Державний торговельно-економічний університет

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Технології тестування програмного забезпечення при створенні MVC-додатка»

Студентки 4 курсу, 8 групи, Тебенькова Катерина Олексіївна

спеціальності 122 «Комп'ютерні науки»

_____
підпис студента

Науковий керівник
доцент кафедри компютерних наук та інформаційних систем,
кандидат економічних наук Шклярський Сергій Михайлович

_____
підпис керівника

Гарант освітньої програми
кандидат технічних наук, доцент Демідов Павло Георгійович

_____
підпис керівника

Київ 2022

Державний торговельно-економіцний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»

Затверджую
Зав. кафедри_____Пурський О.І.
«» 2022р.

Завдання
на випускну кваліøікаційну роботу (проект) студентці

Тебенькова Катерина Олексіївна
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи (проекту)
« Технології тестування програмного забезпечення при створенні MVC-додатка »
Затверджена наказом ректора від *«29» листопада 2021 р. № 3929*
2. Строк здачі студентом закінченої роботи
3. Цільова установка та вихідні дані до роботи
Мета роботи: дослідження технологій тестування програмного забезпечення при створенні MVC-додатка.
Об'єкт дослідження: технології тестування програмного забезпечення.
Предмет дослідження:особливості фреймворків для тестування MVC-додатків.
4. Перелік графічного матеріалу_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Шклярський С.М.		
2	Шклярський С.М.		
3	Шклярський С.М.		

6. Зміст випускної кваліфікаційної роботи(перелік питань за кожним розділом)

7. Календарний план виконання роботи

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*		
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*		
3	*Вступ*		
4	*РОЗДІЛ 1.*		
5	*РОЗДІЛ 2.*		
6	*РОЗДІЛ 3.*		
7	*Висновки*		
8	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*		
9	*Попередній захист випускної кваліфікаційної роботи*		
11	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*		
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*		
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕР*	

6. Дата видачі завдання «» 2021 р.

7. Керівник випускної кваліфікаційної роботи
Шклярський С.М.
(прізвище, ініціали, підпис)

8. Гарант освітньої програми Демідов П.Г.
(прізвище, ініціали, підпис)

9. Завдання прийняв до виконання студент-дипломник
Тебенькова К. О.
(прізвище, ініціали, підпис)

10. Відгук керівника випускної кваліфікаційної роботи

____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____

Керівник випускної кваліфікаційної роботи

____________________ 2022 р.

(підпис, дата)

11. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента ________________________

(прізвище, ініціали)

може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми______________________________Демідов П.Г.______

(підпис, прізвище, ініціали)

Завідувач кафедри_______________________________________Пурський О.І.____

(підпис, прізвище, ініціали)

«_____»__________________2022 р.

Зміст

Анотація 7
Вступ 8
Розділ 1 Аналіз предметної області 9
1.1 Тестування програмного забезпечення 9
1.1.1 Рівні тестування 9
1.1.2 Методи тестування 12
1.1.3 Види тестування 14
1.2 Веб - тестування 18
1.3 MVC-додаток 21
1.4 Висновки до розділу 22
Розділ 2 Автоматизація тестування 24
2.1 Постановка задачі 24
2.2 Автоматизоване тестування 25
2.3 Засоби для автоматизації тестування 26
2.4 Способи автоматизації 28
2.5 Висновки до розділу 29
Розділ 3 Технології тестування 31
3.1 Cucumber 31
3.2 Appium 33
3.3 API для взаємодії з мобільним додатком 34
3.4 Git 37
3.5 Написання тестів 38
3.6 Висновки до розділу 38
Висновки 40
Додаток 41
Список використаних джерел 47

Анотація

Об’єкт дослідження – технології для створення автоматизованих тестів та всебічної перевірки працездатності інформаційних ресурсів веб та мобільного базування.

Комп’ютерні засоби дозволяють: запровадити автоматизацію тестування сайтів та мобільних додатків на базах iOS та Android; перевірити правильність роботи сайтів у короткий проміжок часу; провести перевірку логіки роботи програми і правильність роботи API сервера.

В ході розробки:

- проведено аналіз методів побудови автоматизованого тестування для сайтів;
- сформульовані вимоги до комп’ютерних засобів автоматизації тестування;
- розроблено тести на мові Java,
- проведена робота з REST API та виконана реалізація.

Ключові слова: API, Java, Cucumber, Appium, ТЕСТУВАННЯ, ТЕСТИ, АВТОМАТИЗАЦІЯ.

Annotation

The object of research is the technologies for creating automated tests and comprehensive verification of the operability of web and mobile-based information resources.

Computer tools allow you to: implement automation of testing sites and mobile applications based on iOS and Android; check the correctness of the sites in a short period of time; to check the logic of the program and the correct operation of the API server.

During development:

• analysis of methods for building automated testing for sites;

• formulated requirements for computer-based test automation tools;

• developed tests in Java,

• work with REST API carried out and implemented.

Keywords: API, Java, Cucumber, Appium, TESTING, TESTS, AUTOMATION.

Вступ

Сьогодення неможливо уявити без використання програмного забезпечення. Зараз ніхто не зможе знайти хоча б одну галузь, де не використовується код. Щодня розробляється все більше і більше програм, які необхідні і в навчанні, і в медицині, і в комунікації, і багато ще де. Все це використовує чималі кошти, тому необхідно думати про зниження витрат на розробку програм та покращення якості продукту.

Перед компаніями, які розробляють програмне забезпечення, стоїть важлива задача - тестування програмного продукту перед його впровадженням. Тестування - невід'ємна складова розробки програмного забезпечення. ISO/IEC 12207 є міжнародним стандартом для процесів життєвого циклу розробки програмного продукту.

Тестування - пошук невідповідностей при порівнянні отриманого з очікуваним. Автоматизація тестування дозволяє скоротити витрати, час та ресурси компаній. Ці пункти є досить важливими для бізнесу, тому технології автоматизованого тестування набирають все більшої і більшої популярності серед компаній-розробників.

Мобільні застосунки на сучасному ринку – найбільш перспективна ідея для ведення бізнесу. Не дивлячись на те, що тестування існує вже давно, автоматизація тестування це досить нове поняття. Технології, засоби та ресурси, які використовуються у цій певній області, відносно мало вивчені і тільки виходять на ринок та впевнено завойовують прихильність та довіру компаній-розробників. Предметом дослідження були особливості технологій які застосовуються для тестування програмного забезпечення.

Розділ 1

Аналіз предметної області

1.1 Тестування програмного забезпечення

Насамперед розробка програмного забезпечення - це складний та тривалий процес, який вимагає використання різних типів та інструментів тестування програмного забезпечення. Звичайно методи тестування змінюються з часом. На даний момент ми можемо помітити стрімке збільшення попиту на тестування, яке стає більш складним. Автоматизоване тестування допомагає зекономити час, людські ресурси та скоротити цикл розробки, що призводить до підвищення швидкості розробки.

Тестування забезпечує якість різних частин програми, недолік яких міг в подальшому призвести до втрати грошей та часу.

Слід розуміти, що тестування проводиться на всіх стадіях створення проекту, а не лише в період завершення. Існують випадки, коли воно постійно оновлюється та випускається в нових версіях, щоб постійно перевіряти стабільність раніше працюючих частин.

Дехто думає, що тестування - це відсутність помилок, але в більшості випадків - це пошук якомога більшої кількості помилок задля забезпечення максимальної якості програмного продукту.

1.1.1 Рівні тестування

Однією із фундаментальних частин розробки будь-якого програмного забезпечення є поняття абстракції. В простому розумінні абстракцію можна описати як виділення головного з великої маси інформації або ще як розкладання програми на різні рівні за різними деталями. Для прикладу можна привести систему компанії. Наприклад, в компанії обов'язково є працівники в яких є свої характеристики такі як ім'я, прізвище, вік тощо. А працівники є в кожному відділі та департаменті. Завдяки абстракції нам не доведеться постійно повторювати одні й ті ж характеристики. Ми виділяємо їх як головні і потім дописуємо потрібні нам додаткові.

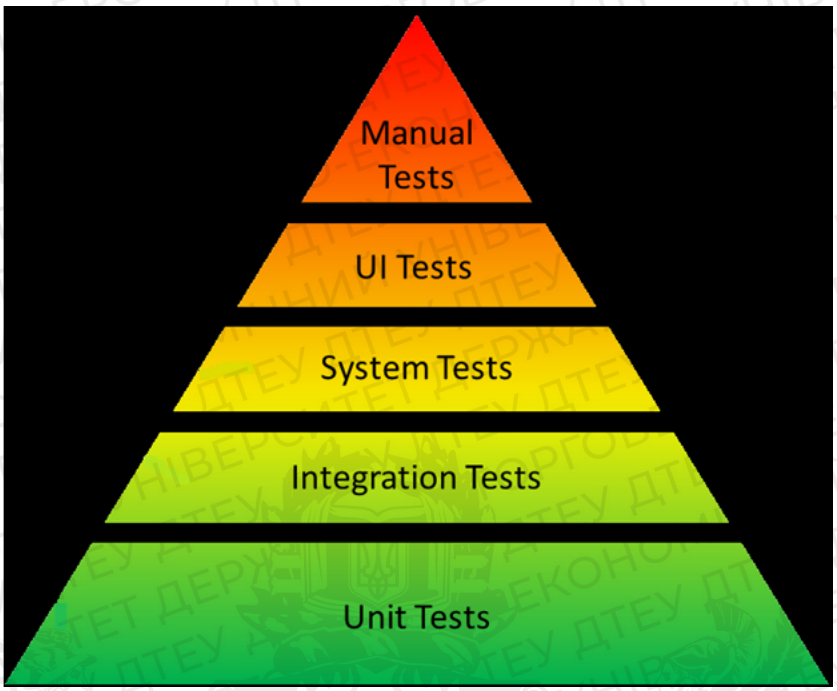

Рис. 1.1 Піраміда тестування програмного забезпечення

На рис. 1.1 наведено піраміду тестування програмного забезпечення.

«Модульне тестування (Unit testing) - це процес виконання та перевірки системного компонента з метою перевірки його функціональності. Зважаючи на ООП мінімальний тестовий випадок охоплює перевірку конструктора та деструктора. Модульні тести зазвичай пишуться розробниками під час розробки програмного забезпечення. Модульне тестування призначене для оцінки одиниць, вироблених на етапі впровадження, і є “найнижчим” рівнем тестування. У деяких випадках модульне тестування проводиться без знання програмного забезпечення, що інкапсулює. Як і при тестуванні модулів, більшість організацій, що займаються розробкою програмного забезпечення, відповідають за програмування модульного тестування.

Мета модульних тестів - перевірити реалізацію, впевнитись, що тестований модуль працює коректно. Через те що кожен модульний тест охоплює лише невелику частину програмного забезпечення, легко знайти причину тесту, який некоректно працює. З іншого боку він перевіряє коректність невеликої частини програми.

Основною частиною модульного тестування є ізоляція кожного блоку, щоб тести провалились через непов'язані зміни. Одним із способів вирішення

залежностей від інших модулів є використання одного об’єкта, який замінює інший модуль під час виконання тесту.

Самостійне написання модульних тестів не дає достатнього охоплення тестом для всієї системи, оскільки модульні тести лише гарантують, що кожен окремий тестований модуль працює належним чином. Через те що модульний тест гарантує лише те, що одиничний блок працює належним чином, помилки все ще можуть виникати в тому, як модулі працюють разом. Наприклад, функція перевірки номеру телефона нічого не варта, якщо модуль, який використовує його, передає номер телефону без кода країни як вхід. Метою інтеграційних тестів є тестування декількох окремих блоків разом, щоб побачити, чи працює більша частина програмного забезпечення належним чином.

Інтеграційне тестування(Integration tests) охоплює перевірку контракту між окремими модулями, які вже пройшли фазу модульного тестування. Ці модулі можуть бути перевірені ітеративно або всі одночасно. Найважливішою частиною цього тестування є перевірка зв'язку між модулями. Існує ймовірність того, що всі модулі успішно пройдуть модульні тести і здається, все працює правильно, але після підключення всіх модулів до однієї системи можуть виникнути серйозні дефекти, які можуть призвести до повної відмови системи. Загалом, основним завданням інтеграційного тестування є запобігання такому жахливому сценарію.

Системне тестування (system testing) перевіряє всі модулі, які пройшли інтеграційне тестування, і всю саму систему, інтегровану з будь-яким придатним обладнанням. Під відповідним обладнанням це може означати настільний комп'ютер для тестування настільних додатків або сервер для web-додатків. Тестування системи призначене для того, щоб визначити, чи відповідає зібрана система її специфікаціям. Він передбачає, що шматки працюють окремо, і запитує, чи працює система в цілому. Цей рівень тестування зазвичай шукає проблеми з дизайном та специфікацією. Це дуже дороге місце для пошуку несправностей нижчого рівня, і зазвичай це роблять не програмісти, а окрема команда тестування.

Метою функціонального тестування є перевірка функціональних вимог

програми на найвищому рівні. Тобто треба переконатись, що фунціонал, який використовується кінцевими користувачами, працює належним чином. » [27]

Можна навести приклад для показу тестування системи, використовуючи наступний сценарій:

1. Вхід до системи;
2. Зробити замовлення;
3. Зробити перевірку;
4. Відредагувати;
5. Видалення вмісту;
6. Вихід із системи.

Приймальне тестування (Acceptance testing) - тестування, яке стоїть на рівень вище, ніж системне тестування. Метою такого тестування є визначення відповідності всім умовам, які поставив замовник. Цей процес може включати оцінку результатів існуючих системних тестів, а також проведення ручного тестування та переконатися, що існують особливості для певних випадків використання. На відміну від нижчих рівнів тестування, тестування рівня прийнятності не тільки забезпечує роботу системи, а й те, що вона містить правильні функції [6].

1.1.2 Методи тестування

«Тестування програмного забезпечення – один з найбільш затратних етапів розробки, на нього виділяється від 50% до 65% загальних витрат до проекту. Існують два основні підходи для реалізації процесу тестування: функціональний ("чорний ящик") та структурний ("білий ящик"). Тому важливо розглянути характерні риси обох підходів, їх види та з'ясувати основні відмінності та особливості застосування кожного з них.

Звернемо більшу увагу до методів функціонального та структурного тестування.

При функціональному тестуванні у тестувальників немає доступу до початкового коду програми, і програма розглядається як «чорний ящик». Тож

його суть полягає в перевірці відповідності програми своїй специфікації. Найбільш оптимальними видами функціонального тестування є:

1. Випадкове (стохастичне) тестування. Створюються незалежні тести із випадково генерованих вхідних даних. Значним недоліком є велика загальна кількість тестів, які треба генерувати з огляду на надійність програми.
2. Тестування за класами еквівалентності. Множина вхідних і вихідних даних розбивається на класи еквівалентності. Поділ проходить таким чином, що кожен тест, що входить до певного класу, є еквівалентним будьякому іншому тесту цього класу, тобто програма однаково реагує на всі тести одного класу.
3. Метод аналізу граничних умов. Перевіряються випадки, що виникають безпосередньо на межах вхідних та вихідних даних. Так як більшість помилок з'являється не в центрі множини допустимих значень, а на верхній та нижній границях, відповідно основна увага зосереджена саме на них.

Структурне тестування полягає в перевірці та аналізі вихідного коду програми. Основними різновидами структурного тестування є: тестування маршрутів, тестування обробки даних та тестування циклів.

1. Тестування маршрутів. Перевірка програмного коду виконується через проходження певного зазначеного шляху. На практиці, під час тестування, часто деякі маршрути залишаються неперевіреними. Складність в тестуванні полягає в тому, що всі маршрути мають бути перевірені ще під час створення програми, що вимагає великих часових затрат.
2. Тестування обробки даних. Процес виконання програми можна розуміти як роботу з певними даними, що передаються із вхідного потоку у вихідний. Із вхідних даних формуються проміжні результати, що аналізуються до моменту їх виведення. Помилки виявляються як перекручування проміжних результатів або, взагалі, їхня відсутність.
3. Тестування циклів. Цикли в програмі можуть значно ускладнити весь процес тестування та відлагодження. Повне тестування повинно включати в себе перевірку всіх можливих маршрутів в кожній ітерації циклу та всіх можливих сполучень циклів із ациклічною частиною маршруту. » [27]

1.1.3 Види тестування

Вид тестування - це засіб для чіткого визначення мети потрібного рівня для програми. Він сфокусований на чітку мету тестування, що виконується певним компонентом або загалом системою. Під метою можна розуміти перевірку надійності, зручності використання, структури, повністю системи, різних елементів, дефектів та різних випадкових змін.

Можна говорити про 4 види тестування програмного забезпечення:

- функціональне тестування (Functional testing);
- нефункціональне тестування (Non-functional testing);
- структурне тестування (Structural testing);
- тестування змін (Change related testing).

Функціональне тестування спрямоване на тестування функцій системи, щоб підтвердити відповідність функції до документації. Воно може бути застосоване згідно зі специфікацією, а також згідно зі знаннями системи.

Переваги функціонального тестування:

- в рамках тестування ми «копіюємо» безпосереднє використання системи;
- тестування, як правило, проводиться в умовах близьких до реальних.

Недоліки:

- існує ймовірність пропустити кілька помилок логіки програмного забезпечення під час перевірки функціоналу програми.

Якщо в рамках функціонального тестування ми відповідаємо на питання «Чи працює система?», то нефункціональне відповідає на питання: «Як добре працює система?». Воно спрямоване на перевірку описаних аспектів, які не відносяться до функцій програмного продукту.

Нефункціональне тестування складається з підвидів:

1. ***Тестування стабільності – Stability/Reliability testing*** – виявлення крешів системи під час використання.
2. ***Юзабіліті тестування – Usability testing*** – дослідження для визначення зручності використання ПЗ.
3. ***Тестування ефективності – Efficiency testing*** – перевірка необхідних обсягів коду і ресурсів QA, що використовуються програмою для виконання окремої функції.
4. ***Тестування ремонтопридатності – Maintainability testing*** – цей підвид нефункціонального тестування визначає наскільки легко підтримувати працездатність системи.
5. ***Перевірка портативності – Portability testing*** – тестування доступності перенесення окремого компонента або всього програмного забезпечення з одного оточення на інше (Windows 8.1 -> Windows 10, Windows -> MacOS).
6. ***Тестування «пра-витоків» – Baseline testing*** – перевірка документації та специфікації, за якою будуть написані тест-кейси. До цього підвиду тестування можна віднести й тестування вимог.
7. ***Приймальне тестування – Compliance/Acceptance testing*** – перевірка продукту на відповідність критеріям готовності.
8. ***Тестування документації – Documentation testing*** – перевірка всієї створеної в рамках тестування документації (від майстер тест-плану до тест-кейсів).
9. ***Тестування витривалості системи – Endurance testing*** – тестування системи при високому навантаженні протягом тривалого періоду часу з метою вивчення її поведінки.
10. ***Тестування навантаження – Load testing*** – як правило, проводиться з метою визначення поведінки ПЗ під очікуваним рівнем навантаження.
11. ***Тестування продуктивності – Performance testing*** – перевірка швидкості роботи ПЗ або його окремих функцій.
12. ***Тестування сумісності – Compatibility testing*** – тестування системи під час роботи в різних середовищах: «залізо», програмна частина тощо.

13. ***Тестування безпеки*** – ***Security testing*** – проводиться для відповіді на питання «Чи є додаток безпечним/захищеним чи ні?».
14. ***Об'ємне тестування*** – ***Volume testing*** – тестування ПЗ з використанням баз даних певного розміру.
15. ***Стрес тестування*** – ***Stress testing*** – це тестування в обмежених умовах, наприклад, перевірка поведінки системи (відсутність крешів) за умов нестачі ресурсів комп'ютера (оперативної пам'яті або місця на HDD/SSD дисках).
16. ***Тестування швидкості відновлення*** – ***Recovery testing*** – проводиться з метою визначення швидкості відновлення системи у разі софтверного креша (падіння програмного забезпечення) або помилки «заліза».

Тестування локалізації, інтернаціоналізація – ***Localization testing*** – перевірка ПЗ на відповідність мовних, культурних та/або релігійних норм. Локалізація – перевірка відображення усіх перекладених текстів програмного забезпечення.

Структурне тестування спрямоване на тестування структури системи або компонента, щоб побачити що відбувається всередині системи.

Методи структурного тестування:

1. строкове покриття (Statement Coverage)
2. покриття шляху (Path Coverage)
3. покриття рішення (Branch Coverage)
4. покриття умови (Condition Coverage)

Переваги:

- можливість виявити та видалити «зайвий» код;
- можливість виявлення потенційних помилок на ранній стадії;
- забезпечує більш ретельне тестування ПЗ;
- не потребує високих витрат людино-годин.

Недоліки:

- вимагає знання коду та інструментів тестування.

При тестуванні змін в системі дуже важливо зрозуміти різницю та межу між поняттями регресійне тестування (Regression testing) та повторне тестування (Retesting).

Регресійне тестування (Regression testing) проводиться з метою перевірки працездатності функціоналу, що існує, та перевірки на відсутність сторонніх помилок після оновлення білда (внесення правок або доповнень в систему).

Повторне тестування (Retesting) – проводиться для підтвердження виправлення помилки та роботи даного функціоналу.

Регресійне тестування (Regression testing)

Переваги:

- підтверджує відсутність багів після додавання фічі або правки коду;
- може бути виконано з використанням інструментів автоматизації;
- допомагає поліпшити якість продукту.

Недоліки:

- даний вид тестування може бути дуже трудомістким.

Повторне тестування (Retesting)

Переваги:

- підтверджує виправлення помилки й коректну роботу функціонала;
- підвищує загальну якість продукту;
- вимагає менше часу на верифікацію;
- не вимагає яких-небудь нових налаштувань середовища тестування.

Недоліки:

- тест-кейси для повторного тестування можуть бути виявлені тільки після першого раунду тестування;
- тест-кейси для повторного тестування не можуть бути автоматизовані;
- вимагає додаткового часу для проходження вже пройдених раніше тест-кейсів.[20]

1.2 Веб - тестування

Веб-тестування - тестування програмного забезпечення для виявлення можливих помилок під час тестування сайтів чи веб-додатків. Додатки мають бути повністю протестовані перед публікацією. Перевірку слід здійснити від початку до кінця, перш ніж програмний продукт потрапить до остаточного користувача. Стосовно веб-сайтів слід обов'язково перевіряти часовий пояс користувачів а також дизайн та функціональність інтерфейсу.

План для тестування веб-додатків складається з шести розділів:

- функціональне тестування
- інтеграційне тестування
- тестування безпеки
- тестування локалізації та глобалізації
- тестування зручності використання
- крос-платформне тестування

Функціональне тестування

В цьому пункті необхідно впевнитися, що програмний продукт відповідає функціональній специфікації, наведеній в документації з розробки.

Перш за все перевіряється тестування форм, такі як реєстрація, авторизація, валідація всіх полів, форми зворотнього зв'язку та посилання на користувацькі угоди.

Реєстрація:

1. Користувач із даними існує в системі.
2. Користувач із даними не існує в системі.

3. Користувач, заблокований у системі, не може пройти повторну реєстрацію.

Авторизація:

1. Користувач існує в системі з введеним логіном та паролем.
2. Користувач не існує у системі.
3. Користувач існує в системі, але пароль неправильний.
4. Користувач із введеним логіном та паролем існує в системі, але заблокований модерацією (сторінка заморожена).
5. Валідація полів введення.

Тестування валідації полів:

1. Максимальна та мінімальна довжина.
2. Діапазон допустимих символів спецсимволи.
3. Обов'язковість заповнення.
4. Переконайтеся, що астеріск (знак зірочки) відображається у всіх обов'язкових полях.
5. Переконайтеся, що система не відображає вікно помилки при незаповнених необов'язкових полях.

Наступним пунктом тестування є пошук. Слід перевірити результати існують або не існують, коректне повідомлення про порожній результат, порожній пошуковий запит та пошук по емодзі.

Далі йде тестування поля. Перевіряються числові поля: вони не повинні приймати літери, дробові значення, негативні значення у числових полях, якщо вони дозволені, ділення на нуль, максимальна довжина кожного поля, поля зі спецсимволами, відсутність тексту який виїжджає за межі поля.

Після цього тестування спливаючих повідомлень. Перевіряються спливаючі повідомлення, відображення повідомлення про оновлення та видалення та повідомлення про помилки введення.

Далі йде тестування функціональності доступних кнопок, перевірка обробки різних помилок(сторінку не знайдено, помилку сервера, тайм-аут тощо), документи, які були завантажені, відкриваються коректно, користувач сам може

завантажувати чи прикріпляти медіа і обов'язково можливість видалення, переконатися, що файли відправляються на сервер тільки після натискання на певну кнопку, не забути про кеш, файли cookie, сесії(користувач очистив кеш браузера, видалив файли cookie знаходячись на сайті та після того як покинув сайт) та DevTools(помилки в консолі, зображення завантажуються).

Інтеграційне тестування

Інтеграційне тестування використовується для того, щоб впевнитися, що додаток сумісний зі сторонніми сервісами. Треба перевірити роботу сторонніх модулів(оплата, карти), рекламу та переходи по сторінкам, показ елементів та кліки.

Тестування безпеки

Перевірка має знаходити недоліки та проблеми з точки зору безпеки програми. Користувач не може авторизуватись: під старим паролем, заблокований у сервісі, досяг ліміту авторизацій, ввів чужий код верифікації. Треба перевірити сторінки, що містять конфіденційні дані(пароль, дані банківської картки, секретні питання)відкриваються через HTTPS, пароль прихований астерисками(знак зірочки) на сторінках, відображення повідомлення про помилку, завершення сесії, SQL-ін'єкції, HTML-ін'єкції, зашифрованість файлів cookie та доступ до контенту.

Тестування локалізації та глобалізації

Тестування глобалізації має на увазі тестування веб-додатка для різних положень, форматів дат, чисел та валют. Тестування локалізації включає тестування WEB додатка з локалізованими рядками, зображеннями та робочими процесами для певного регіону. Треба перевіряти відображення дати та часу відповідно до часового поясу, зміну мови, номери телефону з різними кодами країн, розташування користувача, коректне відображення валютних символів.

Тестування зручності використання

Тестування зручності використання перевіряє певну інформацію для користувача. А саме відсутність орфографічних та граматичних помилок, вирівнювання картинок, інформативні помилки, відступи, стандартний розмір кнопок, відсутність битих посилань, переходи та навігація між сторінками та

розділами меню, скролл з'являється лише тоді, коли він потрібний тощо.

Крос-платформне тестування

Крос-платформне тестування використовується, щоб з'ясувати чи сумісний додаток з іншими браузерами та апаратним забезпеченням. Треба робити перевірку в різних браузерах(Firefox, Chrome, Safari та інші): анімація, верстка, шрифти тощо, брати різні версії ОС: Windows, Mac, Linux та використовувати перегляд мобільних пристроїв. [21]

1.3 MVC-додаток

Моде́ль–вигляд–контро́лер — архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних та їх структуру. Вигляд (View) відповідальний за представлення цих даних користувачеві, тобто інтерфейс програми. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель.

- Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.
  - Активна модель - вигляд відстежує зміни в моделі та реагує на них.
  - Пасивна модель - вигляд оновлюється через контролер.
- Вигляд може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.
- Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду. [22]

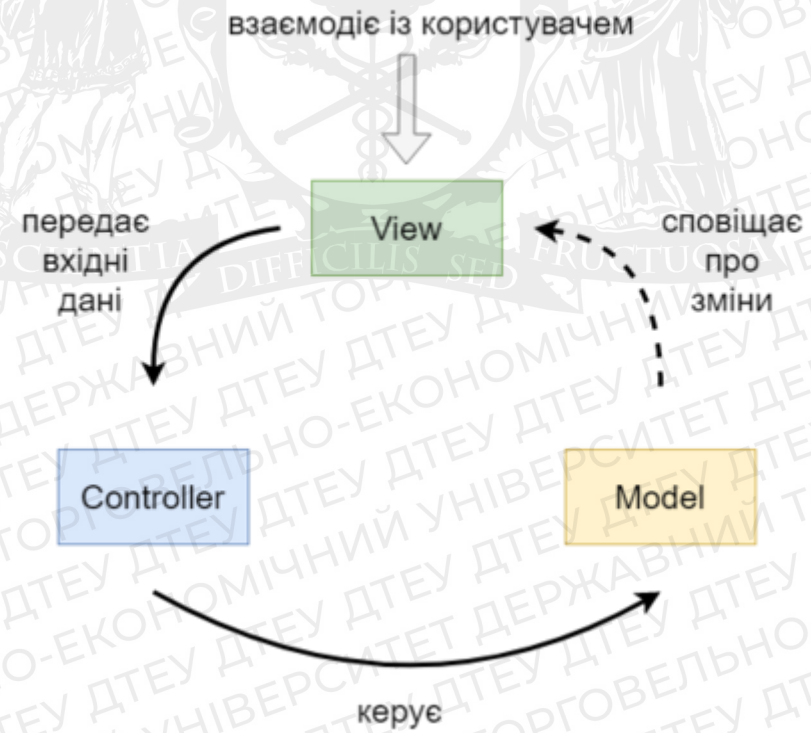

Рис 2. Діаграма взаємодії між компонентами шаблону MVC

1.4 Висновки до розділу

Тестування використовується для забезпечення якості різних частин

програмного проекту. Тестування використовується на будь-яких етапах в тому числі і у випадках коли оновлюється програмне забезпечення. Воно іноді сприймається як демонстрація відсутності помилок, не зважаючи на те що в більшості випадків це підвищення якості програмного продукту.

Оскільки число можливих тестів навіть для нескладних програмних компонентів практично нескінченне, важливо щоб стратегія тестування була в тому, аби провести всі можливі тести з урахуванням наявного часу та ресурсів. В результаті програмне забезпечення слід тестувати стандартним виконанням програми, щоб виявити баги (помилки або інші дефекти).

Якість не є абсолютною, це суб'єктивне поняття. Через це тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність програмного забезпечення. Воно лише порівнює стан і поведінку продукту зі специфікацією. Також потрібно розрізняти тестування програмного забезпечення й забезпечення якості програмного забезпечення, до якого належать всі складові ділового процесу, а не тільки тестування.

Розділ 2

Автоматизація тестування

2.1 Постановка задачі

Автоматизоване тестування програмного забезпечення – частина процесу тестування на етапі контролю якості в процесі розробки програмного забезпечення. Воно використовує програмні засоби для виконання тестів і перевірки результатів виконання, що допомагає скоротити час тестування і спростити його процес. Концепція автоматизованого тестування може мати різні визначення залежно від джерела та контексту. Це може стосуватися автоматичної генерації фактичних тестових кейсів або автоматичного запуску тестових кейсів, розроблених вручну. В даному випадку це тестування на основі розроблених вручну тестових кейсів, які можна повторно запускати сценарієм.

Можна назвати багато причин для використання саме автоматизованого тестування. Через те, що тестові кейси повторюються комп'ютер може виконувати їх на високій швидкості завдяки чому зменшується час та вартість тестування. Використовуючи автоматизоване тестування можна повністю виключити помилки спричинені людським фактором. Серед переваг слід ще зазначити збільшення охоплення тестами та заміщення трудомістких завдань.

Тести можна назвати автоматизованими, якщо при запуску немає прямої взаємодії з боку тестувальника. Автоматизовані тести повинні відповідати таким умовам:

- можливість виконання набору або підмножини тестових кейсів;
- немає необхідності втручатися після початку випробувань;
- важливі параметри тесту встановлюються автоматично;
- результати тесту реєструються автоматично
- можливість порівняння фактичних результатів випробувань із очікуваними– результатами;

Зараз системи автоматизації стають ще більш складними та

вдосконаленими. Побудова гнучкої, розподіленої та сумісної системи автоматизації вимагає системного підходу до розробки програмного забезпечення. В проектах беруть участь люди, які не мають досить великого досвіду в програмній інженерії, тому деякі аспекти можуть втратитись. Тому необхідно додати нові надійні процеси для розробки програмного забезпечення. Через це можна сказати, що методи розробки та тестування вимагають певного контролю.

Перш за все треба пояснити процес, використаний у проекті, потім обговорити сучасні методи та засоби, що використовуються для розробки програмного забезпечення та його тестування. Потрібно розглянути якомога більше способів та методів для автоматизованого тестування.

2.2 Автоматизоване тестування

Автоматизація програмного забезпечення передбачає співпрацю людей, які беруть учать у розробці програмного продукту та його тестуванні, погодженні процесів з командою. Життєвий цикл розробки програмного продукту можна поділити на чотири основні фази:

1. Вимоги та дизайн;
2. Розробка;
3. Тестування;
4. Розгортання та введення в експлуатацію.

На сьогодні доступно багато середовищ прикладного програмування; до них належать як безкоштовні, так і власні інструменти. Часто компанії з автоматизації використовують власне середовище програмування, яке можуть просувати на міжнародний ринок.

Apache - це відкритий вебсервер Інтернет для UNIX-подібних, Microsoft Windows, Novell NetWare та інших операційних систем. Apache Subversion необхідний для системи контролю версій для програми і використовується під час розробки програмного продукту.[25]

Robot Framework – це загальна система автоматизації тестів для

приймального тестування, яка може використовуватися наскрізним тестуванням веб-програми. Robot Framework – це фреймворк, керований ключовими словами, який можна розширити за допомогою зовнішніх бібліотек, що містять ключові слова для певних типів тестування.

Jenkins – провідний сервер автоматизації з відкритим кодом, який можна використовувати для побудови та автоматизації будь-якого проекту. Jenkins може обробляти всі завдання від компіляції вихідного коду до запуску наскрізної автоматизації тестування. Візуалізація 35 результатів тесту буде відповідати основним принципам, які надаватиме Robot Framework.

Створення програми проходить багато рівнів випробувань перед введенням в експлуатацію. Сюди входять:

- Модульне тестування;
- Інтеграційне тестування;
- Приймальне тестування.

Модульне тестування - це тестування, коли розробник пише тестові кейси для перевірки функціональності окремих програм управління. В IT перевіряється взаємодія між програмами управління. Тестові кейси пишуться вручну. Розробники пишуть модульні тести на основі власного розуміння логіки програм.

2.3 Засоби для автоматизації тестування

Щоб реалізувати автоматизації тестування при розробці програмного продукту, слід врахувати, що автоматизація потребує інструменти для її реалізації. В якості інструментів можна використовувати комерційні, безкоштовні, або ж власно розроблені продукти для тестування. У кожного з них є свої плюси та мінуси.

Серед плюсів комерційних продуктів можна зазначити:

- наявність готових автотестів, в наслідку економія часу;
- готові інструменти, що дають можливість проводити відладку самостійно;
- можливість підтримки розробників, які створили програмний продукт.

Серед мінусів комерційних продуктів можна зазначити:

- ціна;
- немає доступу до коду програми;
- суворе наслідування моделі.

Серед плюсів безкоштовних продуктів можна зазначити:

- невелика ціна або її відсутність;
- доступ до вихідного коду програми.

Серед мінусів безкоштовних продуктів можна зазначити:

- Недостатня функціональність;
- Ризик відмови у підтримці програмного інструменту.

Серед плюсів власних продуктів можна зазначити:

- розробка інструмента як результат створення застосунку;
- доступ до вихідного коду;
- гнучкість у розробці утиліти;
- можливість робити зміни в процесі розробки та після неї.

Серед мінусів власних продуктів можна зазначити:

- відсутність гарантій;
- ризики стосовно якості інструменту.

Найбільш відомими комерційними інструментами для автоматизації тестування користувацького інтерфейсу мобільних застосунків є TestComplete(https://smartbear.com/product/testcomplete/overview/), Ranorex(https://www.ranorex.com/), SeeTest Automation(https://docs.experitest.com/display/TDB), Squish(https://www.froglogic.com/squish/).
Стосовно безкоштовних інструментів: Appuim(https://appium.io/), Selenium(https://www.selenium.dev/), Selendroid(http://selendroid.io/), Robotium(https://www.methodsandtools.com/tools/robotium.php),

Calabash(https://smartbear.com/blog/calabash-tutorial-for-mobile-app-testing/).
Для вибору засобів необхідно враховувати такі критерії:

- підтримка мобільних платформ;
- підтримка різних типів застосунків (веб, нативні, гібрид);
- підтримка запуску тестів на емуляторах, реальних пристроях;
- підтримка мов програмування, зручність, інструменти налагодження;
- можливість паралельного запуску тестів;
- створення звітів.

При виборі інструмента краще орієнтуватися на документацію застосунку, поставлені задачі, кваліфікацію програмістів, ризики та фінанси.

2.4 Способи автоматизації

Перш за все слід сказати про різницю між ручним та автоматизованим тестуванням.

Ручне тестування - це процес пошуку недоліків програми, коли спеціаліст з якості відповідає за всі аспекти роботи, будучи потенційним користувачем. Як правило, для вимірювання точності він використовує заздалегідь сплановані плани, які показують результати роботи найголовніших аспектів програми при певних операціях.

Автоматизоване тестування - це тестування, які використовують програмне забезпечення для проведення тестів та оцінки продуктивності. Автоматичне тестування надає переваги, економить час та гроші для компаній.[26]
Існує декілька підходів до автоматизації тестування:

1. Нотування та відтворення.

Цей спосіб характеризується використанням записів та відтворення. Наприклад, натискання мишки на екран можна замінити на автотест. Для цього необхідно використати утиліту, ввімкнути режим запису та зробити певні кроки. Після цього утиліта може відтворювати всі кроки без тестувальника.

2. Написання сценаріїв.

Для цього способу потрібно писати тестові сценаріїв мовами програмування, такими як Java, Python, C/C++, JavaScript тощо. Завдяки цьому можна вирішувати недоліки підтримки тестів. Цей спосіб є досить дорогим та потребує постійної підтримки.

3. Керування даними тестування.

Метод полягає у створенні програмного коду для автоматизації та верифікації його на основі даних. Дані зберігаються у сховищі, онлайн або базі даних. Використовується даних підхід у певних випадках: якщо потрібно реалізувати схожі, однотипні види перевірок для багатьох варіантів вхідних даних.

4. Тестування по ключовим словам.

В даному методі присутні тести, які є не програмним кодом, а послідовністю дій з параметрами, які в свою чергу описані за допомогою ключових слів. Реалізація виконується за допомогою фреймворку. Даний метод отримав велику популярність через простоту.

На сьогодні в галузі автоматизації програмного продукту є дуже відомий шаблон проектування, а саме – Page Object. Наслідування поточного шаблону запобігає дублювання коду, значно спрощує підтримку тестів, їх написання, підходи. Основний концепт шаблону вміщує в себе: виділене розмежування між кодом для тестів та кодом, який є службовим і веде до елементів самого користувацького інтерфейсу, наявність єдиного репозиторію для служб та операцій.

2.5 Висновки до розділу

Безперервне тестування прискорює створення програмного забезпечення, роблячи весь процес тестування швидшим. А завдяки миттєвому зворотному зв'язку, який допомагає вже на перших етапах виявляти помилки та інші проблеми в програмному продукті, гарантує, що команди розробки будуть створювати високоякісні та надійні програми. Більш того, здатність організувати та проводити ефективне тестування може значно знизити витрати в компанії.

Головна мета всіх команд розробників – забезпечити швидке постачання

якісного та надійного програмного продукту. Для того, щоб забезпечити швидкий та ефективний процес постачання, треба безперервне тестування. Автоматизація - ключ до того, щоб програмне забезпечення могло швидко пройти через всі стадії розробки і надати клієнтам свої функції. Однак це не означає, що команди повинні вкладати весь свій час і ресурси в автоматизацію тестування. Треба розуміти, що можна автоматизувати, а що ні. Правильний вибір охоплення тестів на ранніх етапах розробки має велике значення.

Розділ 3

Технології тестування

Візьмемо за приклад схему для розробки та тестування програмного продукту, реалізованого для мобільних телефонів.

3.1 Cucumber

Cucumber (https://cucumber.io/) – це інструмент тестування, який підтримує структуру поведінки (BDD). Він визначає поведінку додатків, використовуючи простий англійський текст, який визначається мовою під назвою Gherkin. Це програмний засіб, з допомогою якого можна проводити тестування різного по функціональності забезпечення. Cucumber - це дуже потужний та гнучкий фреймворк, який можна використовувати у зв'язці з багатьма іншими популярними інструментами. Наприклад, із Selenium – фреймворком для автоматизації веб-додатків. Таким чином, Cucumber дозволяє розуміти основну поведінку «під капотом» для користувача, при цьому мовою, яка є зрозумілою для усіх членів команди.

У Cucumber для написання тестів використовується Gherkin-нотація, яка визначає структуру тесту та набір ключових слів. Тест записується у файл із розширенням *.feature і може містити як один, так і більше сценаріїв.
Слід звернути увагу на структуру сценарію:

1. Отримати початковий стан системи;
2. Щось зробити;
3. Отримати новий стан системи.

Cucumber-проект складається з двох частин – це текстові файли з описом сценаріїв (*.feature) та файли з реалізацією кроків мовою програмування. Для створення проекту можна використовувати систему автоматизації збирання проектів Apache Maven.

Насамперед потрібно додати Cucumber в залежності Maven, приклад зображено на рисунку 3.

```xml
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>1.2.4</version>
</dependency>
```

Рис.3 Залежність в Maven

Для запуску тестів можна використовувати JUnit або TestNG. Їх також необхідно додати в залежності Maven, як показано на рисунку 4.

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>1.2.4</version>
</dependency>
```

Рис. 4 Залежність в Maven

На рисунку 5 зображено приклад написання Cucumber тесту.

```
@CucumberOptions(features = "src/test/resources/features/GSMArenaNews.feature",
        glue = "com.qaprosoft.carina.demo.cucumber.steps",
        plugin={"pretty",
                "html:target/cucumber-core-test-report",
                "pretty:target/cucumber-core-test-report.txt",
                "json:target/cucumber-core-test-report.json",
                "junit:target/cucumber-core-test-report.xml"}
)
public class CucumberWebSampleTest extends CucumberBaseTest {
    //do nothing here as everything is declared in "GSMArenaNews.feature" and steps

}
```

Рис. 5

На рисунку 6 зображено приклад Feature для тесту.

```
Feature: GSM Arena News testing
  In order to use Cucumber in my project, I want to check how to test GSM Arena News page

  @demo
  Scenario: GSM Arena open page - passing
    Given I am on main page
    When I open 'News' page
    Then page 'News' should be open
    And page 'News' should contains all items
```

Рис. 6

Приклад написання кроків зображено на рисунку 7.

```
public class GSMArenaNewsSteps extends CucumberRunner {

    HomePage homePage = null;
    NewsPage newsPage = null;

    @Given("^I am on main page")
    public boolean iAmOnMainPage() {
        homePage = new HomePage(getDriver());
        homePage.open();
        return homePage.isPageOpened();
    }

    @When("^I open 'News' page$")
    public void iOpenNewsPage()  {
        newsPage = homePage.getFooterMenu().openNewsPage();
        Assert.assertTrue(newsPage.isPageOpened(), "News page is not opened!");
    }

    @Then("^page 'News' should be open$")
    public void pageSettingsShouldBeOpen() { Assert.assertTrue(newsPage.isPageOpened(), "News page is not opened!"); }

    @And("^page 'News' should contains all items$")
    public void pageSettingsShouldContainsAllItems() {
        final String searchQ = "iphone";
        List<NewsItem> news = newsPage.searchNews(searchQ);
        Assert.assertFalse(CollectionUtils.isEmpty(news), "News not found!");
        for(NewsItem n : news) {
            System.out.println(n.readTitle());
            Assert.assertTrue(StringUtils.containsIgnoreCase(n.readTitle(), searchQ), "Invalid search results!");
        }
    }
```

Рис. 7

3.2 Appium

Appium - це безкоштовний кросплатформовий інструмент з відкритим вихідним кодом, який допомагає автоматизувати програми як для Android, так і для iOS. Appium дотримується того ж підходу, що й Selenium WebDriver, який

отримує HTTP-запити у форматі JSON від клієнтів та перетворює їх залежно від платформи, на якій він працює.

Для того, щоб запустити тести за допомогою Appium, перш за все необхідно

встановити усі необхідні драйвера та застосувати усі необхідні налаштування. Використовуючи Appium, не потрібно ніяк додатково модифікувати додатки, які тестуються.

Appium запускається окремим процесом, тому зразу після інсталізації усіх необхідних ресурсів є можливість одразу почати використання. Далі потрібно вказати адресу та порт, по яким й буде здійснюватись запуск Appium. Також у файлі для налаштування необхідно вказувати тестовий додаток, платформу та додаткові бібліотеки.

3.3 API для взаємодії з мобільним додатком

API призначений для того, щоб отримувати дані у вигляді коду, що в свою чергу слугує для автоматизації формування звітів. З його допомогою можна звернутись до конкретних елементів, уникаючи користувацького інтерфейсу. Принцип дії полягає у багаторівневій ієрархії, в якій представлені елементи з однаковою структуризацією.

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій. API є абстрактним поняттям — програмне забезпечення, що пропонує деякий API, що часто називають реалізацією даного API. Високорівневі API часто програють у гнучкості. Виконання деяких функцій нижчого рівня стає набагато складнішим, або навіть неможливим.

Якщо відправити запит, який буде неправильно сфорований, то методи повернуть повідомлення про помилку.

Для роботи з API треба знати методи HTTP.

Метод GET вимагає представлення ресурсу. Запити з цього методу можуть лише витягувати дані.

POST використовується для надсилання сутностей до певного ресурсу. Часто викликає зміну стану або якісь побічні ефекти на сервері.

PUT замінює всі поточні уявлення ресурсу даними запиту.

DELETE видаляє вказаний ресурс.

PATCH використовується для часткового зміни ресурсу.

На рисунках 8,9,10 зображений приклад використання GET запиту.

```
public class GetLicenses extends AbstractApiMethodV2 {

    public GetLicenses(){
        super(null,"api.github/_getLicenses/rs.json");
        replaceUrlPlaceholder("base_url", Configuration.getEnvArg("api_url"));
        request.header("Authorization","Bearer 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");
    }
}
```

Рис. 8

```
    @Test()
    @MethodOwner(owner = "tebenkova")
    public void getLicenses(){
        GetLicenses getLicenses = new GetLicenses();
        getLicenses.expectResponseStatus(HttpResponseStatusType.OK_200);
        getLicenses.callAPI();
        getLicenses.validateResponse();
    }
```

Рис. 9


```
[
  {
    "key": "agpl-3.0",
    "name": "GNU Affero General Public License v3.0",
    "spdx_id": "AGPL-3.0",
    "url": "https://api.github.com/licenses/agpl-3.0",
    "node_id": "MDc6TGljZW5zZTE="
  },
  {
    "key": "apache-2.0",
    "name": "Apache License 2.0",
    "spdx_id": "Apache-2.0",
    "url": "https://api.github.com/licenses/apache-2.0",
    "node_id": "MDc6TGljZW5zZTI="
  }
]
```


Рис. 10

На рисунках 11,12,13,14 зображений приклад використання POST запиту.

```
public class PostCreateRelease extends AbstractApiMethodV2 {

    public PostCreateRelease(){
        super("api.github/_postCreateRelease/rq.json","api.github/_postCreateRelease/rs.json");
        replaceUrlPlaceholder("base_url", Configuration.getEnvArg("api_url"));
        request.header("Authorization","Bearer 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");
    }
}
```

Рис. 11

```
    @Test()
    @MethodOwner(owner = "tebenkova")
    public void postCreateRelease(){
        PostCreateRelease postCreateRelease = new PostCreateRelease();
        postCreateRelease.expectResponseStatus(HttpResponseStatusType.CREATED_201);
        postCreateRelease.callAPI();
        postCreateRelease.validateResponse();
    }

}
```

Рис. 12

```
{
  "tag_name": "tag_name"
}
```

Рис. 13

```
{
  "url": "https://api.github.com/repos/KaterinaTebenkova/ja/releases/56369951",
  "assets_url": "https://api.github.com/repos/KaterinaTebenkova/ja/releases/56369951/assets",
  "upload_url": "https://uploads.github.com/repos/KaterinaTebenkova/ja/releases/56369951/assets{?na
  "html_url": "https://github.com/KaterinaTebenkova/ja/releases/tag/tag_name",
  "id": "skip",
  "author": {
    "login": "KaterinaTebenkova",
```

Рис. 14

На рисунках 15,16 зображений приклад використання DELETE запиту.

```
public class DeleteMethod extends AbstractApiMethodV2 {

    public DeleteMethod(){
        super(null, "api.github/_delete/rs.json");
        replaceUrlPlaceholder("base_url", Configuration.getEnvArg("api_url"));
        request.header("Authorization","Bearer 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");
    }
}
```

Рис. 15

```
    @Test()
    @MethodOwner(owner = "tebenkova")
    public void deleteTest(){
        DeleteMethod deleteMethod = new DeleteMethod();
        deleteMethod.expectResponseStatus(HttpResponseStatusType.NO_CONTENT_204);
        deleteMethod.callAPI();
        deleteMethod.validateResponse();
    }
```

Рис. 16

3.4 Git

Git — розподілена система керування версіями файлів та спільної роботи. Це консольна утиліта, для відстеження та ведення історії зміни файлів у вашому проекті. Для того, щоб тестувальники мали доступ до коду програми, їм необхідно знайти його на Git та клонувати собі на комп'ютер, а потім зміни, які були зроблені, додати на гілку, в якій працює команда.

За допомогою Git ви можете відкотити свій проект до старої версії, порівнювати, аналізувати або заливати свої зміни до репозиторію. Репозиторієм називають сховище вашого коду та історію його змін. Git працює локально і всі ваші репозиторії зберігаються у певних папках на жорсткому диску.

Далі наведені основні команди Git.

Команда git add додає вміст робочої директорії в індекс для наступного комміту. Команда git status показує стан файлів у робочій директорії та індексі: які файли змінені, але не додані до індексу; які очікують коміти в індексі. Команда git commit бере всі дані, додані до індексу за допомогою git add, і зберігає їх зліпок у внутрішній базі даних, а потім зсуває покажчик поточної гілки на цей зліпок. Команда git reset використовується в основному для скасування змін. Команда git clean використовується для видалення сміття із робочої директорії. Команда git

branch вміє перераховувати ваші гілки, створювати нові, видаляти та перейменовувати їх. Команда git checkout використовується для перемикання гілок та вивантаження їхнього вмісту в робочу директорію. Команда git merge використовується для злиття однієї або декількох гілок на поточну. Команда git stash використовується для тимчасового збереження всіх незакоммічених змін для очищення робочої директорії без необхідності керувати незавершеною роботою в нову гілку. Команда git fetch зв'язується з віддаленим репозиторієм і забирає з нього всі зміни, яких у вас поки що немає і зберігає їх локально. Команда git pull працює як комбінація команд git fetch та git merge. Git спочатку забирає зміни із зазначеного віддаленого репозиторію, а потім намагається злити їх із поточною гілкою. Команда git push використовується для встановлення зв'язку з віддаленим репозиторієм, обчислення локальних змін відсутніх у ньому, та власне їх передачі у вищезгаданий репозиторій.[24]

3.5 Написання тестів

Для написання тестів наведених у додатку 1 використано сайт *https://www.gsmarena.com/*.

3.6 Висновки до розділу

Сьогодні існує безліч методів для реалізації автоматизованого тестування. Організація автоматизованого тестування сприяє покращенню взаємодії між членами проекту, які створюють програмний продукт. Аналітики, які ставлять цілі, розробники, які пишуть код, тестувальники, які перевіряють програму на проблеми всі разом створюють один єдиний програмний продукт, який потім виводиться на міжнародний ринок.

Як і в будь-якій діяльності, у разі застосування тестування є свої плюси та мінуси. Наприклад, треба писати самі тести та писати їх правильно потрібно будувати свій код таким чином, щоб тести могли виконуватись.

До плюсів, на мою думку, відносяться підвищення якості функціоналу, що розробляється зниження трудовитрат на повторне тестування, тим більше, що тестувати корисно кожне складання цільової системи, підвищення привабливості

компанії для замовників. Застосування технологій автоматизованого тестування є показником високої культури розробки та практично світовою тенденцією.

Також розроблені тести можуть бути монетизовані як продукт інтелектуальної діяльності. Наприклад, після завершення проекту тести можуть бути передані замовнику, якщо замовник планує розвивати систему силами власного ІТ-підрозділу.

Висновки

Дана робота почалася з вивчення і розуміння необхідного і потрібного програмного забезпечення для успішного виконання роботи. Завдяки цій роботі було важливо навчитися писати автоматизовані тести на мові Java і мати можливість запускати тести. Також було проаналізовані теоретичні аспекти написання коду.

Була також проаналізована різниця між ручним та автоматизованим тестуванням. Розглянуто переваги та недоліки кожної сторони. Було розглянуто різні технології тестування та декілька фреймворків для автоматизованого тестування.

Написані тести були перевірені та випробувані, це показало коректність роботи програмного засобу. Тести ефективно виконують поставлені задачі. Вони були написані за допомогою IntelliJ IDEA - інтегрованого середовища розробки програмного забезпечення для багатьох мов програмування, зокрема Java, JavaScript, Python, розроблене компанією JetBrains.

В ході розробки тестів я покращила свої знання в різноманітних технологіях, які використовуються під час розробки автоматизованих тестів.

Отже, забезпечення якості – невід'ємна частина циклу розробки програмного продукту. При кожній незначній зміні програмного продукту існують аргументи щодо необхідності виконання його автоматизованого тестування. Проте деякі програмісти вважають, що перевірка функціональності програмного забезпечення повинна здійснюватися лише у найважливіших частинах циклу розробки продукту.

Додаток

```
public class HomePage extends AbstractPage {
    private static final Logger LOGGER =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

    @FindBy(id = "footmenu")
    private FooterMenu footerMenu;

    @FindBy(xpath = "//div[contains(@class, 'brandmenu-v2')]//a")
    private List<ExtendedWebElement> brandLinks;

    @FindBy(className = "news-column-index")
    private ExtendedWebElement newsColumn;

    @FindBy(id = "header")
    private HeaderMenu header;

    public HomePage(WebDriver driver) {
        super(driver);
        setUiLoadedMarker(newsColumn);
        setPageAbsoluteURL(R.CONFIG.get(Configuration.Parameter.URL.getKey()));
    }

    public FooterMenu getFooterMenu() {
        return footerMenu;
    }

    public HeaderMenu getHeader() {
        return header;
    }

    public BrandModelsPage selectBrand(String brand) {
        LOGGER.info("selecting '" + brand + "' brand...");
        for (ExtendedWebElement brandLink : brandLinks) {
            String currentBrand = brandLink.getText();
            LOGGER.info("currentBrand: " + currentBrand);
            if (brand.equalsIgnoreCase(currentBrand)) {
                brandLink.click();
                return new BrandModelsPage(driver);
            }
        }
        throw new RuntimeException("Unable to open brand: " + brand);
    }

    public WeValuePrivacyAd getWeValuePrivacyAd() {
       return new WeValuePrivacyAd(driver);
    }

}
public class FooterMenu extends AbstractUIObject {
    @FindBy(linkText = "Home")
    private ExtendedWebElement homeLink;

    @FindBy(xpath = "//div[@class='footer-inner']//a[contains(text(),'Compare')]")
    private ExtendedWebElement compareLink;

    @FindBy(linkText = "News")
    private ExtendedWebElement newsLink;

    public FooterMenu(WebDriver driver, SearchContext searchContext) {
        super(driver, searchContext);
    }
```

```
    public HomePage openHomePage() {
        homeLink.click();
        return new HomePage(driver);
    }

    public CompareModelsPage openComparePage() {
        compareLink.click();
        return new CompareModelsPage(driver);
    }

    public NewsPage openNewsPage() {
        newsLink.click();
        return new NewsPage(driver);
    }
}
public class HeaderMenu extends AbstractUIObject {
    @FindBy(xpath = "//i[contains(@class, 'head-icon icon-signout')]")
    private ExtendedWebElement logOutButton;

    @FindBy(id = "login-active")
    private ExtendedWebElement loginIcon;

    public HeaderMenu(WebDriver driver, SearchContext searchContext){
        super(driver, searchContext);
    }

    public LoginMenu openLoginMenu(){
        loginIcon.click();
        return new LoginMenu(driver);
    }

    public boolean isLogOutButtonPresent(){
        return logOutButton.isPresent();
    }
}
public class LoginMenu extends AbstractUIObject {
    @FindBy(id = "login-popup2")
    private ExtendedWebElement loginMenu;

    @FindBy(id = "email")
    private ExtendedWebElement enterEmail;

    @FindBy(id = "upass")
    private ExtendedWebElement enterPass;

    @FindBy(id = "nick-submit")
    private ExtendedWebElement submitButton;

    @FindBy(id = "normal-text res-error")
    private WebElement errorText;

    LoginMenu(WebDriver driver){
        super(driver);
    }

    public boolean isLoginMenuPresent() {
        return loginMenu.isElementPresent();
    }

    private HomePage submitButton() {
        submitButton.click();
        return new HomePage(getDriver());
```

```
    }

    public HomePage login(UserService user) {
        enterEmail.type(user.getUser().getEmail());
        enterPass.type(user.getUser().getPassword());
        return submitButton();
    }

    public LoginPage wrongLogin(UserService user) {
        enterEmail.type(user.wrongLogin().getWrongEmail());
        enterPass.type(user.wrongLogin().getPassword());
        return clickButton();

    }

    public boolean isLoginLabelPresent() {
        return loginMenu.isElementPresent();
    }

    private LoginPage clickButton() {
        submitButton.click();
        return new LoginPage(getDriver());
    }
}
public class LoginPage extends AbstractPage {
    private static final Logger LOGGER = Logger.getLogger(LoginPage.class);

    @FindBy(xpath = "//div[@class= 'normal-text res-error']/child::p")
    private ExtendedWebElement errorText;

    public LoginPage(WebDriver driver) {
        super(driver);
    }

    public String loginError() {
        LOGGER.info("Login result [" + errorText.getText() + "]");
        return errorText.getText();
    }

}
public class User {
    private String email;
    private String password;
    private String wrongEmail;

    public User() {

    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
```

```
    }

    public String getWrongEmail(){ return wrongEmail;}

    public void setWrongEmail(String wrongEmail) { this.wrongEmail = wrongEmail;}

}
public class UserService {

    public User getUser(){
        User user = new User();
        user.setEmail(R.TESTDATA.get("email"));
        user.setPassword(R.TESTDATA.get("password"));
        return user;
    }

    public User wrongLogin(){
        User user = new User();
        user.setWrongEmail(R.TESTDATA.get("wrong_email"));
        user.setPassword(R.TESTDATA.get("password"));
        return user;
    }
}
public class WebTest implements IAbstractTest {
    @Test
    @MethodOwner(owner = "etebenkova")
    public void verifyLogin(){
        // Open GSM Arena home page and verify page is opened
        HomePage homePage = new HomePage(getDriver());
        homePage.open();
        UserService userService = new UserService();
        Assert.assertTrue(homePage.isPageOpened(), "Home page is not opened");
        LoginMenu loginMenu = homePage.getHeader().openLoginMenu();
        Assert.assertTrue(loginMenu.isLoginMenuPresent(),"Login form isn't
present");
        loginMenu.login(userService);
        Assert.assertTrue(homePage.getHeader().isLogOutButtonPresent(),"Button
isn't present");
    }

    @Test
    @MethodOwner(owner = "etebenkova")
    public void wrongLogin(){
        HomePage homePage = new HomePage(getDriver());
        homePage.open();
        UserService userService = new UserService();
        Assert.assertTrue(homePage.isPageOpened(),"Home page isn't opened");
        LoginMenu loginMenu = homePage.getHeader().openLoginMenu();
        Assert.assertTrue(loginMenu.isLoginMenuPresent(),"Login form isn't
present");
        loginMenu.wrongLogin(userService);
        Assert.assertTrue(loginMenu.isLoginLabelPresent(), "isn't present");

    }
public class WebSampleTest implements IAbstractTest {
    @Test()
    @TestLabel(name = "feature", value = {"web", "regression"})
    public void testModelSpecs() {
        // Open GSM Arena home page and verify page is opened
        HomePage homePage = new HomePage(getDriver());
        homePage.open();
        Assert.assertTrue(homePage.isPageOpened(), "Home page is not opened");
```

```
        //Closing advertising if it's displayed
        homePage.getWeValuePrivacyAd().closeAdIfPresent();

        // Select phone brand
        homePage = new HomePage(getDriver());
        BrandModelsPage productsPage = homePage.selectBrand("Samsung");
        // Select phone model
        ModelInfoPage productInfoPage = productsPage.selectModel("Galaxy A52 5G");
        // Verify phone specifications
        SoftAssert softAssert = new SoftAssert();
        softAssert.assertEquals(productInfoPage.readDisplay(), "6.5\"", "Invalid
display info!");
        softAssert.assertEquals(productInfoPage.readCamera(), "64MP", "Invalid
camera info!");
        softAssert.assertEquals(productInfoPage.readRam(), "6/8GB RAM", "Invalid
ram info!");
        softAssert.assertEquals(productInfoPage.readBattery(), "4500mAh", "Invalid
battery info!");
        softAssert.assertAll();
    }


    @Test()
    @TestLabel(name = "feature", value = {"web", "acceptance"})
    public void testCompareModels() {
        // Open GSM Arena home page and verify page is opened
        HomePage homePage = new HomePage(getDriver());
        homePage.open();
        Assert.assertTrue(homePage.isPageOpened(), "Home page is not opened");
        // Open model compare page
        FooterMenu footerMenu = homePage.getFooterMenu();
        Assert.assertTrue(footerMenu.isUIObjectPresent(2), "Footer menu wasn't
found!");
        CompareModelsPage comparePage = footerMenu.openComparePage();
        // Compare 3 models
        List<ModelSpecs> specs = comparePage.compareModels("Samsung Galaxy J3",
"Samsung Galaxy J5", "Samsung Galaxy J7 Pro");
        // Verify model announced dates
        SoftAssert softAssert = new SoftAssert();
        softAssert.assertEquals(specs.get(0).readSpec(SpecType.ANNOUNCED), "2016,
March 31");
        softAssert.assertEquals(specs.get(1).readSpec(SpecType.ANNOUNCED), "2015,
June 19");
        softAssert.assertEquals(specs.get(2).readSpec(SpecType.ANNOUNCED), "2017,
June");
        softAssert.assertAll();
    }

    @Test()
    @TestLabel(name = "feature", value = {"web", "acceptance"})
    public void testNewsSearch() {
        HomePage homePage = new HomePage(getDriver());
        homePage.open();
        Assert.assertTrue(homePage.isPageOpened(), "Home page is not opened!");

        NewsPage newsPage = homePage.getFooterMenu().openNewsPage();
        Assert.assertTrue(newsPage.isPageOpened(), "News page is not opened!");

        final String searchQ = "iphone";
        List<NewsItem> news = newsPage.searchNews(searchQ);
        Assert.assertFalse(CollectionUtils.isEmpty(news), "News not found!");
        SoftAssert softAssert = new SoftAssert();
        for(NewsItem n : news) {
```

```
            System.out.println(n.readTitle());
            softAssert.assertTrue(StringUtils.containsIgnoreCase(n.readTitle(),
searchQ),
                    "Invalid search results for " + n.readTitle());
        }
        softAssert.assertAll();
    }
```

Список використаних джерел

1. D. L. D. Yuan-Fang Li, Paramjit K. Das, “Two decades of web application testing – a survey of recent advances,” Information Systems, vol. 43, pp. 20 – 54, 2014.

2. Jeff Kramer and Orit Hazzan. The role of abstraction in software engineering. In Proceedings of the 28th international conference on Software engineering, pages 1017–1018. ACM, 2006

3. Janardhanudu, Girish, “White Box Testing”, https://buildsecurityin.uscert.gov/daisy/bsi/articles/bestpractices/white-box/259-BSI.html, February 08, 2009. 4. H. Liu and H. B. Kuan Tan, “Covering code behavior on input validation in functional testing,” Information and Software Technology, vol. 51, no. 2, pp. 546–553, Feb. 2009.

5. Shari Lawrence Pfleeger and Joanne M. Atlee. Software engineering: theory and practice (internationaledition). Pearson, 2010.

6. Dorota Huizinga and Adam Kolawa. Automated defect prevention: best practices in software management. Wiley-Interscience, IEEE Computer Society, 2007.

7. Simon Stewart and David Burns. Webdriver 2014-06-18.

8. G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," PloS one, vol. 12, p. e0177459, 2017.

9. E. Dustin, T. Garrett, and B. Gauf, Implementing Automated Software Testing. Pearson Education, 2009.

10. Automation testing. URL:https://smartbear.com/learn/automated-testing/best-practicesfor-automation/. 11. How do use API. URL:https://www.coindesk.com/learn/ethereum-101/ethereumsmart-contracts-work.

12. Lessons Learned in Software Testing/T. Pettyhort – Nevada, 2001. - 300 с.

13. Software Testing Techniques – Москва, Бейзер Б. 1983. – 284 с. 5. Software Testing Automation Tips: 50 Things Automation Engineers Should Know, Nikson, 2002. – 286с.

14. The Way of the Web Tester – A beginners guide to Automating tests – Jonathan Rasmussen, 1998. - 5с.

15. Java for Testers – Alan Richardson, 2019. - 89с.

16. Agile Testing, a practical guide for testers and agile teams. – Lisa Crispin/Janet Gregory, 2010. – 58c.
17.30 Things every software tester should learn - Heather Reid, 2020.
18. Performance Testing with JMeter3 - Bayo Erinle, 2011. – 56c.
19.https://conf.ztu.edu.ua/wp-content/uploads/2016/06/248.pdf
20.https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/
21.https://habr.com/ru/post/542422/
22.https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80
23.https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F
24.https://proglib.io/p/git-cheatsheet
25. https://uk.wikipedia.org/wiki/Apache_HTTP_Server
26. https://ela.kpi.ua/bitstream/123456789/43121/1/SenedzhukAYu_bakalavr.pdf
27.https://er.nau.edu.ua/bitstream/NAU/47730/1/%d0%a4%d0%9a%d0%9a%d0%9f%d0%86_2020_123_%d0%91%d0%b5%d0%bb%d0%be%d0%b7%d1%8c%d0%be%d1%80%d0%be%d0%b2%d0%b0%d0%90%d0%a1.pdf