МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія автоматизованого тестування додатків на мові JavaScript»
здобувачки групи ІН.м - 22 Федорчак Анастасії Федорівни

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Анастасія ФЕДОРЧАК
(підпис)

Керівник,
в.о. завідувача кафедри,
кандидат технічних наук, доцент Ігор ШЕЛЕХОВ
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-22 Федорчак Анастасії Федорівни

1. Тема роботи: «Інформаційна технологія автоматизованого тестування додатків на мові JavaScript»»
затверджую наказом по СумДУ від «06» грудня 2023 року № 1412-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 18 грудня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки веб-додатку 3) Вибір методології, мови програмування та засоби програмної реалізації 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для прогнозування курсу валют*		
3	*Вибір методології, мови програмування та засоби програмної реалізації*		
4	*Аналіз результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 55 стр., 20 рис., 1 додаток, 20 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розробці інформаційої технології автоматизованого тестування додатків на мові JavaScript

Об’єкт дослідження — технологія автоматизованого тестування додатків на мові JavaScript.

Мета роботи — розробка інформаційої технології автоматизованого тестування додатків на мові JavaScript.

Методи дослідження —методи проєктування веб-додатку для написаття модульних тестів на мові JavaScript.

Результати — було проаналізовано можливі способи створення веб-додатків, використовуючи передові технології та інструменти розробки. Цей аналіз допоміг у створенні веб-додатку, призначеного для сприяння розробникам у кращому розумінні процесу тестування та у створенні тестів.

ВЕБ-ДОДАТОК, JAVASCRIPT, REACT
МОДУЛЬНЕ ТЕСТУВАННЯ, АСД

ЗМІСТ

ВСТУП

Актуальність. На сучасному етапі розвитку технологій програмне забезпечення є невід'ємною частиною різноманітних сфер бізнесу, виробництва та повсякденного життя. За останні роки, зросла потреба у цифровій трансформації та створенні програмних продуктів, що підтримують віддалену роботу, онлайн-комунікацію та інші аспекти віртуальної діяльності. Це призвело до значного збільшення популярності JavaScript як однієї з основних мов програмування для створення веб-додатків та веб-сайтів.

У зв'язку з цим, виникає необхідність високоякісного тестування JavaScript-програм, оскільки правильно спроектовані й функціонально відпрацьовані програми є ключовим фактором для успіху та ефективності віртуальних рішень.

Об’єкт дослідження. Тестування програмного забезпечення - це процес перевірки та оцінки того, що програмний продукт виконує все, що від нього очікують. Перевагами тестування є запобігання помилкам, зменшення витрат на розробку та підвищення продуктивності. Дуже часто етап тестування пропускають, і від цього може постраждати продукт і бізнес.Отже, тестування є невід'ємною частиною розробки програмного забезпечення. Одним з найпоширеніших методів перевірки коду є модульне тестування (Unit testing). При правильному виконанні модульне тестування може гарантувати передбачуваний функціонал одиниці коду. Крім того, достатньо протестований код забезпечує користувачам певний рівень довіри. Процес модульного тестування має кілька недоліків: важко визначити, який саме код повинен бути протестований. Інструменти автоматичної генерації тестових кейсів є поширеною альтернативою написанню тестів вручну. Однак, ці інструменти часто зосереджені на повному покритті коду, але створюють нестабільні та ненадійні тести.

Предмет дослідження. Об'єкт та предмет дослідження включають розробку програмного забезпечення для проведення модульних тестів JavaScript

коду. Аналізується спосіб покращення процесу тестування, його автоматизація та уніфікація, спрямовані на забезпечення високої якості розроблених програм.

Гіпотеза. Гіпотеза даного дослідження базується на припущенні, що впровадження програми для автоматизованого Unit тестування JavaScript додатків позитивно вплине на швидкість розробки, покращить якість програм та зменшить кількість помилок у коді.

Новизна. Однією з ключових особливостей даного дослідження є його новизна. Розробка веб-додатку, яке аналізує та рекомендує блоки коду для пріоритетного тестування, є новаторською ініціативою у підвищенні ефективності та рівня якості розробки програмного забезпечення на базі JavaScript.

Структура. Кваліфікаційна робота складається з вступу, аналітичного огляду, вибору методів рішення, програмної реалізацій, висновку, списку використаних джерел та додатку.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Види та типи тестування програмного забезпечення

Тестування програмного забезпечення - це процес, спрямований на оцінку функціональності програмного додатку з метою з'ясування, чи відповідає розроблене програмне забезпечення заданим вимогам, а також виявлення дефектів для забезпечення бездефектності продукту з метою отримання якісного продукту.[1]

Основна мета тестування ПЗ - зменшити ризик і забезпечити впевненість у тому, що програмне забезпечення буде працювати так, як очікується. Проводячи комплексне тестування упродовж всього життєвого циклу розробки ПЗ, розробники можуть підтвердити, що програмне забезпечення відповідає заданим вимогам, безперебійно функціонує в різних середовищах і є надійним. Тестування також допомагає покращити користувацький досвід. Ретельно оцінюючи дизайн інтерфейсу, зручність використання та продуктивність, можна переконатися, що програмне забезпечення відповідає потребам та очікуванням цільових користувачів.[2]

Black-box тестування базується на аналізі функціональності програми без детального знання її внутрішньої реалізації. Тестувальник створює тестові випадки, основані на вимогах до програми, враховуючи при цьому поведінку програми під час різних умов. Цей підхід дозволяє виявляти проблеми в функціональності, але не викриває структурні недоліки.

White-box тестування є методом, що базується на аналізі внутрішньої структури програми. Тестувальник створює тестові випадки, виходячи зі знання структури коду, виконуючи тестування логічних путей, умовних конструкцій та інших аспектів програми. Цей метод дозволяє виявляти помилки в коді, підвищує його якість та ефективність, але може упустити функціональні недоліки.

Grey-box тестування поєднує в собі особливості обох попередніх методів, використовуючи як внутрішні, так і зовнішні аспекти програми. В цьому методі

тестування використовуються знання про структуру програми та функціональність, щоб створити тестові випадки. Він дозволяє виявити як функціональні, так і структурні недоліки програми.

Кожен метод має свої переваги та обмеження. Black-box тестування є важливим для перевірки зовнішньої поведінки програми, white-box - для виявлення структурних недоліків, а grey-box - для поєднання цих підходів та отримання більш повного та комплексного результату. Правильно підібраний метод тестування відіграє ключову роль у підвищенні якості програмного забезпечення.

Існує багато різних типів тестування програмного забезпечення, кожен з яких має певні цілі та стратегії (Рисунок 1. 1).

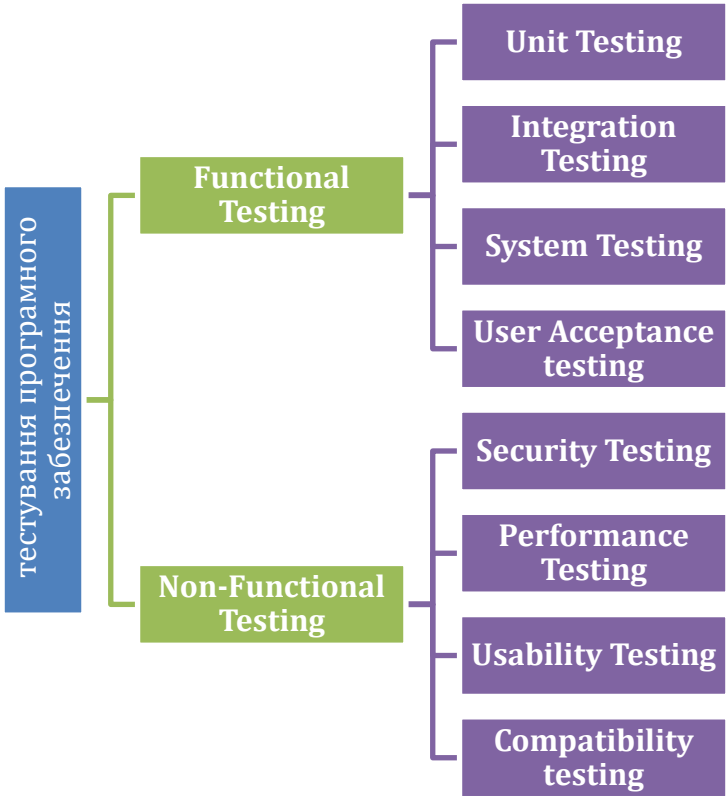

Рисунок 1. 1 - Основні види тестування програмного забезпечення

Функціональне тестування є необхідною складовою процесу розробки програмного забезпечення з кількох ключових причин. Його важливість полягає в перевірці відповідності роботи програми заданим вимогам та очікуванням

користувачів. Цей вид тестування переконується, що функції програми виконуються правильно, що кожна частина працює відповідно до специфікацій та реагує на вхідні дані відповідно.

Функціональне тестування спрямоване на ідентифікацію можливих дефектів у програмному коді, що дозволяє виправити їх на ранніх стадіях розробки. Це дозволяє покращити якість продукту та підвищити його надійність перед випуском на ринок.

Крім того, функціональне тестування сприяє підвищенню задоволеності користувачів продуктом, оскільки воно перевіряє, чи відповідає програма їхнім очікуванням та вимогам. Цей процес допомагає виявляти та виправляти помилки, забезпечуючи зручність користування та надійність функцій, що є важливим для успіху будь-якого програмного продукту.

Нефункціональне тестування оцінює, як працює програмне забезпечення з точки зору таких атрибутів, як масштабованість, надійність, зручність використання, продуктивність і безпека. Нефункціональне тестування гарантує, що програмне забезпечення не лише функціонально надійне, але й забезпечує чудовий користувацький досвід та відповідає очікуванням щодо продуктивності.

Функціональне тестування та нефункціональне тестування має свої типи, як зобразжено на Рисунок 1. 1, але у нашій роботі нас цікавить саме типи функціонального тестування.

Модульне тестування (Unit testing) включає в себе тестування невеликих, незалежних одиниць коду, щоб переконатися, що вони працюють так, як задумано. Зазвичай розробники виконують модульне тестування вручну, використовуючи методи тестування «білого ящика», які заглиблюються у внутрішню логіку і структуру самого коду. Таке тестування можна розглядати як основу для побудови надійних і стійких програмних систем.

Інтеграційне тестування (Integration Testing) - це метод перевірки того, як різні блоки або компоненти програмного додатку взаємодіють один з одним.

Воно використовується для виявлення та вирішення будь-яких проблем, які можуть виникнути при об'єднанні різних частин програмного забезпечення. Інтеграційне тестування зазвичай проводиться після модульного тестування і перед функціональним тестуванням і використовується для перевірки того, що різні модулі програмного забезпечення працюють разом, як задумано.[3]

Тестування системи (System Testing) перевіряє, що інтегровані компоненти та підсистеми програмного забезпечення працюють безперебійно та відповідають визначеним функціональним вимогам. Воно підтверджує, що система виконує свої завдання, правильно обробляє дані та надає очікувані результати. Системне тестування є однією з ключових фаз в процесі тестування програмного забезпечення. Це важливий етап, коли весь програмний продукт тестується як єдина система для перевірки його відповідності вимогам та очікуванням клієнта.

Приймальне тестування (Acceptance Testing) - це тип тестування, коли клієнт/бізнес/замовник тестує програмне забезпечення за допомогою бізнес-сценаріїв у реальному часі. Клієнт приймає програмне забезпечення лише тоді, коли всі функції та можливості працюють так, як очікувалося. Цей вид тестування може включати тестування використання, сумісності, ергономіки, а також здатність вирішувати проблеми, що виникають під час реального використання. Це останній етап тестування, після якого програмне забезпечення запускається у виробництво.[4]

Загалом, функціональне тестування є важливим етапом у розробці програмного забезпечення, де Unit тестування є найважливішим етапом функціонального тестування, оскільки дозволяє виявляти помилки в коді на ранніх етапах, що є найменш дорогими для усунення.

1.2 Unit Testing

Модульне тестування (Unit Testing) - це процес розробки програмного забезпечення, в якому найменші частини програми, що підлягають тестуванню, так звані модулі, окремо перевіряються на правильність роботи. Розробники програмного забезпечення, а іноді і співробітники відділу контролю якості, виконують модульні тести під час процесу розробки.[5] Зазвичай модульне тестування - це найперший рівень тестування, який виконується перед інтеграційним тестуванням. Кількість тестів, які потрібно виконати в кожному циклі, величезна, але час, необхідний для кожного тесту, незначний, оскільки ці одиниці коду відносно прості. Завдяки цьому розробники можуть швидко виконувати модульне тестування самостійно.

Юніт-тести зазвичай складаються з трьох етапів [6]:

1. Планування - розробники розглядають, які блоки в коді їм потрібно протестувати, і як виконати всю відповідну функціональність кожного блоку, щоб ефективно його протестувати.
2. Написання тестових кейсів та скриптів - розробники пишуть код для модульного тестування та готують скрипти для виконання коду.
3. Тестування та результати - нарешті, модульне тестування запускається, і розробники можуть виявити помилки або проблеми в коді та виправити їх.

Переваги модульного тестування:

- Раннє виявлення проблем: Модульне тестування дозволяє розробникам виявляти і виправляти проблеми на ранніх стадіях процесу розробки, перш ніж вони стануть більшими і їх буде складніше виправити.
- Покращення якості коду: Модульне тестування допомагає переконатися, що кожна одиниця коду працює за призначенням і відповідає вимогам, покращуючи загальну якість ПЗ.

- Підвищення впевненості: Модульне тестування дає розробникам впевненість у своєму коді, оскільки вони можуть підтвердити, що кожна частина програмного забезпечення функціонує належним чином.
- Швидша розробка: Модульне тестування дозволяє розробникам працювати швидше і ефективніше, оскільки вони можуть перевіряти зміни в коді, не чекаючи, поки буде протестована вся система.
- Краща документація: Модульне тестування надає чітку та стислу документацію про код та його поведінку, що полегшує іншим розробникам розуміння та підтримку програмного забезпечення.
- Скорочення часу та витрат: Модульне тестування може скоротити час і витрати, необхідні для подальшого тестування, оскільки воно допомагає виявити і виправити проблеми на ранніх стадіях процесу розробки.

Недоліки модульного тестування:

- Процес написання тестових кейсів займає багато часу.
- Юніт-тестування не охоплює всі помилки в модулі, оскільки існує ймовірність наявності помилок в модулях під час інтеграційного тестування.
- Юніт-тестування не є ефективним для перевірки помилок в UI.
- Не охоплює нефункціональні параметри тестування, такі як масштабованість, продуктивність системи тощо.
- Успіх модульного тестування залежить від розробників, які повинні писати чіткі, стислі та вичерпні тестові кейси для перевірки коду.
- Юніт-тестування може бути складним при роботі зі складними модулями, оскільки може бути важко виокремити і протестувати окремі модулі ізольовано від решти системи.

Сьогодні модульні тести можна створювати вручну, записуючи тестові кейси у вигляді коду, або автоматично за допомогою інструментів автоматичної генерації тестових кейсів. Розробники, які створюють тести вручну, часто

використовують фреймворки для тестування [8], які зазвичай надають бібліотеку та програму для написання тестів та їх запуску. Наприклад, Jest та Pytest - фреймворки для тестування JavaScript та Python. Інструменти автоматичної генерації тестових кейсів, такі як Randoop, створюють випадкові послідовності викликів методів і класів, які потім виконуються для створення тверджень. Ці послідовності та твердження потім об'єднуються для створення тестових кейсів.

1.3 Огляд наявних аналогів

Спроби знайти подібні продукти або дослідження призводить до того, що можна знайти думки про те, але не сам інструмент. Оскільки для того, щоб виконати тестування білого ящика у вигляді модульного тестування вимагає досить складних технічних рішень та розуміння коду. Розробники зазвичай використовують своє розуміння внутрішньої роботи коду та розробляють тести вручну, аналізуючи структуру та логіку програми. Такий підхід дозволяє виявляти помилки в коді та перевіряти його різні варіанти виконання. Однак, є кілька інструментів, які можуть сприяти створенню модульних тестів для JavaScript.

Examin - це інструмент для розробників, розгорнутий як розширення для Chrome, який динамічно генерує модульні тести для React-додатків, що знаходяться в розробці. Examin дозволяє розробнику візуально перевіряти модульні тести для конкретних компонентів у будь-якому стані додатку і постійно оновлювати тести при монтуванні та демонтуванні компонентів.[9] Він дійсно допомагає користувачам у створенні тестів. Але, оскільки цей інструмент ніколи не відображає код, що тестується, і не пояснює його жодним чином, можна посперечатися, чи можна його називати модульним тестуванням у білому ящику.

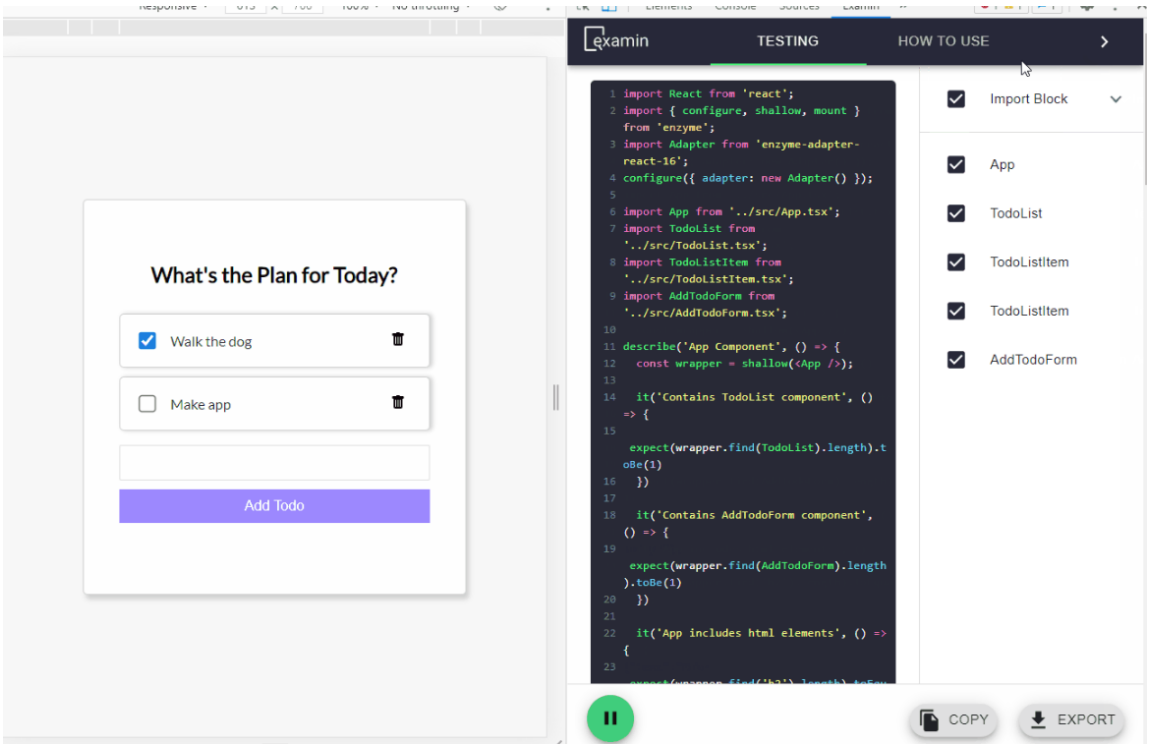

Рисунок 1. 2 – Приклад роботи з Examin

Mocha - це широко використовуваний фреймворк для тестування на JavaScript, що працює на Node.js і в браузерах. Він підтримує асинхронне тестування, запускаючи тести послідовно, що дозволяє створювати більш гнучкі та точні звіти. Mocha має широкий набір плагінів, які дозволяють розширювати його функціональні можливості та інтегрувати його з іншими інструментами тестування. Проте Mocha працює повільніше, ніж інші фреймворки для тестування програмного забезпечення при виконанні великих наборів тестів; розробнику потрібно витратити час на конфігурацію та налаштування фреймворку Mocha. Хоча Mocha є потужним фреймворком для тестування JavaScript, його вибір може бути необґрунтованим для деяких проектів через високу складність конфігурації та потребу використання інших інструментів для повного функціоналу.

Ponicode - це інструмент для автоматизованого створення unit-тестів у мові програмування JavaScript та TypeScript. Він базується на штучному інтелекті та використовує машинне навчання для аналізу коду та створення високоякісних

тестів для функцій, класів та методів у програмному коді. Цей інструмент аналізує ваш код та автоматично створює набір тестів, які враховують різні сценарії та можливі варіанти використання вашої програми. За допомогою штучного інтелекту, Ponicode вивчає та аналізує специфіку вашого проекту для генерації тестів. Інтерфейс користувача Ponicode інтуїтивно зрозумілий та простий у використанні. Користувач може вибрати функцію чи метод, для яких необхідно створити тести, та зробити це за декілька кроків. Проте, серед недоліків можна відзначити те, що інструмент здатний генерувати тести лише для тих випадків, які заздалегідь враховані у його моделі. Також, як і в усіх інструментах автоматизації, створені тести можуть не завжди бути повністю покритими та вимагатимуть періодичного перегляду та поправок для досягнення оптимальності.

1.4 JavaScript

JavaScript (JS) - це кросплатформенна об'єктно-орієнтована мова програмування, яка використовується розробниками для створення інтерактивних веб-сторінок. Вона дозволяє розробникам створювати динамічно оновлюваний контент, використовувати анімацію, спливаючі меню, кнопки, що натискаються, керувати мультимедіа тощо. JavaScript може використовуватися як на стороні клієнта, так і на стороні сервера. У той час як мови HTML і CSS використовуються для надання веб-сторінкам структури і стилю, JavaScript використовується для додавання інтерактивних елементів, які залучають користувачів.

Компанії залежать від мови JavaScript, щоб взаємодіяти зі своїми клієнтами в сучасному онлайн-світі. Серед поширених способів використання JavaScript можна виділити наступні [10]:

- Створення інтерактивних веб-сторінок. JavaScript відповідає майже за будь-яку взаємодію з веб-сайтом, що призводить до змін на сторінці. Без нього можливості Інтернету були б неймовірно обмеженими.
- Front end development. Існує низка популярних фреймворків JavaScript, які допомагають веб-сайтам створювати чудові додатки для своїх користувачів.
- Back end development. Хоча JavaScript прославився як мова інтерфейсного програмування завдяки застосуванню до HTML та CSS, він також має вражаючі можливості як мова бекенд-розробки. Такі фреймворки, як Node.js, дозволяють використовувати JavaScript для створення серверного коду.
- Ігри. JavaScript відповідає за більшість браузерних ігор.
- Штучний інтелект. Відносно недавнім напрямком використання JavasScript є штучний інтелект. Бібліотеки Javascript, такі як TensorFlow, дозволили розробникам використовувати JavaScript для машинного навчання.

Станом на 2023 рік JS залишається найпоширенішою мовою програмування вже кілька років поспіль (**Error! Reference source not found.**) [11].

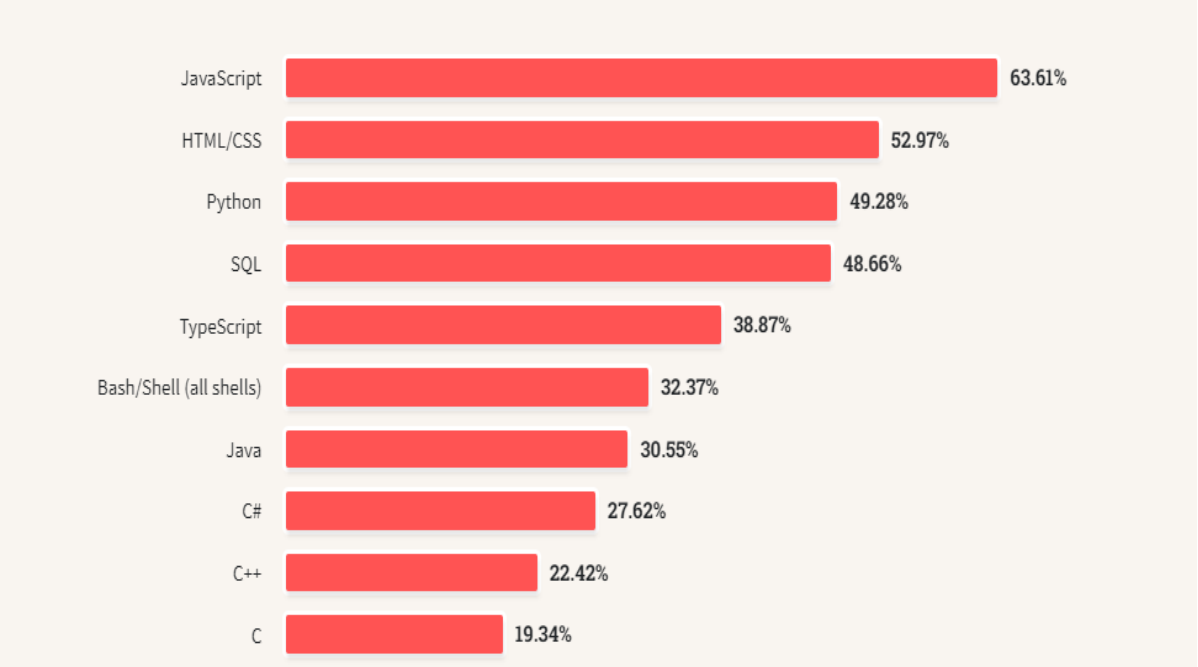

Рисунок 1. 3 – Топ 10 найпопулярніших технологій 2023 року [11]

Спочатку веб-сайти не використовували багато JavaScript, і він в основному використовувався для додавання інтерактивності до веб-сторінок, коли це було необхідно. Це означало, що файли JavaScript були досить короткими. Однак в останні роки веб-додатки стали більш складними, а це означає, що потрібно набагато більше JavaScript. Для того, щоб полегшити обслуговування цих проектів з великими обсягами JavaScript-коду, були впроваджені стандарти, що дозволяють легше розділяти JavaScript на окремі файли так звані модулі. Ці модулі можуть обмінюватися ресурсами один з одним за допомогою import та export.

Сьогодні JavaScript використовується разом з технологіями, визначеними в асинхронний JavaScript і XML (AJAX), для створення більш динамічних веб-сайтів. Це досягається, наприклад, шляхом зміни Document Object Model (DOM) і надсилання API-запитів для отримання та обробки даних. Були навіть випущені інструменти та фреймворки, які дозволяють писати як бекенд-додатки, так і десктопні додатки, використовуючи JavaScript.

JavaScript використовується не лише на клієнтській стороні (у браузері), але і на сервері за допомогою платформи Node.js. Це дозволяє розробникам створювати повноцінні веб-сервери та застосунки, які можуть взаємодіяти з базами даних, виконувати операції на сервері та відправляти відповіді клієнтам.

JavaScript постійно розвивається та оновлюється, з'являються нові функції та можливості. Останні версії стандарту ECMAScript (стандарт, на якому базується JavaScript) пропонують нові можливості, які поліпшують швидкість та ефективність мови, розширюють функціональність та сприяють покращенню розробки програмного забезпечення на JavaScript. JavaScript – це мова з динамічною типізацією. Це може зробити її більш придатною для швидкого створення прототипів, але також вважається, що фактором, що сприяє появі більш схильного до помилок коду [12]. Саме з цієї причини JavaScript потрібно ретельно тестувати перед використанням у виробництві.

1.5 Постановка задачі

У ході роботи було проведено дослідження актуальності інформаційної технології автоматизованого тестування додатків на мові JavaScript. Було виявлено, що модульне тестування є важливим етапом у розробці програмного забезпечення, оскільки дозволяє виявляти помилки в коді на ранніх етапах, коли вони є найменш дорогими для усунення.

Постановка задачі полягає у створенні інструменту для зручного та швидкого створення unit-тестів JavaScript-додатків. Задача включає розробку інтерфейсу, що дозволяє аналізувати проект за залежностями функцій та створювати тести. Результати дослідження показали, що додаток може бути корисним для людей з обмеженим досвідом у модульному тестуванні.

Необхідно реалізувати наступні задачі:

- Дослідження сфери проблеми.
- Вибір інструментів для розробки програмного забезпечення

- Провести аналіз та побудову моделі програмного продукту
- Програмна реалізація.
- Зробити тестування програмного продукту.

Розробка цієї роботи передбачає створення веб-додатку, що спрямована на легке створення модульних тестів для JavaScript-проектів. Основна ідея полягає у створенні середовища, куди користувачі можуть завантажити свій JavaScript-проект. Він повинен надати можливість аналізувати цей проект, визначати найважливіші частини коду та поради для тестування.

Ця платформа має допомагати розробникам визначати, які блоки коду краще тестувати першими, щоб максимально оптимізувати тестування та забезпечити надійність програми. Зокрема, розробникам надається можливість створювати нові тести, вносити зміни до наявних, видаляти непотрібні.

2 ВИБІР МЕТОДІВ РІШЕННЯ

2.1 Абстрактне синтаксичне дерево

Абстрактне синтаксичне дерево (АСД) – це скінченна множина, позначене і орієнтоване дерево, в якому внутрішні вершини співставлені з відповідними операторами мови програмування, а листя з відповідними операндами. Воно є фундаментальною частиною роботи компілятора. Оскільки АСД є "абстрактним", воно не має стандартного представлення, так як кожна мова може мати свій власний специфічний словник. Однак загальною концепцією, яку поділяють усі АСД, є деревоподібне представлення, де перший вузол описує точку входу в документ/програму.

АСД складається з вузлів, які відображають різні конструкції мови програмування. Вузли представляють собою синтаксичні елементи, такі як оператори, вирази, декларації функцій тощо. Кожен вузол містить інформацію про тип та структуру елемента коду. Кожен тип мови програмування має власні правила формування АСД. Наприклад, у JavaScript АСД включає вузли, які представляють оператори, вирази, декларації, інструкції та інші складові мови. Така структура даних створюється для подальшої обробки та аналізу коду.

АСД використовується при компіляції, інтерпретації та аналізі програмного коду. В процесі компіляції, вихідний код перетворюється в АСД для подальшої оптимізації та генерації виконуваного коду. АСД також можна використовувати для створення редакторів коду та інших інструментів, які забезпечують підсвічування синтаксису, завершення коду та інші функції.

За допомогою АСД можна виявляти синтаксичні та семантичні помилки у програмному коді, виконувати автоматичні перетворення, а також аналізувати зв'язки між елементами коду. АСД також дозволяє розробникам визначити структуру програми, виявити залежності та спрощувати процеси аналізу програмного коду.

АСД використовується не тільки для аналізу та оптимізації коду, але й для автоматизації рутинних завдань під час розробки. Наприклад, використання АСД може спрощувати процес рефакторингу, автоматично знаходячи та змінюючи частини коду. Він також корисний для інструментів статичного аналізу, що дозволяють виявляти потенційні проблеми в коді без його виконання.

Побудоване АСД може допомогти розробникам в розумінні того, як взаємодіють різні частини програми, допомагаючи їм приймати більш обґрунтовані рішення щодо оптимізації та удосконалення функціоналу програми. Це дозволяє розробникам краще орієнтуватися в коді та змінювати його без ризику впливу на роботу програми в цілому.

2.2 Вибір апаратно-програмного інструментарію

Для обраної мови JavaScript мало є аналізаторів коду, але ось деякі з них Acorn, Esprima або Meriyah. Для проекту було обрано версію Esprima.

Esprima - це інструмент для виконання лексичного та синтаксичного аналізу JavaScript програм. Сама Esprima також написана на JavaScript.[13] Він використовується для створення абстрактного синтаксичного дерева з вихідного коду JavaScript. Цей інструмент дозволяє розбирати складний JavaScript код у структуровану структуру даних, яка легко аналізується та обробляється програмними засобами.

Оскільки Esprima написана на JavaScript, вона може працювати в різних середовищах JavaScript, включаючи (але не обмежуючись ними):

- Сучасні веб-браузери (остання версія Edge, Chrome, Safari тощо).
- Застарілі веб-браузери (Internet Explorer 10 або новішої версії).
- Node.js.
- Движок JavaScript в Java, наприклад, Rhino та Nashorn

Основна функціональність Esprima полягає в розборі вихідного коду JavaScript і створенні АСД. Це дає можливість використовувати АСД для проведення аналізу коду, генерації власного коду, рефакторингу або створення інструментів для роботи з JavaScript.

Однією з ключових переваг Esprima є його висока продуктивність та ефективність. Це дозволяє швидко аналізувати великі обсяги коду і виконувати операції над ними без значного зниження продуктивності.

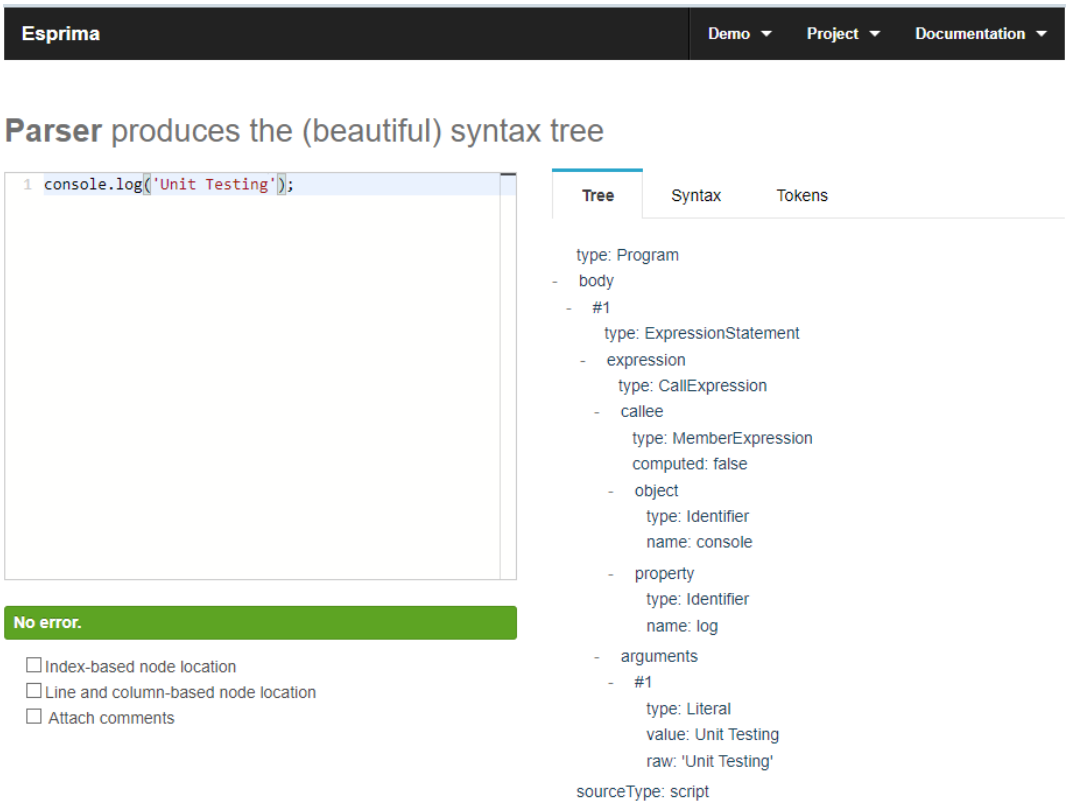

Рисунок 2. 1 – Приклад абстрактного синтаксичного дерева в Esprima

У перші 20 років свого існування JavaScript переважно використовувався для написання скриптів на стороні клієнта. Оскільки мову JavaScript можна було використовувати лише у тезі тегу <script>, розробники мусили опановувати декілька мов і фреймворків для роботи з інтерфейсами та бекендом. Пізніше виникло середовище виконання Node.js, яке забезпечує повне середовище для виконання JavaScript-програм.

Node.js є кросплатформеним середовищем виконання з відкритим вихідним кодом, спрямованим на створення швидких та масштабованих серверних і мережевих додатків. Воно базується на движку виконання JavaScript V8 та використовує подійно-орієнтовану, не блокуючу архітектуру вводу/виводу, що робить його ефективним для реально-часових додатків. Останнім часом Node.js стрімко розвивався завдяки широкому спектру можливостей, які він пропонує, а саме:

- З Node.js досить легко почати працювати. Це ідеальний вибір для початківців у веб-розробці.
- Масштабованість - він забезпечує широкі можливості масштабування додатків. Node.js, будучи однопоточним, здатний обробляти величезну кількість одночасних з'єднань з високою пропускною здатністю.
- Доступно великий набір пакетів Node.js з відкритим вихідним кодом, які можуть спростити роботу. Сьогодні в екосистемі NPM налічується більше мільйона пакетів.
- Крос-платформна підтримка дозволяє створювати SaaS-сайти, десктопні додатки та навіть мобільні додатки, використовуючи Node.js.

Для системи керування базою обрано MongoDB. MongoDB - це документно-орієнтована база даних з відкритим вихідним кодом, яка призначена для зберігання великих обсягів даних, а також дозволяє дуже ефективно працювати з цими даними. Вона відноситься до категорії баз даних NoSQL, оскільки зберігання та пошук даних в MongoDB відбувається не у вигляді таблиць. MongoDB використовує JSON-подібні документи для збереження даних, що дозволяє зберігати різноманітні типи інформації без жорсткої схеми. MongoDB відома своєю простотою в розгортанні та масштабованості, дозволяючи легко розширювати обсяги даних шляхом горизонтального масштабування. Також вона підтримує багато мов програмування та є популярним вибором для веб-додатків, аналітики.

2.3 React.js

React.js - це бібліотека JavaScript з відкритим вихідним кодом, ретельно розроблена компанією Facebook, яка має на меті спростити складний процес створення інтерактивних користувацьких інтерфейсів. Уявіть собі користувацький інтерфейс, побудований за допомогою React, як набір компонентів, кожен з яких відповідає за виведення невеликого, багаторазового фрагменту HTML-коду. [14]

React.js є популярним вибором у розробці веб-інтерфейсів з кількох причин:

- React дозволяє створювати веб-інтерфейси за допомогою компонентів, які можна повторно використовувати та комбінувати. Це спрощує розробку та підтримку великих проектів.
- Використання віртуального DOM дозволяє React ефективно оновлювати інтерфейс, зменшуючи кількість операцій з реальним DOM. Це призводить до поліпшення продуктивності веб-додатків.
- Заснований на компонентах, React легко поєднується з іншими бібліотеками та інструментами, такими як Redux для управління станом додатків, або React Router для навігації між сторінками.
- React має широку та активну спільноту розробників, що означає наявність великої кількості ресурсів, документації та сторонніх бібліотек, які спрощують розробку.
- React використовує JSX - розширення синтаксису JavaScript, що дозволяє писати HTML-подібний код безпосередньо в JavaScript. Це полегшує створення та розуміння структури компонентів та забезпечує більш зрозумілий та зручний код.

Однією з ключових переваг React є простота тестування. Він надає вбудовані інструменти для тестування компонентів, що спрощує роботу

розробників та полегшує виявлення помилок. Щоб створити програму React,треба виконати лише декілька команд, а саме:

1. `npx create-react-app magistr`- налаштує все, що потрібно для запуску програми React;
2. `cd magistr`- команда, щоб перейти до каталогу magistr;
3. `npm start` - команда, щоб запустити програму React magistr.

Усередині цього каталогу буде створено початкову структуру проекту та встановлено перехідні залежності:

```
magistr
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    ├── serviceWorker.js
    └── setupTests.js
```

Рисунок 2. 2 - початкова структура проекту

Треба лише відкрити http://localhost:3000, щоб переглянути в браузері.

Для реалізації unit-тестів у React використовуються бібліотеки, такі як Jest або React Testing Library. Jest - це популярний фреймворк для тестування JavaScript, який має вбудовану підтримку React. Він підтримує тестування всіх видів JavaScript-коду, включаючи React, Node.js, TypeScript і більшість інших проектів на JavaScript. Jest надає можливості для створення та запуску тестів,

роблячи процес тестування більш простим та ефективним. Одна з ключових переваг Jest - це його низький поріг входження. Він встановлюється легко, а почати тестування можна вже за кілька хвилин. Завдяки вбудованому підтримці для Babel, Jest автоматично розпізнає ES6 та вище, що дозволяє тестувати сучасний JavaScript-код без додаткових налаштувань.

Крім того, тести можуть включати в себе різні сценарії, такі як перевірка відображення даних, взаємодія з користувачем або перевірка реакції компонентів на зміни стану. Ці тести допомагають впевнитися в тому, що компоненти працюють відповідно до очікувань та без помилок.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Короткий опис програмної реалізації

Результатом роботи має стати інстумент, який допомагає зорієнтуватися тестувальнику/розробнику, з чого почати процес тестування. Рекомендується виконувати розробку і тестування в порядку від найменш залежної функції до найбільш залежної, тому що це дозволяє:

- Зменшити кількість помилок. Можна тестувати функції ізольовано, не турбуючись про вплив інших функцій. Це допомагає виявити і виправити помилки на ранніх етапах, коли вони є найменш дорогими для усунення.
- Збільшити продуктивність. Можна швидше розробляти та тестувати функціїї, які не залежать одна від одної. Це може допомогти скоротити час і витрати на розробку.
- Покращити якість коду. Можна використовувати код, який вже був протестований і перевірений. Це допомагає забезпечити якість і надійність коду.

Тому метрики, які були обрані, щоб допомогти тестувальнику, з чого почати, - це кількість залежних функцій та кількість залежних функцій, які функція використовує та потребує.

Процес аналізу JavaScript-проекту починається з того, що користувачеві надається можливість визначити, де знаходиться каталог проекту.(Рисунок 3. 1) Першим кроком аналізу є перегляд кожного файлу окремо. Файл аналізується за допомогою Esprima і створюється АСД. Спочатку реєструється вся інформація про експортовані ресурси та вся інформація про імпортовані ресурси. Потім відбувається обхід АСД і записується вся інформація про оголошення змінних, функцій і класів, а також будь-яка інформація про виклики функцій. У записаній інформації, серед інших параметрів, міститься інформація про те, в якій області видимості і де в АСД було знайдено оголошення або виклик. Після повного

обходу всі занотовані оголошення порівнюються з експортованою інформацією, щоб побачити, які оголошення доступні для тестування. Усім кодовим одиницям, що збігаються з експортованим активом, присвоюється унікальний ідентифікатор. Ця інформація зберігається у спеціальній таблиці бази даних. Зареєстровані виклики кодових одиниць порівнюються з інформацією про імпорт, щоб побачити, чи є виклик залежним від імпортованого ресурсу. Усі знайдені залежності зберігаються у спеціальній таблиці бази даних.

Після обробки всіх файлів база даних містить інформацію про всі доступні для тестування одиниці коду проекту. Використовуючи цю інформацію, тепер можна очистити таблицю від інформації про залежності, щоб видалити будь-яку залежність від ресурсу, не визначеного у проекті. Зовнішні оголошені ресурси не є частиною кодових одиниць для тестування, і їх використання не повинно вважатися внутрішньою залежністю. Після того, як таблицю залежностей було очищено, тепер можна обчислити залежність кодової одиниці та інформацію про залежність. Інформація про залежність кожної кодової одиниці отримується шляхом підрахунку кількості рядків таблиці, знайдених при пошуку викликів, що надходять від цієї одиниці. Інформація про кожну кодової одиниці отримується шляхом підрахунку викликів, що надходять на цю одиницю.

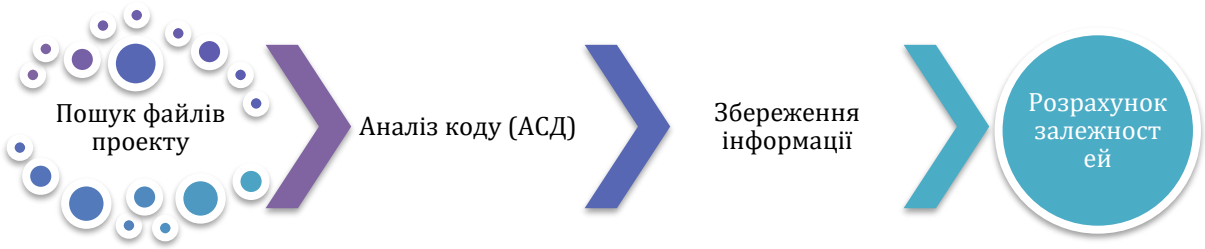

Рисунок 3. 1 - Процес аналізу JavaScript-проекту

Для того, щоб генератор тестів міг створювати тестові оголошення та виклики тверджень, у проекті, що тестується, має бути встановлений фреймворк тестування Jest. Jest - це фреймворк для тестування з відкритим вихідним кодом, побудований на JavaScript, призначений в основному для роботи з веб-додатками на основі React і React Native. [14] Цей фреймворк забезпечує широкий спектр можливостей для тестування, включаючи можливість створення різних типів тестів: від модульних тестів до інтеграційних. Jest відомий своєю швидкістю завдяки вбудованому паралельному виконанню тестів, що дозволяє ефективно виконувати тести великих проектів.

Однією з особливостей Jest є його автоматична підтримка збірки тестів без необхідності встановлення додаткових налаштувань. Він також має вбудований засіб для створення тестів з урахуванням коду за допомогою підходу "з білого ящика", що дозволяє покрити тести всіма можливими шляхами виконання коду. Варто зазначити, що Jest входить до складу пакета npm create-reactapp. Метод тестування з Jest використовується для оголошення нового тесту. Створені тести будуть використовувати вирази порівняння, такі як toBe і toEqual з Jest, щоб стверджувати, що очікуваний результат від функції досягнуто. Генератор тестів створює тести, спочатку створюючи рядкове представлення оголошення модульного тесту Jest. Декларація містить рядок, який може бути або автоматично згенерованим рядком на основі даних у тестових модулях, або користувацьким ім'ям, вказаним користувачем. Далі аналізується кожен тестовий модуль. Дані кожного модуля перетворюються на рядок, який додається до початкового тестового рядка. Коли це буде зроблено для кожного тестового модуля, тестовий рядок записується у файл. Ім'я файлу, який містить згенеровані тести, буде таким самим, як і ім'я файлу, що містить функцію, яка тестується. Однак замість розширення '.js' або '.jsx' файл, що містить згенеровані тести, матиме розширення 'urang.spec.js'. Це означає, що всі тести, пов'язані з функціями в одному файлі, будуть розміщені у відповідному файлі тестового сценарію. Крім

того, ці файли тестів будуть розміщені у тій самій папці, що й файл, який тестується. Це полегшує процес імпорту функцій, які потрібно протестувати.

3.2 Архітектура додатку та організація коду

Додаток був розроблений як веб-додаток, який буде працювати локально на комп'ютері клієнта. На Рисунок 3. 2 показано, як було структуровано додаток.

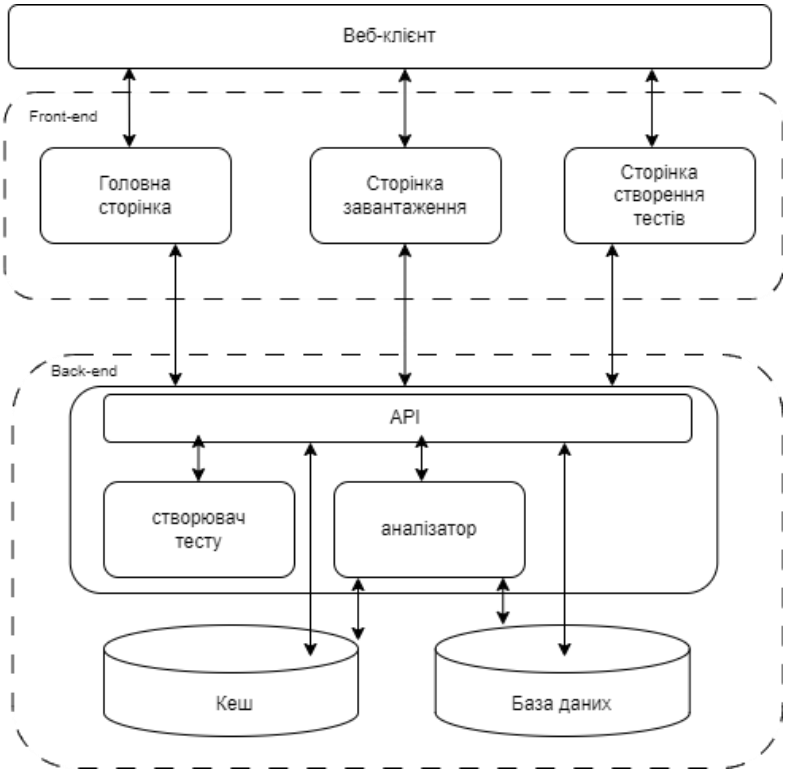

Рисунок 3. 2 – Схема структури програми

Сторінки були написані на JavaScript разом з React та Redux для для управління станами; оформлені за допомогою бібліотеки стилів Bootstrap та CSS SASS.

Внутрішня частина програми для тестування відповідає за аналіз коду проекту користувача і генерацію тестів. Бекенд складається з бази даних і кешу. Крім того, серверний додаток містить компоненти API, аналізатора та генератора тестів. Всі компоненти бекенду були написані на мові Python з використанням декількох фреймворків та бібліотек. Серед них: веб-фреймворк Flask, за

допомогою якого було створено API-додаток; парсер JavaScript Esprima, який використовується аналізатором; PyMongo - бібліотека для взаємодії з базами даних MongoDB.

3.3 Аналіз роботи програми

Розглянемо результати виконаної роботи. Інтерфейс надає користувачеві три сторінки: сторінку вибору проекту (Рисунок 3. 3), сторінку завантаження (Рисунок 3. 4), та сторінку створення тестів (Рисунок 3. 5). Сторінка вибору проекту дозволяє користувачеві вибрати проект для створення тестів. Надається можливість обрати новий проект або вже існуючий, над котрим вже працювали.

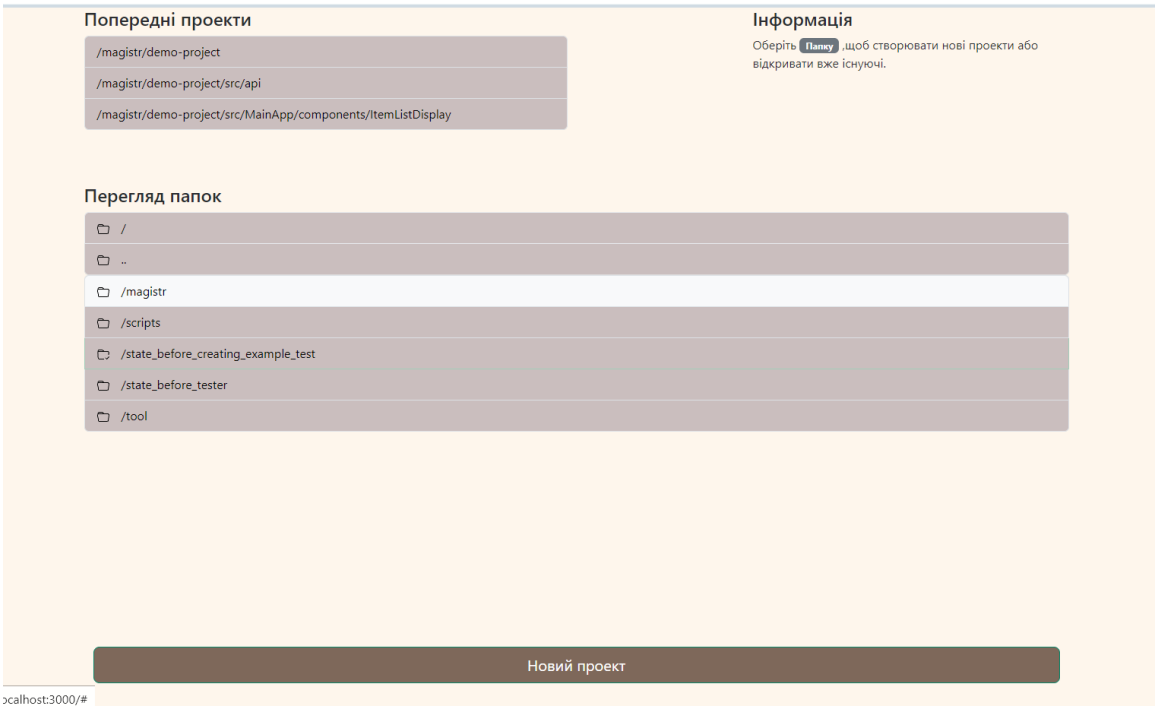

Рисунок 3. 3 – Головна сторінка

Після вибору проекту користувач буде перенаправлений на сторінку завантаження, поки аналізатор розбирає проект. Під час аналізу користувачеві буде показано файл, який аналізується, а також кількість файлів, які залишилося проаналізувати. Коли весь проект буде проаналізовано, користувач буде перенаправлений на сторінку створення тестів.

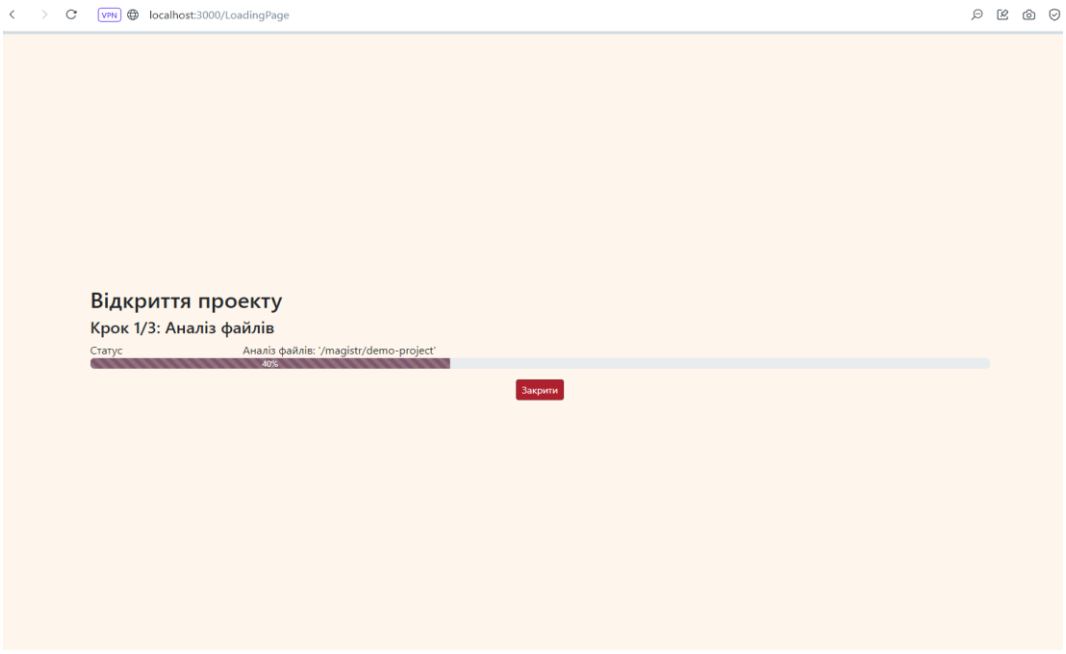

Рисунок 3. 4 – Сторінка завантаження

Головна сторінка надає користувачеві: можливість створювати тести; інформацію про залежності та залежні елементи, щоб допомогти визначити пріоритетність порядку, в якому слід тестувати функції.

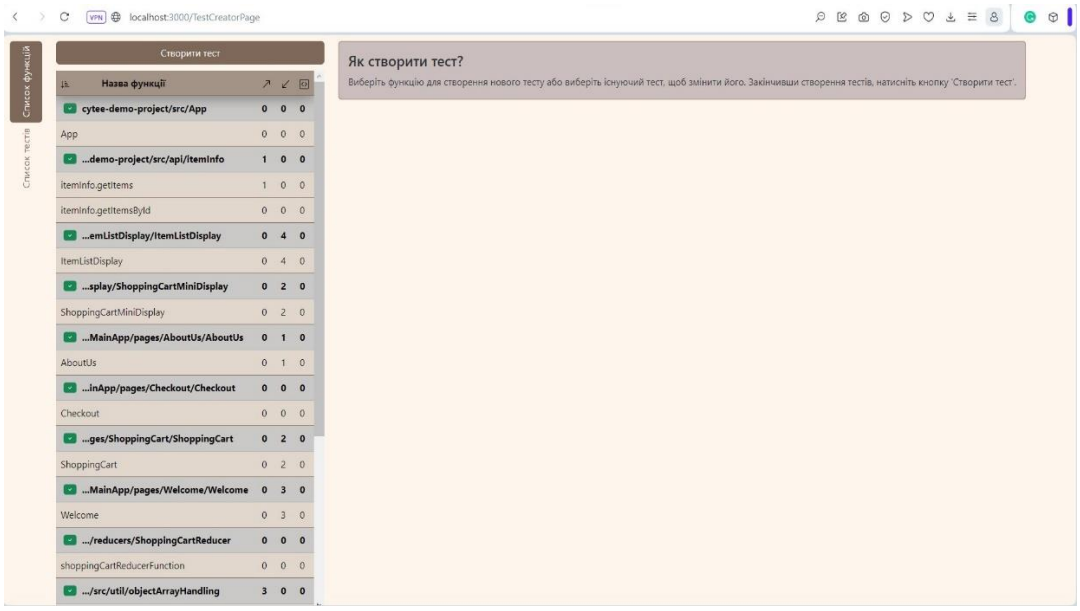

Рисунок 3. 5 – Сторінка створення тестів

Для комфортної роботи можна відсортувати функціїї за кількістю функцій від яких вони залежать (Рисунок 3. 6) та за кількістю функцій які залежать від них (Рисунок 3. 7).

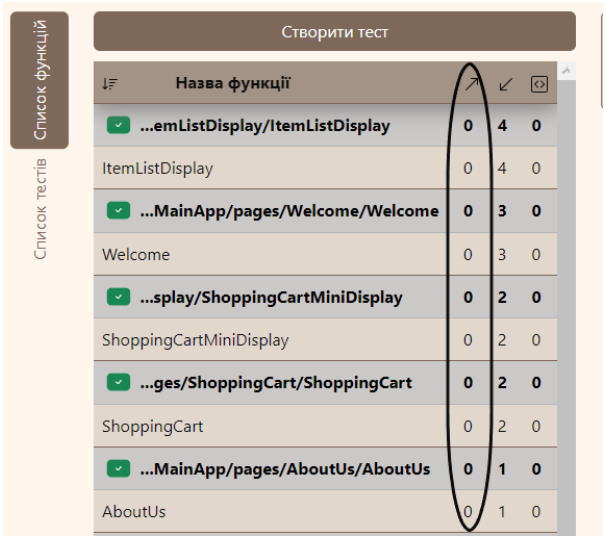

Рисунок 3. 6 – Сортування за залежністю

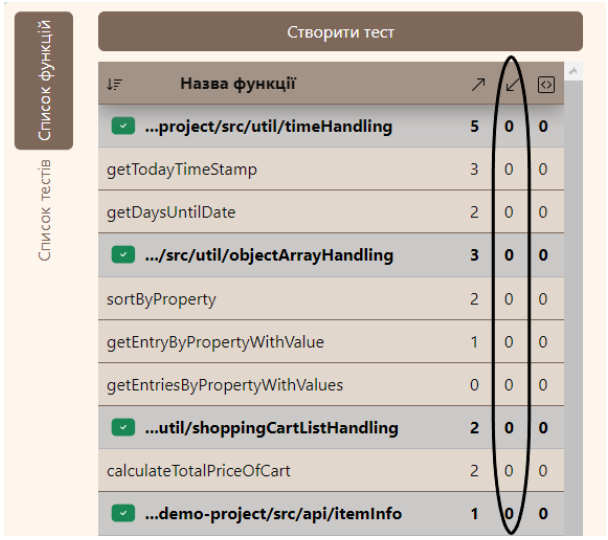

Рисунок 3. 7 – Сортування за залежністю

Якщо відкрити проект над яким вже працювали, то можна переглянути список вже написаних тестів на вкладці «Список тестів» (Рисунок 3. 8).

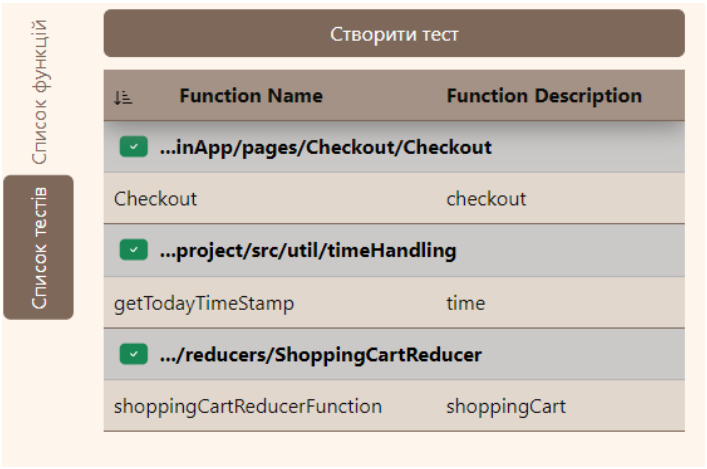

Рисунок 3. 8 – Перегляд вже створених тестів

Якщо змінилась функція або є сумніви у створенному тесті, то на цій же вкладці можна редагувати створені тести обравши його з наявного списку. (Рисунок 3. 9).

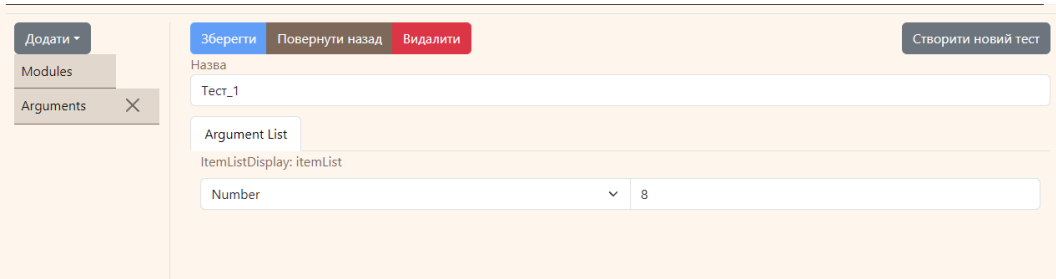

Рисунок 3. 9 – Функція редагування тестів

Після редагування тест можна зберегти або назавжди видалити, якщо це необхідно.

3.4 Тестування

Демонстраційний додаток - це веб-сайт, що нагадує сторінку електронного магазину. Він складається лише з фронтенду, який імітує виклики до бекенду. Інтерфейс складається з трьох сторінок, які включають: головну сторінку, сторінку кошика та сторінку про магазин. Головна сторінка дозволяє користувачам переглядати та додавати товари до кошика. Користувач може відвідати сторінку кошика для того, щоб імітувати покупку. Сторінка «Про нас» містить лише текст про магазин без додаткових можливостей. Додаток був зібраний і запущений з JavaScript-версії пакета npm create-react-app. В результаті для управління станом та побудови інтерфейсу було використано React.

У поточному стані програма здатна створювати модульні тести за допомогою Argument List, Return Value та Exception. (Рисунок 3. 10-Рисунок 3. 12). Argument List дозволяє користувачеві вказати параметри, які слід передати методу.

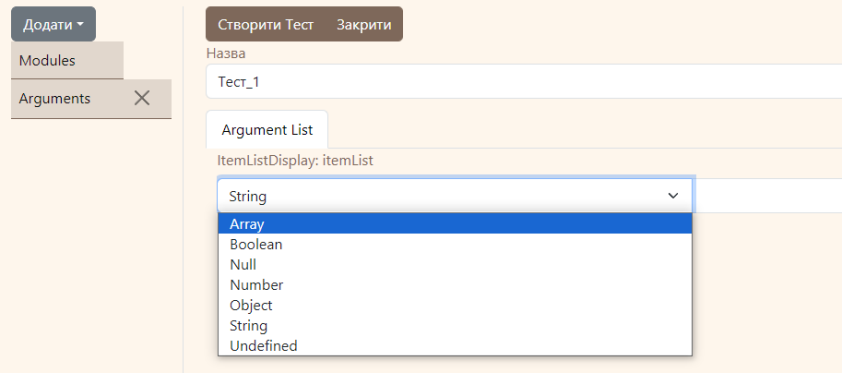

Рисунок 3. 10 – Створення теста за допомогою Argument List

Значення, що повертаються з методу, можна перевірити за допомогою Return Value.

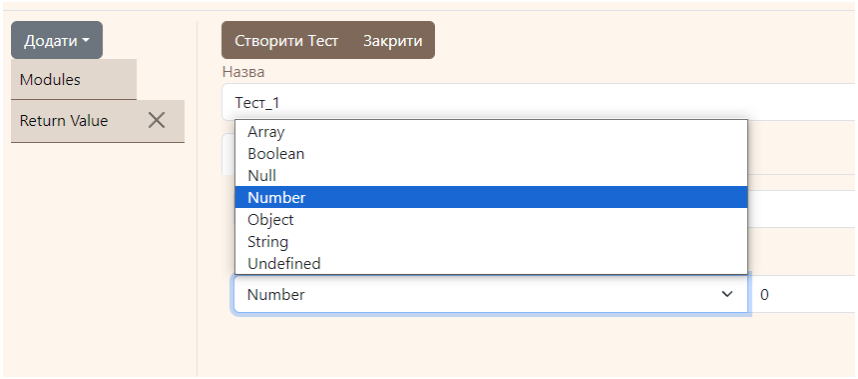

Рисунок 3. 11 – Створення теста за допомогою Return Value

Він надає можливість перевірити, чи повертається певне значення, або що значення не повинно повертатися. Нарешті, Exception дозволяє користувачеві перевірити, чи виникає певна помилка, чи ні. Хоча JavaScript є динамічно типізованою мовою, вона все ще використовує вбудовані структури даних для розділення різних типів значень. З цих структур даних тестова програма дозволяє користувачеві вказати, чи повинен аргумент або значення, що повертається, мати логічний, нульовий, невизначений, рядковий, числовий, об'єктний або масивний тип.

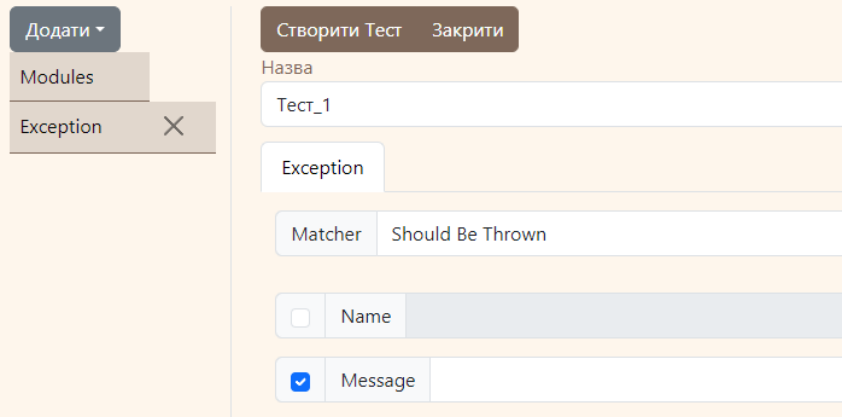

Рисунок 3. 12 – Створення теста за допомогою Exception

У демонстраційному додатку було створено функцію getDiscountPercentage, яка обчислює, скільки відсотків знижки у відсотках становить нова ціна порівняно зі старою, для неї створюється тест, який перевірятиме відмову від неправильного типу для currentPrice (Рисунок 3. 13).

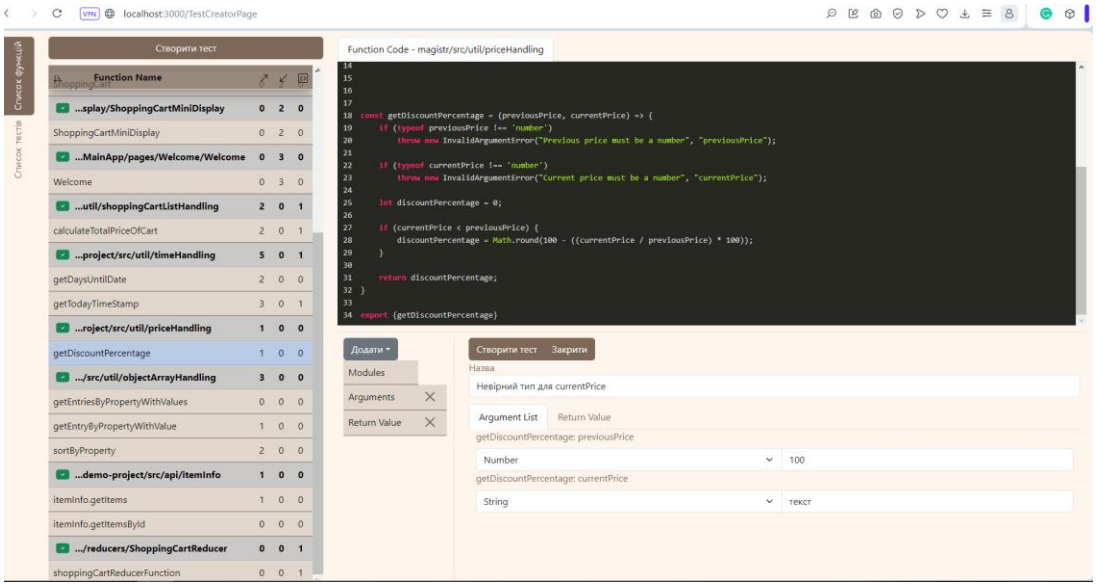

Рисунок 3. 13 - Приклад створення модульного тесту

Після створення тесту було автоматично створенно файли для кожного тесту окремо, які знаходяться у тій самій папці, що і функція.

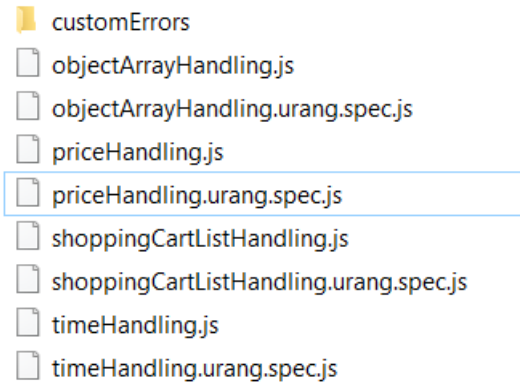

Рисунок 3. 14 – Список створених тестів

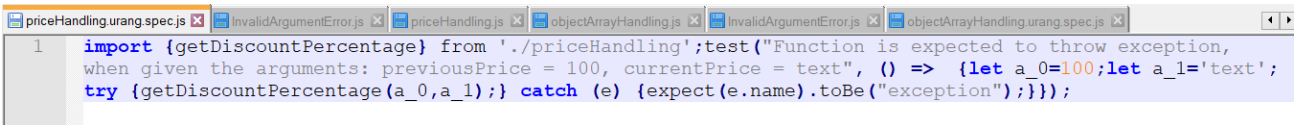

Рисунок 3. 15 - Приклад згенерованого модульного тесту

Результатом розробки став інстурмент, який дає можливість розробнику створювати модульні тести до вже готового продукту.

ВИСНОВКИ

Робота присвячена інформаційній технології автоматизованого тестування додатків на мові JavaScript, яка розкриває ключові аспекти створення та використання модульного тестування в програмуванні. Дослідження було зосереджене на вивченні актуальності використання інструментів тестування для розробки програм на JavaScript.

Перше, що було визначено, це зростаюча популярність мови JavaScript у розробці веб-додатків, що робить важливим використання автоматизованих методів тестування для забезпечення якості програмного забезпечення.

Дослідження дозволило виявити актуальність використання інструментів для створення unit тестів. Це особливо важливо, оскільки тестування дозволяє розробникам ефективно перевіряти окремі компоненти програми на коректність роботи і виявляти помилки на ранніх етапах розробки. Крім того, дослідження продемонструвало потребу в створенні інструментів для тестування відповідно до специфіки JavaScript.

Загальний висновок полягає в тому, що інформаційна технологія автоматизованого тестування додатків на мові JavaScript є важливим напрямком розробки програмного забезпечення. Вона забезпечує високу якість програм шляхом покращення процесу тестування, прискорює розробку та покращує надійність веб-додатків.

Розроблена програма є потужним інструментом для розробників мови JavaScript, що дозволяє створювати unit-тести з великою швидкістю та ефективністю. Забезпечуючи зручний інтерфейс та широкий спектр функціональних можливостей, програма стає невід'ємною складовою розробки програмного забезпечення.

Майбутні плани та можливості для вдосконалення:

- Забезпечення підтримки більш широкого спектру мов: Розширення можливостей підтримки інших мов програмування, що дозволить розробникам з різних спеціалізацій використовувати цей інструмент.
- Удосконалення інтерфейсу користувача: Покращення інтерфейсу для забезпечення ще більшої інтуїтивності та комфорту користувача, спрощуючи процес створення та управління тестами.
- Розширення документації та навчальних матеріалів: Постійне оновлення документації та створення навчальних матеріалів для новачків для забезпечення кращого розуміння можливостей програми та її оптимального використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Is Software Testing | Everything You Should Know [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestingmaterial.com/software-testing/.
2. 21 Types of Software Testing Every Engineer Should Be Using for Better Results [Електронний ресурс] – Режим доступу до ресурсу: https://stratoflow.com/types-of-software-testing/.
3. Types of Software Testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/types-software-testing/.
4. Types Of Software Testing: Different Testing Types With Details [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/types-of-software-testing/
5. unit testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchsoftwarequality/definition/unit-testing#:~:text=Unit%20testing%20is%20a%20software,tests%20during%20the%20development%20process.
6. Unit Testing: Definition, Examples, and Critical Best Practices [Електронний ресурс] – Режим доступу до ресурсу: https://brightsec.com/blog/unit-testing/.
7. Unit Testing – Software Testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/unit-testing-software-testing/
8. Sommerville I. Software Engineering. 9th ed. 2011. 773 p. – C. 6–15.
9. Introducing Examin — An automated unit test generation tool for React [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/codex/introducing-examin-an-automated-unit-test-generation-tool-for-react-19aaeacf6ff6.
10. JavaScript Demystified: Why You Should Learn This Essential Language [Електронний ресурс] – Режим доступу до ресурсу:

https://codeinstitute.net/global/blog/what-is-javascript-and-why-should-i-learn-it/.

11. Most popular technologies [Електронний ресурс] – Режим доступу до ресурсу: https://survey.stackoverflow.co/2023/#technology.
12. Chapter 1. Getting Started [Електронний ресурс] – Режим доступу до ресурсу: https://docs.esprima.org/en/latest/getting-started.html.
13. What is React.js? Uses, Examples, & More [Електронний ресурс] – Режим доступу до ресурсу: https://blog.hubspot.com/website/react-js.
14. Jest Framework Tutorial: How to use it? [Електронний ресурс] – Режим доступу до ресурсу: https://www.browserstack.com/guide/jest-framework-tutorial?utm_source=google&utm_medium=cpc&utm_platform=paidads&utm_content=645400465181&utm_campaign=Search-DSA-NB-T2Geo-Exp&utm_campaigncode=Guide-Page+1012864&utm_term=+&gad_source=1&gclid=EAIaIQobChMIv5GfkPr1ggMVbRAGAB0xaACEEAAYASAAEgLEavD_BwE.
15. Ponicode [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ponicode.
16. JEST [Електронний ресурс] – Режим доступу до ресурсу: https://jestjs.io/uk/
17. Create React App [Електронний ресурс] – Режим доступу до ресурсу: GitHub - facebook/create-react-app: Set up a modern web app by running one command.
18. Functions [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions.
19. JavaScript reference [Електронний ресурс] – Режим доступу до ресурсу: https://devdocs.io/javascript/.
20. CSS reference [Електронний ресурс] – Режим доступу до ресурсу: https://devdocs.io/css/.

ДОДАТОК

TestCreatorPage.jsx

```
import React, {useEffect, useState} from 'react';
import './TestCreatorPage.scss';
import ProjectExplorerSidebar from '../../components/ProjectExplorerSidebar'
import CodeViewingSection from "../../components/CodeViewerSection";
import TestCreatorSection from "../../components/TestCreatorSection";
import {useDispatch, useSelector} from "react-redux";
import {fetchFunctionList, selectFunctionListError, selectFunctionListLoading} from "../../../reducers/functionListSlice";
import {fetchTestList, selectTestListError, selectTestListLoading} from "../../../reducers/testListSlice";
import {removeChangesFromUntestedFunctions} from "../../../util/api";
function TestCreatorPage({}) {
const dispatch = useDispatch();
const projectPath = useSelector(state => state.project.path);
const functionListStatus = useSelector(selectFunctionListLoading);
const functionListError = useSelector(selectFunctionListError);
const testListStatus = useSelector(selectTestListLoading);
const testListError = useSelector(selectTestListError);
const [loadingClearingSate, setLoadingClearingState] = useState('');
const [loadingState, setLoadingState] = useState('');
const [loadingMessage, setLoadingMessage] = useState('');
useEffect(() => {
setLoadingClearingState('loading');
removeChangesFromUntestedFunctions(projectPath).then(() => {
setLoadingClearingState('succeeded')
});
}, [])
useEffect(() => {
if (loadingState !== 'loading') {
if (loadingClearingSate === 'succeeded') {
setLoadingState('loading');
setLoadingMessage('Retrieving functions...');
dispatch(fetchFunctionList(projectPath));
}
}
}, [loadingClearingSate])
useEffect(() => {
if (loadingState !== 'done') {
if (functionListStatus === 'succeeded') {
setLoadingMessage('Отримання тестів...');
dispatch(fetchTestList(projectPath));

} else if (functionListStatus === 'failed') {
```

```
setLoadingState('failed');
setLoadingMessage(functionListError);
}
}
}, [functionListStatus])
useEffect(() => {
if (loadingState !== 'done') {
if (testListStatus === 'succeeded') {
setLoadingState('done');
setLoadingMessage('');
} else if (testListStatus === 'failed') {
setLoadingState('failed');
setLoadingMessage(testListError);
}
}
}, [testListStatus]);
if (loadingState === 'failed') {
return (
<ErrorScreen
errorMessage={loadingMessage}/>
);
}
if (loadingState !== 'done') {
return (
<LoadingScreen
loadingMessage={loadingMessage}/>
);
}
return (
<div className="container-fluid test-
creator-Page-wrapper">
<div className="row h-100">
<div    className="test-creator-side-
panel col-auto">
<ProjectExplorerSidebar/>
</div>
<div className="test-creation-area h-
100 col d-flex flex-column">
<CodeViewingSection/>
<TestCreatorSection/>
</div>
</div>
</div>
);
}
function
LoadingScreen({loadingMessage}) {
return (
<div className='loadingScreen'>
<h1         className='loadingScreen-
header'>Завантаження...</h1>
<span       className='loadingScreen-
content'>{loadingMessage}</span>
</div>
)
}
function  ErrorScreen({errorMessage})
{
return (
<div className='ErrorScreen'>
<h1           className='ErrorScreen-
header'>Помилка!</h1>
<span         className='errorScreen-
content'>{errorMessage}</span>
</div>
)
}export default TestCreatorPage;
ProjectSelectionPage.jsx
import React, {useEffect, useState,
useLayoutEffect} from 'react';
import {useNavigate} from "react-
router-dom";
import {useDispatch} from 'react-
redux';
import {setPath} from
"reducers/project/projectActions";
import FolderBrowser from
"Project/components/FolderBrowser";
import
```

```
'Project/pages/ProjectSelectionPage/P
rojectSelectionPage.scss';
function existingProjects() {
return new Promise((doResolve,
doReject) => {
fetch('/api/get_existing_projects').
then(res => res.json()).then(data =>
{
if (data.status === "OK") {
doResolve(data);
} else {
doReject(data);
}
});
});
}
function listFiles(currentDirectory =
"") {
return new Promise((doResolve,
doReject) => {
fetch('/api/list_files', {
method: 'POST',
headers: {
'Content-Type': 'application/json',
},
body: JSON.stringify({
"subDirectory": currentDirectory
}),
}).then(res => res.json()).then(data
=> {
if (data.status === "OK") {
console.log(data);
doResolve(data);
} else {
doReject(data);
}
});
});
}
const ToolTip =
({subDir, fileListState, curDirState,
isCurDirProjectState,
previousProjectsState,
openProjectCallback}) => {
if(isCurDirProjectState) {
return (<p>
Оберіть <button
type="button"
className="btn btn-success btn-sm"
onClick={openProjectCallback}>
Папку
</button> , щоб створювати нові
проекти або відкривати вже існуючі.
</p>)
} else {
return (<p>
Оберіть <span className="badge bg-
secondary">Папку</span> , щоб
створювати нові проекти або
відкривати вже існуючі.
</p>)
}
}
function ProjectSelectionPage({}){
const [subDir, setSubDir] =
useState("");
const [fileListState,
setFileListState] = useState([]);
const [curDirState, setCurDirState] =
useState("");
const [isCurDirProjectState,
setIsCurDirProjectState] =
useState("");
const [previousProjectsState,
setPreviousProjectsState] =
useState([]);
const navigate = useNavigate();
const dispatch = useDispatch();

const openProjectCallback = () => {
```

```
dispatch(setPath(curDirState));
navigate("/LoadingPage");
}
useEffect(() => {
existingProjects().then((previousProj
ectInfo) => {
setPreviousProjectsState(previousProj
ectInfo["existingProjects"]);
})
}, []);
useEffect(() => {
listFiles(subDir).then((fileListingIn
fo) => {
setFileListState(fileListingInfo["fil
eList"]);
setCurDirState(fileListingInfo["curDi
r"]);
setIsCurDirProjectState(fileListingIn
fo["isCurDirProject"]);
})
}, [subDir]);
return (
<>
<div className="container project-
selection-page">
<div className="row section-top">
<div className="col-6">
<h4>Попередні проекти</h4>
<div className="list-group top-item">
{previousProjectsState.map((previousP
rojectItem) =>
(
<a
href="#"
className="list-group-item list-
group-item-action"
key={previousProjectItem}
onClick={() => {
setSubDir(previousProjectItem)
}}>
{previousProjectItem}
</a>
))
}
</div>
</div>
<div className="col">
<h4>Інформація</h4>
<div className="top-item">
<ToolTip
curDirState={curDirState}
isCurDirProjectState={isCurDirProject
State}
fileListState={fileListState}
previousProjectsState={previousProjec
tsState}
subDir={subDir}
openProjectCallback={openProjectCallb
ack}/>
</div>
</div>
</div>
<div className="row section-folder-
browser">
<div className="col">
<h4>Перегляд папок</h4>
<FolderBrowser
fileListState={fileListState}
curDirState={curDirState}
isCurDirProjectState={isCurDirProject
State}
setSubDir={setSubDir}/>
</div>
</div>
<form className="container section-
form-project-confirmation">
{false && <div className="form-
group">
<input className="form-control"
readOnly value={curDirState || "/"}
```

```
/>
</div>}
<button
type="button"
className={isCurDirProjectState &&
"btn btn-success btn-lg col-12" ||
"btn btn-primary btn-lg col-12"}
onClick={openProjectCallback}
>{isCurDirProjectState && "Відкрити
прект" || "Новий проект"}</button>
</form>
</div>
</>
)
}export default ProjectSelectionPage;
```

LoadingPage.jsx

```
import React, {useEffect, useState}
from 'react';
import {useNavigate} from "react-
router-dom";
import {useSelector} from 'react-
redux';
import
'Project/pages/LoadingPage/LoadingPag
e.scss';
function newProject(pathToProject) {
return new Promise((doResolve,
doReject) => {
fetch('/api/new_project', {
method: 'POST',
headers: {
'Content-Type': 'application/json',
},
body: JSON.stringify({
"pathToProject": pathToProject
}),
}).then(res => res.json()).then(data
=> {
console.log(data);
if (data.status === "OK") {
doResolve(data);
} else {
doReject(data);
}});
});}
function LoadingPage({}){
const navigate = useNavigate();
const projectPath = useSelector(state
=> state.project.path);
const [socket, setSocket] =
useState(null);
const [socketMessage,
setSocketMessage] =
useState({"status": "IDLE"});
const [socketMessages,
setSocketMessages] = useState([]);
const [cancelProcess,
setCancelProcess] = useState(false);
const [processErrorOccurred,
setProcessErrorOccurred] =
useState(false);
const processBarSteps = (step) => {
switch (step) {
case "ANALYZE_PROCESS_FILES":
return [0, 80];
case "ANALYZE_CLEAN_DEPENDENCY":
return [80, 90];
case "ANALYZE_COUNT_DEPENDENCY":
return [90, 100];
default:
return 0; }}
useEffect(() => {
if (projectPath === "") {
navigate("/");
} else {
setSocket(new WebSocket("ws://" +
location.host +
"/sock/new_project"));
}}, []);
useEffect(() => {
```

```
if (socket !== null) {
socket.addEventListener("message", ev
=> {
setSocketMessage(JSON.parse(ev.data))
;
});
}}, [socket]);
useEffect(() => {
if (socketMessage["status"] === "OK")
{
if ("statusCode" in socketMessage) {
if (socketMessage["statusCode"] ===
"WELCOME") {
newProject(projectPath).then(() => {
console.log("Opened!");
})
} else if
(socketMessage["statusCode"] ===
"ANALYZE_COMPLETE") {
navigate("/TestCreatorPage");
}}
} else if (socketMessage["status"]
=== "ERROR") {
if ("statusCode" in socketMessage) {
if (socketMessage["statusCode"] ===
"ANALYZE_ERR_CLIENT_STOP") {
navigate("/");
} else {
setProcessErrorOccurred(true);
}}}
setSocketMessages([JSON.stringify(soc
ketMessage), ...socketMessages]);
}, [socketMessage]);
return (
<><div className="container loading-
page">
<div className="container section-
loader">
<h2>Відриття проекту</h2>
<h4>{
cancelProcess &&
"Процес скасування..." ||
socketMessage["statusCode"] ===
"ANALYZE_PROCESS_FILES" &&
"Крок 1/3: Аналіз файлів" ||
socketMessage["statusCode"] ===
"ANALYZE_CLEAN_DEPENDENCY" &&
"Крок 2/3: Очищення таблиці
залежностей" ||
socketMessage["statusCode"] ===
"ANALYZE_COUNT_DEPENDENCY" &&
"Крок 3/3: Підрахунок залежностей
функцій" ||
"..."}</h4>
<div className="row">
<div className="col-md-2">
Статус</div>
<div
className="col">{socketMessage["messa
ge"]}</div>
</div>
<div className="row">
<div className="col-md-12">
<div className="progress loader-
animation">
{!cancelProcess &&
(socketMessage["statusCode"] ===
"ANALYZE_PROCESS_FILES" ||
socketMessage["statusCode"] ===
"ANALYZE_CLEAN_DEPENDENCY" ||
socketMessage["statusCode"] ===
"ANALYZE_COUNT_DEPENDENCY") && (() =>
{
const progressPercent =
processBarSteps(socketMessage["status
Code"])[0] +
Math.trunc(
(parseInt(socketMessage["currentNumbe
r"]) /
parseInt(socketMessage["goalNumber"])
```

```
) *
(processBarSteps(socketMessage["statu
sCode"])[1] -
processBarSteps(socketMessage["status
Code"])[0])
);
return (<div
className="progress-bar progress-bar-
striped progress-bar-animated"
role="progressbar"
style={{"width" : progressPercent +
"%"}}>
{progressPercent}%
</div>);
})() || (
<div
className="progress-bar progress-bar-
striped progress-bar-animated"
role="progressbar"
style={{"width" : "0%"}}>...</div>
)}</div> </div></div>
<div className="row">
<div className="col-md-12 text-
center">
{processErrorOccurred ? (
<button
type="button"
className="btn btn-warning btn-sm"
onClick={() => {
navigate("/");}}
>Закрити</button>
) : (
<button
type="button"
className="btn btn-danger btn-sm"
onClick={() => {
socket.send(JSON.stringify({
"userAction": "ANALYZE_STOP"
}))
setCancelProcess(true);}}
disabled={cancelProcess}
>Закрити</button>
)} </div></div></div> </div>    </>
)}
export default LoadingPage;
```

test_generator.py

```
import os
from api.instances.database_main
import database_handler
from pprint import pprint
import re
def number_var_formatter(number_arg):
return f"""{number_arg}"""
def string_var_formatter(string_arg:
str) -> str:
return f"""\'{string_arg}\'"""
def boolean_var_formatter(bool_arg:
bool) -> str:
if bool_arg:
return "true"
return "false"
def object_var_formatter(object_arg:
list) -> str:
object_string = "{"
if len(object_arg) > 0:
first = True
for argument_data in object_arg:
if first:
first = False
else:
object_string += ","
object_string +=
argument_data['argument'] + ":"
argument_var_formatter =
TYPE_MATCHER_DICT[
argument_data['type']][
'var_formatter']
if isinstance(argument_var_formatter,
str):
object_string +=
```

```
argument_var_formatter
else:
object_string +=
argument_var_formatter(argument_data[
'value'])
object_string += "}"
return object_string
def array_var_formatter(array_arg:
list) -> str:
array_string = "["
if len(array_arg) > 0:
first = True
for argument_data in array_arg:
if first:
first = False
else:
array_string += ","
argument_var_formatter =
TYPE_MATCHER_DICT[
argument_data['type']][
'var_formatter']

if isinstance(argument_var_formatter,
str):
array_string +=
argument_var_formatter
else:
array_string +=
argument_var_formatter(argument_data[
'value'])
array_string += "]"
return array_string
TYPE_MATCHER_DICT = {
'array': {
'matcher': 'toEqual',
'var_formatter': array_var_formatter
},
'bigInt': {
'matcher': 'toEqual',
'var_formatter': None
},
'boolean': {
'matcher': 'toBeTruthy',
'var_formatter':
boolean_var_formatter
},
'float': {
'matcher': 'toBeCloseTo',
'var_formatter': number_var_formatter
},
'null': {
'matcher': 'toBeNull',
'var_formatter': "null"
},
'number': {
'matcher': 'toEqual',
'var_formatter': number_var_formatter
},
'object': {
'matcher': 'toEqual',
'var_formatter': object_var_formatter
},
'range': {
'matcher': 'toEqual',
},
'string': {
'matcher': 'toEqual',
'var_formatter': string_var_formatter
},
'undefined': {
'matcher': 'toBeUndefined',
'var_formatter': "undefined"
},}
UNIQUE_NUMBER = 0
def __argument_formatter(argument:
dict) -> str:
argument_var_formatter =
TYPE_MATCHER_DICT[
argument['type']]['var_formatter']
if isinstance(argument_var_formatter,
```

```
str):
arg_string = argument_var_formatter
else:
arg_string =
argument_var_formatter(argument['valu
e'])
return arg_string
def
__generate_test_file_path(test_info:
dict) -> str:
path_to_file =
os.path.split(test_info['fileId'])[0]
test_file_path =
test_info['pathToProject'] +
path_to_file + "/"
return test_file_path
def __generate_imports(test_info:
dict) -> str:
function_info_documents =
database_handler.get_function_info({
'pathToProject':
test_info['pathToProject'],
'functionId':
test_info['functionId'],
'fileId': test_info['fileId']
})
if function_info_documents is None:
raise RuntimeError(
f"function with functionId
{test_info['functionId']} doesn't" +
"exist in the database."
)
function_info =
function_info_documents[0]
match function_info['exportInfo']:
case 'export': exported_function_name
= "{" + function_info['exportName'] +
"}"
case 'export default':
exported_function_name =
function_info['exportName']
case _: raise RuntimeError(
f"{function_info['exportInfo']} is a
invalid export type")
relative_import_path = './' +
os.path.basename(
os.path.normpath(test_info['fileId'])
)
import_expression = (f"import
{exported_function_name} from "
f"\'{relative_import_path}\';")
return import_expression
def __generate_test_start(test_info:
dict) -> str:
description_string = ""
if test_info['customName'] == "":
if 'returnValue' in
test_info['moduleData']:
var_formatter = (
TYPE_MATCHER_DICT[
test_info['moduleData']['returnValue'
]['type']
]['var_formatter'])
if isinstance(var_formatter, str):
expected_value = var_formatter
else:
expected_value = var_formatter(
test_info['moduleData']['returnValue'
]['value'])
description_string += ("Function is
expected to return " +
f"{expected_value}")
else:
exception =
test_info['moduleData']['exception'][
'value']
description_string += f"Function is
expected to throw {exception}"
if 'argumentList' in
test_info['moduleData']:
```

```
description_string += ", when given
the arguments: "
first = True
for argument_data in
test_info['moduleData']['argumentList
']:
if first:
first = False
else:
description_string += ', '
argument_name =
argument_data['argument']
if (argument_data['type'] ==
'undefined' or
argument_data['type'] == 'null'):
argument_value =
argument_data['type']
else:
argument_value =
argument_data['value']

description_string +=
f"{argument_name} = {argument_value}"
else:
description_string =
test_info['customName']

return
f"""test("{description_string}", ()
=> """ + """ {"""
def __generate_test_end() -> str:
return """});"""
def __variable_assignment_standard(
variable_name: str,
variable_value: str
) -> str:
return f"""let
{variable_name}={variable_value};"""
def
__generate_variable_declarations(test
_info: dict) -> (str, dict):
if 'argumentList' not in
test_info['moduleData']:
return "", None
arguments =
test_info['moduleData']['argumentList
']
func_arg_var_map = {}
argument_declaration_string = ""
for argument_data in arguments:
global UNIQUE_NUMBER
variable_name = f"a_{UNIQUE_NUMBER}"
UNIQUE_NUMBER += 1
variable_value =
__argument_formatter(argument_data)
argument_declaration_string += f"let
{variable_name}={variable_value};"
if argument_data['argument'] ==
'...':
if '...' not in func_arg_var_map:
func_arg_var_map['...'] = []
func_arg_var_map['...'].append(variab
le_name)
else:
func_arg_var_map[argument_data['argum
ent']] = variable_name
return argument_declaration_string,
func_arg_var_map
def __generate_arguments(
arg_list: list,
expression_name: str,
func_arg_var_map: dict
) -> str:
first_arg = True
argument_string = ""
for argument_data in arg_list:
if first_arg:
first_arg = False
else:
argument_string += ','
```

```
argument_type = argument_data['type']
match argument_type:
case 'Identifier':
argument_string += (
func_arg_var_map
[argument_data['name']])
case 'Property':
argument_string += (
func_arg_var_map
[argument_data['key']['name']])
case 'ObjectPattern':
obj_string = __generate_arguments(
argument_data['properties'],
expression_name,
func_arg_var_map)
argument_string += "{" + obj_string +
"}"
case 'ArrayPattern':
array_string = __generate_arguments(
argument_data['elements'],
expression_name,
func_arg_var_map)
argument_string += "[" + array_string
+ "]"
case 'RestElement':
rest_string = ""
rest_first = True
for rest_arg in
func_arg_var_map['...']:
if rest_first:
rest_first = False
else:
rest_string += ','
rest_string += rest_arg
argument_string += rest_string
case 'AssignmentPattern':
pass
case _:
pass
return argument_string
```

```
def __generate_function_call(
test_info: dict,
func_arg_var_map: dict,/,
return_value: bool = True,
end_of_line: bool = True
) -> (str, str):
result_string = ""
func_return_variable = ""
function_info_documents =
database_handler.get_function_info({
'pathToProject':
test_info['pathToProject'],
'functionId':
test_info['functionId'],
'fileId': test_info['fileId']})
if function_info_documents is None:
raise RuntimeError(
f"function with functionId
{test_info['functionId']} doesn't" +
"exist in the database.")
function_info =
function_info_documents[0]
sub_expression_list =
test_info['functionId'].split('.')
expression_arg_list =
function_info['arguments']
function_call_string = ""
expression_list_index = 0
first_expr = True
db_query_string = ""
for expr in sub_expression_list:
expression_string = ""
if first_expr:
first_expr = False
else:
expression_string += '.'
db_query_string += '.'
db_query_string += expr
regex_result = re.search(r"^(?:\(new
)(.*)(?:\(\)\))", expr)
```

```
if regex_result is not None:
expression_name =
regex_result.group(1)
expression_string += f"(new
{expression_name}"
pprint(expression_name)
expr_arg_list =
expression_arg_list[expression_list_i
ndex]
arg_list =
expr_arg_list[expression_name]
expression_string += "("
expression_string +=
__generate_arguments(
arg_list,
expression_name,
func_arg_var_map)
expression_string += ")"
expression_list_index += 1
else:
expr_func_info =
database_handler.get_function_info({
'pathToProject':
test_info['pathToProject'],
'functionId': db_query_string,
'fileId': test_info['fileId']})
if expr_func_info is None:
expression_string += expr
else:
arg_list =
expression_arg_list[expression_list_i
ndex][expr]
expression_string += expr
expression_string += "("
expression_string +=
__generate_arguments(
arg_list,
expr,
func_arg_var_map)
expression_string += ")"
expression_list_index += 1
function_call_string +=
expression_string
if return_value is True:
func_return_variable = "r_1"
result_string = (f"let
{func_return_variable}={function_call
_string}")
else:
result_string = function_call_string
if end_of_line is True:
result_string += ";"
return result_string,
func_return_variable
def __generate_assert(
test_info: dict,
return_variable: str
) -> str:
if 'returnValue' not in
test_info['moduleData']:
raise RuntimeError('test info is
missing return value ')
expected_data =
test_info['moduleData']['returnValue'
]
match_expression =
TYPE_MATCHER_DICT[expected_data['type
']]['matcher']
if expected_data['type'] in
['undefined', 'null']:
expected_value = (
TYPE_MATCHER_DICT
[expected_data['type']]
['var_formatter'])
else:
expected_value = (
TYPE_MATCHER_DICT
[expected_data['type']]
['var_formatter']
(expected_data['value']))
```

```
assert_string =
f"expect({return_variable})"
if not expected_data['equal']:
assert_string += ".not"
assert_string +=
f".{match_expression}({expected_value
});"
return assert_string
def __generate_exception(
test_info: dict,
function_call_string: str
) -> str:
exception_string = "try {" +
function_call_string + ";} catch (e)
{"
if 'value' in
test_info['moduleData']['exception']:
exception =
test_info['moduleData']['exception'][
'value']
exception_string += "expect(e.name)"
if not
test_info['moduleData']['exception'][
'equal']:
exception_string += ".not"
exception_string += ".toBe(\"" +
exception + "\");"
if 'message' in
test_info['moduleData']['exception']:
exception_message =
test_info['moduleData']['exception'][
'message']
exception_string +=
"expect(e.message)"
if not
test_info['moduleData']['exception'][
'equal']:
exception_string += ".not"
exception_string += ".toBe(\"" +
exception_message + "\");"
exception_string += "}"
return exception_string
def generate_test(
test_info: dict
) -> (str, str):
global UNIQUE_NUMBER
UNIQUE_NUMBER = 0
import_string =
__generate_imports(test_info)
variable_declaration_string,
func_var_arg_map = \
__generate_variable_declarations(test
_info)
if 'returnValue' in
test_info['moduleData']:
function_call_string, return_var =
__generate_function_call(
test_info,
func_var_arg_map)
else:
function_call_string, return_var =
__generate_function_call(
test_info,
func_var_arg_map,
return_value=False,
end_of_line=False,)
if 'returnValue' in
test_info['moduleData']:
assert_string =
__generate_assert(test_info,
return_var)
test_string = (
__generate_test_start(test_info) +
variable_declaration_string +
function_call_string +
assert_string +
__generate_test_end())
else:
assert_string =
__generate_exception(test_info,
```

```
function_call_string)
test_string = (
__generate_test_start(test_info) +
variable_declaration_string +
assert_string +
__generate_test_end())
return test_string, import_string
def generate_tests(test_info_list:
list) -> None:
test_file_dict = {}
for test_info in test_info_list:
abs_func_path =
test_info['pathToProject'] + "/" +
test_info['fileId']
if abs_func_path not in
test_file_dict:
test_file_dict[abs_func_path] = {}
if test_info['functionId'] not in
test_file_dict[abs_func_path]:
test_file_dict[abs_func_path][test_in
fo['functionId']] = []
test_file_dict[abs_func_path][test_in
fo['functionId']].append(
test_info)
for file_path, file_test_info_dir in
test_file_dict.items():
urangutest_file = file_path +
".urang.spec.js"
file_imports = []
file_tests = []
for functionId, func_test_info_list
in file_test_info_dir.items():
func_test_strings = ""
for func_test_info in
func_test_info_list:
test_string, import_string =
generate_test(func_test_info)
func_test_strings += test_string
if import_string not in file_imports:
file_imports.append(import_string)
file_tests.append(func_test_strings)
with open(urangutest_file, 'w') as f:
for import_statement in file_imports:
f.write(import_statement)
for function_tests in file_tests:
f.write(function_tests)
```