords: validation, testing module, events, algorithm, software.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі люди отримують та обробляють інформацію за допомогою різноманітних програм. На даний час більш актуальним стає перевірка правильності роботи цих програм, аби підвищити їх якість та таким чином заохотити нових кінцевих користувачів обрати саме цей продукт. [1, 2]

Останніми роками в галузі тестування програмного забезпечення відбувся великий розвиток із новими трендами в послугах IT-галузі. Впровадження нових технологій принесло останні оновлення у розробці, розробці, тестуванні та постачанні програмного забезпечення [1-3]. Найвищим пріоритетом бізнесу в усьому світі є оптимізація витрат та швидкість виконання. І саме створення інструментів для оптимізації тестування допоможе зменшити час розробки та тестування програмного забезпечення.

Актуальним є створення модуля програмного забезпечення для тестування правильності передачі подій користувачів, що дозволить більш простіше та ефективніше розробляти додатки для кінцевих користувачів на різноманітних платформах таких як Web, Android, iOS і т.д. Зручність такої системи полягає в гнучкому та швидкому налаштуванні списку потрібних для тестування подій, а також налаштуванню їх можливих значень.

Метою роботи є розширення функціональних можливостей програмного забезпечення для проведення валідації подій у форматі JSON. Основним завданням є створення системи валідації подій з гнучкими налаштуваннями. У відміну від існуючих інструментів тестування, має прискорити та спростити процес валідації подій. Мета роботи буде досягнута через отримання підтвердження шляхом порівняння функціоналу розробленої системи з аналогами. Також запропонований підхід забезпечує можливість впровадження системи на різних додатках, в тому числі на Web, Android та iOS системах.

Для досягнення мети необхідно розв'язати наступні задачі:

1. Провести аналіз існуючих засобів тестування, валідації даних та аналогів систем тестування.

2. Провести огляд існуючих програмних засобів для реалізації поставленої задачі.

3. Розробити програмне забезпечення для валідації подій у json форматі.

4. Протестувати розроблене програмне забезпечення.

5. Забезпечити відображення результатів валідації подій.

Об'єктом дослідження є процес валідації подій за допомогою шаблонів.

Предметом дослідження є методи та засоби шаблонізації та валідації подій.

Методи дослідження. У роботі використовуються методи дослідження, а саме аналіз, моделювання, класифікація, узагальнення, спостереження, прогнозування та експерименту; використовується прагматична модель наукового дослідження.

Наукова новизна роботи полягає в тому, що запропонована модифікація програмного забезпечення для проведення валідації подій у форматі JSON, яка за рахунок використання мови програмування Python та безкоштовного вебфреймворку Flask дозволяє, на відміну від існуючих інструментів, дозволяє розширити функціональні можливості програмного забезпечення, а саме надати гнучкість налаштування для різного типу систем, суттєво спростити додавання подій для перевірки нового функціоналу, підтримувати перевірку результатів на різних платформах та надати детальну звітність щодо тестування.

Практична цінність роботи полягає в створенні програмного засобу валідації подій у json форматі на різних системах та з можливістю подальшої модифікації компонентів системи з використанням сучасних технологій розробки. Зручність даного модулю валідації подій значно покращить та пришвидшить роботу тестувальників.

Основною перевагою програмного продукту/системи є гнучке налаштування подій для перевірки (можливі значення, назву, кількість і т.д.). Розроблена

система, на відміну від існуючих, буде одразу визначити шаблон відправлених подій для певної платформи, визначивши мінімальні та максимально допустимі їх значення та обов’язковість інших полів, що значно спростить та пришвидшить роботу тестувальників, яким доводиться мати справу з перевіркою відправлених подій для аналітики. Розроблена система з мінімальними змінами у коді може бути імплементована у інші проекти чи використана як окремий продукт.

Апробація результатів роботи. Результати даної роботи було представлено на Всеукраїнській науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» на кафедрі інтелектуальних інформаційних технологій та автоматизації та опубліковані у вигляді тез доповіді [4] та на LI науково-технічній конференції підрозділів Вінницького національного технічного університету на кафедрі інтелектуальних інформаційних технологій та автоматизації та опубліковані у вигляді тез доповіді, 2022 рік [33].

Впровадження результатів роботи

Результати роботи було впроваджено в систему розповсюдження відео контекту в ТОВ «ПРАГМАТТЕК». Основний модуль роботи прийнятий на отримання свідоцтва авторського права на твір.

1 АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ ТЕСТУВАННЯ ТА АНАЛОГІВ СИСТЕМ ТЕСТУВАННЯ

1.1 Основні поняття тестування та валідації

Тестування програмного забезпечення - це процес перевірки системи з метою виявлення будь-яких помилок, прогалин або відсутніх вимог у порівнянні з фактичними вимогами. Тестування програмного забезпечення в цілому ділиться на два типи - функціональне тестування і нефункціональне тестування.

Тестування слід починати якомога раніше, щоб скоротити витрати і час на доопрацювання і створення програмного забезпечення без помилок, щоб його можна було доставити клієнту. Однак в життєвому циклі розробки програмного забезпечення тестування може бути розпочато з етапу збору вимог і продовжено до тих пір, поки програмне забезпечення не буде запущено у виробництво. Це також залежить від моделі розробки, яка використовується. [5]

Тестування програмного забезпечення має важливе значення, оскільки, якщо в програмному забезпеченні є помилки, їх можна виявити завчасно і вирішити перед доставкою програмного продукту. Правильно перевірений програмний продукт забезпечує надійність, безпеку та високу продуктивність, що в подальшому призводить до економії часу, економічності та задоволеності споживачів.

Тестування важливо, оскільки помилки програмного забезпечення можуть бути дорогими або навіть небезпечними. Помилки програм можуть потенційно спричинити грошові та людські втрати [5-6]

Unit Testing: цього підходу до тестування програмного забезпечення дотримується програміст для тестування модуля програми. Це допомагає розробникам зрозуміти, чи працює окрема одиниця коду належним чином чи ні. [7]

Тестування інтеграції: воно зосереджується на побудові та дизайні програмного забезпечення. Ви повинні бачити, що інтегровані блоки працюють без помилок чи ні.

Тестування системи: за цим методом ваше програмне забезпечення компілюється як одне ціле, а потім перевіряється як одне ціле. Ця стратегія тестування перевіряє функціональність, безпеку, портативність.

Валідація — це процес визначення того, чи відповідає перевірене та створене програмне забезпечення вимогам клієнта або користувача. Обґрунтування або сценарії бізнес-вимог мають бути ретельно перевірені. Тут потрібно перевірити чи функції програмного забезпечення відповідають документації

Під час перевірки правильності розробки програмного забезпечення велику увагу приділяють перевірці кінцевого результату. Для цього на кожному кроці потрібно уважно перевірити як саме обробляється програмний код та які дані програма надає у кінці обробки коду. Навіть один етап який не пройшов перевірку може сильно вплинути на кінцевий результат.

Валідація кожного етапу підтверджує що створений програмний продукт відповідає складеній документації. При валідації отриманий результат порівнюють з очікуваним. Наприклад, одним з процесів тестування додатків є валідація даних які надсилаються. Під час надсилання подій користувача потрібно аби дані які надсилались відповідали дійсності. У такому випадку, аналітика додатку буде точною і дасть можливість тестувальникам, у разі потреби, знайти в чому саме проблема.

Тестування можна класифікувати на три категорії: функціональне тестування, нефункціональне тестування або тестування продуктивності та тестування змін.

Функціональне тестування — це одна з форм тестування, яка використовується для того, щоб переконатися, що функціональні вимоги програмного забезпечення відповідають його фактичним характеристикам. Основна мета функціонального тестування полягає в тому, щоб переконатися, що створений програмний продукт має всі функції, які потрібні клієнту [7].

Нефункціонального тестування відноситься до оцінки атрибутів, які не пов’язані з функціонуванням системи. Ці характеристики встановлюються нефункціональними критеріями, які визначають продукт з точки зору таких речей, як:

- надійність;
- працездатність;
- простота у використанні;
- масштабованість;
- захищеність;
- кросплатформеність.

Тестування змін – цей вид тестування тягне за собою перевірку змін у програмному забезпеченні після виявлення та усунення проблем, а також будь-яких змін у технологічному середовищі або, що більш банально, наказу користувача.

Якщо підвести підсумки, то:

Тестування програмного забезпечення визначається як діяльність, щоб перевірити, чи відповідають фактичні результати очікуваним результатам, і переконатися, що програмна система не містить дефектів.

Тестування важливо, оскільки помилки програмного забезпечення можуть бути дорогими або навіть небезпечними.

Важливими причинами використання тестування програмного забезпечення є: економічна ефективність, безпека, якість продукції та задоволеність споживачів.

Зазвичай тестування класифікується на три категорії: функціональне тестування, нефункціональне тестування або тестування продуктивності та тестування змін.

1.2 Огляд існуючих систем тестування

1.2.1 Огляд інструментарію Selenium

Зі збільшенням попиту на якісніше та надійніше програмне забезпечення, також збільшується попит на програми та інструментарії для тестування.

Selenium - це набір інструментів для автоматизації тестування програмного забезпечення з відкритим вихідним кодом, який став продуктом де-факто у світі забезпечення якості. Selenium підтримує кілька мов програмування, всі основні операційні системи та браузери, і наразі використовується у виробництві в таких компаніях, як Netflix, Google, HubSpot, Fitbit та інших. Весь набір надає широкий спектр рішень для різних проблем і потреб тестування.

Найпопулярнішим інструментарієм для тестування можна вважати Selenium, який має в своєму складі кілька продуктів. Серед них можна виділити Selenium WebDriver, Selenium RC, Selenium Server, Selenium Grid, Selenium IDE.

Кожен продукт має унікальне призначення в процесі тестування і може поєднуватися з іншими продуктами для тестування. [8]

Selenium WebDriver - це бібліотека програмного забезпечення для роботи та роботи з браузерами. Також часто використовується коротша назва WebDriver. Процес автоматизації тестування у WebDriver часто порівнюють з водінням таксі. У водінні таксі та автоматизації тестування є три учасники: замовник/інженер-тестувальник, автомобіль/браузер, водій таксі/WebDriver. Відповідно до цієї аналогії, тестувальник дає команду WebDriver взаємодіяти з елементами так само, як клієнт дає вказівки водієві таксі. Потім WebDriver дає браузеру (автомобілю) команди, які звучать приблизно так: Коли кнопка доступна для натискання, натисніть на неї. Потім браузер надає WebDriver інформацію про значення і статуси веб-елементів, які пізніше надсилаються скрипту.

Багато типів WebDriver розробляється для браузерів Firefox, Internet Explorer та Safari. Драйвер для браузера Google Chrome розробляється в рамках проекту Chromium (рисунок 1.1).

Selenium Server - це сервер, який дозволяє керувати браузером з віддаленої машини через мережу. По-перше, на машині, де повинен працювати браузер, слід встановити та запустити сервер. Потім на іншій машині запускається програма, яка за допомогою спеціального драйвера Remote WebDriver підключається до сервера і надсилає на нього команди. Потім ця машина запускає браузер

і виконує в ньому ці команди, використовуючи драйвер, відповідний цьому браузеру Selenium Server підтримує одночасно два набори команд - для нової версії (WebDriver) та для старої версії (Selenium RC). [7-9]

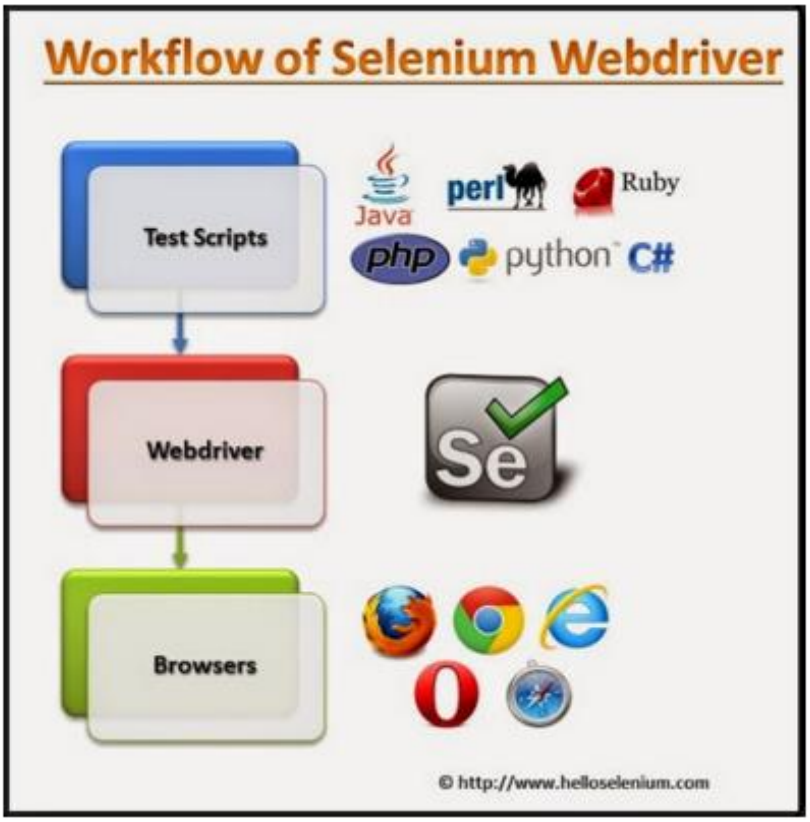

Рисунок 1.1 - Схема роботи Selenium Webdriver

Selenium Server - це сервер, який дозволяє керувати браузером з віддаленої машини через мережу. По-перше, на машині, де повинен працювати браузер, слід встановити та запустити сервер. Потім на іншій машині запускається програма, яка за допомогою спеціального драйвера Remote WebDriver підключається до сервера і надсилає на нього команди. Потім ця машина запускає браузер і виконує в ньому ці команди, використовуючи драйвер, відповідний цьому браузеру Selenium Server підтримує одночасно два набори команд - для нової версії (WebDriver) та для старої версії (Selenium RC). [7-9]

Selenium IDE - це доповнення до Firefox. Він простий у вивченні та ідеально підходить для створення прототипів. Для успішної роботи не потрібні навички програмування в Selenium IDE, оскільки вона в основному записує ваші

дії в браузері і повторює їх. Однак існують численні офіційні та неофіційні плагіни , які можуть покращити цей інструмент.

Selenium Grid дозволяє запускати паралельні тести на декількох машинах і браузерах одночасно. Основна функція цього інструменту - економія часу.

Якщо вам потрібно запустити 100 тестів, але ви використовуєте Selenium Grid для налаштування чотирьох віртуальних або фізичних машин, це займе приблизно чверть часу, який би ви витратили, якби запускали ці тести по одному на одній машині. Враховуючи, що скрипти зазвичай працюють повільно в браузері, використання методів підвищення продуктивності, таких як паралельне тестування, може допомогти вирішити цю проблему. Ви також можете використовувати його для паралельного тестування одного додатка в різних браузерах, коли на одній машині запущено Firefox, на іншій - Chrome і так далі. За допомогою Grid можна створювати різні конфігурації, комбінуючи різні версії браузерів і операційних систем. При використанні у великих виробничих середовищах Grid значно економить час.

До переваг Selenium можна віднести:

1. Безкоштовний: Selenium - не єдиний інструмент автоматизації тестування на ринку, але це єдиний безкоштовний інструмент, який може конкурувати з платними продуктами. Katalon Studio є єдиною життєздатною альтернативою, але він не надає такого великого вибору мов та не працює на Linux. Так як, Selenium з відкритим вихідним кодом, робить цей інструмент практичним та зручним для стартапів та незалежних розробників.

2. Підтримує мобільне тестування: З Selenium ви також можете тестувати нативні, гібридні або веб-мобільні додатки, хоча вам знадобиться додаткове програмне забезпечення. Є два основних варіанти - Appium та Selendroid. Вони обидва засновані на Selenium, тому розробники, які вже знайомі з ним, можуть застосовувати ті ж принципи при тестуванні мобільних додатків. Обидва інструменти мають відкритий вихідний код і велику підтримку спільноти. Основна відмінність полягає в тому, що Appium підтримує пристрої на iOS, Android і Windows, в той час як Selendroid орієнтований виключно на Android. Selendroid

також поставляється в комплекті з Appium, тому при тестуванні Android версій від 2.3 до 4.3 програма автоматично переключиться на Selendroid.

3. Широкий спектр підтримуваних мов, платформ і браузерів: Гнучкість, яку надає Selenium, майже не має аналогів у світі автоматизації тестування. По-перше, ми маємо десять підтримуваних мов, серед яких Java, Ruby, C#, PHP, JavaScript та Python - одні з найпоширеніших мов програмування на сьогоднішній день. Це також єдиний основний інструмент, який охоплює тестування Linux.

4. Величезна спільнота: Enlyft, дослідницька компанія, зазначає, що Selenium займає приголомшливу частку ринку в 27,48% серед усіх інструментів тестування програмного забезпечення, а його найближчий конкурент Apache Jmeter займає трохи більше 10%. Будучи одним з піонерів сучасного автоматизованого тестування, Selenium здобув велику спільноту розробників у таких великих компаніях, як Google, а також у стартапах. У списках вакансій для QA-професіоналів володіння Selenium є однією з обов'язкових навичок.

5. Велика бібліотека плагінів та розширень: Selenium можна розширити за межі його стандартної функціональності за допомогою широкого спектру плагінів. Деякі з них офіційно підтримуються Selenium (наприклад, Appium та Selendroid), але ви можете знайти більше непідтримуваних плагінів на Github. Також, існує велика кількість розширень для легкої інтеграції з вашим улюбленим програмним продуктом, таким як Jenkins або Eclipse. Ці плагіни існують не тільки для WebDriver, але також для Grid і Selenium IDE. Останні є особливо широкими, здатними посилити функціонально слабкий інструмент і зробити його готовим до використання у великих проектах.

До недоліків Selenium варто віднести:

1. Великий поріг входу: Одним із сучасних трендів автоматизованого тестування є тестування без коду. Цей підхід дозволяє виконувати тести будь-кому без глибоких знань програмування. Деякі інструменти тестування, такі як TestComplete, Ranorex або Tricentis, надають таку можливість, створюючи зручний користувацький інтерфейс поверх рівня коду, а також дозволяючи перемикатися між двома режимами. Однак Selenium не дозволяє проводити тестування

без коду. Вам потрібно добре володіти однією з мов програмування, що звужує коло людей, які можуть бути залучені до процесу тестування. Багато компаній, особливо стартапи, як правило, наймають своїх найкращих програмістів для написання функцій продукту, а менш кваліфікованих людей залучають до автоматизованого тестування. Це стосується лише Selenium WebDriver, оскільки використання набагато простішого Selenium IDE не вимагає жодних навичок кодування - але має певні обмеження.

2. Використовується лише для веб-додатків: Selenium - це інструмент, який не можна використовувати для автоматизації тестування десктопних додатків, оскільки він не може розпізнавати об'єкти в десктопних додатках. Він призначений лише для тестування веб-додатків з використанням різних браузерів. Отже, для тестування десктопних додатків вам доведеться знайти або окремий інструмент, такий як WinAppDriver, і використовувати його разом з Selenium (їх можна легко інтегрувати), або комплексний інструмент тестування, такий як Katalon, який може автоматизувати як веб-, так і не веб-тести.

3. Немає вбудованого порівняння зображень: У процесі контролю якості виникають труднощі з автоматизацією перевірки зображень. Найпростіший і найефективніший спосіб вирішити, чи правильно зображення відображається на екрані, - це перевірити його вручну. Хоча для більшості компаній нормально використовувати комбінацію ручного та автоматизованого тестування, це є великим джерелом помилок, оскільки тестування незабаром стає повторюваним, а тестувальники, як правило, перевіряють рідше та гірше той самий функціонал. Одне з поширених рішень цієї проблеми - порівняння зображень. У вас є макет вашого зображення, який програма повинна порівняти з тим, що відображається під час тесту. TestComplete, Ranorex, Katalon Studio та деякі інші вже мають таку функціональність. Щоб зробити це в Selenium, вам знадобиться стороннє програмне забезпечення. Sikuli є поширеним вибором.

4. Відсутність технічної підтримки: Пошук допомоги з Selenium є складною задачею. Поганою стороною цього є те, що саме спільнота надає допомогу, а не творці продукту. Знайти рішення нестандартної проблеми або допомогу з

нестандартними запитами можливо, але складно. У разі потреби ви можете звернутися до консалтингових або сервісних компаній за комерційною підтримкою. Selenium надає короткий список таких компаній на своїй сторінці екосистеми.

5. Відсутність можливостей звітності: Відсутність автоматично згенерованих звітів є однією з найбільших проблем Selenium. Щоб зафіксувати збої в тестуванні в Selenium, ви повинні зробити знімок екрану в момент збою. Це далеко не той формат, який потрібен команді для швидкої діагностики проблеми. Хоча різні постачальники пропонують функціонал для створення звітів з аналітикою на основі даних та інтеграцією інструментів для командної роботи, Selenium змушений покладатися на сторонні рішення.

У висновку, варто сказати, що Selenium не ідеальний, але має широкий інструментарій, щоб тестувальники доклали більше зусиль та ігнорували тенденцію до безкодового тестування.

1.2.2 Огляд інструменту для веб-тестування WAPT

Одною з імплементацій тестування є програмний модуль від WAPT. WAPT – це інструмент стрес-тестування, який дозволяє проводити комплексне та ефективне тестування веб-сайтів та інтернет-додатків із веб-інтерфейсом. WAPT дозволяє тестувати та аналізувати характеристики продуктивності веб-сайту та вузькі місця в різних сценаріях навантаження.

Це рішення застосовується для всіх типів веб-сайтів, мобільних додатків, корпоративних систем ERP та CRM, служб веб-API та платформ IoT. WAPT використовує простий підхід до тестового проектування, записуючи реальні веб-сесії з браузера або мобільного додатка. Він представляє кожен записаний сеанс як послідовність запитів HTTP.

Емуляція базується на появі декількох копій оригінального сеансу. Динамічні значення, вставлені в кожен новий сеанс, роблять його унікальним, а всю емуляцію реалістичною. [10] Більшість з цих вставок виконуються автоматично, тому вам потрібно лише вказати, як обробляти дані, що стосуються конкретних

програм. Сценарії не потрібні, але можуть бути використані для розгляду надзвичайно складних справ. Структура роботи WAPT наведено на рисунку 1.2.

Результати тестів представлені для огляду та аналізу у вигляді описових графіків та таблиць звітів. Вони показують, як продуктивність системи залежить від навантаження, що генерується в різні періоди тестування, а також існує можливість регулювати параметри виконання на льоту, щоб відокремити кожне можливе вузьке місце системи

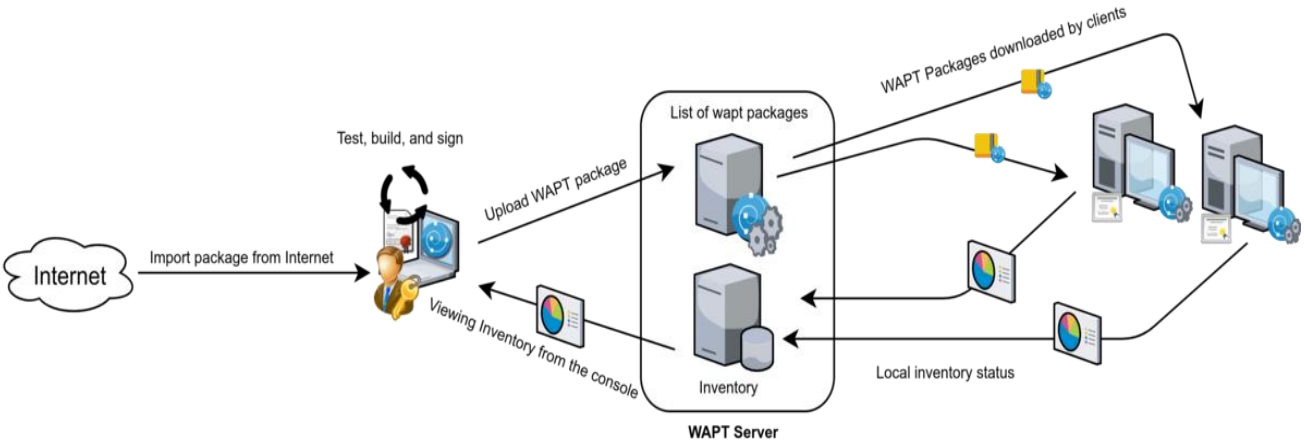

Рисунок 1.2 – Структура роботи WAPT

До переваг WAPT можна віднести:

1.WAPT дозволяє імітувати тисячі віртуальних користувачів на вашому веб-сайті, а також його здатність тестувати різні взаємодії з користувачами.

2.WAPT імітує реальні взаємодії користувачів і намагається змусити своїх віртуальних користувачів виконувати дії, близькі до реальних.

3. WAPT дозволяє створювати докладні звіти, які можна переглядати як у форматі HTML, так і в форматі Excel.

4. Можливість швидко генерувати навантаження, без необхідності довго чекати, поки тисячі ваших віртуальних користувачів будуть готові.

5.WAPT забезпечує автоматичну обробку стандартних сеансових даних і ряд функцій для роботи зі специфічними значеннями. Він підтримує всі типи проксі-серверів: HTTP(S), SOCKS4(5) тощо.

6. Підтримка SSL дозволяє проводити навантажувальне тестування веб-сайтів.

7. Розширені звіти про помилки, які підтримуються інтегрованим переглядачем журналів.

8. Розподілена генерація навантаження, яка також підтримує хмарне тестування і віддалений контроль тестування.

До недоліків WAPT варто віднести:

1. Хоча ви можете тестувати будь-який веб-сайт під управлінням будь-якої ОС і технології, ви не можете встановити WAPT в будь-яку іншу ОС, крім Windows. У сучасному середовищі, де більшість веб-сайтів і веб-додатків працюють на базі Linux, це є великою проблемою.

2. WAPT не дозволяє писати скрипти, а використовує робочий процес Записати > Налаштувати > Перевірити > Виконати > Проаналізувати.

3. WAPT не є інструментом для тестування навантаження, який можна назвати недорогим. Навіть звичайна версія WAPT коштує $700, а просунута версія WAPT Pro - $1200. Однак, якщо ви займаєтеся серйозним тестуванням навантаження, стресу і продуктивності, то ви можете знайти цю ціну виправданою за широкі і унікальні можливості, які надає цей інструмент.

В цілому, WAPT є відносно доступним, простим у використанні інструментом навантажувального, стресового і продуктивного тестування, який може допомогти знайти недоліки в коді розробника, але велика ціна обмежує його у використанні для стартапів та невеликих компаній.

1.2.3 Огляд інструменту тестування Postman

POSTMAN — це клієнт API для написання, тестування, спільного використання та документування API. Він використовується для бекенд-тестування, А саме коли запит надсилається на сервер, і він в свою чергу надсилає відповідь на надісланий запит.

POSTMAN широко використовується розробниками та інженерами з автоматизації для перевірки того, що служба запущена та працює, а також для керування та публікації документації API. Зручний інтерфейс Postman (рисунок 1.3) дає можливість швидко налаштувати запити до серверу та створювати власні колекції.

API (Application Programming Interface) – це спосіб обміну даними з сервера для двох програм або програмних компонентів. Postman допомагає тестувальникам у створенні фіктивних серверів і проектуванні API. Ви можете перевірити, як API реєструє нового користувача програми, а також додавати та видаляти дані про нього на сервері за допомогою Postman.

До основних етапів роботи з Postman відносять:

- створення запиту;
- налаштування параметрів запиту;
- надсилання запиту з встановленими параметрами;
- зберігання запиту для подальшої роботи.

При роботі з API використовується архітектура REST. Для взаємодії по HTTP використовуються 4 наступних метода:

- POST - цей метод створює об'єкт і надсилає дані на сервер.
- GET - отримання даних із сервера;
- DELETE - видалення об'єкта;
- PUT - оновлення об'єкта.

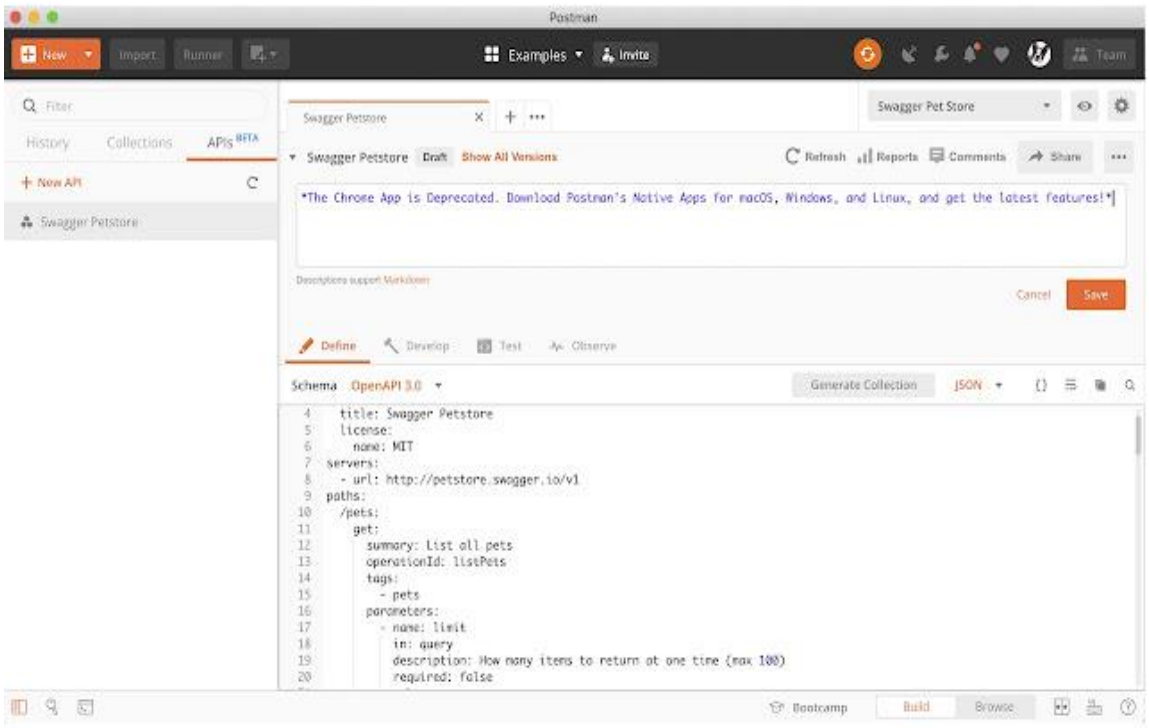

Рисунок 1.3 – Інтерфейс Postman

Postman може працювати на різних операційних системах від Windows, Mac OS та Linux. Це дає розробникам більше простору для розробки API. Для функціональних тестувальників Postman дозволяє тестувальникам створювати тести для викликів API без особливих зусиль, витрачених на написання скриптів.

До переваг Postman можна віднести:

1. Зручність у використанні. Завдяки простому інтерфейсу тестувальники можуть швидко створювати набори тестів, заповнюючи шаблони. Postman також надає фрагменти коду, які підтримують створення скриптів з прикладами перевірок часу відгуку, коду відповіді тощо.

2. Доступність. Користувачі Postman можуть легко отримати доступ до своїх файлів, увійшовши до свого облікового запису на пристрої з встановленим додатком Postman або розширенням для браузера Postman.

3. Різноманітні функціональні можливості. Postman підтримує всі можливі методи HTTP, збереження прогресу, конвертацію з API в код, зміну середовища розробки API та багато іншого.

4. Можливості відстеження запитів. Для HTTP-відповідей у Postman підтримується кілька кодів статусу, за якими користувачі можуть перевірити відповідь. Це, зокрема, успішні запити, порожня відповідь, поганий запит і несанкціонований доступ.

Ці корисні функції спрощують розробникам весь процес створення та обміну API для розробників. Коли справа доходить до тестування API, особливо тестування автоматизації, Postman все ж має значні обмеження.

До недоліків Postman варто віднести:

1. Обмежена область тестування. Хоча Postman ідеально підходить для тестування RESTful API, він не дуже добре розроблений для SOAP API та інших API.

2. Низька можливість повторного використання скриптів. Користувачі Postman не можуть повторно використовувати свої попередньо написані скрипти або додавати нові запити. Це означає, що тестувальникам доводиться створювати нові тестові скрипти знову і знову для кожного проекту.

3. Обмежена інтеграція. Хоча API уможливлюють Agile-процес, сам інструмент не підтримує багато можливостей інтеграції. Це стає перешкодою для підключення Postman до існуючих систем та співпраці всередині команди.

Зважаючи на ці недоліки, Postman часто рекомендується для ручного тестування API, особливо для REST API. При роботі з розширеними і поглибленими тестами API, цей інструмент автоматизації тестування є дуже корисним. Також однією з переваг є добре структуровані колекції тестів і фреймворків. Це економить час і зусилля користувачів, коли справа доходить до пошуку та організації тест кейсів.

Для вирішення поставленої задачі даний продукт підходить нам лише частково, так як основний функціонал полягає у надсиланні запитів до нашого додатку у зручному форматі. У подальшому, Postman буде використовуватись як емулятор самого додатку для надсилання подій.

1.2.4 Огляд інструменту тестування Ranorex

Ranorex Studio - це зручне рішення, яке дозволяє користувачеві створювати, запускати та підтримувати автоматизовані тести. Існує безліч онлайн-уроків для тих, хто ніколи не користувався цим програмним забезпеченням і шукає інформацію про те, як ним користуватися. Однак, навчальні посібники Ranorex можуть зайняти багато часу, оскільки вони містять години відео, а для будь-якого інструменту або програми завжди існує період навчання.

Зручний інтерфейс Ranorex (рисунок 1.4) дає можливість швидко налаштувати автоматизовані тести, створювати власні колекції.

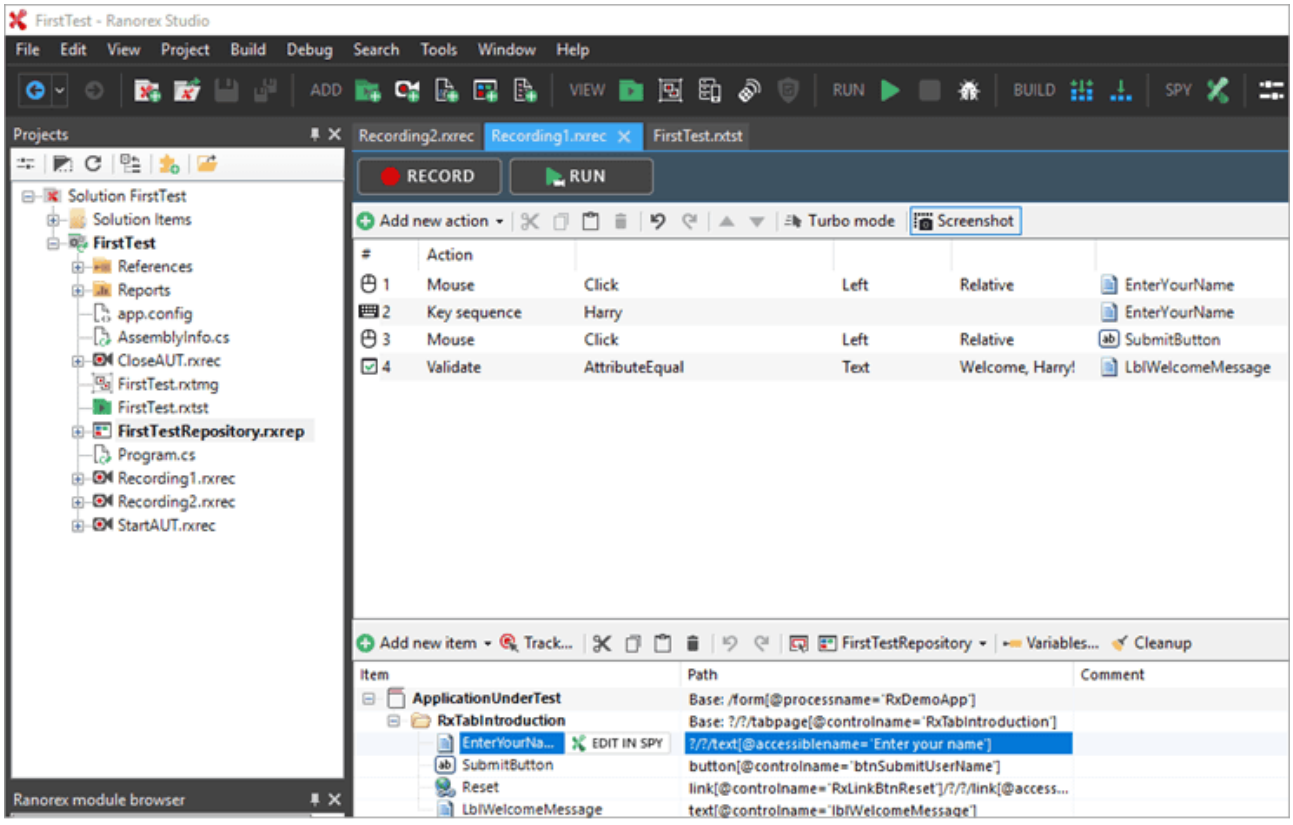

Рисунок 1.4 – Інтерфейс Ranorex

Ranorex Studio має три основні функції, а саме:

1. Ranorex Recorder Ця функція Ranorex Studio робить створення автоматизованих тестів відносно простим. Ranorex Recorder дозволяє користувачеві створювати, записувати, відтворювати та редагувати автоматизовані тести за допомогою репозиторію об'єктів Ranorex. Ця функція використовує підхід з низьким рівнем коду і не вимагає ніяких попередніх знань з програмування, оскільки надає багато розумних дій, таких як "Встановити значення", "Закрити програму" і багато іншого. Цей інструмент автоматизації Ranorex робить запуск повторюваних тестів набагато простішим та ефективнішим.

2. Ranorex Spy. Ключовою частиною будь-якого інструменту автоматизації тестування є здатність програмного забезпечення ідентифікувати елементи інтерфейсу користувача. Ranorex Spy - це інструмент розпізнавання об'єктів, який може миттєво ідентифікувати та відстежувати елементи інтерфейсу користувача, представлені в ієрархічній структурі. Об'єкти можуть бути ідентифіковані за xpath, тегами автоматизації, мітками тощо. Ці елементи інтерфейсу зберігатимуться в репозиторії Ranorex, що дозволить повторно використовувати їх для декількох тестових кейсів за потреби. Повторне використання компонентів має вирішальне значення для створення надійних тестових кейсів.

3. Репозиторій об'єктів Ranorex. Репозиторій об'єктів - це централізована колекція об'єктів і властивостей програми, що тестується. Ця функція програмного забезпечення Ranorex дозволяє тестувальнику створити логічне відображення елементів інтерфейсу користувача, включаючи кнопки, вкладки, випадаючі меню, тестові поля тощо. Ця функція має деревоподібну структуру, розділену на дві колонки: Елемент і Шлях. Колонка "Елемент" містить елементи інтерфейсу користувача, а "Шлях" містить логічне відображення кожного елемента. Ця функція працює з Ranorex Recorder і Spy, щоб дозволити користувачеві легко знаходити і підтримувати об'єкти, що використовуються в тест кейсах.

Ці ключові функції роблять Ranorex Studio доступним для багатьох тестувальників. Крім того, Ranorex використовує C# як основну мову і має всі можли-

вості будь-якого .Net додатку. Отже, якщо потрібні модулі коду, наприклад, підключення до екземпляру бази даних SQL і перевірка результатів, то Ranorex має всю функціональність .Net. для цього.

Використання інструменту автоматизації тестування, такого як Ranorex, має багато переваг, а також кілька недоліків. Переваги варіюються від підвищення загальної продуктивності до специфічних переваг Ranorex. Доступність автоматизації Ranorex є однією з її основних переваг, але використання інструменту автоматизації також економить час і кошти, знижує ризик помилок і покращує якість тестування.

Використання інструменту автоматизації економить час і гроші, оскільки багато тестів програмного забезпечення повторюються і потребують модифікації та застосування до декількох систем. Виконання тестів вручну може зайняти багато часу. Автоматизоване тестування може скоротити цей час до кількох годин. Це також дає користувачам більше часу, щоб зосередитися на основних завданнях, підвищуючи продуктивність.

Автоматизація тестування також значно знижує ризик людської помилки, який виникає при ручному тестуванні. Помилки неминучі навіть для найдосвідченіших і старанних тестувальників. Повторювана природа тестування програмного забезпечення є важкою для ручних тестувальників; автоматизація цих тестів може підвищити продуктивність. Ranorex також створює настроювані звіти для всіх тестів, які можна конвертувати в формати, якими легко ділитися.

Деякі проекти настільки великі та інтенсивні, що навіть у найбільших компаніях не вистачає людей для тестування програмного забезпечення. Автоматизоване тестування покращує якість тестів і проектів, збільшуючи кількість, швидкість і точність тестів. Перевагою Ranorex Studio є можливість використання на різних платформах, тести можна використовувати повторно, і вам не доведеться починати з нуля з кожним новим проектом. Автоматизація тестування має важливе значення для збільшення тестового покриття в міру зростання продукту або бізнесу. Ranorex можна використовувати на мобільних пристроях iOS і Android,

а також в різних браузерах, включаючи Internet Explorer, Google Chrome, Safari і Firefox.

Будь-яке програмне забезпечення для автоматизації має недоліки, і автоматизація Ranorex не є винятком. Однак, ці недоліки не повинні бути перешкодою.

Ranorex не є програмним забезпеченням з відкритим вихідним кодом; ви повинні заплатити за ліцензію, щоб використовувати його. Крім того, якщо ви хочете запустити кілька тестів на декількох пристроях, вам знадобиться ліцензія на час виконання на цьому пристрої, але це лише частина вартості повної студійної ліцензії. Проте, Ranorex має 30-денну безкоштовну пробну версію, тому якщо ви не впевнені, чи підходить цей інструмент для ваших проектів, ви можете спробувати його спочатку.

Користувачі отримають значну користь від розуміння C# або Java та загального ООП. Досвід використання інструментів контролю вихідного коду (GIT) та безперервної інтеграції (CI), Jenkins/Bamboo тощо дозволить проекту повною мірою використати можливості автоматизації тестування та інтегруватися з процесом розробки DevOps.

1.3 Аналіз об’єкту тестування

При тестуванні усіх мобільних додатків одним з ключових етапів є тестування даних, які надсилаються. Після кожної взаємодії з додатком він надсилає певні події, кожен з яких складається з певних параметрів. Ключовими параметрами є назва та час надсилання події. До самої інформації в події можна віднести розмір надісланої інформації, дані про назву елементу з яким була взаємодія, результат взаємодії, код та опис помилки при необхідності, тощо.

Валідація подій є важливим етапом для розробки будь-якого програмного забезпечення, адже надає можливість розробникам та тестувальникам мати точні дані про поведінку розробленого додатку. У подальшому уся потрібна інформація подій надсилається до аналітики для зручного переглядання та формування

статистики стабільності додатку, перегляд трендів у реальному часі, перегляду трафіку даних у системі, тощо.

Для досягнення цих цілей потрібно аби додаток надсилав актуальну інформацію яка відповідає дійсності на момент відправки події. Для таких цілей потрібно проводити валідацію кожного події на його правильність. В кожній системі є певний перелік можливих значень для кожного поля подій. Таким чином, тестувальник може провести валідацію події у наступні кроки:

6. Обирає потрібне програмне забезпечення для валідації подій.
7. Відкриває додаток для якого потрібно провести валідацію.
8. Виконує попередньо складений тест-кейс, який у результаті має відправити подію для валідації.
9. Отримана подія та її параметри перевіряється згідно з документацією на структуру, можливі значення, тощо.

5. Аналіз даних. При позитивних результатах продовжує валідацію подій. При негативних результатах, тестувальник описує перелік проблем з надісланою подією.

Можна прийти до висновку, що створення програмного забезпечення для валідації подій є актуальним, адже це прискорить та спростить роботу тестувальникам. Мануальна перевірка можливих значень забирає багато часу. Саме тому, використання шаблонів для перевірки подій обмежить можливі людські помилки та зменшить час перевірки додатку. В свою чергу це дасть тестувальникам більше часу для інших видів тестування.

1.4 Висновки до розділу

У даному розділі були описані та оглянуті існуючі системи для тестування програмного забезпечення. Серед альтернатив та існуючих систем даного програмного забезпечення були оглянуті інструментарій від Selenium, інструмент для веб-тестування WAPT та інструмент для тестування API Postman.

Найпопулярнішим інструментарієм для тестування можна вважати Selenium, який має в своєму складі кілька продуктів. Серед них можна виділити Selenium WebDriver, Selenium RC, Selenium Server, Selenium Grid, Selenium IDE. Кожен продукт має унікальне призначення в процесі тестування і може поєднуватися з іншими продуктами для тестування.

Одною з імплементацій тестування є програмний модуль від WAPT. Це рішення застосовується для всіх типів веб-сайтів, мобільних додатків, корпоративних систем ERP та CRM, служб веб-API та платформ IoT. WAPT використовує простий підхід до тестового проектування, записуючи реальні веб-сесії з браузера або мобільного додатка. Він представляє кожен записаний сеанс як послідовність запитів HTTP.

Для тестування API за допомогою запитів часто використовують Postman. Даний продукт дозволяє створювати запити для надсилання на сервер з використанням основних методів запитів до серверів по HTTP. У зручному інтерфейсі за допомогою колекцій можна створити потрібний запит, наприклад з події для валідації. Вказавши відповідні параметри, запит відправиться до серверу для подальшої обробки. На практиці, тестувальники створюють такі запити для перевірки логіки програми

Однак, жодний з розглянутих програмних продуктів не підходить для вирішення поставлених задач у відповідному об’ємі. Функціонал оглянутих методів є лише частково корисним для вирішення поставленої задачі. Саме тому необхідним є розробка універсальної системи, що дозволить забезпечити швидку та просту валідацію надісланих подій та їх обробку.

2 ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Огляд та вибір мови програмування

2.1.1 Мова програмування Java

Java була розроблена як мова програмування для побутової електроніки, і вона все ще може знайти своє призначення в цій сфері, оскільки популярність вбудованої розробки та Інтернету речей (IoT) продовжує зростати. Потужний набір функцій неминуче зробили її привабливою для практично всіх типів програмного забезпечення, які тільки можна собі уявити.

Сама мова вважається мовою загального призначення з об'єктно-орієнтованими функціями і часто помилково вважається справжньою мовою об'єктно-орієнтованого програмування (ООП). Хоча вона, безумовно, може імітувати функціональність ООП, Java має примітивні та не примітивні типи даних, що робить її не зовсім ООП, оскільки все в об'єктно-орієнтованих мовах вважається об'єктами.

Тим не менш, розробники Java можуть використовувати такі концепції ООП, як поліморфізм, класи, об'єкти, успадкування та інкапсуляція, що робить код, який ви створюєте, більш читабельним, зручним для супроводу та повторного використання, не кажучи вже про меншу схильність до помилок.

Java - це не лише мова, але й екосистема інструментів, що охоплює майже все, що може знадобитися для розробки на Java. Це включає в себе:

- Java Development Kit (JDK) - за допомогою цього інструментарію та стандартного додатку Notebook ви можете писати та запускати/компілювати код на Java
- Java Runtime Environment (JRE) - інструмент розповсюдження програмного забезпечення, що містить автономну віртуальну машину Java, стандартну бібліотеку Java (Java Class Library) та інструмент конфігурації

- Інтегроване середовище розробки (IDE) - інструменти, які допомагають запускати, редагувати та компілювати код. Найпопулярнішими є IntelliJ IDEA, Eclipse та NetBeans.

До переваг Java можна віднести:

1. Дозволяє багаторазово використовувати код - Java є мовою програмування загального призначення, яка дозволяє програмістам написати програму один раз і запустити її на тій самій платформі або деінде, таким чином дозволяючи їм створювати код для багаторазового використання.

2. WORA (Write once run anywhere) - Java не покладається на одну конкретну платформу для запуску коду. "Байт-код" або скомпільований код Java не залежить від платформи, тобто код можна написати один раз і запустити будь-де, незалежно від операційної системи, за умови, що вона підтримує віртуальну машину Java.

3. Широка екосистема бібліотек - Java має велику екосистему бібліотек, фреймворків та інструментів, які задовольняють різні сфери застосування, такі як веб-розробка, розробка мобільних додатків тощо.

4. Java зменшує ризики безпеки на стороні клієнта - мінімізуючи використання явних вказівників, Java зменшує проблеми та небезпеки безпеки, оскільки вказівник зберігає адресу пам'яті іншого значення, яке може бути використане для отримання небажаного або несанкціонованого доступу. Цю проблему можна вирішити, відмовившись від ідеї вказівників. Крім того, кожна програма на Java має менеджер безпеки, який дозволяє створювати обмеження доступу до класів, таким чином усуваючи ризики.

5. Автоматична збірка сміття - система автоматичного управління пам'яттю Java (Garbage Collection) ефективно справляється з розподілом пам'яті, видаляючи об'єкти, що не використовуються, без втручання користувача.

6. Багатопоточність - Потік - це найменша одиниця програми. Багатопоточність означає, що в Java-програмах можуть виконуватися декілька потоків одночасно. Ці потоки поділяють один і той самий простір пам'яті і виконуються незалежно один від одного. Це дозволяє ефективно використовувати процесор.

7. Багатоцільовість - додатки на Java включають широкий спектр мобільних додатків, розподілених додатків, додатків на основі IoT, ігрових та веб-додатків. Огляд інтерфейсу програмування додатків на основі Java дуже відрізняється від інтерфейсу, створеного за допомогою нативних додатків.

8. Стабільність - будь-яка нова версія Java може бути швидко і легко випущена і використана. Це величезна перевага, яка робить цю мову стабільною та надійною.

9. Активна спільнота та екосистема - Java має велику та сильну спільноту розробників, тому ви можете знайти багато ресурсів, бібліотек та фреймворків, які допоможуть вам у ваших проектах розробки. Вона також має потужну підтримку від Oracle.

До недолів Jaava варто віднести:

1. Низька продуктивність - це повільна і трудомістка мова порівняно з C/C++, оскільки код щоразу інтерпретується в машинний код. Це додає додатковий крок при виконанні коду в JVM. Збирач сміття також збільшує споживання часу та простору пам'яті, оскільки він використовує більше процесорного часу.

2. Відсутність засобів резервного копіювання - Java відома чудовим зберіганням коду та даних, але розробники Java не створюють резервних копій для подальшого використання. Тому багато програмістів утримуються від використання цієї мови в довгостроковій перспективі.

3. Графічний інтерфейс - для створення привабливого графічного інтерфейсу в Java є популярні білдери та фреймворки, такі як Swing та JavaFX, але вони не дуже корисні при створенні складних та багатих інтерфейсів. Тому програмування на мові, яка не забезпечує такий привабливий інтерфейс, як інші, може стати монотонним і негідним.

4. Складні коди - щоб зробити програму керованою, в синтаксисі Java є багато довгих і складних речень і складних слів, які роблять код важким для читання і запам'ятовування. Іншими словами, код Java може бути багатослівним, вимагаючи більше рядків коду для виконання завдань у порівнянні з деякими іншими мовами програмування.

5. Займає багато місця в пам'яті - хоча збір сміття є перевагою в програмуванні на Java, його виконання негативно впливає на ефективність використання пам'яті та продуктивність процесора. Коротше кажучи, Java вимагає величезного обсягу пам'яті та сховища порівняно з C/C++, що варто врахувати перед тим, як змінювати нашу мовну базу.

6. Обмежений низькорівневий доступ - сильні функції безпеки Java обмежують низькорівневий доступ до системних ресурсів та обладнання, що може бути недоліком для певних типів додатків.

Підсумовуючи, можна сказати, що Java пропонує багаторазове використання коду, незалежність від платформи (WORA), широкий спектр бібліотек та надійну систему безпеки.

Однак, Java має кілька недоліків, серед яких відносно низька продуктивність через інтерпретацію віртуальною машиною Java (JVM) та накладні витрати на збір сміття. Їй бракує надійного механізму резервного копіювання, що ускладнює збереження даних. Крім того, Java може бути не найкращим вибором для створення візуально привабливих користувацьких інтерфейсів, оскільки вона може створювати довгий і складний код. Високе споживання пам'яті також може вплинути на ефективність, а також обмежує низькорівневий доступ до системи.

2.1.2 Мова програмування Python

Python - це популярна динамічна мова програмування високого рівня загального призначення, яка присутня на ринку вже майже 30 років. Сьогодні його легко знайти практично у будь-якому місці: веб- і настільні програми, машинне навчання, мережеві сервери та багато іншого. Він використовується для невеликих проектів, а також таких компаній, як Google, Facebook, Microsoft, Netflix, Dropbox, Mozilla або NASA. Python - це найбільш швидкозростаюча мова програмування відповідно до StackOverflow Trends. Indeed.com, всесвітня пошукова система пошуку робочих місць, що займається працевлаштуванням, віднесла

Python до третьої найбільш прибуткової мови програмування у світі. Це означає, що дедалі більше програмістів вивчають цю мову та використовують її. [11]

До переваг Python можна віднести:

1. Python - це мова програмування, яка має англо-подібний синтаксис. Це полегшує читання та розуміння коду. Python дійсно легко підібрати та вивчити, тому багато людей рекомендують Python новачкам. [12] Вам потрібно менше рядків коду для виконання того самого завдання, порівняно з іншими основними мовами, такими як C / C ++ та Java.

2. Підвищена продуктивність. Завдяки простоті Python розробники можуть зосередитись на вирішенні проблеми. Їм не потрібно витрачати занадто багато часу на розуміння синтаксису чи поведінки мови програмування.

3. Python - це інтерпретована мова, що означає, що Python безпосередньо виконує код, рядок за рядком. У разі будь-якої помилки він зупиняє подальше виконання та повідомляє про помилку, що сталася. Python показує лише одну помилку, навіть якщо програма має кілька помилок. [13] Це полегшує налагодження.

4. Python не знає тип змінної, поки ми не запустимо код. Він автоматично призначає тип даних під час виконання. Програмісту не потрібно турбуватися про оголошення змінних та їх типів даних.

5. Безкоштовно та з відкритим кодом. Python підпадає під ліцензію з відкритим кодом, затверджену OSI. Це дозволяє використовувати та розповсюджувати безкоштовно. Ви можете завантажити вихідний код, змінити його і навіть поширити свою версію Python. [14] Це корисно для організацій, які хочуть змінити певну поведінку та використовувати свою версію для розробки.

6. Підтримка величезних бібліотек. Стандартна бібліотека Python величезна, ви можете знайти майже всі функції, необхідні для вашого завдання. Отже, вам не потрібно залежати від зовнішніх бібліотек. Але навіть якщо ви це зробите, менеджер пакунків Python (pip) полегшує імпорт інших чудових пакетів з індексу пакетів Python (PyPi). Він складається з понад 200 000 упаковок.

7. Кроссплатформеність. У багатьох мовах, таких як C / C ++, вам потрібно змінити код, щоб запускати програму на різних платформах. Це не те саме з Python. Ви пишете лише один раз і запускаєте його де завгодно. Однак слід бути обережним, щоб не включати будь-які системно залежні функції.

До недоліків Python варто віднести:

1. Повільніша за скомпільовані мови. Одним з головних недоліків Python є те, що вона працює повільніше, ніж скомпільовані мови, такі як C++ або Java. Це пов'язано з тим, що Python є інтерпретованою мовою, а це означає, що кожен рядок коду виконується інтерпретатором по одному разу. На противагу цьому, скомпільовані мови перетворюються в машинний код перед виконанням, що робить їх швидшими. Ця різниця у швидкості може бути особливо помітною при роботі з великими масивами даних або при виконанні складних обчислень. У цих випадках Python може бути не найкращим вибором для критично важливих до продуктивності додатків. Однак варто зазначити, що існують способи оптимізувати код Python і підвищити його продуктивність, наприклад, використовувати NumPy для числових операцій або Cython для компіляції Python-коду в C.Незважаючи на обмеження продуктивності, Python залишається популярною мовою для створення прототипів та експериментів завдяки простоті використання та великій бібліотеці модулів. Розробникам, яким потрібно оптимізувати свій код для критично важливих до продуктивності додатків, можливо, доведеться розглянути інші мови або інструменти, але для багатьох додатків сильні сторони Python переважають його слабкі сторони.

2. Менш безпечна. З точки зору безпеки, Python вважається менш захищеною, ніж деякі інші мови програмування, такі як Java або C++. Це пов'язано з тим, що Python є динамічно типізованою мовою, що означає, що типи даних визначаються під час виконання, а не під час компіляції. Це може призвести до вразливостей, включаючи переповнення буфера або атаки типу "ін'єкція". Однак варто зазначити, що Python має вбудовані засоби безпеки, такі як стандартний модуль "os" бібліотеки, який забезпечує безпечний доступ до файлів і каталогів. Крім того, для Python доступні сторонні інструменти та бібліотеки, які можуть

допомогти покращити безпеку, наприклад, бібліотека PyCryptodome для шифрування та хешування.

3. Не ідеальне робоче середовище. Одним з потенційних недоліків використання Python в робочому середовищі є те, що вона може не найкраще підходити для всіх типів проектів або команд. Наприклад, якщо проект вимагає високої продуктивності або низькорівневого доступу до системи, кращим вибором може бути мова на кшталт C++.Крім того, динамічна природа Python і відсутність строгої типізації можуть ускладнити підтримку і налагодження коду в міру того, як проекти стають більшими і складнішими. Це може призвести до збільшення часу та вартості розробки, а також до потенційних помилок або вразливостей безпеки.

4. Динамічний набір тексту. Динамічна типізація є ще одним потенційним недоліком використання Python у робочому середовищі. Python дозволяє змінювати тип даних змінної під час виконання програми без необхідності явного оголошення типів. Хоча це може зробити код більш гнучким і легшим для написання, це також може призвести до помилок і неочікуваної поведінки. Наприклад, якщо ви присвоїте змінній рядкове значення, а потім спробуєте виконати над нею математичну операцію, Python видасть помилку типу TypeError. Це може розчарувати розробників, які звикли до більш суворої перевірки типів в інших мовах.

5. Погана багатопоточність. Хоча Python має модуль багатопоточності, він не є по-справжньому багатопотоковим через глобальне блокування інтерпретатора (GIL). GIL гарантує, що лише один потік може виконувати байт-код Python одночасно, навіть на багатоядерних системах. Це означає, що хоча у Python можна створити декілька потоків, вони не можуть працювати паралельно, що обмежує потенційний приріст продуктивності від використання багатопоточності.

Python - популярна мова програмування, яка має багато переваг: простота використання, читабельність та велика спільнота розробників. Однак вона також має деякі обмеження, такі як нижча продуктивність порівняно з компільованими мовами, проблеми з керуванням пам'яттю, динамічною типізацією та сумісністю з версіями.

Загалом, Python є універсальною мовою, яку можна використовувати для широкого спектру застосувань, від веб-розробки до машинного навчання та аналізу даних. Підвівши підсумки, дана мова програмування найкраще підходить для розробки даного програмного забезпечення [15, 19], адже швидкість та простота розробки цієї програми надає можливість у майбутньому пришвидшені та покращенні даної програми.

2.2 Огляд та вибір веб-фреймворку

2.2.1 Веб-фреймворк Django

Django - один з популярних фреймворків, що використовується в мові програмування Python для створення веб-додатків. Представлений у 2005 році, Django використовує ліцензію з відкритим вихідним кодом, що означає, що його вихідний код можна вільно редагувати та змінювати. Загалом, він має на меті зменшити складнощі, з якими стикаються при розробці веб-додатків, за допомогою спрощеного підходу.

Наразі на Django покладаються такі гіганти соціальних мереж та пошукових систем, як Google, Instagram, YouTube. Цей фреймворк побудований з використанням архітектури Model-View-Template (MVT). Через цю архітектуру новачку розробнику може бути складно створювати веб-додаток з нуля.

До переваг Django можна віднести:

1. Обчислювальні можливості. Django є популярним рішенням для різних додатків інтернету речей (IoT). Йому надають перевагу розробники, які працюють над алгоритмами машинного навчання (ML), штучного інтелекту (AI) та IoT. Обчислювальні можливості роблять цей фреймворк ідеальною платформою для додатків машинного навчання. Він також сумісний з деякими потужними бібліотеками машинного навчання, такими як PyTorch, NumPy тощо. Оскільки Django надає можливості регресії та прогнозування, він ідеально підходить для розробки інтелектуальних додатків, будь то IoT, AI або ML.

2. Сумісність з Python. Django базується на мові Python, що дає можливість використовувати деякі з ключових переваг Python. Він успадкував всі переваги Python, такі як простота, підвищення продуктивності та підтримка зовнішніх бібліотек. Все це прискорює процес розробки. Django полегшує програмістам розробку додатків з чистим, читабельним і зручним для підтримки кодом, використовуючи синтаксичні правила Python. Це також допомагає розробникам легко скоротити час розробки, створюючи кастомні веб-додатки без необхідності писати додатковий код.

3. Універсальність та масштабованість. Веб-фреймворк Django може допомогти навіть найбільш завантаженим сайтам задовольнити вимоги високого трафіку, а отже, покращити масштабованість. Про універсальність Django свідчить той факт, що він використовується в різних галузях і додатках, включаючи управління контентом, наукові обчислення, інтелектуальну автоматизацію і навіть управління підприємством.

4. Спільнота Django. Однією з найкращих рис Django є його активна спільнота, яка продовжує активно працювати над покращенням фреймворку, щоб зробити його більш зручним та стабільним. Розробники можуть легко імпортувати специфічні функціональні можливості, розроблені експертами-професіоналами.

До недоліків Django варто віднести:

1. Відсутність чітких стандартів. Основною причиною, чому багато програмістів не люблять Django, є те, що йому не вистачає стандіртів. На відміну від інших фреймворків, таких як Ruby on Rails, в Django все повинно бути чітко визначено. Це призводить до шаблонності конфігурації і, відповідно, до уповільнення процесу розробки.

2. Монолітність. Монолітність Django може бути перевагою для деяких програмістів, але для більшості це є недоліком. Django має певний набір файлів та попередньо визначених змінних, які програмістам потрібно вивчити перед створенням будь-якого проекту.

3. Не підходить для невеликих проектів. Веб-фреймворк Django є проектом з великою кількістю коду, який забирає багато часу на обробку сервером. Це

може негативно вплинути на низькопродуктивні веб-сайти, які працюють на дуже низькій пропускній здатності. Саме тому він не підходить для проектів та продуктів з невеликою кількістю функцій та вимог.

4. Неможливість одночасної обробки декількох запитів. На відміну від багатьох сучасних веб-фреймворків, Django не може дозволити окремим процесам обробляти декілька запитів одночасно. Це змушує розробників придумувати способи, як змусити окремі процеси ефективно обробляти декілька запитів.

5. Складність у вивченні. Одним з головних недоліків Django є те, що він має великий поріг входу. В своїй структурі, він має велику кількість функцій, які тяжко зрозуміти розробникам. Це може бути дуже складно для розробників, які переходять з інших мов. Інший синтаксис також ускладнює швидке освоєння мови.

Підсумувавши все вище згадане, Django поступово стає популярним вибором серед розробників та великих організацій. Надійний фреймворк дозволяє створювати масштабні корпоративні додатки. Але, незважаючи на всі переваги, він не буде найкращим вибором для невеликих проектів. Саме тому, слід розглянути альтернативу веб-фрейворку Django, а саме Flask, який по своїй суті, є мікрофреймворком.

2.2.2 Веб-фреймворк Flask

Веб-фреймворк - це архітектура, що містить інструменти, бібліотеки та функціональні можливості, придатні для побудови та підтримки масових веб-проектів за допомогою швидкого та ефективного підходу. Вони призначені для впорядкування програм та сприяння повторному використанню коду. [15]

Flask - це веб-фреймворк. Це означає, що Flask надає вам інструменти, бібліотеки та технології, що дозволяють створювати веб-додаток. [15] Цей веб-додаток може бути декількома веб-сторінками, щоденником, вікі-програмою чи використовуватися як веб-календар чи комерційний веб-сайт.

Flask є частиною категорій мікрофреймворку. Мікрофреймворк - це, як правило, фреймворк, який майже не залежить від зовнішніх бібліотек. [16] У цьому є плюси і мінуси. Плюси полягають у тому, що фреймворк є легким, існує невелика залежність від оновлення та стеження за помилками безпеки.

До переваг Flask можна віднести:

1. Масштабованість. Це означає, що ви можете використовувати його для неймовірно швидкого зростання технічного проекту, такого як веб-додаток. Якщо ви хочете створити додаток, який починається з малого, але має потенціал для швидкого зростання в напрямках, які ви ще не до кінця опрацювали, то це ідеальний вибір. Простота використання та невелика кількість залежностей дозволяють йому працювати безперебійно, навіть коли він масштабується і розширюється.

2. Гнучкість. Це основна особливість Flask і одна з його найбільших переваг. Це не тільки корисно з точки зору того, що ваш проект може легко рухатися в іншому напрямку, але й гарантує, що структура не зруйнується при зміні частини. Мінімалістичний Flask і його можливість для розробки невеликих веб-додатків означає, що він навіть більш гнучкий, ніж сам Django.

3. Легкий у розробці. Як і в Django, можливість легкої розробки є ключовим фактором для того, щоб веб-розробники могли зосередитися на швидкому кодуванні, не зациклюючись на дрібницях. По суті, мікрофреймворк простий і зрозумілий для веб-розробників, що не тільки економить їх час і зусилля, але й дає їм більше контролю над кодом і можливостями.

4. Легкий. Коли ми використовуємо цей термін по відношенню до інструменту або фреймворку, ми говоримо про його конструкцію - в ньому мало складових частин, які потрібно збирати і перезбирати, і він не покладається на велику кількість розширень для функціонування. Такий дизайн дає веб-розробникам певний рівень контролю.

5. Модульність. Flask також підтримує модульне програмування, тобто його функціональність можна розділити на кілька взаємозамінних модулів. Кожен модуль діє як незалежний будівельний блок, який може виконувати одну

частину функціоналу. Разом це означає, що всі складові частини структури є гнучкими, рухомими і можуть бути протестовані самостійно.

6. Документація. Користувачі Flask знайдуть велику кількість прикладів і порад, розташованих у структурованому вигляді. Це заохочує розробників використовувати фреймворк, оскільки вони можуть легко ознайомитися з різними аспектами та можливостями інструменту. Ви можете знайти документацію Flask на їхньому офіційному сайті.

До недоліків Flask варто віднести:

1. Невелика кількість інструментів. Легкість цього мікрофреймворку неминуче має певні недоліки. Головний з них полягає в тому, що на відміну від Django, Flask не має великого набору інструментів. Це означає, що розробникам доведеться вручну додавати розширення, такі як бібліотеки. І, якщо ви додасте величезну кількість розширень, це може почати сповільнювати роботу самого додатку через безліч запитів.

2. Складно розібратися у великому додатку Flask. Через те, що розробка веб-додатку з використанням Flask може бути заплутаною, веб-розробнику, який приєднується до проекту в середині його реалізації, може бути складно зрозуміти, як він був спроектований. Модульна природа мікрофреймворку може знову стати проблемою для кодерів, яким доведеться ознайомитися з кожною складовою частиною.

3. Витрати на обслуговування. Оскільки Flask настільки універсальний в плані того, з якими технологіями він може взаємодіяти, досить часто компанії, які використовують Flask, несуть додаткові витрати на підтримку цих технологій. Наприклад, якщо технологія, що взаємодіє з вашим додатком Flask, застаріває або припиняє своє існування, компанії доведеться шукати нову сумісну технологію. Чим складнішим стає додаток, тим вищими є потенційні витрати на його підтримку та впровадження.

Варто зазначити, що для невеликих проектів Flask підходить найкраще, а для більш масштабних - веб-фреймворк Django. . Підвівши підсумки, Flask найкраще підходить для розробки даного програмного забезпечення, адже простота

та невеликий розмір Flask є перевагою під час розробки програмного забезпечення.

2.3 Вибір формату даних

2.3.1 Мова розмітки XML

Розширювана мова розмітки (XML) дозволяє визначати та зберігати дані у зручному форматі. XML підтримує обмін інформацією між комп'ютерними системами, такими як веб-сайти, бази даних і сторонні програми. Заздалегідь визначені правила спрощують передачу даних у вигляді XML-файлів через будь-яку мережу, оскільки одержувач може використовувати ці правила для точного та ефективного зчитування даних.

До переваг XML можна віднести:

1. Сумісність. XML сумісний з широким спектром програмного забезпечення і систем, а це означає, що його можна використовувати для обміну інформацією між різними типами комп'ютерів і програм.

2. Гнучкість. XML розроблений таким чином, щоб бути гнучким, а це означає, що його можна використовувати для створення власних тегів і атрибутів. Це дозволяє користувачам створювати власну структуру для своїх даних, що може полегшити організацію та пошук потрібної інформації.

3. Простота використання. XML відносно простий у вивченні та використанні, що робить його доступним для широкого кола користувачів. Це означає, що ним можуть користуватися люди з різним рівнем технічної підготовки, і його можна використовувати для створення найрізноманітніших додатків і проектів.

До недоліків XML варто віднести:

1. Складність. XML може бути досить складним у використанні та розумінні, особливо для тих, хто не знайомий з цією технологією. Це може ускладнити створення, редагування або читання XML-документів.

2. Багатослівність. XML-документи можуть бути досить багатослівними, тобто містити багато непотрібної інформації. Це може зробити їх більшими за розміром і ускладнити роботу з ними.

3. Відсутність стандартизації. Не існує єдиного стандарту для структурування XML-документів, що може призвести до плутанини та проблем із сумісністю.

4. Відсутність підтримки. Не всі програми або мови програмування підтримують XML, що може ускладнити роботу з ним, якщо ви використовуєте програму або мову, яка його не підтримує.

2.3.2 JSON Schema

JSON - це текстовий формат обміну даними, отриманий із JavaScript, який використовується у веб-службах та інших підключених програмах. JSON визначає лише дві структури даних: об'єкти та масиви. Об’єкт - це набір пар імен-значення, а масив - це список значень. JSON визначає сім типів значень: рядок, число, об'єкт, масив, true, false і null. [17]

JSON часто використовується як загальний формат для серіалізації та десеріалізації даних у програмах, що взаємодіють між собою через Інтернет. Ці програми створюються з використанням різних мов програмування і працюють у дуже різних середовищах. JSON підходить для цього сценарію, оскільки це відкритий стандарт, його легко читати і писати, а також він більш компактний, ніж інші подання.

Схема JSON виконує ряд функцій, включаючи перевірку екземпляра JSON. Цей документ визначає мову схеми JSON для опису семантики документів JSON, їх перевірки згідно з попередньо встановленими шаблонами, т.д.

До переваг JSON можна віднести:

1. Швидкодія. Синтаксис JSON дуже простий у використанні. JSON забезпечує легкий синтаксичний аналіз даних і швидку обробку даних. Оскільки його

синтаксис дуже малий і легкий, його швидкодія є значною перевагою у порівнянні з аналогами.

2. Підтримка схем. Він має широкий спектр сумісності браузерів з операційними системами, тому додатки, які використовують JSON, не потребують багато зусиль для того, щоб зробити їх сумісними з браузерами. Під час розробки розробник думає про розробку додатків на різні браузери, але JSON полегшує цю функціональність.

3. Простота і читабельність. Однією з головних переваг JSON є його простота і читабельність. JSON легко писати і розуміти, оскільки він використовує зрозумілий для людини формат пар ключ-значення і масивів. JSON не вимагає ніяких спеціальних тегів, атрибутів або схем, на відміну від XML, який є ще одним поширеним форматом даних для веб-додатків. JSON також підтримує поширені типи даних, такі як рядки, числа, булеві числа, нулі, об'єкти та масиви, які можна вкладати та комбінувати різними способами. JSON можна легко конвертувати в об'єкти JavaScript та з них, що робить його зручним для веб-розробників, які використовують JavaScript як основну мову сценаріїв для веб-додатків.

4. Інструмент для обміну даними:

JSON є найкращим інструментом для обміну даними будь-якого розміру, навіть аудіо, відео тощо. Це тому, що JSON зберігає дані в масивах, що спрощує їх передачу. З цієї причини JSON є чудовим форматом файлів для веб-додатків.

Недоліки JSON:

1. Безпека. Одним з основних недоліків JSON як формату даних для веб-додатків є його проблеми з безпекою та валідацією. JSON вразливий до різних атак, які можуть порушити цілісність і конфіденційність даних та додатку. JSON також не має власної підтримки коментарів, що може ускладнити документування та налагодження даних.

2. Гнучкість та розширюваність

Ще одним недоліком JSON є обмеження його гнучкості та розширюваності як формату даних для веб-додатків. JSON базується на фіксованому і заздалегідь

визначеному наборі типів даних, що може обмежувати діапазон і складність даних, які можуть бути представлені та обмінюватися. У нашому випадку, це є перевагою, адже це дає структурованість та чіткість для перевірки подій.

Підвівши підсумки, варто зазначити, що представлення JSON зазвичай є більш компактними та простими, ніж подання XML, оскільки JSON не має закриваючих тегів, а також має менший розмір файлів, що забезпечує швидку передачу інформації, і саме тому використання JSON є кращим для розробки даного програмного забезпечення.

2.4 Висновки до розділу

Для розробки даного програмного забезпечення було обрано мову програмування Python, що дозволить суттєво підвищити швидкість та простоту розробки цієї програми та надає можливість у майбутньому пришвидшені та покращенні даної програми. Завдяки цьому розробка та створення нових модулів програми буде швидкою та більш ефективною. Використання веб-фреймворків у даному програмному модулю буде сприяти повторному використання модулів.

Flask найкраще підходить для розробки даного програмного забезпечення, адже простота та невеликий розмір Flask є перевагою під час розробки програмного забезпечення.

JSON часто використовується як загальний формат для серіалізації та десеріалізації даних у програмах, що взаємодіють між собою через Інтернет.

Представлення JSON зазвичай є більш компактними та простими, ніж подання XML, оскільки JSON не має закриваючих тегів, а також має менший розмір файлів, що забезпечує швидку передачу інформації.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВАЛІДАЦІЇ ПОДІЙ У JSON ФОРМАТІ

3.1 Підготовка середовища для розробки програмного коду

Перед початком розробки програмного забезпечення було вирішено використати систему контролю версій GitHub [18]. Дана система зручна для розробки програмного забезпечення через свою простоту, надійність, стабільність та можливість відкату до попередніх стабільних версій. Це в свою чергу дає змогу розробникам розробляти програмне забезпечення одночасно у паралельних вітках.

Для ініціалізації репозиторію використовувались наступні команди:

– git clone URL;

– git add;

– git commit -m “first commit”;

– git remote add origin URL;

– git push origin.

Для встановлення python можна використати кілька засобів. Першим засобом для його встановлення є завантаження продукту з офіційного сайту https://www.python.org/ та подальше встановлення на комп’ютер. При наявності на машині встановленого та налаштованого WSL (Windows Subsystem for Linux) можна встановити python у такі ж самі кроки, що і встановлення його на Ubuntu

Для встановлення python через WSL потрібно виконати наступні кроки:

– sudo apt-get update;

– sudo apt-get install python3.10;

– sudo apt-get install python3.10-venv.

Наступним кроком є створення віртуального середовища для розробки. Для цього необхідно виконати такі команди:

– mkdir <ім’я директорії>

– cd < ім’я директорії >

– python3.10 -m venv < ім’я директорії для віртуального середовища>.

У подальшому буде використовуватись середовище розробки під назвою PyCharm. На відміну від аналогів, дане середовище розробки у своєму функціоналі має підтримку запуску програмного коду з консолі та його подальшу обробку [19]. До переваг PyCharm можна віднести:

– PyCharm простий у використанні та налаштуванні.

– Синтаксис і обслуговування помилок можуть допомогти в аналізі синтаксичних помилок перед компіляцією коду, зменшуючи помилки.

– PyCharm постачається з низкою плагінів, які можуть допомогти вам покращити ваш проект.

– Допомога при імпорті дозволяє імпортувати бібліотеки, яких не вистачає.

– Залежно від контексту завершення коду дозволяє заповнювати імена класів, методів, аргументів та інших змінних.

– Але також слід привести недоліки PyCharm:

– Найбільшим недоліком PyCharm є те, що він займає більше місця, ніж інші текстові редактори, зменшуючи функціональність коду.

– Встановлення змінної віртуального середовища може бути складним для новачків.

– Професійна версія досить дорога.

– Видання спільноти підходить виключно для розробки Python і не дозволяє використовувати інші мови програмування.

Після завантаження проекту з GitHub директорія з проектом буде представлена у боковій панелі вікна розробки PyCharm, що зображено на рис. 3.1. За допомогою цього меню навігація між файлами розробки є зручною та швидкою, а паралельні вікна дають можливість порівнювати код під час його написання.

Для встановлення потрібних для програми бібліотек слід виконати наступні команди, які зображено на рис. 3.2 та рис. 3.3:

1. source venv/bin/activate - запускаємо попередньо встановлене віртуальне середовище.

2. pip install -r requirements/base.txt – встановлюємо потрібні бібліотеки.

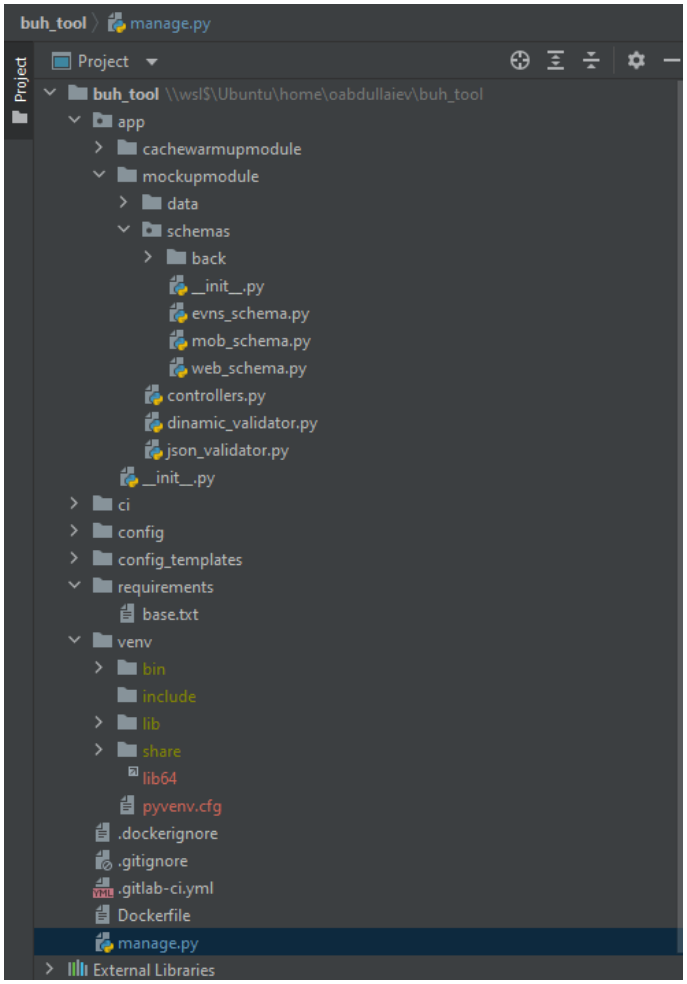

Рисунок 3.1 – Бокова панель та структура проекту

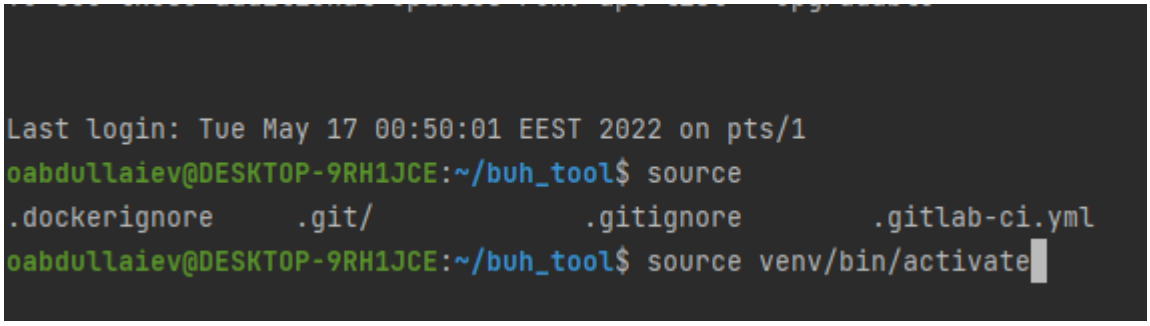

Рисунок 3.2 – Запуск віртуального середовища

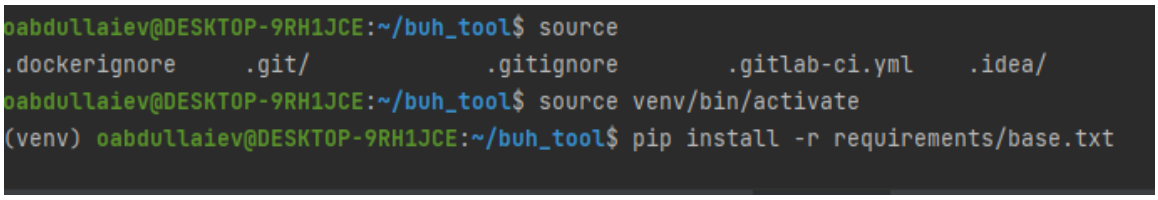

Рисунок 3.3 – Встановлення бібліотек

Після налаштування віртуального середовища та встановлення потрібних бібліотек, модуль готовий для запуску. Для цього використовується Flask - веб-фреймворк, який дозволяє створити додатки на програмній мові Python.

Flask є веб-фреймворком. Веб-фреймворк - це, як правило, фреймворк, який майже не залежить від зовнішніх бібліотек. фреймворк є легким, існує невелика залежність від оновлення та стеження за помилками безпеки. Даний фреймворк є гнучким та дозволяє використовувати різноманітні бібліотеки, на відміну від його аналога Django

До мінусів даного веб-фреймворку можна віднести те, що Flask є синхронним [16], але це не заважає йому обробляти близько 120 запитів, що конкурує навіть зараз у порівнянні з асинхронними рішеннями

Для запуску Python сервера необхідно прописати у консолі наступну команду (рис.3.4):

/home/oabdullaiev/buh_tool/venv/bin/python /home/oabdullaiev/buh_tool/manage.py runserver --threaded --port=1111

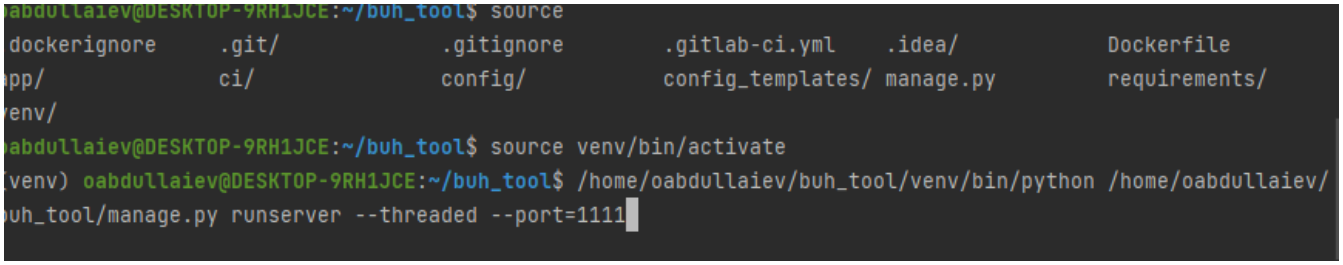

Рисунок 3.4 – Запуск серверу

UML-deployment diagram модулю валідації подій наведено у додатку Б.

3.2 Розробка алгоритму перевірки подій на основі шаблонів

У основі модуля для валідації подій лежить алгоритм перевірки подій. Для цього було спроектовано та розроблено модуль валідації подій на основі шаблонів. Основна логіка цього модулю відтворена у функції validate файлу json_validator.py.

Першим кроком є перевірка платформи додатку задля використання потрібного шаблону подій:

```
def decorated(actions_array, schema=None, methods=None,
    format_checker=None, *args, **kwargs):
        input_json = request.get_json()
        try:
            if input_json["app_info"] == "ANDROID" or input_json["app_info"]
    == "IOS" or input_json["app_info"] == "STB" or input_json["app_info"]
    == "tvOS":
                schema=mob_schema
            elif input_json["app_info"] == "WEB" or input_json["app_info"] ==
    "SmartTV":
                schema=web_schema
            else:
                schema=evns_schema
        except:
            #in case when json not have app_info
            schema=evns_schema
        logging.info(schema['definitions']['title'])
```

У цьому фрагменті коду можна замінити перевірку наприклад на ім’я проекту замість платформи. Таким чином, з’являється можливість використовувати модуль валідації подій обираючи потрібний параметр для перевірки згідно певних шаблонів. Наприклад, при отриманні двох повністю різних між собою версій.

Після отримання потрібної платформи, модуль перевіряє, чи є доступна для цієї платформи схема шаблонів. Якщо, для певної платформи немає шаблонів, то програма обирає стандартизований для всіх платформ:

```
else:
    schema=evns_schema
```

Наступний крок безпосередньо валідація json схем згідно шаблонів. Для цього використовується json validator бібліотека, яка порівнює чи збігаються параметри події згідно раніше прописаних шаблонів. Якщо сама подія у вигляді json був пустим, то модуль повідомить про це в консолі та логах програми:

```
if request.method in methods:
    validator = self.validator if self.validator else
validator_for(schema)
    if not input_json or input_json is None:
        logging.critical(json.dumps(input_json))
        raise JsonIncorect('Json is null')
```

Якщо певний параметр події має помилку, наприклад не потрапляє в межі допустимих значень, то інформація щодо цього буде виведена у консоль на відповідь запиту:

```
            for error in errors:
        message = error.message
        message = message.replace("'", "", 4)
        #logging.warning(f'message:{message}')
        if list(error.path):
            actual_errors = list(error.path)[0]
            actual_errors = "\"" + actual_errors +"\":\""+ message +
"\","

            #logging.warning(actual_errors)
        else:
```

```
                    actual_errors = "\"ValidationError_"+str(iter_i) +"\":\""
+ message + "\","
                    iter_i+=1
                tmp_error_string += actual_errors
            tmp_error_string = "{"+tmp_error_string[:-1] +"}"
            tmp_error_string=eval(tmp_error_string)

            full_error = dict(derrors, **tmp_error_string)
            if full_error:
                tmp_error_string =
json.dumps([{"all_errors":full_error,"schema_errors":tmp_error_string,"dinamic_er
rors":derrors}])
                tmp_error_string = json.loads(tmp_error_string)
                raise JsonValidationError(tmp_error_string)
```

Для форматування та виведення інформації після валідації подій використовується наступна функція, яка формує json для зручного аналізу проблеми. Спершу, формується список помилок, а далі підбиваються підсумки помилок:

```
def json_validator(e):
    errors = str(e)
    return jsonify({"error":errors})

def validation_error(e):
    input_json_with_err = request.get_json()
    try:
        evn_name = input_json_with_err["evn"]
    except:
        evn_name = None
    err =str(e)
    errors=eval(err)
    response = jsonify({
```

```
    "head": {
        "title": "event",
        "itype": "event",
        "event_name": evn_name,
        "ts": unixtime,
    },
    "actual_errors":errors,
    "event":request.get_json(),
    "status": 400
    })
    errors= json.dumps({"actual_errors":errors})
    errors = json.loads(errors)
    input_json_with_err.update(errors)
    evns_fail.insert(0,(input_json_with_err))
    logging.critical(json.dumps(input_json_with_err))
    return response,400
def create_pass_message():
    input_json = request.get_json()
    evns_pass.insert(0,input_json)
    return jsonify({
        "head": {
        "title": "event",
        "itype": "event",
        "event": input_json["evn"],
        "ts": unixtime,
            },
        "info":{
            "n_events": 1
        },
        "status": "null"})
```

Для опису структури проекту, було створено UML діаграма модулю валідації, UML діаграма модулю перевірки події, діаграма класів модулю валідації подій, UML-class diagram модулю валідації подій та UML-object diagram модулю валідації подій, які наведено у додатку Б.

3.3 Розробка шаблонів перевірки валідації подій

Для валідації подій у даному проекті використовується Json Schema – словник, який дозволяє перевіряти документи json згідно їх шаблонам. Схема JSON виконує ряд функцій, включаючи перевірку схем JSON. Перевіряючи надані можливі значення та їх формат JSON Schema визначає згідно шаблону чи є надіслана подія правильною.

Основне використання схеми JSON — опис структури та перевірки документів JSON. Сьогодні ми маємо доступ до величезної кількості інструментів, які дозволяють нам генерувати схему JSON на основі наших власних документів JSON. Ми можемо легко перетворити документ JSON у схему JSON [17].

Тип об'єкта JSON може бути оголошено або як примітив (тривіальний), інший об'єкт JSON або як масив, як показано на рис. 3.5.

Після створення схеми ми можемо використовувати її для перевірки нашого документа JSON, наприклад, у нашому API. Як окремий продукт, JSON Schema має свій власний інтерфейс. Приклад використання та інтерфейс показано на рисунку 3.6

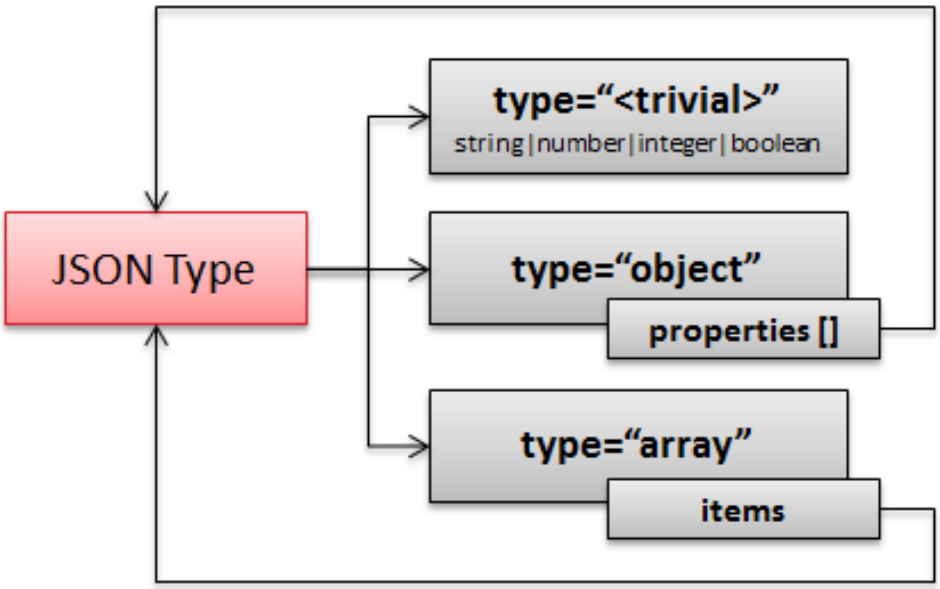

Рисунок 3.5 – Структура JSON об'єкту

.

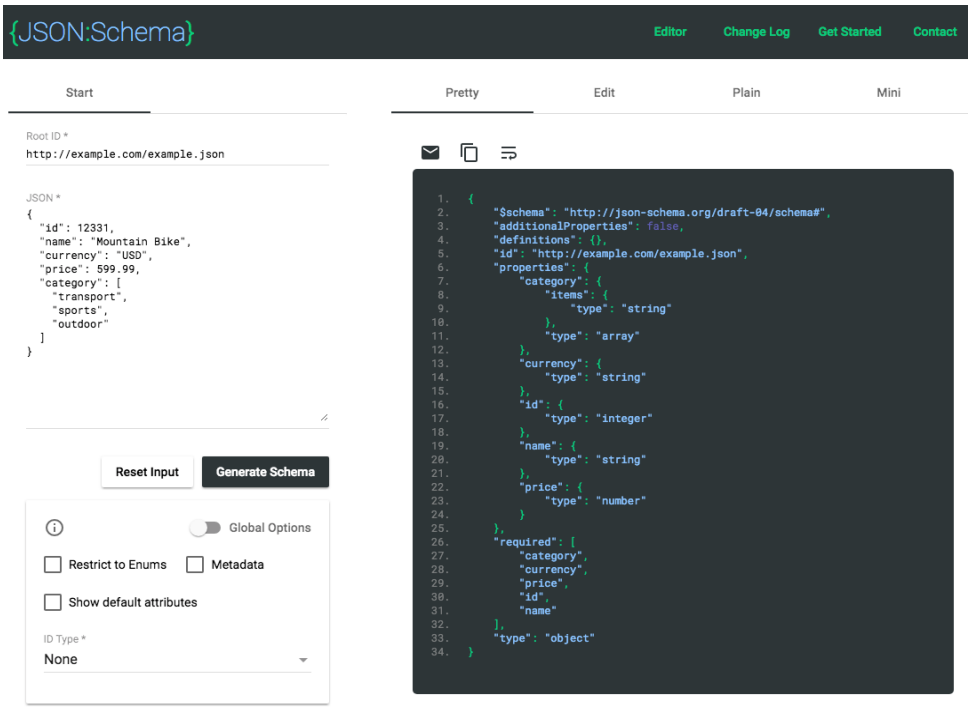

Рисунок 3.6 – Інтерфейс JSON Schema та приклад формату даних

Функціонал json Schema дозволяє визначити які параметри для події є обов'язковими, які з них є можливими, а також перевіряє значення цих полів. Наприклад, якщо подія media_play не має поля title, який є обов'язковим, то JSON Schema запише цю помилку для наступного опрацювання. Також, можна

визначати довжину значення для параметру події. Якщо згідно шаблону параметр не дотримується довжини значення, то відповідне повідомлення також з’явиться у виводі списку помилок.

```
"device_id": {
  "$id": "#/definitions/parameters/properties/device_id",
  "type": "string",
  "minLength": 10,
  "maxLength": 80,
  "title": "The Device_id Schema",
  "default": "",
  "examples": [
    "8d24c56613ffa4da"
  ],
  "pattern": "^(.*)$"
 },
```

Завдяки гнучкості алгоритму валідації подій, тестувальники можуть використовувати одразу кілька схем під різні платформи. Таким чином, для кожної платформи розробники можуть надавати шаблони для перевірки подій. Обов’язковість певних параметрів спрощують та прискорюють роботу тестувальників.

```
"allOf": [
 {
   "$ref" : "#/definitions/parameters",

 },

 {
   "if": {
    "properties": {
      "app_info": { "enum":["IOS","ANDROID","STB","tvOS","tvOS-Demo"] }
     }
```

```
    },
   "then": {
    "required": [
     "evn",
     "etype",
     "ts",
     "device_id",
     "device_type",
     "device_info",
     "version",
     "app_info",
     "customer",
     "app_version",
     "network",
     "device_os",
     "temp_user_id",
     ],
    }
  },
```

Завдяки required полям тестувальник буде допускати менше помилок при тестуванні програмного забезпечення, а його робота буде більш ефективною. UML-use case diagram модулю валідації подій наведено у додатку Б.

3.4 Висновки до розділу

Під час розробки даного програмного забезпечення для валідації подій у json форматі було використано певний ряд технологій для реалізації поставленої задачі до яких входять: python – швидка та гнучка мова програмування, Flask –

веб-фреймворк для підняття серверу, json schema – для побудови шаблонів json подій та інші.

Модуль для валідації подій у json форматі є гнучким у налаштуванні, так як, розроблена система шаблонів на основі JSON Schema дозволяє конфігурувати обов’язкові поля та параметри, їх мінімальні та максимальні значення, типи, патерни і т.д. Розроблене програмне забезпечення дозволяє проводити валідацію подій на основі різних схем. Таким чином тестувальники можуть перевірити події для різних платформ з мінімальними змінами в структурі подій.

Перевірка одразу кількох платформ додатку одночасно прискорить валідацію подій для тестувальників, так як, різні команди використовують свій окремо створений шаблон. Для відправки подій та симуляції додатку використовується Postman, як інструмент для API тестування.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Відлагодження програми та виправлення помилок

Під час створення будь якого програмного продукту одним з необхідним шляхом є тестування, відлагодження та виправлення помилок. Життєвий цикл розробки програмного забезпечення наведено на рис.4.1. Найчастіше, саме неуважність програміста призводить до появи проблем в роботі програми. Але також потрібно мати на увазі, що при першому релізі та запуску, жодна програма не дасть потрібного результату [20-22]. Окрім цього існують також інші джерела виникнення помилок. Наприклад, самі вхідні дані, проблеми з обладнанням, помилки самого користувача і т.д.

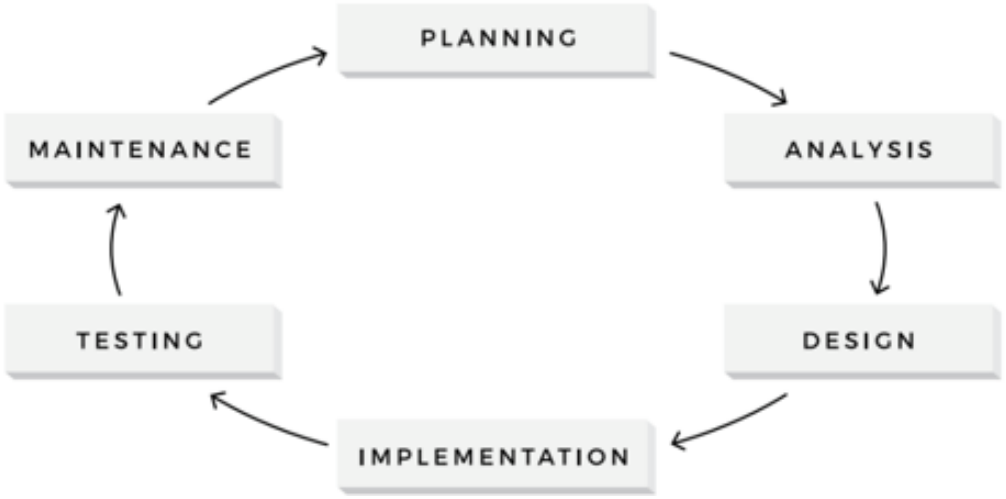

Рисунок 4.1 – Життєвий цикл розробки програмного забезпечення

Після розробки програмного коду, програмний продукт передається тестувальнику на тестування. Його робота полягає у всебічній перевірці роботи програми згідно попередньо складеної документації. Існує декілька основних типів помилок серед яких потрібно розглянути синтаксичні помилки та логічні помилки.

Синтаксичні помилки виникають при написанні коду, коли програміст не дотримувався правил написання коду. В дуже частих випадках, саме середовище

розробки вкаже на помилку. Якщо у коді виявляються синтаксичні помилки, то середовище розробки не запустить процес компіляції та відмовиться запускати код. У частих випадках це пропущені коми, фігурні дужки чи неправильне написання функцій.

Логічними помилками називають помилки, які виникають через невірність алгоритму. Коли програмному забезпеченню не вдається виконати те, що планував програміст, кажуть, що він зробив логічну помилку. Саме середовище розробки запустить скомпільований код, але правильність вихідних даних не буде вірною. Синтаксичну помилку, на відміну від логічної, набагато легше виявити та виправити. Логічна помилка ж потребує часу та відлагодження для виправлення [22]. Для її виявлення програмісту потрібно визначити помилку шляхом перевірки коду.

Можемо підвести підсумки, що тестування програмного продукту є однією із складових під час розробки. Найчастіше, помилка виникає через людську необачність. Коли програмному забезпеченню не вдається виконати те, що планував програміст, це означає, що він зробив логічну помилку. Синтаксичні помилки зазвичай легко виправити, оскільки компілятор повідомить вам, де сталася проблема, і все, що вам потрібно зробити, це виправити її.

4.2 Види тестування

Тестування виявляє наявність помилок. Мета тестування — знайти недоліки в програмному забезпеченні. Однак, як би ретельно не оцінювався продукт, ми ніколи не можемо бути впевненими, що він бездефектний. Тестування виконується для того, щоб зменшити кількість проблем, які ще не виявлено. Виконати бездоганне тестування неможливо, тому тестування передбачає досягнення такої якості продукту, яким буде задоволений звичайний користувач.

Неможливо провести вичерпне тестування. У програмі немає можливості перевірити всі можливі комбінації введених даних, сценаріїв і передумов. Замість того, щоб витрачати тижні на вигадування мільйонів менш ймовірних ситуацій, краще зосередитися на тих, які можуть мати найбільший вплив [23,24].

Повторення одного і того ж набору тестів не призведе до виявлення нових проблем. Ці тестові сценарії стають застарілими, як тільки виявлені проблеми виправляються. У результаті дуже важливо регулярно аналізувати та оновлювати тести, щоб адаптуватися та, можливо, виявити додаткові недоліки.

Кластеризація дефектів. Цей підхід часто називають реалізацією принципу Парето для тестування програмного забезпечення. Це говорить про те, що близько 80% всіх помилок часто виявляються лише в 20% компонентів системи. В результаті, якщо проблема виявлена в одному модулі програмної програми, цілком імовірно, що є інші. Ось чому необхідно належним чином перевірити цей аспект продукту.

Важливий також контекст тестування. Різні програми слід оцінювати порізному залежно від їх функції чи сектора. Хоча безпека може бути головним пріоритетом для фінансового продукту, для ділового веб-сайту це не так. Останнє, навпаки, підкреслює зручність використання та швидкість.

Класифікація видів тестування та сам процес тестування є комплексним. Сам процес тестування програмного забезпечення можна поділити на статичне та динамічне тестування, що показано на рис. 4.2.

Є певний перелік ознак для класифікації видів тестування:

За об'єктом тестування:

- функціональне тестування (functional testing);
- тестування продуктивності (performance testing):
- навантажувальне тестування (load testing);
- стрес-тестування (stress testing);
- тестування стабільності (stability / endurance / soak testing);
- тестування зручності використання (usability testing);
- тестування інтерфейсу користувача (ui testing);

– тестування безпеки (security testing);
– тестування локалізації (localization testing);
– тестування сумісності (compatibility testing).

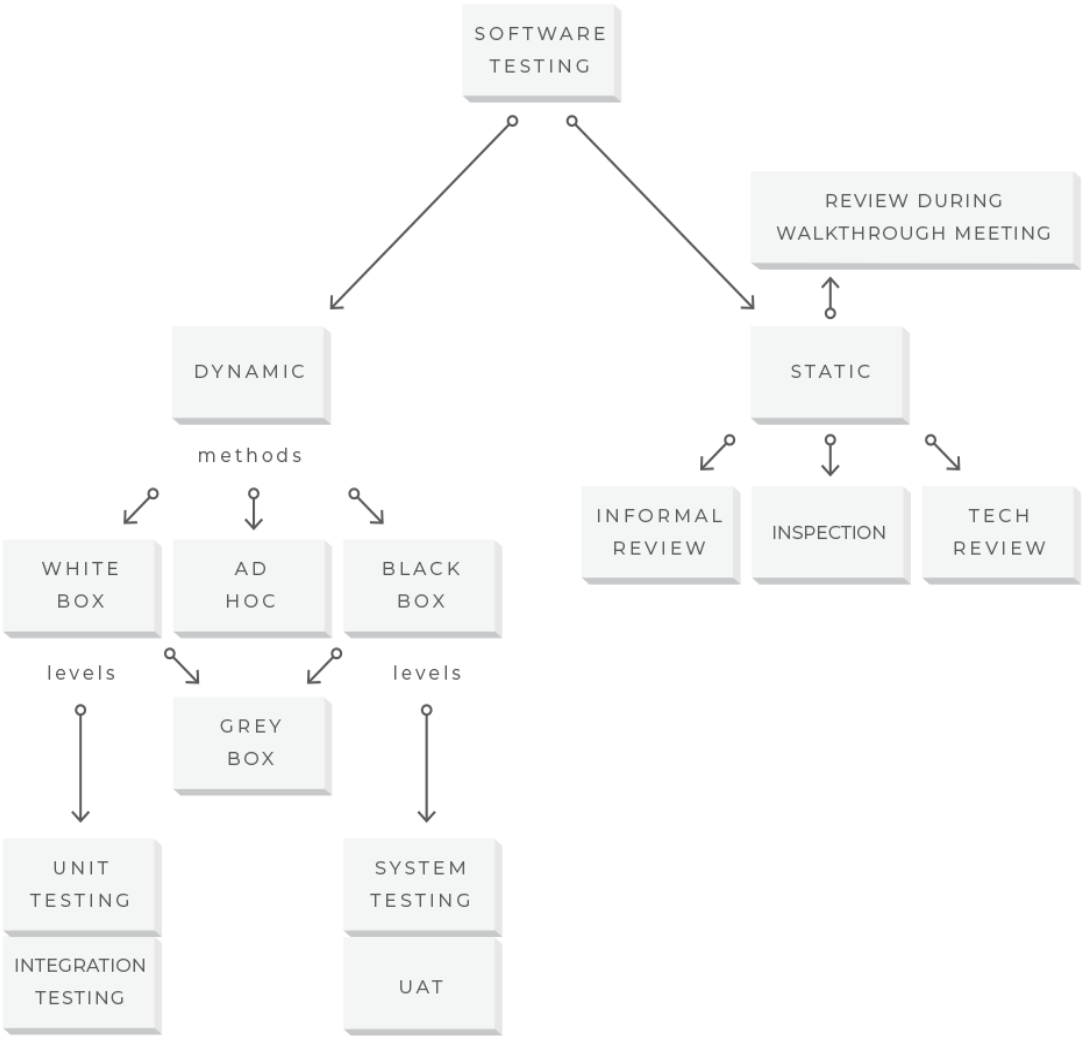

Рисунок 4.2 – Розподіл на статичне та динамічне тестування

За ступенем автоматизації:
– ручне тестування (manual testing);
– автоматизоване тестування (automated testing);
– напів-автоматизоване тестування (semiautomated testing).

За ступенем ізольованості компонентів:
– компонентне (модульне) тестування (component / unit testing);
– інтеграційне тестування (integration testing);

– системне тестування (system / end-to-end testing).

За часом проведення тестування:

– Альфа-тестування (alpha testing):

– тестування при прийманні (smoke testing);

– тестування нової функціональності (new feature testing);

– регресійне тестування (regression testing);

– тестування при здачі (acceptance testing);

– бета-тестування (beta testing).

За ознакою позитивності сценаріїв:

– позитивне тестування (positive testing);

– негативне тестування (negative testing).

За ступенем підготовленості до тестування:

– тестування за документацією (formal testing);

– ед хок (інтуїтивне) тестування (ad hoc testing).

За знанням системи:

– тестування чорного ящика (black box);

– тестування білого ящика (white box);

– тестування сірого ящика (gray box).

Підвівши підсумки можна зазначити:

– тестування повинно проводитися неупередженими фахівцями в незалежний спосіб.

– окрім дійсних і очікуваних введених значень, перевірте їх на недійсні та неочікувані.

– тестуванню підлягає лише статична частина програмного забезпечення (в процесі тестування не слід вносити жодних змін).

– щоб визначити передбачувані результати тесту, використовуйте розгорнуту та ретельну документацію.

4.3 Розгляд інструментів для автоматизованого тестування

Для ефективного тестування програмного забезпечення набирають популярності інструменти для автоматизованого тестування. Були оглянуті популярні інструменти тестування, серед яких жоден не впорався з поставленою задачею для тестування програмного модулю у повній мірі. Оглянуті інструменти або підходять тільки для певних задач, або можуть допомогти тільки в тестуванні певної частини поставленої задачі.

Серед розглянутих інструментів автоматизації тестування було обрано:

– Selenium IDE
– Selenium Web Driver
– Apache JMeter
– Silk Test
– Postman
– Paw
– Insomnia

Selenium IDE являє собою плагін для браузеру, який записує дії користувача, та у подальшому зберігати їх для подальшої обробки, наприклад, в WebDriver або Selenium RC. В його складі є купа інструментів та різних підходів для автоматизації тестування сайтів та інших додатків на різних платформах. Даний інструмент автоматизованого тестування має у своєму складі технічну документацію на різних мовах, що полегшує його налаштування та роботу з ним.

WebDriver є однією з найпотужніших і найвідоміших технологій Selenium. WebDriver — це розширена версія Selenium RC, яка додає додаткові функції та усуває багато недоліків Selenium RC. На відміну від Selenium IDE, WebDriver підтримує широкий спектр сучасних браузерів і систем. WebDriver не вимагає запуску сервера Selenium перед виконанням тестових сценаріїв.

Selenium WebDriver - це бібліотека програмного забезпечення для роботи та роботи з браузерами. WebDriver - це сімейство драйверів для різних браузерів (рис. 4.3), а також набір клієнтських бібліотек для цих драйверів на різних мовах програмування, таких як Java, Python, C ++ тощо.

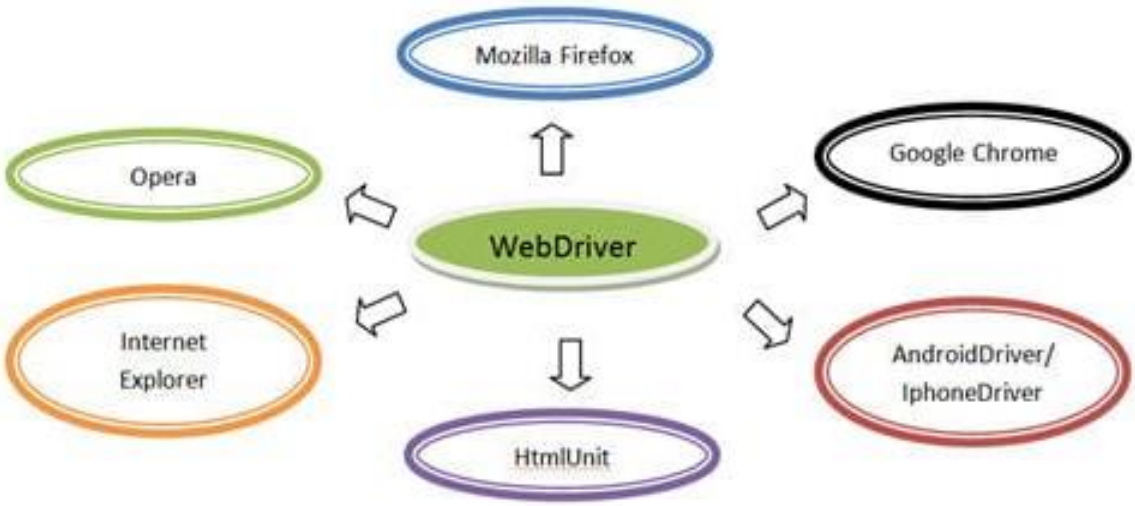

Рисунок 4.3 – Сумісність WebDriver з різними браузерами

Імітуючи взаємодію користувача з WebDriver, ви можете автоматизувати Microsoft Edge. Модульні тести WebDriver забезпечують такі переваги перед модульними тестами JavaScript, які виконуються у браузері:

– WebDriver надає вам доступ до функцій і даних, яких немає у JavaScript у браузерах.

– WebDriver є більш точним, ніж модульні тести JavaScript для моделювання подій на рівні користувача або ОС.

– За один сеанс тестування WebDriver підтримує багато вікон, вкладок і веб-сторінок.

– В одній системі WebDriver запускає численні сеанси Microsoft Edge.

Apache Jmeter використовується для перевірки навантаження на систему, або іншими словами навантажувального тестування. Даний інструмент автоматизованого тестування може проводити навантажувальні тести для різних типів з'єднання, таких наприклад як JMS, IMAP, FTP, SOAP, HTTP, JDBC, TCP, LDAP, POP3. Найголовнішою перевагою є можливість створення та надсилання великої кількості запитів, імітуючи навантаження робочих систем, наприклад, відео-хостингу.

JMeter також можна використовувати як монітор, хоча він частіше використовується для простого моніторингу, а не для складного моніторингу. Його також можна використовувати для виконання деяких функціональних тестів. Jmeter також має інтерфейс Selenium, що дозволяє йому виконувати сценарії автоматизації поряд із тестуванням продуктивності або навантаженням [25].

Однією з основних переваг JMeter є додавання сторонніх плагінів, які допомагають користувачам створювати свої доповнення та поширювати їх. Найпопулярнішими є саме видозмінені графіки результатів навантаження. Для цього потрібно спочатку встановити Plugin Manager, який надалі у вигляді вікна надає можливість додавати та видаляти потрібні плагіни. Зразок відображення графіків в JMeter показано на рис. 4.4.

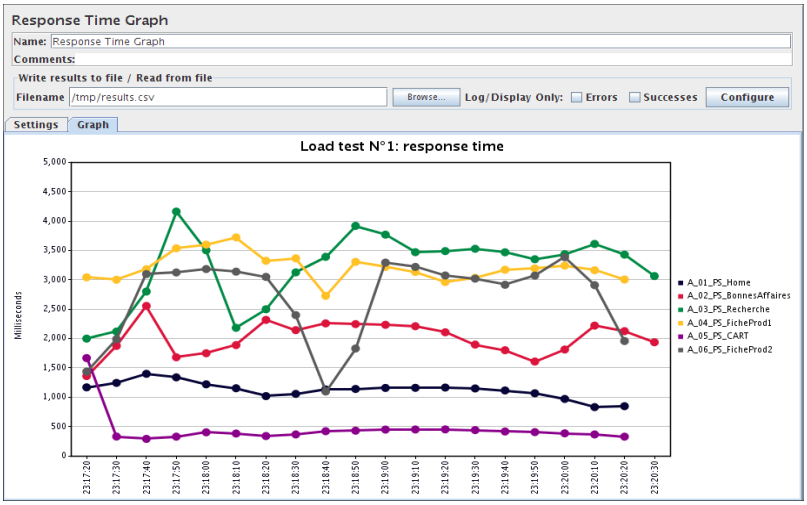

Рисунок 4.4 – Графік навантаження в JMeter

Silk Test – це інструмент, створений виключно для оцінки регресії та функціональності. Silk Test — це найпопулярніший у галузі інструмент функціонального тестування для додатків електронного бізнесу, незалежно від того, чи є вони на базі Windows, веб-, Java або клієнт/сервер. Silk Test також включає планування тестів, керування, прямий доступ до бази даних і перевірку, а також потужну та надійну мову сценаріїв 4Test, систему відновлення автоматичного тестування та можливість тестування на багатьох платформах, браузерах і технологіях.

Існує два способи створити автоматизовані тести за допомогою Silktest:

– Під час переміщення по програмі використовуючи команду Record Testcase, щоб записувати дії та процеси перевірки.

– Використовуючи мову сценаріїв Visual 4Test, написавши тест-кейс вручну.

Під час використання програми команда Capture / Testcase використовується для запису дій і процесів перевірки. Visual 4Test — це об'єктно-орієнтована мова, яка використовується для запису тестів. Записані тести відображаються як логічний слід усіх дій, які зробив користувач. Система перевірки Silk Test дозволяє записати крок перевірки, вибрати зі списку атрибутів, той що релевантний та перевіряється. Також, є можливість перевірити, чи текст збережено в текстовому полі.

У Silktest існує можливість писати тести, які можуть виконувати широкий спектр варіантів тестів. Ключове тут повторне використання. Тест-кейс можна створити з використанням таких параметрів, як вхідні дані та очікувані результати. Цей «керований даними» тест-кейс насправді є прикладом тестів, які керують і перевіряють додаток шляхом виконання певних завдань. Дані, які містить кожен екземпляр, відрізняються [26]. Зміни в графічному інтерфейсі призведуть до меншої роботи щодо оновлення тестів, оскільки за допомогою цього методу розробляється значно менше тестів.

POSTMAN — це клієнт API для написання, тестування, спільного використання та документування API. Він використовується для бекенд-тестування, А саме коли запит надсилається на сервер, і він в свою чергу надсилає відповідь на надісланий запит.

Завдяки підтримці API, інструментів, інтелектуальності, робочим просторам та інтеграціям ви можете створювати кращі та швидші API. Він заснований на програмному забезпеченні з відкритим кодом.

Репозиторій API — за допомогою уніфікованої платформи ви можете швидко зберігати, класифікувати та співпрацювати над усіма своїми матеріалами, пов’язаними з API, такими як тест-кейси, специфікації та документація.

Інструменти — він пропонує різноманітні інструменти API, які допомагають у проектуванні, тестуванні, документації та інших аспектах життєвого циклу API.

Інтелектуальність – він надає способи взаємодії зі складними даними та уявлення про операції API, такі як сповіщення, пошук, попередження про безпеку та звіти, тощо.

Робочі простори — Postman дозволяє організовувати та співпрацювати над роботою API з будь-якої точки світу. Особисті, командні та громадські робочі місця доступні для використання.

Postman може бути інтегрований зі сховищами коду, CI/CD тощо.

Paw: найдосконаліший інструмент API для Mac. Paw — це повнофункціональний клієнт HTTP, який допомагає вам тестувати й описувати API, які ви розробляєте чи використовуєте для Mac. Для створення запитів, оцінки відповідей сервера, розробки клієнтського коду та експорту визначень API він надає чудовий рідний інтерфейс macOS. Кожна функція розроблена інтуїтивно і доступна за допомогою комбінації клавіш миші або клавіатури.

Завдяки Paw for Teams ви можете тримати всіх в курсі. Створіть команду, запросіть свою команду, і всі безперешкодно отримають зміни. Кожен може працювати зі своєю власною гілкою та об’єднувати зміни лише тоді, коли буде готовий; він майже такий же потужний, як Git, і безперебійний, як синхронізація в реальному часі.

Insomnia REST — це надійний клієнт REST API з відкритим кодом для надійного зберігання, організації та виконання запитів REST API. Завдяки підтримці керування файлами cookie, змінних середовища, генерації коду та аутентифікації клієнт Insomnia REST є чудовою альтернативою Postman для доставки запитів REST і GraphQL. Він сумісний з платформами Windows, Mac OS X і Linux. Insomnia також має зручний інтерфейс (рис. 4.5) із розширеними можливостями, такими як засоби безпеки, генерація коду та змінні середовища.

Insomnia автоматично отримає схему з кінцевої точки, якщо ви виберете GraphQL як тип тіла у спадному меню. Потім схема використовується для заповнення системи автозаповнення, що робить створення запиту надзвичайно швидким. Крім того, він безшумно впорядковує всі заголовки, гарантуючи, що все працює гладко.

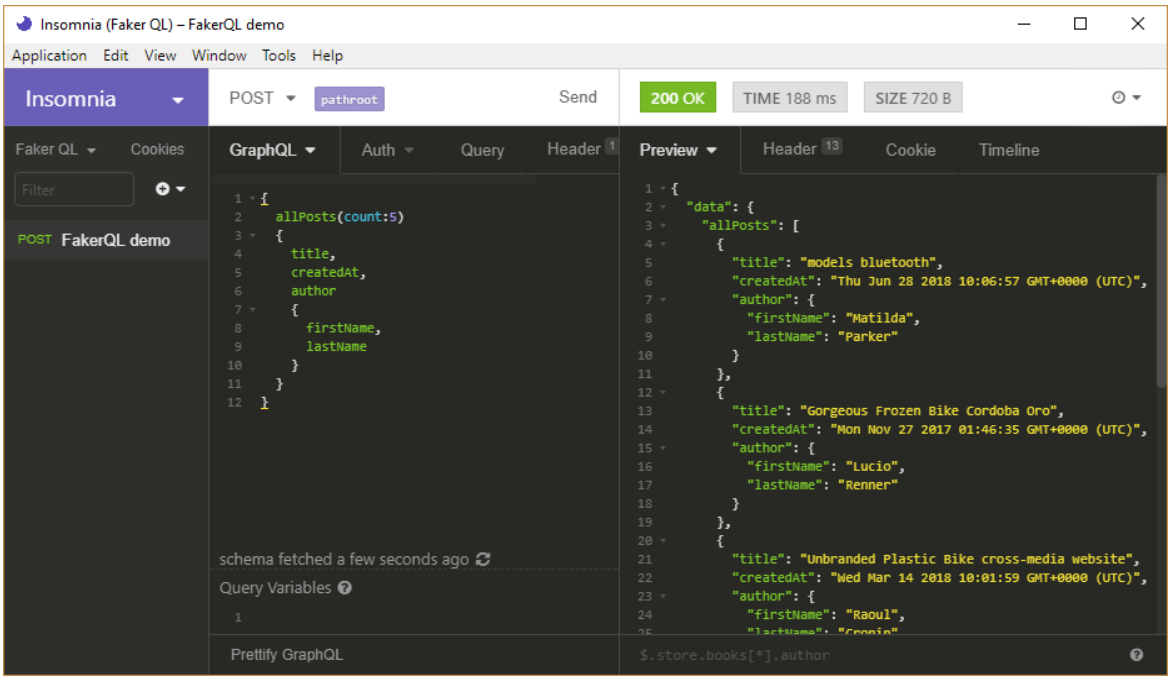

Рисунок 4.5 – Інтерфейс Insomnia

Ось деякі характеристики клієнта Insomnia REST [27]:

– кросплатформена підтримка;
– можливість виконувати запити REST, SOAP, GraphQL і GRPC;
– можливість зберігати, організовувати та виконувати запити REST API;
– вміння організувати запити в робочих просторах і групах;
– підтримка конструктора параметрів рядка запиту;
– можливість експорту та спільного використання робочих просторів;
– підтримка ланцюжкових запитів.

Деякі елементи Insomnia не підтримуються Postman. Нижче наведено деякі з цих характеристик:

– нові плагіни можна створювати за допомогою Insomnia;

– змінні середовища є одним із найкорисніших елементів Insomnia, оскільки вони можуть заощадити багато часу на введення тексту вручну;

– генерація фрагментів коду: Insomnia дозволяє створювати фрагменти коду різними мовами;

– Формат відповіді: Insomnia дозволяє переглядати відповіді, відмінні від JSON і XML, такі як сторінки HTML, зображення та навіть документи PDF.

4.4 Організація тестування програмного забезпечення

Як говорилось раніше, під час розробки програмного забезпечення невід'ємною частиною є тестування – а саме перевірка результату роботи програми. Робота тестувальника знайти баги – різницю між очікуваним та фактичним результатом.

Для досягнення своєї цілі, тестувальники діють за попередньо підготовленими тест-кейсами. Тест-кейси — це послідовність дій, що виконуються для забезпечення належної роботи певної функції або можливостей вашої програми. Тест-кейси — це набір тестових процедур, даних, передумов і постумов, створених для конкретного тестового сценарію з метою перевірки будь-якої вимоги [28,29]. Тест-кейс включає конкретні змінні або умови, за допомогою яких тестувальник може порівняти очікувані та фактичні результати, щоб визначити, чи функціонує програмний продукт відповідно до вимог замовника.

Далі наведено як можна написати базові тест-кейси у випадку мануального тестування:

1. Тест-кейс звичайної поведінки користувача. Для початку потрібно визначити що саме потрібно протестити, а саме поведінку звичайного користувача.

2. Перевірка даних. Ідентифікація тестових даних може зайняти багато часу і іноді може вимагати створення тестових даних заново. Причину цього необхідно задокументувати.

3. Опис кроків для відтворення. Потрібно розписати як саме буде відтворюватись сценарій перевірки з усією потрібною інформацією.

4. Перевірка тест-кейсу. Далі ми описуємо що саме ми очікуємо отримати в результаті наших дій.

Тест-кейси мають бути прості, виключати повторень, відповідати поведінці користувача, і т.д. Основні та важливі аспекти тест-кейсів описані на рис. 4.6

Рисунок 4.6 – Аспекти тест-кейсів

Опрацювавши вище описану інформацію щодо складання тест-кейсів, тестувальники тестують та валідують програми на наявність помилок

4.5 Підготовка та написання тестів для тестування модулю валідації подій

До тестування програмного забезпечення слід підходити продумано і з використанням різних визначених методів, щоб покрити програмне забезпечення всіма можливими тест-кейсами, випадками[30].

Є важливим підготовка тестів, що є обов’язковими для проходження smoke-test. Основні тести для тестування розробленого програмного забезпечення наведено в таблиці 4.1.

Таблиця 4.1 – Основні тести для тестування модулю валідації подій

№	Назва тесту	Кроки для відтворення	Результат, що очікується
1	2	3	4
1.	Завантаження веб сторінки	Перейти за посиланням http://127.0.0.1:1111/	Сторінка повинна бути завантажена. Код відповіді від сервера повинен бути 200.
2.	Запит наявних подій	Виконайте команду curl «http://127.0.0.1:1111/evns»	Сервер повинен повернути код 200. Сторінка повинна містити усі раніше надіслані події у форматі json
3.	Перевірка надсилання правильних даних	Виконайте команду curl, яка має в своєму складі подію у json форматі. Поля події мають співпадати з прописаними у схемі	Сервер повинен повернути код 200. Надіслана подія, яка пройшла перевірку має також повернути код 200 та відобразитись на сторінці
4.	Перевірка надсилання неправильних даних	Виконайте команду curl, яка має в своєму складі подію у json форматі. Поля події не мають співпадати з прописаними у схемі	Сервер повинен повернути код 200. Надіслана подія, яка не пройшла перевірку має також повернути код 400 та відобразитись на сторінці
5.	Перевірка надсилання подій без усіх потребуючих полів	Виконайте команду curl, яка має в своєму складі подію у json форматі. Поля події мають співпадати з прописаними у схемі, але без 1 або усіх потрібних полів	Сервер повинен повернути код 200. Надіслана подія , який не пройшов перевірку має також повернути код 400 та відобразитись на сторінці.

4.6 Відображення результатів валідації подій

Надсилання подій до модулю реалізовано за допомогою програмного додатку Postman, який в свою чергу є інструментом для тестування API[31]. Він

дозволяє надсилати події імітуючи додаток для надсилання подій, що зображено на рис. 4.7.

За допомогою юзер інтерфейсу будуть обиратись необхідні для надсилання та валідації подій згідно встановлених можливих значень.

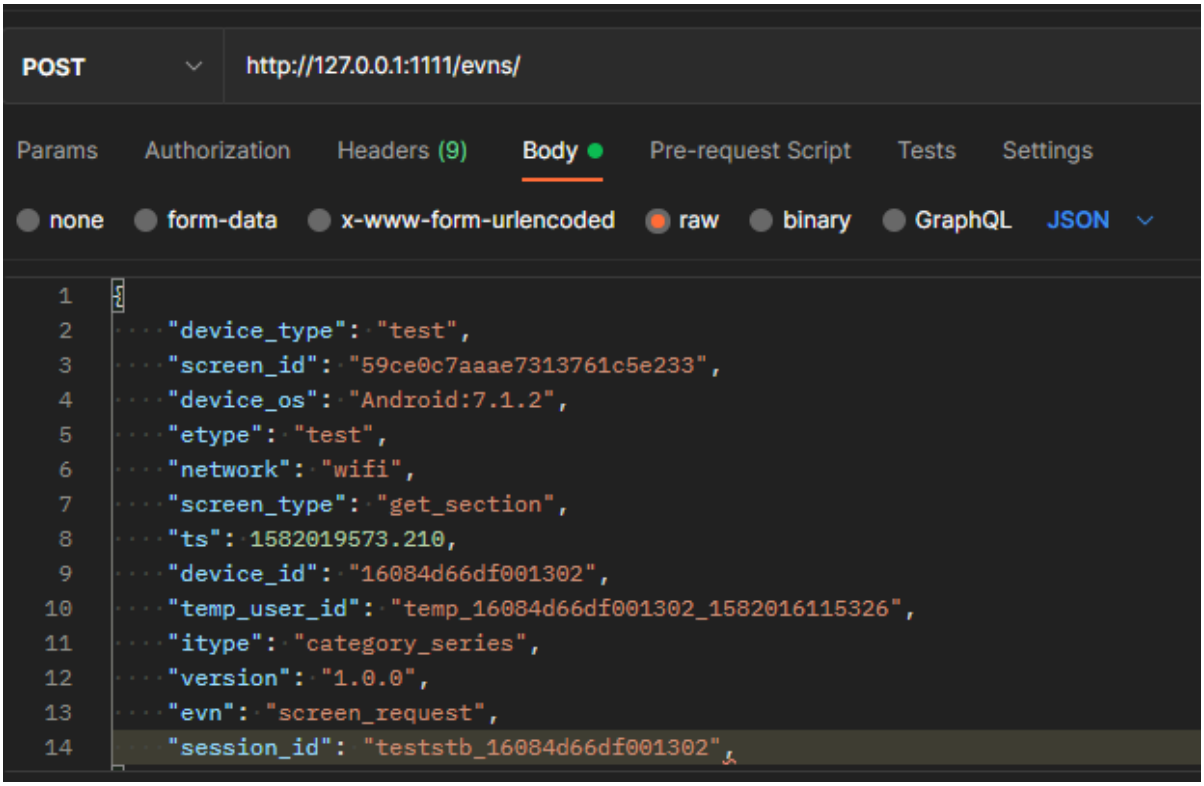

Рисунок 4.7 – Надсилання подій за допомогою Postman

Створимо запит для надсилання подій. Для цього використовуємо метод POST. В полі адреси потрібно вказати адресу піднятого серверу, на який надсилаються події. У нашому випадку, сервер піднятий на локальній машині, тому вказуємо http://127.0.0.1:1111/evns/. У поле body потрібно вказати нашу подію у json форматі, який імітує у нашому випадку дію додатку.

Після надсилання події, Postman надасть нам відповідь на надісланий запит з усією інформацією щодо події, а саме чи пройшов він перевірку, які поля не були використані, чи пройшли поля під валідацію щодо схеми і т.д. Детально вивід інформації показано на рис. 4.8.

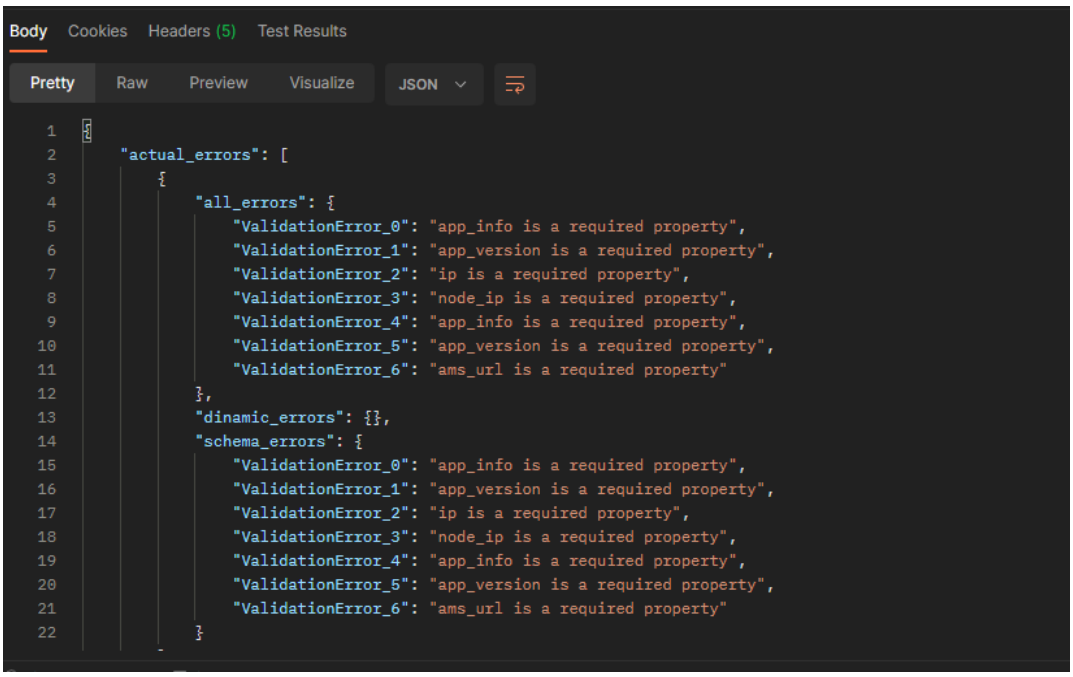

Рисунок 4.8 – Вивід інформації щодо надісланого події в Postman

Для відображення результатів роботи програми було імплементовано сервер на основі Flask. За його допомогою можна переглядати результат роботи модулю у будь-якому браузері, відображення результатів валідації подій наведено у додатку Б.

4.7 Порівняння розробленої системи з аналогами

Перевірка та порівняння роботи полягає в тому, що запропоноване програмне забезпечення для проведення валідації подій у форматі JSON, яка за рахунок використання мови програмування Python та безкоштовного веб-фреймворку Flask дозволяє, на відміну від існуючих інструментів, розширити функціональні можливості програмного забезпечення, а саме надати гнучкість налаштування для різного типу систем, суттєво спростити додавання подій для перевірки нового функціоналу, підтримувати перевірку результатів на різних платформах та надати детальну звітність щодо тестування. Для цього було проведено порівняння розробленого програмного забезпечення з аналогами. Результати порівняння наведено в таблиці 4.2.

Таблиця 4.2 – Результати порівняння розробленої системи з аналогами

Фунціонал	Selenium	WAPT	Postman	Ranorex	JSON-Event-Validator
Гнучкість налаштування	✓	X	✓	✓	✓
Простота налаштування	X	X	✓	X	✓
Кросс-платформеність	X	✓	X	✓	✓
Перевірка різних платформ	✓	✓	✓	X	✓
Валідація запитів згідно шаблону	✓	X	X	✓	✓
Робота з великим навантаженням	X	✓	X	X	✓
Звітність роботи	X	✓	✓	✓	✓

Як можна побачити, мета роботи була досягнута через отримання підтвердження шляхом порівняння функціоналу розробленої системи з аналогами. Переважна кількість аналогів має неповний функціонал для вирішення задачі валідації подій.

4.8 Висновки до розділу

У даному розділі було оглянуто процес відлагодження програмного забезпечення та види тестування. Тестування виконується для того, щоб зменшити кількість проблем, які ще не виявлено. Виконати бездоганне тестування неможливо, тому тестування передбачає досягнення такої якості продукту, яким буде задоволений звичайний користувач. Сам процес тестування програмного забезпечення можна поділити на статичне та динамічне тестування.

Були оглянуті популярні інструменти тестування, серед яких жоден не впорався з поставленою задачею для тестування програмного модулю у повній мірі. Оглянуті інструменти або підходять тільки для певних задач, або можуть допомогти тільки в тестуванні певної частини поставленої задачі.

Були підготовлені та написані тести для тестування модулю валідації подій у json форматі. Для надсилання подій було обрано Postman, як універсальний програмний засіб тестування API. Саме програмне забезпечення дозволяє імітувати додаток для надсилання подій. Результати відображаються як і у відповіді на подію в самому Postman, так і на піднятому сервері Flask за допомогою будь-якого браузеру.

5 ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит розроблених модулів тестування для валідації подій у json форматі

Як було зазначено раніше, сьогодні люди отримують та обробляють все більше інформації за допомогою використання різноманітних програм, перевірка правильності яких стає актуальною і важливою задачею.

Оскільки останнім часом в галузі тестування програмного забезпечення відбувся великий розвиток із новими трендами в послугах ІТ-галузі, то впровадження нових технологій безпосередньо вплинуло на оновлення у розробці, тестуванні та постачанні програмного забезпечення, висунувши на перше місце такі пріоритети, як оптимізація витрат та швидкість виконання. У зв’язку з цим створення нових інструментів для оптимізації тестування має зменшити час розробки та тестування програмного забезпечення.

Тому метою виконаної магістерської кваліфікаційної роботи була розробка програмного забезпечення тестування для валідації подій у json форматі.

Для цього було: проаналізовано існуючі засоби тестування, валідації даних та аналогів систем тестування; проведено огляд існуючих програмних засобів для реалізації поставленої задачі; розроблено програмне забезпечення тестування для валідації подій у json форматі; забезпечено відображення результатів валідації подій.

В результаті було створено модулю тестування для валідації подій у json форматі на різних системах та з можливістю подальшої модифікації компонентів системи з використанням сучасних технологій розробки. Зручність даного модулю валідації подій значно покращить та пришвидшить роботу тестувальників.

Для встановлення рівня комерційного потенціалу розробленого модуля тестування для валідації подій у json форматі було проведено його технологічний аудит, для чого було запрошено 3-х відомих експертів: кандидата технічних наук, доцента Марія БАРАБАН, кандидата технічних наук, доцента Володимир

ГАРМАШ і професор, в. о. завідувача кафедри автоматизації та інтелектуальних інформаційних технологій Олег БІСІКАЛО

Оцінювання комерційного потенціалу розробленого модуля тестування валідації подій у json форматі здійснювалося за критеріями, зведеними в таблиці 5.1.

Таблиця 5.1 – Критерії оцінювання рівня комерційного потенціалу будь-якої розробки та їх бальна оцінка (за шкалою: 0 - 1 - 2 - 3 - 4 балів)

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
1	2	3	4	5	6
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 5.1

1	2	3	4	5	6
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
Ринкові перспективи					
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Продовження таблиці 5.1

1	2	3	4	5	6
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка документів та отримання великої кількості документів на виробництво продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Запрошені експерти оцінили розроблений модуль тестування валідації подій у json форматі таким чином, який наведено в таблиці 5.2:

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали експерта		
	Марія БАРАБАН	Володимир ГАРМАШ	Олег БІСІКАЛО
	Бали, виставлені експертами:		
1	3	4	4
2	4	3	3
3	3	4	3
4	4	4	3
5	3	3	3
6	3	4	4
7	3	4	3
8	4	4	3
9	3	3	4
10	4	4	3
11	4	3	3
12	4	4	4
Сума балів	$СБ_1 = 42$	44	40

Середньоарифметична сума балів $\overline{СБ}$, що їх виставили експерти, становила:

$$\overline{СБ} = \frac{\sum_{1}^{3} Б_i}{3} = \frac{42 + 44 + 40}{3} = \frac{126}{3} = 42,0.$$

Загальний рівень комерційного потенціалу будь-якої розробки було встановлено за критеріями, наведеними в таблиці 5.3 [32].

Таблиця 5.3 – Рівні технічного та комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень технічного та комерційного потенціалу розробки
0– 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Керуючись рекомендаціями таблиці 5.3, можна зробити висновок, що розроблений модуль тестування для валідації подій у json форматі було оцінено експертами у 42 бали, що свідчить, що розробка має комерційний потенціал, який вважається «високим».

Це пояснюється тим, що розроблений модуль тестування валідації подій у json форматі буде одразу визначити шаблон відправлених подій для певної платформи, визначати мінімальні і максимально допустимі їх значення та обов'язковість інших полів, що значно спростить та пришвидшить роботу тестувальників, яким доводиться мати справу з перевіркою відправлених подій для аналітики.

5.2 Розрахунок витрат на розроблення модуля тестування для валідації подій у json форматі

Під час виконання роботи було зроблено такі витрати:

1. Основна заробітна плата виконавців $З_о$:

$$З_о = \frac{М}{Т_р} \cdot t \text{ грн}, \qquad (5.1)$$

де М –місячний посадовий оклад конкретного виконавця, грн;

У 2023 році величини посадових окладів науковців знаходиться в межах (6700…25700)грн/місяць;

$Т_р$ –число робочих днів в місяці; приймемо $Т_р = 22$ дні.

Розрахунки основної заробітної плати виконавців зведемо до таблиці 5.4:

Таблиця 5.4 – Розрахунок основної заробітної плати розробників

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день (або за годину), грн	Число днів роботи	Витрати на оплату праці, грн	Примітка
1.Науковий керівник магістерської квалі-фікаційної роботи	20835	947,05	20 годин	3156,83 ≈ ≈3157	6 годин у день
2. Студент-розробник-магістрант	6700	304,55	77 днів	23450,35≈ ≈23451	
3. Консультант з еко-номічної частини	19000	863,63	1,5 годин	215,90 ≈ 216	6 годин у день
4. Інші консультанти	18907	945,33	3 дні	2838	
Всього				29660	

2. Додаткова заробітна виконавців плата $З_д$ розраховується як (10…12)% від величини основної заробітної плати виконавців, тобто:

$$З_д = (0{,}1...0{,}12) \cdot З_о . \qquad (5.2)$$

Провівши розрахунки отримаємо:

$$З_д = 0{,}1055 \times 29660 = 3129{,}13 \approx 3130 \text{ грн.}$$

3. Нарахування на заробітну плату $Н_{зп}$ розраховуються за формулою:

$$Н_{зп} = (З_о + З_д) \cdot \frac{\beta}{100}, \quad (5.3)$$

де $З_о$ – основна заробітна плата виконавців, грн;

$З_д$ – додаткова заробітна плата виконавців, грн;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування; $\beta = 22\%$.

Провівши розрахунки отримаємо:

$$Н_{зп} = (29660+3130) \times 0,22 = 7213,80 \approx 7214 \text{ грн.}$$

4. Витрати на матеріали М розраховуються по кожному виду матеріалів:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \text{ грн}, \quad (5.4)$$

де $Н_i$ – витрати матеріалу *i*-го найменування, кг;

$Ц_i$ –вартість матеріалу *i*-го найменування, грн/кг.;

$К_i$ –коефіцієнт транспортних витрат, $К_i = (1,1…1,15)$;

$В_i$ –маса відходів матеріалу *i*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *i*-го найменування, грн/кг;

n –кількість видів матеріалів.

5. Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ грн}, \quad (5.5)$$

де $Н_i$ –кількість комплектуючих *i*-говиду, шт.;

$Ц_i$ –ціна комплектуючих *і*-го виду, грн;

$К_i$ –коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

n –кількість видів комплектуючих.

За аналогією з іншими розробками вартість всіх використаних матеріальних ресурсів становить приблизно 1200 грн.

6. Амортизацію А обладнання, комп'ютерів та приміщень А можна розрахувати за формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \text{ грн}, \qquad (5.6)$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань: $Н_а$ = (2...25)%;

Т –термін використання обладнання, приміщень тощо, місяці.

Зроблені розрахунки зведено до таблиці 5.5:

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
1. Персональні комп'ютери, принтери тощо	59700	25	3,2 (70%)	2786
2. Приміщення кафедри та факультету	46700	2,5	3,2 (80%)	249,07 ≈250
Всього				А = 3036

7. Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д}, \quad (5.7)$$

де В – вартість 1 кВт-год. електроенергії, в 2023 р. В ≈ 4,5 грн/кВт;

П –установлена потужність обладнання, кВт; П = 1,60 кВт;

Ф –фактична кількість годин роботи обладнання, годин. Приймемо, що Ф = 350 годин;

$К_п$– коефіцієнт використання потужності; $К_п$< 1 = 0,79.

$К_д$ – коефіцієнт корисної дії, $К_д$ = 0,69.

Тоді витрати на електроенергію складуть:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д} = \frac{4,5 \cdot 1,6 \cdot 350 \cdot 0,79}{0,69} = 2885,22 \approx 2886 \text{ грн.}$$

8. Інші витрати $В_{ін}$ можна прийняти як (50…300)% від основної заробітної плати виконавців, тобто:

$$В_{ін} = К_{ін} \times З_о = (0,5..3,0) \times З_о. \quad (5.8)$$

Так як було визначено, що $К_{ін}$ = 0,5. Тоді:

$$В_{ін} = 0,5 \times 29660 = 14830 \text{грн.}$$

9. Сума всіх попередніх статей витрат дає нам витрати на виконання цього етапу роботи безпосередньо розробником-магістрантом – В.

$$В = 29660 + 3130 + 7214 + 1200 + 3036 + 2886 + 14830 = 61956 \text{ грн.}$$

10. Розрахунок загальних витрат на розробку та остаточне доопрацювання виконаної роботи здійснюється за формулою:

$$ЗВ = \frac{В}{\beta}, \quad (5.9)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Оскільки розробка ще потребує незначного доопрацювання, то можна прийняти, що $\beta \approx 0{,}85$.

$$ЗВ = \frac{61956}{0{,}85} = 72889{,}41\,грн$$

Тобто прогнозовані загальні витрати на розроблений модуль тестування валідації подій у json форматі становлять приблизно 73 тисячі грн

5.3 Розрахунок економічного ефекту від можливої комерціалізації розробки програмного забезпечення

Аналіз ринку показує, що розроблений модуль тестування валідації подій у json форматі буде мати значний попит на ринку у компаній, які займаються розробкою кроссплатформених застосунків для перегляду відео і e-commerce застосунків; у медіа та розважальних закладах; у соціальних мережах та спільнотах, в освіті, медицині та фітнесі, в мобільних додатках, технічній підтримці тощо.

Тобто, якщо розробка буде впроваджена з 1 січня 2024 року, то її результати будуть виявлятися протягом 2024-го, 2025-го та 2026-го років. Прогноз зростання попиту на нашу розробку складає по роках:

а) 2023 рік – 1 шт., тобто це наша розробка;

б) 2024 р. – + 3 шт. до базового року (тобто 3 клієнтів);

в) 2025 р. – + 5 шт. до базового року (тобто 5 клієнтів);

г) 2026 р. – + 7шт. до базового року (тобто 7 клієнтів).

За висновками експертів потенційна ціна реалізації розробки на сучасному ринку становить приблизно $5 тисяч, або приблизно 180 тисяч грн, в той час, як приблизно подібні (а не ідентичні) розробки, що якось виконують зазначені вище функції, вартують на ринку до 100 тисяч грн.

Можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від виведення розробки на ринок, становитиме:

$$\Delta П_i = \sum_{1}^{n} (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}), \qquad (5.10)$$

де $\Delta Ц_o$ – покращення основного якісного показника від впровадження результатів розробки у цьому році.

$$\Delta Ц_o = 180 - 100 = + 80 \text{ тисяч грн;}$$

де N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 1 шт.;

ΔN – покращення основного кількісного показника від впровадження результатів розробки.

Таке покращення становитиме по роках, відповідно: у 2024 році – + 3 шт., у 2025 році + 5 шт., та у 2026 році + 7шт.;

$Ц_o$ – основний якісний показник (тобто ціна), який визначає обсяг діяльності у році після впровадження результатів розробки, грн; $Ц_o$= 180 тисяч грн;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для даного випадку n = 3;

λ – коефіцієнт, який враховує сплату податку на додану вартість; $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати $\rho = (0{,}2...0{,}5)$; візьмемо $\rho = 0{,}5$;

υ – ставка податку на прибуток. У 2023-26 роках υ=18% (наше припущення).

Тоді можливе зростання чистого прибутку $\Delta П_1$ для потенційного інвестора протягом першого року від можливого впровадження розробки (2024 р.) становитиме:

$$\Delta П_1 = [80 \cdot 1 + 180 \cdot 3] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 212 \text{ тис. грн.}$$

Можливе зростання чистого прибутку $\Delta П_2$ для потенційного інвестора від можливого впровадження розробки протягом другого (2025) року складе:

$$\Delta П_2 = [80 \cdot 1 + 180 \cdot 5] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 335 \text{ тис. грн.}$$

Можливе зростання чистого прибутку $\Delta П_3$ для потенційного інвестора від можливого впровадження розробки протягом третього (2026) року складе:

$$\Delta П_3 = [80 \cdot 1 + 180 \cdot 7] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 458 \text{ тис. грн.}$$

Приведена вартість зростання всіх чистих прибутків від можливого впровадження і комерціалізації розробки становитиме:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої роботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Так як, т = 3 роки;

τ – ставка дисконтування. Приймемо τ = 0,10 (10%);

t – період часу від моменту початку роботи над створенням програмного засобу (модуля) тестування валідації подій у json форматі до моменту отримання можливих чистих прибутків потенційним інвестором.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації розробки, складе:

$$ПП = \frac{212}{(1+0,1)^2} + \frac{335}{(1+0,1)^3} + \frac{458}{(1+0,1)^4} \approx 175 + 252 + 313 = 740 \text{ тисяч грн.}$$

Теперішня вартість інвестицій PV, що повинні бути вкладені у реалізацію розробки: $PV = (1,0…5,0) \times В_{заг}$.

$$PV = (1,0…5,0) \times 73 = 2,0 \times 73 = 146 \text{тисяч грн.}$$

Абсолютний ефект від можливих вкладених в реалізацію розробки інвестицій $Е_{абс}$.

$$Е_{абс} = ПП – PV, \qquad (5.12)$$

де ПП – приведена вартість збільшення всіх чистих прибутків для потенційного інвестора від можливої комерціалізації розробки, грн;

PV – теперішня вартість інвестицій PV = 146 тисяч грн.

Абсолютний ефект від можливого впровадження розробки складе:

$$Е_{абс} = 740 – 145 = 594 \text{ тисяч грн.}$$

Далі розрахуємо внутрішню дохідність $E_в$ вкладених інвестицій:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.13)$$

де $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс}$ = 594 тис. грн;

PV –теперішня вартість початкових інвестицій PV = 146 тис. грн;

$T_ж$ – життєвий цикл розробки, роки.

$T_ж$ = 4 років (2023-й, 2024-й, 2025-й, 2026-й роки)

Тоді отримаємо:

$$E_в = \sqrt[4]{1 + \frac{594}{146}} - 1 = \sqrt[4]{1 + 4,0685} - 1 = \sqrt[4]{5,0685} - 1 = 1,50 - 1 = 0,50 = 50,0\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією розробки.

Мінімальна дохідність або мінімальна ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \quad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022-2023 роках в Україні d = (0,10...0,12);

f– показник, що характеризує ризикованість вкладень;

f= (0,1...0,50). Приймемо f = 0,30.

$$\tau_{мін} = 0,12 + 0,30 = 0,42 \text{або } \tau_{мін} = 42\%.$$

Оскільки величина $E_в$ = 50,0%> $\tau_{мін}$ = 42%, то потенційний інвестор у принципі може бути зацікавлений у фінансуванні та комерціалізації розробки.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленого модуля тестування валідації подій у json форматі.

Термін окупності $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}. \quad (5.15)$$

Термін окупності $T_{ок}$ коштів становитиме:

$$T_{ок} = \frac{1}{0,50} = 2,00 \text{ років} < 3 \text{ років},$$

що свідчить про потенційну доцільність комерціалізації розробленого модуля тестування валідації подій у json форматі.

Далі проведено моделювання залежності величини внутрішньої дохідності вкладених потенційних інвестицій в комерціалізацію розробленого модуля тестування валідації подій у json форматі від рівня інфляції в країні.

Якщо рівень інфляції в країні зросте до 20%, то:

$$ПП = \frac{212}{(1+0,2)^2} + \frac{335}{(1+0,2)^3} + \frac{458}{(1+0,2)^4} \approx 147 + 194 + 221 = 562 \text{ тисяч грн.}$$

Абсолютний ефект від можливого впровадження розробки складе:

$$E_{абс} = 562 - 146 = 416 \text{ тисяч грн.}$$

Далі розрахуємо внутрішню дохідність $E_в$ вкладених інвестицій:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.13)$$

де $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс}$ = 416 тис. грн;

PV –теперішня вартість початкових інвестицій PV = 146 тис. грн;

$T_ж$ – життєвий цикл розробки, роки.

$$E_в = \sqrt[4]{1+\frac{416}{146}} - 1 = \sqrt[4]{1+2,8493} - 1 = \sqrt[4]{3,8493} - 1 = 1,40 - 1 = 0,40 = 40,0\%.$$

Оскільки величина $E_в$ = 40,0%≈ $\tau_{мін}$ = 42%, то потенційний інвестор у принципі може бути зацікавлений у фінансуванні та комерціалізації розробки. Якщо рівень інфляції в країні зросте до 30%, то:

$$ПП = \frac{212}{(1+0,3)^2} + \frac{335}{(1+0,3)^3} + \frac{458}{(1+0,3)^4} \approx 97 + 153 + 161 = 411 \text{ тисяч грн.}$$

Абсолютний ефект від можливого впровадження розробки складе:

$$E_{абс} = 411 - 146 = 265 \text{ тисяч грн.}$$

Далі розрахуємо внутрішню дохідність $E_в$ вкладених інвестицій:

$$E_в = \sqrt[T_ж]{1+\frac{E_{абс}}{PV}} - 1,$$

де $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс}$ = 265 тис. грн;

PV –теперішня вартість початкових інвестицій PV = 146 тис. грн;

$T_ж$ – життєвий цикл розробки, роки.

$$E_в = \sqrt[4]{1+\frac{265}{146}} - 1 = \sqrt[4]{1+1,8151} - 1 = \sqrt[4]{2,8151} - 1 = 1,295 - 1 = 0,295 = 29,5\%.$$

Оскільки величина $E_в = 29,5\% < \tau_{мін} = 42\%$, то потенційний інвестор у принципі може бути не зацікавлений у комерціалізації розробки.

Зроблені розрахунки у вигляді графіків наведено на рис. 5.1.

Аналіз діаграм на рис 5.1 показує, що при рівні інфляції в 10% величина внутрішньої дохідності інвестицій становить $E_в = 50,0\%$, що більше порогового значення $\tau_{мін} = 42\%$ і тому комерціалізація розробки може бути доцільною.

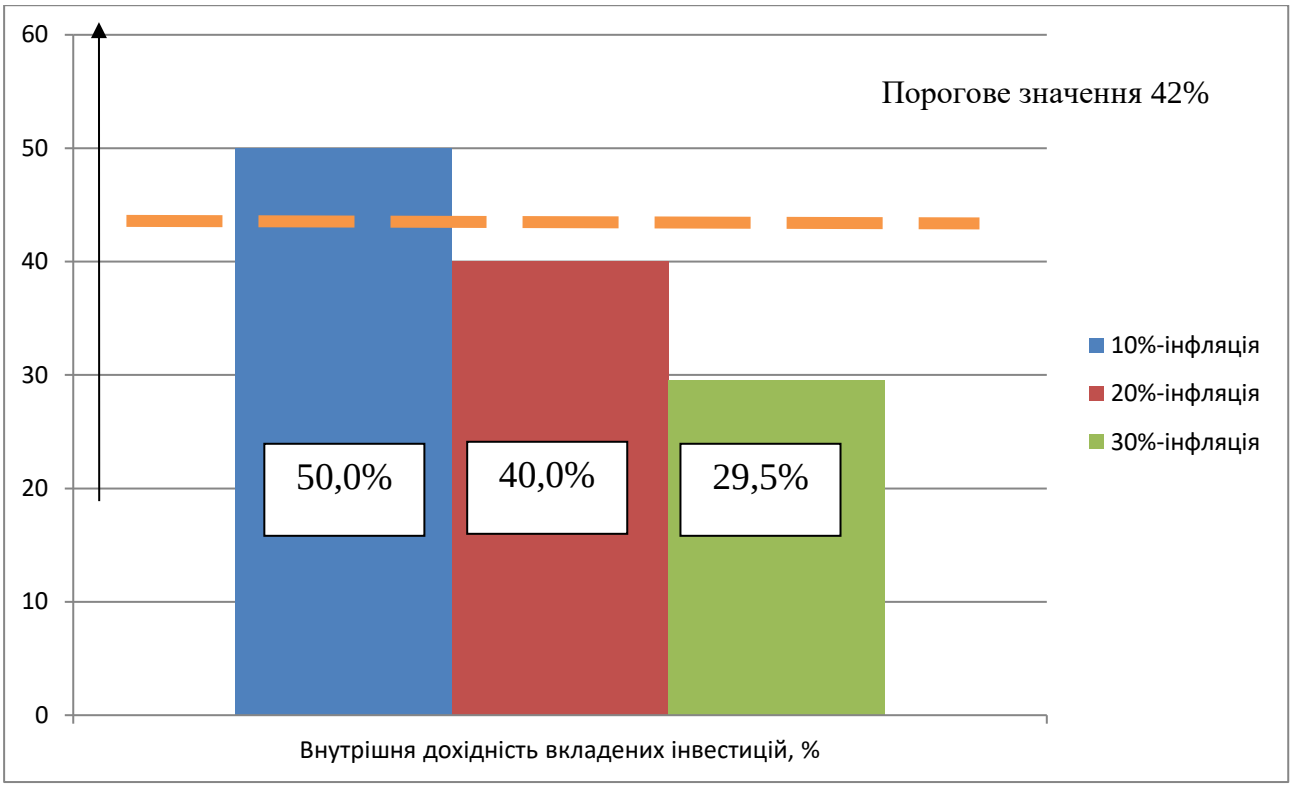

Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій від рівня інфляції в країні (10%, 20% і 30%)

При рівні інфляції в 20% величина внутрішньої дохідності інвестицій становить $E_в = 40,0\%$, що практично дорівнює пороговому значенню $\tau_{мін} = 42\%$ і тому комерціалізація розробки може бути доцільною. При рівні інфляції в 30% величина внутрішньої дохідності інвестицій, вкладених в комерціалізацію розробки, становить $E_в = 29,5\%$, що менше порогового значення $\tau_{мін} = 42\%$, і тому комерціалізація розробки потенційним інвестором може бути під питанням. Але остаточне рішення з цього питання потребує проведення додаткових розрахунків

(можливо – зниження рівня ризикованості вкладень, збільшення попиту на розробку, збільшення ціни реалізації розробки тощо).

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю 5.6.

Таблиця 5.6 – Результати виконаної економічної частини

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1. Витрати на розробку	Не більше 80 тисяч грн	73 тис. грн.	Досягнуто
2. Абсолютний ефект від впровадження розробки, тисяч грн	Не менше ≈ 500 тисяч грн	594 тисяч грн (при 10%-інфляції)	Виконано
3. Внутрішня дохідність інвестицій, %	не менше 42%	50,0% (при 10%-інфляції)	Досягнуто
4. Термін окупності інвестицій, роки	до 3-ти років	2,00 років	Виконано

Таким чином, основні техніко-економічні показники розробленого модуля тестування валідації подій у json форматі, визначені у технічному завданні, виконані.

ВИСНОВКИ

Всі задачі, поставлені перед магістерською дипломною роботою виконано в повному обсязі.

У роботі було проведено аналіз та розробку програмного модулю для валідації подій у json форматі. Основна мета полягала розширенні функціональних можливостей програмного забезпечення для проведення валідації подій у форматі JSON. За допомогою базового юзер інтерфейсу обираються необхідні для валідації події та шаблони їх можливих значень. Результати валідації подій, а саме чи підпадають вони під вказані параметри можна переглядати у браузері за допомогою піднятого серверу або безпосередньо у інструменті надсилання подій у вигляді відповіді на надіслану подію. Мета роботи була досягнута через отримання підтвердження шляхом порівняння функціоналу розробленої системи з аналогами.

Були оглянуті сучасні інструменти для тестування, такі як Selenium, WAPT, Postman та Ranorex. Основними їх перевагами є швидкість та простота у використані. Для розв’язання поставленої задачі їх функціонала недостатньо. У даній роботі на основі знань про принцип роботи цих інструментів було розроблено свою систему валідації подій на основі шаблонізації json schema та вебфреймворку Flask.

За основу мови програмування було взято мову Python. Дана мова програмування найкраще підходить для розробки даного програмного забезпечення, адже швидкість та простота розробки цієї програми надає можливість у майбутньому пришвидшені та покращенні даної програми

Було розроблено програмне забезпечення для валідації подій у json форматі, що на відміну від існуючих інструментів тестування дозволяє прискорити та спростити процес валідації подій. За допомогою шаблонів обираються необхідні для надсилання та валідації подій згідно встановлених можливих значень.

Для даного модулю валідації подій було складено тест-кейси та проведено тестування. Було перевірено основний функціонал модулю на наявність помилок. Було забезпечено відображення результатів валідації подій. Також, було проведено порівняння функціональних можливостей з аналогами. Для відображення результатів роботи програми було імплементовано сервер на основі Flask. За його допомогою можна переглядати результат роботи модулю у будь-якому браузері.

Мета дослідження – розширити функціональні можливості програмно забезпечення для валідації подій у json форматі за рахунок того, що на відміну від існуючих інструментів тестування дозволить прискорити та спростити процес валідації подій, буде гнучким для різного типу систем та суттєво спростить додавання подій для перевірки нового функціоналу. Також немає обмежень для запуску програмного продукту на різних операційних системах, в т.ч. Android та iOS.

Система має актуальну архітектуру та використовує сучасні технології розробок та конструювання та з мінімальними змінами у коді може бути імплементована у інші проекти чи використана як окремий продукт.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Latest Trends in Software Testing 2020 – 2021 .URL: https://www.testingxperts.com/knowledge-center/latest-trends/ (дата звернення 10.10.2023).

2. An introduction to the Flask Python web app framework. OpenSource.com : . URL: https://opensource.com/article/18/4/flask (дата звернення 10.10.2023).

3. Selenium with Python . URL: https://selenium-python.readthedocs.io/ (дата звернення 10.10.2023) .

4. Абдуллаєв О. А., Богач І. В. Валідація івентів у форматі JSON з використанням Python та фреймворка Flask // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» : збірник матеріалів. – Вінниця: ВНТУ, 2023. - Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/view/18645(дата звернення 10.10.2023).

5. Whittaker J. A. Exploratory software testing: Tips, tricks, tours, and techniques to guide test design. Upper Saddle River, NJ : Addison Wesley Professional, 2009.

6. Test the Performance of Web Applications Under Load, .URL:: https://www.loadtestingtool.com/ (дата звернення 10.10.2023).

7. Spillner A. Software Testing Foundations / Spillner A., Linz T., Schaefer H. – [4th Edition] – Rocky Nook, 2014. – 305 p. – ISBN 978-1-937538-42-2

8. Singla S. Selenium keyword driven automation testing framework / S. Singla, H. Kaur // International journal of Advanced Research in Computer Science and Software Engineering. – 2014. – Vol. 4, Issue 6. – P. 125-129. .URL:: http://www.ijarcsse.com/docs/papers/Volume_4/6_June2014/V4I6-0157.pdf

9. Richardson A. Selenium Simplified / Richardson A. Compendium Developments, 2010. 75 p. ISBN 978-0-9567332-1-4

10. Al Sweigart Automate the Boring Stuff with Python: монографія, 2015, 1196 с.

11. Welcome to Python.org, . URL: https://www.python.org/ (дата звернення 10.10.2023).

12. Full Stack Python, . URL: https://www.fullstackpython.com/flask.html (дата звернення 10.10.2023).

13. What are the pros and cons of Python? . URL: https://www.quora.com/What-are-the-pros-and-cons-of-Python (дата звернення 10.10.2023)

14. Python Pros and Cons. . URL: https://www.netguru.com/blog/python-pros-and-cons (дата звернення 10.10.2023).

15. Introduction to Flask, . URL: https://pymbook.readthedocs.io/en/latest/flask.html (дата звернення 10.10.2023).

16. Flask web development one drop at a time. . URL:https://flask.palletsprojects.com/en/1.1.x/ (дата звернення 10.10.2023).

17. Introduction to JSON. . . URL: https://javaee.github.io/tutorial/jsonp001.html (дата звернення 10.10.2023).

18. What is GitHub? . URL: https://www.youtube.com/watch?v=w3jLJU7DT5E/ (дата звернення 10.10.2023).

19. PyCharm. The Python IDE for Professional Developers . URL: https://www.jetbrains.com/pycharm/ (дата звернення 10.10.2023).

20. What is Software Testing? Definition, Basics & Types. . URL: https://www.guru99.com/software-testing-introduction-importance.html (дата звернення 10.10.2023).

21. Software testing. . URL: https://www.ibm.com/topics/software-testing (дата звернення 10.10.2023).

22. Eric Matthes Python Crash Course: монографія, 2019, 544 с.

23. James Whittaker. «Exploratory software testing»: монографія Addison-Wesley, 2010, 224 с.

24. Ron Patton — «Software Testing»: монографія, 2005, 408 с.

25. JMeter Guide Overview. . URL: https://www.loadview-testing.com/jmeter-load-performance-testing-tutorial/ (дата звернення 10.10.2023).

26. Silk Test Functional test automation for web, mobile, rich-client, and enterprise applications. . URL: https://www.microfocus.com/en-us/products/silk-test/overview (дата звернення 10.10.2023)

27. Insomnia vs. Postman vs. Paw. . URL: https://rapidapi.com/blog/insomnia-vs-postman-vs-paw/ (дата звернення 10.10.2023).

28. How to write Test Cases in Software Testing? . URL: https://www.browserstack.com/guide/how-to-write-test-cases (дата звернення 10.10.2023).

29. What Is Test Case? How To Write Test Cases In Software Testing? . URL: https://testsigma.com/guides/what-is-test-case/ (дата звернення 10.10.2023).

30. Test Suites and Their Test Cases: The Hierarchy Explained . URL: https://www.testim.io/blog/test-suite/ (дата звернення 10.10.2023).

31. API testing . URL: https://www.postman.com/api-platform/api-testing/ (дата звернення 10.10.2023).

32. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. / Укладачі В.О. Козловський, О.Й. Лесько, В.В.Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

33. Проектування та розробка модуля тестування для валідації івентів у json форматі/ І. В. Богач, О.А. Абдуллаєв. / матер. LI Науково-технічній конференції ВНТУ, 30-31 травня, 2022. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2022/paper/view/15463. – (дата звернення 01.06.2022).

Додатки

Додаток А (обов’язковий)

Технічне завдання

ЗАТВЕРДЖУЮ

Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО

_____« 12 » _жовтня_ 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

«Проектування та розробка модулів тестування для валідації подій у json форматі»

08-31.МКР.001.02.000 ТЗ

Керівник роботи
к.т.н., доц. каф. АІІТ
Ілона БОГАЧ
«_ _» __ _____2023 р.

Виконавець:
ст. гр. 1ІСТ-22М
Олексій АБДУЛЛАЄВ
«_ _» _ _____2023 р.

Вінниця ВНТУ 2023

1. Назва та галузь застосування

Магістерська кваліфікаційна робота: «Проектування та розробка модулів тестування для валідації подій у json форматі». Галузь застосування – інформаційні технології.

2. Підстава для розробки

Розробку системи здійснювати на підставі наказу по університету№ 247 від 18 09 2023 та завдання до магістерської кваліфікаційної роботи, складеного та затвердженого кафедрою «Автоматизації та інтелектуальних інформаційних технологій»

3. Мета та призначення розробки

Метою роботи є розробка програмного забезпечення для валідації подій у json форматі.

4. Джерела розробки

1. Spillner A. Software testing foundations: A study guide for the certified tester exam : foundation level, ISTQB compliant. 3rd ed. Santa Barbara, CA : Rocky Nook, 2011. 284 p.

2. Whittaker J. A. Exploratory software testing: Tips, tricks, tours, and techniques to guide test design. Upper Saddle River, NJ : Addison Wesley Professional, 2009.

3. Black R. Foundations of Software Testing ISTQB Certification. 3rd ed. Cengage Learning EMEA, 2012. 242 p.

5. Показники призначення

Основні технічні вимоги та мінімальні системні вимоги до програми:

- ОС: Windows XP
- Процесор: Core 2 Duo
- Оперативна пам’ять: 2 GB ОП

– Відеокарта: Intel HD Graphics 4000
– Місце на диску: 500 MB доступного місця

Результати роботи програми:

– виведення списку успішних подій;
– виведення списку проблемних подій;
– конфігурація фільтрів для виведення подій;
– конфігурація шаблонів подій;
– виведення подій згідно фільтрації;
– виведення статистики подій;

6. Економічні показники

– витрати на розробку – не більше 80 тис. грн;
– абсолютний ефект від впровадження розробки – не менше 500 тис. грн;
– внутрішня дохідність інвестицій – не менше 42%;
– термін окупності – не більше 3 років.

7. Стадії розробки

1. Розділ 1 «Аналіз існуючих засобів тестування та аналогів систем тестування» має бути виконаний до 02.10.2023.
2. Розділ 2 «Огляд існуючих програмних засобів для реалізації поставленої задачі» має бути виконаний до 16.10.2023.
3. Розділ 3 «Розробка програмного забезпечення для валідації подій у json форматі» має бути виконаний до 08.11.2023.
4. Розділ 4 «Тестування програмного забезпечення» має бути виконаний до 14.11.2023.
5. Економічний розділ має бути виконаний до 17.11.2023.

8. Порядок контролю та приймання

1. Рубіжний контроль провести до 01.11.2023.

2. Попередній захист магістерської кваліфікаційної роботи провести до 21.11.2023.

3. Захист магістерської кваліфікаційної роботи провести в період з 11.12.2023 до 22.12.2023.

Додаток Б (обов’язковий)

Графічна частина

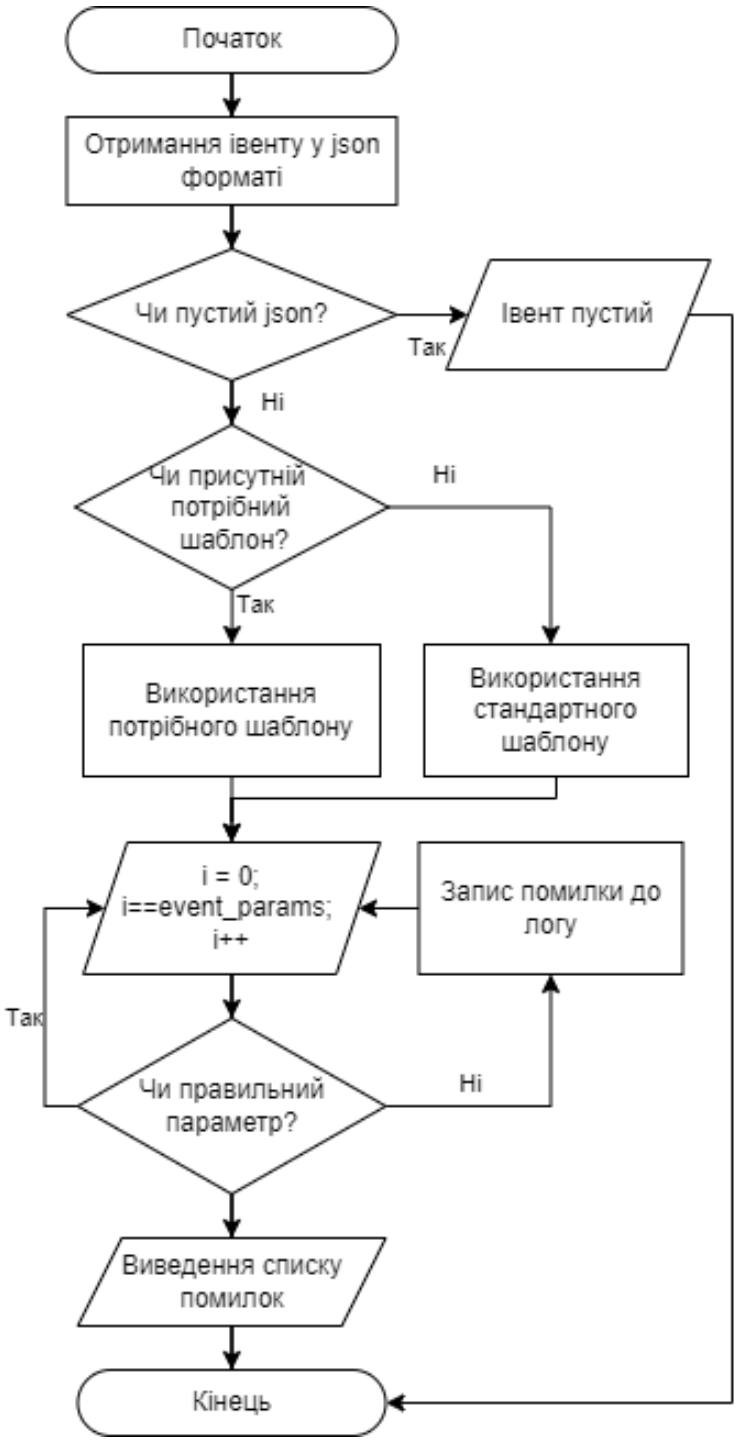

Рисунок Б.1 – UML діаграма модулю валідації подій

Продовження додатка Б

Рисунок Б.2 – UML діаграма модулю перевірки події

Продовження додатка Б

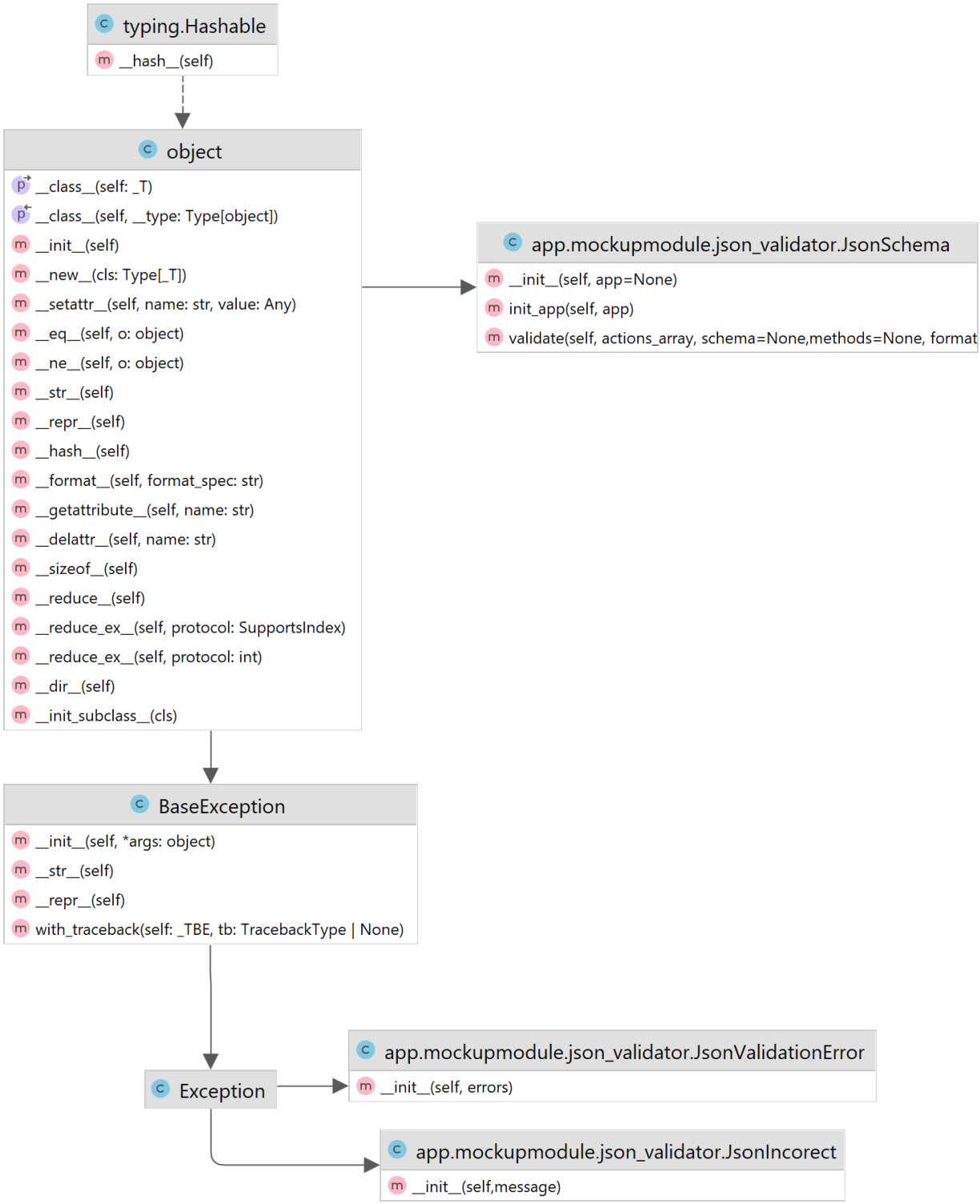

Рисунок Б.3 – Діаграма класів модулю валідації подій

Продовження додатка Б

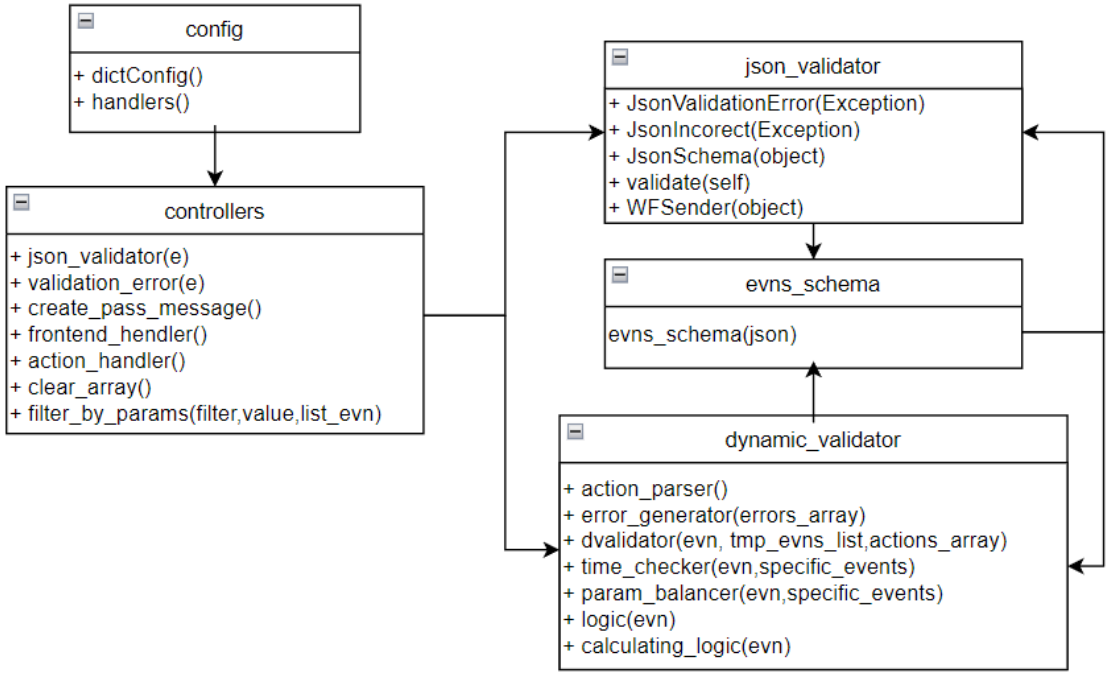

Рисунок Б.4 – UML-class diagram модулю валідації подій

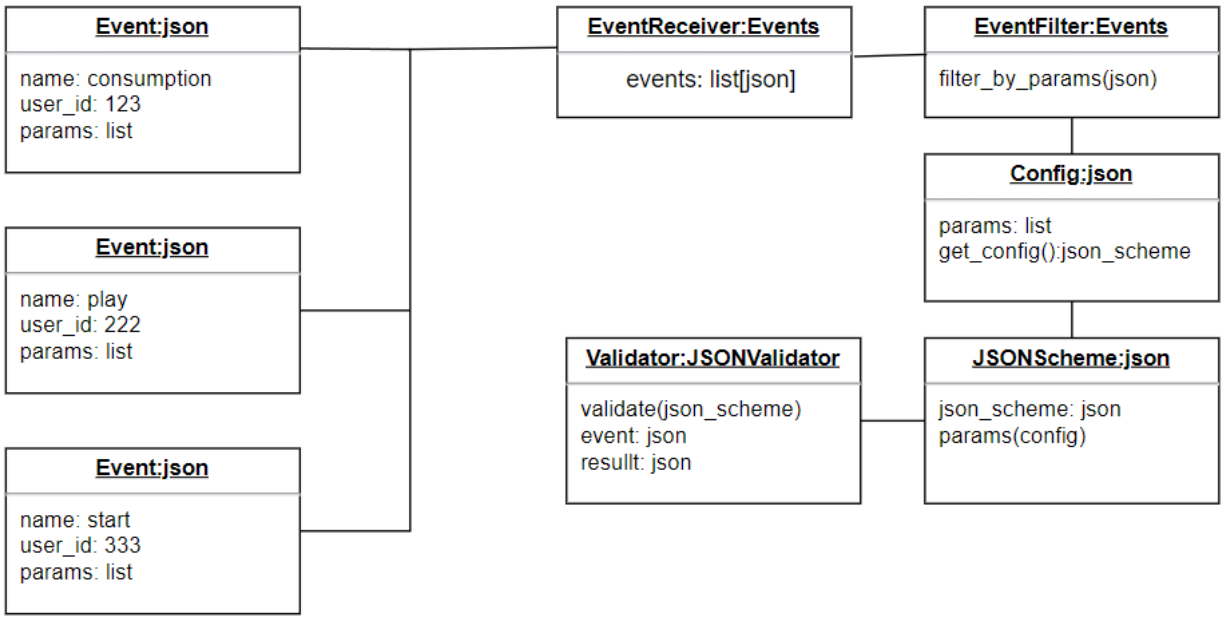

Рисунок Б.5 – UML-object diagram модулю валідації подій

Продовження додатка Б

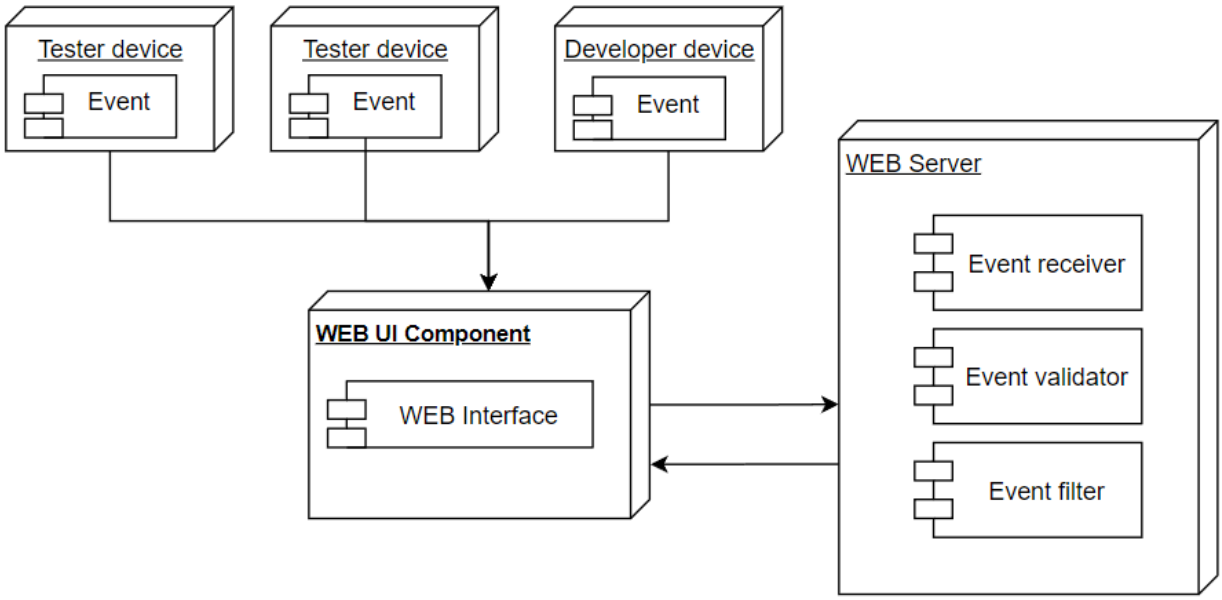

Рисунок Б.6 – UML-deployment diagram модулю валідації подій

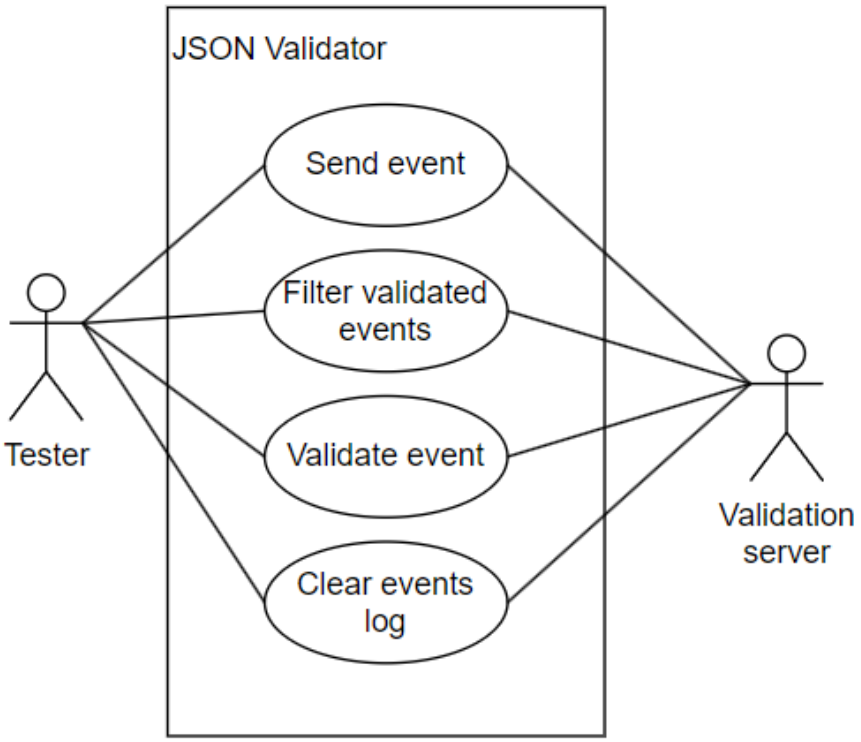

Рисунок Б.7 – UML-use case diagram модулю валідації подій

Продовження додатка Б

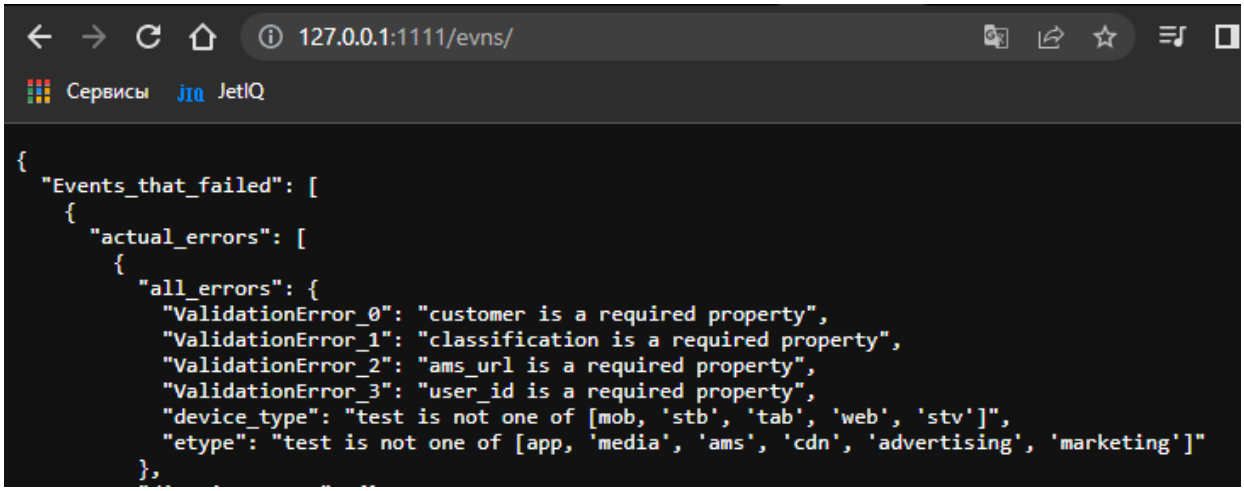

Рисунок Б.8 – Відображення результатів валідації подій

Додаток В (обов’язковий)

Лістинг програмного забезпечення

controllers.py:

```
from flask import Flask, Blueprint, jsonify, request,render_template
from app.mockupmodule.json_validator import JsonSchema, JsonValidationError,JsonIncorect
import datetime,time
import json
import logging
from flask_cors import CORS


mockup = Blueprint('mockup', __name__,url_prefix ='/evns')
schema = JsonSchema(Flask(__name__))

evns_pass = []
evns_fail = []
actions_array = []
unixtime=time.time()

@mockup.errorhandler(JsonIncorect)
def json_validator(e):
errors = str(e)
return jsonify({"error":errors})

@mockup.errorhandler(JsonValidationError)
def validation_error(e):
input_json_with_err = request.get_json()
try:
evn_name = input_json_with_err["evn"]
except:
evn_name = None
err =str(e)
errors=eval(err)
response = jsonify({
"head": {
"title": "event",
"itype": "event",
"event_name": evn_name,
"ts": unixtime,
},
"actual_errors":errors,
"event":request.get_json(),
"status": 400
})

errors= json.dumps({"actual_errors":errors})
errors = json.loads(errors)
input_json_with_err.update(errors)
evns_fail.insert(0,(input_json_with_err))
logging.critical(json.dumps(input_json_with_err))
return response,400


#Data input point
@mockup.route('/',methods=['POST'])
@schema.validate(actions_array)
def create_pass_message():
input_json = request.get_json()
evns_pass.insert(0,input_json)
return jsonify({
"head": {
"title": "event",
"itype": "event",
"event": input_json["evn"],
"ts": unixtime,
},
"info":{
"n_events": 1
},
"status": "null"})
```

```
@mockup.route('/',methods=['GET'])
def frontend_hendler():
if request.args.get('filter') != None :
filter_param = request.args.get('filter')
value_param = request.args.get('value')
tmp_list = evns_fail+evns_pass
return jsonify({"filtered":filter_by_params(filter_param,value_param,tmp_list)})
if request.args.get('evn') == 'pass':
return jsonify(evns_pass)
if request.args.get('evn') == 'failed':
return jsonify(evns_fail)
return jsonify({
"Events_that_pass":evns_pass,
"Events_that_failed":evns_fail
})

####
@mockup.route('/actions/',methods=['GET','POST'])
def action_handler():
if request.method == 'POST':
try:
if request.get_json()['action'] and request.get_json()['ts']:
actions_array.append(request.get_json())
return jsonify({
"action":actions_array[0].get('action'),
"ts": actions_array[0].get('ts'),
}),201
except:
return jsonify({
"Error":"Required fields are not specified. check if 'action' and 'ts' are present"
}),400
return jsonify(actions_array)


@mockup.route('/delete/', methods=['GET'])
def clear_array():
if request.args.get('param') == 'pass':
evns_pass.clear()
comment = 'array with pass evetns was deleted'
logging.info(comment)
return comment
if request.args.get('param') == 'failed':
evns_fail.clear()
comment = 'array with failed evetns was deleted'
logging.info(comment)
return comment
elif request.args.get('param') == 'pass,failed' or request.args.get('param') == 'failed,pass':
 evns_pass.clear()
 evns_fail.clear()
 comment = 'events list was deleted'
 logging.info(comment)
 return comment
comment = "Please add query '?param=pass' or '?param=failed' for delete. Also for delete all can use sepatator ',' - example '?param=pass,failed'"
return comment


def filter_by_params(filter,value,list_evn):
elements = []
for element in list_evn:
validator = element.get(filter, None)
if element and validator is not None:
if element[filter] == value:
elements.append(element)
return elements

json_validator:
from functools import wraps, partial, update_wrapper
from flask import current_app, request, jsonify
from jsonschema.validators import validator_for
from jsonschema import Draft7Validator
from app.mockupmodule.dinamic_validator import dvalidator
from app.mockupmodule.schemas.evns_schema import evns_schema
from app.mockupmodule.schemas.mob_schema import mob_schema
from app.mockupmodule.schemas.web_schema import web_schema
import requests
import json
import logging
```

```
# tmp events erray with 500 last evns
tmp_evns_list = []

#https://python-jsonschema.readthedocs.io/en/stable/validate/

class WFSender(object):
def __init__(self,event):
try:
event = json.dumps(event)
headers = {
'Connection': 'keep-alive',
'Content-type': 'application/json',
'Origin': 'https://test.com',
}
response = requests.post('https://test.com/event/evn/', headers=headers, data=event)
logging.info(f"responce code:{response.status_code},body:{json.loads(response.content)}")

except Exception as err:
logging.critical(f'Error:{err}')


class JsonValidationError(Exception):
def __init__(self, errors):
self.errors = errors


class JsonIncorect(Exception):
def __init__(self,message):
self.message = message


class JsonSchema(object):
def __init__(self, app=None):
self.app = app
self.config = {}
self.validator_cls = None
if app is not None:
self.init_app(app)

def init_app(self, app):
self.app = app
self.config = app.config.copy()
self.config.setdefault('JSON_SCHEMA_METHODS', ['POST'])
self.config.setdefault('JSON_SCHEMA_FORMAT_CHECKER', Draft7Validator)
self.config.setdefault('JSON_SCHEMA_DEF', evns_schema)


def validate(self, actions_array, schema=None,methods=None, format_checker=None):
def wrapper(fn):
@wraps(fn)
def decorated(actions_array, schema=None, methods=None, format_checker=None, *args, **kwargs):
input_json = request.get_json()
try:
if input_json["app_info"] == "ANDROID" or input_json["app_info"] == "IOS" or input_json["app_info"] == "STB" or input_json["app_info"] ==
"tvOS":
schema=mob_schema
elif input_json["app_info"] == "WEB" or input_json["app_info"] == "SmartTV":
schema=web_schema
else:
schema=evns_schema
except:
#in case when json not have app_info
schema=evns_schema
logging.info(schema['definitions']['title'])
validator_kwargs = {
'schema': schema if schema else self.config.get('JSON_SCHEMA_DEF'),
'format_checker': format_checker if format_checker else self.config.get('JSON_SCHEMA_FORMAT_CHECKER')
}
# use default methods if not supplied as arguments to decorator
if not methods:
methods = self.config.get('JSON_SCHEMA_METHODS')
# tmp events erray with 500 last evns
if len(tmp_evns_list) >= 499:
logging.warning(len(tmp_evns_list))
del tmp_evns_list[-250:]
tmp_error_string = ""

tmp_evns_list.insert(0,input_json)
# check jsonschema
```

```
if request.method in methods:
validator_cls = self.validator_cls if self.validator_cls else validator_for(schema)
validator = validator_cls(**validator_kwargs)
if not input_json or input_json is None:
logging.critical(json.dumps(input_json))
raise JsonIncorect('Json is null')
# handle errors json and dinamic
else:
# dinamic
try:
derrors = dvalidator(input_json, tmp_evns_list,actions_array)
derrors = json.dumps(derrors)
derrors=eval(derrors)
except Exception as err:
 derrors ={}
 logging.error(err)
# json schema
errors = sorted(validator.iter_errors(input_json), key=lambda e: e.path)
iter_i = 0 # don`t fine solution
for error in errors:
message = error.message
message = message.replace("'", "", 4)
#logging.warning(f'message:{message}')
if list(error.path):
actual_errors = list(error.path)[0]
actual_errors = "\"" + actual_errors +"\":\""+ message + "\","
actual_errors = actual_errors.replace("'", '\\"', 50)
actual_errors = actual_errors.replace("'::", '\\"::\\"', 50)
actual_errors = actual_errors.replace('>"', '\\"', 50)
#logging.warning(actual_errors)
else:
actual_errors = "\"ValidationError_"+str(iter_i) +"\":\"" + message + "\","
iter_i+=1
tmp_error_string += actual_errors
tmp_error_string = "{"+tmp_error_string[:-1] +"}"
tmp_error_string=eval(tmp_error_string)

full_error = dict(derrors, **tmp_error_string)
if full_error:
tmp_error_string = json.dumps([{"all_errors":full_error,"schema_errors":tmp_error_string,"dinamic_errors":derrors}])
tmp_error_string = json.loads(tmp_error_string)
raise JsonValidationError(tmp_error_string)
return fn(*args, **kwargs)
# the wrapper() func ctx has access to format_checker & methods, but the decorator
# won't, so we use partial, where those args are passed in
pfunc = partial(decorated, actions_array=actions_array, format_checker=format_checker, methods=methods, )
# this is needed because partial() doesn't add in __name__ attribute to the created
# partial function, which Flask requires
update_wrapper(pfunc, decorated)

return pfunc

return wrapper
controllers.py:
from flask import Flask, Blueprint, jsonify, current_app, request
from apscheduler.schedulers.background import BackgroundScheduler
from selenium import webdriver
import requests
import json
import logging
import time
import os



cachewarmup = Blueprint('cachewarmup', __name__,url_prefix ='/evns')
app = current_app._get_current_object()

def url_validator(url):
logging.info("fn: url_validator")
try:
result = urlparse(url)
return all([result.scheme, result.netloc, result.path])
except:
return False


def auth_to_test():
headers = app.config['HEADERS']
```

```
data = {
"login":app.config['TEST_GUI_LOGIN'],
"password":app.config['TEST_GUI_PASSWORD']
}
data = json.dumps(data)
url = app.config['TEST_GUI_LOGIN_PATH']
login = requests.post(url, headers=headers,data=data)

logging.info(f'Getting DS statuscode: {login.status_code}')
return login.cookies


chatids_list = []
dashboardids_list = []
execute_time = ['-1']

@cachewarmup.route('/brow')
def test_browser():
import subprocess
subprocess.call('google-chrome-stable --version', shell=True)
subprocess.call('google-chrome-stable --headless --no-sandbox', shell=True)
subprocess.call('chromedriver', shell=True)
print('----------')
subprocess.call(app.config['WEB_DRIVER_PATH'], shell=True)
return 'please see logs'

@cachewarmup.route('/startwarmup/')
def parser_datas():
logging.info(app.config['TEST_GUI_DOMAIN'])
page =1
logging.info("init webdriver parameter")
options = webdriver.ChromeOptions()
options.add_argument("--start-maximized") #open Browser in maximized mode
options.add_argument("--no-sandbox") #bypass OS security model
options.add_argument("--disable-dev-shm-usage") #overcome limited resource problems
options.add_experimental_option("excludeSwitches", ["enable-automation"])
options.add_experimental_option("useAutomationExtension", False)
options.binary_location = app.config['BROWSER_LOCATION']
is_url = url_validator(app.config['WEB_DRIVER_PATH'])
if app.config['BROWSER_HEADLESS']:
options.add_argument('--headless')
if is_url:
driver = webdriver.Remote(command_executor=app.config['WEB_DRIVER_PATH'],desired_capabilities=options.to_capabilities())
logging.warning('using remove web-driver')
elif not is_url:
driver = webdriver.Chrome(chrome_options=options,executable_path=app.config['WEB_DRIVER_PATH'])
logging.warning('using local web-driver')
driver.get(app.config['TEST_GUI_DOMAIN'])
name = driver.find_element_by_css_selector('input[formcontrolname=login]')
password = driver.find_element_by_css_selector('input[type=password]')
name.send_keys(app.config['TEST_GUI_LOGIN'])
password.send_keys(app.config['TEST_GUI_PASSWORD'])
driver.find_element_by_css_selector('button[type=submit]').click()


headers = app.config['HEADERS']
logging.info("get headers and cookies")
cookies = auth_to_test()
logging.info("request for get DS")
url = app.config[test]
dashboards_metadata = requests.get(url, cookies=cookies)
dashboards_metadata_to_json = json.loads(dashboards_metadata.text)
logging.warning(f'Getting DS statuscode: {dashboards_metadata.status_code}')
for dashboards_id in dashboards_metadata_to_json:
try:
logging.info(f"dashboards_id_list:{dashboards_id['id']}")
driver.execute_script("window.open('');")
driver.switch_to.window(driver.window_handles[page])
full_url = app.config['TEST_GUI_DOMAIN'] + '/#/dashboard/{0}/'.format(dashboards_id['id'])
logging.warning(full_url)
driver.get(full_url)
page +=1
if page == app.config['PAGE'] :
driver.switch_to.window(driver.window_handles[app.config['PAGE']-(app.config['PAGE']-1)])
time.sleep(app.config['SLEEP_TIME']+1)

driver.close()
driver.switch_to.window(driver.window_handles[page-2])
page-=1
```

```
except:
driver.close()
driver.quit()

try:
warm_report = app.config['WARMUP_REPORT']
except:
warm_report = False
logging.info(f'warm-up reports:{warm_report}')
if warm_report:
for dashboards_id in dashboards_metadata_to_json:
#try:
#logging.info('dashboards_id:')
#logging.info(dashboards_id['id'])
url_d = app.config[test_config'] + '{0}/'.format(str(dashboards_id['id']))
dashboards_d = requests.get(url_d, cookies=cookies)
chartsLayout = dashboards_d.json()['chartsLayout']

for dashboard_id in chartsLayout:
logging.info('----------dashboards_d----------')
driver.execute_script("window.open('');")
driver.switch_to.window(driver.window_handles[page])
full_url = app.config['TEST_GUI_DOMAIN'] +'/#/dashboard/{0}/{1}'.format(dashboards_id['id'],dashboard_id['dashboardId'])
logging.warning(full_url)
driver.get(full_url)
page +=1

if page == app.config['PAGE'] :
driver.switch_to.window(driver.window_handles[app.config['PAGE']-(app.config['PAGE']-1)])
time.sleep(app.config['SLEEP_TIME'])

driver.close()
driver.switch_to.window(driver.window_handles[page-2])
page-=1

#except:
#logging.critical("some BIG Issueee!!!!!!")
#break

driver.close()
driver.quit()
counter =0
#tmp solution for remove Z prosses
try:
logging.info('re-view Z pros')
#z_pros = True
for counter in range(15):
try:
z_pros = os.waitpid(-1, os.WNOHANG)
logging.info(f'Z pros present: {z_pros}')
except:
logging.warning(f'issue with remove zpros: {z_pros}')
except ChildProcessError:
logging.warning(f'not have Z pros: {z_pros}')
return 'succeed'


@cachewarmup.route('/execution/')
def change_execution():
if len(execute_time) > 1:
logging.info(f'old execute time removed {execute_time[0]}')
try:
del execute_time[1]
except Exception as err:
logging.warning(f'empty {err}')
if request.args.get('time'):
execute_time.insert(0,request.args.get('time'))
logging.info(f'set new execute time {execute_time[0]}')
job_manage(action='del')
job_manage()
return execute_time[0]
else:
return execute_time[0]


def job_manage(action='skip'):
if action == 'add':
if execute_time[0] != '-1':
print('run')
```

```
sched.add_job(parser_datas,trigger='cron', minute=f'{execute_time[0]}', id='warm_up')
logging.info('add warm_up job')
else:
print('not run')
logging.info('not run job')
if action == 'del':
try:
sched.remove_job('warm_up')
logging.info('delete warm_up jub')
except Exception as err:
logging.warning(err)
else:
logging.info('not run job')

sched = BackgroundScheduler(daemon=True)
job_manage(action='add')
sched.start()

evns_schema:
evns_schema = {
"definitions": {
"title": "The ROOT PARAMETERS",
"parameters": {
"$id": "#/definitions/parameters",
"type": "object",
"additionalProperties" : False,
"title": "The Parameters Schema",
"properties": {
###################################
"evn":{
"$id": "#/definitions/parameters/properties/evn",
"type": "string",
"enum":
["player_ready","start","chu","stop","media_error","app_open","app_closed","app_ready","app_error","auth_sent","sync_sent","screen_request","play_request","seek","media_buffering_start","media_buffering_completed","media_is_playing","media_pause","media_resume","app_paused","auth_ok","auth_error","sync_ok","sync_error","screen_ready","play_response","play_error","try_play_premium","meta_request","meta_response","meta_error","sid_create","ams_request","ams_response","item_related","purchase","recommendation","media_bitrate_switch","media_audio_switch","media_subtitle_switch","media_screen_switch","app_hidden","app_shown","screen_response","popup_shown","popup_clicked","banner_click","fungus_login_attempt","fungus_login_success","user_interaction","share_item","external_application_opened","products_updated","favorite_item","unfavorite_item","like_item","unlike_item","product_offer_played_on","search","notification_received","notification_action","in_box_open","upgrade","first_run","media_cast","abort","set_pn_permission","fungus_login_failure","fungus_logout_attempt","fungus_logout_success","fungus_logout_failure","ad_show","ad_skip","ad_ignored","ad_clicked","ad_request","ad_switch","dl_start","dl_pause","dl_error","dl_completed","sys_status"],
"title": "The EVN parameter",
"default": "",
"examples": [
"chu"
],
"pattern": "^(.*)$"
},
###################################
"etype": {
"$id": "#/definitions/parameters/properties/etype",
"type": "string",
"enum": ["app","media","ams","cdn","advertising","marketing"],
"title": "The Etype Schema",
"default": "",
"examples": [
"media"
],
"pattern": "^(.*)$"
},
###################################
"ts": {
"$id": "#/definitions/parameters/properties/ts",
"type": "number",
"title": "The Ts Schema",
"default": 0.0,
"examples": [
1582639989.325
],
"pattern": "^(.*)$"
},
###################################
"user_id": {
"$id": "#/definitions/parameters/properties/user_id",
"type": "string",
"minLength": 24,
"maxLength": 24,
"title": "The User_id Schema",
```

```
"default": "",
"examples": [
"5d1f8352436fa61f282e488d"
],
"pattern": "[0-9a-z]"
},
##################################
"user_id_MOB": {
"$id": "#/definitions/parameters/properties/user_id_MOB",
"type": ["string", "null"],
"minLength": 24,
"maxLength": 24,
"title": "The User_id Schema",
"default": "",
"examples": [
"5d1f8352436fa61f282e488d"
],
"pattern": "[0-9a-z]"
},
##################################
"device_id": {
"$id": "#/definitions/parameters/properties/device_id",
"type": "string",
"minLength": 10,
"maxLength": 80,
"title": "The Device_id Schema",
"default": "",
"examples": [
"8d24c56613ffa4da"
],
"pattern": "^(.*)$"
},
##################################
"device_type": {
"$id": "#/definitions/parameters/properties/device_type",
"type": "string",
"enum": ["mob","stb","tab","web","stv"],
"title": "The Device_type Schema",
"default": "",
"examples": [
"stb"
],
"pattern": "^(.*)$"
},
##################################
"device_info": {
"$id": "#/definitions/parameters/properties/device_info",
"type": "string",
"minLength": 15,
"maxLength": 160,
"title": "The Device_info Schema",
"default": "",
"examples": [
"com.playbox.dev, q201 + Amlogic q201, AppVersion: 2.8.76.27, 7.1.2, Language: tha"
],
"pattern": "^(.*)$"
},
##################################
"device_os": {
"$id": "#/definitions/parameters/properties/device_os",
"type": "string",
"minLength": 5,
"maxLength": 45,
"title": "The Device_os Schema",
"default": "",
"examples": [
"Android:7.1.2"
],
"pattern": "^(.*)$"
},
##################################
"session_id": {
"$id": "#/definitions/parameters/properties/session_id",
"type": "string",
"minLength": 5,
"maxLength": 146,
"title": "The Session_id Schema",
"default": "",
"examples": [
```

```
"ira_8d24c56613ffa4da"
],
"pattern": "^(.*)$"
},
################################
"session_id_MOB": {
"$id": "#/definitions/parameters/properties/session_id_MOB",
"type": ["string", "null"],
"minLength": 5,
"maxLength": 146,
"title": "The Session_id Schema",
"default": "",
"examples": [
"ira_8d24c56613ffa4da"
],
"pattern": "^(.*)$"
},
################################
"network": {
"$id": "#/definitions/parameters/properties/network",
"type": "string",
"enum":["wifi","3g","4g","fbb","ethernet","unknown"],
"title": "The Network Schema",
"default": "",
"examples": [
"ethernet"
],
"pattern": "^(.*)$"
},
################################
"version": {
"$id": "#/definitions/parameters/properties/version",
"type": "string",
"minLength": 1,
"maxLength": 10,
"title": "The Version(SDK version) Schema",
"default": "",
"examples": [
"1.0.0"
],
"pattern": "^(.*)$"
},
################################
"old_version": {
"$id": "#/definitions/parameters/properties/old_version",
"type": "string",
"minLength": 1,
"maxLength": 25,
"title": "The Version Schema",
"default": "",
"examples": [
"ANDROID 2.8.72.1 (1.0.0)"
],
"pattern": "^(.*)$"
},
################################
"ip": {
"$id": "#/definitions/parameters/properties/ip",
"type": "string",
"minLength": 13,
"maxLength": 13,
"title": "The Version Schema",
"default": "",
"examples": [
"185.133.64.44"
],
"pattern": "^(.*)$"
},
################################
"screen_mode": {
"$id": "#/definitions/parameters/properties/screen_mode",
"type": "string",
"enum": ["full","portrait","landscape"],
"title": "The Screen_mode Schema",
"default": "",
"examples": [
"full"
],
"pattern": "^(.*)$"
```

```
},
##################################
"language": {
"$id": "#/definitions/parameters/properties/language",
"type": "string",
"enum": ["eng","tha","chi","zho","ukr","rus","fra","jpn"],
"title": "The Language Schema",
"default": "",
"examples": [
"tha"
],
"pattern": "^(.*)$"
},
##################################
"tag": {
"$id": "#/definitions/parameters/properties/tag",
"type": "string",
"minLength": 3,
"maxLength": 20,
"title": "The Language Schema",
"default": "",
"examples": [
"ugc"
],
"pattern": "^(.*)$"
},
##################################
"app_info": {
"$id": "#/definitions/parameters/properties/app_info",
"type": "string",
"enum":["WEB","IOS","ANDROID","STB","SmartTV","tvOS","tvOS-Demo"],
"title": "The App_info Schema",
"default": "",
"examples": [
"STB"
],
"pattern": "^(.*)$"
},
##################################
"app_version": {
"$id": "#/definitions/parameters/properties/app_version",
"type": "string",
"minLength": 3,
"maxLength": 35,
"title": "The App_version Schema",
"default": "",
"examples": [
"STB 2.8.76.27"
],
"pattern": "^(.*)$"
},

##################################
"customer": {
"$id": "#/definitions/parameters/properties/customer",
"type": "string",
"minLength": 2,
"maxLength": 35,
"const": "ais",
"title": "The Customer Schema",
"default": "",
"examples": [
"ais"
],
"pattern": "^(.*)$"
},
##################################
"error_description": {
"$id": "#/definitions/parameters/properties/error_description",
"type": "string",
"minLength": 4,
"maxLength": 345,
"title": "The Error_description Schema",
"default": "",
"examples": [
""
],
"pattern": "^(.*)$"
},
```

```
#################################
"error_description_MOB": {
"$id": "#/definitions/parameters/properties/error_description_MOB",
"type": ["string", "null"],
"minLength": 4,
"maxLength": 345,
"title": "The error_description_MOB Schema",
"default": "",
"examples": [
""
],
"pattern": "^(.*)$"
},
```

Додаток Г

Довідка про впровадження результатів роботи

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "ПРАГМАТТЕК" / LLC "PragmatTech"
21036, Україна, м. Вінниця, вул. І. Бойка, буд.15, офіс 85; ЄДРПОУ 44420499;
pragmat.tech.ua@gmail.com

Затверджую

Директор ТОВ «ПРАГМАТТЕК»

_____ О.О. Бабієць

«01» 12 2023 р.

АКТ

Впровадження результатів магістерської дипломної роботи

Абдуллаєва Олексія Алліжановича на тему:

«Проектування та розробка модулів тестування для валідації івентів у json форматі»

Комісія ТОВ «ПРАГМАТТЕК» у складі залучених фахівців з розробки програмного безпечення Побережника Станіслава Васильовича та Пшеничної Галини Володимирівни розглянула магістерську дипломну роботу Абдуллаєва Олексія Алліжановича та встановила, що використання запропонованого програмного забезпечення, дозволяє забезпечити функціонал, необхідний для надання послуг аналітики для системи стрімінгу, розповсюдження та управління відео контентом для ряду клієнтів, задовольняє вимогам по ресурсоємкості, швидкодії та стійкості до навантаження.

Актуальність та практична цінність роботи Абдуллаєва О.А. не викликає сумнівів. Розроблене програмне забезпечення за рахунок гнучкого налаштування, дозволяє забезпечити перевірку аналітичної інформації від клієнтів на різних платформах.

Беручи до уваги вищевказані переваги, результати роботи було впроваджено до системи обслуговування бізнесів, що отримують від компанії послуги з розповсюдження відео контенту та стрімінгу, включаючи систему аналітики.

Члени комісії:

_____ С.В. Побережник

_____ Г.В. Пшенична

Додаток Д (обов'язковий)

Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Проектування та розробка модулів тестування для валідації подій у json форматі

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ: кафедра Автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних технологій та автоматизації
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 93.0% Схожість 7.0%

Аналіз звіту подібності (відмітити потрібне):

☑ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку (підпис) Роман МАСЛІЙ (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи (підпис) Олексій АБДУЛЛАЄВ (прізвище, ініціали)

Керівник роботи (підпис) Ілона БОГАЧ (прізвище, ініціали)