Державний торговельно–економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Програмний модуль автоматизованого тестування графічного інтерфейсу веб-додатків»

Студента 4 курсу, 7 групи,
спеціальності 121 «Інженерія програмного забезпечення»
освітньої програми «Інженерія програмного забезпечення» Мельниковича Валентина Васильовича

_____
підпис студента

Науковий керівник
кандидат педагогічних наук,
доцент кафедри інженерії програмного забезпечення та кібербезпеки Котенко Наталія Олексіївна

_____
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент кафедри інженерії програмного забезпечення та кібербезпеки Рзаєва Світлана Леонідівна

_____
підпис гаранта

КИЇВ – 2023

Державний торговельно–економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Мельниковича Валентина Васильовича
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Програмний модуль автоматизованого тестування графічного інтерфейсу веб-додатків»

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета дипломної роботи — дослідження методів та алгоритмів автоматизації тестування веб-додатків, а також розробка системи автоматизованого тестування веб-додатків..

Об'єкт дослідження є модулі автоматизованого тестування графічного інтерфейсу веб-додатків.

Предмет дослідження розробки є сам процес автоматизації тестування веб-додатків. Будуть розглядатись різні методи, алгоритми та підходи до автоматизації тестування, спрямовані на поліпшення ефективності та якості тестування веб-додатків

Для вирішення поставлених завдань необхідно використовувати методи наукового пізнання. Зокрема, такі методи, як абстракція (узагальнення та виділення головних аспектів проблеми), порівняння (аналіз та порівняння різних підходів до автоматизації тестування), узагальнення (виведення загальних принципів та висновків на основі проведених досліджень), аналогія (використання аналогічних прикладів для розуміння та розв'язання проблеми), індукція (загальний висновок на основі конкретних спостережень) та дедукція (виведення конкретних висновків з загальних принципів). Ці методи допоможуть аналізувати, оцінювати та узагальнювати результати дослідження.

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ В СФЕРІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ ВЕБ-ДОДАТКІВ.

1.1. Веб-додаток: сутність поняття

1.2. Аналіз можливості та особливості тестування веб-додатків

1.3. Характеристика сучасних систем автоматизованого тестування веб-додатків

1.4. Висновок до розділу 1

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ПРИНЦИПІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБДОДАТКІВ

2.1. Цільовий аналіз системи автоматизованого тестування веб-додатків

2.2. Дослідження алгоритмів реалізації системи автоматизованого тестування веб-додатків

2.3. Дослідження інструментарію для розробки та удосконалення системи автоматизованого тестування веб-додатків

2.4. Висновок до розділу 2

РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБДОДАТКІВ

3.1. Проектування системи автоматизованого тестування веб-додатків

3.2. Розробка системи автоматизованого тестування веб-додатків

3.3 Тестування системи автоматизованого тестування веб-додатків

3.4. Висновок до розділу 3

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2022*	*21.09.2022*
2.	*Розробка та затвердження завдання на проєкт*	*14.11.2022*	*14.11.2022*
3.	*Вступ та перелік літературних джерел*	*23.12.2022*	*23.12.2022*
4.	*Розділ 1. Огляд та аналіз існуючих рішень в сфері автоматизованого тестування графічного інтерфейсу веб-додатків*	*27.01.2023*	*27.01.2023*
5.	*Розділ 2. Дослідження принципів автоматизованого тестування веб-додатків*	*03.03.2023*	*03.03.2023*
6.	*Розділ 3. Проектування системи автоматизованого тестування веб-додатків*	*14.04.2023*	*14.04.2023*
7.	*Висновки*	*28.04.2023*	*28.04.2023*
8.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*17.05.2023*	*17.05.2023*
9.	*Підготовка автореферату та презентації доповіді*	*26.05.2023*	*26.05.2023*
10.	*Попередній захист випускного кваліфікаційного проєкту*	*29.05.2023 – 02.06.2023*	
11.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*05.06.2023*	*05.06.2023*
12.	*Здача прошитого випускного кваліфікаційного проєкту на кафедру*	*05.06.2023*	*05.06.2023*
13.	*Публічний захист випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту Котенко Н.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Рзаєва С.Л.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Мельникович В.В
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента _____Мельниковича В.В.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Рзаєва С.Л.

(прізвище, ініціали, підпис)

Завідувач кафедри _____ Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження, програмний модуль присвячений розробці та реалізації автоматизованого тестування графічного інтерфейсу веб-додатків. Метою роботи було створення ефективного та надійного інструменту для тестування графічного інтерфейсу веб-додатків з метою забезпечення якості та функціональності системи.

В ході порівняльного аналізу аналогічних рішень були визначені основні функціональні вимоги до програмного модуля автоматизованого тестування графічного інтерфейсу веб-додатків.

Модуль був реалізований з використанням сучасних технологій та мов програмування, що дозволило досягти високої продуктивності та надійності системи тестування. Була проведена серія тестів, які підтвердили функціональність та якість програмного модуля.

Готовий програмний модуль автоматизованого тестування графічного інтерфейсу веб-додатків успішно протестовано. Отриманий модуль представляє собою ефективне рішення для автоматизації тестування графічного інтерфейсу веб-додатків, що сприяє швидкій та зручній перевірці функціональності та якості цих додатків.

Ключові слова: графічний інтерфейс, база даних, веб-додаток, автоматизований.

ABSTRACT

According to the purpose of the research, the software module is dedicated to the development and implementation of automated testing of the graphical interface of web applications. The goal of the work was to create an effective and reliable tool for testing the graphical interface of web applications in order to ensure the quality and functionality of the system.

In the course of a comparative analysis of similar solutions, the main functional requirements for the software module for automated testing of the graphical interface of web applications were determined.

The module was implemented using modern technologies and programming languages, which made it possible to achieve high performance and reliability of the testing system. A series of tests was conducted that confirmed the functionality and quality of the software module.

The ready-made software module for automated testing of the graphical interface of web applications has been successfully tested. The resulting module is an effective solution for automating the testing of the graphical interface of web applications,which facilitates quick and convenient testing of the functionality and quality of these applications.

Keywords: graphical interface, database, web application, automated.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

АЗ – апаратне забезпечення

ПК – персональний комп’ютер

БД – база даних

ОС – операційна система

ОП – оперативна пам’ять

СІ – сервер інтеграції

API – прикладний програмний інтерфейс

ТЗ – технічне завдання

GUI – графічний інтерфейс користувача

ЗМІСТ

					ДТЕУ 121 07–18.БР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		23.12.22	Програмний модуль автоматизованого тестування графічного інтерфейсу веб-додатків	*Стадія*	*Арку*	*Аркушів*
Керівник		Котенко Н.О.		23.12.22		*З*	*3*	*55*
Гарант		Рзаєва С.Л.		23.12.22		*Факультет інформаційних технологій 4 курс, 7 група*		
Розробив		Мельникович В.В.		23.12.22	*Зміст*			

ВСТУП

Актуальність. Методи і алгоритми проектування автоматизованих засобів тестування зорієнтовані на швидку розробку і ефективне використання цих засобів в ітеративних процесах розробки програмного продукту. Підходи, які можуть допомогти розробити автоматизовані засоби тестування з меншими затратами і швидко:

1. Рефакторинг тестів: при ітеративному тестуванні ви часто отримуєте нові версії програмного продукту з високою частотою. Замість того, щоб розробляти нові тести з нуля кожного разу, слід удосконалювати існуючі тести шляхом рефакторингу. Можна виявити загальні шаблони або функціональності, які залишаються стабільними між версіями, і перенести їх у загальні тести, що дозволить ефективно використовувати їх для нових версій програмного продукту.
2. Використання фреймворків тестування: використання готових фреймворків тестування може значно спростити розробку автоматизованих засобів тестування. Фреймворки, такі як Selenium для веб-додатків або Appium для мобільних додатків, надають потужний набір інструментів і функціональностей для автоматизації тестування.
3. Тестування на рівні API: тестування на рівні API дозволяє проводити швидке та ефективне тестування функціональності програмного продукту без необхідності взаємодії з інтерфейсом користувача. Це дозволяє скоротити час розробки та виконання тестів, оскільки взаємодія з API може бути більш прямолінійною і швидкою.

Мета дипломної роботи — дослідження методів та алгоритмів автоматизації тестування веб-додатків, а також розробка системи автоматизованого тестування веб-додатків.

Для досягнення поставленої мети були сформульовані наступні завдання:

1. Огляд літератури: проведення детального огляду існуючих методів і алгоритмів автоматизації тестування веб-додатків. Вивчення різних підходів, фреймворків та інструментів, які використовуються для автоматизації тестування веб-додатків.

2. Вибір методу тестування: на основі огляду літератури вибір підходу або методу тестування веб-додатків, який відповідає вашим потребам і вимогам дослідження. Наприклад, це може бути тестування на рівні інтерфейсу користувача (UI), тестування на рівні API або комбінація різних рівнів тестування.

3. Розробка системи тестування: розробка системи автоматизованого тестування веб-додатків на основі обраного методу тестування. Використання відповідних фреймворків і інструментів для реалізації автоматизованих тестів. Забезпечення, щоб система мала гнучкість і масштабованість, щоб легко вносились зміни і додавались нові тести залежно від розширення веб-додатків.

4. Експерименти і оцінка: проведення експеримента з розробленою системою тестування, виконання різних тестів на реальних веб-додатках. Оцінювання результатів тестування, враховуючи якість і покриття тестів, швидкість виконання тестів та ін.

Об'єктом дослідження є програмні комплекси, призначені для тестування веб-додатків.

У роботі будуть досліджуватись ці комплекси та їхня ефективність у контексті автоматизації тестування веб-додатків.

Предметом дослідження є процес автоматизації тестування веб-додатків.

Будуть розглядатись різні методи, алгоритми та підходи до автоматизації тестування, спрямовані на поліпшення ефективності та якості тестування веб-додатків.

Для вирішення поставлених завдань необхідно використовувати методи наукового пізнання. Зокрема, такі методи, як абстракція (узагальнення та виділення головних аспектів проблеми), порівняння(аналіз та порівняння різних підходів до автоматизації тестування), узагальнення (виведення загальних принципів та висновків на основі проведених досліджень), аналогія (використання аналогічних прикладів для розуміння та розв'язання проблеми), індукція (загальний висновок на основі конкретних спостережень) та дедукція (виведення конкретних висновків з загальних принципів). Ці методи допоможуть аналізувати, оцінювати та узагальнювати результати дослідження.

РОЗДІЛ 1
ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ В СФЕРІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ ВЕБ-ДОДАТКІВ.

1.1. Веб-додаток: сутність поняття

Веб-додатків є веб-сайтом, але з особливістю, що сторінки в ньому містять частково або повністю несформований вміст. Це означає, що фінальний вигляд та вміст сторінки формуються лише після запиту відвідувача до веб-сервера. Формування остаточного вмісту залежить від конкретного запиту, який створюється на основі дій користувача, таких як введення даних, вибір параметрів або взаємодія з різними елементами на сторінці. Це призводить до створення динамічних сторінок, які можуть змінюватися і адаптуватися до конкретних потреб користувача.

Такі веб-додатки дозволяють створювати більш інтерактивні та персоналізовані веб-сайти, оскільки вони можуть змінювати свій вміст в реальному часі, відповідаючи на взаємодію користувача. Це дає більшу гнучкість та можливості для розробників створювати багатофункціональні та динамічні веб-додатки. Однак це також ставить вимоги до тестування таких додатків, оскільки потрібно переконатися, що вони працюють правильно в різних сценаріях взаємодії з користувачем та генерують правильний вміст згідно з вхідними параметрами. Так, використання веб-додатків дійсно приносить певну користь як для відвідувачів веб-сайтів, так і для їхрозробників. Деякі можуть бути пов'язані з використанням веб-додатків, включають:

1. Для відвідувачів веб-сайтів: швидкий доступ до інформації: веб-додатки можуть забезпечувати швидкий та зручний доступ до необхідної інформації на веб-сайтах з великим обсягом даних. Вони можуть пропонувати розширені функціональні можливості пошуку, фільтрації та сортування, що допомагають знайти потрібну інформацію швидко і ефективно.
2. Персоналізація та інтерактивність: веб-додатки дозволяють створювати персоналізований вміст для користувачів на основі їхніх вимог і взаємодії. Вони можуть надавати інтерактивні елементи, які дозволяють відвідувачам активно взаємодіяти з веб-сайтом, наприклад, заповнювати форми, розміщувати коментарі, зберігати налаштування та інше.
3. Мобільний доступ: веб-додатки можуть бути доступні через мобільні пристрої, що дозволяє користувачам отримувати доступ до веб-сайту з будь-якого місця та в будь-який час. Це забезпечує зручність і доступність для користувачів, які використовують мобільні пристрої
4. Розширені можливості функціоналу: веб-додатки дозволяють розробникам створювати складніші функціональні можливості на веб-сайтах. Вони можуть використовувати скриптові мови, такі як JavaScript, для взаємодії з веб-сторінками та маніпулювання вмістом. Це дозволяє створювати багатофункціональні додатки, такі як онлайн-магазини з кошиком покупок, соціальні мережі зі змістом у реальному часі, інтерактивні форми та інше.
5. Більша гнучкість та швидкість розробки: веб-додатки дозволяють розробникам швидко створювати та запроваджувати новий функціонал на веб-сайті. Вони можуть використовувати розроблені раніше компоненти та бібліотеки для швидкого побудови нових функціональних елементів. Крім того, деякі фреймворки, такі як

Angular, React або Vue.js, надають структуру та інструменти для швидкої розробки складних веб-додатків.

6. Підтримка багатоплатформеності: веб-додатки зазвичай можуть працювати на різних платформах і пристроях, що забезпечує широку аудиторію та більшу доступність. Розробникам не потрібно створювати окремі версії для різних операційних систем чи пристроїв. Застосування веб-стандартів та адаптивного дизайну дозволяє підтримувати різні розміри екранів і пристрої з одним кодом.
7. Простота розгортання та оновлення: веб-додатки зазвичай простіше розгорнути та оновити.

Веб-додатки, які дозволяють здійснювати пошук, упорядковувати та переміщатися по вмісту, є дуже корисними для відвідувачів веб-сайтів. Дані додатки надають зручні інструменти для навігації та взаємодії з обсягом інформації на веб-сайті.

Microsoft MSDN є прикладом веб-додатку, який забезпечує розширені можливості пошуку та упорядкування вмісту. MSDN надає доступ до документації, ресурсів та інструментів для розробників Microsoft. Користувачі можуть швидко знайти необхідну інформацію за допомогою пошуку, фільтрів та упорядкування згідно з різними категоріями або технологіями .

Amazon є ще одним прикладом веб-додатку, який надає великий обсяг вмісту та забезпечує зручну навігацію та пошук. Відвідувачі можуть шукати товари за різними критеріями, використовувати фільтри для точнішого упорядкування результатів та зручно переміщатися по категоріях та підкатегоріях товарів.

Ці веб-додатки демонструють, як використання розширених функцій пошуку, упорядкування та навігації допомагає відвідувачам швидко та ефективно знаходити потрібну інформацію або товари на веб-сайтах з великим обсягом даних.

Необхідно вказати на важливу користь веб-додатків - збір, зберігання і аналіз отриманих даних від відвідувачів сайту. Розглянемо деякі приклади, що ілюструють цю можливість.

1. Інтерактивні сторінки банків: веб-додатки, що використовуються банками, дозволяють клієнтам здійснювати онлайн-банківські операції, такі як переказ коштів, перевірка балансу рахунків, оплата рахунків тощо. Ці додатки збирають дані про транзакції та фінансові операції клієнтів, зберігають їх у базі даних і надають можливість аналізувати ці дані для звітності та аналітики.
2. Контроль товарних запасів: веб-додатки можуть бути використані для відстеження та керування запасами товарів у компаніях. Вони дозволяють збирати дані про наявність товарів, замовлення та розподіл запасів, що допомагає компаніям планувати поставки, відстежувати попит та аналізувати ефективність управління запасами.
3. Соціологічні дослідження та опитування: веб-додатки надають можливість проводити соціологічні дослідження та збирати дані від користувачів через опитування та форми заповнення. Ці дані можуть бути збережені в базі даних для подальшого аналізу, статистичних обробок та формування звітів для висновків та прийняття рішень.
4. Форми для зворотного зв'язку з користувачами: веб-додатки часто містять форми зворотного зв'язку, що дозволяють користувачам залишати відгуки.

Необхідно вказати на ще одну користь веб-додатків - звільнення веб-дизайнера від рутинної роботи постійного оновлення HTML-сторінок сайту. Давайте розглянемо деякі приклади, що ілюструють цю можливість.

1. Веб-версія журналу «The Economist»: веб-додаток, який використовується «The Economist», дозволяє редакторам журналу завантажувати та оновлювати нові статті та вміст безпосередньо в

системі управління вмістом (Content Management System, CMS). Додаток автоматично оновлює HTML-сторінки сайту згідно з новим вмістом, що дозволяє редакторам сконцентруватись на створенні якісного контенту, а не на технічних аспектах оновлення сторінок.

2. Служби новин CNN: Подібно до «The Economist», веб-додаток служб новин, таких як CNN, дозволяє редакторам і журналістам оновлювати новини та інформацію безпосередньо в CMS. Нові матеріали автоматично відображаються на веб-сайті, що дозволяє підтримувати актуальність і свіжість контенту без необхідності ручного оновлення HTML-сторінок.

Загалом, веб-додатки, які підтримують автоматичне оновлення веб-сайту на основі нового вмісту, роблять процес управління контентом більш ефективним та зручним для постачальників вмісту, оскільки вони можуть швидко публікувати свіжу інформацію без втрати часу на рутинну технічну роботу.

Статична веб-сторінка є незмінною і завжди відображається перед користувачем в тому самому вигляді, як його зберігає сервер. При запиті веб-браузера сервер просто відправляє статичну сторінку без будь-яких змін. Це означає, що вміст та вигляд статичної сторінки залишаються незмінними незалежно від контексту або взаємодії з користувачем. Наприклад, проста статична сторінка може містити лише текст та зображення, які залишаються постійними під час відображення на різних пристроях або при різних діях користувача.

З іншого боку, динамічна веб-сторінка зазнає змін перед тим, як вона відправляється до веб-браузера користувача. Це дозволяє серверу генерувати сторінку на основі різних факторів, таких як дані з бази даних, параметри запиту, інтерактивна взаємодія з користувачем та інші фактори. Наприклад,

динамічна сторінка може відображати персоналізовану інформацію для кожного користувача, залежно від його профілю або історії перегляду.

Динамічні веб-додатки зазвичай використовують серверні технології, такі як PHP, Python, Ruby, або фреймворки, щоб обробляти дані та генерувати сторінки на льоту перед відправкою їх до веб-браузера користувача. Це дозволяє створювати більш інтерактивні та гнучкі веб-додатки, які можуть змінювати свій вміст.

1.2. Аналіз можливості та особливості тестування веб-додатків

Веб-додаток як клієнт-серверний додаток, де браузер виступає клієнтом, а веб-сервер – сервером. Це означає, що взаємодія між користувачем і додатком відбувається через комунікацію між браузером і сервером.

Браузер, як клієнт, відправляє запити до веб-сервера, який обробляє ці запити і надсилає відповідь браузеру. Відповідь може бути статичною сторінкою (наприклад, HTML, CSS, зображення) або динамічною сторінкою, яку веб-сервер генерує на льоту відповідно до запиту.

Як вже було зазначено, важливо тестувати як клієнтську, так і серверну частини веб-додатка. Тестування клієнтської частини включає перевірку коректності відображення та взаємодії елементів у браузері, тестування взаємодії з користувачем, перевірку функціональності JavaScript і так далі.

Тестування серверної частини включає перевірку правильності обробки запитів, реагування на помилки, відповідність базовим вимогам безпеки, оптимізацію продуктивності і багато іншого. Важливо також тестувати взаємодію між клієнтом і сервером, зокрема перевіряти передачу даних, аутентифікацію та авторизацію, зберігання сесій та інші аспекти комунікації.

Під час тестування веб-додатків варто враховувати різні сценарії використання, перевіряти роботу на різних пристроях та браузерах, а також здійснювати навантажувальне тестування для перевірки продукту.

Деякі види веб-додатків. Дозвольте надати короткий опис кожного з них:

1. Рейтинги: Веб-додатки для оцінки і ранжування елементів, таких як продукти, послуги, відгуки тощо. Користувачі можуть залишати свої оцінки та коментарі.
2. Форми відправки повідомлень: Додатки, які дозволяють користувачам надсилати повідомлення через веб-сайт, наприклад, зворотний зв'язок або контактні форми.
3. Форми реєстрації: Веб-додатки, що містять форми для реєстрації користувачів, де вони можуть створювати облікові записи та входити на сайт.
4. Гостьові книги: Додатки, які дозволяють користувачам залишати свої коментарі, відгуки або привітання на веб-сайті.
5. Форуми: Веб-додатки для обговорення різних тем та взаємодії між користувачами шляхом створення тем, відповідей та обміну повідомленнями.
6. Форми завантаження (upload): Додатки, що дозволяють користувачам завантажувати файли на сервер, наприклад, зображення, відео або документи.
7. Системи голосування: Веб-додатки, які дозволяють користувачам брати участь у голосуванні за певні питання або опитування.
8. Пошукові системи: Додатки, що забезпечують пошук інформації на веб-сайті чи в Інтернеті за допомогою ключових слів або фраз.
9. Движки сайту: Веб-додатки, які забезпечують основну функціональність сайту, таку як навігація, відображення вмісту та інші елементи.
10. Content Management System (CMS): Системи управління контентом.

Можна перерахувати основні вразливості, які властиві веб-додаткам. Короткий опис кожної з них:

1. Перевірка вхідних даних: Веб-додатки можуть бути вразливими до атак, якщо не виконується адекватна перевірка та очищення вхідних даних, що передаються через веб-форми. Це може призвести до виконання шкідливого коду або некоректної обробки даних.
2. Некоректна обробка вхідних даних: Недостатня перевірка та обробка вхідних даних може призвести до використання шкідливих символів, таких як нульовий байт або символи рівня директорій, що можуть використовуватися для доступу до конфіденційної інформації або виконання несанкціонованих операцій.
3. Переповнення буферу: Вразливості, пов'язані з переповненням буферу, можуть виникати, коли програма не обмежує кількість даних, які можуть бути введені у буфер, що може призвести до перезапису важливих даних або виконання шкідливого коду.
4. Некоректна робота з файлами: Якщо веб-додаток некоректно обробляє імена файлів, передані ззовні (наприклад, через GET або POST запити), це може призвести до виконання небезпечних дій, таких як незаконне завантаження чутливих файлів або внедрення шкідливого коду.
5. Некоректна робота з паролями: Веб-додатки повинні коректно обробляти, зберігати та передавати паролі користувачів, щоб уникнути несанкціонованого доступу до облікових записів.
6. Невідповідність прав доступу: Неправильно налаштовані права доступу до файлів, неправильні права програм на сервері, недостатні або неправильні права, надані веб-додаткам або програмам на веб-сервері, можуть спричинити ризик злому системи або несанкціонованого доступу до ресурсів.

7. Некоректна робота з базами даних: Недостатня обробка вхідних даних або некоректні запити до баз даних можуть призвести до вразливостей, таких як SQL-ін'єкції, де зловмисник може виконувати шкідливі команди бази даних або отримувати конфіденційну інформацію.
8. Неоптимізований програмний код: Поганий або неоптимізований програмний код може призводити до перевантаження веб-сервера та зниження продуктивності, особливо при обробці некоректних вхідних даних або атаках.
9. Вразливість до DoS та DDoS атак: Веб-додатки та системи можуть бути вразливими до атак типу DoS (зав'язування послуги) та DDoS (розподілене зав'язування послуги), де зловмисники спрямовують велику кількість запитів на веб-додаток з метою перевантаження його та відмови у наданні послуг користувачам.

Врахування цих вразливостей та застосування відповідних заходів безпеки дуже важливі для забезпечення безпеки та захищеності веб-додатків.

У сучасних веб-додатках зберігання даних зазвичай здійснюється на сервері, а взаємодія з користувачем та передача інформації відбувається через мережу. У такому середовищі можуть виникати різні помилки та проблеми, і важко визначити точне місце їх виникнення.

Наприклад, помилка може бути спричинена проблемами з мережевим з'єднанням, некоректною обробкою даних на клієнтському браузері або на сервері, помилками в мережевих протоколах або конфігурації сервера, а також проблемами в самому веб-додатку.

Коли користувач спостерігає помилку в мережевому середовищі, важко визначити її точну причину та місце виникнення. Повідомлення про помилку, яке користувач отримує, може бути загальним повідомленням про помилку або статусом HTTP, який надає загальну інформацію про помилку, але не завжди деталізовану інформацію про причину помилки.

У таких випадках для виявлення та виправлення помилок може бути необхідно проводити аналіз логів, спостереження за мережевим трафіком та використання інших інструментів для діагностики проблеми. Це може вимагати співпраці між різними командами, такими як розробники веб-додатків, мережеві адміністратори та інші фахівці.

Тестування веб-орієнтованих додатків включає деякі особливості, які відрізняються від тестування класичних програм. Основні особливості тестування веб-додатків пов'язані з їх середовищем функціонування, компонентною структурою, технологічними особливостями та взаємодією з мережею та багатьма користувачами. Деякі з цих особливостей включають:

1. Режими роботи: Веб-додатки можуть працювати в різних режимах, таких як розробка, тестування, виробництво тощо. Кожен режим має свої особливості, що потребують окремого тестування.
2. Інсталяція, запуск, зупинка та видалення: Веб-додатки можуть мати складний процес інсталяції та конфігурації, а також специфічні процедури запуску, зупинки та видалення. Ці процеси також потребують тестування.
3. Формування інтерфейсів: Веб-додатки взаємодіють з користувачем через веб-інтерфейси, які можуть бути складними з точки зору дизайну, функціональності та взаємодії з користувачем. Тестування таких інтерфейсів включає перевірку сумісності з різними браузерами, адаптивність до різних пристроїв та перевірку коректності відображення.
4. Права доступу: Веб-додатки можуть мати різні рівні доступу для різних користувачів, наприклад, адміністраторів, модераторів або звичайних користувачів. Тестування включає перевірку коректності реалізації прав доступу та перевірку безпеки даних.

Технологічні відмінності між класичними додатками і веб-додатками є суттєвими. Основні відмінності технологічного характеру включають:

1. Клієнт-серверна архітектура: Веб-додатки базуються на клієнт-серверній архітектурі, де браузер виступає як клієнт, а веб-сервер обробляє запити і надсилає відповіді. Це відрізняється від класичних додатків, які можуть працювати самостійно на одному комп'ютері без використання мережі.
2. Використання веб-протоколів: Веб-додатки взаємодіють з клієнтами за допомогою стандартних веб-протоколів, таких як HTTP і HTTPS. Це відрізняється від класичних додатків, які можуть використовувати інші протоколи, такі як TCP/IP або UDP.
3. Мови програмування та фреймворки: Веб-додатки можуть використовувати різні мови програмування, такі як JavaScript, Python, Ruby, PHP тощо, а також фреймворки, спеціально розроблені для розробки веб-додатків, наприклад, Django, Ruby on Rails, Laravel, Angular, React тощо. Класичні додатки, зазвичай, використовують мови програмування, які найкраще відповідають їхнім потребам.
4. Використання веб-сервісів: Веб-додатки можуть взаємодіяти з іншими додатками та сервісами шляхом використання веб-сервісів, таких як REST або SOAP. Це дозволяє розширювати функціональність веб-додатків і підключати різні сервіси та джерела даних.
5. Безпека: веб-додатки мають свої особливості стосовно безпеки. Структурні відмінності між класичними додатками і веб-додатками впливають на процес тестування.

Основні структурні відмінності включають:

1. Монолітна структура проти багатокомпонентної структури: класичні додатки зазвичай мають монолітну структуру, де весь код знаходиться в одному або декількох модулях. У веб-додатках використовується

багатокомпонентна структура, де функціональність розподіляється між багатьма модулями, такими як сервер баз даних, веб-сервер, сервер додатків тощо. Тестування веб-додатків вимагає перевірки взаємодії між цими компонентами.

2. Використання серверів баз даних: Веб-додатки зазвичай використовують сервери баз даних для зберігання і керування даними. Тестування веб-додатків включає перевірку коректності взаємодії з сервером баз даних, валідацію запитів і перевірку правильності зберігання даних.
3. Використання веб-серверів та серверів додатків: веб-додатки використовують веб-сервери для обробки HTTP-запитів та сервери додатків для виконання програмного коду. Тестування веб-додатків включає перевірку правильності конфігурації та роботи веб-серверів і серверів додатків.
4. Клієнтська і серверна частини: веб-додатки включають клієнтську та серверну частини. Тестування веб-додатків вимагає перевірки як на клієнтській, так і на серверній стороні, зокрема, перевірку коректності відображення інтерфейсу, валідацію введених даних, перевірку правильності обробки запитів, відмінності в режимах роботи класичних додатків і веб-додатків впливають на підхід до тестування.

Основні відмінності режимів роботи включають:

1. Режим реального часу: класичні додатки зазвичай працюють в режимі реального часу, що означає, що вони негайно реагують на дії користувача і надають результати або відображення змін у режимі майже миттєво. При тестуванні класичних додатків важливо перевірити, чи відбувається реагування відповідно до дій користувача і чи відображаються зміни в реальному часі.

2. Режим "запит-відповідь": веб-додатки працюють в режимі "запит-відповідь", де клієнт (браузер) надсилає запити на сервер, а сервер обробляє запити і надсилає відповіді назад. Це означає, що веб-додатки повідомляють про результати та зміни після того, як був зроблений відповідний запит. При тестуванні веб-додатків важливо перевірити, чи відбувається правильна обробка запитів і надсилання відповідей, а також чи правильно оновлюються дані та відображаються зміни після кожного запиту.
3. Асинхронність: веб-додатки можуть використовувати асинхронні запити та відповіді, що дозволяє здійснювати взаємодію з сервером без необхідності оновлювати всю сторінку. Це може вплинути на тестування, оскільки потрібно перевіряти правильну обробку асинхронних запитів та забезпечувати синхронність формування інтерфейсу відрізняється у класичних додатках і веб-додатках.

Основні відмінності включають:

1. Усталені і стандартизовані технології: класичні додатки зазвичай використовують усталені та стандартизовані технології для формування інтерфейсу користувача. Це можуть бути технології, такі як графічні бібліотеки або інтерфейси користувача, які підтримуються платформою, на якій працює додаток (наприклад, Windows API або Cocoa для мобільних додатків). Тестування таких додатків вимагає перевірки відповідності інтерфейсу встановленим стандартам та очікуванням користувачів.
2. Технології, що розвиваються: веб-додатки використовують технології, які швидко розвиваються та змінюються. Це можуть бути технології веб-розробки, такі як HTML, CSS, JavaScript та фреймворки і бібліотеки, що забезпечують інтерактивність і стиль інтерфейсу. У світі веб-розробки постійно з'являються нові технології та рішення, і веб-додатки можуть

використовувати різні комбінації цих технологій. Тестування веб-додатків вимагає оцінки сумісності та коректності відображення інтерфейсу на різних браузерах та пристроях, а також перевірки відповідності сучасним трендам та очікуванням користувачів.

3. Конкуренція між технологіями: веб-додатки можуть використовувати різні технології для формування інтерфейсу, існує безліч конкуренції.

1.3. Характеристика сучасних систем автоматизованого тестування веб-додатків

Object Driven Testing (тести, керовані об'єктами) використовується для автоматизації тестування, де тестові скрипти проектуються у вигляді класів, що містять логіку роботи з додатком. Цей підхід спрощує процес створення та підтримки тестових скриптів, оскільки використовуються лише методи "високого рівня", що дозволяють приховати деталі реалізації конкретних дій.

У програмному засобі TestComplete, є спеціальний елемент проекту ODT (Object-Driven Testing), який надає можливість представляти тестові скрипти у вигляді класів з властивостями і методами. Це дозволяє організувати тестові скрипти в структурованій формі і зручно управляти ними.

Варто зазначити, що в мовах JScript і VBScript, які використовуються в TestComplete, ви можете створювати класи, користуючись вбудованими можливостями цих мов. Це означає, що можна створювати власні класи, методи та властивості для реалізації потрібної логіки тестування в рамках мови програмування, яку ви обрали.

Застосування підходу Object Driven Testing допомагає зробити автоматизоване тестування більш структурованим, гнучким та ефективним, забезпечуючи легкість управління тестовими скриптами та їх підтримку

JMeter є популярним інструментом для тестування продуктивності і функціональності різних типів додатків, включаючи веб-додатки, файлові сервери, веб-сервери і бази даних. Основні можливості JMeter включають:

1. Тестування продуктивності: JMeter дозволяє створювати симульоване навантаження на веб-додаток, що дозволяє вимірювати його продуктивність. Ви можете налаштувати JMeter так, щоб він симулював N-ну кількість користувачів і потоків, які відвідують ваш додаток, і вимірювати час відгуку, пропускну здатність та інші метрики продуктивності.
2. Функціональне тестування: крім тестування продуктивності, JMeter також може використовуватись для функціонального тестування додатків. Ви можете створювати тестові сценарії з різними діями та перевіряти правильність відповідей додатку. JMeter також підтримує твердження, які дозволяють автоматично перевіряти очікувані результати тесту.
3. Аналіз результатів: JMeter надає можливість аналізувати результати тестування в графічному та статистичному вигляді. Ви можете переглядати графіки продуктивності, розподіл часу відгуку, розмір відповідей і багато іншого. Це дозволяє отримати усереднений результат і зробити висновки про продуктивність і функціональність вашого додатку.

Загалом, JMeter є потужним інструментом для тестування веб-додатків і надає широкий набір функцій для аналізу продуктивності та функціональності додатків в підході Object Driven Testing (ODT) структура даних розділена на класи і дані. Класи містять властивості і методи, які використовуються для реалізації логіки тестування. Властивості можуть бути звичайними змінними або масивами, що зберігають дані, необхідні для тестування. Методи, в свою

чергу, відповідають функціям, які виконують певні операції над даними або виконують тестові дії.

У ODT дані можуть містити звичайні змінні, масиви та об'єкти класів. Об'єкти класу можуть мати власні методи, які будуть доступні лише для цього конкретного об'єкта, а не для інших об'єктів того ж класу. Це означає, що можна визначити спеціальні методи для конкретних об'єктів, які використовуються лише в їх контексті.

Ця особливість дозволяє гнучко налаштовувати тестові скрипти і реагувати на конкретні особливості окремих об'єктів. Наприклад, можна створити додаткові методи для обробки певних дій або станів лише в певних об'єктах, що полегшує роботу з різними варіаціями тестових сценаріїв.

Ця можливість дозволяє зберігати код тестових скриптів більш чистим і структурованим, а також сприяє повторному використанню коду, оскільки методи можна перевикористовувати для конкретних об'єктів у межах класу.

1.4. Висновки до розділу 1

У цьому розділі були розкриті основні поняття про веб-додатки, їх сутність та особливості. Виявлено принципи тестування веб-додатків та наведено принципові відмінності від тестування класичних програмних додатків.

Описано основні програми для тестування веб-додатків, зокрема JMeter і TestComplete, які можуть використовуватись для автоматизованого тестування веб-додатків.

Наступним кроком є розгляд методологічних аспектів автоматизованого тестування веб-додатків. Це означає вивчення різних підходів та методик, які можуть бути використані для планування, розробки і виконання тестів веб-додатків. Такі методології можуть включати в себе розробку тестових

сценаріїв, використання тестових фреймворків, аналіз результатів тестування та інші етапи тестового циклу.

Детальне вивчення методологічних аспектів автоматизованого тестування веб-додатків допоможе покращити ефективність тестування, забезпечити надійність і якість веб-додатків та сприяти розробці більш ефективних тестових процесів.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ПРИНЦИПІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

2.1. Цільовий аналіз системи автоматизованого тестування веб-додатків

Для розробки тестів веб-додатків слід звернути особливу увагу на методи, що реалізують критерії функціонального тестування. Функціональне тестування зосереджується на перевірці, чи виконується очікувана функціональність додатку, як відповідає на коректні та некоректні вхідні дані, і як взаємодіє з іншими компонентами системи. Основні методи функціонального тестування включають:

1. Тестування чорного ящика (black-box testing): при цьому методі тестування тестувальник не має доступу до внутрішньої структури додатку і концентрується на зовнішньому поведінці системи.
2. Тестування білого ящика (white-box testing): цей метод включає аналіз внутрішньої структури додатку, його коду та алгоритмів. Тестувальник створює тести на основі цієї внутрішньої інформації.
3. Тестування одиниць програмного забезпечення (unit testing): воно спрямоване на тестування окремих компонентів додатку, таких як функції, класи або модулі. Тестування одиниць зазвичай виконується розробниками програмного забезпечення.

Крім того, для розробки тестів веб-додатків важливо враховувати модель предметної області розробки. Це означає вивчення основних

компонентів та функцій додатку, розуміння взаємодії між ними і визначення основних шляхів тестування. Модель предметної області можна краще зрозуміти за допомогою вивчення функціональності та особливості додатку який сприятиме більш ефективному розробленню тестових сценаріїв.

Діаграми функціональності (functional diagrams) і попарне тестування (pairwise testing) є корисними методами для розробки тестів і перевірки функціональності веб-додатків.

1. Діаграми функціональності (functional diagrams): діаграми функціональності - це графічні представлення функціональності системи, які допомагають зрозуміти і візуалізувати логіку тестування. Вони можуть бути у формі блок-схем, графіків потоку даних або інших графічних зображень. Діаграми функціональності допомагають ідентифікувати основні функції додатку, його компоненти та взаємодію між ними. Вони можуть використовуватися для розробки тестових сценаріїв і визначення тестових наборів.
2. Попарне тестування (pairwise testing): попарне тестування – це метод, який дозволяє зменшити кількість тестових наборів, при цьому забезпечуючи високу покриття комбінацій параметрів. Ідея полягає в тому, щоб вибрати представники з усіх можливих комбінацій параметрів, таким чином зменшуючи кількість тестів без втрати значущості. При попарному тестуванні використовуються статистичні алгоритми для вибору оптимальних комбінацій. Цей метод допомагає ефективно тестувати різноманітні комбінації параметрів, що зазвичай приводить до виявлення більшої кількості дефектів.

Існує цілий ряд комбінаторних стратегій, які розроблені з метою допомогти тестувальникам вибрати підмножину вхідних комбінацій для максимального виявлення дефектів:

1. Вибіркове тестування (Selective Testing): цей підхід передбачає вибір конкретних комбінацій параметрів для тестування, заснований на аналізі ризиків і пріоритетів. Тестувальник обирає найбільш важливі та ризиковані комбінації для перевірки.
2. "Кожен-вибір" (Each-Choice): цей підхід полягає в тому, щоб включити кожну можливу комбінацію значень параметрів хоча б один раз у тестовий набір. Це допомагає забезпечити покриття всіх комбінацій параметрів і можливих взаємодій між ними.
3. "Підстава вибору" (Base Choice): в цьому підході тестувальник обирає базові комбінації параметрів, які представляють основні функціональні можливості системи. Інші комбінації генеруються шляхом внесення змін до базових комбінацій. Це допомагає зменшити кількість тестів, але все ще забезпечує покриття важливих варіацій параметрів.
4. Антирандомізація (Antirandom): цей метод вибирає комбінації параметрів, які є протилежними до випадкових комбінацій. Ідея полягає в тому, щоб уникнути можливості, що усі значення будуть випадковими, і зосередитися на специфічних комбінаціях, які можуть призвести до дефектів.
5. Стратегії тестування t-способами (t-Way Testing Strategies).

Обидва методи, діаграми функціональності і попарне тестування, можуть бути використані разом для розробки комплексних тестових сценаріїв і забезпечення високої якості тестування функціональності веб-додатків.

TypeScript – мультипарадигмальна, обєктно-орієнтована мова програмування, яка є зворотно сумісною з JavaScript і компілюється в останній.

Метод парного тестування (pairwise testing) базується на ідеї алгоритму всіх пар (All-Pairs Algorithm), який є комбінаторним підходом, спеціально розробленим для парного тестування.

Основна ідея методу парного тестування полягає в тому, що більшість помилок виявляються тільки при певних комбінаціях значень вхідних параметрів. Замість того, щоб тестувати всі можливі комбінації, метод парного тестування вибирає комбінації, в яких кожна пара параметрів з'являється разом принаймні один раз. Це дозволяє зменшити кількість тестових наборів, порівняно з випадковим вибором комбінацій.

Алгоритм всіх пар генерує комбінації значень параметрів таким чином, щоб кожна пара параметрів з'являлася разом принаймні один раз. Це досягається шляхом вибору комбінацій, в яких кожен параметр має всі можливі значення, але змінюється лише один параметр одночасно, тоді як інші залишаються незмінними.

Важливо зазначити, що число комбінацій, що генеруються методом парного тестування, буде значно менше, ніж при використанні ортогональних масивів (orthogonal arrays). Це дозволяє ефективно використовувати обмежені ресурси тестування та зменшити час, потрібний для виконання тестів, при збереженні високої покриття комбінацій параметрів.

Метод парного тестування є популярним підходом для ефективного тестування веб-додатків, оскільки вони зазвичай мають багато параметрів, і перевірка всіх можливих комбінацій може бути непростою.

2.2. Дослідження алгоритмів реалізації системи автоматизованого тестування веб-додатків

Поетапний алгоритм реалізації системи автоматизованого тестування веб-додатків може включати наступні пункти:

Аналіз вимог: ретельно проаналізувати вимоги до веб-додатку, які включають функціональні та нефункціональні вимоги. Розуміння вимог допоможе створити ефективну стратегію тестування.

Планування тестування: розробити план тестування, в якому визначте обсяг робіт, враховуючи ресурси, графік і пріоритети. Визначити види тестів, які потрібно провести, такі як функціональні, інтеграційні, навантажувальні тести тощо.

Вибір інструментів: вибрати підходящі інструменти для автоматизованого тестування веб-додатків, такі як фреймворки для написання тестів, інструменти для створення тестових наборів, На основі вимог до інформаційного забезпечення, які були визначені раніше, можна розробити моделі баз даних для програмного забезпечення керування замовленнями в закладі швидкого харчування.

Створення тестових сценаріїв: Розробити тестові сценарії, які включають послідовність кроків для перевірки різних функціональних аспектів веб-додатку. Використавши знання про додаток і його вимоги для створення ефективних тестів.

Налаштування тестового середовища: Підготувати тестове середовище, яке включає налаштування серверів, баз даних, налаштування тестових данних та інших залежностей. Це дозволить виконувати тести в контрольованому середовищі.

Реалізація автоматизованих тестів: Використовуючи обрані інструменти для автоматизованого тестування, реалізувати тестові сценарії.

Аналіз вимог є критичним кроком у розробці системи автоматизованого тестування веб-додатків. Цей процес включає ретельне проаналізування функціональних та нефункціональних вимог до додатку. Наведені деякі кроки, які можна виконати під час аналізу вимог:

- Зберегти всі документи з вимогами, такі як технічне завдання, специфікації, описи функцій тощо.

- Розібратися зі структурою додатку: Розуміння структури додатку, його компонентів і модулів допоможе зрозуміти, як взаємодіють різні частини системи і які функціональні області варто тестувати окремо.
- Визначити функціональні вимоги: Ідентифікувати всі функції, які має виконувати додаток. Розбити їх на окремі складові, щоб мати чітке уявлення про функціональні області, які потрібно перевірити.
- Розглянути нефункціональні вимоги: Окрім функціональних вимог, звернути увагу на нефункціональні вимоги, такі як продуктивність, надійність, безпека, сумісність з браузерами і платформами тощо. Визначити, які аспекти нефункціональності важливі для додатку і вимагають спеціальної уваги під час тестування.
- Встановити пріоритети: Визначити пріоритети тестування на основі важливості функцій та ризиків. Визначити, які функції потрібно перевірити в першу чергу і які можуть бути нижчим пріоритетом.
- Створити стратегію тестування: На основі аналізу вимог розробити ефективну стратегію тестування.

Планування тестування є важливим етапом у розробці системи автоматизованого тестування веб-додатків. Ось деякі кроки, які можна виконати під час планування тестування:

- Визначення обсягу робіт: Розглянути всі функціональні та нефункціональні вимоги і визначте обсяг робіт, який потрібно виконати. Врахувати рівень докладності тестування, необхідний для кожної функції, та складність додатку в цілому.
- Встановлення пріоритетів: Визначити пріоритети для різних функцій та компонентів додатку на основі їх важливості та критичності. Важливо перевірити критичні функції та області, які можуть мати найбільший вплив на користувачів або на безпеку додатку.

- Визначення видів тестів: Врахувати різні види тестів, які потрібно провести. Це можуть бути функціональні тести, інтеграційні тести, системні тести, навантажувальні тести, безпекові тести тощо. Вибрати ті тести, які найбільш підходять для вашого додатку і вимог.
- Визначення тестових сценаріїв: Розробити тестові сценарії для кожного виду тестів, включаючи послідовність дій та очікувані результати. Поміркувати про різні сценарії взаємодії з додатком, різні шляхи навігації та можливі варіанти введення даних.
- Розподіл ресурсів: Оцінити необхідні ресурси для виконання тестування, такі як людські ресурси, час, обладнання, програмне забезпечення.

При автоматизованому тестуванні веб-додатків існує багато інструментів і фреймворків, які можуть бути використані. Вибір конкретних інструментів залежить від ваших потреб і вимог проекту. Ось кілька популярних інструментів, які можна розглянути:

Фреймворки для написання тестів:

- Selenium: Відкритий фреймворк для автоматизації веб-додатків. Підтримує різні мови програмування, такі як Java, Python, C#, і дозволяє створювати тестові сценарії для різних браузерів.
- Cypress: Модерний фреймворк для автоматизації веб-додатків. Забезпечує простоту використання та широкі можливості для написання тестів. Працює тільки з браузером Chrome.

Інструменти для створення тестових наборів:

- TestNG: Розширення для фреймворку JUnit, яке дозволяє створювати структуровані тестові набори, виконувати параметризовані тести та забезпечувати більшу гнучкість управління тестами.

- pytest: Фреймворк для тестування, який пропонує простий синтаксис і широкі можливості для структурування тестів, параметризації і фікстур.

Засоби для управління тестовими сценаріями:

- TestRail: Інструмент для керування тестовою документацією, планування тестів та відстежування результатів тестування.
- Jira: Популярна система керування проектами, яка може використовуватись для створення та відстежування задач тестування, управління багами та спільної роботи команди.

Інші інструменти:

- Postman: - це інструмент для тестування API, який дозволяє створювати, відправляти та перевіряти запити до веб-сервісів. Він надає гнучкі можливості для автоматизованого тестування API, включаючи створення тестових сценаріїв, валідацію відповідей, збір результатів тестування та генерацію звітів. За допомогою Postman можна:

Створювати запити: можна створювати HTTP-запити, включаючи GET, POST, PUT, DELETE тощо. Ви можете налаштовувати заголовки, параметри запиту та тіло запиту з використанням різних форматів даних, таких як JSON, XML і т.д.

Створювати тестові сценарії: можна створювати тестові сценарії для автоматизованої перевірки відповідей веб-сервісу. Наприклад, ви можете перевірити код статусу відповіді, наявність або значення певних полів у відповіді.

Збирати результати тестування: Postman надає можливість збирати результати тестування, включаючи інформацію про пройдені та невдачні тести. Можна переглядати та аналізувати результати тестів безпосередньо в інтерфейсі Postman.

Генерувати звіти: Postman дозволяє генерувати звіти про тестування, які можуть бути експортовані у різних форматах, таких як HTML або JSON. Це дозволяє документувати результати тестування та спільно користуватись ними з іншими членами команди.

Postman є потужним інструментом для тестування API, який забезпечує швидкість, легкість використання та багатофункціональність для розробки та виконання автоматизованих тестів API.

Наведено кілька прикладів тестових сценаріїв для перевірки різних функціональних аспектів веб-додатку:

1. Реєстрація нового користувача:
   - Крок 1: Відкрити додаток і перейти до сторінки реєстрації.
   - Крок 2: Ввести обов'язкові поля (наприклад, ім'я, електронну пошту, пароль).
   - Крок 3: Надіслати форму реєстрації.
   - Крок 4: Перевірити, чи відображається повідомлення про успішну реєстрацію.
   - Крок 5: Залогінитися використовуючи введені при реєстрації дані.
   - Крок 6: Перевірити, чи відображається ім'я користувача у розділі профілю.
2. Додавання елемента до кошика:
   - Крок 1: Увійти в обліковий запис користувача.
   - Крок 2: Перейти до сторінки з каталогом товарів.
   - Крок 3: Вибрати товар і додати його до кошика.
   - Крок 4: Перейти до кошика і перевірити, чи відображається обраний товар.
   - Крок 5: Змінити кількість товару в кошику і перевірити, чи змінюється загальна вартість.

- Крок 6: Видалити товар з кошика і перевірити, чи зник відповідний запис у кошику.

3. Пошук і фільтрація товарів:

- Крок 1: Відкрити сторінку пошуку товарів.
- Крок 2: Ввести ключове слово для пошуку.
- Крок 3: Натиснути кнопку пошуку і перевірити, чи відображаються відповідні результати.
- Крок 4: Використовуючи фільтри (наприклад, ціновий діапазон, категорія), обмежити результати пошуку і перевірити, чи змінюються вони відповідно.

Оцінка ресурсів для виконання тестування може варіюватися в залежності від конкретних вимог і обсягу проекту. Однак, основні ресурси, які необхідно враховувати, включають:

1. Людські ресурси:

- Тестувальники: Визначити кількість тестувальників, які потрібні для виконання тестування. Це може залежати від складності проекту, обсягу робіт і термінів.
- Тестовий лідер або менеджер тестування: Виконання плану тестування, розподіл завдань, взаємодія з командою розробників і звітування.
- Бізнес-аналітики: Допомога у розумінні вимог і сприяння у створенні ефективних тестових сценаріїв.
- Власник продукту або представник клієнта: Забезпечення правильності вимог і участь у тестуванні з точки зору користувача.

2. Час:

- Визначити графік тестування, у якому враховано час на підготовку тестових сценаріїв, виконання тестів, аналіз результатів та виправлення виявлених дефектів.

- Забезпечити достатній час для повного покриття функціональних і нефункціональних вимог, а також для проведення різних типів тестів (функціональні, інтеграційні, навантажувальні тощо).

3. Обладнання:
   - Врахувати необхідне обладнання для виконання тестування, таке як комп'ютери, мобільні пристрої, сервери тощо.
   - Забезпечити необхідне середовище тестування, включаючи встановлення тестового середовища, бази даних, серверів тощо.
4. Програмне забезпечення:
   - Вибрати необхідні інструменти.

2.3. Дослідження інструментарію для розробки та удосконалення системи автоматизованого тестування веб-додатків

Важливо, щоб інструмент автоматизованого тестування розпізнавав елементи управління в додатку, оскільки це визначає його здатність ефективно взаємодіяти зі сторінками та функціями додатку. Якщо інструмент не може правильно розпізнати елементи, це може призвести до проблем з написанням і підтримкою скриптів автоматизованого тестування.

Ось кілька критеріїв, на які варто звернути увагу при виборі інструменту для автоматизованого тестування:

1. Розпізнавання елементів управління: Переконайтеся, що інструмент має добре розроблені засоби для розпізнавання різних типів елементів управління, таких як кнопки, поля введення, випадаючі списки тощо. Це дозволить легко ідентифікувати ці елементи під час запису та відтворення тестів.
2. Підтримка різних платформ та технологій: Переконайтеся, що інструмент підтримує технології, на яких базується ваш веб-додаток. Наприклад, якщо ваш додаток побудований з використанням JavaScript

та фреймворку React, переконайтеся, що інструмент може ефективно взаємодіяти з такими технологіями.

3. Наявність плагінів та розширень: Перевірте, чи існують плагіни або розширення для інструменту, які можуть додатково покращити його функціональність та підтримку. Це може включати плагіни для конкретних фреймворків або інструменти для роботи з базами даних.
4. Зручність використання та навчання: Оцініть, наскільки легко розуміти інтерфейс.

Зручність використання та навчання є важливими критеріями при виборі інструменту для автоматизованого тестування. Оцінка зручності використання включає наступні аспекти:

1. Інтерфейс користувача: Інструмент повинен мати інтуїтивно зрозумілий та зручний інтерфейс користувача, що дозволяє легко взаємодіяти з ним. Навігація та розташування функцій повинні бути логічними та зрозумілими.
2. Документація та підтримка: Інструмент повинен мати достатньо документації, яка надає детальну інформацію про його функціональність та використання. Наявність довідкових матеріалів, посібників або форумів підтримки допоможе користувачам швидше освоїти інструмент і вирішити можливі проблеми.
3. Легкість навчання: Важливо, щоб інструмент був легким у навчанні. Інструмент з інтуїтивним інтерфейсом та простими у використанні функціями дозволить тестувальникам швидко оволодіти ним і почати створювати автоматизовані тести.
4. Ємність спільноти користувачів: Якщо інструмент має активну спільноту користувачів, це може бути великою перевагою. Спільнота може надати підтримку, допомогти вирішити проблеми та поділитися корисними порадами та прикладами.

Узагальнюючи, важливо знайти інструмент, який має інтуїтивний інтерфейс, хорошу документацію, легкий у навчанні та підтримується активною спільнотою користувачів.

Врахування часу, необхідного для підтримки тестових скриптів, є важливим критерієм при виборі інструменту для автоматизованого тестування. Якщо скрипти написані з використанням обраного інструменту вимагають значних зусиль для зміни або підтримки, це може бути недоцільним у виробничому середовищі.

Один із способів оцінити цю зручність полягає в перевірці, наскільки легко можна змінити елементи інтерфейсу в скриптах. Як ви вказали, уявіть, що пункт меню змінився, і спробуйте замінити його назву в скрипті. Якщо заміна вимагає значних зусиль, включаючи переписування скрипта цілком, це може бути неефективним.

Найкращим варіантом є інструмент, який дозволяє використовувати змінні для ідентифікації елементів у скриптах. Замість жорсткої фіксації назв елементів інтерфейсу, можна винести їх у змінні, що дозволить швидко змінювати значення цих змінних і оновлювати скрипти. Це зменшить час і зусилля, необхідні для підтримки скриптів при змінах в інтерфейсі додатку.

Отже, при виборі інструменту для автоматизованого тестування варто звернути увагу на гнучкість та легкість зміни скриптів, зокрема на наявність можливості використання змінних для ідентифікації елементів у скриптах. Це дозволить ефективно підтримувати скрипти і швидко адаптуватись до змін в додатку.

Зручність інструмента для написання нових скриптів є ще одним важливим аспектом при виборі автоматизованого тестування. Наступні фактори варто враховувати при оцінці зручності інструмента:

1. **Час на написання скриптів**: Інструмент повинен забезпечувати швидкий та ефективний процес написання нових скриптів. Це може

включати зручний синтаксис, автодоповнення, шаблони скриптів та інші функції, що спрощують процес написання коду.

2. **Структурування коду**: Інструменти, які підтримують об'єктно-орієнтоване програмування (ООП), можуть забезпечити більшу структурованість та повторне використання коду. Це дозволяє розбити скрипти на класи та функції, що полегшує розуміння та підтримку коду.
3. **Читабельність коду**: Інструмент повинен сприяти написанню читабельного коду, що легко зрозуміти та підтримувати. Це може включати чітке форматування, правила іменування змінних та функцій, коментарі і документацію коду.
4. **Середовище розробки для рефакторинга**: Інструменти, які підтримують рефакторинг (переробку коду), спрощують процес внесення змін в скрипти, зберігаючи при цьому їх функціональність. Це може включати функції, такі як автоматичне перейменування, виділення методів, оптимізація імпортів і т.п.

При виборі інструменту для автоматизованого тестування варто оцінювати, наскільки зручно ним користуватись для написання нових скриптів, зокрема швидкість та зручність.

Інструмент для автоматизованого тестування є програмним забезпеченням, призначеним для автоматизації процесу тестування програмного забезпечення. Він дозволяє створювати тест-скрипти, які включають набори інструкцій для автоматичної перевірки певної частини програми. Ці скрипти можуть бути налагоджені, виконані і аналізовані за допомогою інструменту для автоматизованого тестування.

Такі інструменти надають різноманітні функції, такі як запис і відтворення дій користувача, генерація тестових даних, перевірка очікуваного результату, звітність про результати тестування і багато іншого. Вони забезпечують ефективність і надійність процесу тестування шляхом

автоматизації повторюваних завдань і швидкого виявлення проблем у програмному забезпеченні.

Найпоширеніші приклади інструментів для автоматизованого тестування включають Selenium, Appium, JUnit, TestNG, Cucumber, Robot Framework та інші. Кожен з цих інструментів має свої особливості і може бути використаний для різних типів тестування, таких як веб-додатки, мобільні додатки, API і т. д.

Застосування інструментів для автоматизованого тестування сприяє покращенню якості програмного забезпечення, скороченню часу тестування і зниженню його вартості.

Звичайно, давайте коротко розглянемо кожен з цих інструментів для автоматичного тестування:

1. Selenium WebDriver: Selenium WebDriver є одним з найпопулярніших інструментів для автоматизованого тестування веб-додатків. Він надає API для взаємодії з веб-елементами і виконання дій, таких як введення тексту, клік, наведення курсора і т. д. Використовуючи Selenium WebDriver, можна створювати тестові скрипти на різних мовах програмування, таких як Java, Python, C#, і запускати їх на різних браузерах.
2. Maven: Maven є інструментом для управління проектами в Java, включаючи проекти для автоматичного тестування. Він надає структуру проекту, залежності, плагіни та інші інструменти для збирання, виконання та управління проектом. За допомогою Maven можна легко налаштовувати залежності для Selenium WebDriver, TestNG і інших бібліотек, що використовуються в автоматичному тестуванні, і забезпечувати повторне використання коду та ефективне управління залежностями.

3. Intellij IDEA: Intellij IDEA є інтегрованою середовищем розробки (IDE), яка надає потужні можливості для написання, відлагодження і запуску тестових скриптів. Вона підтримує різні мови програмування, включаючи Java, і надає розширення та плагіни для автоматичного тестування, такі як плагін TestNG. Intellij IDEA має багато корисних функцій, таких як автодоповнення коду, рефакторинг, інструменти для відлагодження та інші, що полегшують процес написання та управління тестовим кодом.
4. TestNG: TestNG є фреймворком для тестування, який надає розширені можливості.

2.4. Висновки до розділу 2

Ми плануємо провести практичне дослідження та розробку системи автоматизованого тестування веб-додатків. Використання інструментів PICT і All-Pairs для генерації тестових наборів може бути корисним підходом для покриття різноманітних комбінацій параметрів та значень у вашій системі.

Перш ніж розпочати розробку, важливо планувати і проектувати систему автоматизованого тестування:

1. Визначити обсяг та мету нашої системи автоматизованого тестування. Які функції та аспекти веб-додатку ми плануємо перевіряти, які вимоги до якості ми хочемо задовольнити.
2. Вибір підходу до автоматизованого тестування. Ми можемо використовувати запис і відтворення дій користувача, написання скриптів на мові програмування або використання фреймворків, таких як Behavior Driven Development (BDD).
3. Розроблення архітектури системи автоматизованого тестування. Розділімо наші тести на окремі модулі або компоненти, що дозволить легко управляти тестами та забезпечити їх повторне використання.

4. Написання тестових скриптів для автоматизованого тестування веб-додатків. Використання вибраного інструментарю, такого як Selenium WebDriver, Maven та TestNG, для створення, виконання та аналізу результатів тестів.
5. Виконання тестування нашої системи автоматизованого тестування на реальних веб-додатках. Зібрання даних про результати тестування, виявлені дефекти та інші метрики, щоб оцінити ефективність системи.

РОЗДІЛ 3
ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

3.1. Проектування системи автоматизованого тестування веб-додатків

Для розробки тестів реєстрації користувачів сайту з використанням вказаних значень параметрів, визначимо наступні етапи:

1. Визначення списку тестових сценаріїв: Створення списку сценаріїв реєстрації користувачів, в яких будуть перевірятися різні комбінації значень параметрів. Наприклад:
   - Тест реєстрації з правильними значеннями всіх параметрів.
   - Тест реєстрації з неправильним значенням логіну.
   - Тест реєстрації з неправильним значенням паролю.
   - Тест реєстрації з неправильним значенням адреси електронної пошти.
   - Тест реєстрації з неправильним значенням прапорця згоди.
2. Написання тестових скриптів: Використання обраного інструментарію для автоматизованого тестування (наприклад, Selenium WebDriver, TestNG) для написання тестових скриптів для кожного сценарію. У тестових скриптах ми можемо встановлювати значення параметрів реєстрації, заповнювати форму реєстрації, натискати кнопку "Зареєструватися" і перевіряти очікуваний результат.

1. Налагодження і виконання тестів: Запуск тестових скриптів і перевірка, що вони виконуються без помилок. При необхідності виправлення помилок та додаткові налаштування тестових сценаріїв.
3. Аналіз результатів: Після виконання тестів аналіз їх результатів. Перевірка, чи були успішно пройдені тести з правильними значеннями параметрів і чи були виявлені помилки в тестах з неправильними значеннями. За допомогою звітів та журналів тестування підготовка висновків про якість реєстрації.

Можна розробити програму, яка використовує рекурсивний алгоритм для обчислення всіх можливих комбінацій значень параметрів і формує таблицю рішень. Для цього можна використати структуру ступінчастого масиву для представлення діаграми (булевого графа). Ось загальна структура програми:

1. Визначення параметрів реєстрації: Визначення параметри, таких як логін, пароль, адреса електронної пошти та прапорець згоди. Визначення допустимих значень для кожного параметра (правильне та неправильне значення).
2. Створення функції для генерації комбінацій: Написання рекурсивної функції, яка буде генерувати всі можливі комбінації значень параметрів. Ми можемо використовувати ступінчастий масив або інші структури даних для збереження цих комбінацій.
3. Створення функції для перевірки комбінацій: Написання функції, яка буде перевіряти кожну комбінацію значень параметрів. Ми можемо використовувати умовні оператори або функції для перевірки правильності значень параметрів.
4. Створення таблиці рішень: Використовуючи результати перевірки комбінацій, побудуймо таблицю рішень, де вказано, які комбінації мають правильні значення параметрів, а які - неправильні.

5. Виведення таблиці рішень: Виведення таблиці рішень на екран або збереження її у файл для подальшого аналізу.

Це загальна структура програми, яка може бути розширена і доповнена залежно від наших потреб і конкретної мови програмування, яку ми використовуємо.

3.2. Розробка системи автоматизованого тестування веб-додатків

PICT (Pairwise Independent Combinatorial Testing) є вільним інструментом, розробленим Microsoft, і він дійсно добре підходить для створення набору комбінацій тестових даних швидко, особливо коли маємо велику кількість незначно пов'язаних параметрів. Використання парного тестування дозволяє зменшити кількість тестових випадків, не втрачаючи при цьому покриття різних комбінацій параметрів.

Однак, важливо правильно створити модель, щоб тестове покриття було задовільним. Це означає, що потрібно аналізувати параметри та їх взаємозв'язки, визначати, які комбінації параметрів мають сенс для тестування, і враховувати специфіку системи, яку ви тестуєте.

PICT допомагає згенерувати ефективний набір комбінацій тестових даних, але його результати мають бути перевірені та належним чином адаптовані до конкретних потреб вашого проекту та системи, що тестується.

Так, інструмент PICT і техніка попарного тестування зазвичай застосовуються у разі взаємодіючих параметрів, оскільки вони дозволяють ефективно перевірити комбінації значень цих параметрів, які можуть впливати на результат. Попарне тестування ставить за мету перевірити всі можливі комбінації пар параметрів, забезпечуючи тим самим широке покриття можливих взаємодій.

У випадку невзаємодіючих параметрів, коли вплив одного параметра не залежить від значень інших параметрів, окрема перевірка кожного параметра

може бути достатньою. Такі параметри можуть включати незалежні функціональні або конфігураційні параметри, які не впливають один на одного.

Проте, коли взаємодія між параметрами має значення, наприклад, коли ви перевіряєте різні поєднання операційних систем, браузерів та дозволів монітора, важливо перевірити комбінації цих параметрів. В такому випадку, застосування техніки попарного тестування та використання інструменту PICT може бути корисним для ефективного покриття можливих комбінацій значень параметрів та виявлення проблем, які можуть виникнути внаслідок їх взаємодії.

```python
from metacomm import allpairs

# Визначення вхідних параметрів
parameters = [
    ["GoodLogin", "NoLogin"],
    ["GoodPswd", "NoPswd"],
    ["GoodEmail", "NoEmail"],
    ["CheckAgree", "NoAgree"]
]

# Використання AllPairs для генерації комбінацій
combos = allpairs(parameters)

# Виведення згенерованих комбінацій
for combo in combos:
    print(combo)
```

Рис. 3.1. Основний Python-код для використання модуля AllPairs та генерації комбінацій параметрів

У цьому прикладі вхідні параметри задаються у вигляді списку списків, де кожен внутрішній список представляє рядки значень для певного параметра. В даному випадку, ми визначили чотири параметри: "логін",

"пароль", "електронна пошта" та "погоджувальний прапорець", і для кожного параметра вказали дві можливі значення.

Потім ми викликаємо функцію allpairs з пакету metacomm для генерації всіх можливих комбінацій параметрів. Результатом буде список комбінацій, які потім можуть бути використані для тестування реєстрації користувача на сайті AllPairs.

Даний рядок команд запуску консолі для використання програми All-Pairs виглядає наступним чином:

C:\allpairs.exe C:\input.txt > C:\re.txt

Для виконання цієї команди необхідно мати програму All-Pairs (allpairs.exe) та текстовий файл (input.txt), що містить таблицю параметрів розділену табуляцією. Шляхи до програми allpairs.exe, вхідного файлу input.txt та вихідного файлу re.txt повинні бути вказані повністю, включаючи диск та розташування файлів.

Після виконання цієї команди в консолі, програма All-Pairs обробить вхідний файл і згенерує результат в файлі re.txt. Результуючий файл міститиме готовий перелік перевірки з унікальними комбінаціями параметрів згідно зі зазначеними у вхідному файлі обмеженнями.

Pairings - це кількість унікальних комбінацій (пар) параметрів, що використовуються в тестовому випадку. Наприклад, якщо у вас є тестовий випадок з трьома параметрами A, B і C, і для кожного параметра є 2 можливих значення, то загальна кількість можливих пар (унікальних комбінацій) буде 2 * 2 * 2 = 8.

Pairing details - це перелік всіх пар, які утворюються з комбінацій параметрів. Наприклад, якщо у вас є тестовий випадок з параметрами A, B і C, то pairing details буде містити всі можливі комбінації цих параметрів, наприклад (A1, B1, C1), (A1, B1, C2), (A1, B2, C1) і так далі.

Appearances - це кількість разів, коли певна пара (комбінація параметрів) з'являється в тестових випадках. Наприклад, якщо певна пара (A1, B2, C1) з'являється в 5 різних випадках, то її appearances буде 5.

Cases - це номери випадків, в яких фігурує певна пара (комбінація параметрів). Наприклад, якщо певна пара (A1, B2, C1) з'являється в 3-му, 8-му і 10-му випадках, то cases будуть містити номери цих випадків (3, 8, 10).

Ці дані допомагають аналізувати використання різних комбінацій параметрів у тестових випадках та оцінювати покриття тестування за допомогою різних комбінацій.

3.3. Тестування системи автоматизованого тестування веб-додатків

Узагальнена структура системи автоматизації тестування, яка включає таку інформацію, може мати наступну організацію:

1. Набір тестів:
   - Набір тестів, який включає в себе достатню кількість тестів для покриття тестованої програми згідно з обраним критерієм тестування.
   - Результати ручної або автоматичної розробки (генерації) тестових наборів.
   - Драйвер або монітор пропуску тестового набору, що виконує контроль проходження тестів.
2. Log-файл:
   - Результати прогону тестового набору, які записуються в Log-файл.
   - Log-файл містить траси (протоколи) подій, які відображають послідовність подій, значень змінних або їх комбінацій, а також точки реалізації цих подій на графі програми.

- Траси можуть включати явно та неявно задані мітки, які визначають шлях реалізації трас на керуючому графі програми, значення змінних на цих мітках, проміжні результати тощо.

3. Статистика тестового циклу:
   - Результати пропуску кожного тесту з тестового набору.
   - Порівняння результатів з еталонними значеннями.
4. Факти для прийняття рішень:
   - Факти, що послужили підставою для прийняття рішення про продовження або закінчення тестування, наприклад, кількість помилок, виконання критеріїв якості тощо.
5. Критерій покриття:
   - Критерій покриття, який визначає обсяг покриття програмного коду або функціональності.
   - Ступінь задоволення критерію покриття, досягнутий в циклі тестування.

Цей процес дозволяє систематично виявляти, відслідковувати і вирішувати проблеми в проекті, забезпечуючи постійне поліпшення якості програмного комплексу з кожною новою збіркою.

В ітераційному процесі розробки програмного комплексу засоби автоматизації тестування відіграють важливу роль. Після виправлення помилок і збірки програми, нова версія надсилається на наступний цикл тестування. Автоматизовані тести дозволяють швидко перевірити результати виправлень і контролювати рівень якості, досягнутий в продукті.

Засоби автоматизації тестування забезпечують швидку і ефективну перевірку функціональності, надійності і продуктивності програмного комплексу на кожному етапі розробки. Вони допомагають виявляти помилки та проблеми швидше, виконувати тести в автоматичному режимі та забезпечувати стабільну якість продукту.

Завдяки автоматизованому тестуванню можна виявити помилки швидше, підтримувати стабільність якості програмного комплексу і забезпечити його високу якість перед випуском на ринок.

За результатами дослідження, виявлено, що використання автоматизованого тестування значно прискорює процес виконання тестів у порівнянні з ручним тестуванням.

За наведеними даними, було проведено тестування 100 тестів на комп'ютері з процесором Intel core i5-4430 CPU @ 3.00 GHz та 8 ГБ оперативної пам'яті. Результати показують, що використання середовища розробки Eclipse дозволило виконати ці 100 тестів за 35 хвилин 58 секунд, тоді як у середовищі Intellij IDEA цей час скоротився до 21 хвилини 12 секунд.

При умові, що людина виконувала б ті самі 100 тестів, час виконання становив би близько 24 годин або 3 робочих дні.

Таким чином, можна зробити висновок, що використання автоматизованого тестування у середовищі розробки, таких як Eclipse та Intellij IDEA, може суттєво зекономити час та прискорити процес тестування порівняно з ручним виконанням тестів.

3.4. Висновок до розділу 3

Інтеграція автоматизованого тестування в середовищі розробки, такому як IntelliJ IDEA, може дійсно забезпечити швидкий та зручний процес тестування.

IntelliJ IDEA є популярним інтегрованим середовищем розробки (IDE) для мови програмування Java, але також підтримує інші мови програмування. Його багатофункціональність та широкі можливості редагування, налагодження та автоматизованого тестування роблять його зручним інструментом для розробки і виконання автоматизованих тестів.

Використання середовища розробки для автоматизації тестування має свої переваги, зокрема:

1. Інтегрований підхід: Можна створювати, редагувати та запускати тестові сценарії безпосередньо в середовищі розробки, що спрощує роботу з тестами і полегшує їх налагодження.
2. Покращена продуктивність: Завдяки інтеграції з середовищем розробки, можна швидше переключатися між кодом програми та тестами, використовувати автодоповнення та інші корисні функції, що полегшують процес написання та виконання тестів.
3. Розширюваність: IntelliJ IDEA має велику кількість плагінів та розширень, які дозволяють розширювати функціональність автоматизованого тестування, додавати підтримку для різних фреймворків, інструментів і мов програмування.

Проте, важливо враховувати, що ефективність автоматизованого тестування також залежить від якості самого тестування, правильного вибору тестових сцен

ВИСНОВКИ

Було досліджено різні методи та алгоритми автоматизації тестування веб-додатків у рамках дипломної роботи. Було проведено огляд засобів і методів автоматизації тестування, що дозволило зрозуміти, які інструменти доступні для автоматизованого тестування веб-додатків.

Також було розкрито теорію тестування веб-додатків, що є важливим аспектом при плануванні та виконанні тестів. Вивчення методологій автоматичного тестування дозволило засвоїти ефективні підходи та методики для автоматизованого тестування веб-додатків.

Крім того, запропоновано заходи для підвищення продуктивності тестування, що є важливим аспектом у веб-розробці. Це може включати в себе оптимізацію процесу тестування, використання ефективних інструментів та підходів, а також створення надійних тестових сценаріїв.

Дослідження та пропозиції, щодо підвищення продуктивності тестування, мають потенціал для покращення якості та ефективності розробки веб-додатків. Такі дослідження допомагають вдосконалювати методи та практики автоматизації тестування і сприяють створенню більш стійкого та надійного програмного забезпечення.

Основною метою тестування є забезпечення якості розроблюваного веб-додатку. Надійність є одним з ключових параметрів якості програми і вимірюється її здатністю працювати без відмов протягом певного періоду часу. Надійність враховує вартість для користувача кожної відмови, оскільки відмова може призвести до негативного впливу на користувача, втрати даних або інших небажаних наслідків.

Тестування веб-додатку є процесом, спрямованим на виявлення помилок в програмі. Кожна виявлена в процесі тестування помилка свідчить про відмову програми. У цьому контексті "вдалим" тестом є той, на якому виконання програми завершилося з помилкою, оскільки він допомагає ідентифікувати проблеми в програмі і дає можливість їх виправити. З іншого боку, "невдалим" тестом є той, що не зміг виявити помилку в програмі. Невдалий тест може створити хибне враження про надійність програми, оскільки він не розкриває потенційні проблеми, які можуть виникнути в реальних умовах використання.

У процесі тестування веб-додатку важливо забезпечувати якомога більшу кількість вдалої тестової покриття, щоб зменшити ймовірність невиявлення помилок. Розробка різноманітних тестових сценаріїв і використання різних методів та інструментів допомагають забезпечити ефективне тестування веб-додатків та покращити їх надійність.

Деякі основні переваги автоматизації:

1. Можливість запускати тестові скрипти в будь-який час доби: Автоматизовані тестові скрипти можна запускати автоматично в будь-який зручний для вас час, навіть коли ви не присутні перед комп'ютером. Це дозволяє зекономити час і забезпечує неперервність тестування.
2. Паралельне тестування: Завдяки автоматизації ви можете запускати тестові скрипти на декількох віддалених машинах одночасно. Це дозволяє проводити тестування паралельно з ручним, що прискорює процес тестування і забезпечує більш швидке виявлення помилок.

Незважаючи на переваги, автоматизація тестування також має свої обмеження і недоліки:

1. Витрати часу на розробку та підтримку скриптів: Розробка якісних та стабільних автоматизованих тестових скриптів може зайняти значну кількість часу. Крім того, для підтримки автоматизованих тестів

потрібно проводити постійну роботу з оновленням скриптів при зміні функціональності веб-додатку.

2. Обмеження функціональності, що піддається автоматизації: Не всі аспекти функціональності веб-додатку легко автоматизувати. Деякі випадки взаємодії з користувачем, складні алгоритми або елементи, які не підтримуються автоматизаційними інструментами, можуть потребувати ручного тестування.

Необхідно обережно підходити до вибору тестових сценаріїв для автоматизації тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bugs Catcher / Способ доступа: URL: http://bugscatcher.net/archives/1232. – WebDriver
2. C. Poole, J. Terrell, S. Busoli. NUnit 2012. [Электронный ресурс]. URL: http://www.nunit.org
3. XSpider 7.8 // PositiveTechnologies.2002-2012. [Электронный ресурс]. URL: http://www.ptsecurity.ru/xs7
4. C. Poole, J. Terrell, S. Busoli. NUnit 2012. [Электронный ресурс]. URL: http://www.nunit.org
5. Короткий посібник по React JS / Хабр [Електронний ресурс]. — Режим доступу: https://habr.com/post/248799/
6. Офіційна документація по Angular.js. [Електронний ресурс]. — Режим доступу: https://angularjs.org/api/
7. Пан К. С., Цымблер М. Л. Розробка паралельної СУБД на основі послідовної СУБД PostgreSQL з відкритим вихідним кодом. *Вісник ЮУрГУ*. Серия: Математичне моделювання і програмування. 2012. №18 (277). С. 112-119.
8. “Повернення на роботу після COVID-19”: IQAir. [Електронний ресурс]. — Режим доступу: https://www.iqair.com/blog/health-wellness/return-to-work-after-covid-19
9. Програмування на Python / Под ред. Марк Лутц - М., Символ-Плюс, 2011. - 15 с
10. Про проект SaveEcoBot. [Електронний ресурс]. — Режим доступу: https://www.saveecobot.com/static/about

11. Чепегин И. Д. Серверный JavaScript – переваги і недоліки node. JS // Вісник науки і освіти. 2020. №12-1 (90). С. 18-19.
12. Що таке Virtual DOM? / Хабр. [Електронний ресурс]. — Режим доступу: https://habr.com/post/256965/
13. Язык программирования С++.Специальное издание / Под ред. Бьерн Страуструп - М., Бином 2011 - 45 с.
14. Agrawal R., Ailamaki A., Bernstein P.A. et al. The Claremont Report on Database Research // Commun. ACM. 2009. Vol. 52, No. 6. P. 56-65. DOI: 10.1145/1516046.1516062.
15. Codd E.F. A Relational Model of Data for Large Shared Data Banks // Commun. ACM. 1970. Vol. 13, No. 6. P. 377-387. DOI: 10.1145/362384.362685.
16. Davoudian A., Chen L., Liu M. A Survey on NoSQL Stores // ACM Comput. Surv. 2018. Vol. 51, No. 2. P. 40:1-40:43. DOI: 10.1145/3158661.
17. Developer Survey Results 2019. [Електронний ресурс]. — Режим доступу: https://insights.stackoverflow.com/survey/2019/
18. Frawley W.J., Piatetsky-Shapiro G., Matheus C.J. Knowledge Discovery in Databases: an Overview // Knowledge Discovery in Databases. AAAI/MIT Press, 1991. P. 1-30.
19. JavaSpript. Подробное руководство / Под ред. Дэвид Флэнаган. СПб, Символ-Плюс, 2008. 20-23 с.
20. Kotowski N., Lima A.A.B, Pacitti E., Valduriez P., Mattoso M. Parallel Query Processing for OLAP in Grids. Concurrency and Computation: Practice and Experience, 2008, vol. 20, no. 17, pp. 2039 - 2048.
21. MongoDB Production Deployments [Електронний ресурс]. — Режим доступу: http://www.mongodb.org/about/ production-deployments
22. Paes M., Lima A.A.B., Valduriez P., Mattoso M. High-Performance Query Processing of a Real-World OLAP Database with ParGRES. VECPAR, Lecture Notes in Computer Science, 2008, vol. 5336, pp. 188-200.

23. State and Lifecycle - React [Електронний ресурс]. — Режим доступу: https://reactjs.org/ docs/state-and-lifecycle.html
24. Stonebraker M., Madden S., Dubey P. Intel “Big Data” Science and Technology Center Vision and Execution Plan // SIGMOD Record. 2013. Vol. 42, No. 1. P. 44-49. DOI: 10.1145/2481528.2481537.

ДОДАТКИ

ДОДАТОК А

```
<component name="InspectionProjectProfileManager">
  <profile version="1.0">
    <option name="myName" value="Project Default" />
    <inspection_tool class="DuplicatedCode" enabled="true" level="WEAK WARNING"
enabled_by_default="true">
      <Languages>
        <language minSize="54" name="Python" />
      </Languages>
    </inspection_tool>
    <inspection_tool class="PyInterpreterInspection" enabled="false" level="WARNING"
enabled_by_default="false" />
    <inspection_tool class="PyPackageRequirementsInspection" enabled="true"
level="WARNING" enabled_by_default="true">
      <option name="ignoredPackages">
        <value>
          <list size="3">
            <item index="0" class="java.lang.String" itemvalue="Tensorflow" />
            <item index="1" class="java.lang.String" itemvalue="Sklearn" />
            <item index="2" class="java.lang.String" itemvalue="Opencv" />
          </list>
        </value>
      </option>
    </inspection_tool>
    <inspection_tool class="PyPep8Inspection" enabled="true" level="WEAK WARNING"
enabled_by_default="true">
      <option name="ignoredErrors">
        <list>
          <option value="E501" />
        </list>
      </option>
    </inspection_tool>
  </profile>
</component>
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ProjectRootManager" version="2" project-jdk-name="Python 3.6" project-
jdk-type="Python SDK" />
</project>
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ProjectModuleManager">
    <modules>
      <module fileurl="file://$PROJECT_DIR$/.idea/UIED.iml"
filepath="$PROJECT_DIR$/.idea/UIED.iml" />
    </modules>
  </component>
</project>
<?xml version="1.0" encoding="UTF-8"?>
<module type="PYTHON_MODULE" version="4">
  <component name="NewModuleRootManager">
    <content url="file://$MODULE_DIR$">
      <sourceFolder url="file://$MODULE_DIR$" isTestSource="false" />
      <sourceFolder url="file://$MODULE_DIR$/resnet" isTestSource="false" />
    </content>
    <orderEntry type="inheritedJdk" />
    <orderEntry type="sourceFolder" forTests="false" />
  </component>
  <component name="TestRunnerService">
    <option name="PROJECT_TEST_RUNNER" value="py.test" />
  </component>
```

```
</module>
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="VcsDirectoryMappings">
    <mapping directory="$PROJECT_DIR$" vcs="Git" />
  </component>
</project>
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="AutoImportSettings">
    <option name="autoReloadType" value="SELECTIVE" />
  </component>
  <component name="ChangeListManager">
    <list default="true" id="b4069649-920d-465f-ac6b-bac85007c2bb" name="Default"
comment="">
      <change beforePath="$PROJECT_DIR$/.idea/workspace.xml" beforeDir="false"
afterPath="$PROJECT_DIR$/.idea/workspace.xml" afterDir="false" />
    </list>
    <option name="SHOW_DIALOG" value="false" />
    <option name="HIGHLIGHT_CONFLICTS" value="true" />
    <option name="HIGHLIGHT_NON_ACTIVE_CHANGELIST" value="false" />
    <option name="LAST_RESOLUTION" value="IGNORE" />
  </component>
  <component name="FileTemplateManagerImpl">
    <option name="RECENT_TEMPLATES">
      <list>
        <option value="Python Script" />
      </list>
    </option>
  </component>
  <component name="Git.Settings">
    <option name="RECENT_GIT_ROOT_PATH" value="$PROJECT_DIR$" />
  </component>
  <component name="JupyterTrust" id="2ce0fe3c-0081-4dca-a6a6-b1219d650764" />
  <component name="ProjectId" id="1dwCihBTog6GQX95sgxr7TpM6ZO" />
  <component name="ProjectLevelVcsManager">
    <ConfirmationsSetting value="2" id="Add" />
  </component>
  <component name="ProjectViewState">
    <option name="hideEmptyMiddlePackages" value="true" />
    <option name="showLibraryContents" value="true" />
    <option name="showMembers" value="true" />
  </component>
  <component name="PropertiesComponent">
    <property name="RunOnceActivity.OpenProjectViewOnStart" value="true" />
    <property name="RunOnceActivity.ShowReadmeOnStart" value="true" />
    <property name="WebServerToolWindowFactoryState" value="false" />
    <property name="last_opened_file_path" value="$PROJECT_DIR$" />
    <property name="node.js.detected.package.eslint" value="true" />
    <property name="node.js.detected.package.tslint" value="true" />
    <property name="node.js.path.for.package.eslint" value="project" />
    <property name="node.js.path.for.package.tslint" value="project" />
    <property name="node.js.selected.package.eslint" value="(autodetect)" />
    <property name="node.js.selected.package.tslint" value="(autodetect)" />
    <property name="restartRequiresConfirmation" value="false" />
    <property name="settings.editor.selected.configurable"
value="com.jetbrains.python.configuration.PyActiveSdkModuleConfigurable" />
    <property name="two.files.diff.last.used.file"
value="$PROJECT_DIR$/../UI2CODE/Element-Detection/merge.py" />
  </component>
  <component name="RecentsManager">
    <key name="CopyFile.RECENT_KEYS">
```

```
      <recent name="D:\git_file\github\doing\UIED\data\demo" />
</key>
    <key name="MoveFile.RECENT_KEYS">
      <recent name="D:\git_file\github\doing\UIED\detect_compo" />
      <recent name="D:\git_file\github\doing\UIED\detect_compo\deprecated" />
      <recent name="D:\git_file\github\doing\UIED\utils" />
      <recent name="D:\git_file\github\doing\UIED" />
      <recent name="D:\git_file\github\doing\UIED\result_processing" />
    </key>
  </component>
  <component name="RunManager" selected="Python.run_single">
    <configuration name="merge" type="PythonConfigurationType" factoryName="Python"
temporary="true" nameIsGenerated="true">
      <module name="UIED" />
      <option name="INTERPRETER_OPTIONS" value="" />
      <option name="PARENT_ENVS" value="true" />
      <envs>
        <env name="PYTHONUNBUFFERED" value="1" />
      </envs>
      <option name="SDK_HOME" value="" />
      <option name="WORKING_DIRECTORY" value="$PROJECT_DIR$/detect_merge" />
      <option name="IS_MODULE_SDK" value="true" />
      <option name="ADD_CONTENT_ROOTS" value="true" />
      <option name="ADD_SOURCE_ROOTS" value="true" />
      <EXTENSION ID="PythonCoverageRunConfigurationExtension" runner="coverage.py" />
      <option name="SCRIPT_NAME" value="$PROJECT_DIR$/detect_merge/merge.py" />
      <option name="PARAMETERS" value="" />
      <option name="SHOW_COMMAND_LINE" value="false" />
      <option name="EMULATE_TERMINAL" value="false" />
      <option name="MODULE_MODE" value="false" />
      <option name="REDIRECT_INPUT" value="false" />
      <option name="INPUT_FILE" value="" />
      <method v="2" />
    </configuration>
    <configuration name="merge2" type="PythonConfigurationType" factoryName="Python"
temporary="true" nameIsGenerated="true">
      <module name="UIED" />
      <option name="INTERPRETER_OPTIONS" value="" />
      <option name="PARENT_ENVS" value="true" />
      <envs>
        <env name="PYTHONUNBUFFERED" value="1" />
      </envs>
      <option name="SDK_HOME" value="" />
      <option name="WORKING_DIRECTORY" value="$PROJECT_DIR$" />
      <option name="IS_MODULE_SDK" value="true" />
      <option name="ADD_CONTENT_ROOTS" value="true" />
      <option name="ADD_SOURCE_ROOTS" value="true" />
      <EXTENSION ID="PythonCoverageRunConfigurationExtension" runner="coverage.py" />
      <option name="SCRIPT_NAME" value="$PROJECT_DIR$/merge2.py" />
      <option name="PARAMETERS" value="" />
      <option name="SHOW_COMMAND_LINE" value="false" />
      <option name="EMULATE_TERMINAL" value="false" />
      <option name="MODULE_MODE" value="false" />
      <option name="REDIRECT_INPUT" value="false" />
      <option name="INPUT_FILE" value="" />
      <method v="2" />
    </configuration>
    <configuration name="run_single" type="PythonConfigurationType" factoryName="Python"
temporary="true" nameIsGenerated="true">
      <module name="UIED" />
      <option name="INTERPRETER_OPTIONS" value="" />
      <option name="PARENT_ENVS" value="true" />
```

```
      <envs>
        <env name="PYTHONUNBUFFERED" value="1" />
      </envs>
      <option name="SDK_HOME" value="" />
      <option name="WORKING_DIRECTORY" value="$PROJECT_DIR$" />
      <option name="IS_MODULE_SDK" value="true" />
      <option name="ADD_CONTENT_ROOTS" value="true" />
      <option name="ADD_SOURCE_ROOTS" value="true" />
      <EXTENSION ID="PythonCoverageRunConfigurationExtension" runner="coverage.py" />
      <option name="SCRIPT_NAME" value="$PROJECT_DIR$/run_single.py" />
      <option name="PARAMETERS" value="" />
      <option name="SHOW_COMMAND_LINE" value="false" />
      <option name="EMULATE_TERMINAL" value="false" />
      <option name="MODULE_MODE" value="false" />
      <option name="REDIRECT_INPUT" value="false" />
      <option name="INPUT_FILE" value="" />
      <method v="2" />
    </configuration>
    <configuration name="run_testing(Used for Adjusting)" type="PythonConfigurationType"
factoryName="Python" temporary="true" nameIsGenerated="true">
      <module name="UIED" />
      <option name="INTERPRETER_OPTIONS" value="" />
      <option name="PARENT_ENVS" value="true" />
      <envs>
        <env name="PYTHONUNBUFFERED" value="1" />
      </envs>
      <option name="SDK_HOME" value="" />
      <option name="WORKING_DIRECTORY" value="$PROJECT_DIR$" />
      <option name="IS_MODULE_SDK" value="true" />
      <option name="ADD_CONTENT_ROOTS" value="true" />
      <option name="ADD_SOURCE_ROOTS" value="true" />
      <EXTENSION ID="PythonCoverageRunConfigurationExtension" runner="coverage.py" />
      <option name="SCRIPT_NAME" value="$PROJECT_DIR$/run_testing(Used for Adjusting).py"
/>
      <option name="PARAMETERS" value="" />
      <option name="SHOW_COMMAND_LINE" value="false" />
      <option name="EMULATE_TERMINAL" value="false" />
      <option name="MODULE_MODE" value="false" />
      <option name="REDIRECT_INPUT" value="false" />
      <option name="INPUT_FILE" value="" />
      <method v="2" />
    </configuration>
    <configuration name="text_detection" type="PythonConfigurationType"
factoryName="Python" temporary="true" nameIsGenerated="true">
      <module name="UIED" />
      <option name="INTERPRETER_OPTIONS" value="" />
      <option name="PARENT_ENVS" value="true" />
      <envs>
        <env name="PYTHONUNBUFFERED" value="1" />
      </envs>
      <option name="SDK_HOME" value="" />
      <option name="WORKING_DIRECTORY" value="$PROJECT_DIR$/detect_text" />
      <option name="IS_MODULE_SDK" value="true" />
      <option name="ADD_CONTENT_ROOTS" value="true" />
      <option name="ADD_SOURCE_ROOTS" value="true" />
      <EXTENSION ID="PythonCoverageRunConfigurationExtension" runner="coverage.py" />
      <option name="SCRIPT_NAME" value="$PROJECT_DIR$/detect_text/text_detection.py" />
      <option name="PARAMETERS" value="" />
      <option name="SHOW_COMMAND_LINE" value="false" />
      <option name="EMULATE_TERMINAL" value="false" />
      <option name="MODULE_MODE" value="false" />
      <option name="REDIRECT_INPUT" value="false" />
```

```
      <option name="INPUT_FILE" value="" />
      <method v="2" />
    </configuration>
    <list>
      <item itemvalue="Python.merge2" />
      <item itemvalue="Python.merge" />
      <item itemvalue="Python.run_single" />
      <item itemvalue="Python.run_testing(Used for Adjusting)" />
      <item itemvalue="Python.text_detection" />
    </list>
    <recent_temporary>
      <list>
        <item itemvalue="Python.run_single" />
        <item itemvalue="Python.run_testing(Used for Adjusting)" />
        <item itemvalue="Python.merge" />
        <item itemvalue="Python.merge2" />
        <item itemvalue="Python.text_detection" />
      </list>
    </recent_temporary>
  </component>
  <component name="SpellCheckerSettings" RuntimeDictionaries="0" Folders="0"
CustomDictionaries="0" DefaultDictionary="application-level" UseSingleDictionary="true"
transferred="true" />
  <component name="SvnConfiguration">
    <configuration />
  </component>
  <component name="TaskManager">
    <task active="true" id="Default" summary="Default task">
      <changelist id="b4069649-920d-465f-ac6b-bac85007c2bb" name="Default" comment="" />
      <created>1581826543105</created>
      <option name="number" value="Default" />
      <option name="presentableId" value="Default" />
      <updated>1581826543105</updated>
      <workItem from="1593326898629" duration="4378000" />
      <workItem from="1594035929667" duration="39000" />
      <workItem from="1594085137817" duration="1275000" />
      <workItem from="1594100418101" duration="3473000" />
      <workItem from="1594163935256" duration="1077000" />
      <workItem from="1594182213679" duration="2275000" />
      <workItem from="1595462115111" duration="3597000" />
      <workItem from="1595466120556" duration="3158000" />
      <workItem from="1595488240502" duration="74000" />
      <workItem from="1595723287741" duration="773000" />
      <workItem from="1596417993986" duration="2972000" />
      <workItem from="1596442796733" duration="9601000" />
      <workItem from="1596604406529" duration="10000" />
      <workItem from="1596685531295" duration="1526000" />
      <workItem from="1596792120495" duration="27485000" />
      <workItem from="1596926803756" duration="894000" />
      <workItem from="1596955557805" duration="717000" />
      <workItem from="1597015667916" duration="1488000" />
      <workItem from="1598487885922" duration="962000" />
      <workItem from="1601609000493" duration="599000" />
      <workItem from="1601849117964" duration="1226000" />
      <workItem from="1601850762269" duration="6477000" />
      <workItem from="1601933366434" duration="6847000" />
      <workItem from="1602130037944" duration="4303000" />
      <workItem from="1602199380249" duration="5493000" />
      <workItem from="1603669721746" duration="6042000" />
      <workItem from="1604011435077" duration="517000" />
      <workItem from="1604016832655" duration="5114000" />
      <workItem from="1604037397074" duration="12109000" />
```

```
      <workItem from="1604564719252" duration="1133000" />
      <workItem from="1604619358289" duration="17569000" />
      <workItem from="1604874207809" duration="171000" />
      <workItem from="1605071062104" duration="3079000" />
      <workItem from="1605086142565" duration="6397000" />
      <workItem from="1625010508533" duration="4541000" />
      <workItem from="1625099073176" duration="49720000" />
      <workItem from="1625443415902" duration="26350000" />
      <workItem from="1625529598285" duration="26479000" />
      <workItem from="1625613709029" duration="14000" />
      <workItem from="1625730508694" duration="928000" />
      <workItem from="1625809233064" duration="837000" />
      <workItem from="1626009011038" duration="18000" />
      <workItem from="1626307428798" duration="1983000" />
      <workItem from="1628054466383" duration="1894000" />
      <workItem from="1628122812217" duration="7049000" />
      <workItem from="1630237947629" duration="453000" />
      <workItem from="1630268189943" duration="20000" />
      <workItem from="1630297231550" duration="5710000" />
      <workItem from="1630312264694" duration="7894000" />
      <workItem from="1631149307515" duration="752000" />
      <workItem from="1631576239206" duration="805000" />
      <workItem from="1631584649434" duration="2272000" />
      <workItem from="1648024033251" duration="1883000" />
</task>
    <servers />
  </component>
  <component name="TestHistory">
    <history-entry file="py_test_in_test_lucky_py - 2020.03.04 at 05h 51m 04s.xml">
      <configuration name="py.test in test_lucky.py" configurationId="tests" />
    </history-entry>
  </component>
  <component name="TypeScriptGeneratedFilesManager">
    <option name="version" value="3" />
  </component>
  <component name="XDebuggerManager">
    <breakpoint-manager>
      <breakpoints>
        <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
          <url />
          <line>147</line>
          <option name="timeStamp" value="3" />
        </line-breakpoint>
        <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
          <url />
          <line>134</line>
          <option name="timeStamp" value="4" />
        </line-breakpoint>
        <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
          <url />
          <line>135</line>
          <option name="timeStamp" value="5" />
        </line-breakpoint>
        <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
          <url />
          <line>136</line>
          <option name="timeStamp" value="6" />
        </line-breakpoint>
        <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
          <url>file://$PROJECT_DIR$/detect_text_east/lib_east/eval.py</url>
          <line>263</line>
          <option name="timeStamp" value="67" />
```

```
      </line-breakpoint>
      <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
        <url>file://$PROJECT_DIR$/result_processing/eval_classes.py</url>
        <line>92</line>
        <option name="timeStamp" value="92" />
      </line-breakpoint>
      <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
        <url>file://$PROJECT_DIR$/detect_text_east/lib_east/eval.py</url>
        <line>108</line>
        <option name="timeStamp" value="93" />
      </line-breakpoint>
      <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
        <url>file://$PROJECT_DIR$/cnn/CNN.py</url>
        <line>62</line>
        <option name="timeStamp" value="94" />
      </line-breakpoint>
      <line-breakpoint enabled="true" suspend="THREAD" type="python-line">
        <url>file://$PROJECT_DIR$/run_single.py</url>
        <line>27</line>
        <option name="timeStamp" value="101" />
      </line-breakpoint>
    </breakpoints>
    <default-breakpoints>
      <breakpoint type="python-exception">
        <properties notifyOnTerminate="true" exception="BaseException">
          <option name="notifyOnTerminate" value="true" />
        </properties>
      </breakpoint>
    </default-breakpoints>
  </breakpoint-manager>
</component>
<component name="com.intellij.coverage.CoverageDataManagerImpl">
  <SUITE FILE_PATH="coverage/UIED$view_gt.coverage" NAME="view_gt Coverage Results"
MODIFIED="1596418105849"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$/result_processing" />
  <SUITE FILE_PATH="coverage/UIED$run_single.coverage" NAME="run_single Coverage
Results" MODIFIED="1648024874423"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$" />
  <SUITE FILE_PATH="coverage/UIED$main_single.coverage" NAME="main_single Coverage
Results" MODIFIED="1594102734340"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$" />
  <SUITE FILE_PATH="coverage/UIED$run_batch.coverage" NAME="run_batch Coverage Results"
MODIFIED="1596448499254"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$" />
  <SUITE FILE_PATH="coverage/UIED_block$main_single.coverage" NAME="main_single
Coverage Results" MODIFIED="1594035942350"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$" />
  <SUITE FILE_PATH="coverage/UIED$run_testing_Used_for_Adjusting_.coverage"
NAME="run_testing(Used for Adjusting) Coverage Results" MODIFIED="1631149318891"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$" />
```

```
    <SUITE FILE_PATH="coverage/UIED$merge2.coverage" NAME="merge2 Coverage Results"
MODIFIED="1625271385465"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$" />

    <SUITE FILE_PATH="coverage/UIED$run_testing_Use_Me_for_Adjusting_.coverage"
NAME="run_testing(Use Me for Adjusting) Coverage Results" MODIFIED="1605091364353"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$" />
    <SUITE FILE_PATH="coverage/UIED$text_detection.coverage" NAME="text_detection
Coverage Results" MODIFIED="1625107691453"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$/detect_text" />
    <SUITE FILE_PATH="coverage/UIED$merge.coverage" NAME="merge Coverage Results"
MODIFIED="1625284362497"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$/detect_merge" />
    <SUITE FILE_PATH="coverage/UIED$experiment.coverage" NAME="experiment Coverage
Results" MODIFIED="1605087951044"
SOURCE_PROVIDER="com.intellij.coverage.DefaultCoverageFileProvider" RUNNER="coverage.py"
COVERAGE_BY_TEST_ENABLED="true" COVERAGE_TRACING_ENABLED="false"
WORKING_DIRECTORY="$PROJECT_DIR$/result_processing" />
  </component>
</project>
```

```
from os.path import join as pjoin
import os


class Config:

    def __init__(self):
        # setting CNN (graphic elements) model
        self.image_shape = (64, 64, 3)
        # self.MODEL_PATH = 'E:\\Mulong\\Model\\UI2CODE\\cnn6_icon.h5'
        # self.class_map = ['button', 'input', 'icon', 'img', 'text']
        self.CNN_PATH = 'E:/Mulong/Model/rico_compos/cnn-rico-1.h5'
        self.element_class = ['Button', 'CheckBox', 'Chronometer', 'EditText',
'ImageButton', 'ImageView',
                              'ProgressBar', 'RadioButton', 'RatingBar', 'SeekBar',
'Spinner', 'Switch',
                              'ToggleButton', 'VideoView', 'TextView']
        self.class_number = len(self.element_class)

        # setting EAST (ocr) model
        self.EAST_PATH = 'E:/Mulong/Model/East/east_icdar2015_resnet_v1_50_rbox'

        self.COLOR = {'Button': (0, 255, 0), 'CheckBox': (0, 0, 255), 'Chronometer':
(255, 166, 166),
                      'EditText': (255, 166, 0),
                      'ImageButton': (77, 77, 255), 'ImageView': (255, 0, 166),
'ProgressBar': (166, 0, 255),
                      'RadioButton': (166, 166, 166),
                      'RatingBar': (0, 166, 255), 'SeekBar': (0, 166, 10), 'Spinner':
(50, 21, 255),
                      'Switch': (80, 166, 66), 'ToggleButton': (0, 66, 80), 'VideoView':
(88, 66, 0),
                      'TextView': (169, 255, 0), 'NonText': (0,0,255),
                      'Compo':(0, 0, 255), 'Text':(169, 255, 0), 'Block':(80, 166, 66)}

    def build_output_folders(self):
        # setting data flow paths
        self.ROOT_INPUT = "E:\\Mulong\\Datasets\\rico\\combined"
        self.ROOT_OUTPUT = "E:\\Mulong\\Result\\rico\\rico_uied\\rico_new_uied_v3"

        self.ROOT_IMG_ORG = pjoin(self.ROOT_INPUT, "org")
        self.ROOT_IP = pjoin(self.ROOT_OUTPUT, "ip")
        self.ROOT_OCR = pjoin(self.ROOT_OUTPUT, "ocr")
        self.ROOT_MERGE = pjoin(self.ROOT_OUTPUT, "merge")
        self.ROOT_IMG_COMPONENT = pjoin(self.ROOT_OUTPUT, "components")
        if not os.path.exists(self.ROOT_IP):
            os.mkdir(self.ROOT_IP)
        if not os.path.exists(self.ROOT_OCR):
            os.mkdir(self.ROOT_OCR)
        if not os.path.exists(self.ROOT_MERGE):
            os.mkdir(self.ROOT_MERGE)
class Config:

    def __init__(self):
        # Adjustable
        # self.THRESHOLD_PRE_GRADIENT = 4                 # dribbble:4 rico:4 web:1
        # self.THRESHOLD_OBJ_MIN_AREA = 55                # bottom line 55 of small circle
        # self.THRESHOLD_BLOCK_GRADIENT = 5

        # *** Frozen ***
        self.THRESHOLD_REC_MIN_EVENNESS = 0.7
        self.THRESHOLD_REC_MAX_DENT_RATIO = 0.25
```

```
        self.THRESHOLD_LINE_THICKNESS = 8
        self.THRESHOLD_LINE_MIN_LENGTH = 0.95
        self.THRESHOLD_COMPO_MAX_SCALE = (0.25, 0.98)  # (120/800, 422.5/450) maximum
height and width ratio for a atomic compo (button)
        self.THRESHOLD_TEXT_MAX_WORD_GAP = 10
        self.THRESHOLD_TEXT_MAX_HEIGHT = 0.04  # 40/800 maximum height of text
        self.THRESHOLD_TOP_BOTTOM_BAR = (0.045, 0.94)  # (36/800, 752/800) height ratio
of top and bottom bar
        self.THRESHOLD_BLOCK_MIN_HEIGHT = 0.03  # 24/800

        # deprecated
        # self.THRESHOLD_OBJ_MIN_PERIMETER = 0
        # self.THRESHOLD_BLOCK_MAX_BORDER_THICKNESS = 8
        # self.THRESHOLD_BLOCK_MAX_CROSS_POINT = 0.1
        # self.THRESHOLD_UICOMPO_MIN_W_H_RATIO = 0.4
        # self.THRESHOLD_TEXT_MAX_WIDTH = 150
        # self.THRESHOLD_LINE_MIN_LENGTH_H = 50
        # self.THRESHOLD_LINE_MIN_LENGTH_V = 50
        # self.OCR_PADDING = 5
        # self.OCR_MIN_WORD_AREA = 0.45
        # self.THRESHOLD_MIN_IOU = 0.1            # dribbble:0.003 rico:0.1 web:0.1
        # self.THRESHOLD_BLOCK_MIN_EDGE_LENGTH = 210   # dribbble:68 rico:210 web:70
        # self.THRESHOLD_UICOMPO_MAX_W_H_RATIO = 10   # dribbble:10 rico:10 web:22

        self.CLASS_MAP = {'0':'Button', '1':'CheckBox', '2':'Chronometer',
'3':'EditText', '4':'ImageButton', '5':'ImageView',
                  '6':'ProgressBar', '7':'RadioButton', '8':'RatingBar', '9':'SeekBar',
'10':'Spinner', '11':'Switch',
                  '12':'ToggleButton', '13':'VideoView', '14':'TextView'}
        self.COLOR = {'Button': (0, 255, 0), 'CheckBox': (0, 0, 255), 'Chronometer':
(255, 166, 166),
                      'EditText': (255, 166, 0),
                      'ImageButton': (77, 77, 255), 'ImageView': (255, 0, 166),
'ProgressBar': (166, 0, 255),
                      'RadioButton': (166, 166, 166),
                      'RatingBar': (0, 166, 255), 'SeekBar': (0, 166, 10), 'Spinner':
(50, 21, 255),
                      'Switch': (80, 166, 66), 'ToggleButton': (0, 66, 80), 'VideoView':
(88, 66, 0),
                      'TextView': (169, 255, 0),

                      'Text':(169, 255, 0), 'Non-Text':(255, 0, 166),

                      'Noise':(6,6,255), 'Non-Noise': (6,255,6),

                      'Image':(255,6,6), 'Non-Image':(6,6,255)}
```

```
import multiprocessing
import glob
import time
import json
from tqdm import tqdm
from os.path import join as pjoin, exists
import cv2

import detect_compo.ip_region_proposal as ip


def resize_height_by_longest_edge(img_path, resize_length=800):
    org = cv2.imread(img_path)
    height, width = org.shape[:2]
    if height > width:
        return resize_length
    else:
        return int(resize_length * (height / width))


if __name__ == '__main__':
    # initialization
    input_img_root = "E:/Mulong/Datasets/rico/combined"
    output_root = "E:/Mulong/Result/rico/rico_uied/rico_new_uied_v3"
    data = json.load(open('E:/Mulong/Datasets/rico/instances_test.json', 'r'))

    input_imgs = [pjoin(input_img_root, img['file_name'].split('/')[-1]) for img in
data['images']]
    input_imgs = sorted(input_imgs, key=lambda x: int(x.split('/')[-1][:-4]))  # sorted
by index

    key_params = {'min-grad': 10, 'ffl-block': 5, 'min-ele-area': 50, 'merge-contained-
ele': True,
                  'max-word-inline-gap': 10, 'max-line-ingraph-gap': 4, 'remove-top-bar':
True}

    is_ip = False
    is_clf = False
    is_ocr = False
    is_merge = True

    # Load deep learning models in advance
    compo_classifier = None
    if is_ip and is_clf:
        compo_classifier = {}
        from cnn.CNN import CNN
        # compo_classifier['Image'] = CNN('Image')
        compo_classifier['Elements'] = CNN('Elements')
        # compo_classifier['Noise'] = CNN('Noise')
    ocr_model = None
    if is_ocr:
        import detect_text.text_detection as text

    # set the range of target inputs' indices
    num = 0
    start_index = 30800  # 61728
    end_index = 100000
    for input_img in input_imgs:
        resized_height = resize_height_by_longest_edge(input_img)
        index = input_img.split('/')[-1][:-4]
        if int(index) < start_index:
            continue
```

```
        if int(index) > end_index:
            break

        if is_ocr:
            text.text_detection(input_img, output_root, show=False)

        if is_ip:
            ip.compo_detection(input_img, output_root, key_params,
classifier=compo_classifier, resize_by_height=resized_height, show=False)

        if is_merge:
            import merge
            compo_path = pjoin(output_root, 'ip', str(index) + '.json')
            ocr_path = pjoin(output_root, 'ocr', str(index) + '.json')
            merge.merge(input_img, compo_path, ocr_path, output_root,
is_remove_top=key_params['remove-top-bar'], show=True)

        num += 1


from os.path import join as pjoin
import cv2
import os
import numpy as np


def resize_height_by_longest_edge(img_path, resize_length=800):
    org = cv2.imread(img_path)
    height, width = org.shape[:2]
    if height > width:
        return resize_length
    else:
        return int(resize_length * (height / width))


def color_tips():
    color_map = {'Text': (0, 0, 255), 'Compo': (0, 255, 0), 'Block': (0, 255, 255), 'Text
Content': (255, 0, 255)}
    board = np.zeros((200, 200, 3), dtype=np.uint8)

    board[:50, :, :] = (0, 0, 255)
    board[50:100, :, :] = (0, 255, 0)
    board[100:150, :, :] = (255, 0, 255)
    board[150:200, :, :] = (0, 255, 255)
    cv2.putText(board, 'Text', (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
    cv2.putText(board, 'Non-text Compo', (10, 70), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0,
0), 2)
    cv2.putText(board, "Compo's Text Content", (10, 120), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0, 0, 0), 2)
    cv2.putText(board, "Block", (10, 170), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
    cv2.imshow('colors', board)


if __name__ == '__main__':

    '''
        ele:min-grad: gradient threshold to produce binary map
        ele:ffl-block: fill-flood threshold
        ele:min-ele-area: minimum area for selected elements
        ele:merge-contained-ele: if True, merge elements contained in others
        text:max-word-inline-gap: words with smaller distance than the gap are counted as
a line
```

```
        text:max-line-gap: lines with smaller distance than the gap are counted as a
paragraph

        Tips:
        1. Larger *min-grad* produces fine-grained binary-map while prone to over-segment
element to small pieces
        2. Smaller *min-ele-area* leaves tiny elements while prone to produce noises
        3. If not *merge-contained-ele*, the elements inside others will be recognized,
while prone to produce noises
        4. The *max-word-inline-gap* and *max-line-gap* should be dependent on the input
image size and resolution

        mobile: {'min-grad':4, 'ffl-block':5, 'min-ele-area':50, 'max-word-inline-gap':6,
'max-line-gap':1}
        web   : {'min-grad':3, 'ffl-block':5, 'min-ele-area':25, 'max-word-inline-gap':4,
'max-line-gap':4}
    '''
    key_params = {'min-grad':10, 'ffl-block':5, 'min-ele-area':50,
                  'merge-contained-ele':True, 'merge-line-to-paragraph':False, 'remove-
bar':True}

    # set input image path
    input_path_img = 'data/input/497.jpg'
    output_root = 'data/output'

    resized_height = resize_height_by_longest_edge(input_path_img, resize_length=800)
    color_tips()

    is_ip = True
    is_clf = False
    is_ocr = True
    is_merge = True

    if is_ocr:
        import detect_text.text_detection as text
        os.makedirs(pjoin(output_root, 'ocr'), exist_ok=True)
        text.text_detection(input_path_img, output_root, show=True, method='google')

    if is_ip:
        import detect_compo.ip_region_proposal as ip
        os.makedirs(pjoin(output_root, 'ip'), exist_ok=True)
        # switch of the classification func
        classifier = None
        if is_clf:
            classifier = {}
            from cnn.CNN import CNN
            # classifier['Image'] = CNN('Image')
            classifier['Elements'] = CNN('Elements')
            # classifier['Noise'] = CNN('Noise')
        ip.compo_detection(input_path_img, output_root, key_params,
                           classifier=classifier, resize_by_height=resized_height,
show=False)

    if is_merge:
        import detect_merge.merge as merge
        os.makedirs(pjoin(output_root, 'merge'), exist_ok=True)
        name = input_path_img.split('/')[-1][:-4]
        compo_path = pjoin(output_root, 'ip', str(name) + '.json')
        ocr_path = pjoin(output_root, 'ocr', str(name) + '.json')
        merge.merge(input_path_img, compo_path, ocr_path, pjoin(output_root, 'merge'),
```

```
                    is_remove_bar=key_params['remove-bar'],
is_paragraph=key_params['merge-line-to-paragraph'], show=True)
from os.path import join as pjoin
import cv2
import os


def resize_height_by_longest_edge(img_path, resize_length=800):
    org = cv2.imread(img_path)
    height, width = org.shape[:2]
    if height > width:
        return resize_length
    else:
        return int(resize_length * (height / width))


def nothing(x):
    pass


if __name__ == '__main__':
    '''
        ele:min-grad: gradient threshold to produce binary map
        ele:ffl-block: fill-flood threshold
        ele:min-ele-area: minimum area for selected elements
        ele:merge-contained-ele: if True, merge elements contained in others
        text:max-word-inline-gap: words with smaller distance than the gap are counted as
a line
        text:max-line-gap: lines with smaller distance than the gap are counted as a
paragraph

        Tips:
        1. Larger *min-grad* produces fine-grained binary-map while prone to over-segment
element to small pieces
        2. Smaller *min-ele-area* leaves tiny elements while prone to produce noises
        3. If not *merge-contained-ele*, the elements inside others will be recognized,
while prone to produce noises
        4. The *max-word-inline-gap* and *max-line-gap* should be dependent on the input
image size and resolution

        mobile: {'min-grad':4, 'ffl-block':5, 'min-ele-area':50, 'max-word-inline-gap':6,
'max-line-gap':1}
        web   : {'min-grad':3, 'ffl-block':5, 'min-ele-area':25, 'max-word-inline-gap':4,
'max-line-gap':4}
    '''
    key_params = {'min-grad':10, 'ffl-block':5, 'min-ele-area':50, 'merge-contained-
ele':False,
                  'max-word-inline-gap':10, 'max-line-gap':4, 'remove-top-bar':True}

    # set input image path
    input_path_img = 'data/input/4.jpg'
    output_root = 'data/output'

    resized_height = resize_height_by_longest_edge(input_path_img)
    is_clf = False
    is_ocr = False
    if is_ocr:
        import detect_text.text_detection as text
        os.makedirs(pjoin(output_root, 'ocr'), exist_ok=True)
        text.text_detection(input_path_img, output_root, show=False)
```

```
    '''
    ******** Testing with adjustable parameters ********
    '''
    testing_ip = True
    testing_merge = False

    cv2.namedWindow('parameters')
    if testing_ip:
        cv2.createTrackbar('min-grad', 'parameters', 4, 20, nothing)
        cv2.createTrackbar('min-ele-area', 'parameters', 20, 200, nothing)
        while(1):
            key_params['min-grad'] = cv2.getTrackbarPos('min-grad', 'parameters')
            key_params['min-ele-area'] = cv2.getTrackbarPos('min-ele-area', 'parameters')
            import detect_compo.ip_region_proposal as ip
            os.makedirs(pjoin(output_root, 'ip'), exist_ok=True)
            # switch of the classification func
            classifier = None
            if is_clf:
                classifier = {}
                from cnn.CNN import CNN
                # classifier['Image'] = CNN('Image')
                classifier['Elements'] = CNN('Elements')
                # classifier['Noise'] = CNN('Noise')
            ip.compo_detection(input_path_img, output_root, key_params,
                               classifier=classifier, resize_by_height=resized_height,
show=True, wai_key=10)

    if testing_merge:
        cv2.createTrackbar('max-word-inline-gap', 'parameters', 4, 20, nothing)
        cv2.createTrackbar('max-line-gap', 'parameters', 20, 200, nothing)
        while(1):
            key_params['max-word-inline-gap'] = cv2.getTrackbarPos('max-word-inline-gap',
'parameters')
            key_params['max-line-gap'] = cv2.getTrackbarPos('max-line-gap', 'parameters')
            import detect_merge.merge as merge
            name = input_path_img.split('/')[-1][:-4]
            compo_path = pjoin(output_root, 'ip', str(name) + '.json')
            ocr_path = pjoin(output_root, 'ocr', str(name) + '.json')
            merge.merge(input_path_img, compo_path, ocr_path, output_root=None,
is_remove_top=key_params['remove-top-bar'], show=True, wait_key=10)
```