КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня магістра
за спеціальністю 122 Комп'ютерні науки

на тему:

АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ

Виконала студентка 2 курсу магістратури
Віта МЕДОВЩУК _____
(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Олексій ФЕДОРУС _____
(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
« 15 » травня 2023 р.,
протокол № 10
Завідувач кафедри
В. М. Терещенко _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг дипломної роботи складає 54 сторінки, 6 рисунків, використано 7 джерел.

Перелік ключових слів: МАНУАЛЬНО, АВТОМАТИЗАЦІЯ, ТЕСТУВАННЯ, ТЕХНОЛОГІЇ, АНАЛІЗ, МЕТОДИ, ТЕХНОЛОГІЇ, ВЕБ-ЗАСТОСУНКИ, ТИПИ.

Об'єктом дослідження є опис процесу тестування, методи та технології тестування веб-застосунків.

Метою роботи є загальна характеристика процесу тестування, аналіз та порівняння різних методів та технологій для визначення їх позитивних та негативних аспектів.

Для досягнення поставлених цілей, були використані такі методи дослідження: практика на виробництві, інтерв’ю з експертами в області тестування веб-додатків, також аналіз літературних джерел та застосування тестування на реальних веб-додатках.

Висновки являють собою порівняння та аналіз різних методів і технік тестування веб-додатків, виявлення їх сильних і слабких сторін, а також рекомендації щодо їх використання.

Дослідження є актуальним у своїй галузі, оскільки з кожним днем з’являється все більше веб-застосунків, а отже, зростає необхідність у забезпеченні якості програмного забезпечення.

Результати дослідження можуть бути використані для покращення процесу тестування веб-застосунків та як підґрунтя для подальших досліджень в цій сфері. Результати цієї роботи можуть бути використані в будь-якій сфері, де використовуються веб-застосунки.

Пропозиції щодо розвитку об'єкта дослідження полягають у подальшому дослідженні та використанні нових методів та технологій для тестування.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ПЗ - програмне забезпечення;

ЖЦ - життєвий цикл;

IEEE - The Institute of Electrical and Electronics Engineers, інститут інженерів з електротехніки та електроніки;

WBD - Wideband Delphi, широкосмугова техніка Wideband Delphi;

WBS - Work breakdown structure, розбиття робіт на елементи;

PERT - Program Evaluation Review Technique, техніка оцінки та рецензування програм.

ВСТУП

Сучасний розвиток Інтернету призвів до значного зростання використання веб-застосунків, що призвело до збільшення вимог до якості програмного забезпечення. Тестування веб-застосунків є важливим етапом розробки цих застосунків. Тому, зростає значення вивчення та вдосконалення методів та технологій тестування веб-застосунків.

Актуальність дослідження полягає в тому, що веб-застосунки є важливими для бізнесу та розвитку суспільства, і недоліки в їх якості можуть призвести до значних витрат та втрат довіри користувачів. Також, зростаюча складність веб-застосунків створює виклики для їх тестування, тому важливо знайти найкращі підходи для ефективного тестування веб-застосунків.

Метою даного дослідження є аналіз методів та технологій тестування веб-застосунків з метою визначення найбільш ефективних та оптимальних з них. Для досягнення цієї мети поставлені наступні завдання:

- опис загального процесу тестування;
- оцінка сучасного стану методів та технологій тестування веб-застосунків;
- вивчення найбільш популярних методів та технологій тестування веб-застосунків;
- порівняння та аналіз знайдених методів та технологій тестування веб-застосунків з метою визначення їх ефективності та оптимальності;
- систематизація існуючого досвіду використання методів та технологій тестування веб-застосунків;
- розроблення рекомендацій щодо використання найбільш ефективних та оптимальних методів та технологій тестування веб-застосунків.

Описані завдання взаємопов’язані та доповнюють один одного для досягнення головної мети дослідження. В результаті цього дослідження буде

розроблено стратегію для забезпечення якості веб-застосунків, які використовуються в різних сферах, та зробити їх більш ефективними.

Об'єктом дослідження є процес тестування веб-застосунків, який є необхідною складовою процесу розробки програмного забезпечення.

Методи та засоби дослідження включають аналіз та порівняння різних методів тестування веб-застосунків, розробку автоматизованого тестування та використання спеціальних інструментів для проведення тестування, робота з фахівцями у сфері тестування веб-застосунків, проведення практичної роботи на реальних веб-застосунках.

Можливі сфери застосування досліджень полягають в покращенні процесу тестування веб-застосунків на різних етапах розробки, а також в забезпеченні високої якості веб-застосунків, що має важливе значення для підприємств та організацій, які надають веб-послуги та працюють у веб-середовищі.

Дослідження проводилось в рамках комплексних досліджень з іншими взаємопов'язаними проектами, спрямованими на розробку веб-застосунків. Результати дослідження допоможуть вдосконалити методи та технології тестування веб-додатків, що в подальшому сприятиме покращенню якості програмного забезпечення.

1 ІСТОРІЯ РОЗВИТКУ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є важливою частиною процесу розробки програмного забезпечення. Історія розвитку тестування ПЗ досить багатоаспектна і починається зі створення перших комп'ютерів в середині 20 століття.

Цікавим фактом є те, що термін "баг" набув популярності завдячуючи Грейс Гоппер у 1947 році. Вона працювала над комп'ютером Mark II Aiken Relay Calculator, і виявила, що причиною помилки в роботі комп'ютера був метелик, який потрапив до нього. Вона зберегла цього метелика як перший "баг" в історії програмування.

В 1957 році Чарльз Бейкер в своїй роботі "Flowchart Techniques for Structured Programming" розрізняв тестування і налагодження програмного забезпечення. Він вважав, що налагодження - це процес виявлення та усунення помилок в програмі, тоді як тестування - це процес перевірки, що програма виконується вірно. Це дозволило розробити більш структурований підхід до тестування програмного забезпечення.

У 1958 році була створена перша команда тестування для розробки програмного забезпечення. Цей крок став першим кроком у напрямку впровадження стандартів тестування в розробку програмного забезпечення.

У 1961 році Е.Д. Лідс та Г. С. Вайнберг опублікували книгу "Основи програмування", в якій були викладені основи тестування програмного забезпечення.

В 1970 році Вінстон Ройс запропонував модель життєвого циклу програмного забезпечення, відому як «модель водоспаду». Ця модель передбачала послідовність етапів у розробці програмного забезпечення,

включаючи визначення вимог, проектування, розробку, тестування і впровадження.

У 1979 році Гленфорд Майєрс видав книгу "Мистецтво тестування програмного забезпечення", яка стала досить важливою для розвитку тестування ПЗ. У цій книзі автор запропонував декілька методик тестування, включаючи техніку еквівалентних класів, техніку граничних значень та метод структурного тестування.

Також в книзі Майєрс зосередився на важливості взаємодії між розробниками програмного забезпечення та тестувальниками. Він відзначив, що обидва ці процеси повинні взаємодіяти, щоб забезпечити якість продукту та вчасну його випуск на ринок.

У 1983 році було опубліковано стандарт IEEE 829, який став першим стандартом, що описує процес тестування програмного забезпечення. Цей стандарт визначав структуру тестової документації та вимоги до неї, такі як план тестування, відповідність тестових сценаріїв вимогам, звіти про виконання тестів та інші.

У 1980-х роках тестування програмного забезпечення продовжувало еволюціонувати, з’явилися нові методи та технології. На початку десятиріччя було введено поняття тест-кейсу - формального опису тестової ситуації та очікуваних результатів. Одним із найважливіших досягнень 1980-х років була модель життєвого циклу програмного забезпечення V-модель, яка включала етапи відповідно до вимог, проектування, розробки, тестування та підтримки ПЗ. У 1980-х роках стали популярними інструменти автоматизації тестування, які допомагали скоротити час тестування та знизити витрати. Наприклад, в 1983 році з’явився перший інструмент автоматизації тестування - автоматизована система тестування.

В 1990-х роках тестування програмного забезпечення продовжувало розвиватися, включаючи розробку нових методів та інструментів для тестування. Основні тенденції в цей період включають наступне:

- розробка автоматизованих інструментів тестування. У 1993 році компанія Mercury Interactive випустила QuickTest Professional застосунок, що дозволив автоматизувати тестування за допомогою запису тестових скриптів та візуального програмування;
- використання методології scrum. Scrum був розроблений у 1995 році Джеффом Сазерлендом, Кеном Швабером та Майклом Бейдлом як підхід до управління проектами в галузі програмної інженерії;
- розвиток тестування безпеки: у зв'язку зі збільшенням кількості хакерських атак та витоків даних, увага була приділена тестуванню безпеки ПЗ. З'явилися спеціальні методики тестування для виявлення вразливостей ПЗ та підвищення рівня безпеки;
- використання тестування продуктивності: з'явилися інструменти для тестування продуктивності, такі як LoadRunner та JMeter, що дозволяли вимірювати продуктивність та ефективність ПЗ під навантаженням.

У 2000-х роках тестування ПЗ продовжувало розвиватися і досконалюватися. З'явилося багато нових методологій тестування, інструментів, підходів, технологій і стандартів у відповідь на нові виклики та проблеми, що з’явились у цей період. Було розроблено багато нових інструментів для автоматизації тестування. Окрім цього, з'явилися нові напрямки в тестуванні, такі як тестування мобільних додатків, тестування веб-додатків та інші.

Сьогодні тестування програмного забезпечення є важливим елементом процесу розробки ПЗ і включає в себе широкий спектр методів та підходів. Завдання тестування полягає у перевірці правильності та надійності роботи ПЗ.

2 ОСНОВНІ ПОНЯТТЯ ТЕСТУВАННЯ

Тестування програмного забезпечення - це процес визначення якості програмного продукту шляхом перевірки його відповідності вимогам та очікуванням користувачів. В рамках процесу тестування виконуються різні види тестів з метою забезпечення функціональності, стабільності та безпеки ПЗ. Деякі з основних понять, що використовуються в тестуванні, описані нижче.

Життєвий цикл програмного забезпечення - це процес від розробки до виведення програмного продукту на ринок, а також підтримки його після випуску.

Мануальне тестування - це процес, в якому людина вручну виконує тести, щоб перевірити функціональність програмного забезпечення. У мануальному тестуванні фахівець виконує дії, які зазвичай виконує кінцевий користувач, щоб перевірити, чи працює програма з точки зору коректності, швидкості та зручності використання.

Автоматизоване тестування - це процес, в якому тести виконуються автоматично з використанням програмного забезпечення. Тестувальник пише скрипти, які можуть виконувати тестові випадки, тим самим забезпечуючи швидше та більш ефективне тестування.

Тест-кейс (тестовий випадок) - це документ, що містить інформацію про те, як тестувальник має виконувати конкретний тест. Він містить опис вхідних даних, очікуваних результатів тесту, кроків для виконання тесту та іншу додаткову інформацію, необхідну для виконання тесту.

Тест-сьют (тестовий набір) - це колекція тест-кейсів, які пов'язані між собою та призначені для виконання в певному порядку. Тест-сьют може містити різні типи тестів.

Баг (дефект) - це проблема або помилка, виявлена під час тестування. Баги можуть бути різних типів, від незначних помилок, таких як орфографічні помилки, до серйозних проблем, які можуть призвести до падіння системи.

Чек-ліст (контрольний список) - це інструмент для систематичної перевірки виконання певних завдань, умов або критеріїв. Чек-ліст зазвичай включає список пунктів, що потрібно перевірити або підтвердити, та використовується для забезпечення повноти, точності та достовірності тестування.

Тест-план (тестовий план) - це документ, що описує план тестування для конкретного продукту або проекту. Тест-план містить інформацію про те, які тести будуть виконані, хто буде їх виконувати, коли тести будуть проведені, як будуть оцінюватися результати тестів, як будуть вирішуватися виявлені проблеми тощо.

Тест-дизайн - це процес планування та створення тест-кейсів та тест-сценаріїв для виконання тестування програмного забезпечення. Метою тест-дизайну є створення ефективних тестів, які покривають різні аспекти функціональності, надійності, продуктивності, безпеки та інших властивостей програмного забезпечення.

Рівень тестування - це ступінь абстракції, на якому виконуються тести. Існують наступні рівні тестування: модульне тестування, інтеграційне тестування, системне тестування, приймальне тестування.

Типи тестування визначають різні аспекти тестування програмного забезпечення та можуть бути класифіковані за різними критеріями. Основними критеріями класифікації типів тестування є ціль тестування, методика тестування та фаза розробки, на якій вони проводяться. Основні типи тестування за цілями: функціональне тестування, нефункціональне тестування, регресійне тестування.

3 ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ

Життєвий цикл тестування програмного забезпечення - це сукупність етапів тестування, які відбуваються в процесі розробки ПЗ. Цей цикл складається з наступних етапів:

Планування тестування - це перша активність, яка передує початку процесу тестування. На цій стадії визначається, які процедури тестування будуть використовуватися, які ресурси потрібні для тестування, який час потрібний для виконання тестування, які будуть вимоги до програмного забезпечення, які будуть критерії прийняття рішень під час тестування та інші аспекти.

Моніторинг та контроль тестування - це процес контролю, відстеження та збору інформації про процес тестування. На цій стадії проводиться контроль за ресурсами, які беруть участь у тестуванні, виконується моніторинг роботи тестувальної команди, відстежується виконання плану тестування та ін.

Аналіз тестування - це процес визначення того, що потрібно тестувати та які тестові умови необхідно створити на основі аналізу базису тестування. Аналіз тестування включає оцінку базису тестування, виявлення можливих дефектів, визначення властивостей та встановлення пріоритетів тестових умов для кожної властивості.

Проектування тестів - це процес розробки тестових сценаріїв та тестових наборів на основі вимог до програмного забезпечення та інших вихідних даних. Цей процес включає в себе визначення тестових цілей, вибір методів проектування тестів, створення прототипу тестових наборів.

Реалізація тестів - це активність, яка передбачає налагодження тестового середовища, а також реалізацію тест-кейсів та інших тестових скриптів. Реалізація тестів також може включати в себе виконання ручного або автоматизованого тестування, залежно від вимог проекту.

Виконання тестів - це активність, яка передбачає запуск тестів та збір результатів їх виконання. Виконання тестів може бути автоматизованим або ручним, залежно від вимог проекту та наявних ресурсів. Результати тестування можуть бути документовані та збережені для подальшого аналізу.

Завершення тестування - це активність, яка передбачає оцінку результатів тестування та підготовку звіту про тестування. У звіті про тестування мають бути описані виявлені дефекти, їх ступінь критичності та рекомендації щодо подальших кроків у відношенні цих дефектів та продукту в цілому та ін. Також у звіті можуть бути вказані результати певних метрик тестування, наприклад, покриття тестами, кількість виявлених дефектів тощо.

Даний цикл може повторюватися кілька разів у процесі розробки програмного забезпечення. Кожне повторення дозволяє виявляти та виправляти помилки та удосконалювати роботу програми. В наступних розділах розберемо детальніше кожен етап процесу тестування.

4 ПЛАНУВАННЯ ТЕСТУВАННЯ

4.1 Тест-план

Тест-план - це документ, який містить список робіт з тестування для проектів. Планування тестування залежить від вимог до проекту та стратегії тестування, використовуваних методів та життєвих циклів розробки, обсягу тестування, цілей, ризиків, обмежень, критичності та доступності ресурсів. Впродовж всієї розробки проекту стає доступною більше інформації, і в тест-план можуть бути включені додаткові дані.

Планування тестування - це неперервна діяльність, яка виконується протягом всього життєвого циклу продукту. Також життєвий цикл продукту може вийти за межі проекту та включати в себе фазу супроводження. План тестування може бути виконаний на різних рівнях деталізації, наприклад, на рівні системного тестування, приймального тестування та ін. Крім того, до списку заходів можуть включатись питання планування аналізу, проектування, реалізації та виконання тестів, оцінка результатів тестування та визначення метрик для моніторингу та контролю.

Головною метою тест-плану є забезпечення планування і організації всіх процесів тестування. Тест-план допомагає забезпечити, що тестування проводиться у відповідності до специфікацій та вимог, які були встановлені перед початком розробки, зменшує ризик помилок та дозволяє більш ефективно керувати процесом тестування. Він також дозволяє забезпечити якість продукту та його придатність до використання, а також знижує загальні витрати на тестування завдяки кращому плануванню тестування та забезпеченню правильного використання ресурсів.

Зазвичай, тест план складається з наступних розділів:

- вступ - опис проекту та його мета, зокрема з точки зору тестування;

- об'єкт тестування - список компонентів або функціональних вимог, що повинні бути протестовані;
- цілі тестування - визначаються з урахуванням вимог до програмного продукту та орієнтованих на користувача функцій. Вони допомагають визначити, що потрібно тестувати та які очікувані результати повинні бути отримані після виконання тестів;
- стратегія тестування - опис методології, яка буде використовуватися для проведення тестування;
- ресурси - опис доступних ресурсів (наприклад, технічних засобів, людських ресурсів та бюджету);
- рівні тестування - опис різних рівнів тестування, що будуть проводитися (наприклад, модульний, інтеграційний, системний, приймальний);
- типи тестування - опис різних видів тестування, що будуть проводитися (наприклад, функціональне тестування, тестування продуктивності, тестування безпеки тощо);
- критерії прийняття - опис критеріїв, за якими буде оцінюватися успішність тестування;
- ризики - опис можливих ризиків, які можуть виникнути під час тестування та стратегії їх управління;
- розклад - графік виконання робіт з урахуванням залежностей між різними етапами тестування;
- метрики - опис метрик, які будуть використовуватися для оцінки ефективності тестування;
- вимоги до середовища - опис вимог до середовища, в якому буде проводитися тестування.

В залежності від проекту деякі пункти можуть бути виключені або додані. Приклад тест-плану наведений в додатку А.

4.2 Стратегії тестування

При плануванні тестування важливим етапом також є вибір стратегії тестування. Стратегії тестування можна поділити на кілька типів, серед яких найпоширенішими є:

- аналітичний підхід - ґрунтується на аналізі певного фактора (наприклад, вимоги або ризики). До прикладу, в стратегії на основі ризиків, тести розробляються та впорядковуються за пріоритетами залежно від рівня ризику;
- тестування на основі моделей - підхід, при якому тести розробляються на основі моделі певного аспекту продукту, такого як функція, бізнес-процес, внутрішня структура або нефункціональна характеристика. Прикладами є моделі бізнес-процесів, моделі стану;
- методичний підхід - ґрунтується на систематичному використанні певного наперед визначеного набору тестів або тестових умов, таких як класифікація загальних або ймовірних типів відмов, список важливих характеристик якості або корпоративний стандарт;
- направлений підхід - визначається перш за все порадами, керівництвами або інструкціями зацікавлених сторін, експертів предметної галузі або експертів з технологій, які можуть бути поза командою тестування або організацією;
- тестування на основі мінімізації регресу - націлене на перевірку працездатності існуючих можливостей ПЗ. Ця стратегія тестування передбачає повторне використання існуючого тестового забезпечення (особливо тестових сценаріїв та тестових даних), обширну автоматизацію регресійних тестів та стандартні набори тестів;
- реактивний підхід - тестування в цьому випадку не є спланованим заздалегідь, але залежить від тестованого компонента, системи або подій,

які відбуваються під час виконання тестів. Нові тести проектуються, розробляються та виконуються на основі знань, отриманих під час тестування. Дослідницьке тестування є поширеною методикою, яку застосовують у реактивних стратегіях.

4.3 Ризики проекту та продукту

Ризики проекту та продукту - це можливі негативні наслідки, що можуть виникнути в ході розробки та експлуатації продукту, а також виконання проекту в цілому. Ризики можуть бути пов'язані з низкою факторів, таких як технології, людські ресурси, фінансові ресурси, зміни у вимогах та умовах ринку, непередбачувані обставини тощо.

У кожного проекту та продукту можуть бути свої ризики, тому планування та управління ризиками - важлива складова частина успішної реалізації проекту та розробки продукту. Необхідно відповідально підходити до ідентифікації та оцінки ризиків, а також розробляти стратегії зменшення та управління ризиками.

Ідентифікація ризиків включає огляд проекту та продукту, виявлення потенційних загроз та аналіз їх впливу на проект та продукт. Оцінка ризиків включає оцінювання імовірності виникнення ризиків та їх впливу на проект та продукт. Планування стратегій зменшення та управління ризиками передбачає розробку конкретних дій для зменшення впливу ризиків, а також відповідальності та ресурсів для їх виконання.

Незалежно від того, чи це проект розробки програмного забезпечення, будівництво будівлі або розвиток бізнесу, ризики завжди присутні. Розуміння та ефективне управління ризиками можуть допомогти зменшити вплив негативних наслідків та досягти успішної реалізації проекту чи розробки продукту.

Існує безліч ризиків, які можуть виникнути під час проектування, розробки та виконання проекту. Ось декілька загальних ризиків проекту:

- технічні проблеми: пов'язані з технічними аспектами проекту, такими як недооцінка складності програмного забезпечення, відмови обладнання;
- організаційні проблеми: пов'язані з нестачею кваліфікованих кадрів, проблемами з командоутворенням, втратою ключових співробітників тощо;
- проектні проблеми: можуть бути пов'язані з нестачею коштів для проекту, збільшенням вартості проекту внаслідок затримок, додаткових витрат на рішення проблем тощо;
- політичні проблеми: наприклад, тестувальники не можуть адекватно повідомляти про свої потреби та/або результати тестування;
- ризики залежності від сторонніх постачальників: пов'язані зі змінами у відносинах зі сторонніми компаніями, незадовільною роботою постачальників тощо.

Ці ризики можуть виникнути окремо або взаємодіяти між собою, тому важливо проводити систематичний аналіз ризиків та розробляти плани дій для мінімізації їх впливу на проект.

Звичайні приклади ризиків продукту такі:

- проблеми з безпекою даних: можливість порушення конфіденційності, цілісності та доступності даних у системі;
- некоректне відображення на різних екранах та розширеннях;
- низька продуктивність: продукт може мати проблеми зі швидкодією та продуктивністю при роботі з великим обсягом даних або при великій кількості користувачів;
- недоступність: система може бути недоступною для користувачів з певних регіонів або певних типів пристроїв;

Ризики продукту можуть призвести до погіршення якості продукту, збільшення витрат на його виробництво та експлуатацію, а також до втрати репутації компанії. Отже, управління ризиками продукту - це важлива складова частина успішної розробки та експлуатації продукту.

4.4 Методи оцінки часу тестування

Оцінювання тестування програмного забезпечення - це процес визначення часу, зусиль і ресурсів, необхідних для виконання тестування програмного продукту. Цей процес може бути складним, оскільки він включає в себе багато факторів, таких як складність програмного продукту, типи тестів, які необхідно виконати, рівень досвіду команди тестування тощо.

Нижче наведено опис декількох методів оцінки часу тестування програмного забезпечення.

Експертна оцінка: цей метод включає в себе залучення досвідчених тестувальників із команди, які можуть зробити оцінку часу, зусиль та ресурсів, необхідних для виконання тестування програмного продукту.

Серед плюсів експертної оцінки можна відзначити:

- швидкість: оцінку можна провести досить швидко, особливо якщо експерти попередньо готові до процесу та мають необхідні знання та досвід;
- доступність: зазвичай для експертної оцінки не потрібні складні математичні формули та спеціалізовані програмні засоби. Це може зробити процес оцінки доступним для широкого кола фахівців;
- висока якість: якщо вибрані експерти досвідчені та мають достатні знання в галузі, то їхні оцінки можуть бути досить точними та відображати реальну ситуацію.

Серед мінусів експертної оцінки можна виділити:

- суб'єктивність: оцінки можуть бути досить суб'єктивними, оскільки кожен експерт має свої власні думки та погляди на ситуацію. Це може призвести до різних результатів від різних експертів;
- витратність: якщо для оцінки потрібно використовувати багато експертів, то це може призвести до значних витрат на оплату їхньої праці та час, необхідний для організації процесу оцінки.

Методика WBD - це метод оцінки, який включає в себе експертну групу, яка надає свої прогнози індивідуально та обговорює їх на спільній зустрічі, щоб дійти консенсусу. Плюси методики та мінуси методики WBD ті ж самі, що в експертній методиці. Цей процес складається з декількох етапів:

- формування команди експертів: відбір людей, які мають необхідний досвід та знання, щоб відповісти на поставлені питання;
- формулювання питань: питання формулюються чітко та конкретно;
- анонімне опитування експертів: кожен експерт заповнює анкету з відповідями на поставлені питання. Цей етап проводиться анонімно, щоб уникнути впливу соціальної дезінформації та бажання підлаштувати відповіді під очікування групи;
- обробка результатів: результати опитування обробляються та узагальнюються для отримання середнього значення або медіани;
- повторне опитування: якщо результати неоднозначні, експерти знову заповнюють анкети, щоб уточнити свої відповіді;
- обговорення результатів: отримані результати обговорюються та аналізуються командою експертів.

Методика WBS (розбивка робіт на елементи) - це методика оцінки, яка розбиває проект на окремі елементи і оцінює кожен елемент окремо.

Плюси:

- за допомогою методики WBS можна створити повну картину всього проекту і його окремих елементів;

- завдяки методиці WBS можна забезпечити більш точну оцінку бюджету і кількості ресурсів, що необхідні для виконання проекту;
- WBS сприяє прозорості та взаєморозумінню між учасниками проекту.

Мінуси:

- методика WBS може займати багато часу, особливо на початковій стадії проекту, де потрібно визначити всі складові;
- недостатня увага до деталей може призвести до недооцінки ризиків та можливих проблем у процесі виконання проекту.
- важко передбачити всі можливі варіанти;
- велика кількість рівнів WBS може призвести до перевантаження менеджера проекту, що може вплинути на якість його управління.

Методика відсоткового розподілу - методика полягає у розподілі відсотків зусиль між різними етапами проекту або робочими групами, щоб визначити загальну кількість зусиль, необхідних для завершення проекту.

До переваг методики Percentage distribution можна віднести:

- простоту і швидкість використання;
- високу точність, враховуються реальні витрати на кожний етап проекту;
- можливість швидко адаптувати витрати на нові умови або зміни в проекті.

До недоліків методики можна віднести:

- відсутність детального розгляду кожного етапу проекту;
- неможливість передбачити можливі ризики та ускладнення, які можуть виникнути на окремих етапах проекту;
- високу залежність точності оцінок від досвіду та знань експертів.

Методика PERT - це методика, яка ґрунтується на трьох змінних для передбачення тривалості проекту: найбільш оптимістичний, найбільш песимістичний і найбільш ймовірний варіанти. Коли всі три змінні визначені, оцінка (E) розраховується таким чином:

$$E = (O + 4M + P) / 6$$

Переваги методики PERT:

- дозволяє оцінити тривалість проекту в умовах нестабільності його виконання та відсутності повної інформації про проект;
- забезпечує реалістичну оцінку, яка враховує ризики, що пов'язані з роботою.

Недоліки методики PERT:

- враховуються тільки 3 значення, що може бути недостатньо для деяких проектів, де є багато невизначеностей;
- методика не враховує можливість зміни умов під час розробки проекту, що може призвести до неправильних розрахунків;
- необхідно мати достатньо досвіду і знань, щоб правильно визначити O, M і P значення, що може бути проблемою для початківців в галузі.

На проекті не обов’язково використовувати тільки одну методику. Поєднання різних методик оцінки може бути ефективним підходом для забезпечення точності і достовірності оцінки. Наприклад, можна використовувати комбінацію методик PERT та WBS. WBS допомагає розбити проект на менші частини, що дає змогу краще зрозуміти обсяг робіт, необхідних для завершення проекту. Потім, використовуючи PERT, можна визначити тривалість кожної з цих частин та зіставити їх зі загальною тривалістю проекту.

Однак, слід пам'ятати, що поєднання методик може призвести до більшої складності процесу оцінки та зайняти більше часу.

5 АНАЛІЗ ТЕСТУВАННЯ

Аналіз тестування - це оцінка базису тестування з метою визначення функцій, які потрібно протестувати, та встановлення відповідних умов для проведення тестування.

Аналіз тестування складається з таких основних етапів:

а) аналіз базису тестування. Для аналізу тестування використовують базис тестування, що відповідає відповідному рівню тестування. Базисом тестування можуть бути, наприклад:

1) специфікації вимог, такі як бізнес-вимоги, функціональні вимоги, системні вимоги, історії користувачів, бізнес-потреби, сценарії використання системи або аналогічні робочі продукти, що описують функціональні та нефункціональні компоненти або поведінку системи;
2) інформація про проектування та реалізацію, така як діаграми або документи архітектури системи або програмного забезпечення, специфікації проектування, потоки виклику, діаграми моделювання (наприклад, UML або діаграми "сутність-зв'язок"), специфікації інтерфейсів або аналогічні робочі продукти, що визначають структуру компонента або системи;
3) реалізація самого компонента або системи, включаючи код, метадані та запити до бази даних, а також інтерфейси;
4) звіти аналізу ризиків, у яких можуть бути розглянуті функціональні, нефункціональні та структурні аспекти компонента або системи.

б) оцінка базису тестування. Під час оцінки базису тестування та елементів тестування виявляють дефекти різних типів, таких як:

1) неоднозначність: вказує на те, що елементи базису тестування або самі тестові сценарії можуть мати більше одного можливого варіанту інтерпретації. Це може призвести до неточного або некоректного тестування;
2) пропуски: означає, що деякі важливі аспекти функціональності системи можуть бути пропущені під час планування тестування. Це може призвести до пропуску дефектів та некоректного вимірювання покриття тестами;
3) неспівпадіння: вказує на те, що елементи базису тестування можуть не відповідати вимогам або специфікаціям. Це може призвести до некоректного вимірювання покриття тестами та пропуску дефектів;
4) неточність: означає, що елементи базису тестування можуть бути неточними або нечіткими. Це може призвести до пропуску дефектів та некоректного вимірювання покриття тестами;
5) протиріччя: вказує на те, що елементи базису тестування можуть бути взаємно суперечливими або несумісними між собою. Це може призвести до некоректного тестування та пропуску дефектів;
6) зайві твердження: означає, що елементи базису тестування або самі тестові сценарії можуть містити непотрібні, зайві або дублюючі елементи. Це може призвести до непотрібного збільшення обсягу тестування та зниження ефективності процесу;

в) встановлення властивостей тестування;

г) становлення та пріоритезація умов для кожної властивості на основі аналізу базису тестування з урахуванням функціональних, нефункціональних та структурних характеристик, інших бізнес- та технічних факторів, а також рівнів ризику.

Етап аналізу тестування або вимог є дуже важливим етапом в процесі розробки програмного забезпечення. Це забезпечує, що тестування буде

проводитись належним чином та відповідати вимогам замовника. Аналіз тестування допомагає визначити, що саме потрібно тестувати, які функції повинні бути протестовані, які умови тестування необхідно створити для перевірки відповідності функціональних, нефункціональних та структурних характеристик програмного забезпечення вимогам.

Правильно зроблений аналіз тестування допомагає уникнути помилок та недоліків в програмному забезпеченні, що може призвести до серйозних проблем у подальшому використанні продукту. Також аналіз тестування допомагає забезпечити повноту і адекватність тестів, що зменшує ризик пропуску дефектів та забезпечує високу якість продукту.

6 ПРОЕКТУВАННЯ ТЕСТІВ

Проектування тестування складається з наступних основних етапів:

- розроблення та пріоритезація тестових сценаріїв і наборів тестових сценаріїв;
- визначення необхідних тестових даних для підтримки тестових умов та тестових сценаріїв;
- проектування тестового середовища та визначення необхідної інфраструктури та інструментів.

6.1 Фактори вибору методів проектування тестів

Методи проектування тестів - це систематичний підхід до створення тест-кейсів та тест-скриптів, що дозволяє ефективно виявляти дефекти в програмному забезпеченні. Ці методи включають в себе різні стратегії тестування, які можуть бути використані для розробки тестових сценаріїв, які забезпечують відповідність програмного забезпечення вимогам та специфікаціям.

Вибір методу проектування тестів залежить від наступних факторів:

- тип системи або компонента. Різні типи систем або компонентів можуть вимагати різних підходів до проектування тестів;
- складність системи або компонента. Чим більша складність системи або компонента, тим більш детальне проектування тестів може знадобитись;
- нормативні документи. Нормативні документи можуть вимагати використання певних методів проектування тестів або накладати обмеження на використання інших методів;

- вимоги користувачів або контракту. Вимоги користувачів або контракту можуть вимагати певного рівня якості або певних видів тестування;
- типи ризиків. Різні типи ризиків можуть вимагати різних видів тестування та підходів до проектування тестів;
- задачі тестування. Залежно від задач тестування можуть вимагатись різні методи проектування тестів;
- доступна документація. Доступність документації може впливати на вибір методу проектування тестів;
- знання та досвід тестувальників. Знання та досвід тестувальників можуть впливати на вибір методу проектування тестів;
- доступні інструменти. Доступність інструментів може впливати на вибір методу проектування тестів;
- ресурси часу та бюджету. Ресурси часу та бюджету можуть впливати на вибір методу проектування тестів.

6.2 Види методів проектування тестів

Методи проектування тестів можна розділити на три основні категорії: методи білого ящика, методи чорного ящика та методи засновані на досвіді.

Методи білого ящика базуються на детальному знанні внутрішньої структури системи або компонента, який тестується. Такі методи вимагають знання мов програмування, архітектури системи, алгоритмів та інших технічних аспектів.

Методи, засновані на досвіді, використовують розуміння того, як компоненти повинні поводитися в певних умовах. Ці методи базуються на знаннях про те, які типові помилки можуть виникати в певних ситуаціях.

Методи чорного ящика, навпаки, ґрунтуються на відсутності знань про внутрішню структуру системи або компонента. Такі методи орієнтовані на

тестування зовнішньої поведінки системи або компонента. До методів чорного ящика відносять метод еквівалентних класів, метод граничних значень, метод парних значень та інші.

На практиці були використані методи чорного ящика (еквівалентного розбиття, граничних значень) та методи засновані на досвіді.

6.3 Методи чорного ящика

В даному розділі наводиться декілька методів чорного ящика. Поєднання різних методів проектування тестів може допомогти забезпечити більш ефективне і повне тестування системи або компоненту.

Прикладом може бути поєднання методів класів еквівалентності і граничних значень. Воно може допомогти покрити як базові, так і виняткові значення вхідних даних. Метод класів еквівалентності допомагає розбити діапазон вхідних даних на класи, для яких поведінка системи очікується однаковою. За своєю суттю, це зменшує кількість тестових випадків, які потрібно створити, щоб покрити всі варіанти вхідних даних.

З іншого боку, метод граничних значень зосереджується на межах між класами еквівалентності та на межах діапазону вхідних даних. Він дозволяє виявити помилки, які можуть виникнути на межі діапазону вхідних даних. Таким чином, поєднання методу класів еквівалентності та методу граничних значень може забезпечити більш повне тестування системи або компонента.

Важливо знати, що поєднання методів тестування може збільшити складність тестування і вимагати більше зусиль від команди тестування. Тому вибір методів тестування для поєднання повинен бути обґрунтованим і здійснюватися з урахуванням потреб проекту і ресурсів, які доступні для тестування. Методи чорного ящика можуть бути застосовані на різних рівнях

тестування, таких як модульне, інтеграційне, системне та приймальне тестування.

6.3.1 Метод аналізу граничних значень

Метод аналізу граничних значень є методом проектування тестів, що базується на використанні максимальних і мінімальних значень параметрів вхідних даних, які визначають межі для найбільш істотних варіантів вхідних даних. Цей метод є одним з найбільш ефективних і простих методів для виявлення помилок в програмному забезпеченні.

Метод аналізу граничних значень може бути реалізований двома способами: з використанням двох граничних значень або з використанням трьох граничних значень. У першому випадку враховуються мінімальне та максимальне допустимі значення, а в другому - мінімальне, середнє та максимальне значення.

При використанні двох граничних значень тести формуються для мінімального та максимального значень, при цьому враховуються лише можливі значення на межі діапазону, де система повинна коректно працювати. Якщо значення за межами діапазону, то вони вважаються помилковими.

При використанні трьох граничних значень тести беруться граничне значення, значення до і після межі. Цей підхід дозволяє детальніше перевірити роботу системи на межах діапазону та середині діапазону.

Метод аналізу граничних значень допомагає зменшити кількість тестових даних, які потрібно використовувати для виявлення дефектів, що робить його економічно ефективним методом. Однак, даний метод не враховує інших факторів, які можуть впливати на поведінку програми, тому рекомендується застосовувати його у поєднанні з іншими методами проектування тестів для отримання максимальної ефективності тестування.

До переваг методу аналізу граничних значень можна віднести:

- ефективність: метод дозволяє виявити багато помилок в програмному забезпеченні за мінімальний час;
- простоту і зрозумілість: метод не вимагає глибоких знань в області програмування та тестування;
- ефективність використання ресурсів: метод може бути застосований до будь-якої складності програмного забезпечення, не потребує великих витрат часу та коштів на розробку тестових сценаріїв.

До недоліків методу аналізу граничних значень можна віднести:

- нездатність виявляти складні помилки;
- вимога до детальної специфікації: метод потребує детальної специфікації для ефективного використання.

6.3.2 Метод еквівалентного розбиття

Метод еквівалентного розбиття - це метод проектування тестів, який полягає в тому, щоб розділити вхідні дані на еквівалентні класи, які повинні вести себе однаково. Для кожного класу вхідних даних потрібно створити тести, які покривають всі можливі варіанти в межах цього класу.

Переваги методу еквівалентного розбиття:

- ефективність: цей метод дозволяє зменшити кількість тестів, не втрачаючи покриття;
- ефективність в часі: зменшення кількості тестів також знижує час, потрібний на їх виконання.

Мінуси методу еквівалентного розбиття:

- обмеження методу: метод еквівалентного розбиття може бути недостатнім для складних систем, де існує багато класів еквівалентності та перехресних залежностей;

- неможливість охоплення всіх варіантів: існує можливість пропустити деякі варіанти поведінки, які не входять до жодного з класів еквівалентності, тому потрібно додатково використовувати інші методи проектування тестів.

6.3.3 Метод таблиці альтернатив

Метод таблиці альтернатив - це метод проектування тестів, який використовується для тестування системи на основі логіки прийняття рішень.

За допомогою методу таблиці альтернатив, можна побудувати таблицю, де кожний рядок описує ситуацію, а кожен стовпець - дії, які повинна виконати система відповідно до вхідних даних.

Така таблиця може включати у себе різноманітні комбінації вхідних даних та умов, що допомагає ідентифікувати можливі варіанти роботи системи і визначити, як система повинна поводитися в різних ситуаціях.

Переваги методу таблиці альтернатив:

- дозволяє виявити помилки та пропуски в специфікації вимог;
- допомагає пришвидшити процес проектування тестів та зменшити кількість тест-кейсів.

Мінуси методу таблиці альтернатив:

- складно використовувати для систем з великою кількістю варіантів поведінки;
- може вимагати багато часу на побудову таблиці, особливо, якщо у системи багато вхідних даних і варіантів їх обробки.

6.3.3 Метод таблиці переходів

Метод таблиці переходів - це метод проектування тестів, який використовується для тестування послідовних станів системи або програми. Він полягає у визначенні всіх можливих переходів між станами та проектуванні тестових сценаріїв для кожного з них.

Для використання методу таблиці переходів потрібно спочатку визначити всі можливі стани системи та переходи між ними. Потім для кожного переходу проектується тестовий сценарій, щоб перевірити, чи працює перехід правильно. Тестові сценарії можуть включати в себе вхідні дані, дії користувача та очікувані результати.

Метод таблиці переходів може бути особливо корисним для тестування програм, які мають багато можливих станів та переходів між ними. Цей метод дозволяє зменшити кількість тестів, які потрібно провести, забезпечуючи при цьому достатню покриття для всіх можливих переходів. Однак, як і з іншими методами проектування тестів, не гарантується, що всі можливі помилки будуть виявлені.

6.3.4 Попарне тестування

Попарне тестування є технікою дизайну тестів, яка дозволяє зменшити кількість тестових випробувань, зберігаючи високий рівень покриття.

Основна ідея полягає в тому, що в рамках цього методу формуються комбінації параметрів, що тестуються, таким чином, щоб кожне значення кожного параметра було протестоване разом з кожним значенням інших параметрів. Таким чином, попарне тестування дозволяє виявляти взаємодії між параметрами та зменшує кількість необхідних тестів без втрати покриття.

Одним з головних переваг попарного тестування є зменшення кількості тестових випробувань. При цьому зберігається можливість виявити більшість дефектів, пов'язаних з взаємодією вимог. Однак, цей метод не гарантує виявлення всіх можливих дефектів, тому деякі тестові випробування можуть бути пропущені. Також, дизайн таблиць покриття може бути складним для великих і складних систем.

6.4 Методи білого ящика

Метод білого ящика (White box testing) передбачає тестування програми з урахуванням її внутрішньої структури та логіки роботи.

Одним з підходів у методі білого ящика є тестування та покриття операторів. Цей підхід полягає в тому, щоб перевірити кожен оператор програмного коду та переконатись, що він працює правильно.

Тестування та покриття умов спрямоване на перевірку логічних умов у коді, а також коду, що виконується залежно від результату умови. Критерій покриття умов передбачає наявність достатнього набору тестів, які дозволяють виконати всі можливі результати кожної умови щонайменше один раз.

6.5 Методи засновані на досвіді

Методи засновані на досвіді - це методи тестування програмного забезпечення, що базуються на досвіді тестувальників, які використовують свій професійний досвід, знання і інтуїцію для виявлення потенційних дефектів в програмному забезпеченні.

6.5.1 Метод припущень про помилки

Метод припущень про помилки - це підхід до запобігання помилок, дефектів та відмов, який базується на знаннях тестувальника про потенційні помилки та дефекти, які можуть виникнути в процесі розробки програмного забезпечення.

Цей метод передбачає врахування таких факторів як: історія роботи додатку в минулому, найбільш імовірні типи дефектів, які можуть виникнути в процесі розробки, типи дефектів, які були виявлені в подібних додатках.

Спрощений підхід до методу припущень про помилки полягає в тому, що тестувальник створює список потенційних помилок, дефектів та відмов та розробляє тести, спрямовані на пошук цих помилок у програмному забезпеченні.

6.5.1 Дослідницьке тестування

Під час дослідницького тестування, тестувальник проводить низку тестів, які не плануються заздалегідь, а відбуваються в процесі тестування. Він спробує різні комбінації введення даних та взаємодії з продуктом, щоб виявити помилки та несправності, які можуть бути невідомі заздалегідь.

Цей метод зазвичай застосовується тоді, коли новий функціонал або виправлення були додані до програмного продукту, або якщо тестувальники не знайомі з деталями тестованої системи.

6.5.2 Тестування на основі чек-листів

Тестування на основі чек-листів - це метод тестування, який базується на перевірці відповідності певного компонента або системи заздалегідь складеному переліку (чек-листу) вимог, специфікацій та очікувань. Цей метод тестування зазвичай використовують для перевірки функціональності, а також

для виявлення відсутності або наявності певних дефектів. У тестуванні на основі чек-листів використовують розроблений заздалегідь план, що містить набір тестів, які мають бути виконані, а також очікувані результати для кожного тесту.

Тестування на основі чек-листів може бути корисним методом для тестування програмного забезпечення, оскільки дозволяє систематизувати тестування та забезпечити виконання необхідних тестів. Крім того, використання чек-листів дозволяє проводити тестування в більш структурованому та організованому форматі, що сприяє більш ефективному виявленню дефектів.

7 РЕАЛІЗАЦІЯ ТЕСТІВ

Реалізація тестів - це процес, що складається з наступних основних активностей:

- розробка та визначення пріоритетів процедур тестування і, можливо, створення автоматизованих сценаріїв тестування;
- створення наборів тестів (рисунок 7.1) з процедур тестування та, якщо це є, автоматизованих сценаріїв тестування;
- організація наборів тестів з розкладом виконання тестів таким чином, щоб тести виконувалися ефективно;
- побудова тестового середовища та перевірка правильності налаштування усього необхідного;
- підготовка тестових даних та правильне завантаження їх в тестове середовище.

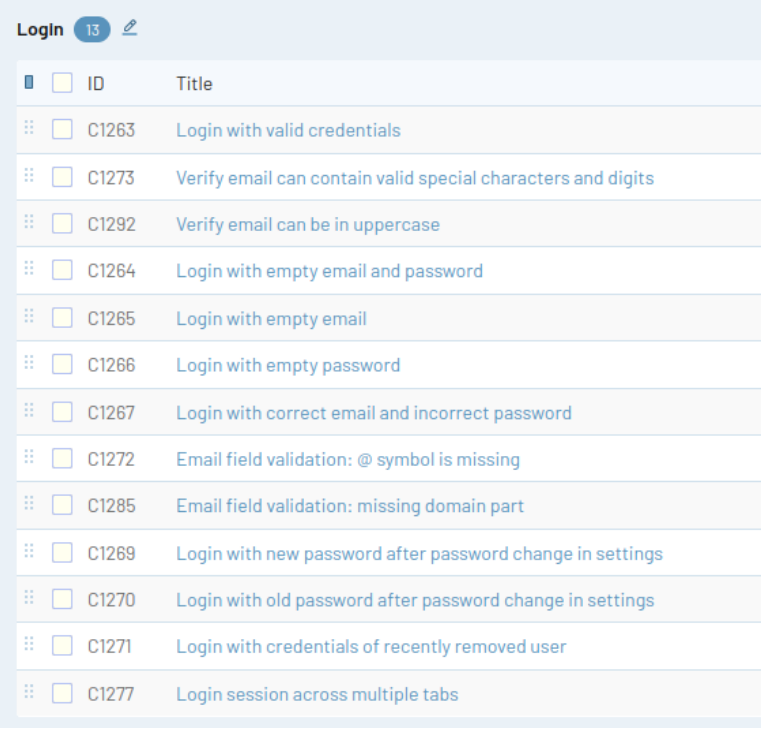


Рисунок 7.1 – Тестовий набір для частини веб-додатку (вхід користувача)

Часто завдання з проектування тестів та реалізації тестів об'єднуються в одну роботу. Аналітики тестування визначають процедури тестування та автоматизовані скрипти тестування, які можуть бути згруповані і організовані в тестові набори. Це дозволяє виконувати пов'язані тестові кейси (рисунок 7.2) разом.

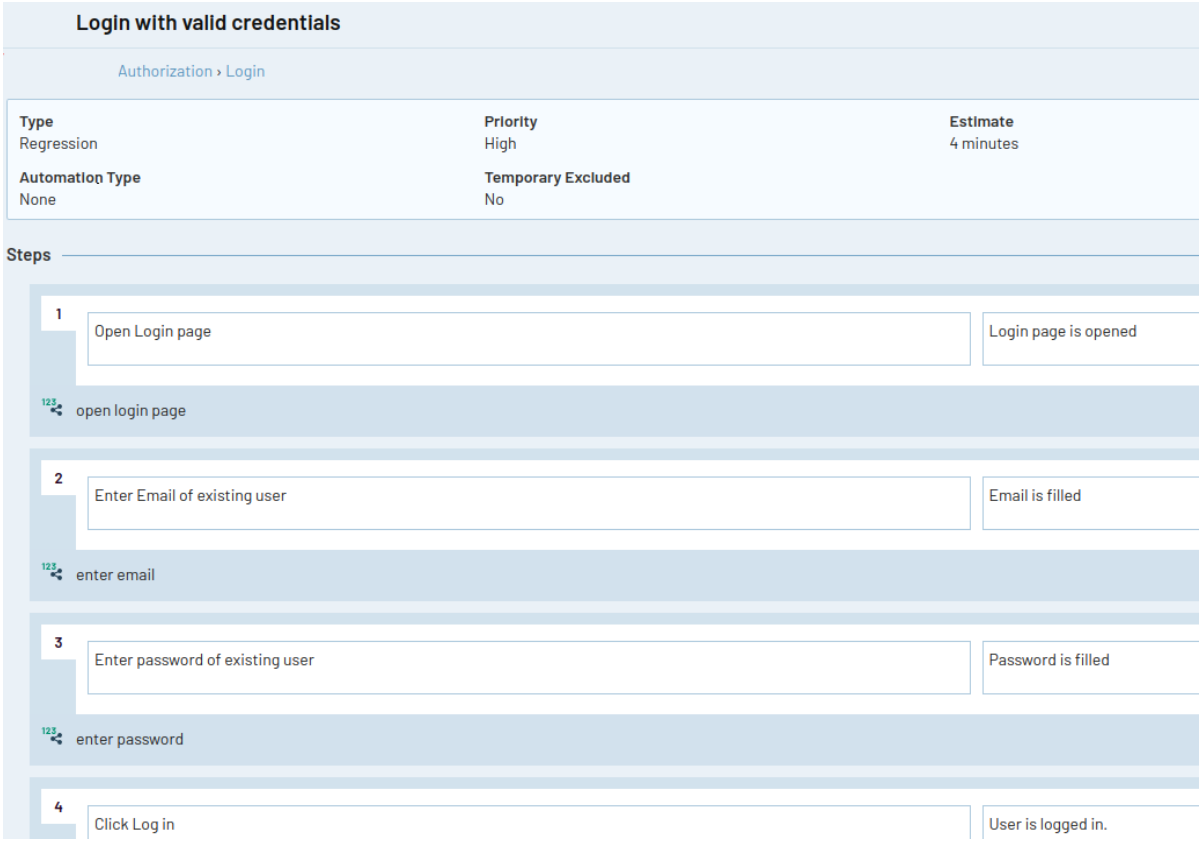


Рисунок 7.2 – Тестовий випадок (вхід користувача)

Якщо використовується стратегія тестування на основі оцінки ризиків, рівень ризику буде головною умовою при визначенні послідовності виконання тестових кейсів. Можуть бути інші фактори, такі як наявність потрібних людей, обладнання, даних та функціональності, яка повинна бути протестована.

Чудовим інструментом для створення тестових наборів є TestRail. У TestRail ви можете створювати тест кейси, визначати їх параметри та властивості, які охоплюють опис тестової ситуації, очікуваних результатів, приклади даних, передумови, очікувані наслідки, пріоритет тощо.

8 ВИКОНАННЯ ТЕСТІВ

Під час виконання тестів набори тестів запускаються згідно з розкладом виконання тестів.

Виконання тестів складається з наступних основних дій:

- запис ідентифікаторів та версій елементів тестування або об'єкта тестування, інструментів тестування та тестового забезпечення;
- виконання тестів вручну або за допомогою інструментів виконання тестів;
- порівняння фактичних та очікуваних результатів;
- аналіз відхилень для встановлення їх можливих причин;
- складання звітів про дефекти на основі спостережуваних відмов;
- протоколювання результатів виконання тестів (наприклад, пройдено, не пройдено, блокування);
- повторення тестових дій, результати яких призвели до кожного з відхилень, або в рамках запланованого тестування.

8.1 Баг-репорт

Баг-репорт - це документ, який містить опис проблеми або помилки, виявленої під час тестування програмного продукту або в процесі його використання. Баг-репорт містить інформацію про те, як і в яких умовах відбувалося виникнення помилки, а також про те, які кроки необхідно виконати, щоб відтворити проблему.

Управління баг-репортами включає в себе процеси, пов'язані з реєстрацією, відстеженням, аналізом, вирішенням та документуванням проблем, які були виявлені під час тестування програмного продукту. Для ефективного управління баг-репортами використовуються спеціальні системи

управління помилками, які дозволяють відслідковувати стан кожного баг-репорту, встановлювати пріоритети, призначати відповідальних за вирішення проблеми, моніторити час вирішення та інші аспекти.

Чудовим інструментом для управління баг-репортами є Jira. Jira - це система управління проектами, яка також може використовуватися для управління багами в програмному забезпеченні. Ця система дозволяє створювати, відстежувати та керувати різними видами задач, включаючи баги.

Jira забезпечує можливість організації процесу відстеження багів від їх виявлення до виправлення. За допомогою Jira можна легко створювати нові баги, присвоювати їм пріоритет, відстежувати стан роботи над виправленням, комунікувати з командою розробників і тестувальників, а також створювати звіти про продуктивність.

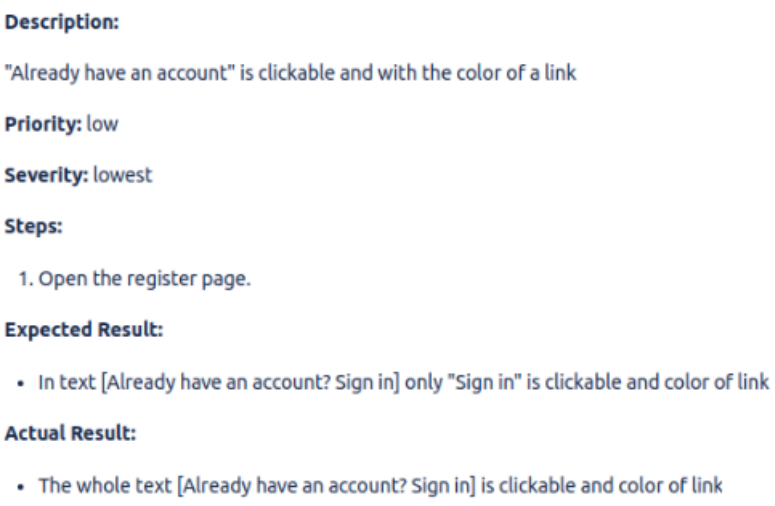
Процес управління баг-репортами може включати такі кроки:

- реєстрація баг-репортів - створення нового запису про проблему, який містить опис помилки, деталі відтворення, інформацію про конфігурацію, лог-файли тощо;
- класифікація - встановлення пріоритету та серйозності проблеми, які відображають ступінь її впливу на продукт;
- призначення відповідального - визначення відповідальної особи або команди, яка візьме на себе вирішення проблеми;
- аналіз - дослідження причин виникнення проблеми, оцінка впливу на продукт та визначення шляхів вирішення проблеми;
- вирішення - розробка та впровадження виправлення, яке вирішує проблему;
- перевірка - тестування виправлення, щоб переконатися, що проблема була вирішена;
- документування - оновлення стану баг-репорта та створення звіту про вирішення проблеми.

Баг-репорти є важливою складовою процесу тестування програмного забезпечення, тому їх правильна підготовка та документування є важливою частиною процесу розробки програмного продукту.

Звіт про дефект зазвичай містить наступну інформацію:

- ідентифікатор дефекту;
- заголовок та короткий опис виявленого дефекту;
- дату звіту про дефект, інформацію про автора звіту;
- ідентифікацію елемента тестування та середовища;
- опис дефекту, достатній для його відтворення та прийняття рішення, включаючи кроки відтворення, системні журнали, скріншоти, записи;
- очікувані та фактичні результати;
- критичність або ступінь впливу дефекту на програмне забезпечення;
- пріоритет для виправлення;
- статус дефекту (наприклад, відкрито, відкладено, дублікат, очікує виправлення, очікує перевірки, повторно відкрито, закрито);
- висновки, рекомендації та згоди;
- області, на які вплине виправлення дефекту;
- посилання, включаючи посилання на тестовий сценарій, який виявив дефект.


Description:

"Already have an account" is clickable and with the color of a link

Priority: low

Severity: lowest

Steps:

1. Open the register page.

Expected Result:

- In text [Already have an account? Sign in] only "Sign in" is clickable and color of link

Actual Result:

- The whole text [Already have an account? Sign in] is clickable and color of link

Рисунок 8.1 - Приклад баг-репорту

8.2 Критерії входу та виходу

Критерії входу та виходу є важливими елементами управління процесом тестування, оскільки вони допомагають забезпечити, що тестування відбувається відповідно до вимог та має чіткі цілі та результати. Визначення та дотримання цих критеріїв може допомогти зменшити ризики та витрати на тестування та розробку програмного продукту.

Критерії входу визначають вимоги, які повинні бути виконані перед початком тестування.

Типові критерії входу включають:

- наявність вимог, користувацьких історій та/або моделей;
- наявність елементів тестування, які задовольняють критеріям виходу для попередніх рівнів тестування;
- доступність тестового середовища;
- наявність необхідних інструментів тестування;
- наявність тестових даних та інших необхідних ресурсів.

Критерії виходу під час тестування можуть включати наступне:

- виконання запланованих тестів;
- досягнення певного рівня покриття;
- кількість відкритих дефектів нижче зазначеного порогового значення;
- низька оцінка кількості ще не виявлених дефектів;
- відповідність вимогам щодо оцінок надійності, продуктивності, практичності, безпеки та інших характеристик якості.

9 ЗАВЕРШЕННЯ ТЕСТУВАННЯ

Активності зі завершення тестування полягають у зборі даних з виконаних активностей тестування для узагальнення досвіду, тестового забезпечення та будь-якої іншої відповідної інформації. Закінчення тестування відбувається на важливих етапах проекту, наприклад, після випуску програмного забезпечення, завершення (скасування) проекту тестування, завершення рівня тестування або завершення супроводу релізу.

Завершення тестування включає наступні основні активності:

- перевірка закриття всіх звітів про дефекти або задач продукту для будь-яких дефектів, які залишаються нереалізованими в кінці виконання тестування;
- створення загального звіту про тестування для передачі зацікавленим сторонам;
- завершення та архівування тестового середовища, тестових даних, інфраструктури тестування та іншого тестового забезпечення для подальшого використання;
- передача тестового забезпечення команді супроводу, іншим проектним командам або зацікавленим сторонам, які можуть здобути користь з його використання;
- аналіз отриманих уроків для визначення змін, необхідних для майбутніх ітерацій, релізів та проектів;
- використання зібраної інформації для підвищення зрілості процесу тестування.

Звіт, підготовлений під час тестування, називається звітом про процес тестування; звіт, підготовлений після тестування, називається підсумковим звітом про тестування.

Звіт про процес тестування може включати:

- статус активностей з тестування та прогрес порівняно з планом тестування;
- фактори, що перешкоджають прогресу;
- тестування, заплановане на наступний звітний період;
- якість об'єкту тестування.

Коли критерії виходу виконані, керівник тестування складає підсумковий звіт про тестування. Цей звіт містить короткий опис проведеного тестування (на основі останнього звіту про хід тестування та іншої інформації).

Зазвичай підсумкові звіти про хід тестування включають:

- резюме проведеного тестування;
- інформацію про те, що сталося під час тестування;
- інформацію про відхилення від плану, включаючи відхилення в графіку, тривалості виконання або витрат;
- інформацію про якість тестування та якість продукту з точки зору критеріїв виходу або критеріїв завершення;
- інформацію про фактори, які блокували або продовжують блокувати тестування;
- метрики дефектів, тестових сценаріїв, покриття, прогресу тестування та використання ресурсів;
- інформацію про залишкові ризики;
- перелік тестових артефактів, які можна повторно використовувати.

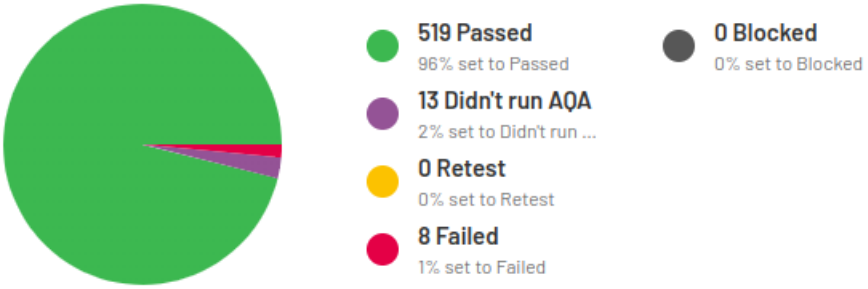


Рисунок 9.1 - Приклад результатів тестового прогону

10 АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ

Автоматизоване тестування - це процес виконання тестів з використанням спеціального програмного забезпечення, яке автоматизує виконання тест-кейсів та звітність про їх результати.

Основні переваги автоматизованого тестування включають:

- збільшення швидкості виконання тестів та скорочення часу для виявлення помилок;
- підвищення точності тестування та скорочення ризику людської помилки;
- забезпечення повторюваності тестів та зниження залежності від конкретних спеціалістів;
- збільшення тестового покриття та забезпечення більш широкого охоплення функціональності.

Етапи автоматизованого тестування дуже схожі з етапами мануального тестування, можуть відрізнятись залежно від методології тестування та інструментів, які використовуються, але загалом можна виділити наступні етапи:

- аналіз вимог: першим етапом автоматизованого тестування є аналіз вимог до тестування, щоб зрозуміти, які тести необхідно виконати, і які ресурси будуть потрібні для їх автоматизації;
- планування: на даному етапі плануються стратегії і тестові сценарії, які потрібно автоматизувати. Визначаються критерії успішності, параметри для запуску тестів та очікувані результати;
- розробка скриптів: після того, як визначені тести, які потрібно автоматизувати, розробляються скрипти для їх автоматичного виконання. Скрипти можуть бути написані вручну або згенеровані автоматично;

- налагодження: після написання скриптів проводяться тестування на відповідність очікуваним результатам. Під час цього етапу виявляються та виправляються помилки;
- виконання тестів: після налагодження скриптів вони виконуються автоматично;
- аналіз результатів: після виконання тестів аналізуються результати, які допомагають виявити помилки та проблеми;
- звітність: після аналізу результатів складається звіт, в якому фіксуються помилки та проблеми, а також результати тестування;
- підтримка: автоматизоване тестування потребує постійної підтримки та оновлення скриптів під час змін в програмному забезпеченні.

Інструменти автоматизованого тестування використовуються для створення, виконання та управління автоматизованими тестами. Деякі з найпопулярніших інструментів для автоматизованого тестування включають:

- Selenium: це безкоштовний відкритий інструмент для автоматизованого тестування веб-додатків. Він дозволяє створювати скрипти тестування на різних мовах програмування, таких як Java, Python, Ruby та інші;
- Appium: це інструмент для автоматизованого тестування мобільних додатків на платформах Android та iOS. Він підтримує різні мови програмування, такі як Java, Python, Ruby, PHP та інші;
- JMeter: це безкоштовний інструмент для тестування продуктивності веб-додатків та служб в Інтернеті. Він може емулювати велику кількість користувачів та запитів на сервер;
- Postman: це інструмент для тестування API та веб-сервісів. Він дозволяє створювати та виконувати HTTP запити до API, включаючи GET, POST, PUT та DELETE;

У своїй роботі я використовувала такий інструмент, як Playwright. Playwright - це відкритий та безкоштовний крос-браузерний інструментарій

автоматизації тестування веб-додатків, який дозволяє розробникам тестувати веб-додатки в браузері на різних платформах. Playwright підтримує різні мови програмування, такі як JavaScript, TypeScript, Python, Java та C #, і дозволяє тестувати веб-додатки в Chrome, Firefox, Safari та Edge.

Playwright має ряд інших корисних функцій, таких як можливість відлагоджувати тестові сценарії, генерувати звіти про тестування, виконувати елементарні та складні тестові сценарії, інтегруватися з іншими інструментами тестування та багато іншого. Ці функції роблять Playwright потужним та ефективним інструментом для автоматизації тестування веб-додатків. Нижче наведені рисунки з прикладами автотестів.

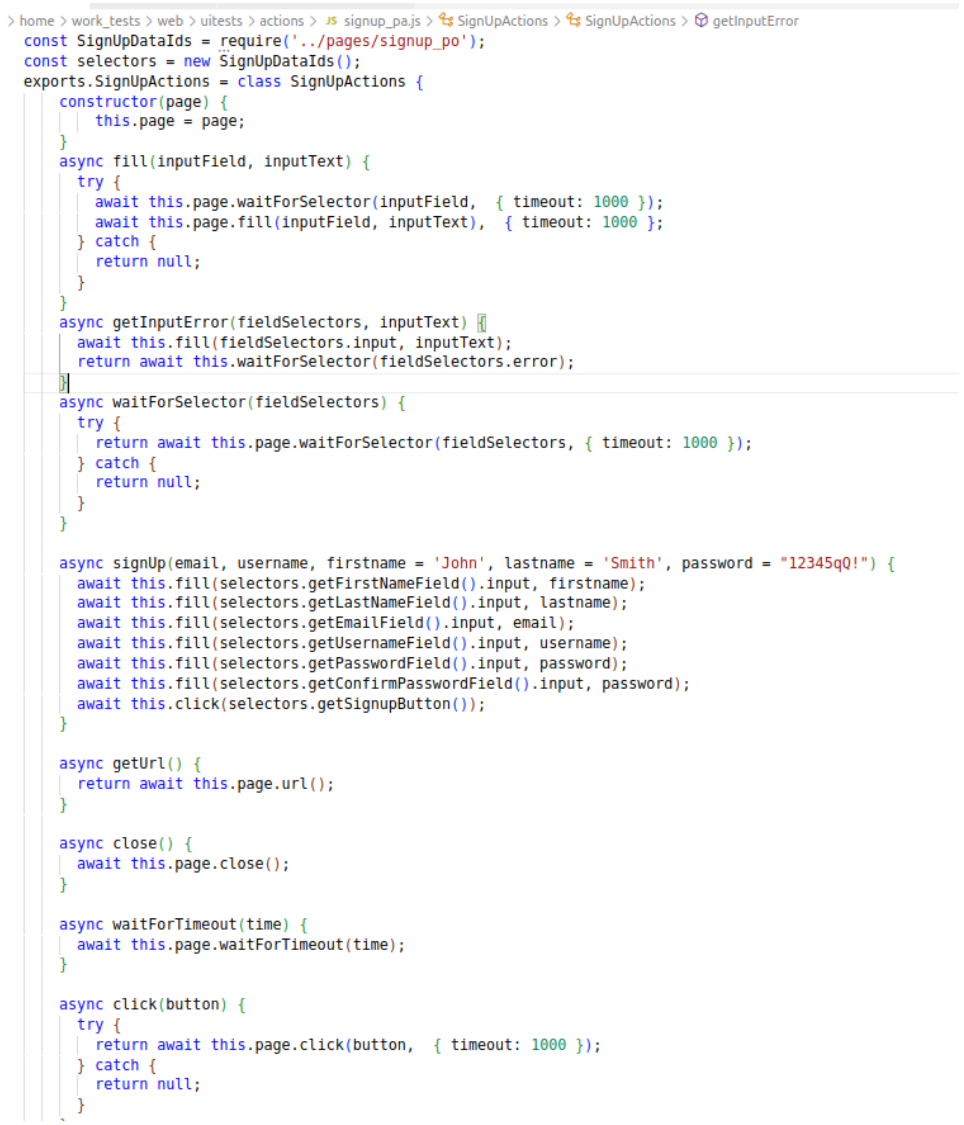


Рисунок 10.1 - Допоміжний клас для автотестів реєстрації

```
test.describe('Confirm Password field', () => {
  let confirmPasswordError, confirmPassword = selectors.getConfirmPasswordField();

  test('Password and Confirm Password Field are the same', async ({}) => {
    await signUpSteps.fill(selectors.getPasswordField().input, '12345qQ!');
    await signUpSteps.fill(confirmPassword.input, '12345qQ!');
    await signUpSteps.click(selectors.getSignupButton());
    confirmPasswordError = await signUpSteps.waitForSelector(confirmPassword.error);
    expect(confirmPasswordError).toBeNull();
  });

  test('Password and Confirm Password Field are not the same', async ({}) => {
    await signUpSteps.fill(selectors.getPasswordField().input, '12345qQ!');
    await signUpSteps.fill(confirmPassword.input, '12345qA!');
    await signUpSteps.click(selectors.getSignupButton());
    confirmPasswordError = await signUpSteps.waitForSelector(confirmPassword.error);
    expect(confirmPasswordError).not.toBeNull();
  });
});

test('Registration of user', async ({}) => {
  await signUpSteps.signUp('john@radency.com', 'john');
  emailError = await signUpSteps.waitForSelector(selectors.getEmailField());
  expect(emailError).toBeNull();
  await signUpSteps.waitForTimeout(1000);
  let pageUrl = await signUpSteps.getUrl();
  expect(pageUrl).toEqual(process.env.BASE_URL);
});

test('Already exist email', async ({page}) => {
  url.gotoSignUp(page);
  newPage = new SignUpActions(page);
  await newPage.signUp('john1@radency.com', 'john1');
  await signUpSteps.waitForTimeout(1000);
  await signUpSteps.signUp('john1@radency.com', 'john1');
  await signUpSteps.waitForTimeout(1000);
  usernameError = await signUpSteps.waitForSelector(selectors.getUsernameField().error);
  emailError = await signUpSteps.waitForSelector(selectors.getEmailField().error);
  expect(usernameError).not.toBeNull();
  expect(emailError).not.toBeNull();
});

test('Sign in button', async ({}) => {
  await signUpSteps.waitForSelector(selectors.getSigninLink())
  await signUpSteps.click(selectors.getSigninLink());
  await signUpSteps.waitForTimeout(1000);
  const pageUrl = await signUpSteps.getUrl();
  expect(pageUrl.includes(url.SignIn)).toBe(true);
});

test.afterEach(async ({}) => {
  await signUpSteps.close();
});
```

Рисунок 10.2 - Тести для сторінки реєстрації користувачів

ВИСНОВКИ

Тестування є важливим етапом розробки будь-якого програмного забезпечення. Тестування допомагає виявляти та виправляти помилки, що можуть призвести до некоректної роботи програмного забезпечення. Також воно дозволяє підтвердити коректну роботу програми в умовах, аналогічних реальному середовищу її використання. Це дозволяє зменшити ризики виникнення проблем після випуску продукту та збільшити його якість.

У роботі було розглянуто основні поняття тестування, життєвий цикл тестування та його етапи, зокрема планування, аналіз, проектування, реалізацію, виконання та завершення тестування. Також в роботі було висвітлено питання автоматизованого тестування. Описано різні методи тестування, зокрема методи чорного ящика, білого ящика та методи засновані на досвіді. Розглянуто критерії входу та виходу тестування, які допомагають визначити, коли тестування може бути завершено.

Загалом, робота демонструє, що тестування є невід'ємною складовою процесу розробки програмного забезпечення, яка допомагає забезпечити якість та надійність продукту, зменшити ризики та збільшити задоволеність користувачів. Узагальнення отриманих знань дозволяє стверджувати, що тестування програмного забезпечення є складним і багатоетапним процесом, вимогливим до уваги та ретельного аналізу.

ДЖЕРЕЛА ПОСИЛАННЯ

1. Certified Tester Foundation Level Syllabus [Електронний ресурс] / [O. Klaus, P. Tauhida, B. Rex та ін.]. – 2018. – Режим доступу до ресурсу: https://www.rstqb.org/en/downloads.html?file=files/content/rstqb/downloads/ISTQB%20Downloads/ISTQB_CTFL_Syllabus_2018_GA.pdf&cid=23567.
2. Vangie B. Regression Testing [Електронний ресурс] / Beal Vangie. – 2001. – Режим доступу до ресурсу: https://www.webopedia.com/definitions/regression-testing/.
3. Functional vs. Non-Functional Requirements: The Definitive Guide [Електронний ресурс] – Режим доступу до ресурсу: https://qracorp.com/guides_checklists/functional-vs-non-functional-requirements/.
4. History of Software Testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/history-of-software-testing/.
5. Joris M. The History of Software Testing [Електронний ресурс] / M. Joris, G. Dorothy – Режим доступу до ресурсу:https://www.testingreferences.com/testinghistory.php.
6. Anand K. Techniques to Prioritize Requirements [Електронний ресурс] / Krishnan Anand – Режим доступу до ресурсу: https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/3332/Techniques-to-Prioritize-Requirements.aspx.
7. Старух А.І. Місце тестування в процесі розробки программного забезпечення [Електронний ресурс] /Старух А.І. – Режим доступу до ресурсу:https://financial.lnu.edu.ua/wp-content/uploads/2019/09/LEKTSIYA-1.pdf.

ДОДАТОК А

Нижче наведений приклад тестового плану.

A.1 Product information

This Test strategy is the generic strategy which is applicable to the project. All features related to the components specified in functional requirements are to be tested in accordance with requirements specified in the Project Specification and acceptance criteria specified in the Jira tasks. Every new module has to be covered by testing activities to ensure product quality.

A.2 Functional Requirements

Functional requirements define the basic system behaviour. Essentially, they are what the system does or must not do, and can be thought of in terms of how the system responds to inputs. Functional requirements need to be clear, simple, and unambiguous. E.g.:

№	**Features**	**Priority***
1	Sign up with Google	High

A.3 Non - functional Requirements

- security: we are going to implement authentication using JWT tokens and refresh tokens. User’s password is hashed in DB;
- reliability and availability: failure is not acceptable;

- scalability: the highest workloads under which the system will still perform as expected is 1000 users at the same time;
- usability: application should be easy-to-use and should make documentation a joy to write;
- performance: all web pages shall load within 2 seconds;
- environmental: Google Chrome Version 101.0.4951.54 (Official Build) (64-bit);

A.4 Objectives and tasks

The task is to provide a product which is created according to the requirements and acceptance criteria.

A.5 Risks

- tight time limits that influence the testing flow;
- war problem;
- changes in the requirements.

A.6 Environment

- browsers: Chrome, Mozilla;
- screen resolution: 1920×1080 (8.89%), 1366×768 (8.44%), 360×640 (7.28%);
- language: English/Ukrainian.

A.7 How are we going to test all of it?

Test Strategy:

- analytical: this type of test strategy is based on an analysis of requirements.
- methodical: this type of test strategy relies on making systematic use of some

predefined set of tests or test conditions, such as a taxonomy of common or likely types of failures, a list of important quality characteristics.

Test Levels: Integration, System, Acceptance Testing.

Estimation Techniques: PERT, WBS, Percentage distribution, Experience-based.

Test Activities:

- requirement analysis;
- test plan creation;
- test design;
- test design update;
- test implementation;
- API testing;
- manual test execution;
- test automation;
- automation tests execution;
- defect lifecycle;
- test automation maintenance;
- regression testing;
- test completion.

A.6 Automation

Automation testing's primary objective is to reduce the manual effort for repeating testing tasks (such as Regression and Smoke Tests) and to reduce the time required for the execution of such test. Automated tests are conducted each time before the end of the sprint.

Tools:

- API Testing: Postman;
- UI Automation Testing : Cypress in Visual Studio Code (JS).

A.7 Bugs

Bug it is when the expected and actual behavior is not matching. A **Defect life cycle** is a cycle of a defect from which it goes through covering the different states in its entire life.

The ticket life cycle: creating a ticket; the ticket will go to In progress; review code; deployed in; testing; stage release; testing Failed or Done.

If a bug is found, the ticket is returned to the fix, and the life cycle is repeated.

A.9 Release readiness criteria

- permissible criticality of registered defects: non-block defects;
- recommendations which defects to fix: depending on the criticality and seriousness of the Bug first critical Bugs with high priority then Major so on. Trivial with low priority last;
- testing depth: all possible tests;
- types of testing that should be performed: Functional/Regression.

A.10 Testing Team

Vita Medovshchuk - Manual/ Automation QA Engineer

Responsibilities:

- Manual Tasks: test planning, requirements analysis, test design, implementation, execution, defects creation and verification, test result analysis;
- Automation Tasks: plus test data preparation, test users preparation, maintenance of the existent tests.