НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Система генерації автоматичних тестів для вебзастосунків»

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-94
Гереншстейн Петро Анатолійович _____

Керівник:
Асистент кафедри ІСТ
Шинкевич Микола Костянтинович _____

Рецензент:
Доцент кафедри ОТ, кандидат технічних наук
Верба Олександр Андрійович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інтегровані інформаційні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Гереншейну Петру Анатолійовичу

1. Тема проєкту «Система генерації автоматичних тестів для вебзастосунків», керівник проєкту Шинкевич Микола Констянтинович, затверджені наказом по університету від «31» травня 2022 р. № 2101-с

2. Термін подання студентом проєкту: 12 червня 2023 року

3. Вихідні дані до проєкту: веб-застосунок - можливість створення акаунту, можливість входу в акаунт, можливість роботи в різних браузерах, на різних системах. Можливість створення автоматизованих тестів. Можливість редагування та збереження автоматичних тестів. Можливість запуску тестів. Можливість зберігати результати тестів. Можливість переглядати помилки

4. Зміст пояснювальної записки: огляд предметної області та аналіз існуючих рішень для реалізації, аналіз існуючих інструментів розробки веб-застосунків, розробка та декомпозиція інформаційної моделі системи, розробка серверної та клієнтських частин веб-застосунку.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): діаграма класів серверної та клієнтскої частин веб-застосунку, діаграма прецедентів веб-застосунку, діаграма послідовності веб-застосунку

6. Дата видачі завдання 1 березня 2022 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Огляд існуючих рішень	21.04.2023	
2	Аналіз засобів реалізації	28.04.2023	
3	Бізнес-аналіз системи. Моделювання системи.	05.05.2023	
4	Розробка алгоритмів, розробка серверної частини застосунку	12.05.2023	
5	Розробка бази даних та клієнтської частини застосунку	19.05.2023	
6	Оформлення документації дослідницького проєкту	27.05.2023	

Студент Петро ГЕРЕНШТЕЙН

Керівник Микола ШИНКЕВИЧ

АНОТАЦІЯ

Геренштейн П.А. Система генерації автоматичних тестів для веб-застосунків. КПІ ім. Ігоря Сікорського. Київ, 2023. Проект включає 70 сторінок тексту, 32 ілюстрації, 3 таблиці, 23 посилання на літературні джерела, та 4 конструкторські документи.

Ключові слова: веб-застосунок, система генерації автоматичних тестів, автоматизоване тестування, цикл розробки веб-застосунків, тестові запуски, тестові фреймворки

Об’єктом дослідження є створення та використання автоматизованих тестів для веб-застосунків. Предметом дослідження є система для генерації автоматичних тестів веб-застосунків.

Метою роботи є розробка системи, здатної автоматично генерувати тести для веб-застосунків, з метою запобігання виникненню дефектів на різних етапах розробки. В ході розробки індивідуального дослідницького проєкту було створено інформаційну модель системи, тестову документацію, діаграму класів клієнтської та серверної частин веб-застосунку, алгоритм роботи програми, досліджено існуючі рішення, що спрямовані на розв'язання схожих задач, та спроектовано та розроблено веб-застосунок для генерації автоматичних тестів для веб-застосунків

Результати цієї роботи можуть бути використані для підвищення якості кінцевого продукту, шляхом оптимізації та прискорення процесів тестування.

ANNOTATION

Herenstein P.A. System for Generating Automatic Tests for Web Applications. Igor Sikorsky KPI. Kyiv, 2023. The project includes 70 pages of text, 32 illustrations, 3 tables, 23 references to literary sources, and 4 engineering documents. Keywords: web application, automatic test generation system, automated testing, web application development cycle, test runs, testing frameworks.

The object of research is the creation and use of automated tests for web applications. The subject of research is a system for generating automatic tests for web applications.

The aim of the work is to develop a system capable of automatically generating tests for web applications, with the aim of preventing the occurrence of defects at various stages of development. During the development of an individual research project, an informational model of the system was created, test documentation was developed, a class diagram for the client and server parts of the web application was designed, a program operation algorithm was developed, existing solutions aimed at solving similar tasks were researched, and a web application for generating automatic tests for web applications was designed and developed.

The results of this work can be used to improve the quality of the final product by optimizing and accelerating testing processes.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА94.050БАК.004 ПЗ	Пояснювальна записка	70		
6	А3	ІА94.050БАК.004 Д1	Діаграма класів серверної	1		
7			частини веб-застосунку			
8	А3	ІА94.050БАК.004 Д2	Діаграма класів клієнтської	1		
9			частини веб-застосунку			
10	А3	ІА94.050БАК.004 Д3	Діаграма прецедентів	1		
11						
12	А3	ІА94.050БАК.004 Д3	Діаграма послідовності	1		
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІА94.050БАК.004 ТП		
Розроб.		Гереншейн П.А.			Система генерації автоматичних тестів для вебзастосунків Відомість проєкту	Літ.	Аркуш 1 / Аркушів 1
Керівн.		Шинкевич М.К.				Т	
						КПІ ім. Ігоря Сікорського Група ІА-94	
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «Система генерації автоматичних тестів для вебзастосунків»

Київ – 2023 року

ЗМІСТ

ВСТУП

В цифрову епоху, роль інформаційних систем надзвичайно важлива, при цьому їх важливість регулярно збільшується. Вони сприяють ефективному виконанню багатьох завдань, що охоплюють різні сфери діяльності людини. При цьому розробка інформаційних систем стають більш доступними не лише великим компаніям, а і стартапам та маленьким командам. Кожна така компанія може випускати власний продукт, але так як на ринку лишається все менше вільних ніш, продукт повинен не лише задовольняти потреби користувача, а і дбати про UX – досвід юзера під час користування додатком.

Враховуючи необхідність існування та коректного функціонування інформаційних систем, проведення глибокого, якісного тестування стає важливою проблемою. Передусім це стосується тестування інтерфейсу користувача, від якого залежить зручність використання та простота розуміння системи користувачем, що може безпосередньо вплинути на retention - час утримання юзера в додатку, що в свою чергу може збільшити прибуток від користувача.

Ще одним фактором, що впливає на тестування є складність і масштаб сучасних інформаційних систем. Постійне ускладнення інформаційних систем ускладнює тестування, та створює потребу в різноманітних високорівневих інструментах, що дозволять запровадити та спростити процес автоматизації тестування, що включатиме UI тестування, end-to-end тестування, API-тестування, т.д. Автоматизація процесу тестування може значно знизити час та ресурси, потрібні для тестування, імплементувати автоматичне тестування в CI/CD процес, а також підвищити його якість та надійність.

Процес автоматизації являє собою розробку скрипта, що виконуватиме певні запрограмовані дії, ці дії є статичними, що підвищує потребу в розробці якісної тестової документації, зокрема тест-кейсів та тест-планів що

вимагатимуть роботи QA-інженера. Найчастіше розробкою тестової документації займається Manual QA інженер.

Цей дипломний проєкт, що носить назву "Система автоматичної генерації UI тестів", присвячена розробці системи, яка зможе автоматично генерувати тести для перевірки інтерфейсу користувача. Ця система може спростити процес побудови автоматичних тестів, та дозволити Manual Quality Assurance інженерам будувати тести без допомоги Automation Quality Assurance інженерів, використовуючи зручний користувацький інтерфейс системи генерації автоматичних тестів.

Автоматизація проектів дозволить суттєво зменшити кількість рутинних тестів, які потрібно виконувати Manual QA інженерам, зменшити потребу в Automation QA інженерам, та може допомогти зменшити витрати на утримання команди тестувальників.

Система для генерації автоматичних тестів може допомогти покращити процес тестування застосунків, спростити інтеграцію Manual Quality Assurance інженерів в процеси створення автоматичних тестових скриптів. Така система може бути легко запроваджена в життєві цикли розробки та тестування системи.

1 АСПЕКТИ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ ТА ОСНОВНІ ПОНЯТТЯ ТЕСТУВАННЯ

1.1 Огляд основних понять

1.1.1. Веб-застосунок

Веб-застосунок - це комп'ютерна програма, яка використовує веб-браузери як клієнтське обладнання для взаємодії з користувачами. Він створюється за допомогою веб-технологій, таких як HTML, CSS та JavaScript, і може виконуватися на будь-якому пристрої, який має веб-браузер.

Веб-застосунки можуть включати широкий спектр функцій, включаючи, але не обмежуючись, обмін електронною поштою, онлайн-торгівлю, управління базами даних, корпоративні внутрішні мережі, колаборативне робоче місце, редактори документів в режимі реального часу та багато іншого.

Веб-застосунки відрізняються від традиційних настільних програм, оскільки вони не потребують встановлення на пристрій користувача. Вони дозволяють користувачам отримувати доступ до своїх даних та використовувати програму з будь-якого місця, де є доступ до Інтернету.

Архітектура веб-застосунків може бути складною, залежно від їхньої складності та функціональності. Вона зазвичай включає серверну частину, яка обробляє бізнес-логіку, веб-сервер, що слугує за посередника між користувачем і сервером, та клієнтську частину, яка відображає інтерфейс користувача і забезпечує взаємодію користувача з застосунком.

Сучасні веб-застосунки також можуть включати інтеграцію з зовнішніми сервісами, такими як соціальні мережі, системи оплати, API третіх сторін тощо. Це розширює можливості веб-застосунку і забезпечує додаткову цінність для користувачів.

1.1.2. Система тестування

Система автоматизованого тестування (Test Framework) - це структурований набір автоматизованих тестів, які використовуються для функціонального тестування програмного забезпечення. Такі системи зазвичай організовані за допомогою встановлених правил та настанов для створення та проектування тестових випадків, що підлягають автоматизації, враховуючи вимоги до програмного забезпечення.

Система автоматизованого тестування може включати різноманітні техніки та методики тестування, такі як модульне тестування, інтеграційне тестування, функціональне тестування, тестування процесів користувача (end-to-end), регресійне тестування. Методики можуть включати в себе стандарти кодування, обробку тестових даних, репозиторії об'єктів, процеси зберігання результатів тестування та методи отримання доступу до зовнішніх ресурсів.

Мета автоматизованої системи тестування - забезпечити надійне та ефективне тестування, зменшити час та зусилля, потрібні для виконання тестів, і підвищити загальну продуктивність тестування. Незважаючи на те, що використання такої системи не є обов'язковим, її впровадження може забезпечити значні переваги, такі як покращення якості програмного забезпечення, зменшення помилок, покращення продуктивності тестування та забезпечення вищого рівня впевненості в якості програмного забезпечення [1].

1.1.3. Життєвий цикл розробки програмного забезпечення

Життєвий цикл програмного забезпечення (англ. – Software Development LifeCycle) – це сукупність процесів, що використовуються в галузі програмного забезпечення для опису циклу проектування, розробки, тестування та підтримки продукту (програмного забезпечення). Основним призначенням SDLC є виорбництво ПЗ, що відповідає чи перевершує очікування замовника чи клієнта,

в найкоротші терміни. Узагальнене представлення циклу зображено на рисунку 1.1

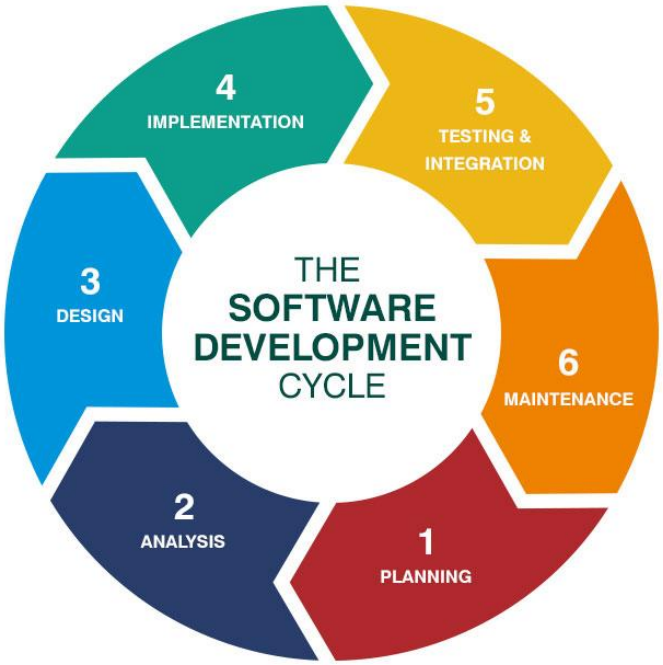

Рисунок 1.1 — зображення циклу програмного забезпечення

Загалом, можна виділити 6 основних етапів циклу:

- Планування;
- Аналіз;
- Дизайн;
- Розробка;
- Тестування;
- Підтримка.

В залежності від проекту, методології або типу компанії певні компоненти можуть відрізнятись, зокрема часто додається фаза Deployment.

Розглянемо цикл розробки програмного забезпечення

На етапі планування та аналізу буде розроблено модель системи, визначено та проаналізовано функціональні та нефункціональні вимоги, та

критерії прийняття продукту. Після завершення етапу планування буде проаналізовано існуючі рішення, розглянуто можливості їх покращення

На етапі дизайну буде виконано моделювання системи, декомпозиція системи, діаграма діяльності та прецедентів.

Найбільш об’ємним етапом буде розробка. Вхідними критеріями цього проекту буде інформаційна модель системи, декомпозиція її компонентів, дизайн зовнішнього вигляду клієєнтської частини застосунку. Безпосередньо на етапі розробки буде обрано стек технологій, за допомогою яких буде виконуватись розробка. Вихідним критерієм буде програмний застосунок, готовий до тестування.

Цикл розробки включатиме в себе наступні розділи:

- Розробка серверної частини (Backend);
- Розробка та інтеграція бази даних;
- Розробка клієнтської частини застосунку (Frontend).

1.1.4. Життєвий цикл тестування програмного забезпечення

Після етапу розробки іде ще один важливий етап – тестування. Тестування в свою чергу має власний життєвий цикл: STLC, або Software Development LifeCycle. Життєвий цикл тестування зображено на рисунку 1.2

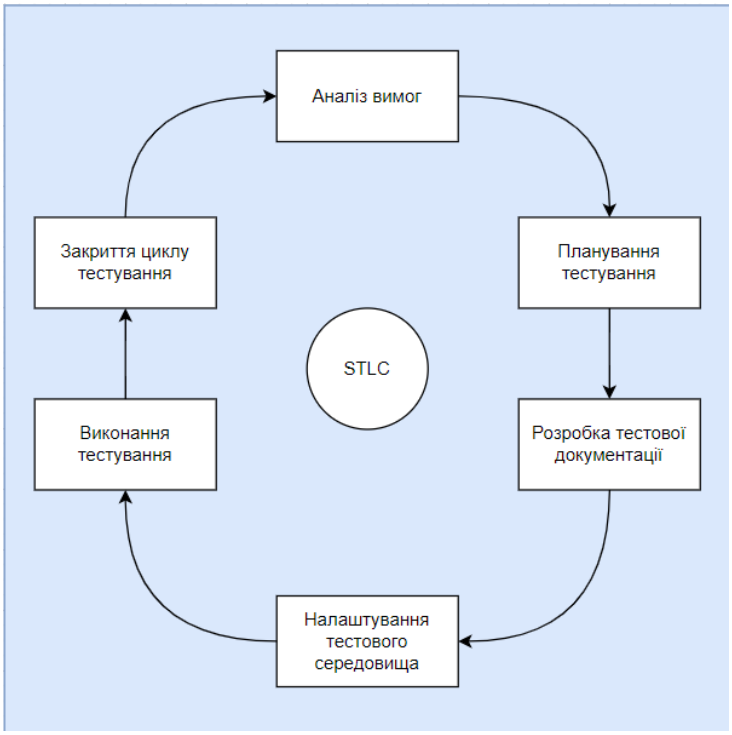

Рисунок.1.2 – Життєвий цикл тестування

Він включає в себе 6 етапів, перший з яких починається одночасно з генерацією фічі. Повний список етапів тестування:

1) Аналіз вимог. На цьому етапі команда тестувальників вивчає та аналізує бізнес вимоги та технічні специфікації продукту. Мета - зрозуміти, що система повинна робити, щоб визначити, що буде перевірено під час тестування. Цей етап може включати взаємодію з розробниками, бізнес-аналітиками і кінцевими користувачами;

2) Планування Тестування (Test Planning). Тут визначаються стратегія тестування, ресурси, графік і критерії успіху. Команда тестування розробляє документ, відомий як план тестування, який описує, як вони планують перевірити систему;

3) Розробка тестової документації. На цьому етапі команда тестування створює тестові сценарії, тест-кейси, чеклисти для тестування та тестові скрипти на основі аналізу вимог. Ці документи детально описують алгоритми, види, типи тестування та послідовності кроків, що QA-команда буде виконувати

безпосередньо під час тестування, а також очікувані результати, та критерії за якими буде визначатись пройшов той чи інший тест чи ні;

4) Налаштування тестового середовища. Тестове середовище - це сукупність апаратного та програмного забезпечення, яка використовується для проведення тестів. Воно повинно бути якомога ближче до виробничого (Production) середовища. Цей крок може включати в себе встановлення та налаштування бази даних, браузерів і будь-якого іншого необхідного програмного забезпечення;

5) Виконання тестів. Після налаштування тестового середовища та розробки необхідної документації, команда тестування може починати виконувати тести. Крім безпосереднього виконання тестів, цей крок включає в себе відстеження всіх дефектів, або багів, оформлення результатів тестування, відповідно до критеріїв, визначених на етапі розробки тестової документації;

6) Закриття результатів тестування. Останнім кроком є аналіз результатів тестування, вирішення відкритих питань та підготовка звітів про тестування для зацікавлених сторін. Цей процес включає перегляд результатів, порівняння цих результатів з визначеними критеріями успіху, та визначення, чи можна продукт випустити.

1.2 Тестування

Процес тестування являє собою сукупність функціональних та нефункціональних перевірок, що має за мету забезпечення відповідності програмного забезпечення очікуваній поведінці, функціональним та нефункціональним вимогам до продукту. Крім виконання тестів, тестування включає в себе їх планування, документування тестів, оформлення результатів тестування, т.д. За видами тестування поділяється на мануальне (ручне) та автоматичне.

1.2.1 Мануальне тестування

Мануальне тестування - це процедура чи набір перевірок програмного забезпечення без використання автоматизованих інструментів або скриптів. В цьому виді тестування, інженер із забезпечення якості самостійно виконує різні набори кроків, описані в тестовій документації, щоб перевірити якість та стабільність роботи програмного забезпечення, а також виявити і задокументувати помилки.

На першому етапі мануального тестування, інженер із забезпечення якості розробляє стратегію тестування, де визначається які частини продукту будуть тестуватися, та визначає критерії, за якими буде визначатись результат проходження тестування.

Після цього, інженер з забезпечення якості декомпозує задачу тестування, розробляє тестові сценарії, тестові випадки (тесткейси) та чеклисти, які конкретизують дії, які тестувальник виконуватиме під час тестування, а також очікувані результати цих кроків. Тест кейси можуть бути простими, наприклад перевірка коректної валідації вводу даних, або більш складними, наприклад перевірка взаємодії між різними компонентами, або мікросервісами системи, або інтеграцію зі сторонніми сервісами, чи API.

Мануальне тестування важливе, оскільки воно дозволяє тестерам виявити помилки, які можуть бути пропущені автоматизованими системами. При активному використанні мануального тестування, члени QA-команди глибше ознайомлюються з програмним продуком, що в свою чергу дозволяє тестувальникам краще зрозуміти поведінку системи. Такі навички зокрема допомагають досвідченим та добре інтегрованим інженерам із забезпечення якості передбачати ймовірні проблемні місця під час тестування нового функціоналу

1.2.2 Автоматичне тестування

Автоматичне тестування - це процес, в якому автоматизовані скрипти та програмні інструменти використовуються для виконання тестів, отримання та перевірки результатів. Подібно до мануального тестування, автоматичне займається перевіркою роботи програми, але без прямого втручання людини в безпосередній процес тестування.

Основний принцип автоматичного тестування полягає в створенні тестових сценаріїв, які можуть бути повторно використані. Ці сценарії описують набір конкретних кроків, які має виконати тестовий скрипт, та очікувані результати для кожного, або деяких кроків. Ці сценарії виконуються автоматично, та можуть бути збережені та перезапущені на різних етапах тестування.

Перевагою автоматичного тестування є його швидкість, ефективність і передбачуваність. Автоматичне тестування може бути виконане значно швидше, ніж мануальне тестування, може виконуватись паралельно для різних компонентів системи. Також воно дозволяє швидко виявити та локалізувати помилки, що з'явилися під час розробки або модифікації програмного забезпечення, або модулі в яких виникають помилки, з метою подальшого дослідження мануальними тестувальниками.

Однак, необхідно зазначити недоліки автоматичного тестування, зокрема те, що таке тестування вимагає від тестувальника відповідних навичок, а також апаратних та програмних ресурсів. Для розробки ефективних тестових сценаріїв та їх оптимізації тестувальнику потрібно мати досвід роботи з мовами програмування, а також інструментів та фреймворками, призначеними для автоматичного тестування.

Автоматичне тестування є ключовим інструментом в сучасних методологіях розробки програмного забезпечення, таких як Agile, де швидкість та ефективність є важливими. Воно дозволяє командам швидко виявляти та

виправляти помилки, що підвищує якість продукту та скорочує час виходу нового функціоналу для широкого кола користувачів.

1.2.3 Типи тестування

Попри те, що процес тестування динамічний та може відрізнятись для різних команд та проектів, можна виділити узагальнену послідовність різних етапів тестування

1) Модульне тестування, іноді називається юніт-тестуванням, спрямоване на перевірку окремих компонентів програми – модулів, або компонентів. Цей етап відбувається на ранніх стадіях розробки та може здійснюватись розробниками. Головною метою є виявлення помилок у функціонуванні окремих модулів перед інтеграцією цих модулів до системи;

2) Інтеграційне тестування. Після того, як модулі пройшли етап модульного тестування, вони інтегруються до системи. На цьому етапі здійснюється інтеграційне тестування, що має за мету перевірку взаємодії між модулями та виявлення проблем, які можуть виникнути при їх взаємодії;

3) Системне тестування. На цьому етапі відбувається тестування всієї системи. Мета системного тестування – верифікація відповідності системою заданим вимогам, та її поведінка в реальному середовищі співпадає з очікуваною. Системне тестування може включати різні типи тестування, такі як функціональне та нефункціональне тестування, тестування продуктивності, безпеки, юзабіліті та інші види тестування;

4) Регресійне тестування. Цей етап тестування відбувається після внесення змін до програмного забезпечення, наприклад виправлення помилок, додавання нового функціоналу. Мета регресійного тестування - переконатися, що внесені зміни не порушили роботу вже існуючого програмного забезпечення. Регресійне тестування часто включає в себе автоматичне тестування. Це

дозволяє швидко і ефективно перевіряти роботу різних компонентів складних систем; [2]

5) Димчате тестування (англ. Smoke testing)

Димчасте тестування (smoke testing) - це процес випробування основних функцій програмного забезпечення для переконання, що вони працюють належним чином перед проведенням більш глибокого тестування.

Назва "димчасте тестування" походить від аналогії з електронікою: коли новий пристрій вперше підключається до джерела живлення, інженери стежать, чи не іде дим з пристрою, щоб переконатися, що він відповідає основним вимогам безпеки.

У контексті програмного забезпечення, димчасте тестування зосереджується на основних компонентах програми. Це може включати в себе такі аспекти, як вхід в систему, навігація по основних сторінках, т.д. Головною метою є визначення, чи готове програмне забезпечення до подальшого більш детального тестування.

Хоча димчасте тестування не гарантує, що програмне забезпечення не має дрібних помилок, воно допомагає забезпечити коректну поведінку основної частини застосунку; [3-4]

6) Наскрізне тестування Наскрізне тестування, також відоме як End-to

End (E2E) тестування, - це підхід, який перевіряє, як система в цілому взаємодіє з декількома компонентами та службами, включаючи сторонні системи та інтерфейси, з метою симулювання реального сценарію користувача.

Цей підхід до тестування передбачає відтворення поведінки користувача у системі від початку до кінця. Він допомагає переконатися, що всі взаємозалежні компоненти системи працюють разом і дають очікуваний результат.

Розглянемо приклад E2E тестування для веб-сайта електронної комерції: тест може початися з входу користувача на сайт, додавання товару до кошика, переходу до оформлення замовлення, введення даних платіжної карти,

завершення покупки та отримання електронного підтвердження покупки. Всі ці кроки симулюють реальний сценарій користувача.

Запуск такого тесту вручну може бути часозатратним, але з автоматизацією E2E тестування можна виконувати ефективно і швидко. За допомогою автоматизованих E2E тестів, розробники можуть створювати сценарії, які могуть бути використані в будь-який час, що полегшує пошук помилок та забезпечує стабільність роботи системи [4].

Усе більшу роль у процесах тестування займає автоматизація. Це підтверджується розвитком та широким застосуванням наскрізного тестування, яке здатне ефективно перевіряти роботу системи, симулюючи поведінку користувача. Але хоча автоматизовані тести спроможні ефективно викривати помилки і гарантувати стабільність роботи системи, існує області, де важливу роль відіграє людський чинник.

Мануальне тестування, незважаючи на технологічні прогреси в області автоматизації, не втратило своєї значимості. Компетентні мануальні тестувальники, що мають глибоке розуміння продукту та бізнес-логіки, завжди будуть у великому запиті. Відсутність розуміння автоматизації тестування та відповідних навичок тестування стає великим викликом для мануальних тестувальників, які хочуть перейти до автоматичного тестування.

Тому важливо визнати, що автоматизація та мануальне тестування не є взаємовиключаючими, а доповнюють одне одного. З цього приводу розробка високорівневих систем, що спрощують інтеграцію мануальних тестувальників в автоматизацію, стає актуальною задачею в області тестування програмного забезпечення.

1.3.Особливості розробки веб-застосунків

Веб-застосунки зазвичай складаються з клієнтської частини, серверної частини та бази даних. Клієнтська частина може виконувати певні дії, та існувати

без серверної, наприклад, сайти візитки (Landing page), але в масштабних веб-застосунках більшість логіки виноситься на серверну сторону, в тому числі всі операції не пов'язані з користувацьким інтерфейсом.

Таким чином, для роботи з даними та виконання операцій, не пов'язаних з браузером клієнта будь-якому веб-застосунку потрібна серверна частина. Класична модель веб-застосунку зображена на рисунку 1.3

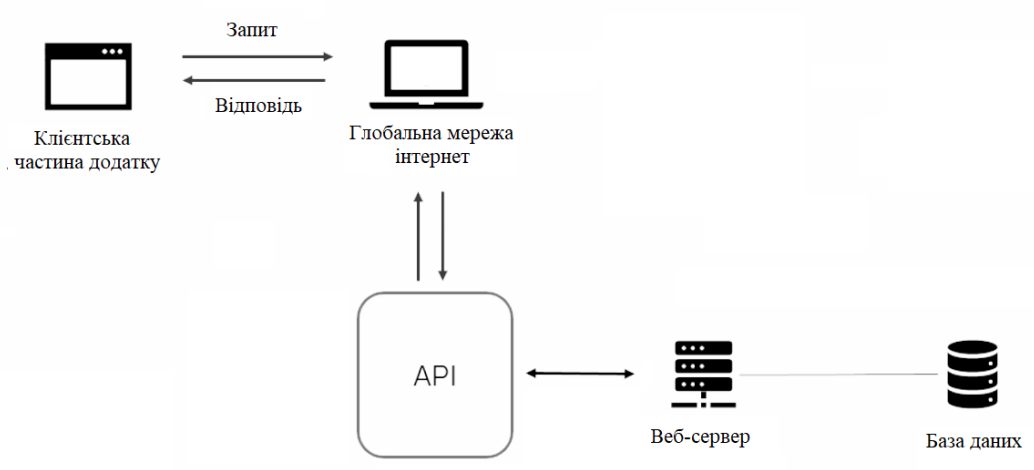

Рисунок 1.3 — Модель веб-застосунку

За визначенням, що надає AWS: «API – це механізм, що дозволяє двом компонентам програмного забезпечення взаємодіяти між собою за допомогою певних визначень та протоколів. Наприклад, система погодного бюро зберігає дані про погоду по днях, програма погоди «говорить» з цією системою за допомогою API і показує вам щоденні оновлення погоди на вашому телефоні»[5].

Перефразовуючи це можна сказати що API обробляє запити з клієнтського боку, якщо запит є дійсним та коректним а користувач має доступ до даних, які запитує, API передає запит на сервер, далі сервер обробляє запит, та надсилає відповідь до API, де відповідь обробляється, отримує статус код та у вигляді відповіді повертається на клієнтську частину з якої було виконано запит

Існує декілька видів API та багато підходів до їх побудови, але задача, яку виконує API незмінна. Кінцеві точки (ендпоінти) API відокремлюють клієнтський додаток від серверу та баз даних, таким чином API залишається проміжним шаром між клієнтом і сервером з головною задачею – обробка даних та взаємодія між клієнтом і сервером

Висновок до першого розділу

У першому розділі було розглянуто основні поняття, що стосуються тестування, веб-застосунків та систем автоматизованого тестування. Було оглянуте поняття життєвого циклу розробки програмного забезпечення, надано огляд кожного етапу даного життєвого циклу, виділено їх параметри, вхідні та вихідні критерії

Обґрунтовано необхідність розробки серверної частини застосунку для генерації автоматичних тестів для веб-застосунків, враховуючи велику кількість операцій з даними, така система не змогла б існувати без серверної частини, що оброблятиме запити до цих даних.

Попри всі переваги автоматичного тестування, потреба в мануальних тестувальниках не зменшується, при цьому збільшується попит на тестувальників, що одночасно вміють здійснювати і мануальне і автоматичне тестування.

Враховуючи зростаючу кількість веб-застосунків та швидкість їх розробки, імплементація автоматизованого тестування стає єдиним варіантом для компаній, що розробляють такі застосунки , що збільшує попит не лише на тестувальників, здатних розробляти системи автоматизованого тестування, а і на інструменти, що можуть допомогти інтегрувати автоматизоване тестування в процеси тестування веб-застосунків в межах проекту, та спростити перехід мануальних тестувальників в автоматизаторів.

2 АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ ТА СТВОРЕННЯ АВТОМАТИЗОВАНИХ ТЕСТІВ

Система генерування автоматичних тестів має включати в себе не лише веб-застосунок. Вихідним критерієм цієї системи має бути створений файл для автоматичного тестування, який користувач зможе виконати чи завантажити для локального виконання. Крім цього, система повинна мати користувацький інтерфейс в браузері, в цій сфері єдиним вибором є JavaScript – єдина мова, яку здатен інтерпретувати кожен браузер.

Відповідно до цього, в даному розділі необхідно розглянути не лише мови програмування, а і фреймворки для створення автоматизованих тестів

2.1. Вибір мови програмування

У світі інформаційних технологій вибір мови програмування відіграє важливу роль у процесі розробки будь-якого продукту, включаючи системи автоматичного тестування. Цей вибір може суттєво вплинути на продуктивність, швидкість розробки, а також на майбутнє обслуговування та розширення системи. У цьому підрозділі ми поглиблено розглянемо критерії вибору мови програмування для автоматичного тестування, а також обговоримо декілька популярних мов, що часто використовуються у цій області.

В 2023 році, стаття присвячена рейтингу мов програмування на веб-сайті DOU вийшла з заголовком що містив наступну тезу: «Javascript/TypeScript завойовують світ, Python увійшов у топ-3» [6].

Потрібно врахувати, що TypeScript це надбудова над JavaScript, що містить у собі логіку пов’язану з типізацією змінних та полів, а також створення власних типів. Хоч ця логіка і є синтаксичним цукром, вона робить TypeScript надзвичайно популярною серед JavaScript розробників, однієї з найширших спільнот розробників

На рисунку 2.1 наведена статистика використання мов програмування в роботі серед розробників з опитування Dou

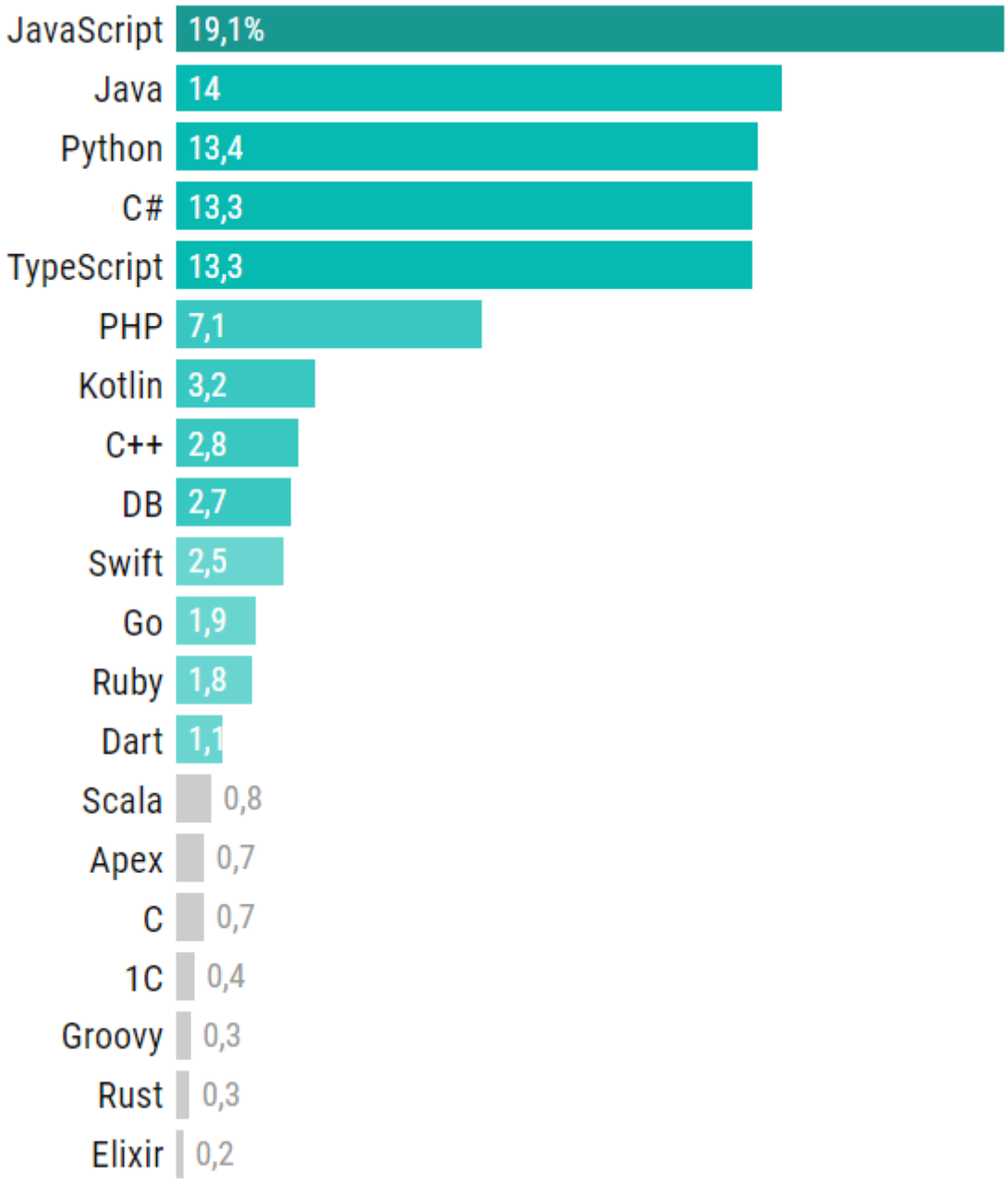

Рисунок 2.2 — Рейтинг мов програмування за версією Dou

Трійка лідерів списку – JavaScript, Java та Python відповідно. TypeScript знаходиться на 5 місці, та разом з JavaScript має 32.4% ринку, що складає майже третину.

При цьому відсоток Java розробників поступово зменшується, натомість відсоток користувачів JavaScript та TypeScript значно збільшився в останні 10 років. Динаміка популярності мов програмування за період з 2013 по 2023 рік за версією Dou зображена на рисунку 2.2

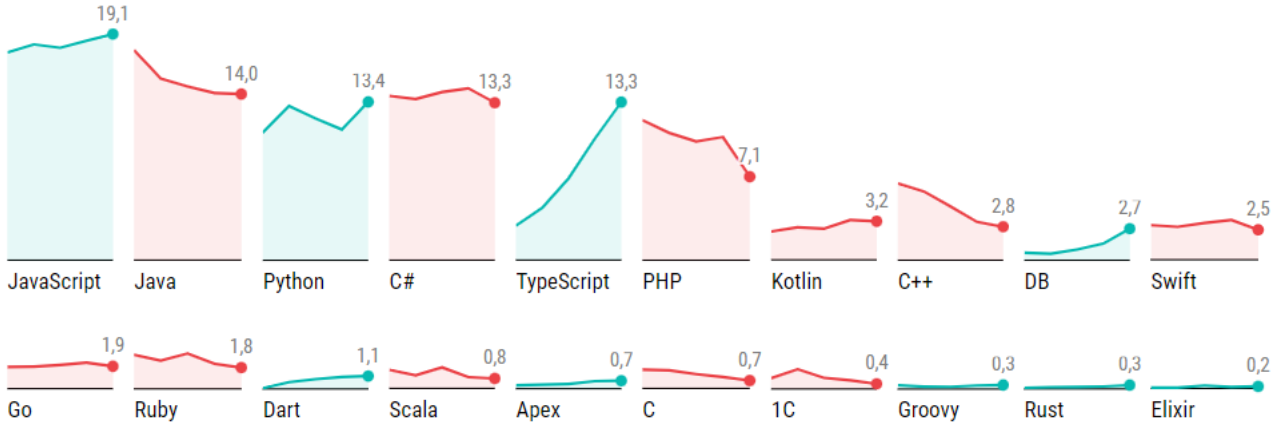

Рисунок 2.3 — Динаміка популярності мов програмування за версією Dou

Але попри позитивну динаміку JavaScript, Java та Python досі лишаються дуже популярними, а кількість проектів написаних на Java настільки висока, що мова ще дуже довго залишатиметься на ринку. Відповідно, потрібно використати їх для порівняння.

JavaScript є кроспарадигменою мовою програмування, яка використовує функції першого класу. Як мова сценаріїв для веб-сторінок, вона займає важливе місце в сучасному програмуванні, але використовується також у небраузерних середовищах, таких як Node.js, Adobe Acrobat та Apache CouchDB.

Одна з основних функцій JavaScript - це забезпечення динамічного контенту на веб-сторінках. Вона надає можливість оновлювати вміст сайту в реальному часі, забезпечуючи анімацію, оновлювані діаграми, прокручування відео в плеєрі і т.д. Все це реалізовано за допомогою JavaScript.

JavaScript також використовується в широкому спектрі застосунків, не обмежуючись тільки клієнтською частиною веб-застосунків. Велика кількість бібліотек та фреймворків доступна для JavaScript, що дозволяє розширити

можливості цієї мови в різних напрямках, включаючи клієнтську частину веб-застосунку та автоматизоване тестування.

JavaScript є універсальною мовою, що дозволяє вивчити одну мову і застосовувати один і той самий синтаксис для написання функцій. Хоча для повноцінної роботи розробникам все одно потрібно досліджувати відповідні бібліотеки та фреймворки, базові знання мови залишаються незмінними.

Крім JavaScript, є важливо також розглянути TypeScript. TypeScript - це типізована надмножина JavaScript, яка додає типізацію змінних до синтаксису JavaScript. Хоча TypeScript перекладає код на JavaScript для виконання в браузері, він містить ряд власних доповнень до синтаксиса JavaScript.

Основна мета TypeScript - надати статичну перевірку типів для JavaScript, щоб зменшити кількість типових помилок, що виникають під час написання коду [7].

Python - це високорівнева, інтерпретована мова програмування, відома своїм чистим синтаксисом і великою кількістю бібліотек. Ця мова призначена для швидкої розробки програм та зручного читання коду, завдяки своєму дизайну, що включає в себе структуру блоків коду, що розділені відступами.

Основна мета Python - максимізувати читабельність та ефективність коду. Це відбувається через впровадження ключових принципів, викладених в "The Zen of Python", документі, що описує філософію цієї мови [8].

Python застосовується в широкому спектрі областей, включаючи веб-розробку, науку про дані, штучний інтелект, машинне навчання та автоматизацію. Завдяки своєму розмаїттю бібліотек і фреймворків, Python відкриває безліч можливостей для розробників у різних сферах.

Python є мовою вищого рівня, що підтримує кілька парадигм програмування, включаючи процедурне, об'єктно-орієнтоване та функціональне програмування. Це робить Python універсальним інструментом, що дозволяє розробникам використовувати різні підходи до розробки програмного забезпечення, в залежності від конкретних потреб і обставин.

Однією з ключових властивостей Python є його гнучкість. Python дозволяє вам писати скрипти для автоматизації рутинних завдань, створювати складні веб-застосунки, а також аналізувати великі обсяги даних, створюючи широкий спектр різноманітних візуалізацій, від ствопчикових графіків до складних гістограм.

В контексті безпосередньо веб-застосунків, Python широко використовується для веб-розробки. Flask і Django - два найпопулярніші веб-фреймворки Python, які дозволяють швидко створювати масштабовані та надійні веб-застосунки.

Java – це високорівнева, об'єктно-орієнтована мова програмування, заснована на принципах "Напиши один раз, запусти будь-де". Завдяки своєму універсальному JVM (Java Virtual Machine) вона дозволяє коду Java працювати на різноманітних платформах.

Java надійна і стабільна мова програмування, зокрема, для великих систем. Вона відома своєю сильною типізацією, автоматичним керуванням пам'яттю, вбудованою підтримкою многопоточності, а також широким спектром стандартних бібліотек [9].

Java використовується в різних областях розробки ПО: від веб-застосунків, корпоративних систем і мобільних додатків до програмування вбудованих систем і "інтернету речей". Засоби мови Java, як-то абстракція даних, інкапсуляція, наслідування і поліморфізм, дозволяють створювати модульні програми з чіткою структурою і високою масштабованістю.

Java постійно оновлюється та вдосконалюється, щоб відповідати сучасним вимогам до програмування, при цьому ретельно дбаючи про зворотню сумісність. Це забезпечує Java дуже тривале утримання в рейтингу найкращих мов програмування і гарантує її актуальність в майбутньому.

Ця мова має значне застосування в корпоративній (Enterprise) сфері та веб-розробці. Spring Framework – це найпопулярніший фреймворк Java, який забезпечує розширену підтримку для створення веб-застосунків на Java. Spring

забезпечує широкий спектр функцій, що допомагають керувати ускладненими процесами розробки корпоративних застосунків [10].

У таблиці 2.1. наведена інформація стосовно розглянутих вище мов програмування в контексті універсальності їх використання в контексті системи для генерації автоматичних тестів для веб-застосунків.

Таблиця 2.1 — Порівняння універсальності мов використання

Спектр використання:	JavaScript/ TypeScript	Python	Java
Клієнтська частина веб-застосунку	+	-	-
Серверна частина веб-застосунку	+	+	+
Написання автоматизованих тестів	+	+	+

2.2. Технології для розробки клієнтської частини веб-застосунку

2.2.1. React

React, відомий також як React.js або ReactJS, є відкритим JavaScript-фреймворком для побудови інтерфейсів користувача, який був розроблений і підтримується Facebook. Основою React є ідея "компонентів" - ізольованих шматків коду, які можна легко перевикористовувати в межах проекту, або навіть між проектами. Всі компоненти незалежні, і їх можна використовувати для побудови більш складних UI.

React зосереджується на створенні SPA (односторінкових додатків) і використовує віртуальний DOM для ефективного оновлення і рендерингу компонентів. Це означає, що React створює копію DOM в пам'яті, вносить зміни

до цієї копії, а потім оновлює реальний DOM, мінімізуючи кількість потрібних оновлень, що дозволяє досягти високої продуктивності [11].

React не є повноцінним фреймворком, а скоріше бібліотекою для побудови інтерфейсів користувача. Він не включає в себе ряд інших важливих аспектів, таких як управління станом або маршрутизація, але для цього можна використовувати додаткові бібліотеки, наприклад Redux для управління станом та React Router для маршрутизації.

React популярний серед розробників, багато великих компаній використовують його у своїх проектах, зокрема Airbnb, Netflix та Instagram. Він має велику та активну спільноту, що означає, що знайти рішення для типових проблем або отримати допомогу в разі потреби зазвичай не складно.

Однак, хоча React пропонує велику гнучкість, це може зробити процес вивчення та використання його трохи більш складним, оскільки розробники повинні самі вирішувати, які додаткові бібліотеки використовувати і як їх використовувати.

Перевагами React JS є:

- Швидкість та ефективність;
- Велика спільнота розробників;
- Гнучкість у виборі бібліотек та архітектури;
- Підтримка з боку Facebook.

При цьому недоліками можна вважати:

- Висока крива навчання для більш складних концепцій;
- Недостатність офіційних настанов;
- Не повністю комплексне рішення, вимагає підключення сторонніх бібліотек.

2.2.2. Angular

Angular - це відкритий фреймворк, розроблений і підтримуваний Google, який слугує для створення ефективних і масштабованих веб-додатків. Первісно представлений у 2010 році як AngularJS, цей фреймворк впослідстві був повністю переписаний і релізований як Angular у 2016 році.

Angular використовує компонентну архітектуру для структурування коду, подібно до React. Він має сильну зв'язність між компонентами і включає в себе ряд інтегрованих рішень для різних завдань, як-то форми, HTTP-запити, маршрутизація і інше. Завдяки цьому, Angular називають "опініонованим" фреймворком, оскільки він має чітке бачення того, як має бути структурований код.

Angular написаний на TypeScript, який пропонує статичну типізацію і покращені засоби для рефакторингу коду. Це може сприяти підвищенню продуктивності розробників та якості коду, хоча вивчення TypeScript може зайняти додатковий час для тих, хто звик до JavaScript.

Angular пропонує високу продуктивність завдяки таким механізмам, як відстеження змін через зони та асинхронне рендеринг через Angular Universal.

Однак, Angular може виявитися складним для вивчення, особливо для новачків. Його архітектура та API можуть бути складними, а навчальна крива вважається досить крутою. Також, враховуючи те, що Angular - це великий фреймворк, його розмір може мати негативний вплив на продуктивність веб-додатків, особливо на мобільних пристроях з обмеженими обчислювальними ресурсами [12].

Перевагами Angular є:

- Повністю комплексне рішення (не вимагає сторонніх бібліотек);
- Підтримка TypeScript з коробки;
- Мощна система декларативних шаблонів;
- Велика спільнота розробників;
- Підтримка з боку Google.

Але недоліками залишаються:

– Висока складність навчання;
– Велика кількість необов'язкових концепцій та складностей;
– Більший розмір фреймворку порівняно з React та Vue.js.

2.2.3. Vue.js

Vue.js є прогресивною JavaScript-технологією, призначеною для створення інтерактивних веб-інтерфейсів. Цей фреймворк зосереджений на зручності розробки, орієнтуючись на модель компонентів, що дозволяє забезпечити чітку структуру коду і легкість його підтримки.

Головна особливість Vue.js полягає в його гнучкості і доступності для новачків. Розробники мають можливість створювати прості веб-додатки, не вдаючись до складних концепцій, але при цьому можуть послідовно вивчати продвинуті можливості фреймворка для розширення функціоналу своїх проектів.

Vue.js використовує систему одностороннього потоку даних у компонентах для спрощення слідкування за змінами стану програми, але також підтримує двостороннє зв'язування даних для елементів форм та інших елементів користувацького інтерфейсу.

Vue.js характеризується маленьким розміром і високою продуктивністю, що робить його особливо корисним для веб-додатків, що вимагають швидкої реакції та загрузки. Цей фреймворк може використовуватись як бібліотека для додавання інтерактивності до існуючих сторінок, так і як повноцінний фреймворк для розробки великих проектів [13].

Хоча Vue.js пропонує широкий спектр можливостей, він може мати деякі обмеження, особливо для масштабних комерційних проектів. Це може включати менший набір вбудованих інструментів порівняно з іншими більш розробленими фреймворками, такими як Angular.

Крім низького розміру, Vue.js має ширший набір переваг, зокрема:

– Простота і гнучкість;
– Низька складність навчання;
– Добре документовано;

Але так як це продукт, що підтримується спільнотою, Vue має відповідний список недоліків, зокрема:

– Менша спільнота розробників
– Обмежені можливості для дуже великих та складних проектів
– Менший набір вбудованих інструментів порівняно з Angular

2.2.4. Порівняння інструментів для розробки клієнтської частини застосунку

Очевидно що кожен інструмент має великий список переваг, що завоював для них високу популярність. Згідно з NPM, найбільш популярним залишається React JS, на 2 місці Vue.js [14]. Графік з порівнянням кількості встановлень цих фреймворків наведено на рисунку 2.3, порівняння у наведено у таблиці 2.2.

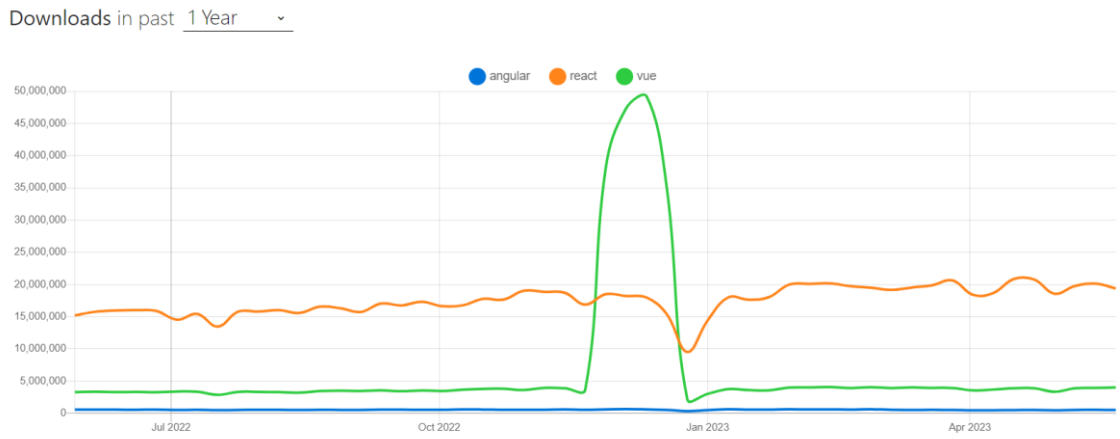

Рисунок 2.4 — Порівняння кількості встановлень за фреймворком

Таблиця 2.2 — Порівняння фреймворків для клієнтької частини застосунку

Критерій	React	Angular	Vue
Випущено	2013	2010, 2016 (Angular 2+)	2014
Розробник	Facebook	Google	Evan You
Мова програмування	JavaScript, JSX	TypeScript	JavaScript, TypeScript
Підхід	Бібліотека, що має фокус на віртуальному DOM і компонентах	Повноцінний фреймворк з набором інструментів	Прогресивний фреймворк, що має компонентний підхід
Модель стану	Додаткові бібліотеки (Redux, MobX)	RxJS	Vuex
Маршрутизація	Додаткова бібліотека (React Router)	Вбудована	Вбудована, Vue Router
Спільнота та підтримка	Велика	Велика	Менша за розміром, активно розширюється
Навчальні матеріали	Велика кількість	Велика кількість	Кількість менша, але якість висока
Використання в промисловості	Широке використання	Широке використання, особливо в корпоративних проектах	Швидко набирає популярність, використовується в проектах різного рівня

Після глибокого аналізу та порівняння React, Angular і Vue.js, Vue.js було обрано як оптимальний варіант. Він поєднує простоту і гнучкість React з декларативним стилем Angular, при цьому маючи низьку важкість навчання. Vue.js також дуже швидкий і ефективний, він добре документований і має активну спільноту і широко використовується в розробці веб-застосунків.

2.3 Технології для розробки серверної частини веб-застосунку

Якщо для клієнтської частини вибір мови програмування був відсутній, для серверної частини він наявний. Обирати потрібно буде з типових фреймворків: для Java – Spring framework, для Python – Django, для JavaScript – Express.

2.3.1 Spring Framework.

Java є однією з найбільш широко використовуваних мов програмування в світі і знаходить своє застосування в розробці великої кількості різних типів застосунків, включаючи веб-застосунки. Основна перевага Java полягає в її універсальності, надійності та масштабованості, що робить її ідеальним вибором для великих та складних проектів.

Spring Framework є одним з найпопулярніших фреймворків для розробки веб-застосунків на Java. Він надає розробникам широкий спектр інструментів та можливостей, включаючи інверсію контролю (IoC), аспектно-орієнтоване програмування (AOP), а також підтримку тестування та інтеграції. Spring також включає в себе модулі для роботи з даними, веб-службами, безпекою та іншими[10] .

Особливість Spring полягає в тому, що він є відкритим фреймворком. Це означає, що ви можете використовувати лише ті компоненти, які вам потрібні для вашого проекту, що робить Spring гнучким і легко адаптованим.

З плюсів Java і Spring можна виділити:

– Висока продуктивність та стабільність;

– Широкий спектр інструментів та бібліотек для розробки вебзастосунків;

– Чудова підтримка з боку спільноти і багато навчальних ресурсів.

З недоліків:

– Висока складність такрива навчання, особливо для новачків;

– Вимогливість до ресурсів, особливо при використанні великих бібліотек і фреймворків;

– Потреба в додатковій конфігурації і налаштуванні для отримання максимальної продуктивності.

Враховуючи всі переваги і недоліки, потрібно також виділити те, що Spring не відповідає вимогам застосунку через те, що попри всі переваги швидкість розробки на Java буде відносно низькою, що змушує звернутись до інших інструментів.

2.3.2 Python та Django Framework

Django — це високорівневий фреймворк для веб-розробки, створений на мові програмування Python. Він спроектований так, щоб допомагати розробникам створювати комплексні веб-застосунки, базовані на базі даних, виконуючи більшість рутинної роботи за розробника, що дозволяє їм зосередитись на унікальному для їхнього проекту коді.

Django слідує принципу "DRY" (Don't Repeat Yourself), що означає, що він намагається максимально уникнути дублювання коду, надаючи набір перевикористовуваних компонентів, які можуть бути використані в різних проектах [9].

Ось деякі ключові особливості Django:

–Організованість: Django допомагає розробникам структурувати свої

проекти і код, використовуючи систему "додатків". Кожен додаток Django - це самодостатній модуль, який можна використовувати в одному або багатьох проектах;

–Батарейки включені: Django включає багато компонентів "з коробки", включаючи обробник запитів, маршрутизатор URL, систему шаблонів, ORM для роботи з базами даних, систему аутентифікації, систему адміністрування та багато іншого;

–Безпека: Django включає багато вбудованих засобів безпеки, включаючи захист від розповсюджених атак, таких як SQL Injection, Cross-site Scripting (XSS) і Cross-site Request Forgery (CSRF);

–Масштабованість: Django може ефективно працювати на великих проектах і допомагає управляти великими базами даних.

Незважаючи на багаті можливості Django, він може бути великим і складним для вивчення для новачків. Він також може бути надмірно складним для дуже простих веб-застосунків. Однак для середніх і великих проектів Django є дуже потужним інструментом.

2.3.3 Express.js

Express.js - це один з найпопулярніших та широко використовуваних фреймворків для Node.js, що служить для створення веб-додатків. Цей фреймворк, що постійно розвивається та вдосконалюється, відзначається своєю легкістю, швидкістю і простотою використання.

В основі Express.js лежить простота. Він не нав'язує конкретних правил або структур, надаючи розробникам свободу керувати процесом створення веб-додатків, зберігаючи при цьому стійкість і ефективність.

Ось більш детальний огляд ключових характеристик Express.js:

– Middleware: В серці Express.js лежить концепція middleware. Middleware – це функції, які мають доступ до об'єкту запиту (req), об'єкту відповіді (res) та

наступної функції middleware у циклі життя запиту/відповіді додатка. Middleware може виконувати різні завдання, такі як логування, обробка помилок, перевірка аутентифікації тощо;

– Маршрутизація: Express.js має вбудований маршрутизатор, який допомагає керувати запитами в різних маршрутах додатка. Крім того, він підтримує розширений набір HTTP-вербів, що полегшує розробку RESTful API;

– Інтеграція з базами даних: Express.js дозволяє інтегруватися з різними базами даних. Разом з ним можна використовувати ORM, такі як Sequelize для SQL баз даних або Mongoose для MongoDB, для спрощення взаємодії з документоподібними базами даних;

– Шаблонізація: Express.js підтримує різні двигуни шаблонів, що допомагають створювати динамічні HTML-сторінки. З Express JS можна використовувати EJS, Pug, Mustache та інші шаблонізатори, щоб створювати інтерактивні веб-сторінки;

– Безпека: Express.js дозволяє використовувати різні модулі безпеки, такі як helmet, які допомагають захистити додаток від відомих веб-загроз, таких як XSS і CSRF[15] .

Express.js - це потужний інструмент, який допомагає розробникам створювати як прості, так і складні веб-додатки з максимальною продуктивністю і мінімумом непотрібних навантажень. Цей фреймворк балансує між швидкістю виконання та часом розробки, що робить його найкращим вибором для системи для генерації автоматичних тестів для веб-застосунків.

2.4 Фреймворки для розробки автоматичних тестів

Так як основною задачею системи буде розробка автоматичних тестів, потрібно обрати фреймворк на якому будуть створюватись тести. Наразі існує декілька найбільш популярних фреймворків для тестування. Їх порівняння наведено у таблиці 2.3.

Таблиця 2.3 — Порівняння тестових фреймворків для JavaScript.

	Playwright	**Puppeteer**	**Selenium**
Підтримка браузерів	Chromium, Firefox, WebKit	Chromium, Firefox	Всі браузери
Мови програмування	JavaScript, Python, C#, Java	JavaScript, Node.js	Java, C#, Python, Ruby, JavaScript
Передвстановлені залежності	Немає	Node.js	Java, WebDriver
Запуск з допомогою проектів	Так	Ні	Ні
Запуск в "головному" режимі	Так	Так	Так
Запуск в "безголовому" режимі	Так	Так	Так (для Chrome та Firefox)
Підтримка мобільних браузерів	Так (Chromium та WebKit)	Ні	Так
Взаємодія з іншими вікнами/вкладками браузеру	Так	Так	Так
Автоматичне очікування	Так	Так	Обмежено
Підтримка проксі	Так	Обмежено	Так
Паралельне виконання тестів	Так	Обмежено	Так
Мультиплатформеність	Так	Так	Так
Вбудовані інструменти для тестування	Ні	Ні	Так (з Selenium Grid)

Виходячи з порівняння, можна спостерігати, що Playwright, Puppeteer та Selenium мають велику кількість спільних функцій та властивостей. Вони всі

дозволяють здійснювати безголове тестування, мають можливість взаємодії з різними вікнами/вкладками браузера, забезпечують можливість роботи в мережі та перехоплення запитів, і надають інструменти для створення скріншотів та запису відео.

Втім, при глибшому аналізі, Playwright вибивається вперед за деякими ключовими параметрами. Наприклад, Playwright підтримує більше браузерів за замовчуванням, включаючи Chromium, Firefox та WebKit, тоді як Puppeteer підтримує лише Chromium та Firefox, а Selenium вимагає наявності WebDriver для роботи з браузерами [16].

Крім того, Playwright не вимагає жодних передвстановлених залежностей, що спрощує його встановлення та запуск. Puppeteer, напроти, вимагає наявності Node.js, а Selenium вимагає наявності Java і WebDriver [17].

Ще одним важливим перевагою Playwright є його здатність запускати тести з допомогою вбудованих проектів, що дозволяє розгорнути більш гнучкі та модульні тестові сценарії. Puppeteer та Selenium такої можливості не мають.

У кінцевому підсумку, Playwright видається найбільш об'єктивним варіантом для автоматизованого тестування веб-додатків, завдяки його універсальності, гнучкості та масштабованості. Крім того відсутність потреби в передвстановлених залежностях означає, що для запуску тестів потрібно лише ініціалізувати репозиторій Playwright на серверній частині застосунку, після чого тести можна буде запускати безпосередньо на сервері.

Висновок до другого розділу

Цей розділ приділяв увагу ґрунтовному аналізу і порівнянню різноманітних сучасних технологій, включаючи мови програмування, фреймворки і інструменти автоматизованого тестування, що використовуються в веб-розробці.

У контексті фронтенд розробки, було досліджено та зіставлено три відомі фреймворки: React, Angular, та Vue.js. Ці інструменти мають кожен свої унікальні характеристики і переваги. React, розроблений Facebook, забезпечує велику гнучкість за рахунок своєї компонентної архітектури. Angular, розробка Google, надає впевнений, комплексний фреймворк з інтегрованою підтримкою типів. Однак, в контексті цього аналізу, Vue.js виявився найбільш привабливим вибором, завдяки його легкості, гнучкості, простоті вивчення та дружньому середовищу для розробників.

З бекенд перспективи, були досліджені та порівняні три основні технології: Java із Spring Framework, Python із Django Framework, та Express.js на базі Node.js. Java з Spring Framework пропонує міцність Java та різноманіття можливостей для створення великих веб-додатків. Python із Django Framework відомий своєю швидкістю розробки та чистотою коду. Express.js, в свою чергу, дозволяє використовувати простоту та швидкість JavaScript для створення масштабованих серверних рішень.

Щодо автоматизованого тестування, було вивчено три ключові інструменти: Playwright, Puppeteer та Selenium. Вони дозволяють автоматизувати веб-браузери для тестування веб-додатків. По закінченню аналізу, Playwright було обрано як найбільш придатний інструмент, оскільки він надає найбільший набір функцій та підтримує всі основні веб-браузери.

3 АНАЛІЗ ТА РОЗРОБКА СИСТЕМИ

3.1 Аналіз існуючих рішень

3.1.1 Selenium IDE

Selenium IDE (Integrated Development Environment) - це інструмент для запису та відтворення, що розроблявся для автоматизації тестування веб-застосунків. Selenium IDE був вперше представлений в 2006 році як розширення для браузера Firefox і був частиною набору інструментів Selenium.

Selenium IDE має дружній до користувача інтерфейс, що дозволяє розробникам і тестувальникам створювати тестові сценарії використовуючи команду для запису дій користувача. Ці сценарії можна зберегти, відтворити, відредагувати та експортувати в різні мови програмування (Java, C#, Python і Ruby).

Користувацький інтерфейс Selenium IDE наведено на рисунку 3.1

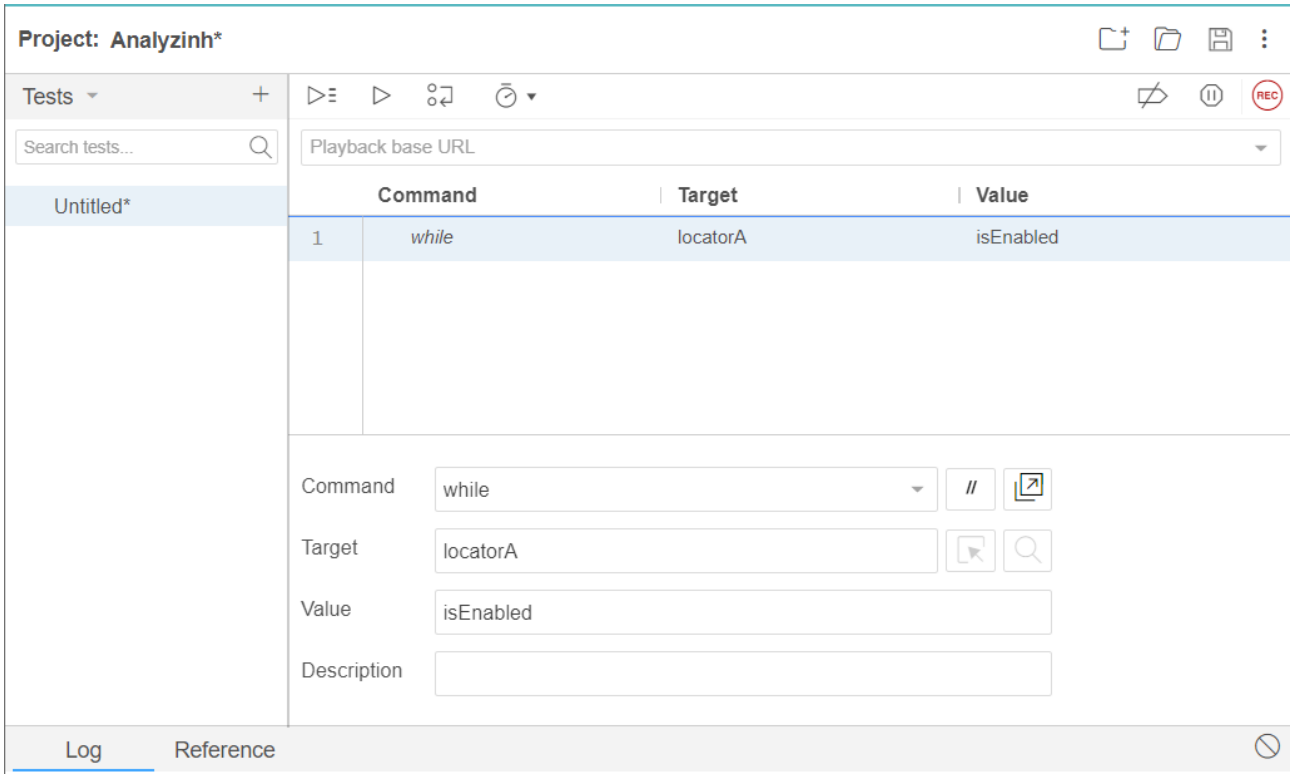

Рисунок 3.1 — Користувацький інтерфейс Selenium IDE

Головною перевагою Selenium IDE є можливість записувати та відтворювати тести. Крім цього, Selenium IDE надає можливість зберігати та експортувати тести, з метою подальшого локального виконання. Попри це, Selenium IDE не є потужним інструментом для створення тестів, адже має свої обмеження

Переваги Selenium IDE:

– Простота використання: Selenium IDE має інтуїтивний інтерфейс, що робить його доступним для непрофесійних користувачів.

–Швидкість розробки: Завдяки функції запису та відтворення, можна швидко створити прості тестові сценарії.

Недоліки Selenium IDE та можливі покращення:

– Обмежений обсяг автоматизації: Selenium IDE не надає можливостей для реалізації складних тестових сценаріїв, таких як підтримка циклів, умовних операторів, змінних тощо.

– Відсутність підтримки кросбраузерного тестування: Selenium IDE було розроблено як розширення Firefox, тому воно не може тестувати веб-застосунки в інших браузерах. Покращення підтримки кросбраузерного тестування було б важливим покращенням.

– Відсутність вбудованого механізму обробки результатів: Selenium IDE не надає вбудованого механізму для створення докладних звітів про тестування. Додання такого механізму допомогло б зробити процес тестування більш ефективним і прозорим.

– Залежність від GUI: Selenium IDE залежить від
графічного інтерфейсу, що обмежує його використання для тестування веб-застосунків без графічного інтерфейсу. Додавання підтримки тестування без GUI також було б цінним покращенням.

–Використання мови Selenese: Selenium IDE використовує власну мову команд - Selenese, що може бути незручно для тестувальників, які вже знайомі з

іншими мовами програмування, та не додає швидкості адаптації мануальних тестувальників до поширених інструментів тестування Підтримка стандартних мов програмування була б цінним покращенням.

3.2 Розробка інформаційної моделі системи

3.2.1 Визначення функціональних та нефункціональних вимог

Вимоги до програмного забезпечення можна визначити як набір функціональних та нефункціональних потреб, що мають бути реалізовані в системі.

Тобто, вимоги до програмного забезпечення можна сформулювати як набір розширених вимог користувачів. При цьому їх можна поділити на функціональні та нефункціональні вимоги [18].

Функціональні вимоги описують конкретні функції або дії, які програмне забезпечення повинно виконувати. Вони містять деталі про взаємодію користувача з програмним забезпеченням, включаючи вхідні та вихідні дані, обробку даних та інтерфейс користувача.

Нефункціональні вимоги стосуються характеристик системи в цілому, таких як продуктивність, надійність, безпека, сумісність та інші властивості, які не пов'язані з конкретними функціями програмного забезпечення. Вони також містять вимоги до документації та вимоги до підтримки та обслуговування.

Оформлені та чітко визначені вимоги до програмного забезпечення є важливим етапом в процесі розробки, оскільки вони забезпечують зрозуміння мети програми, а також визначають очікувані результати для розробників, тестувальників, керівників проекту та користувачів.

«Система для генерації автоматичних тестів» має виконувати функції генерації, зберігання тестів, можливість їх виконання та перевиконання на сервері та зберігання результатів. Основною метою розробки системи є

спрощення автоматизації тестування та спрощення інтеграції мануальних тестувальників в процеси автоматичного тестування.

Акторами системи може бути 2 категорії користувачів: Зареєстровані користувачі та незареєстровані користувачі. Незареєстрований користувач не матиме доступу до системи. Зареєстровані користувачі зможуть генерувати тести, виконувати їх на сервері та завантажувати для локального виконання.

Вимоги до даних користувачів:

1) Зареєстрований користувач:

a) Користувач має електронну пошту, що зафіксована за активним юзером в базі даних;

b) Користувач має коректний пароль, хеш якого зберігається в базі даних та дозволяє увійти в застосунок.

2) Незареєстрований користувач:

a) Користувач має електронну пошту, що не зареєстрована в базі даних;

b) Користувач не має паролю.

Функціональні вимоги:

1) Зареєстрований користувач

a) Можливість входу до особистого акаунту;

b) Можливість генерування тестів за допомогою запису дій користувача;

c) Можливість генерування тестів за допомогою UI;

d) Можливість запуску тестів на сервері;

e) Перегляд результатів поточних тестів;

f) Можливість завантаження тестів для локального виконання;

g) Можливість виходу з особистого акаунту.

2) Незареєстрований користувач

a) Можливість створення акаунту

b) Повернення на екран входу в акаунт у випадку переходу на іншу сторінку за допомогою прямого переходу за посиланням

Нефункціональні вимоги:

- Додаток має бути інтуітивно зрозумілим
- Додаток повинен містити візуальні підказки
- Додаток повинен забезпечувати human-readable формат результатів тестування

Виходячи з вимог, зазначених вище, розроблювана інформаційна система має володіти зрозумілим інтерфейсом, мати функціонал для запису дій юзера та перетворення їх в тести, парсер вихідних даних тестування.

Для забезпечення безпеки системи будуть передбачені певні обмеження:

- Вхід до системи за особистим логіном, яким є електронна пошта
- Пароль довжиною не менше 8 символів
- Хешування паролю перед записом до бази даних
- Відображення даних що стосуються користувача
- Обмеження доступу до програмних функцій для різних категорій користувачі

2.2 Функціональна модель системи

Мета розробки функціональної схеми системи - це візуальна подача взаємодії між різними компонентами системи та процесами, які вони виконують. Цей інструмент допомагає усім учасникам процесу розробки програмного забезпечення краще розуміти, як система має працювати, і як різні частини взаємодіють між собою [19].

Існують певні стандарти розробки функціональних схем та моделей, найбільш поширеними є методології IDEF0 та IDEF3.

Контекстна діаграма рівня A-0 для системи для генерації автоматичних тестів представлена на рисунку 3.1

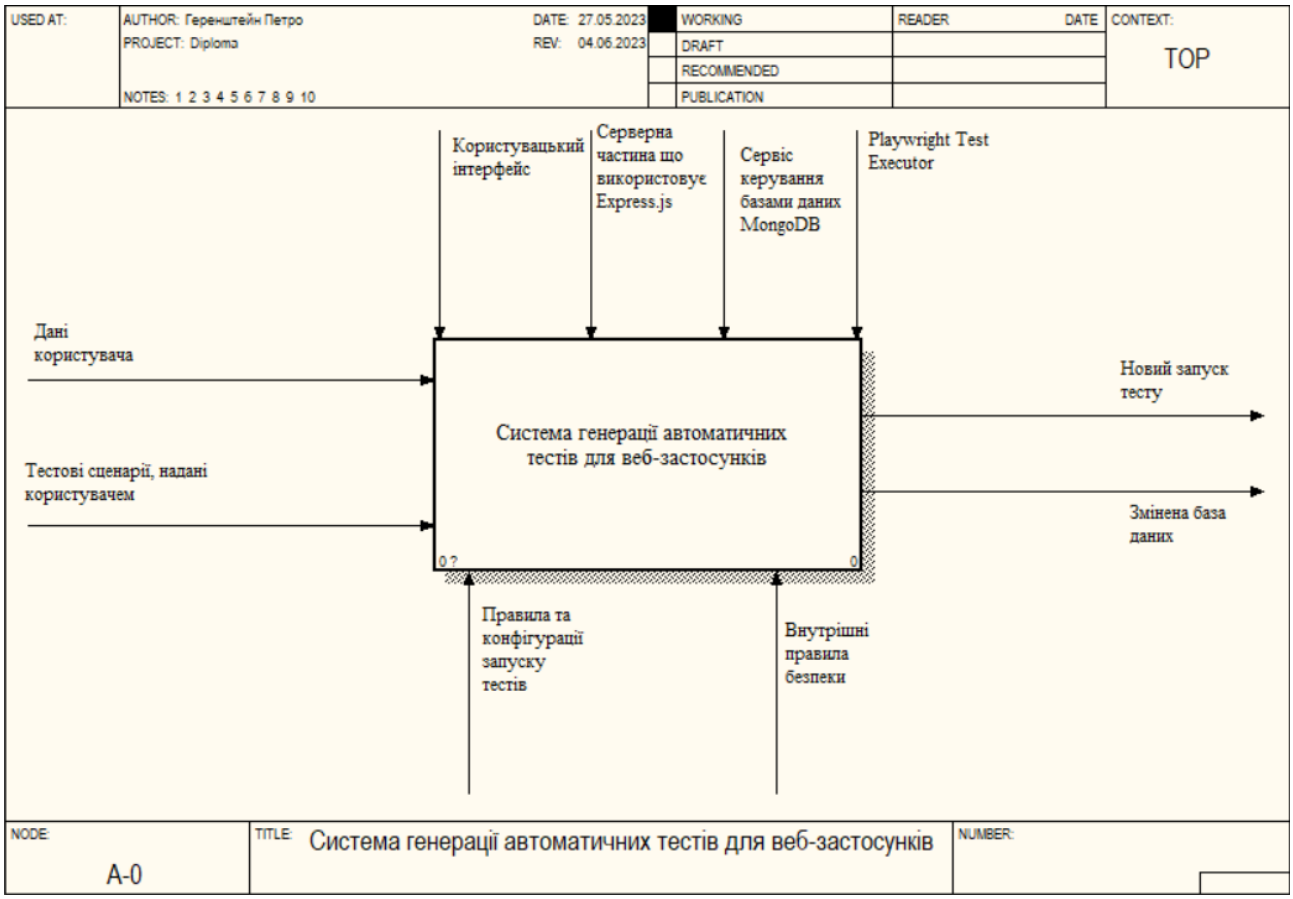

Рисунок 03.2 — Контекстна діаграма «Система для генерації автоматичних тестів»

Для більш розгорнутого представлення системи її потрібно декомпозувати. Декомпозицію 1 рівня контекстної диаграми для системи «Система генерації автоматичних тестів для веб-застосунків» зображено на рисунку 3.2

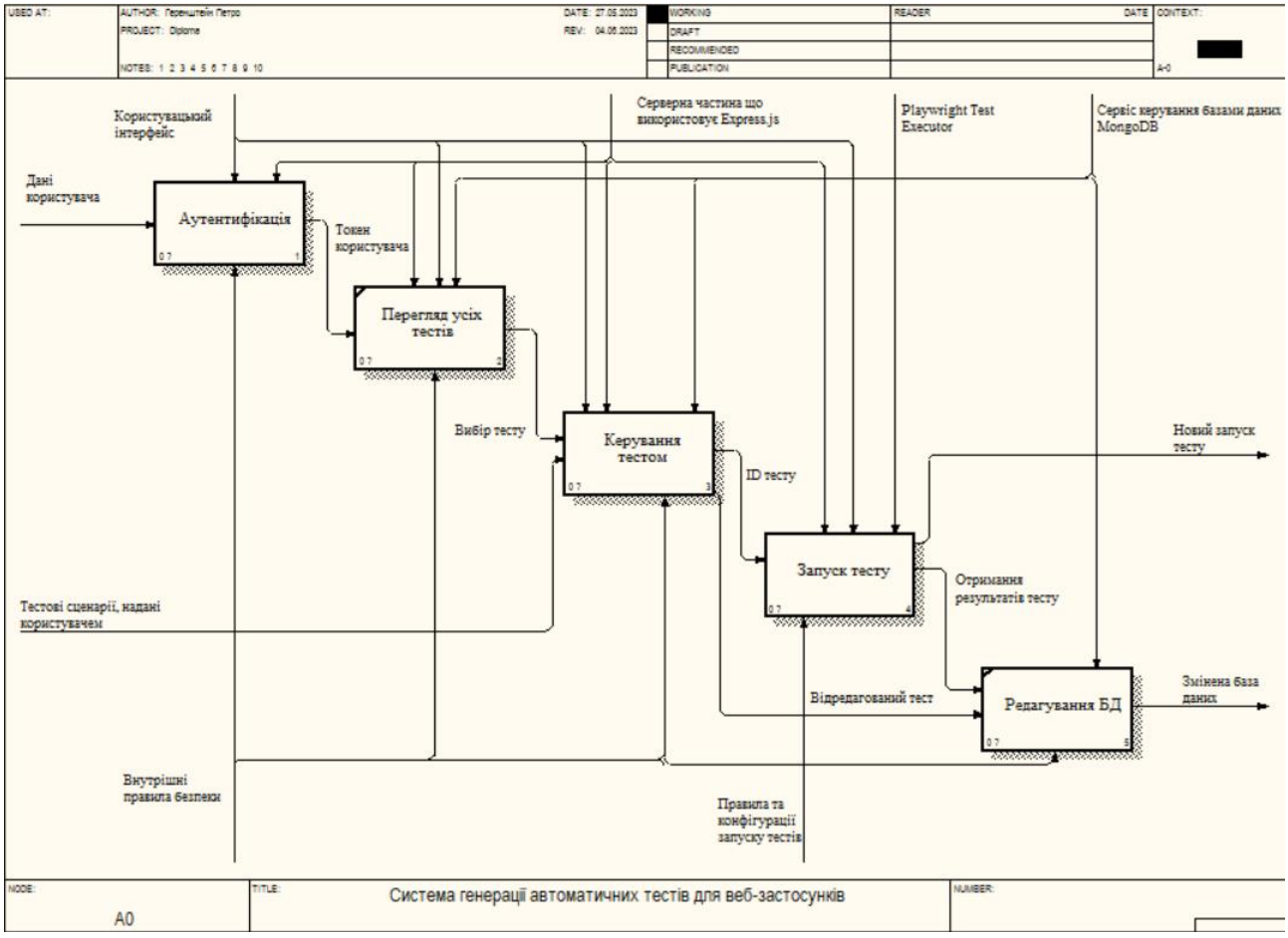

Рисунок 3.3 — декомпозиція 1 рівня контекстної діаграми «Система генерації автоматичних тестів для веб-застосунків»

Розглянемо механізми, виходи та керування що поступають на кожен процес.

Процес «Аутентифікація»:

1) Вхідні критерії:

    a) Дані користувача.

2) Механізми:

    a) Користувацький інтерфейс;

    b) Серверна частина застосунку;

    c) Сервіси керування базами даних MongoDB.

3) Керування:

    a) Внутрішні правила безпеки.

4) Вихід:

a) Токен користувача (у випадку успішної авторизації).

Процес «Перегляд усіх тестів»:

1) Вхід:

a) Токен користувача.

2) Механізми:

a) Серверна частина застосунку;

b) Сервіси керування базами даних MongoDB;

c) Клієнтський інтерфейс.

3) Керування:

a) Внутрішні правила безпеки.

4) Вихід:

a) Вибір юзером тесту.

Процес «Керування тестом»:

1) Вхід:

a) Обраний юзером тест;

2) Механізми:

a) Серверна частина застосунку;

b) Сервіси керування базами даних MongoDB;

c) Клієнтський інтерфейс.

3) Керування:

a) Внутрішні правила безпеки.

4) Вихід:

a) Унікальний ID тесту.

Процес «Запуск тесту»:

1) Вхід:

a) ID тесту

2) Механізми:

a) Серверна частина застосунку;

b) Сервіси керування базами даних MongoDB;

c) Клієнтський інтерфейс

d) Playwright test executor

3) Керування:

a) Правила та конфігурації для запуску тестів

4) Вихід:

a) Новий запуск тестів

b) Отримання результатів тестів

Процес «Редагування БД»:

1) Вхід:

a) Результати тестів;

b) Відредагований тест.

2) Механізми:

a) Серверна частина застосунку;

b) Сервіси керування базами даних MongoDB;

c) Клієнтський інтерфейс;

d) Playwright test executor.

3) Керування:

a) Внутрішні правила безпеки.

4) Вихід:

a) Оновлена БД.

Найбільш об’ємною та критичною для функціонування системи функцією є «Запуск тестів». З метою більш детального огляду, виконано декомпозицію блоку методологією IDEF0. Результат декомпозиції функції «Запуск тестів» зображено на рисунку 3.4

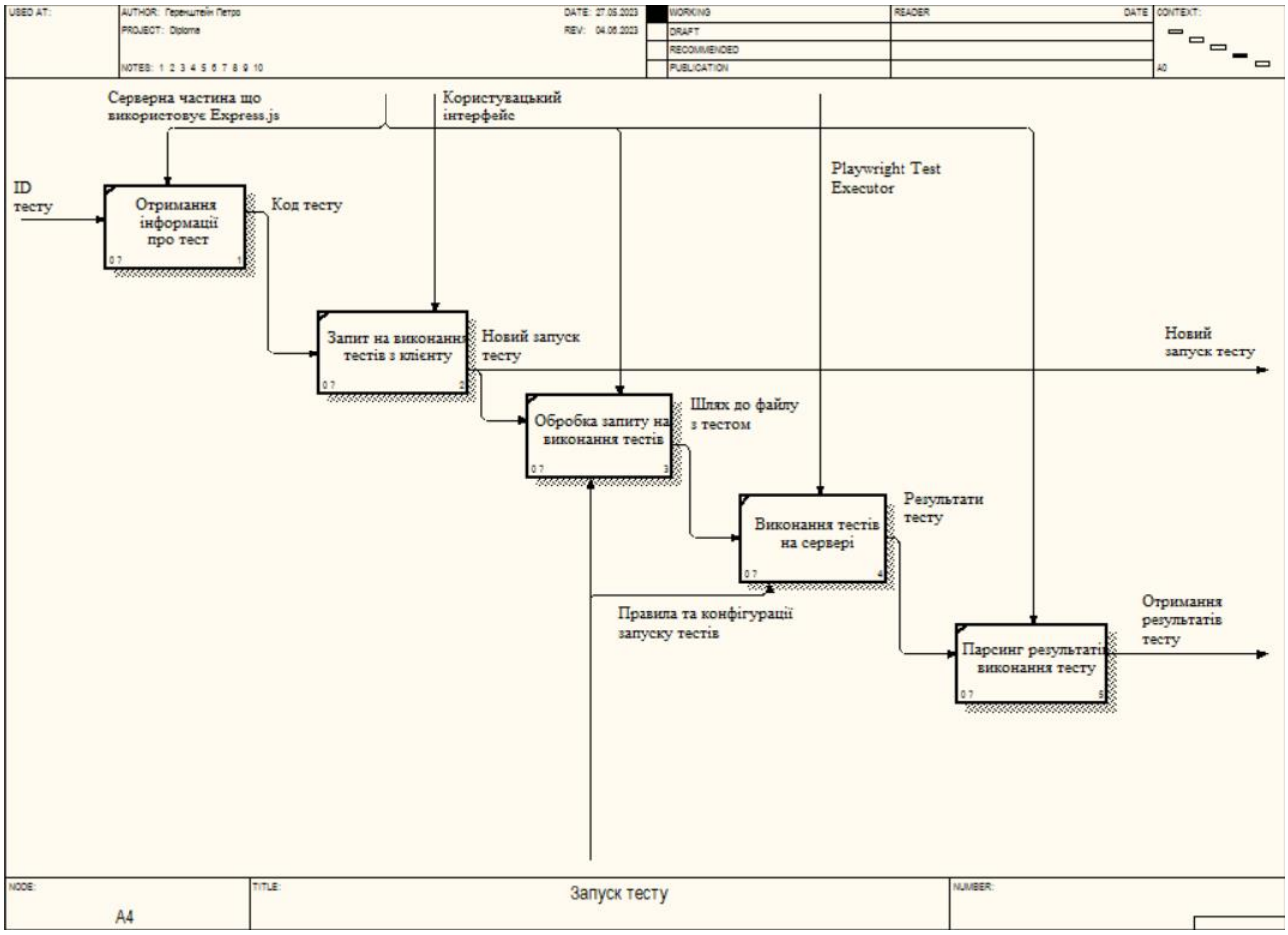

Рисунок 3.4 — Декомпозиція функції «Запуск тесту»

Розглянемо механізми, виходи та керування що поступають на кожен процес:

Процес «Отримання інформації про тест»:

1) Вхідні критерії:

а) ID тесту.

2) Механізми

а) Серверна частина застосунку.

3) Керування:

4) Вихід:

а) Код тесту.

Процес «Запит на виконання тестів з клієнту»:

1) Вхід:

а) Код тесту.

2) Механізми:

а) Користувацький інтерфейс.

3) Керування:

4) Вихід:

а) Новий запуск тесту.

Процес «Обробка запиту на виконання тесту»:

1) Вхід:

а) Новий запуск тесту.

2) Механізми:

а) Серверна частина застосунку.

3) Керування:

а) Правила та конфігурації запуску тестів.

4) Вихід:

а) Шлях до тестового файлу.

Процес «Виконання тестів на сервері»:

1) Вхід:

а) Шлях до тестового файлу.

2) Механізми:

а) Playwright test executor.

3) Керування:

а) Правила та конфігурації для запуску тестів.

4) Вихід:

а) Результати тесту.

Процес «Парсинг результатів виконання тесту»

1) Вхід:

а) Результати тесту.

2) Механізми:

а) Серверна частина застосунку;

3) Керування:

4) Вихід:

а) Отримання результатів тесту

Також, потрібно проаналізувати можливості та вимоги до модуля «Авторизація». Для цього виконуємо декомпозицію функції методологією IDEF3. Декомпозиція блоку «Автентифікація» зображена на рисунку 3.5.

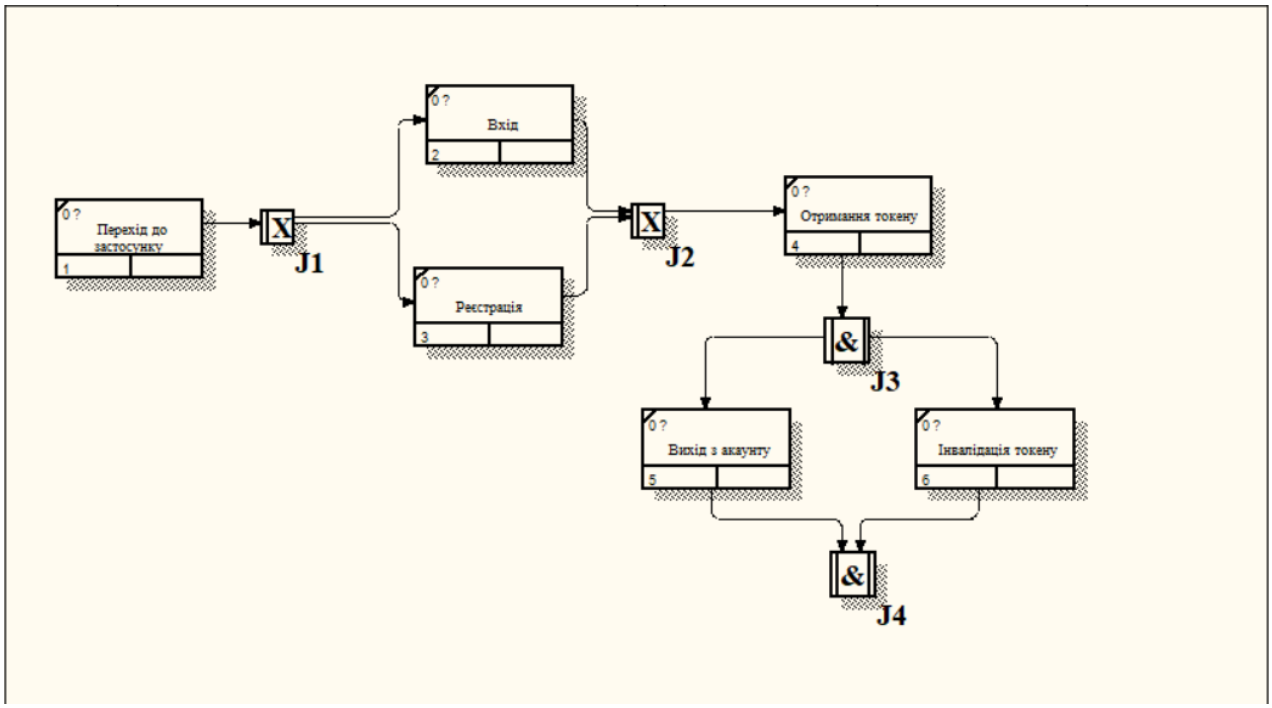

Рисунок 3.5 — Декомпозиція блоку «Автентифікація» методологією IDEF3

Декомпозиція містить два перехрестя виключаюче або було обрано тому, що користувач на етапі авторизації може або зареєструватись або увійти. У випадку реєстрації користувач має переходити на головну сторінку, а не на екран входу до застосунку. У випадку з синхронним І, у випадку якщо користувач вийде з облікового запису, його токен одразу буде інвалідовано.

Основними її компонентами діаграми у даному випадку є:

– Шість одиниці роботи – «Перехід до застосунку» «Вхід», «Реєстрація», «Отримання токену», «Вихід з акаунту», «Інвалідація токену»;

– перехрестя типу «Ексклюзивне АБО»;

– перехрестя типу «Синхронне І»

3.3. Розробка застосунку

3.3.1. Серверна частина веб-застосунку

Для розробки серверної частини було обрано Express JS. Загальна структура файлів серверної частини системи для генерації автоматичних тестів зображена на рисунку 3.6.

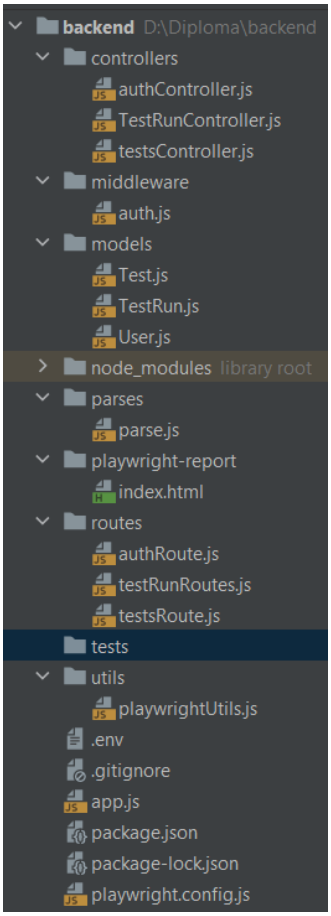

Рисунок 3.6 — структура файлів веб-застосунку

В файлі app.js знаходяться основні конфігурації серверної частини застосунку: дані для підключення до MongoDB, конфігурація роутерів та порту на якому запускається серверна частина. Зміст app.js зображено на рисунку 3.7.

Рисунок 3.7 — зміст файлу app.js

Проект містить 3 роутери, що використовуються для навігації між кінцевими точками застосунку.

У каталозі "middleware" міститься файл "auth.middleware.js". Цей файл містить в собі проміжну обробку - спеціалізовану функцію, яка має доступ до об'єктів req (запит) і res (відповідь), а також до наступної функції проміжної обробки в програмному циклі "запит-відповідь". Даний підхід реалізовано для виявлення, чи було успішно пройдено авторизацію в застосунку. У випадку неуспішної авторизації, запит, фактично, перебуває у стані очікування. Проте, при успішному проходженні авторизації, функція проміжної обробки передає запит до наступної функції в стеку обробки.

В каталозі Models знаходяться моделі даних “User”, “Test” та “TestRun”, кожна з моделей містить в стобі різні параметри що використовуються для зберігання екземплярів цих моделей в документоподібній базі даних MongoDB.

На рисунку 3.8, 3.9 та 3.10 наведено структуру моделей даних.

Рисунок 3.8 — модель даних User

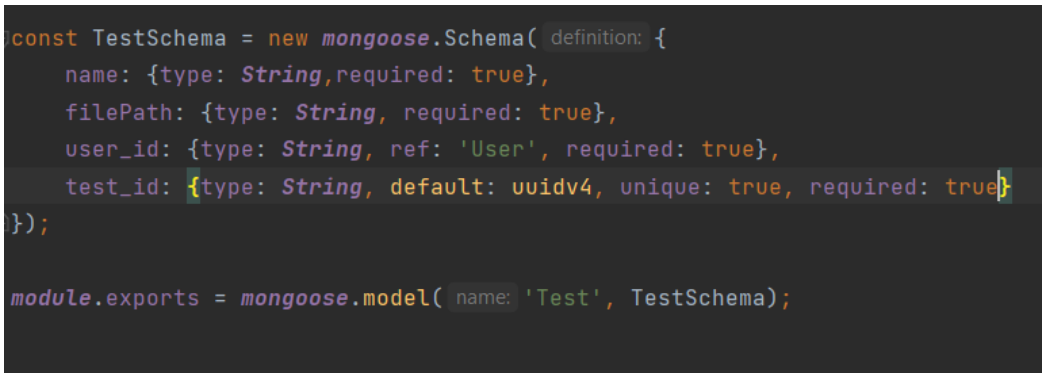

Рисунок 3.9 — модель даних Test

```
const mongoose = require('mongoose');
const { v4: uuidv4 } = require('uuid');
const errorSchema = new mongoose.Schema( definition: {
    message: { type: String, required: true }
});
const testRunSchema = new mongoose.Schema( definition: {
    testrun_id: { type: String, default: uuidv4, unique: true, required: true },
    test: { type: String, ref: 'Test', required: true },
    user: {
        user_id: { type: String, required: true },
        username: { type: String, required: true },
    },
    testTitle: { type: String, required: true },
    testResults: [
        {
            testTitle: { type: String, required: true },
            status: { type: String, required: true },
            duration: { type: Number, required: true },
            errors: [errorSchema],
        },
    ],
    status: {
        type: String,
        enum: ['passed', 'failed'],
        default: 'failed',
    },
    date: { type: Date, default: Date.now },
});

module.exports = mongoose.model( name: 'TestRun', testRunSchema);
```

Рисунок 3.10 — модель даних TestRun

На зазначених вище рисунках, представлені моделі, створені з використанням спеціалізованого пакета Mongoose. Цей інструмент забезпечує простий підхід на основі схем для структурування даних додатку. Він включає ключові функції, такі як вбудоване перетворення типів, валідація, створення запитів, а також хуки для бізнес-логіки.

Mongoose полегшує процес створення моделей, представляючи їх у вигляді простих об'єктів з парою ключ-значення. Саме так вони будуть представлені в MongoDB.

На рисунках 3.11, 3.12, 3.13 можна побачити приклад структур даних, що були створенц на основі моделей "Test",”TestRun”, “User” .

```
{
    "_id": "647325d9ea036c50772af34d",
    "name": "UUID",
    "filePath": "D:\\Diploma\\backend\\tests\\503a7e35-9bb9-40c3-9ae1-55dd4085c627.test.js",
    "user_id": "3f2122b6-3bd7-45b4-a538-5e70f1f3e1e4",
    "test_id": "ddb13b29-4a2f-4987-9854-252262f39e80",
    "__v": 0
}
```


Рисунок 3.11 — зразок структури даних Test


```
_id: ObjectId('64752234da774337ecd3e70a')
testrun_id: "8e85fbd1-7d02-46bf-8fe6-72593ddba1f5"
test: "8fa4b190-1512-4e1d-8bac-4ab0bf9f0412"
▾ user: Object
    user_id: "6677de95-a1e2-4cb1-acab-b01131f374fd"
    username: "aocrassl@gmail.com"
testTitle: "Test Test"
▾ testResults: Array
  ▾ 0: Object
      testTitle: "Test1"
      status: "passed"
      duration: 2029
    ▾ errors: Array
      _id: ObjectId('64752234da774337ecd3e70b')
  ▸ 1: Object
  ▸ 2: Object
status: "failed"
date: 2023-05-29T22:07:48.245+00:00
__v: 0
```


Рисунок 3.12 — зразок структури даних TestRun


```
_id: ObjectId('647366dd89bfe0bc78e9a818')
username: "aocrassl@gmail.com"
password: "$2a$10$yz6ZdOIiPF/3iAPa8IbEJuBPqwHyB.viCEw.CqQWt4ApmJT8T9aVu"
user_id: "6677de95-a1e2-4cb1-acab-b01131f374fd"
__v: 0
```


Рисунок 3.13 — зразок структури даних User

Між цими структурами даних присутні зв’язки типу один до одного, таким чином модель Test пов’язана з моделями User, модель TestRun з моделями Test та User.

Для реалізації цього зв’язку було використано передачу об’єкту юзера в тест на етапі створення нового тесту. Кінцеву точку для створення тестів наведено на рисунку 3.14.

```
exports.createTest = async (req, res) => {
    const { name, tests } = req.body;
    const userId = req.user.id;
    try {
        let testCode = `'Test Start' \n`;
        testCode += `const { test, expect } = require('@playwright/test');\n\n`;
        for (const testObject of tests) {
            testCode += `test('${testObject.name}', async ({ page }) => {\n`;
            testCode += `  await page.goto('${testObject.url}');\n`;
            for (const [key, value] of Object.entries(testObject.locators)) {
                testCode += `const ${key} = page.locator('${value}');\n`;
            }
            for (const action of testObject.actions) {
                testCode += `  await ${action}\n`;
            }
            for (const assertion of testObject.assertions) {
                testCode += `  await ${assertion}\n`;
            }
            testCode += '});\n';}
        const testId = uuidv4();
        const dir = path.join(__dirname, '..', 'tests');
        if (!fs.existsSync(dir)) {fs.mkdirSync(dir, options: { recursive: true })}
        const filePath = path.join(dir, `${testId}.test.js`);
        fs.writeFileSync(filePath, testCode);
        const test = new Test( doc: { name, filePath, testCode, user_id: userId, test_id: testId });
        res.json({ message: 'Test created successfully', testId, testName: name });
    } catch (err) {
        console.log(err)
        res.status(500).json({ message: 'Server error' })}
};
```

Рисунок 3.14 — кінцева точка для створення тестів

В системі реалізовано методи для отримання конкретних тестів та тестранів за їх унікальними ідентифікаторами та можливість запускати тести на сервері. В випадку запуску на сервері, в відповідь на запит повертається оброблений результат тестів. Приклад такого результату зображено на рисунку 3.15.

```
{
    "message": "Test run completed",
    "testRun": {
        "testSuiteTitle": "8fa4b190-1512-4e1d-8bac-4ab0bf9f0412",
        "testTitle": "Test Test",
        "status": "passed",
        "testResults": [
            {
                "testTitle": "Test1",
                "status": "passed",
                "duration": 4601,
                "errors": []
            },
            {
                "testTitle": "Tes21",
                "status": "passed",
                "duration": 4382,
                "errors": []
            },
            {
                "testTitle": "Test3",
                "status": "passed",
                "duration": 4528,
                "errors": []
            }
        ],
        "date": "2023-05-29T22:14:26.215Z"
    }
```


Рисунок 3.15 — результат виконання тесту

В каталозі routes знаходяться три файли маршрутизації. Вони визначають те, як серверна частина застосунку реагує на клієнтські запити до конкретних кінцевих точок. Такі кінцеві точки визначаються всередині самих файлів маршрутизаторів. Програмна реалізація роутеру для тестів наведена на рисунку 3.16

```
router.post( path: '/create', auth, TestController.createTest);
router.get( path: '/:id', auth, TestController.getTest);
router.get( path: '', auth, TestController.getTests);
router.delete( path: '/delete/:id', auth, TestController.deleteTest);
router.get( path: '/:id/run', auth, TestController.runTest);
router.put( path: '/:id/edit', auth, TestController.editTestCode);
```

Рисунок 3.16 — Маршрути наведені в роутері TestRoutes

В каталозі Controllers зберігаються контролери. Кожен файл маршрутизації вказує яким методом можна звернутись до певної кінцевої точки, також кожна вказує функцію з певного контролеру що виконується при запиті на кінцеву точку. Таким чином контролер є файлом в якому зберігається бізнес-логіка застосунку що потім експортується в файли-маршрутизатору. В застосунку реалізовано 3 контролери:

– Контролер авторизації;

– Контролер тестів;

– Контролер запусків тестів.

Кожен контролер відповідає за операції пов’язані з певною моделлю даних.

Ще одним каталогом є tests. В каталозі tests зберігаються тести, які створив користувач. Тести зберігаються в форматі ідентифікатор_тесту.test.js. Таким чином, тести можна буде виконати за допомогою команди npx playwright test назва_файлу_тесту, після чого тест буде виконано в браузері в безголовому режимі, його результат буде передано до файлу parse, звідки після парсинку результат виконання потрапить до контролеру запуску тестів, де тест буде збережено до бази даних. Для того, щоб результати було зручно парсити було обрано репортинг у форматі JSON [20].

Налаштування, пов’язані з Playwright зберігаються в файлі playwright_config.js. Ці конфігурації будуть застосовані на всіх користувачів. Вони розраховані на те, щоб тести не виконувались занадто довго, не затримуючи користувачів.

3.3.2. Розробка клієнтської частини застосунку

Front-End частина застосунку розроблена на React використовує базову структуру, в якій роутер контролює переходи між сторінками, а сторінки використовують компоненти. Сторінки при цьому можуть складатись з

декількох компонентів, крім того було використано найкращу практику в розробці на Vue – директиви.

Директиви у Vue.js — це спеціальні атрибути, які можуть бути додані до будь-якого HTML-тегу в шаблоні Vue. Вони розпізнаються завдяки префіксу v- і призначені для динамічного взаємодію з HTML елементами, залежно від даних, що визначені в Vue екземплярі або компоненті [21].

Розглянемо розроблені сторінки, починаючи з сторінки логіну. Зовнішній вигляд наведено на рисунку 3.17.

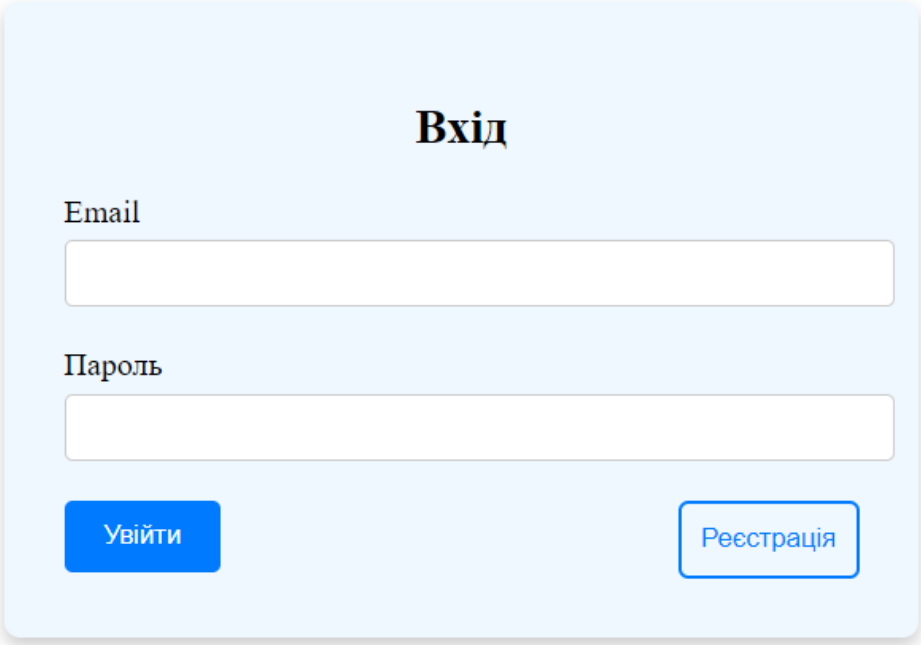

Рисунок 3.17 — Зовнішній вигляд форми для входу в систему

В системі передбачено підказки для користувача у випадку помилки: неправильний пароль, неіснуючний користувач, тощо. Сумарно для сторінок входу та реєстрації передбачено 6 підказок, не враховуючи вбудовані для елементів вводу типу email.

Зовнішній вигляд вікна реєстрації з помилкою «Такий користувач вже існує» наведено на рисунку 3.18.

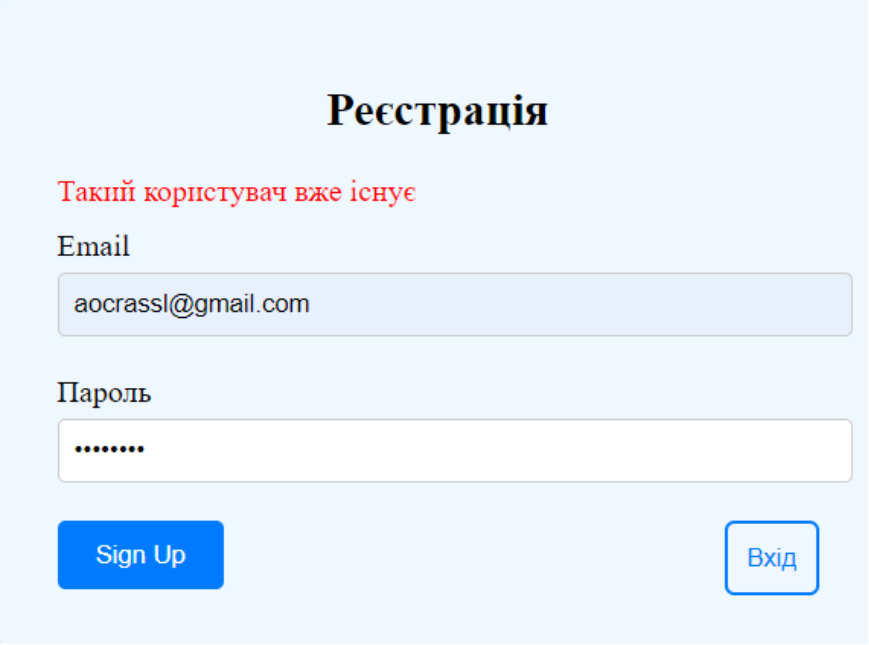

Рисунок 3.18 —Зовнішній вигляд вікна реєстрації з помилкою «Такий користувач вже існує»

Програмна реалізація для обробки помилок наведена на рисунку 3.19

```
async login() {
  try {
    const response = await api.login( credentials: {
      username: this.username,
      password: this.password,
    });
    if (response.status === 200) {
      const token = response.data.token;
      localStorage.setItem('token', token);
      await this.$router.push('/main');
    }
  } catch (error) {
    console.log(error.response.status)
      if (error.response.status === 404) {
        this.errors.general = "Такого користувача не існує";
      } else if (error.response.status === 400) {
        this.errors.general = "Неправильний пароль";
      } else {
        this.errors.general = "Сталась помилка. Будь ласка спробуйте пізніше";
      }
```

Рисунок 3.19 —реалізація обробки помилок на сторінці входу

Реалізація виконується за рахунок перевірки статусу запиту, що повертається з серверної частини після виконання запиту.

На сторінці реєстрації форма стилістично не відрізняються. На ній передбачені наступні попередження:

– Користувач вже існує;

– Пароль менше 8 символів;

– Помилка з боку серверу.

У випадку успішного входу в застосунок або реєстрації, користувач отримує токен, що зберігається у local storage в браузері, та одразу переходить на головний екран.

На головному екрані користувач може побачити всі тести, які він створив. При переході користувачу показується блокуючий екран завантаження, що очікує поки клієнтська частина отримає відповідь від серверу з усіма тестами, які користувач зберіг. Крім того на головному екрані присутня кнопка створити тест, що відкриває вікно для створення нового тесту Крім того сторінка має порожній стан. Вигляд картки на головному екрані у стані коли в користувача є тест наведено на рисунку 3.20.

Рисунок 3.20 — Вигляд картки тесту на головному екрані

Тут в користувача є декілька варіантів. Перший – натиснути на картку тесту і перейти на екран з тестом, де буде зберігатись інформація про всі запуски

тестів, код тесту з можливістю його редагування. Зона в якій користувач може переглядати тести зображена на рисунку 3.21.

Рисунок 3.21 — Область для редагування тестів

В цій текстовій області користувач може редагувати код тесту. Для зберігання користувач має натиснути на “Save”. Також з цієї сторінки користувач має можливість запустити тести. Поки тест виконується, користувачу буде показано екран завантаження, потім список запусків тестів буде оновлено автоматично. У випадку якщо тест не було запущено, користувачу буде показано порожній стан. Стан завантаження зображено на рисунку 3.22. Вигляд списку запусків тестів на сторінці тесту зображено на рисунку 3.23.

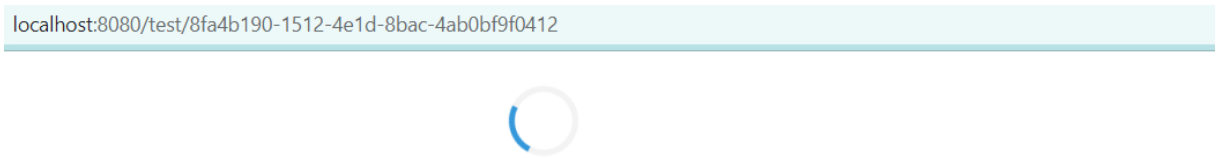

Рисунок 3.22 — Стан завантаження на сторінці тесту

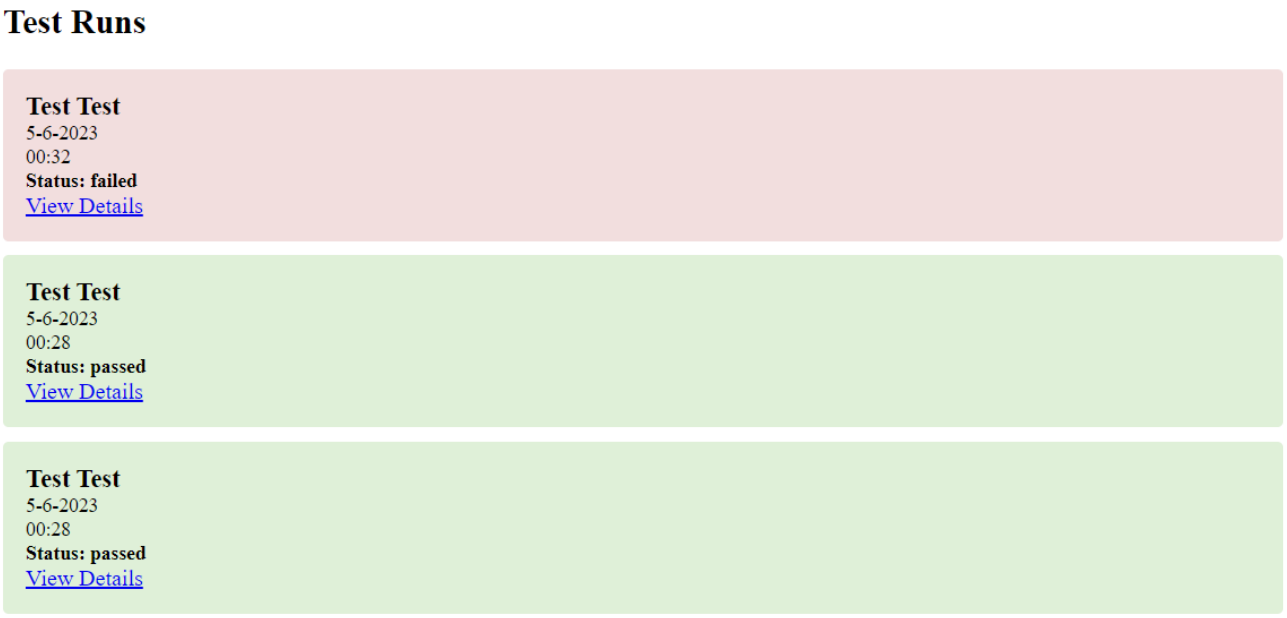

Рисунок 3.23 — Вигляд списку запусків тестів на сторінці тесту

Кожен запуск тестів має унікальний ідентифікатор, статус. При виконанні йому автоматично встановлюється час виконання. В залежності від статусу, колір картки змінюється.

Статус визначається за принципом «Успішний запуск тесту це той, при якому всі тести було виконано успішно».

Також в запусках тестів зберігається інформація про помилки та тести. Зовнішній вигляд сторінки з результатами тестового запуска зображена на рисунку 3.24.

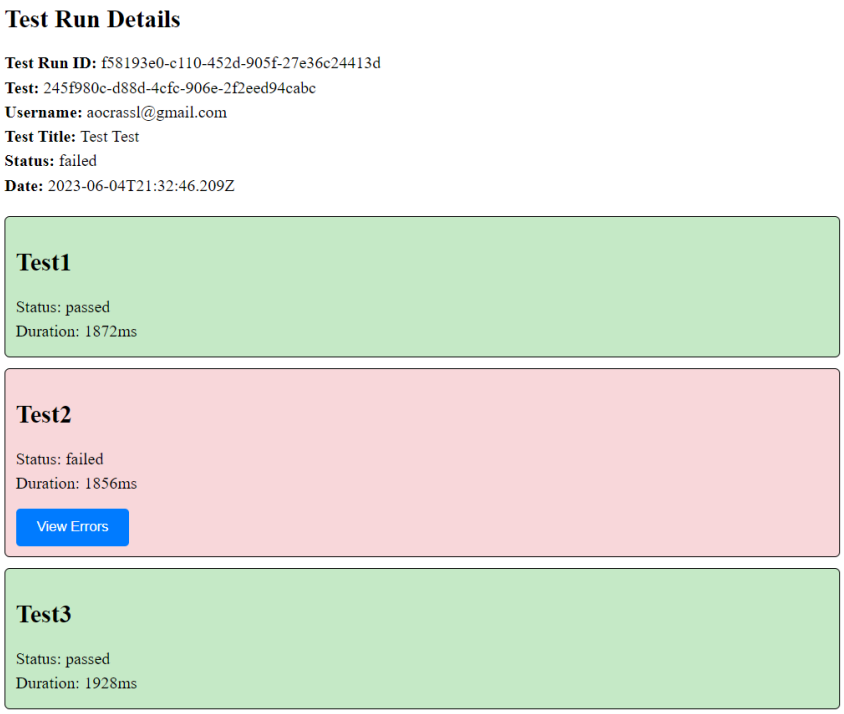

Рисунок 3.24 — Зовнішній вигляд сторінки з результатами тестового запуска

Для тестів, що не були успішно виконані, можна продивитись помилки. Зовнішній вигляд модального вікна з помилками зображено на рисунку 3.25.

Рисунок 3.25 — Модальне вікно з помилками що сталися під час виконання тесту

Крім того, користувач може створити новий тест через користувацький інтерфейс. Для цього на головному екрані потрібно натиснути на “Create Test”.

Зовнішній вигляд модального вікна для створення тестів зображено на рисунку 3.26.

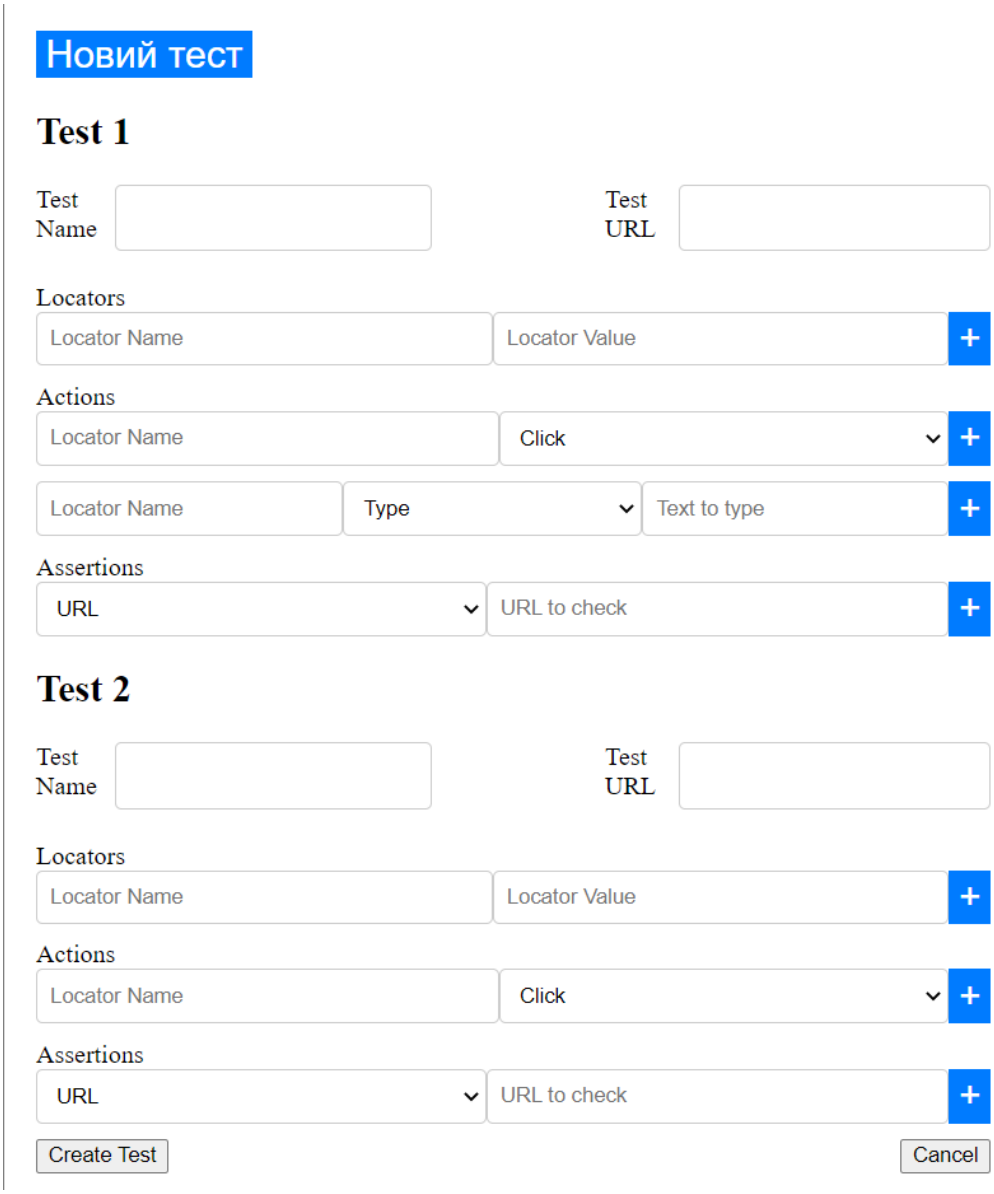

Рисунок 3.26 — Зовнішній вигляд модального вікна для створення тестів

На цьому модальному вікні передбачено створення основних компонентів тесту:

– Додавання базового URL для тесту

– Ініціалізація локаторів

– Можливість виконання з ними базових операцій

– Можливість перевірки URL сторінки

– Можливіть перевірки тексту локатору

З цього екрану користувач може створити декілька тестів, при цьому всі вони будуть записані в один тестовий файл та реактивно з’явиться на головному екрані користувача, зможе бути запущений.

Дії, що виконуються з елементами будуть виконуватись в тому ж порядку, в якому вони були вказані в модальному вікні.

При натисканні на кнопку Create Test, застосунок формує JSON, що передається в тілі запиту на кінцеву точку для створення тестів, де цей JSON обробляється і парситься в код, придатний до виконання.

При натисканні на кнопку Cancel користувач повертається на головний екран.

3.4. Ймовірні покращення

Після завершення розробки та тестування веб-застосунку «Система для генерації автоматичних тестів для веб-застосунків», що може використовуватись для перевірки концепції, було розглянуто декілька ймовірних додаткових покращень, які допоможуть підвищити продуктивність, ефективність та надійність інструменту. Такими покращеннями зокрема можуть бути:

– Хмарне розгортання: Наразі інструмент працює лише у локальному середовищі, що обмежує його доступність. Перенесення додатку на хмарний хостинг значно покращить доступність інструменту, дозволивши користувачам використовувати його незалежно від їх місцезнаходження. Хмарний хостинг також забезпечує більшу масштабованість і надійність, що дозволить

інструменту легко адаптуватися до збільшення обсягу користувачів і завантаження.

– Оптимізація генерації локаторів: Хоча наразі існує базова стратегія створення локаторів: опція з їх додаванням через користувацький інтерфейс, найбільшим покращенням для системи буде створення інструменту, що дозволить використовувати запис дій користувача з метою подальшого використання отриманих даних для створення тестів.

– Інтеграція з CI/CD інструментами: Хоча процеси звітування та кросбраузерного тестування вже впроваджені та налаштовані, інтеграція з системами неперервної інтеграції та доставки (CI/CD), такими як Jenkins, Travis CI, GitLab CI, може подальше покращити автоматизацію. Це забезпечить більш гладке злиття коду, автоматичний запуск тестів системи при змінах коду та автоматичне впровадження змін в середовище, доступне кінцевим користувачам.

– Підтримка багатомовності: Щоб зробити інструмент доступним для більш широкої аудиторії, можна додати підтримку створення тестів не лише мовою JavaScript, але й іншими мовами, які підтримує Playwright.

Всі ці покращення мають на меті покращити користувацький досвід та продуктивність системи, розширивши можливості системи та покращивши процеси навколо її розробки.

Висновок до третього розділу

Третій розділ було присвячено розробці інформаційної моделі системи. У ньому було розглянуто поняття метолологій IDEF, методи декомпозицій. Розроблено інформаційну модель системи, виконано декомпозицію ключових компонентів і функцій діаграми методологіями IDEF0 та IDEF3.

Відповідно до побудованої декомпозиції було визначено функціональні та нефункціональні вимоги до системи. З урахуванням визначених вимог було обрано інструменти для розробки серверної та клієнтської частини застосунку.

Для розробки клієнтської частини було обрано Vue.js – фреймворк, що поєднує порівняну простоту синтаксису та декларативну верстку елементів користувацького інтерфейсу.

Для розробки серверної частини було обрано Express.js. Причинами вибору стала не лише базова мова програмування – JavaScript, а і висока швидкість розробки, передбачена захищеність за допомогою Middleware запитів та високу швидкість роботи системи.

Останнім обраним компонентом був фреймворк, що використовуватиметься для генерації тестів. Враховуючи тенденції на ринку в якості базової мови для створення автоматичних тестів було обрано JavaScript, та виходячи з цього було створено порівняльну характеристику найпопулярніших сучасних фреймворків для автоматизованого тестування серед яких найкращим було обрано Playwright. Цей інструмент має найпростіший синтаксис, крім того, вбудовані проекти для браузерів та можливість запуску тестів у безголовому режимі дозволять значно спростити подальший хостинг веб-застосунку «Система для автоматичної генерації тестів».

Після визначення стеку технологій було розроблено та протестовано, за допомогою сервісу для API тестування «Postman» серверну частину застосунку, визначено вимоги до клієнтської частини та їх взаємодії. Під час тестування серверної частини було порівняно різні репортери для результатів тесту та розроблено структуру, що дозволяє обробляти та візуалізувати результати.

Останньою частиною була клієнтська частина застосунку. Після визначення основних сторінок та компонентів було розроблено основні сторінки системи, побудовано процес взаємодії компонентів створення, виконання тестів, автентифікації.

ВИСНОВОК

У процесі розробки проекту було проведено дослідження та аналіз існуючих інструментів та технологій, що застосовуються для тестування та розробки веб-застосунків. В ході аналізу серед конкурентів було обрано такі популярні інструменти, як Playwright, Vue.js, та Express.js, що надають широкі можливості для автоматизації тестування та розробки веб-застосунків, а також найбільше підходять бізнес-вимогам застосунку.

Під час аналізу системи були виявлені основні функціональні та нефункціональні вимоги до системи та бізнес-процеси. Застосовуючи методології моделювання IDEF0 та IDEF3, було побудовано інформаційну модель системи, та створено декомпозицію її компонентів, яка дозволила візуалізувати структуру та функціональність системи. Це сприяло кращому розумінню взаємозв'язків між компонентами системи та більш ефективному плануванню розробки.

Під час розробки було створено та протестовано систему для створення та керування тестами для веб-застосунків, їх редагування, запуску, тощо. Було реалізовано функціональності, які дозволяють користувачам створювати, редагувати та виконувати тести. Крім того, було розроблено можливість перегляду результатів тестових запусків, що надає користувачам підвищену зручність у відстеженні стану виконання тестів.

Під час тестування системи було виявлено та усунуто деякі помилки та проблеми, що можуть виникати під час використання системи. Це сприяло поліпшенню якості та надійності системи, а також забезпечило користувачів надійним та ефективним інструментом для керування тестами.

Після завершення процесу розробки, було визначено шляхи ймовірних покращень інструменту: хостинг, інтеграція з CI, та додавання можливості створення тестів різними мовами

Найбільшою точкою для зростання було визначено додавання можливості запису дій користувача. Це допоможе пришвидшити розробку та значно покращити користувацький досвід, проте це вимагатиме розробки інших інструментів, наприклад браузерних розширень, що будуть взаємодіяти з клієнтською та серверною частиною застосунків

Висновок розробки проекту полягає в тому, що було успішно виконано аналіз, створено інформаційну модель розроблено та протестовано систему для створення автоматичних тестів для веб-застосунків. Така система може бути легко запроваджена в існуючі в командах інженерів з забезпечення якості процеси, спростити інтеграцію мануальних тестувальників в автоматизацію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. BigWater Consulting [Веб-сайт]. URL: https://bigwater.consulting/2019/04/08/software-development-life-cycle-sdlc/
2. GeeksForGeeks [Веб-сайт]. URL: https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/
3. Guru 99 [Веб-сайт] URL: https://www.guru99.com/smoke-testing.html
4. UTOR [Веб-сайт]. URL: https://u-tor.com/topic/software-testing-stages-explained
5. AWS Amazon [Веб-сайт] URL: https://aws.amazon.com/what-is/api/
6. Рейтинг мов програмування DOU [Веб-сайт]. URL: https://dou.ua/lenta/articles/language-rating-2023/
7. Офіційна сторінка TypeScript [Веб-сайт], URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html
8. Peps.Pyton [Веб-сайт], URL - https://peps.python.org/pep-0020/
9. Oracle Java Documentation, [Веб-сайт], URL: https://docs.oracle.com/en/java/
10. Spring Framework documentation [Веб-сайт], URL: https://docs.spring.io/spring-framework/reference/overview.html
11. React.dev [Веб-сайт], URL: https://react.dev/blog/2023/03/16/introducing-react-dev
12. Angular.io [Веб-сайт], URL: https://angular.io/
13. Vue JS official [Веб-сайт], URL: https://vuejs.org/
14. NPM trends [Веб-сайт], URL: https://npmtrends.com/angular-vs-react-vs-vue
15. Django [Веб-сайт], URL: https://www.djangoproject.com/start/overview/
16. SmartBear [Веб-сайт]. URL: https://smartbear.com/learn/automated-testing/test-automation-frameworks/
17. BrowserStack test frameworks comparision [Веб-сайт] URL: https://www.browserstack.com/guide/best-test-automation-frameworks
18. Step by step: [Веб-сайт]. URL: https://step.org.ua/konspekt/infmodel/tema2

19. IDEF: навч. посібник / П.В. Шаціло, В.В. Цуркан. – К.: Вид-во ІСЗЗІ НТУУ “КПІ”, 2011. – 147 с.
20. Playwright reporters [Веб-сайт], URL : https://playwright.dev/docs/test-reporters
21. Vue.js директиви [Веб-сайт], URL: https://vuejs.org/api/built-in-directives.html#v-text