МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії (ЗФН)
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО
“___”_____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИЦІ ОСВІТНЬОГО СТУПЕНЯ

“МАГІСТРА”

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ “ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ”

Тема: “Система крос-платформного та крос-браузерного тестування”

Виконавиця: Броніцька Анастасія Сергіївна

Керівник: к.т.н., доцент Моденов Юрій Борисович

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки комп'ютерної та програмної інженерії (ЗФН)

Кафедра комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Аліна САВЧЕНКО
«___»___2021 р.

ЗАВДАННЯ

на виконання дипломної роботи студентки

Броніцької Анастасії Сергіївни

1. Тема роботи: «Система крос-платформного та крос-браузерного тестування» затверджена наказом ректора від 12.10.2021 р. №2229/ст.

2. Термін виконання роботи: з 12.10.202 по 31.12.2021

3. Вихідні дані до роботи: прототип системи крос-платформного та крос-браузерного тестування, послідовність дій для створення кращої платформи для тестування, процес організації функціонального тестування.

4. Зміст пояснювальної записки: існуючі системи крос-платформного та крос-браузерного тестування, створення прототипу користувацького інтерфейсу, проведення UI тестування.

5. Перелік обов'язкового графічного матеріалу: слайди, презентація.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання етапів	Примітка
1.	Отримання завдання на дипломну роботу та побудова плану виконання роботи.	12.10.2021 – 15.10.2021	
2.	Аналіз існуючих рішень	16.10.2021 – 21.10.2021	
3.	Опрацювання літературних джерел	22.10.2021 – 26.10.2021	
4.	Аналіз процесу організації функціонального тестування	27.10.2021 – 03.11.2021	
5.	Аналіз ціноутворення у платформах тестування	04.11.2021 – 09.11.2021	
6.	Створення прототипу користувацького інтерфейсу	10.11.2021 – 17.11.2021	
7.	Проведення UI тестування	18.11.2021 – 29.11.2021	
8.	Оформлення пояснювальної записки дипломної роботи	30.11.2021 – 08.12.2021	
9.	Оформлення презентації для захисту дипломної роботи	09.12.2021 – 12.12.2021	
10.	Підготовка до захисту дипломної роботи	13.12.2021 – 20.12.2021	

7. Дата видачі завдання: 12.10.2021р.

Керівник дипломної роботи _____ Юрій МОДЕНОВ

(підпис керівника) (ПІБ)

Завдання прийняла до виконання _____ Анастасія БРОНІЦЬКА

(підпис випускниці) (ПІБ)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Система крос-платформного та крос-браузерного тестування” складається зі вступу, семи розділів, висновків, списку використаних джерел і містить 70 сторінок тексту, 19 рисунків та 3 таблиці. Список використаних джерел містить 20 найменувань.

Метою дипломної роботи є створення прототипу платформи для крос-платформного та крос-браузерного тестування.

Предметом дослідження є платформа для забезпечення функціональної якості користувацьких інтерфейсів.

Об’єктом дослідження є забезпечення функціональної якості користувацьких інтерфейсів.

Ключові слова: ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, КРОС-БРАУЗЕРНЕ ТЕСТУВАННЯ, КРОС-ПЛАТФОРМНЕ ТЕСТУВАННЯ, ПРЕДМЕТНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ, ПЛАТФОРМА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

DDD – Domain Driven Design – Предметно-орієнтоване проєктування

IoT – Internet of Things – Інтернет речей

CI/CD – Continuous integration / Continuous delivery – Неперервна інтеграція/ Неперервна доставка

ОС – Операційна система

TCO – Total cost of ownership / Сукупна вартість володіння

ВСТУП

Метою дипломної роботи є створення прототипу платформи для крос-платформного та крос-браузерного тестування.

Більшість ПЗ, так чи інакше, націлені на взаємодію з людиною через візуальний інтерфейс, відповідно є попит на засоби, що будуть гарантувати якість розроблених застосунків.

Кількість можливих платформ для взаємодії зростає постійно, ще 10 років тому люди працювали лише через комп'ютери, тепер ж є IoT засоби, мобільні телефони, планшети, годинники, тощо, тому кількість технічної роботи й кількості необхідного обладнання для тестування значно зросла.

Тож роботи по забезпеченню якості застосунків є вкрай актуальними з наукового боку, та, особливо, з практичного.

Робота представляє модель платформи, що буде надавати потужності користувачам для крос-платформного та крос-браузерного тестування, що дозволить їм самостійно не розбиратись у технічних деталях інтеграції з різними системами, такими як ОС, браузери, віртуалізація, й купувати обладнання, а одразу зосередитись на розробці свого продукту.

У ході дипломної роботи були вибрані технології для реалізації системи.

За допомогою вибраних технологій розроблено масштабуєму, й просту систему крос-браузерного та крос-платформного тестування.

Наведено приклад функціонування платформи на двох різних ОС (Linux та MacOS) й у двох різних браузерах (Chrome та Firefox) на прикладі популярного українського сайту новин UKR.NET.

Створено прототип користувацького інтерфейсу системи для виконання основних операцій у системі.

Дипломна робота може зацікавити всіх, хто пов'язує свою діяльність з розробкою якісного ПЗ.

Дипломна робота допоможе:

- зрозуміти основні проблеми й виклики для забезпечення крос-платформного та крос-браузерного тестування ПЗ;
- зрозуміти ситуацію на ринку, які продукти вже є в наявності, їх переваги та недоліки;
- ознайомитись зі створеною моделлю платформи;
- побачити як модель працює на прикладі тестування відомого сайту новин UKR.NET;
- обрати стратегію тестування інтерфейсу для компаній, що займаються розробкою власного ПЗ.

Дипломний проект представляє наукову цінність:

- в сфері розвитку тестування якості ПЗ на IoT засобах;
- систематизації проблем при створенні й впровадженні системи для крос-платформного тестування;
- систематизації проблем при створенні й впровадженні системи для крос-браузерного тестування.

Над прототипом системи можна продовжувати роботу у наступних напрямках:

- нарощення кількості підтримуваних ОС та браузерів й їх версій;
- оптимізації використання ресурсів;
- UI для кінцевих користувачів;
- просування платформи у конкурентному середовищі.

РОЗДІЛ 1
ПРОБЛЕМАТИКА ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Впровадження крос-платформного тестування в компаніях

Важливо розуміти, що реалізувати крос-платформне тестування можливо, навіть у маленьких компаніях з чисельністю інженерів близько 10. Головною проблемою, є раціональність такого рішення.

Вон Вернон у своїх книгах по DDD писав, що продукт необхідно розділяти на 3 частини.

Перша частина – це ядро вашої системи, саме воно створює конкурентну перевагу вашої компанії, на цю частину має бути виділено найбільше ресурсів та найкращих розробників, а ПЗ добре захищене.

Друга частина – це допоміжна частина, проте вона має суттєвий вплив на результати роботи ядра, таку систему можна передавати у розробку вендорам.

Третя частина – це утиліта, що може бути придбана, або взята готовою, й не потребує концентрації інженерів компанії [1].

Ця робота створює продукт, що буде належати до третьої групи ПЗ інших компаній, це означає, що інженери інших компаній й самі могли б створити подібну систему, проте ця система має зберегти їх час.

1.2. Проблематика крос-платформного тестування

1.2.1 Проблематика крос-платформного тестування в сучасних методологіях розробки

Провідні компанії по розробці ПЗ віддають перевагу розробці за моделлю CI/CD, що включає в себе необхідність постійного й повторюваного тестування без

взаємодії з людиною.

З іншого боку, крос-платформне тестування вимагає виділення обчислювальних потужностей для розгортання платформ. Таким чином має бути створена модель, що буде динамічно керувати інфраструктурою, виділяти необхідні ресурси, налаштовувати їх відповідно до потреб користувача, надавати їх у використання, відслідковувати використання для виставлення ціни послуг, делокувати ресурси після використання й оптимізовувати використання ресурсів.

Однією з найбільших проблем цього виду тестування є відсутність стабільності розробленого продукту, адже розробка залежить від інтерфейсу сторонніх систем (ОС, Браузера, Сервісу віртуалізації) - ці ресурси змінюються, при чому доволі часто, й не завжди є backward-compatible, а старі версії стають недоступними.

Зворотна сумісність (Backward compatibility) – це властивість системи, яка забезпечує взаємодію зі старішою версією системи, в результаті чого інші програми чи люди можуть продовжувати працювати з новою версією без переробки програми.

Якщо компанія розробляє платформу для крос-браузерного та крос-платформного тестування самотужки, вона мусить виділяти інженера або цілу команду, яка буде відслідковувати новини про зміни в інтерфейсі взаємодії та вносити зміни, якщо необхідно.

1.2.2 Складнощі при розробці платформи для крос-платформного крос-браузерного тестування

Першою проблемою для розробника подібної системи є необхідність десь запускати тести. Їх можна запускати локально, проте тестувальник буде обмежуватись можливостями ОС та потужністю обладнання. Обидві проблеми є нетривіальними для вирішення.

Проблеми ОС можна вирішити використовуючи віртуалізацію, проте автоматизувати таку систему й зробити її розширювальною в організації буде дуже складно. Для того щоб можна було створювати потужні віртуальні машини (ВМ)

потрібні будуть дорогі робочі машини (в деяких випадках зробити віртуалізацію можливо буде лише на MacBook Pro).

Другою проблемою для розробника подібної системи є неоднаковість налаштувань, поведінки та коду браузерів. Таким чином, для підтримки крос-браузерності код, який тестує бізнес логіку, потрібно буде додатково обробляти для запуску в певній версії певного браузера.

Третьою проблемою для розробника стане раціональне використання ресурсів. Браузер може використовувати різну кількість потужностей машини чи ВМ, залежно від того, що виконується в браузері, тому буде існувати конфлікт між стабільністю роботи системи (дати більше пам'яті ніж потрібно, щоб завжди вистачало) й ефективністю.

Четвертою проблемою розробки платформи для тестування, стане розподілення внутрішніх та клієнтських помилок, а також помилок тестуємих систем. Таким чином можуть виникати False-Positive, False-Negative, True-Positive та True-Negative результати, для того, щоб їх виявляти потрібно мати систему зворотного зв'язку із користувачем а також механізм моніторингу та корекції системи.

Сьогодні в країнах, де IT галузь більше розвинена у напрямку outstaff та outsource (до яких належить й Україна), просувати подібні платформи надзвичайно складно, бо розробники сильно сфокусовані на вирішенні технічних задач й часто самі створюють проблеми (потреби) й вирішують їх.

Розробники схильні ухилятися від придбання сервісів-утиліт, хоча один місяць компенсації одного досвідченого розробника коштує більше ніж півроку підписки на існуючі подібні системи.

1.2.3 Проблеми безпеки при роботі з платформами тестування

Частою проблемою для компаній у використанні платформ є питання безпеки.

Всі існуючі платформи тестування, які продаються за PaaS (platform-as-a-service) та SaaS (software-as-a-service) моделями підтримують та рекомендують

використовувати виключно синтетичне тестування, тобто тестування без використання важливих (реальних, користувацьких) даних.

Платформа як послуга (PaaS) — це крок далі від повного локального управління інфраструктурою. Постачальник розміщує апаратне та програмне забезпечення у власній інфраструктурі та надає цю платформу користувачеві як інтегроване рішення, стек рішень або послугу через Інтернет-з'єднання.

Насамперед корисний для розробників і програмістів, PaaS дозволяє користувачеві розробляти, запускати та керувати власними програмами без необхідності створювати та підтримувати інфраструктуру або платформу, зазвичай пов'язану з цим процесом [2].

Програмне забезпечення як послуга (SaaS), також відоме як служби хмарних додатків, є найповнішою формою послуг хмарних обчислень, яка забезпечує програму, якою керує постачальник, через веб-браузер.

Оновлення програмного забезпечення, виправлення помилок і загальне обслуговування програмного забезпечення обробляються постачальником, а користувач підключається до програми через панель інструментів або API. Програмне забезпечення не встановлюється на окремі машини, а груповий доступ до програми є більш надійним.

SaaS — чудовий варіант для малого бізнесу, у якого немає персоналу або здатності для встановлення та оновлення програмного забезпечення, а також для програм, які не потребують особливої налаштування або які будуть використовуватися лише періодично [2].

Для систем, які були невірно спроектовані, це стає проблемою, бо в них деякі операції може зробити лише супер-користувач, який сумнівно захищений лише тим, що доступ до його логіну має обмежене коло осіб. Для впровадження функціонального крос-платформного тестування в таких системах необхідно багато додаткових зусиль по переробці системного дизайну, можливо, імплементації бізнес логіки (саме її імплементації в коді) й рефакторингу коду.

Наступною проблемою може стати необхідність розгортання тестової інфраструктури у приватній мережі компанії, якщо для приватної хмари це зробити

відносно просто, то для on-premis архітектур це питання може бути серйозним викликом, або бути не вирішуваним зовсім.

Локальне програмне забезпечення (On-premise) встановлюється та працює на комп’ютерах особи чи організації, яка використовує це програмне забезпечення, а не на віддаленому об’єкті, наприклад, хмарі.

Програмне забезпечення складається з бази даних та модулів, які об'єднані, щоб задовольнити потреби організацій щодо автоматизації корпоративної бізнес-системи та її функцій [2].

Деякі приватні мережі, наприклад, такі як корпоративний ВПН, налаштовані таким чином, що не підтримують сервіс-сервіс аутентифікацію, а націлені лише на аутентифікацію людей, й водночас компанія немає засобів автоматизованого розгортання сервісів, тому не може розгорнути платформне ПО у себе.

Існують окремі організації, що впроваджують функціональне тестування в таких компаніях. Вони продають й платформу, й фахівців, що змінюють структуру та ПО клієнта, щоб інтегруватися.

1.3 Існуючі рішення

1.3.1 Sauce Labs

Sauce Labs - це компанія, яка надає можливість автоматизованого тестування хмарних веб і мобільних додатків та пропонує своїм користувачам крос-платформне крос-браузерне тестування, code-less тестування, error-reporting, beta-тестування для мобільних додатків, UI-тестування, АПІ-тестування, автоматизацію тестування [3].

Цей застосунок задовольняє функціональні потреби більшості компаній. Серед переваг також наявність code-less тестування, тобто можливості створення перевірок знавцями бізнесу, стейк-холдерами або продукт-оунерами без потреби розробників й навичок програмування.

До переваг цієї платформи можна віднести й партнерство з усіма хмарними провайдерами, це дозволяє легко інтегруватися із продуктом компанії, що розгорнуто у хмарі.

Можливість автоматизації тестування й побудови процесу неприпиненого тестування допомагає забезпечити високу якість продукту.

Серед користувачів продукту багато всесвітньо відомих компаній й організацій, а саме: VISA, splunk, Walmart, Alaska Airlines, Deutsche Bank та інші.

До недоліків цього продукту можна віднести доволі високу ціну експлуатації (TCO), відсутність можливості побудови авто-тестів за допомогою комп'ютерного зору (QA інженер робить тест мануально 1 раз, й система вчить ці дії, а потім повторює їх), не гнучке оформлення пакетів послуг (користувачу нав’язують купувати те, що йому не потрібно, спершу як бонус, щоб купити необхідну функцію).

1.3.2 Functionize

Ця платформа є стартапом і відкриває для користувачів можливості створення code-less функціонального синтетичного тестування. Тести створюються за допомогою AI, QA достатньо 1 раз виконати тест мануально у браузері й система запам'ятає ці дії, оптимізує їх і буде виконувати їх подібно до людини (людська швидкість, вразливість, тощо). Крім того бот буде не тільки шукати помилки, а й знаходити будь-які візуальні чи експлуатаційні зміни порівняно з попереднім станом [4].

Платформа надає можливість інтегруватися з такими інструментами, як:

- Jenkins – платформа дозволяє виконувати тести Functionize як частину jenkins pipeline jobs, таким чином тести Functionize будуть запускалися щоразу, коли запускається збірка;
- Azure devOps piplines – платформа надає можливість виконувати тести Functionize під час кожної збірки;
- Pager duty – платформа дозволяє надсилати повідомлення про виконання тестів і повідомляти про тести, які потребують негайної уваги;
- Test Rail – допомагає тримати всіх у курсі прогресу покриття тестами та надає можливість зробити репорт результатів тестування в TestRail.

За допомогою цієї платформи впродовж 1 тижня можна покрити функціональним тестуванням чималий продукт.

Проте, цінова модель “плати за тест, що ти запустив” без можливості фіксованої підписки може відштовхнути користувачів, бо прив’язавшись до цієї платформи перейти з неї буде складно, а ціна буде збільшуватись зі збільшенням власного продукту. Крім того, компанія все ще знаходиться у стадії стартапу, тому документації по вирішенню проблем не багато, а служба підтримки компанії працює повільно й неякісно.

1.3.3 UiPath

Найпотужніша платформа для тестування застосунків. UiPath пропонує повний функціонал по розгортанню платформи для автоматизації усіх рівнів тестування, пропонує інженерну підтримку, якщо клієнт не має власних фахівців, AI-driven, Robotics, Real-Human тестування, бета-тестування, побудову внутрішніх процесів компанії по забезпеченню якості.

Платформа автоматизації поєднує в собі рішення Robotic Process Automation (RPA) з повним набором можливостей, які дозволяють кожній організації масштабувати цифрові бізнес-операції з безпрецедентною швидкістю.

UiPath дозволяє автоматизувати повторювані завдання, які зазвичай виконуються людьми. Технологія поєднує в собі імітацію того, як люди читають екрани комп’ютерів (AI Computer Vision) з API, і надає користувачам доступ до попередньо створених компонентів автоматизації, які можна комбінувати для автоматизації рутинних процесів [5].

Серед клієнтів компанії всесвітньо відомі бренди, для яких якість єдиний спосіб залишатися в бізнесі: Google, NASA, EQUIFAX, Airbus, HP, Autodesk.

Єдиною проблемою для використання цього рішення для багатьох компаній є ціна. Стартовий пакет застосунку коштує декілька мільйонів доларів в рік, а преміум пакети, що включають допомогу фахівців з інтеграції і побудови процесів – десятки мільйонів доларів.

1.3.4 UFT MicroFocus

Micro Focus UFT – це автоматизоване програмне забезпечення для тестування, призначене для тестування різних систем. Micro Focus UFT виконує функціональне та регресійне тестування через інтерфейс користувача, такий як native GUI або веб - інтерфейс [6].

Компанія MicroFocus володіє багатьма продуктами у сфері розробки ПО, 1 з них це UFT (Unified Functional Testing).

UFT підтримує інтерфейси ключових слів та сценаріїв та має графічний інтерфейс користувача. UFT використовує мову сценаріїв Visual Basic Scripting Edition (VBScript) для визначення процедури тестування та маніпулювання об’єктами та елементами керування програми, що тестується. UFT дозволяє розробникам перевіряти всі три рівні операцій програми з однієї консолі: інтерфейс, рівень обслуговування та рівень бази даних.

UFT працює шляхом ідентифікації об’єктів в інтерфейсі користувача програми або на веб - сторінці та виконання бажаних операцій (наприклад, клацання мишею). UFT також може захоплювати властивості об'єктів, такі як ім'я або ідентифікатор.

Уніфіковане функціональне тестування HPE використовує мову сценаріїв VBScript для визначення процедури тестування та маніпулювання об’єктами та елементами керування програми, що тестується. Для виконання більш складних дій користувачам необхідно маніпулювати базовим VBScript [7].

Незважаючи на те, що UFT зазвичай використовується для автоматизації тестових випадків на основі інтерфейсу користувача, він також може автоматизувати не UI тестові випадки, такі як операції з файловою системою, тестування баз даних, веб-служб та API.

Основна відмінність компанії, що вона продає не тільки платформу, а й допомагає з її інтеграцією. Основні замовники компанії це державні установи, банки та організації, що вимагають жорстких галузевих сертифікацій.

Компанія є успішною у своїй ніші, проте не всім замовникам потрібна допомога з сертифікаційним визнанням якості ПО.

1.3.5 CrossBrowserTesting SmartBear

Компанія SmartBear пропонує своїм користувачам продукт CrossBrowserTesting він надає функціонал по тестуванню в 5-ти основних браузерах та на віртуальних машинах, що симулюють мобільні телефони.

CrossBrowserTesting — це інструмент на основі SaaS, який повністю працює в режимі онлайн.

Cross browser testing надає можливість робити автоматичні знімки екрана веб - сайту в декількох веб - браузерах і на різних пристроях, роблячи візуальну перевірку та повторне тестування простим та порівнює ці знімки з попередніми версіями [8].

CrossBrowserTesting пропонує тестування в реальному часі на реальних пристроях. Тестування в режимі реального часу дає змогу віддалено взаємодіяти з веб-сайтом на різних пристроях, ніби веб-сайт переглядається на одному пристрої з тими самими вибраними конфігураціями. Роблячи це, можливо дізнатися, як веб-сайт насправді виглядатиме на пристроях, і візуально підтвердити, що він відображається правильно. Для цього не знадобляться додаткові пристрої для встановлення.

Серед переваг цього застосунку - відносна дешевизна порівняно до продуктів, що коштують мільйони.

Серед недоліків - можлива різниця у поведінці віртуальної ОС мобільного телефона, порівняно до реальних пристроїв, та повільна робота інфраструктури. Відсутність партнерства з хмарними провайдерами призводить до того, що зробити інтергацію складніше, а можливість дотримання вимог безпеки менше.

1.3.6 Appium

Appium це Open-Source розробка, призначена для тестування Native та мобільних застосунків. Appium не надає платформи для тестування (код запускається локально, або на серверах користувача). Застосунок дає додатковий рівень абстракції над Selenium та iOS й Android SDK, для простішого написання

тестів. Appium керує програмами iOS, Android та Windows за протоколом WebDriver, надає можливість автоматизувати будь-який мобільний додаток з будь-якої мови та будь-якої тестової бази [9].

До переваг цього рішення можна віднести: безкоштовність та спрощення розробки функціональних тестів для мобільних застосунків.

До недоліків можна віднести: відсутність платформи або хмари й відповідно потужностей на яких запускаються тести - це означає, що для того, щоб запустити тест для iOS, необхідно мати MacBook. Цей недолік значно підвищує початкові витрати на обладнання компанії. Також, суттєвим недоліком є те, що для використання системи користувач має мати гарні навики програмування (неможливо створювати code-less тести) й розібратися у роботі бібліотеки.

1.3.7 Perfecto

Perfecto – це хмарне програмне забезпечення для тестування веб- та мобільних додатків, розроблене, щоб допомогти підприємствам у різних галузях, таких як роздрібна торгівля, банківська справа, фінанси, страхування, медіа та розваги, технології тощо, тестувати програми на різних операційних системах і пристроях. Це дозволяє користувачам автоматизувати процеси, пов’язані з розширеними тестовими сценаріями, і отримати доступ до версій і пристроїв браузера відповідно до індивідуальних вимог.

Perfecto пропонує своїм користувачам можливість створювати тести за допомогою побудови діаграми бізнес-процесу, що потім транслюється в код, й потім, підтримки за допомогою AI.

Perfecto охоплює автоматизацію функціонального та нефункціонального тестування, включаючи навантажувальне тестування та тестування API. Крім того, Perfecto дозволяє автоматизувати тестування аудіо та візуальних зображень, різних локацій, мереж, тощо [10].

Perfecto підтримує створення тестів в стилі Gherkin, значно зменшує обслуговування за допомогою вбудованого AI, який самостійно відновлює

автоматизовані тестові сценарії, дозволяє виконувати сценарії автоматизації за допомогою будь -якого інструменту CI та переглядати результати на інформаційній панелі Perfecto CI.

Продукт є більш спрощеною версією порівняно із SauceLabs чи Functionize, проте є дешевшим.

Серед недоліків продукту можна відмітити відсутність партнерства з хмарними сервісами й важче розгортання в умовах приватної мережі, що є необхідним для більшості компаній з точки зору забезпечення безпеки.

Висновок до 1 розділу

У даному розділі були розглянуті проблеми та основні вимоги щодо створення, інтеграції, впровадження, експлуатації та володіння системами крос-платформного крос-браузерного тестування. Було визначено, що:

- система має добре інтегруватися із CI/CD;
- динамічно керувати інфраструктурою розгортання без людини, або з мінімальним втручанням людини у процес;
- відділяти кінцевого користувача платформи від прямої взаємодії з ОС та браузером, які постійно змінюються, й потребують рефакторингу коду для інтеграції;
- надавати можливість розгорнути тести у хмарі, для того, щоб не купувати й підтримувати власну інфраструктуру;
- раціонально керувати інфраструктурою користувача платформи, допомагати користувачу використовувати обчислювальні потужності раціонально;
- вміти розрізняти типи помилок системи, відділяти помилки тестуємої системи від помилок клієнта та внутрішніх помилок платформи;
- мати потужний внутрішній моніторинг для відслідковування проблем платформи;
- добре інтегруватись у приватні мережі.

Було виділено пріоритети користувачів систем:

- стабільність системи та її результатів;
- ціна;
- безпека;
- рівень автоматизації створення коду, розгортання серверів, підтримки;
- codeless системи.

Було виділено основні типи продуктів, що існують на ринку:

- створення системи забезпечення якості та процесів тестування з 0, за допомогою платформи вендора, тренерів, спеціалістів з інтеграції, їх розробників;
- платформа плюс допоміжний персонал по інтеграції;
- code-less платформа (не вимагає навичок програмування чи налаштувань від користувача);
- платформа;
- продукт, розгорнутий на інфраструктурі клієнта;
- бібліотека.

В результаті аналізу було виявлено, що на ринку існує багато рішень, проте всі вони мають недоліки для користувачів, й мала утилітна система без надлишкового функціоналу, який вартує додаткових грошей, котра буде керувати й надавати у використання інфраструктуру в якості платформи для тестування, й підтримувати codeless тестування, чи хоча б, бібліотеки для легшої підтримки тестів й добре інтегруватися у приватні мережі, буде мати попит.

РОЗДІЛ 2
ОРГАНІЗАЦІЯ ФУНКЦІОНАЛЬНОГО ТЕСТУВАННЯ

2.1 Формування обмежених контекстів та єдиної мови бізнесу

Для того, щоб організувати якісне функціональне тестування, не залежно від того крос-платформне воно чи ні, необхідно проаналізувати який функціонал має продукт. Повертаючись до методології DDD, необхідно створити обмежені контексти, карту контекстів, та під області [1].

Тепер необхідно розподілити компоненти ПЗ, що відображають обмежені контексти в коді, на 3 групи, які вже були визначені в розділі 1. Очевидно, що не має жодної потреби проводити функціональне тестування ПЗ третьої групи. Більшість цих ПЗ – це Open-Source утиліти чи придбані сервіси, якщо це не так, то скоріше за все це помилка архітектури та розподілу ресурсів організації.

Покривати функціональним тестування необхідно саме смислове ядро системи, й якщо необхідно, критичні контексти та під області 2-ї групи.

Створення обмежених контекстів повинно було призвести до формування єдиної мови у кожному з контекстів, потрібно розвити цей процес й єдину мову. Необхідно ретельно пропрацювати функції, й їх описання лінгвістичними конструкціями, у кожному з контекстів, скоріше за все, що перевірка виконання цих функцій й буде абстрактним рівнем опису сценарію тестування. На цій стадії необхідно відокремитись від деталей технічної реалізації платформи, а сконцентруватись на виявлені того, що (який функціонал) необхідно тестувати.

Заміна лінгвістичного підходу, тобто мови користувачів системи, технічним підходом (тестування поточної імплементації) призведе до того, що тести будуть

перевіряти поведінку системи, як її запланували розробники, а не те, як її використовують реальні користувачі – таке функціональне тестування це просто нераціональна витрата ресурсів компанії.

2.2 Стадія стратегічного проектування

Проблема стадії стратегічного проектування полягає в тому, що компанія, що хоче організувати функціональне тестування, вже витратила, скоріше за все, місяці роботи для виконання пункту 2.1, проте жодної строчки коду, чи тесту ще не написано, й це може непокоїти вище керівництво компанії, які питають про результат, й сказати, що тепер необхідно витратити ще місяці на проектування дуже часто є нереально, бо команду з реалізації проекту просто звільнять. Проте потрібно пам’ятати, що не можна вибирати між гарним системним дизайном та його відсутністю, це завжди вибір між гарним та поганим дизайном – відмова від проектування це вибір поганого дизайну.

Для того щоб досягти мети по впровадженню функціонального тестування можна йти двома шляхами.

Перший шлях полягає у поясненні важливості всіх стадій впровадження цього виду тестування, й виділенню часу під них - це кращий варіант, проте не завжди він є реальним.

Другий шлях є компромісом, проектування методом моделювання – створення малого прототипу або прототипів майбутнього рішення для демонстрації - це призведе до втрат додаткових ресурсів, проте дасть мінімальний продукт для демонстрації виконаної роботи вищому керівництву.

Не дивлячись на те, що моделювання буде забирати додаткові ресурси у команди, воно може допомогти у процесі створення вимог до кінцевої системи функціонально тестування.

Під час виконання пункту 2.1 компанія вже відділила ядро системи, проте й воно само може бути значним за розмірами.

На стадії стратегічного проектування потрібно оцінити, які засоби забезпечення якості вже існують, й який функціонал необхідно покрити функціональним UI тестуванням.

В залежності від того, що необхідно тестувати наступним кроком проектування буде створення специфікації для продукту тестування. Оскільки функціональне тестування не є смисловим ядром продукту компанії, то буде раціонально купити вже готову платформу. Необхідно вибрати існуючі продукти на ринку і перейти до фази тактичного проектування й прототипування можливої системи, використовуючи різні сервіси й перевіряючи чи задовольняють вони вимогам специфікації.

Після вибору задовільних платформ необхідно перейти до складання ТСО моделі кожної з них.

2.3 Опис процесу створення ТСО

Сукупна вартість володіння (total cost of ownership) – це загальна величина цільових витрат, які власник несе з моменту вступу в право власності на певний продукт до моменту виходу з права власності та виконання власником зобов'язань, пов'язаних з володінням, у повному обсязі.

Перше, що необхідно зазначити, це те, що процес створення ТСО (total cost of ownership) вимагає затвердженої специфікації до системи тестування. Кожна із систем, для якої створюється ТСО повинна реалізовувати вимоги специфікації повністю. Іншими словами, приймати участь у цьому “змаганні” можуть лише системи, що підходять для вирішення задачі, так як не можна порівняти за допомогою ТСО дешевшу, але не до кінця задовільну систему, із задовільною і дорогою – для таких порівнянь є інші, комплексні методи.

Для порівняння платформ тестування раціонально використати саме ТСО як метод оцінки, оскільки можна чітко визначити функціональні вимоги, а на ринку є достатня кількість компаній, що надають послуги серед яких можна обирати.

Для того, щоб побудувати ТСО, раціонально створити прототипи систем із використанням кожного потенційного рішення. Для цього необхідно скласти діаграму use-case для цієї платформи й визначитися з усіма унікальними й необхідними сценаріями використання. Після створення прототипів необхідно залишити для розглядання лише ті системи, що задовільнили вимогам специфікації по всім use-case.

ТСО програмного забезпечення зазвичай складається з двох основних частин – це ціна ПЗ й винагорода людей, що його експлуатують або розробляють.

Якщо система-претендент – це стороння система, зазвичай, ціна визначається шляхом перемовин із провайдером і заключенням контракту. Для подібних систем ціна залежить від кількості користувачів системи, кількості підтримуваних ОС, ціни інфраструктури, виду тестування, кількості тестування й регіональності. Чим більше тестів й інших можливостей компаній купує, тим, зазвичай, дешевше вартість одиниці продукції, проте загальна сума зростає.

Якщо система-претендент – це система, яку планують розробити всередині компанії або за допомогою вендора – оцінити її вартість значно складніше. Ціна буде включати ціну інфраструктури, ціну розробників системи, ціну постійної підтримки цієї системи, ціну створення навчального матеріалу для роботи із системою новачками. Для того, щоб оцінити ці витрати зазвичай також використовують метод прототипування, коли в першій ітерації створюється простий, але функціонально репрезентативний прототип, а потім проводиться стратегічне планування, в ході якого оцінюється скільки ресурсів буде витрачено на заміну частин прототипу системою, що буде задовольняти специфікації.

Для подібних систем розроблених всередині компанії великою проблемою є ціна підтримки, адже програми 3-ої групи не належать до смислового ядра бізнесу і вимагають від команди розробників поглиблюватись у зовсім інший контекст.

Важливо зазначити, що навіть при використані добре автоматизованих систем, витрати на винагороду працівникам, що будуть працювати над тестуванням, можуть зменшуватись, проте завжди будуть необхідними.

Наразі на ринку не існує жодного ультимативного сервісу, що зможе забезпечити якість ПЗ клієнтів без втручання інженерів й фахівців предметної області.

При виборі системи для тестування важливо оцінювати наскільки ця система зможе зменшити витрати на персонал, а також наскільки легко буде знайти кваліфікований персонал, що зможе працювати із цією системою, важливо пам’ятати, що інженери також мають свої особистості переваги й надають перевагу у роботі добре зарекомендованим сервісам з якими у них є успішний досвід впровадження чи використання.

Повертаючись до основ розробки продукту за методологією DDD, коли мова йде про розробку ПЗ, що відноситься до допоміжного 3-го класу, правильним вибором є використати вже готове рішення, придбати вже готовий сервіс, чи найняти окрему компанію (вендора), що спеціалізується на подібних рішеннях, для розробки внутрішньої системи, при умові, що всі існуючі не задовольняють вимогам.

Висновок до 2 розділу

У розділі 2 описана покрокова послідовність дій для створення кращої платформи для тестування, а саме:

- формування обмежених бізнес контекстів;
- формування єдиної мови бізнесу;
- відокремлення смислового ядра продукту;
- створення специфікації;
- створення прототипу;
- TCO аналіз.

Для компаній важливо дотримуватись цієї послідовності дій, щоб досягти своєї мети – впровадження функціонального тестування.

Для розробників платформ тестування важливо розуміти цей алгоритм вибору, для того, щоб отримувати конкурентну перевагу та вміло просувати свій продукт на ринку.

Описано процес створення TCO моделі для платформи крос-платформного та крос-браузерного тестування. Описано основні вимоги та рекомендації до цього документа.

Надано поради щодо вибору правильної моделі на основі парадигми DDD.

РОЗДІЛ 3
ЦІНОУТВОРЕННЯ У ПЛАТФОРМАХ ТЕСТУВАННЯ

3.1 Розподіл повноважень компонентів платформи

Будь-який засіб тестування може відноситися до засобів моніторингу чи засобів аналізу помилок, чи бути комбінацією цих двох.

Важливо розуміти, що одна система не може одночасно забезпечувати гарні результати в обох напрямках. Це зумовлено тим, що для того, щоб зробити засіб для аналізу помилок треба мати доступ до внутрішнього коду системи й даних. З іншого боку функціональний моніторинг є еквівалентом до користування системою реальним користувачем, тому всі запити повинні йти зовні. Будь-яка спроба створити єдину універсальну систему зазнає поразки.

Проте користувачі після отримання інформації про поломку хочуть одразу знати про те, чому вона відбулась. Тому платформи, фактично, з'єднують 2 застосунки. Крім з'єднання систем моніторингу й аналізу ще додається аналітика – засоби відео реєстрації тестів, машинного навчання, у деяких випадках ще й машинного зору.

Для того, щоб отримати конкуренту перевагу для цього проекту необхідно зменшити його вартість порівняно до існуючих застосунків. Крім технічної оптимізації, що будуть зроблені, необхідно оптимізувати й бізнес модель платформи.

Зазвичай, користувачам важливіше знати, чи працює їх продукт зараз і приносить користь, чи ні, а засоби аналізу помилок можуть бути різними. Виходячи

з цього, продукт буде орієнтуватись саме на створення платформи з функціонального моніторингу, тобто оцінки працездатності продуктів, а не аналізу причин їх помилок, логування чи дебагу.

На версії прототипу із застосунку також буде виключено й засоби створення code-less тестів із застосуванням машинного зору.

Зменшення кількості компонентів до одного значно зменшить кількість витрат, при цьому застосунок буде цінним для користувачів, так як він все ще здатен повідомляти про стан системи.

3.2 Цінова політика

Користуватися застосунком можна буде на основі підписок двох видів.

Перша підписка не буде вимагати щомісячних платежів, й буде працювати за моделлю плати за те, що використовуєш.

Середня ціна на ринку за такі тести 6$ за тисячу тестів в місяць, й додаткова плата за зберігання даних (згідно цін провайдера) й плата за кількість користувачів.

В таблиці 3.1 наведено порівняння цін на підписку помісячно у різних провайдерів моніторингу даних.

Таблиця 3.1

Порівняння цін на підписку

Сервіс	Ціна	Посилання
AppDynamics	~6 $ / 1000 tests / month	https://www.appdynamics.com/pricing
New relic	~10$ / 1000 tests / month	https://newrelic.com/pricing
Dynatrace	~ 6.9 $ / 1000 tests / month	https://www.dynatrace.com/pricing/

Друга підписка буде вимагати підписки мінімум на 3 місяці. Оскільки платформа сама буде розгорнута у хмарі, прогнозованість навантаження від постійних користувачів дозволить оптимізувати власну інфраструктуру й самим оформити довготривалу підписку на хмарні ресурси.

В таблиці 3.2 наведено різницю цін в залежності від терміну орендування її у хмарного провайдера.

Таблиця 3.2

Порівняння цін на підписку

Найменування типу	Опис	Ціна за місяць USD	Ціна за місяць при оренді на рік	Ціна за місяць при оренді на три роки
t4g.nano	Посилання: https://calculator.aws Найменший з інстансів загального призначення, зазвичай використовується для малих застосунків, чи тестування	6.07	4.825 (на 20.5 % меньше)	4.168 (на 31.3% меньше)
t4g.large	Посилання: https://calculator.aws/ 2 vCPU, 8Gb. Підходить для розгортання невеликого кластеру,	52.06	31.68 (на 39.1% меньше)	21.46 (на 58.7% меньше)

	або застосунку на prod.			
c6g.8xlarge	Посилання: https://calculator.aws/ 32vCPU, 64 Gib, 12Gb/s, 90 000 IOPS. Підходять для високонавантажених сервісів, аналітики	797.24	469.91 (на 41% меньше)	301.64 (на 62% меньше
t4g.2xlarge	Посилання: https://calculator.aws/ 8 vCPU, 32 Gib, 5 Gb/s, 20 000 IOPS	199.22	117.82 (на 40% меньше)	76.3 (на 61.4% меньше)

Дана порівняльна таблиця цін дозволяє побачити, що оренда інфраструктури на 1 рік вперед дозволить зекономити в середньому 30% вартості інфраструктури, а на 3 роки – 60%.

Зменшення собівартості розробленої системи дозволить отримати конкурентну перевагу у ціновій конкуренції шляхом зниження кінцевої ціни для споживачів.

Висновок до 3 розділу

У розділі 3 було розглянуто функції систем моніторингу та засобів аналізу помилок у системах функціонального тестування. Було обране спрямування даної системи на функції моніторингу.

Розроблена цінова політика, що дозволить застосунку бути конкурентоспроможним на ринку подібних систем.

Були розглянуті моделі оплати для користувачів системи. Розглянуті можливості оптимізації витрат користувачем шляхом оформлення довготривалої підписки.

Розглянуто засоби досягання зменшення собівартості системи для надання можливості знижки користувачам, що оформлять довготривалу підписку.

РОЗДІЛ 4
ПРОТОТИПУВАННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

4.1 Опис користувацького інтерфейсу

На рисунку 4.1 зображено головну сторінку застосунку. В центрі підкреслено переваги цього застосунку, й є можливість для переходу до створення користувацького акаунту.

У верхній частині розташовано керуючу панель.

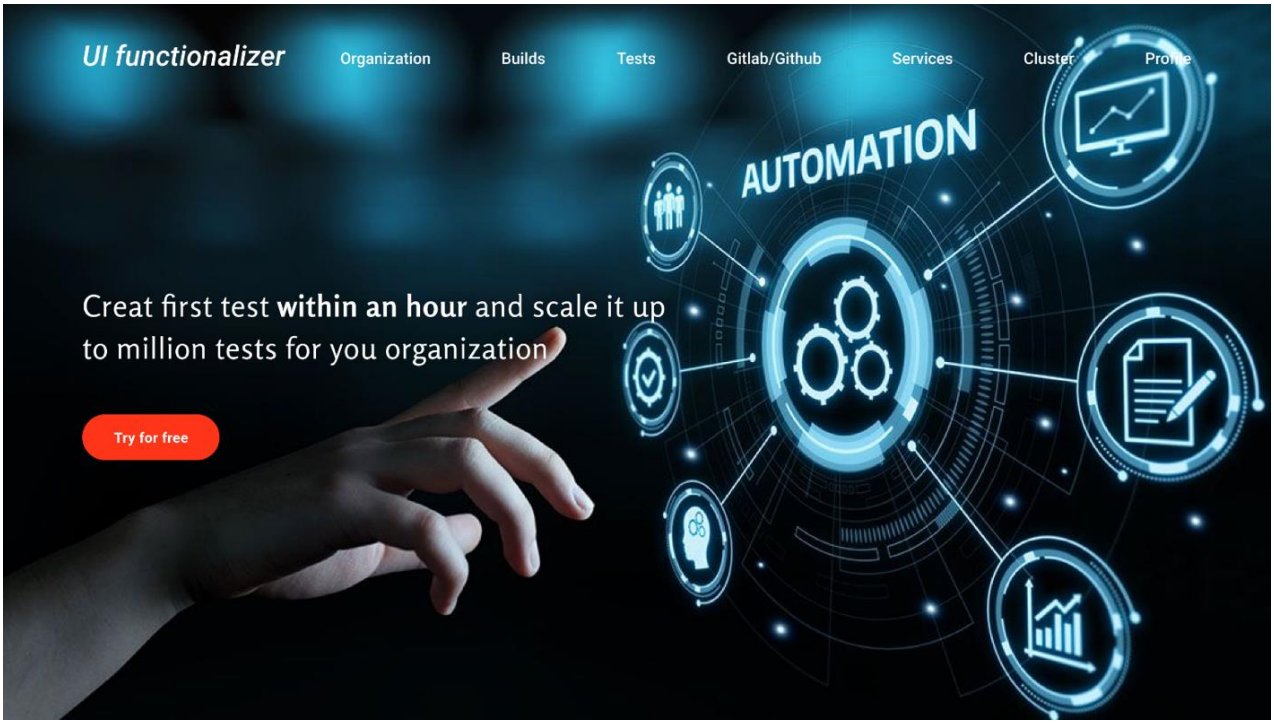

Рис. 4.1. Головна сторінка застосунку

На рисунку 4.2 та 4.3 видно, як користувач може перейти до своїх виконаних тестів. Він може переглянути останні тести, знайти певний тест по його імені, чи

знайти по тегу (зазвичай 1 запуск тестів має загальний тег, щоб їх можна було швидше групувати й знаходити).

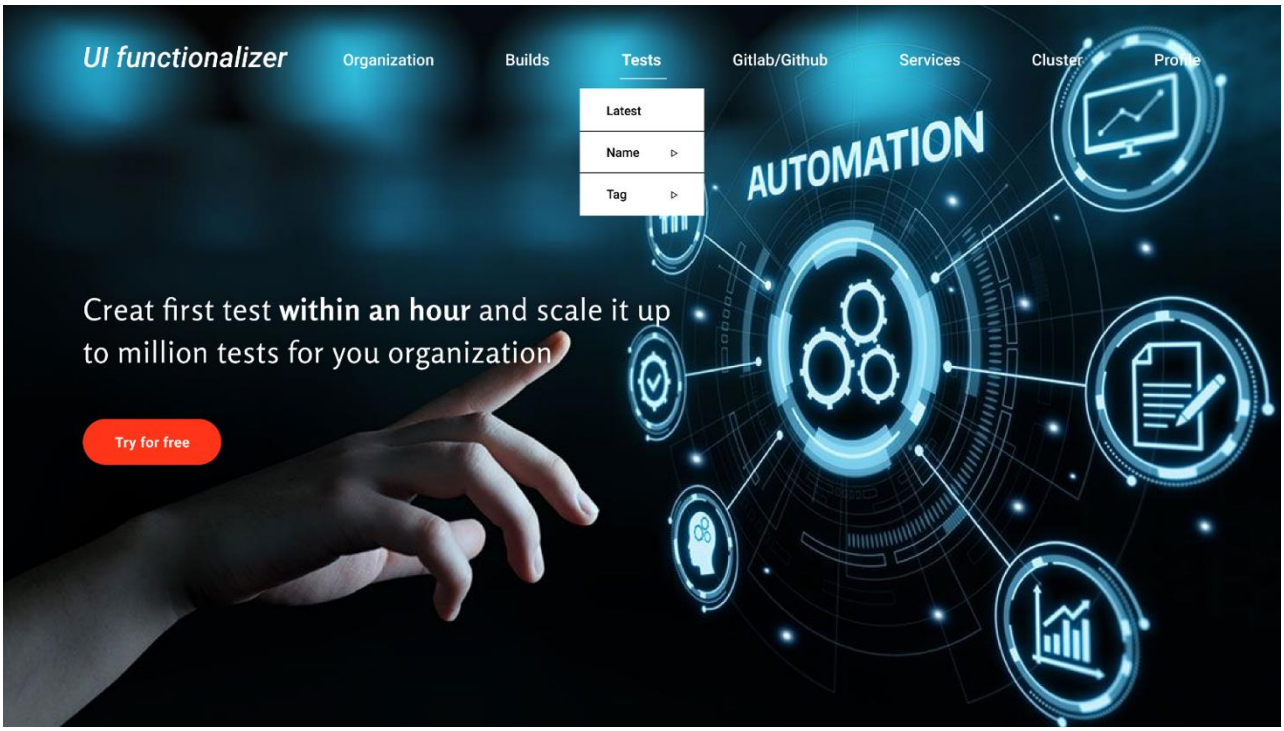

Рис. 4.2. Вкладка тестів

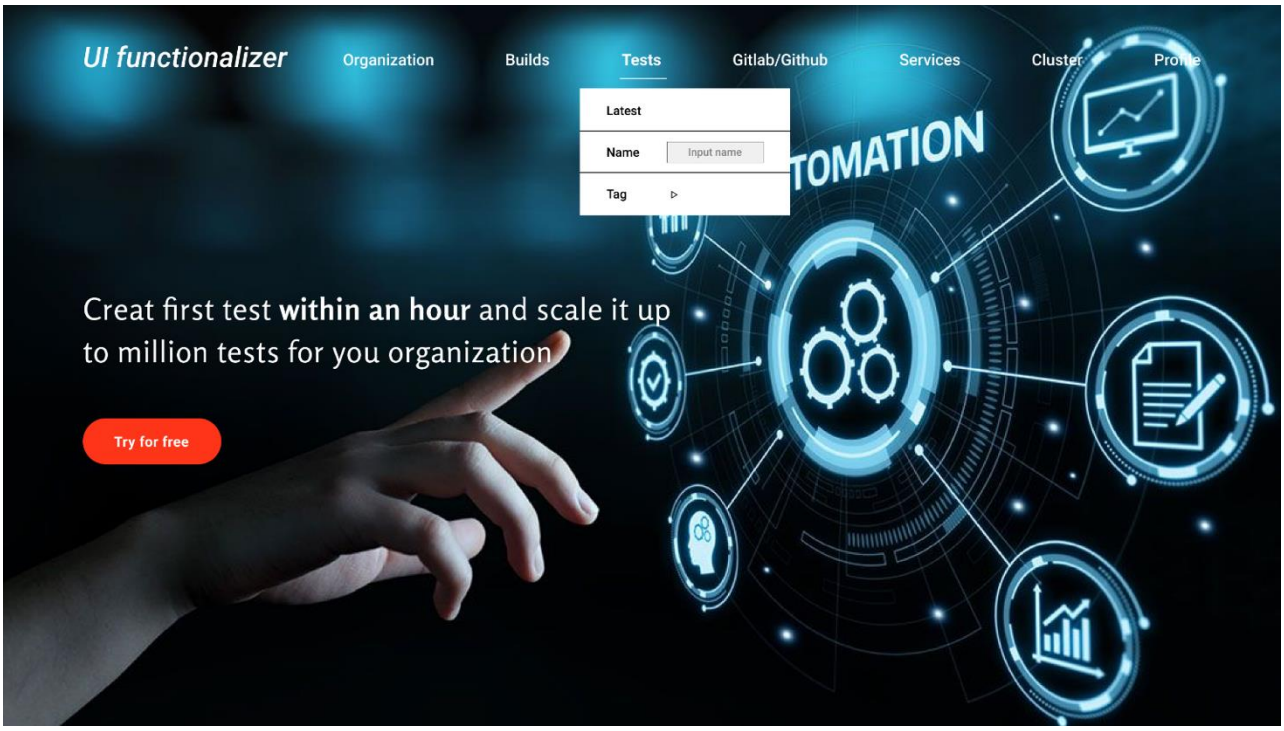

Рис 4.3. Пошук тесту по імені

На рисунку 4.4 показано, як користувач може швидко проаналізувати поточний стан його продукту. У вкладці сервіси видаються імена й стан у вигляді зеленого, помаранчевого чи червоного сигналу всіх тестованих сервісів цього користувача.

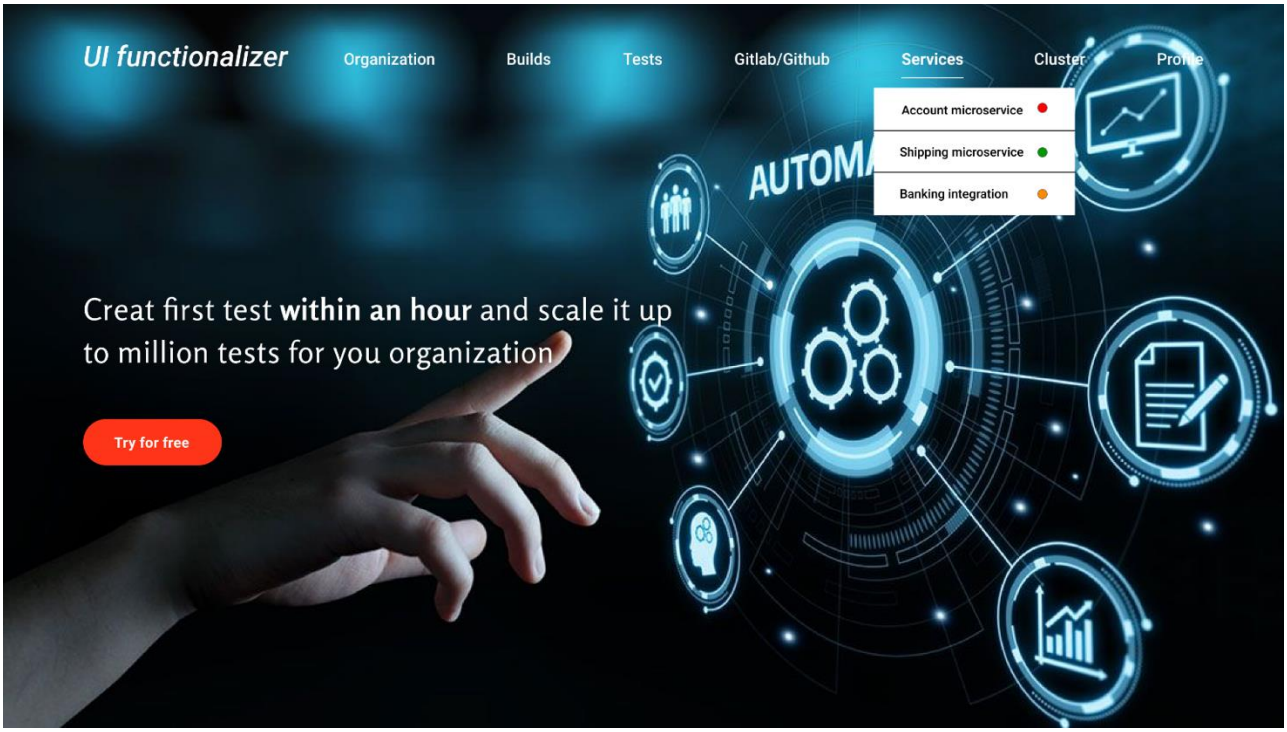

Рис 4.4. Статус сервісів користувача

На рисунку 4.5 показано таблицю результатів.

Результати включають загальний тег (якщо вказано), імена тестів, результати, дата, параметри запиту, кластер потужностей платформи для тестування (один користувач створеної платформи може придбати більш ніж 1 кластер, наприклад, щоб розділити потужності між різними частинами організації, чи між реальним середовищем чи середовищем розробника, тому кожен тест зберігає інформацію, де його було запущено) й імена сервісів.

Також а рисунку 4.5 у полі результатів є кнопка, що дає можливість користувачу повідомити розробникам платформи про неправильну поведінку платформи.

Дана функціональність необхідна, так як збій у роботі платформи може призводити до неправильних значень тестів користувачів. Для того, щоб швидко вирішувати такі проблеми існує така форма.

Дана кнопка допомагає розробникам платформи дізнаватись про помилку, якщо вона все ж таки не була перехоплена при розробці та тестуванні, й також із такими користувачами зв’язується служба підтримки, щоб знайти для них обхідний шлях для успішного користування платформою, чи зробити їм компенсацію, якщо їх робота заблокована.

Результати цих повідомлень потрапляють у аналітичну БД платформи, яка оцінює якість роботи системи й буде описана пізніше.

Рис. 4.5. Таблиця результатів тестів

На рисунку 4.6 представлено візуальний вигляд екрану нотифікації. Він має багато полів, більшість з яких авто-заповнюється значеннями з попередньої форми – це ім’я, тег, сервіс, кластер, результат та атрибути. Користувач повинен ввести правильний з його точки зору результат, який не співпадає з отриманим.

Також користувач може змінити атрибути запиту, щоб деталізувати помилку – для цього потрібно натиснути на поле атрибутів.

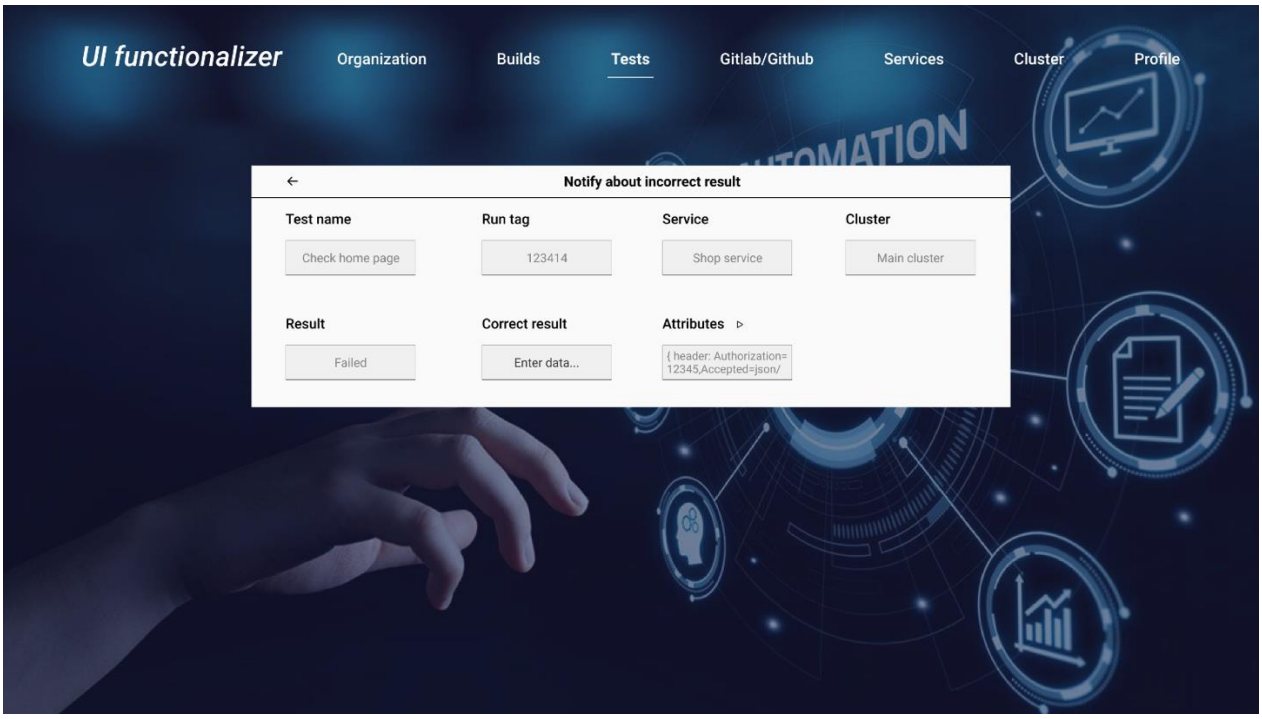

Рис. 4.6. Екран нотифікації про помилку

На рисунку 4.7 представлено екран редагування атрибутів запиту. Користувач може змінити фактичні дані, що були використані платформою, якщо вважає, що вони помилкові – це дозволить розробникам платформи значно швидше знайти помилку, якщо вона існує.

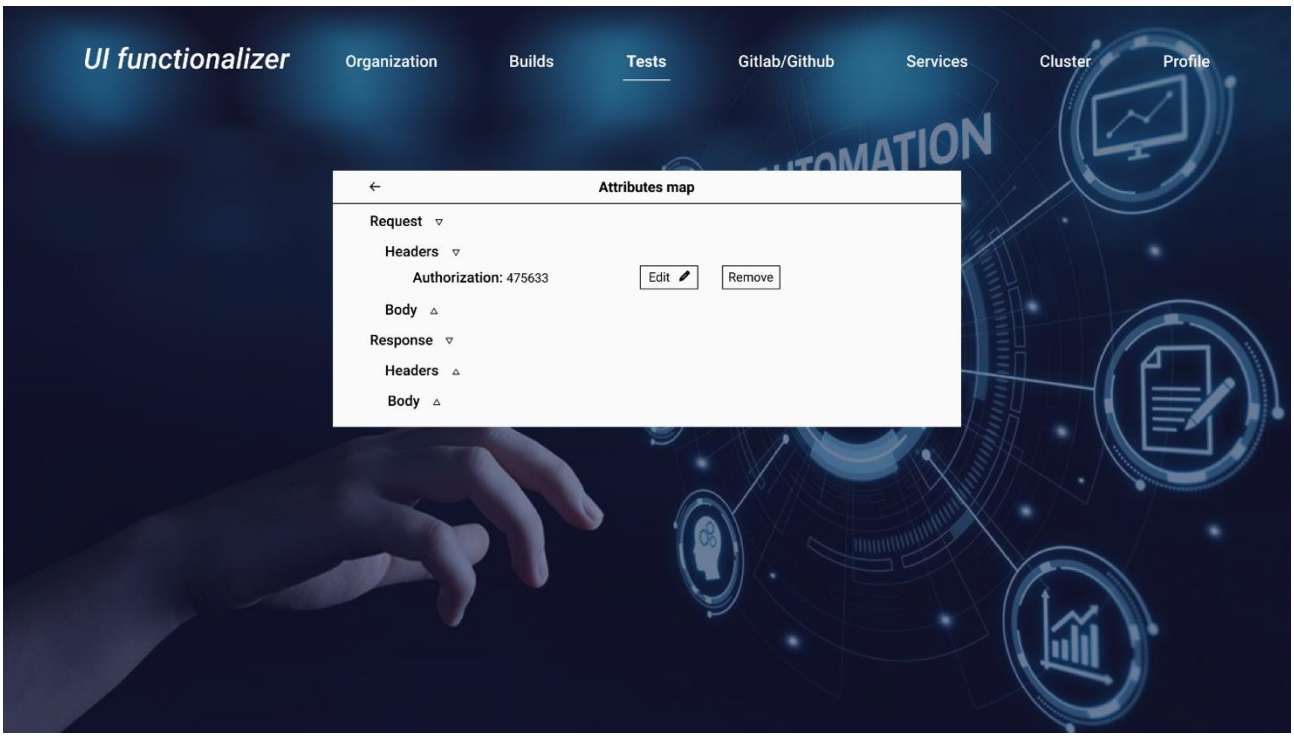

Рис. 4.7. Екран редагування атрибутів запиту

Висновок до 4 розділу

У розділі 4 було продемонстровано результати прототипування користувацького інтерфейсу для системи взаємодії із платформою тестування.

У ході прототипування було покрито функціональні вимоги до користувацького інтерфейсу для виконання основних операцій у системі.

Були стаорені прототипи наступних екранів:

- головна сторінка;
- розділ збереження результатів тестування;
- пошук;
- статус сервісів користувача;
- нотифікації про помилку;
- редагування атрибутів.

РОЗДІЛ 5
АНАЛІЗ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ

5.1 Предметно-орієнтоване проектування

Предметно-орієнтоване проектування (domain-driven design) - це підхід до моделювання складного об'єктно-орієнтованого програмного забезпечення.

Предметно-орієнтоване проектування не є конкретною технологією чи методологією. DDD – це набір правил, які дозволяють приймати правильні проектні рішення. Даний підхід дозволяє значно прискорити процес проектування програмного забезпечення у незнайомій предметній галузі [11]. Основною метою DDD є отримання якісної моделі ПЗ, яка буде максимально точно відображати поставлені бізнес-цілі.

Підхід DDD особливо корисний у ситуаціях, коли розробник не є фахівцем у галузі продукту, що розробляється. Наприклад: програміст неспроможний знати всі області, у яких потрібно створити ПО, але за допомогою правильного уявлення структури, у вигляді предметно-орієнтованого підходу, може легко спроектувати додаток, грунтуючись на ключових моментах і знаннях робочої області.

Ключовим елементом DDD є обмежений контекст. Обмежений контекст (Bounded Context) – це явна межа, всередині якої існує модель предметної області. Саме на основі контекстів можна розділити код на модулі/пакети/компоненти таким чином, щоб зміни в кожному з них мали мінімальний (або нульовий) вплив на інших. Для кожної системи існує 3 головних обмежених контекста, які описані в розділі 1.1.

5.2 Java

Java – одна з найбільш розповсюджених мов програмування серед написаних на ній Back-End додатків.

Java код найчастіше переводиться в байт-код, тому Java системи можуть працювати на будь-якій платформі, для якої існує реалізація віртуальної Java-машини [12].

Віртуальна машина Java (JVM) – є основною частиною виконавчої системи Java, Java Runtime Environment (JRE). JVM виконує байт-код Java, який створюється з Java-коду Java компілятором (javac).

Javac входить до складу Java Development Kit. JDK – це комплект для розробки на мові Java. JDK також включає в себе стандартні бібліотеки Java класів, документацію, та JRE.

Java має велику кількість стандартів, тому розроблені системи на Java зрозумілі навіть для початківців. Java орієнтована на розробку Open Source, через це можна легально використовувати багато безкоштовних програм на Java.

5.3 Docker

Docker — програмне забезпечення для автоматизації розгортання та керування програмами в середовищах з підтримкою контейнеризації. Дозволяє «упакувати» додаток з усім його оточенням та залежностями у контейнер, який може бути розгорнутий на будь-якій Linux-системі, а також надає набір команд для керування цими контейнерами [13].

Основні можливості Docker:

- розміщення в ізольованому оточенні різних комбінацій виконуваних файлів, бібліотек, файлів конфігурації, та скриптів;
- підтримка роботи на комп’ютерах на базі архітектури x86_64 з системою на базі ядра Linux;
- можливість роботи в оточеннях великих дистрибутивів Linux;

- використання контейнерів для ізоляції процесів від інших процесів і основної системи;
- ізоляція на рівні файлової системи, тобто кожен процес виконується у окремій кореневій файловій системі;
- ізоляція ресурсів: споживання системних ресурсів, таких як витрата пам'яті і навантаження на CPU, можуть обмежуватися окремо для кожного контейнера;
- ізоляція на рівні мережі: кожен ізольований процес має доступ тільки до пов'язаного з контейнером мережевого простору імен, включаючи віртуальний мережевий інтерфейс і прив'язані до нього IP-адреси.

5.4 Kubernetes

Kubernetes - це відкрита система автоматичного розгортання, масштабування та управління застосунками у контейнерах. Система підтримує ряд інструментаріїв з управління контейнерами, у тому числі Docker [14].

Kubernetes Engine це продукт Google Cloud Platform.

Google Cloud Platform (GCP) — набір хмарних служб Google, що покращують управління даними та зменшують додаткові витрати на управління інфраструктурою, підтримку серверів та налаштування мереж.

Окрім інструментів для керування, також надається ряд модульних хмарних служб, таких як обчислення, зберігання даних, аналіз даних та машинне навчання.

Google Cloud Platform надає такі оточення як інфраструктура як послуга, платформа як послуга та безсерверні обчислення [15].

Kubernetes вводить ряд понять:

Вузол (node) – це окрема фізична або віртуальна машина, на якій розгорнуті та виконуються контейнери додатків. Кожен вузол у кластері містить послуги для запуску додатків у контейнерах (наприклад Docker), а також компоненти, призначені для управління вузлом.

Под (pod) - базова одиниця для запуску та управління додатками: контейнери, яким гарантований запуск на одному вузлі, забезпечується поділ ресурсів і міжпроцесорна взаємодія і надається унікальна у межах кластера IP-адреса.
Том (volume) - загальний ресурс зберігання для спільного використання контейнерів, розгорнутих в межах одного пода.

Kubernetes надає:

- моніторинг сервісів та розподіл навантаження. Kubernetes може виявити контейнер, використовуючи ім'я DNS або власну IP-адресу. Якщо трафік у контейнері високий, Kubernetes може збалансувати навантаження та розподілити мережний трафік, щоб розгортання було стабільним;
- автоматичне розгортання та відкати. Використовуючи Kubernetes, можна описати бажаний стан розгорнутих контейнерів та змінити фактичний стан на бажаний;
- автоматичне розподілення навантаження. Можна вказати Kubernetes, скільки ЦП та пам'яті (ОЗУ) потрібно кожному контейнеру. Kubernetes може розмістити контейнери на вузлах так, щоб найбільш ефективно використовувати ресурси;
- самоконтроль. Kubernetes перезапускає контейнери, що відмовили, замінює і завершує роботу контейнерів, які не проходять певну користувачем перевірку працездатності, і не показує їх клієнтам, поки вони не будуть готові до обслуговування;
- керування конфіденційною інформацією та конфігурацією. Kubernetes може зберігати та керувати конфіденційною інформацією, такою як паролі, OAuth-токени та ключі SSH. Можна розгортати та оновлювати конфіденційну інформацію та конфігурацію програми без змін образів контейнерів та не розкриваючи конфіденційну інформацію у конфігурації стека.

5.5 Selenium-Grid

Selenium Grid — це сервер, який дозволяє тестам використовувати екземпляри веб-браузера, запущені на віддалених машинах. Selenium Grid має такі складові, як Hub та Node.

У Selenium Grid один сервер виконує роль центрального центру (хаб). Hub - це центральна точка, яка приймає запити та спрямовує їх до Node. У Grid може бути тільки один Hub. Тести зв’язуються з центром, щоб отримати доступ до екземплярів браузера. Хаб має список серверів, які надають доступ до екземплярів браузера (вузлів WebDriver), і дозволяє тестам використовувати ці екземпляри [16].

Node – це Selenium інстанс, який запускатиме команди, що завантажуються в Hub. Node може бути багато у гриді. Node – може запускатися на різних операційних системах із різними браузерами.

Selenium Grid дозволяє запускати тести паралельно на кількох машинах і централізовано керувати різними версіями браузера та конфігураціями браузера (замість кожного окремого тесту).

5.6 Cucumber

Cucumber — це програмний інструмент, який підтримує розробку, орієнтовану на поведінку (BDD) [17].

Центральним у підході Cucumber BDD є його звичайний аналізатор мови під назвою Gherkin. Gherkin дозволяє описувати очікувану поведінку програмного забезпечення мовою, зрозумілою клієнтам. Таким чином, Cucumber дозволяє виконувати функціональну документацію, написану діловим текстом. Cucumber часто використовується для тестування програмного забезпечення. Він запускає автоматизовані тести, написані у стилі BDD.

Cucumber підтримує різноманітні мови програмування за допомогою різних реалізацій, включаючи Java та JavaScript.

Cucumber тести поділяються на окремі функції (features). Ці функції підрозділяються на сценарії, які є послідовністю кроків.

Feature — це варіант використання, який описує конкретну функцію програмного забезпечення, що тестується. Функція складається з трьох частин: ключове слово Feature, назва функції, додатковий опис у наступних рядках.

Кожна функція складається з набору сценаріїв. Сценарій — це послідовність дій, що відображає 1 на 1 послідовність коду певної функції.

Gherkin – це мова, яку Cucumber використовує для описання тестових випадків. Він розроблений так, щоб бути нетехнічним і читабельним, і описує випадки використання системи.

Метою синтаксису Gherkin є просування поведінкових методів розробки (BDD) для всієї команди, включаючи бізнес-аналітиків та менеджерів. Також синтаксис мови Gherkin призначений для забезпечення простої документації коду, що тестується.

5.7 Figma

Figma – це сервіс для розробки інтерфейсів та прототипування з можливістю організації спільної роботи в режимі реального часу. З сервісом можливо працювати онлайн у браузері та у клієнтському додатку.

Редактор Figma підходить для створення простих прототипів, та для створення складних прототипів, таких як мобільні додатки чи портали. У Фігмі можна відобразити елементи інтерфейсу, створити інтерактивний прототип сайту та програми, ілюстрації, векторну графіку [18].

Прототип – це модель сайту або додатку. За допомогою прототипу замовнику простіше оцінити, як буде виглядати продукт та як люди користуватимуться ним. Щоб створити прототип сайту, дизайнер малює екрани та створює зв'язки між ними.

Фігма надає можливість відкрити та відобразити дизайн на екрані смартфона, планшета та інших пристроїв.

Фігма має основні інструменти для роботи з векторними об'єктами. Редактор дозволяє експортувати дизайн у формат SVG, імпортувати векторні об'єкти з Adobe Illustrator або редактора Sketch.

Головні переваги Фігма над іншими редакторами:

- кросплатформність. Працювати в сервісі можна як з браузера, так і з користувацького додотку будь-якої операційної системи;
- хмарний сервіс. Результати роботи Фігма зберігаються у хмарі, тому їх бачать усі члени команди;
- зворотній зв'язок. До макету у Фігма члени команди можуть залишати коментарі та отримувати відгук від колег;
- підтримка пристроїв. Дизайн Фігма можна відобразити не тільки на комп’ютерах, але й на мобільних пристроях на планшетах.

Висновок до 5 розділу

В даному розділі було проаналізовано технології для реалізації тестування, а саме:

- DDD – методологія моделювання роботи;
- Java – мова програмування, на якій було написано тестовий проект;
- Docker – технологія контейнеризації, що допомагає збирати образи системи, що надаються користувачу;
- K8S – система розгортання кластеру (інфраструктури проекту);
- Selenium Grid – застосунок, що дозволяє запускати UI тести у певному браузері, певної платформи;
- GCP – хмарні середовища. Цільове місце розгортання інфраструктури;
- Cucumber – фреймворк для організації тестування;
- Figma – редактор для створення прототипу продукту.

За допомогою розглянутих технологій було розроблено масштабуємий, простий й функціональний проект крос-браузерного та крос-платформного тестування.

Розглянуто деталі і переваги використання DDD у проектуванні.

Наведено детальний опис технологій контейнерезації й розгортання застсоунку.

Вибір правильних технологій дозволить створювати застосунок легше й краще.

РОЗДІЛ 6
ОПИС ПЛАТФОРМИ

6.1 Функціональний опис платформи

На рисунку 6.1 зображено use-case діаграму платформи.

Діаграма варіантів використання (use-case diagram) – показує систему, людей, організації чи інші системи, які взаємодіють з системою, також, діаграма показує основні функції, які людина виконує з додатоком [19].

Першою дією користувач заповнює базову інформацію про себе для створення його аканту, щоб активувати пробний період. В результаті цієї операції користувач отримує у користування найпростіший кластер, що є безкоштовним, для того, щоб користувач зміг спробувати застосунок на його власних прикладах.

Наступні дні користувач буде тестувати систему, чи задовольняє вона його потреби.

Далі, якщо в користувача є бажання повноцінно використовувати застосунок, він має обрати 1 з наявних моделей підписки. За бажанням, клієнт може звернутися у службу підтримки для отримання спеціальної пропозиції. Такі пропозиції робляться великим клієнтам, що планують запускати хоча б 1 млн тестів щомісяця.

Після покупки підписки цей користувач стає Organization Owner й Billing Manager, й набуває особливих прав. Він є Organization Admin й може давати цю роль іншим. В свою чергу Admin може запрошувати інших учасників до роботи в організації – це дасть їм доступ до корпоративного кластеру.

Корпоративні користувачі можуть запускати тести на корпоративному кластері й бачити стан їх системи у реальному часі.

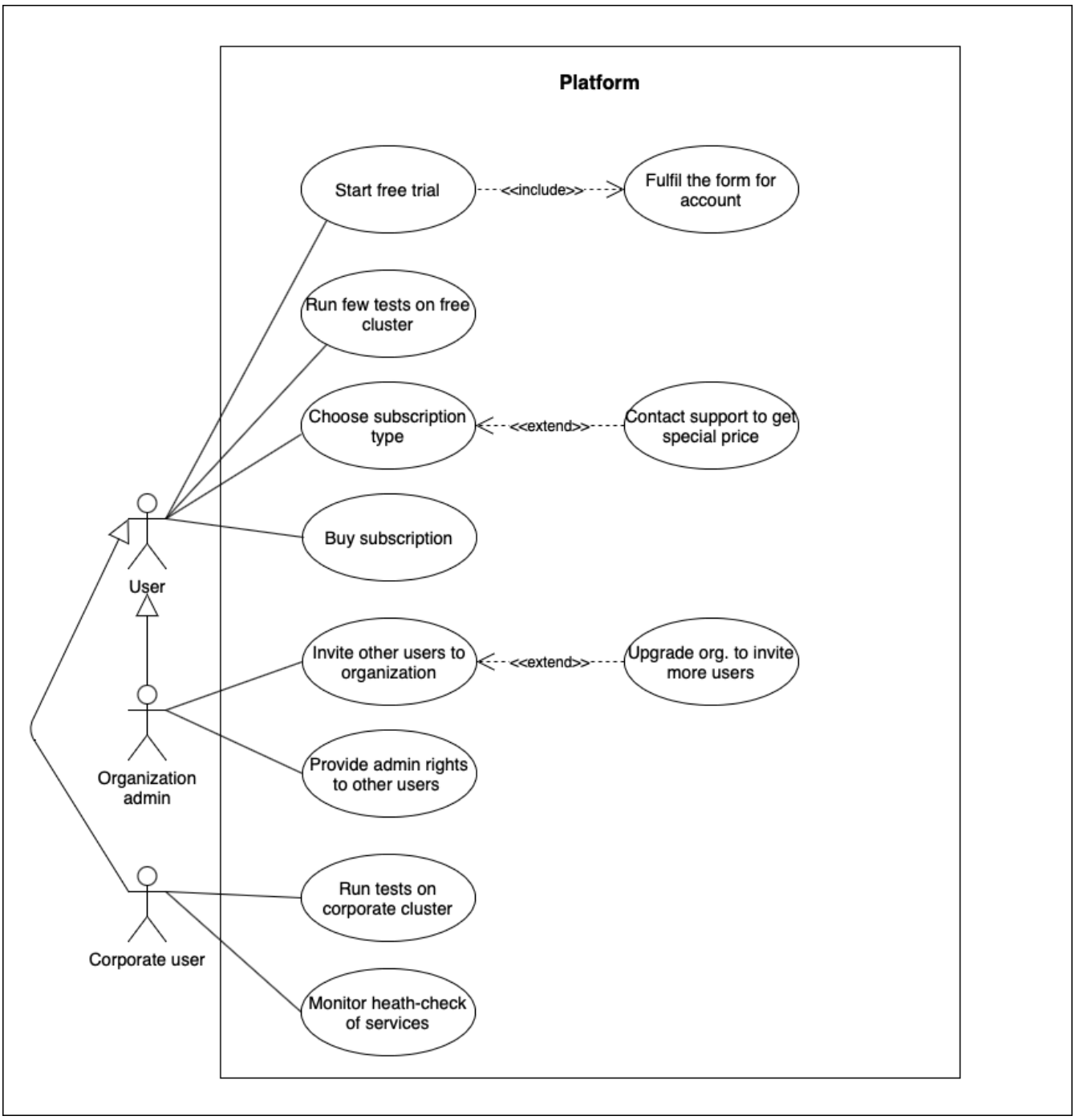

Рис. 6.1. Use-case діаграма застосунку

На рисунку 6.2 представлена flowchart діаграма, що показує послідовність дій.

Flowchart діаграма – це графічне зображення кроків певного процесу. Діаграма ілюструє кроки процесу в послідовному порядку і використовуються для представлення алгоритму чи процесу. Також діаграма широко використовуються для документування, вивчення, планування, покращення та передачі складних процесів у зрозумілих діаграмах [20].

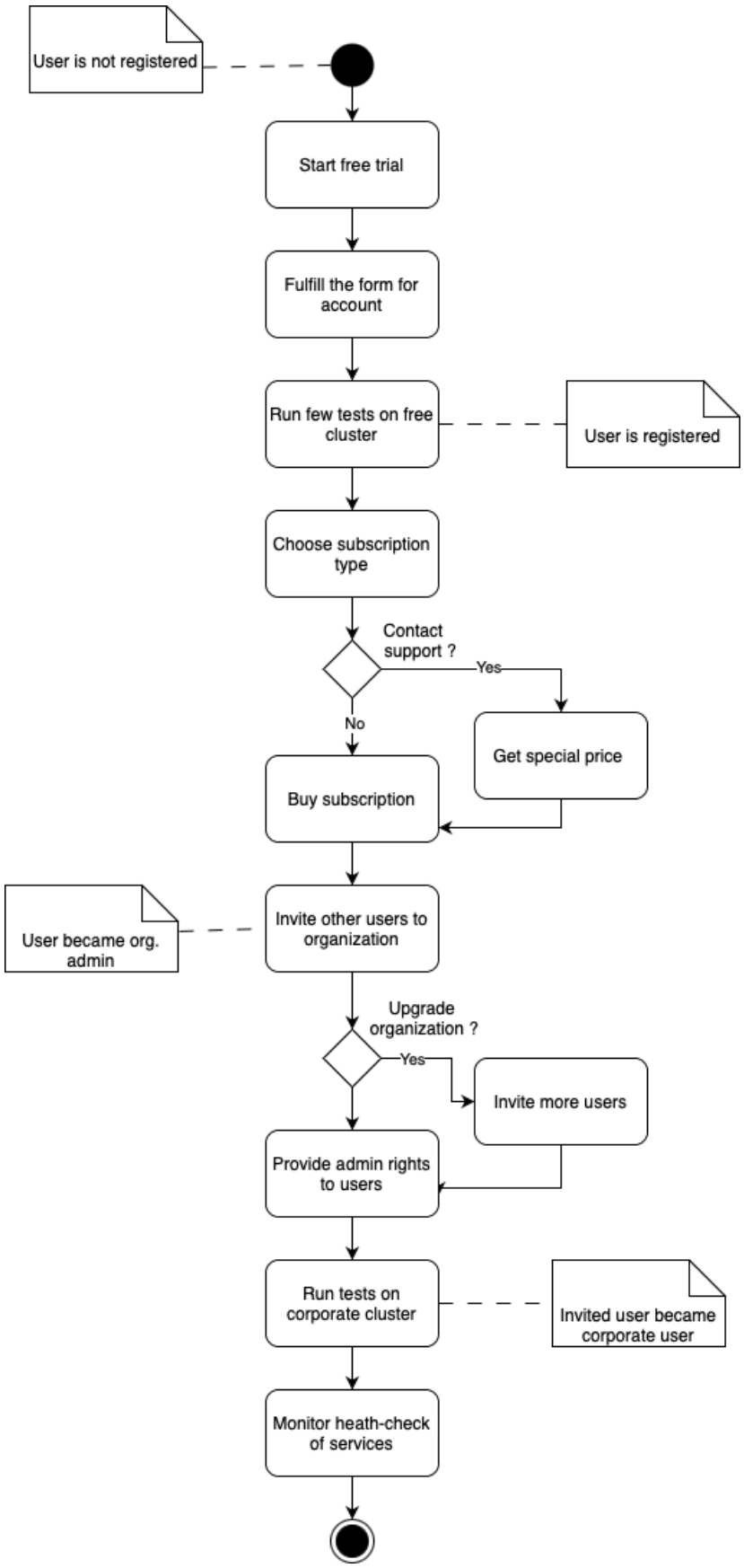

Рис. 6.2. Flowchart діаграма

6.2 Нефункціональний опис платформи

На рисунку 6.2 показано інфраструктурну схему платформи. Важливо розрізняти, що дана інфраструктура необхідна самій платформі для роботи й опрацювання клієнтських запитів з UI. Інфраструктура не надається у використання користувачам. Огляд користувацької орендної інфраструктури буде продемонстровано далі.

Інфраструктура платформи організована у event-driven стилі, що дає змогу системі добре масштабуватися.

На рисунку показані лише основні підписники подій. Важливо розуміти, що у фінальній стадії продукту їх може бути значно більше, наприклад для аналітики, внутрішнього моніторингу системи чи будь-яких інших цілей.

Організація ПЗ за моделлю Pub/Sub дозволяє додавати необмежену кількість підписників у подальшому й не витрачати час під час написання прототипу на планування того, що може статися через роки.

Код системи здебільше запущено у Cloud Function Engine, що працює й оплачується лише коли використовується. Це найкраща модель для такого застосунку, бо це дозволить зменшити витрати на початковому етапі, коли користувачів буде мало, а зі збільшенням витрат при зростанні навантаження будуть зростати й прибутки з продажу продукту, що компенсує витрати.

Pub/Sub NEW_USER й CF USER_CREATOR відповідальні за обробку use-case, коли користувач заповнив свої данні й хоче отримати безкоштовну підписку. CF повинна створити користувача й зробити запит на безкоштовну підписку.

Pub/Sub NEW_ORGANIZATION відповідальний за створення нової організації у системі, заповнення її юридичних даних, платіжних домовленостей, контактів й певних деталей договору.

SUBSCRIPTION_MANAGER – цей сервіс відповідальний за роботу із підписками: створення нових, запит по оплаті за існуючі, закриття неоплачених, апгрейд існуючих.

Pub/Sub NEW_CLUSTER_REQUEST й CF CLUSTER_PROVIDER безпосередньо запускають процес виділення кластеру користувачеві після успішного створення підписки.

Pub/Sub INVITATION й CF INVITATION_SENDER відповідальні за відправку запрошень вступу у організацію.

API_GATEWAY й CF INVITED_USER_CREATOR відповідальні за обробку прийняття запиту на вступ до організації. CREATOR повинен створити користувача у системі, якщо такого немає, й додати його до відповідної організації.

Pub/Sub USER_STATE й CF USER_STATE_MANAGER відповідальні за обробку змін у акантах користувачів. Зміна особистих даних, засобів оплати, покидання організації за власним бажанням, тощо.

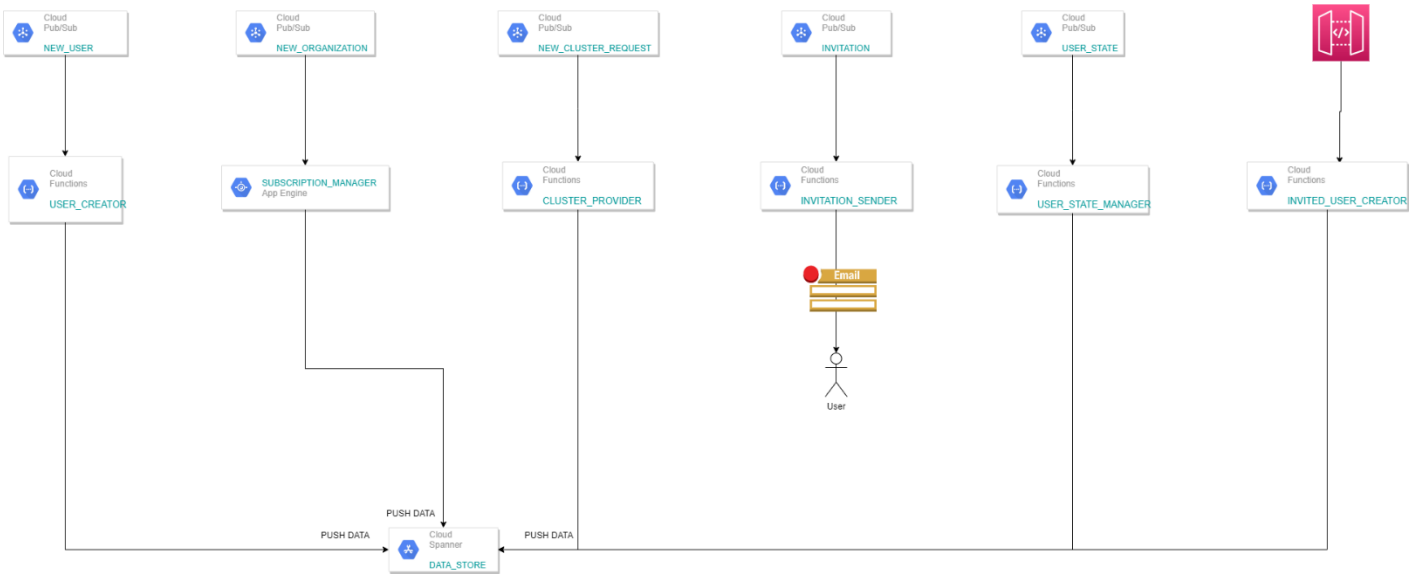

Рис. 6.3. Інфраструктурна схема застосунку

Висновок до 6 розділу

У цьому розділі представлено функціональний, нефункціональний та інфраструктурний описи проекту. Представлено use-case, user-flow та інфраструктурна діаграми.

Виділені основні дії користувача у застосунку, а саме:

- оформлення підписки;
- виконання тестів;
- запрошення інших користувачів до роботи;
- моніторинг стану застосунку.

Вироблено архітектуру застосунку, у event-driven парадигмі. Основним транзакційним сховищем є Pub/Sub, код виконується у GCP CF, що дозволяє заощадити кошти.

РОЗДІЛ 7
ОПИС ПРОЦЕСУ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ

7.1. Реалізація ECC

Перше, що треба зазначити – це те, що система повністю реалізовує 5 Essentials Cloud Characteristics [2], тому може назватись хмарною платформою. Розглянемо детальніше кожний з них.

On-demand self-service – можливість отримати сервіс без людської комунікації. В розробленому прототипі системи це так й працює. На рисунку 6.3 у попередньому розділі показано місце (CF CLUSTER_PROVIDER) в якому автоматизовано, без людини, починається процес створення кластеру. На цьому етапі аналізуються вимоги до майбутнього кластеру в залежності від підписки користувача й через k8s sdk подається команда на створення.

Broad network access – впроваджені ресурси доступні через мережу й підтримують взаємодію з основними клієнтами по взаємодії. Користувач розробленої платформи буде керувати своєю організацією через 2 основних інтерфейса взаємодії: UI платформи та розробленої SDK для інтеграцій.

Resource pooling – характеристика при якій обчислювальні потужності провайдера можуть обслуговувати відразу декількох користувачів, з динамічною прив’язкою ресурсів до користувача, коли користувач не має впливу на фактичну інфраструктуру, а лише на високому рівні може налаштовувати деякі її параметри (такі як локація, потужність, тощо). В розробленому прототипі реалізована ця можливість. Кластер може бути як виділеним для одного клієнта, так і загальним для декількох, звісно від цього залежить й ціна. Крім того, навіть прив’язані до одного користувача кластери, насправді, працюють динамічно, вони заздалегідь

купуються у хмарних провайдерів, таких як AWS чи GCP, по вигідній ціні (за підпискою на 3 роки), а потім динамічно прив’язуються до користувачів в залежності від потреб.

Rapid elasticity – ресурси можуть бути еластично додані й звільнені, у деяких випадках автоматично, користувач може розраховувати, що ресурси провайдера необмежені й він може отримати їх у будь-який момент.

Measured service – використання ресурсів контролюється й фіксується, відповідно ціна може бути виставлена по моделі – “плати, за те що використовуєш”. Процес ціноутворення платформи було розглянуто у розділі 3, проте дійсно, для є підписки, що працюють за цією моделлю – ціна виставляється в залежності від використаних ресурсів, а не фіксовано.

7.2. Опис інфраструктури кластеру

На рисунку 7.1 представлено схему розгортання кластеру, який надається у користування клієнту.

Першою дією створюється k8s master node – вузел, що керує усім кластером, через нього відбувається керування, моніторинг та доступ до кластеру.

Другою дією створюється певна метадата (дані кластеру записуються у сховище).

Третьою дією створюється Deployment і сервіс, що має назву Selenium-Grid master, важливо розрізняти цей master й master з першої дії.

Перший майстер керує k8s кластером, до складу якого входить й Selenium-Grid master. Selenium-Grid master керує Selenium кластером, в більшості випадків, це означає пересилання трафіку на інстанс з відповідною ОС та браузером, крім того, через master йде моніторинг роботи Selenium та браузерів.

Четвертою дією розгортається Cluster Monitoring Tool. На версії прототипу використовується продукт Lens [21], проте це може бути змінено на вимогу клієнта.

Наступними кроками створюється група адміністраторів платформи, що мають доступ до керування інфраструктурою, й група користувачів, що отримує певні

права по моніторингу їх кластера (зазвичай ця інформація потрібна для відслідковування стану кластера й його поточного навантаження).

Далі відбувається створення Deployment, Service та декількох Pod сервісу Selenium-Grid worker. Pre-init контейнер цих Pod завантажує дані зі сховища стосовно Selenium-Grid master (адрес для реєстрації) й вимоги до браузера. Далі відбувається автоматичне завантаження браузера та підключення до master.

Після успішного завершення цих дій кластер стає готовим для використання – відповідна інформація відправляється клієнту.

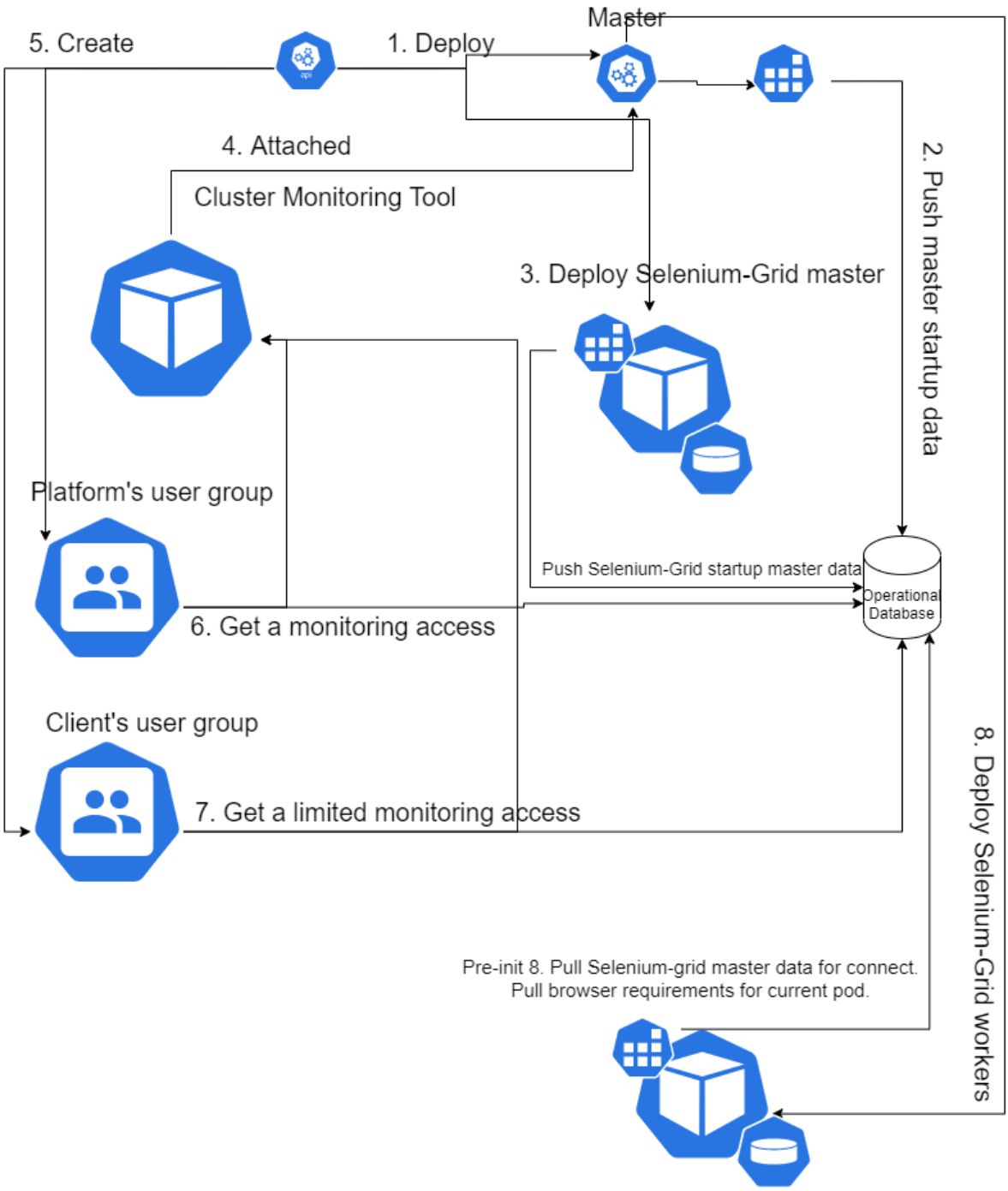

Рис. 7.1. Інфраструктура кластеру

7.3 Приклад функціонування

Приклад буде наведено на найбільшому українському порталі новин UKR.NET. Функціональні тести будуть перевіряти наступні сценарії:

– перевірка, що головна сторінка відчиняється;

- перевірка, що розділ погода працює й можна перейти на сайт sinoptic.ua й перевірити погоду на сьогодні;
- перевірка, що вкладка обміну валюти – працює й показує курс валют на сьогодні;
- перевірка, що ціни на бензин та газ відображаються у вкладці пальне.

Звісно, що це не повна функціональна перевірка цього сайту, але ці приклади допоможуть показати функціональні можливості бібліотеки.

Важливо розуміти й саму суть перевірки. Зрозуміти, що сайт UKR.NET працює, чи ні, з функціональної точки зору може людина, яка ним користувалася, й має користувацький досвід, або добре натренована система комп'ютерного зору й прийняття рішень.

Оскільки розроблена система є повністю автоматизованою, то варіант з людиною відразу є неможливим, а розробка подібної системи зору й ML, буде просто занадто дорогою для більшості користувачів. Такі системи вже існують – це було детально описано в 1 розділі роботи, й мета цього застосунку - це надавати можливості для перевірки health-check дешевше. Тому для перевірки будуть використані певні графічні елементи, наявність й коректне функціонування яких, буде свідчити про добрий стан здоров'я сервісу.

На рисунках 7.2 й 7.3 продемонстровано тестування головної сторінки порталу новин. Це можна зрозуміти по надпису "Chrome is being controlled by automated test software", тобто усі дії там відбуваються автоматизовано. Цей тест фактично перевіряє наявність на сторінці у правильному місці (CSS атрибут) картинки UKR.NET із посиланням всередині.

Тест натискає на цю картинку, й перевіряє, що він повернеться на сайт UKR.NET.

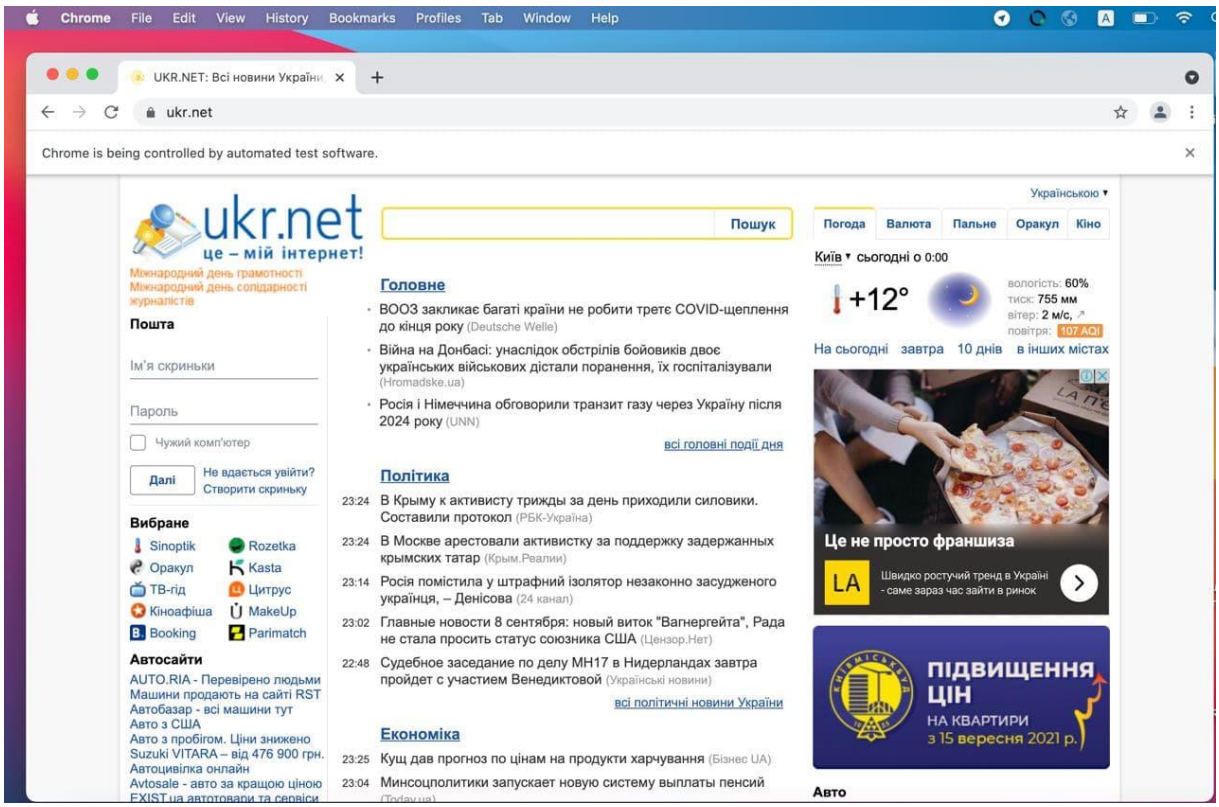

Рис. 7.2. Перевірка головної сторінки у браузері Chrome 93

Рис. 7.3. Перевірка головної сторінки у браузері Firefox 92

На рисунках 7.4 та 7.5 продемонстровано перевірка вкладки пальне. Для цього тест знаходить правий розділ керування вкладками, шукає в ньому вкладку пальне

й натискає на неї, перевіряє, що з’являється табличка із даними про пальне. Пошук відбувається по DOM за допомогою CSS Path.

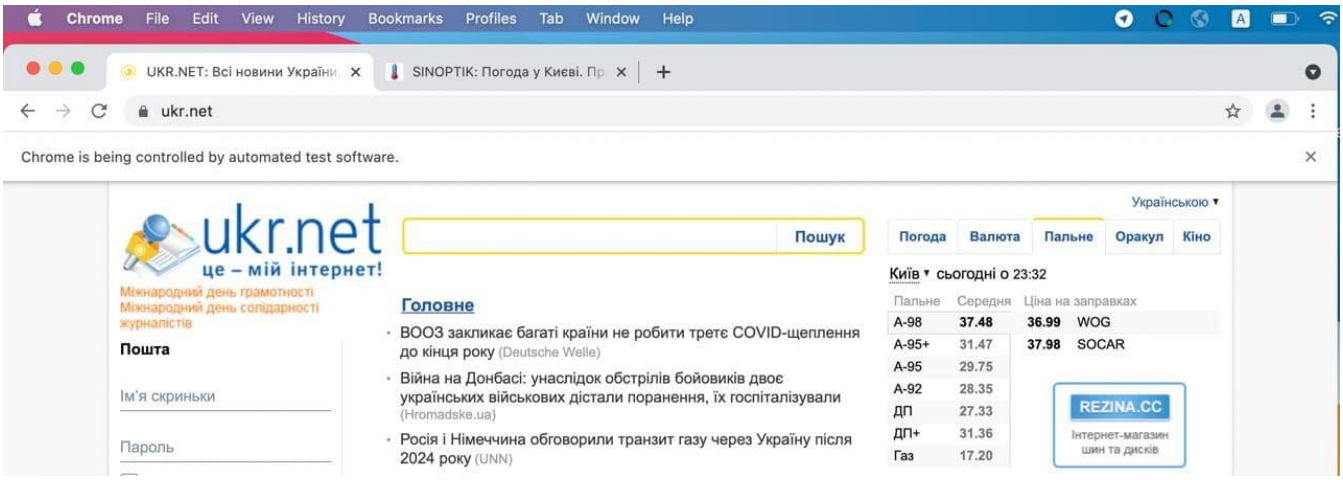

Рис. 7.4. Перевірка цін на пальне у мережах АЗС у браузері Chrome 93

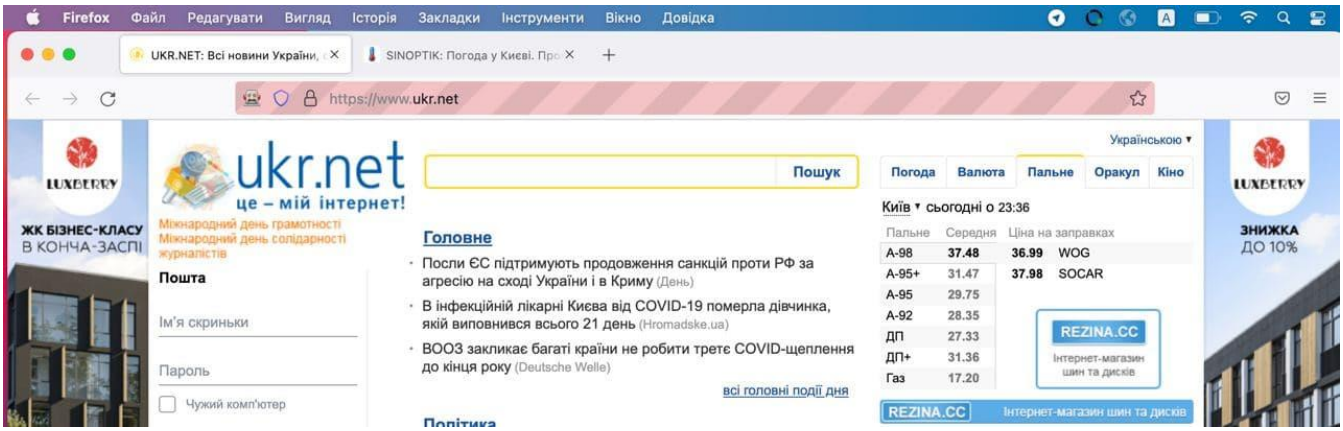

Рис. 7.5. Перевірка цін на пальне у мережах АЗС у браузері Firefox 92

На рисунках 7.6 та 7.7 показано можливість перевірки переходів на інший ресурс, перевірки й повернення. У прикладі зроблено перехід з UKR.NET на вкладку sinoptik.ua, звідки береться поточна погода, перевірка контенту й перехід назад.

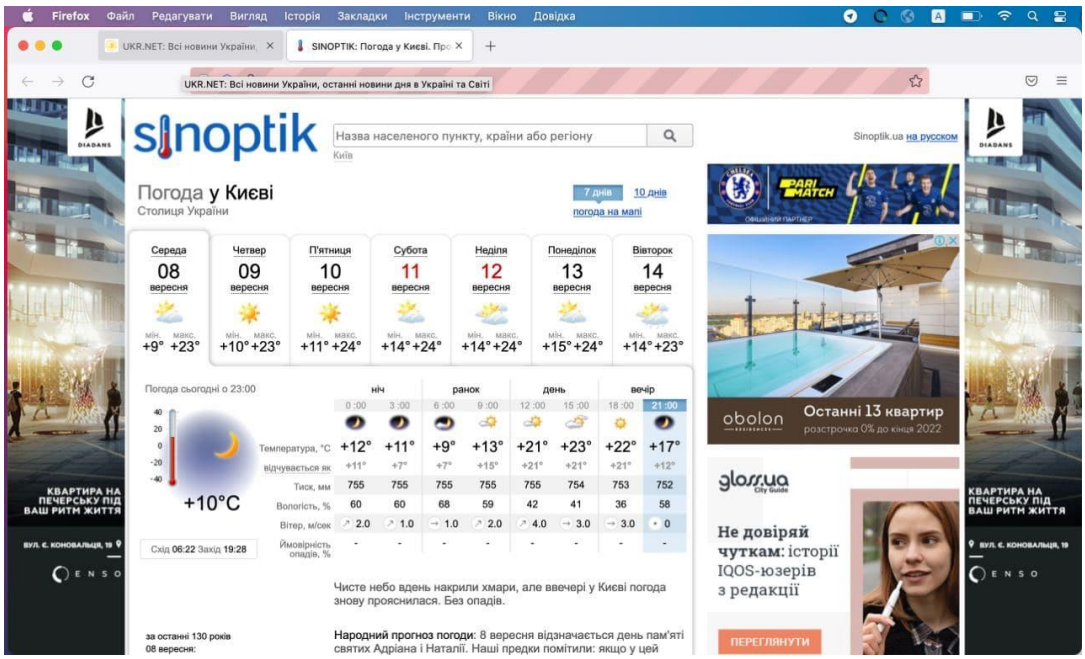

Рис.7.6. Перевірка з переходом на інший ресурс у браузері Chrome 93

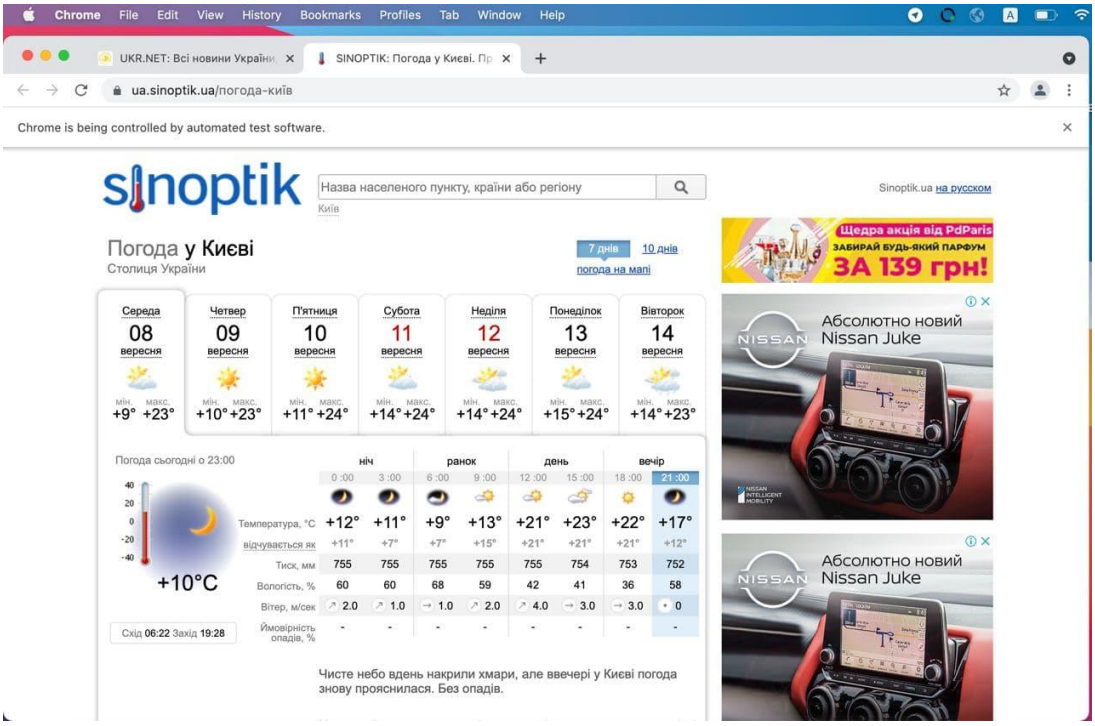

Рис.7.7. Перевірка погоди з переходом на інший ресурс у браузері Firefox 92

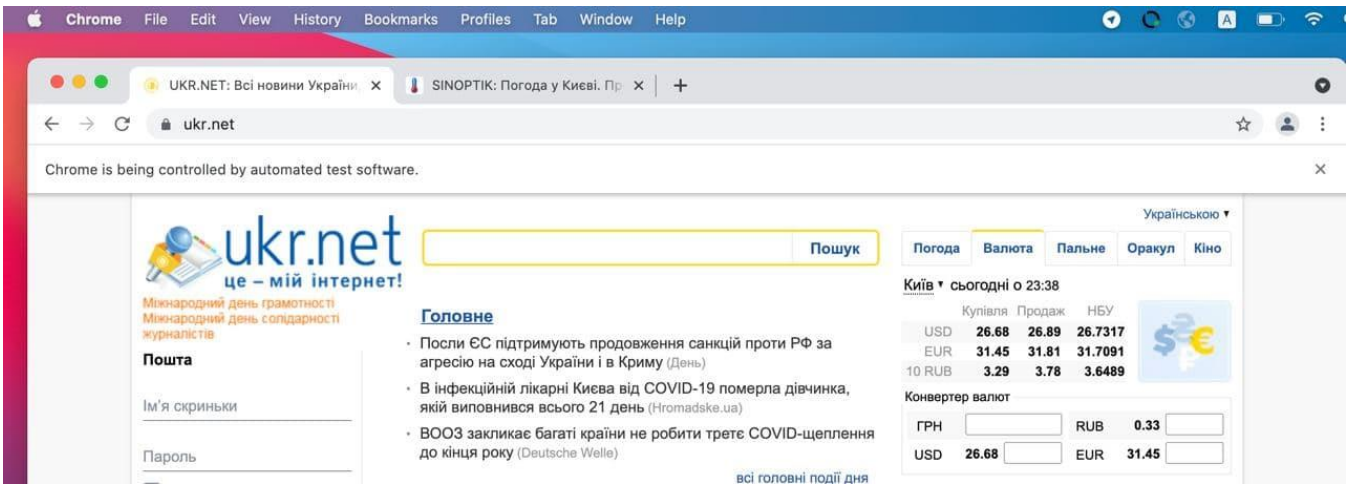

Рис.7.8. Перевірка валют у браузері Chrome 93

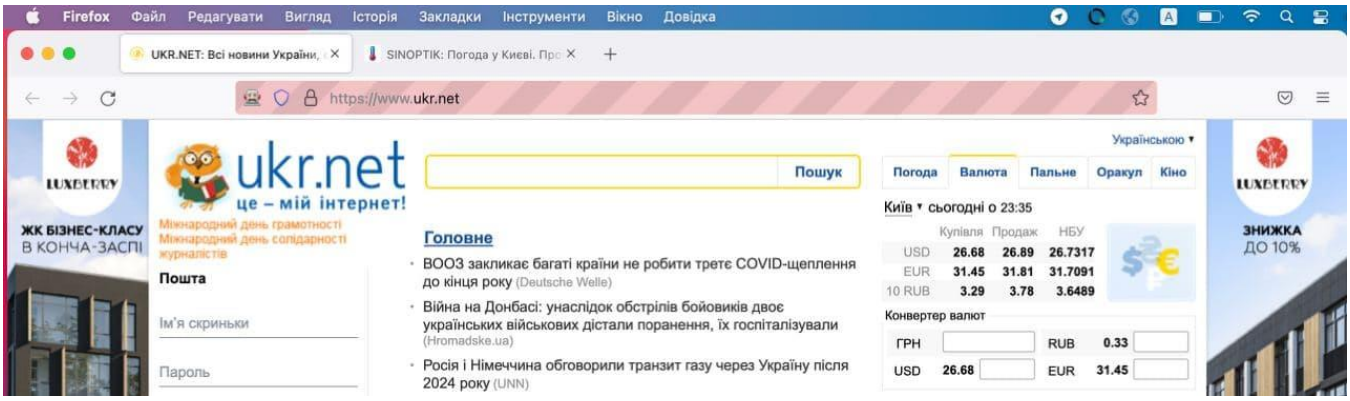

Рис. 7.9. Перевірка валют у браузері Firefox 92

Важливо зазначити, що для клієнта виконання цих операцій у різних браузерах виглядає дуже просто. Клієнт змінює 2 параметри в запиті, browserName та browserVersion. Вся робота по опрацюванню відбувається на стороні платформи.

Для того, щоб обробити цей запит, заздалегідь підняті потрібні версії браузеру (підтримувані версії зазначаються у переліку на сайті, або узгоджуються окремо із клієнтом) на потрібних ОС, робота інстансів оптимізується, щоб вони не простоювали.

Оптимізацією на низькому рівні (Service pod) виконує K8S. Для цього система збирає середній показник CPU Utilization за попередній період, й використовує ці показники для розрахування граничних параметрів Pod для цього кластеру у майбутньому, що значно покращує Auto-scaling.

Для K8S планувальника дуже важливо знати приблизні показники навантаження на Pod, для того, щоб планувати масштабування системи (запобігати стрибкам навантаження, та зменшувати кількість вільних ресурсів – перекидати їх на потреби інших сервісів).

Кожний інстанс у кластері створюється за допомогою Docker. Система підтримує усі основні ОС, проте процес їх ініціалізації є однаковим – береться вже налаштований батьківський образ Docker image, і на нього завантажується потрібний браузер й потім встановлюється й підключається до master. Відмінності між платформами полягають лише у невеликих відмінностях синтаксису терміналу системи.

Висновок до 7 розділу

У даному розділі наведено опис шляхів реалізації розробленою платформою 5 ECC.

Наведено опис внутрішньої архітектури, що передаються у користування кінцевим користувачам. Наведено описи розподілу повноважень у кластері й засобів моніторингу.

Наведено приклад функціонування платформи на двох різних ОС (Linux та MacOS) й у двох різних браузерах (Chrome та Firefox) на прикладі популярного українського сайту новин UKR.NET, а саме розроблені перевірки:

- доступу до головної сторінки;
- розділу погоди, пального, обміну валют;
- можливості переходу між різними табами сайту;
- можливість переходу за посиланням на інші ресурси й правильність цільових посилань.

Наведено опис як масштабувати систему для підтримки інших ОС та браузерів – цей процес є автоматизованим на рівні збірки Docker image.

ВИСНОВКИ

В дипломній роботі проаналізовано проблеми функціонального крос-платформного та крос-браузерного тестування й критичні вимоги до систем у цій галузі. Було визначено проблеми, з якими стикаються компанії в залежності від сфери роботи.

Детально розглянуто існуючі рішення і виявлено, що існуючі рішення можна поділити на наступні сервіси:

- створення системи забезпечення якості та процесів тестування з нуля, за допомогою платформи вендора, тренерів, спеціалістів з інтеграції, їх розробників;
- платформа плюс допоміжний персонал по інтеграції;
- code-less платформа (не вимагає навичок програмування чи налаштувань від користувача);
- платформа;
- продукт, розгорнутий на інфраструктурі клієнта;
- бібліотека.

Було виділено пріоритети користувачів систем:

- стабільність системи та її результатів;
- ціна;
- безпека;
- рівень автоматизації створення коду, розгортання серверів, підтримки;
- codeless системи.

Була запропонована покрокова послідовність дій стоврення кращої платформи для тестування, а саме:

- формування обмежених бізнес контекстів;
- формування єдиної мови бізнесу;
- відокремлення смислового ядра продукту;

– створення специфікації;
– створення прототипу;
– TCO аналіз.

Розроблена цінова політика, що дозволить застосунку бути конкурентоспроможним на ринку подібних систем.

Були розглянуті моделі оплати для користувачів системи. Розглянуті можливості оптимізації витрат користувачем шляхом оформлення довготривалої підписки.

Розглянуто засоби досягання зменшення собівартості системи для надання можливості знижки користувачам, що оформлять довготривалу підписку.

У ході прототипування було покрито функціональні вимоги до користувацького інтерфейсу для виконання основних операцій у системі, зокрема були розроблені наступні екрани:

– головна сторінка;
– розділ збереження результатів тестування;
– пошук;
– статус сервісів користувача;
– нотифікації про помилку;
– редагування атрибутів.

Проаналізовано технології для реалізації тестування, а саме:

– DDD – методологія моделювання роботи;
– Java – мова програмування, на якій було написано тестовий проект;
– Docker – технологія контейнеризації, що допомагає збирати образи системи, що надаються користувачу;
– K8S – система розгортання кластеру (інфраструктури проекту);
– Selenium Grid – застосунок, що дозволяє запускати UI тести у певному браузері, певної платформи;
– GCP – хмарні середовища. Цільове місце розгортання інфраструктури;
– Cucumber – фреймворк для організації тестування;
– Figma – редактор для створення прототипу продукту.

За допомогою розглянутих технологій було розроблено масштабуємий, й простий прототип платформи для крос-браузерного та крос-платформного тестування.

Виділені основні дії користувача у застосунку, а саме:

- оформлення підписки;
- виконання тестів;
- запрошення інших користувачів до роботи;
- моніторинг стану застосунку.

Вироблено архітектуру застосунку, у event-driven парадигмі.

Наведено опис внутрішньої архітектури, що передається у користування кінцевим користувачам та опис розподілу повноважень у кластері й засобів моніторингу.

Наведено приклад функціонування платформи на двох різних ОС (Linux та MacOS) й у двох різних браузерах (Chrome та Firefox) на прикладі популярного українського сайту новин UKR.NET, а саме розроблені перевірки:

- доступу до головної сторінки;
- розділу погоди, пального, обміну валют;
- можливості переходу між різними табами сайту;
- можливість переходу за посиланням на інші ресурси й правильність цільових посилань.

Наведено опис як масштабувати систему для підтримки інших ОС та браузерів – цей процес є автоматизованим на рівні збірки Docker image.

Серед основних переваг системи можна виділити її меншу ціну порівняно з існуючими аналогами, орієнтованість на хмарні застосунки й сучасність.

Серед недоліків прототипу платформи можна виділити необхідність просування й конкуренцію на високо-конкурентному ринку, що зробить задачу пошуку клієнтів більш важкою, а термін повернення інвестицій – довшим.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вон Вернон Предметно-орієнтоване проектування: Вид-во “Вільямс”, 2016. – 688 с.
2. IaaS vs PaaS va SaaS [Electronic resource] – Access mode: https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas?sc_cid=7013a000002pgRcAAI&gclid=CjwKCAiA1aiMBhAUEiwACw25MURYnJhVUMe9uvA2yf_R8B4QbbCVqeahwPHH4ldi4DwSAAB5lX7BBoCQMQQAvD_BwE&gclsrc=aw.ds (last access: 13.09.2021 p). – Title from the screen.
3. Sauce Labs [Electronic resource] – Access mode: https://saucelabs.com (last access: 13.09.2021 p). – Title from the screen.
4. Functionize [Electronic resource] – Access mode: https://www.functionize.com (last access: 15.09.2021 p). – Title from the screen.
5. UiPath [Electronic resource] – Access mode: https://www.uipath.com (last access: 15.09.2021 p). – Title from the screen.
6. MicroFocus [Electronic resource] – Access mode: https://www.microfocus.com/en-us/home (last access: 15.09.2021 p). – Title from the screen.
7. Micro Focus Unified Functional Testing [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Micro_Focus_Unified_Functional_Testing (last access: 16.09.2021 p). – Title from the screen.
8. SmartBear [Electronic resource] – Access mode: https://smartbear.com/product/testcomplete/cross-browser-testing-tool/ (last access: 16.09.2021 p). – Title from the screen.
9. Appium [Electronic resource] – Access mode: https://appium.io (last access: 17.09.2021 p). – Title from the screen.
10. Perfecto [Electronic resource] – Access mode: https://www.perfecto.io (last access: 17.09.2021 p). – Title from the screen.
11. Domain-driven design [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Domain-driven_design (last access: 26.09.2021 p). – Title from the screen.

12. Java [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Java_(programming_language) (last access: 02.10.2021 p). – Title from the screen.
13. Docker [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Docker_(software) (last access: 02.10.2021 p). – Title from the screen.
14. Kubernetes [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Kubernetes (last access: 06.10.2021 p). – Title from the screen.
15. GCP [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Google_Cloud_Platform (last access: 11.10.2021 p). – Title from the screen.
16. Selenium Grid [Electronic resource] – Access mode: https://www.selenium.dev/documentation/grid/ (last access: 11.10.2021 p). – Title from the screen.
17. Cucumber [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Cucumber_(software) (last access: 17.10.2021 p). – Title from the screen.
18. Figma [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Figma_(software) (last access: 04.11.2021 p). – Title from the screen.
19. Use case diagram [Electronic resource] – Access mode: https://en.wikipedia.org/wiki/Use_case_diagram (last access: 07.11.2021 p). – Title from the screen.
20. Flowchart diagram [Electronic resource] – Access mode: https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial (last access: 14.11.2021 p). – Title from the screen.
21. Lens [Electronic resource] – Access mode: https://k8slens.dev (last access: 14.11.2021 p.) – Title from the screen.

Додаток А

Лістинг коду тестових методів

docker-entrypoint.sh

```
#!/bin/bash
freeMem=awk '/MemFree/ { print int($2/1024) }' /proc/meminfo
s=$(($freeMem/10*8))
x=$(($freeMem/10*8))
n=$(($freeMem/10*2))
export JVM_ARGS="-Xmn${n}m -Xms${s}m -Xmx${x}m"
java -jar /home/nonroot/sel
```

Dockerfile

```
FROM openjdk:11-jre-slim-buster
#Create non-root user
RUN groupadd nonroot \
        --gid 1501 \
  && useradd nonroot \
        --gid 1501 \
        --create-home \
        --shell /bin/bash \
        --uid 1500
ENV HOME=/home/nonroot
RUN apt-get -y update
RUN apt-get install -y apt-transport-https
RUN apt-get install -qq -y curl unzip

# Install Chrome
ARG CHROME_VERSION="88.0.4324.96-1"
RUN apt-get install -y  gnupg2
# Install wget.
RUN apt-get install -y wget
# Set the Chrome repo.
RUN wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - \
    && echo "deb http://dl.google.com/linux/chrome/deb/ stable main" >>
/etc/apt/sources.list.d/google.list
# Install Chrome.
RUN apt-get install -y gconf-service libasound2 libatk1.0-0 libcairo2 libcups2 libfontconfig1 libgdk-
pixbuf2.0-0 libgtk-3-0 libnspr4 libpango-1.0-0 libxss1 fonts-liberation libappindicator1 libnss3 lsb-
release xdg-utils
# install chrome
RUN wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
RUN dpkg -i google-chrome-stable_current_amd64.deb; apt-get -fy install
# Install Chrome Driver
ARG CHROME_DRIVER_VERSION="88.0.4324.96"
RUN if [ -z "$CHROME_DRIVER_VERSION" ]; \
  then CHROME_MAJOR_VERSION=$(google-chrome --version | sed -E "s/.* ([0-9]+)(\.[0-
9]+){3}.*/\1/") \
```

```
    && CHROME_DRIVER_VERSION=$(wget --no-verbose -O -
"https://chromedriver.storage.googleapis.com/LATEST_RELEASE_${CHROME_MAJOR_VERSION}
"); \
  fi \
  && echo "Using chromedriver version: "$CHROME_DRIVER_VERSION \
  && wget --no-verbose -O /tmp/chromedriver_linux64.zip
https://chromedriver.storage.googleapis.com/$CHROME_DRIVER_VERSION/chromedriver_linux64.zi
p \
  && unzip /tmp/chromedriver_linux64.zip -d /home/nonroot \
  && rm /tmp/chromedriver_linux64.zip

WORKDIR /home/nonroot
COPY selenium-server-standalone-3.141.59.jar ./
COPY docker-entrypoint.sh ./
RUN chown -R nonroot:nonroot /home/nonroot/*
USER 1500:1501
RUN chmod 755 /home/nonroot/docker-entrypoint.sh\
    && chmod 755 /home/nonroot/chromedriver
ENTRYPOINT ["/home/nonroot/docker-entrypoint.sh"]
```

main.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hub
  labels:
    app: hub
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hub
  template:
    metadata:
      labels:
        app: hub
    spec:
      containers:
        - name: app
          image: docker.io/anastasiia12/hub:1.0
          ports:
            - containerPort: 9001
---
apiVersion: v1
kind: Service
metadata:
  name: hub
spec:
```

```
  selector:
app: hub
  type: NodePort
  ports:
    - name: serv
      protocol: TCP
      port: 9001
      targetPort: 9001
```

docker-entrypoint.sh

```
#!/bin/bash
freeMem=awk '/MemFree/ { print int($2/1024) }' /proc/meminfo
s=$(($freeMem/10*8))
x=$(($freeMem/10*8))
n=$(($freeMem/10*2))
export JVM_ARGS="-Xmn${n}m -Xms${s}m -Xmx${x}m"
java -Dwebdriver.chrome.driver="/home/nonroot/chromedriver" -
Dwebdriver.firefox.driver="/home/nonroot/geckodriver" -jar /home/nonroot/selenium-server-
standalone-3.141.59.jar -role webdriver -hub http://172.17.0.3:4444/grid/register/ -port 8082
```

Dockerfile

```
FROM openjdk:11-jre-slim-buster
#Create non-root user
RUN groupadd nonroot \
        --gid 1501 \
  && useradd nonroot \
        --gid 1501 \
        --create-home \
        --shell /bin/bash \
        --uid 1500
ENV HOME=/home/nonroot
RUN apt-get -y update
RUN apt-get install -y apt-transport-https
RUN apt-get install -qq -y curl unzip
# Install Chrome
ARG CHROME_VERSION="88.0.4324.96-1"
RUN apt-get install -y  gnupg2
# Install wget.
RUN apt-get install -y wget
# Set the Chrome repo.
RUN wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - \
    && echo "deb http://dl.google.com/linux/chrome/deb/ stable main" >>
/etc/apt/sources.list.d/google.list
# Install Chrome.
RUN apt-get install -y gconf-service libasound2 libatk1.0-0 libcairo2 libcups2 libfontconfig1 libgdk-
pixbuf2.0-0 libgtk-3-0 libnspr4 libpango-1.0-0 libxss1 fonts-liberation libappindicator1 libnss3 lsb-
release xdg-utils
# install chrome
```

```
RUN wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
RUN dpkg -i google-chrome-stable_current_amd64.deb; apt-get -fy install
# Install Chrome Driver
ARG CHROME_DRIVER_VERSION="88.0.4324.96"
RUN if [ -z "$CHROME_DRIVER_VERSION" ]; \
  then CHROME_MAJOR_VERSION=$(google-chrome --version | sed -E "s/.* ([0-9]+)(\.[0-
9]+){3}.*/\1/") \
    && CHROME_DRIVER_VERSION=$(wget --no-verbose -O -
"https://chromedriver.storage.googleapis.com/LATEST_RELEASE_${CHROME_MAJOR_VERSION}
"); \
  fi \
  && echo "Using chromedriver version: "$CHROME_DRIVER_VERSION \
  && wget --no-verbose -O /tmp/chromedriver_linux64.zip
https://chromedriver.storage.googleapis.com/$CHROME_DRIVER_VERSION/chromedriver_linux64.zi
p \
  && unzip /tmp/chromedriver_linux64.zip -d /home/nonroot \
  && rm /tmp/chromedriver_linux64.zip
WORKDIR /home/nonroot
COPY selenium-server-standalone-3.141.59.jar ./
COPY docker-entrypoint.sh ./
COPY geckodriver ./
RUN chown -R nonroot:nonroot /home/nonroot/*
USER 1500:1501
RUN chmod 755 /home/nonroot/docker-entrypoint.sh\
    && chmod 755 /home/nonroot/chromedriver\
     && chmod 755 /home/nonroot/selenium-server-standalone-3.141.59.jar\
     && chmod 755 /home/nonroot/geckodriver
ENTRYPOINT ["/home/nonroot/docker-entrypoint.sh"]
```

main.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: linux-node
  labels:
    app: linux-node
spec:
  replicas: 1
  selector:
    matchLabels:
      app: linux-node
  template:
    metadata:
      labels:
        app: linux-node
    spec:
      containers:
        - name: app
          image: docker.io/anastasiia12/node:9.0
```

```
 privileged: true
 ports:
          - containerPort: 9001
         securityContext:
          allowPrivilegeEscalation: true
---
apiVersion: v1
kind: Service
metadata:
  name: linux-node
spec:
  selector:
    app: linux-node
  ports:
    - protocol: TCP
      port: 9001
      targetPort: 9001
```

DriverType

```
public enum DriverType {
    CHROME,
    FIREFOX,
    EDGE,
    REMOTE
}
```

Main

```
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.remote.RemoteWebDriver;

import java.net.MalformedURLException;

public class Main {

    private static final String MAIN_PAGE_IMAGE_XPATH =
"//*[@id=\"main\"]/div[2]/header/section[1]/a";
    private static final String WEATHER_PAGE_LINK_XPATH =
"//*[@id=\"main\"]/div[3]/div/div[1]/section/ul/li[1]";
    private static final String WEATHER_PAGE_IMAGE_AND_LINK_TO_SINOPTIC_XPATH =
"//*[@id=\"sinoptik_container\"]";
    private static final String CURRENCY_PAGE_LINK_XPATH =
"//*[@id=\"main\"]/div[3]/div/div[1]/section/ul/li[2]";
    private static final String CURRENCY_TABLE_XPATH = "//*[@id=\"currency\"]/div[4]";
```

```
    private static final String GAS_PAGE_LINK_XPATH =
"//*[@id=\"main\"]/div[3]/div/div[1]/section/ul/li[3]";
    private static final String GAS_PRICES_TABLE_XPATH = "//*[@id=\"fuel\"]/div[3]";
    private static final String UKR_NET_MAIN_PAGE_URL = "https://www.ukr.net/";
    private static final String SINOPTIC_DOMAIN_NAME = "https://ua.sinoptik.ua/";
    private static final String CHROME_DRIVER_PATH =
"/Users/anastasiiabronitska/IdeaProjects/Master_Diplom/src/main/resources/chromedriver-93";
    private static final String FIREFOX_DRIVER_PATH =
"/Users/anastasiiabronitska/IdeaProjects/Master_Diplom/src/main/resources/geckodriver";
    private static final String CHROME_DRIVER_PATH_PROP = "webdriver.chrome.driver";
    private static final String FIREFOX_DRIVER_PATH_PROP = "webdriver.gecko.driver";
    private static final DriverType CURRENT_DRIVER_TYPE = DriverType.CHROME;
//TODO: change this prop for logic turn

    public static void main(String[] args) throws MalformedURLException {
        WebDriver driver = null;

        if (DriverType.CHROME == CURRENT_DRIVER_TYPE) {
            System.setProperty(CHROME_DRIVER_PATH_PROP, CHROME_DRIVER_PATH);
            ChromeOptions immutableCapabilities = new ChromeOptions();
            immutableCapabilities.setHeadless(false);
            driver = new ChromeDriver(immutableCapabilities);
        } else if (DriverType.FIREFOX == CURRENT_DRIVER_TYPE) {
            System.setProperty(FIREFOX_DRIVER_PATH_PROP, FIREFOX_DRIVER_PATH);
            driver = new FirefoxDriver();
        } else {
            FirefoxOptions mutableCapabilities = new FirefoxOptions();
            mutableCapabilities.addArguments("headless", "no-sandbox", "disable-gpu", "remote-debugin-
port=9222", "screen-size=1200x800");
            mutableCapabilities.setCapability("browserName", "firefox");
            mutableCapabilities.setCapability("platformName", "LINUX");
//            mutableCapabilities.setCapability("");
//            mutableCapabilities.setHeadless(true);
//            mutableCapabilities.addArguments("--no-sandbox");

//            immutableCapabilities.setCapability("browserVersion", "88");

            driver = new RemoteWebDriver(mutableCapabilities);
        }
        driver.get(UKR_NET_MAIN_PAGE_URL);

        try {
            checkThatWeOnUKRNETMainPage(driver);
            checkWeatherTab(driver);
            checkCurrencyTable(driver);
            checkGasPricesTable(driver);
            System.out.println("Success");
        } catch (Throwable throwable) {
            System.out.println("Exception: " + throwable.getLocalizedMessage());
        } finally {
```

```
            System.out.println("Exit");
            driver.quit();
        }
    }

    public static void checkThatWeOnUKRNETMainPage(WebDriver driver) {
        WebElement mainImage = driver.findElement(By.xpath(MAIN_PAGE_IMAGE_XPATH));
        mainImage.click();
        String currentUrl = driver.getCurrentUrl();
        if (!UKR_NET_MAIN_PAGE_URL.equals(currentUrl)) {
            throw new RuntimeException("UKR.NET's Main page isn't working!");
        }
    }

    public static void checkWeatherTab(WebDriver driver) {
        driver.findElement(By.xpath(WEATHER_PAGE_LINK_XPATH)).click();

driver.findElement(By.xpath(WEATHER_PAGE_IMAGE_AND_LINK_TO_SINOPTIC_XPATH)).click
();
        Object[] pages = driver.getWindowHandles().toArray();
        driver.switchTo().window((String) pages[1]);
        String currentUrl = driver.getCurrentUrl();
        if (!currentUrl.contains(SINOPTIC_DOMAIN_NAME)) {
            throw new RuntimeException("SINOPTIC site forwarding for weather isn't working");
        }
        driver.switchTo().window((String) pages[0]);
    }

    public static void checkCurrencyTable(WebDriver driver) {
        driver.findElement(By.xpath(CURRENCY_PAGE_LINK_XPATH)).click();
        boolean currencyTableDisplayed =
driver.findElement(By.xpath(CURRENCY_TABLE_XPATH)).isDisplayed();
        if (!currencyTableDisplayed) {
            throw new RuntimeException("Currency table isn't visible");
        }
    }

    public static void checkGasPricesTable(WebDriver driver) {
        driver.findElement(By.xpath(GAS_PAGE_LINK_XPATH)).click();
        boolean gasPricesTableDisplayed =
driver.findElement(By.xpath(GAS_PRICES_TABLE_XPATH)).isDisplayed();
        if (!gasPricesTableDisplayed) {
            throw new RuntimeException("GAS table isn't visible");
        }
    }
}
```