

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ
DEVELOPMENT OF WEB APPLICATION TESTING SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-218

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Авраменко І.Ю.

(прізвище та ініціали)

Керівник Гладкова О.М.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(прізвище та ініціали)

2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 20 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

- Авраменко Ігорю Юрійовичу
(прізвище, ім'я, по батькові)
1. Тема проєкту (роботи) Розробка програмного забезпечення тестування вебзастосунку. Development of Web Application Testing Software.
 керівник проєкту (роботи) Гладкова Ольга Миколаївна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102
 2. Строк подання студентом проєкту (роботи) червня 2022 року
 3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
 4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка завдання. 2. Вибір інструментарію розробки системи. 3. Проєктування та розробка програмного застосунку для автоматизованого тестування. 4. Експлуатація, тестування та експериментальне дослідження програми.
 5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Гладкова О.М., доцент		
Нормоконтролер	Липовець М.В., асистент		

7. Дата видачі завдання “04” квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Вибір інструментарію розробки.	4 тиждень	Розділ 2
4	Розробка програмного застосунку.	5–6 тиждень	Розділ 3
5	Оформлення пояснювальної записки та документів до неї.	7–8 тижні	Розділ 4, Додатки
6	Нормоконтроль та рецензування.	9 тиждень	
7	Захист роботи.	10 тиждень	
8			
9			
10			
11			
12			

Студент(ка)

(підпис) Авраменко І.Ю.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Гладкова О.М.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 62 с., 10 табл., 26 рис., 2 дод., 16 джерел.

ТЕСТУВАННЯ, АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ,
ВЕБЗАСТОСУНОК, JAVA, MAVEN.

Об'єкт дослідження – тестування вебзастосунків.

Предмет дослідження – автоматизація тестування вебзастосунків.

Мета роботи – розробити програмне забезпечення для автоматизації тестування вебзастосунку.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- виконати дослідження існуючих методів тестування та інструментів розробки;
- розробити структуру проєкту;
- розробити алгоритм роботи програмного забезпечення;
- розробити програмне забезпечення.

Матеріали, методи та технічні засоби: об'єктно орієнтоване програмування, Java мова програмування, середовище розробки IntelliJ IDEA, драйвер браузера Selenium WebDriver.

Результати. Розроблено програмне забезпечення для тестування вебзастосунку, яке дозволяє прискорити тестування функціональних блоків вебзастосунку за рахунок автоматизації тестових сценаріїв.

Галузь використання – тестування вебзастосунків.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 62 pages, 10 tables, 26 figures, 2 appendixes, 16 sources.

TESTING, AUTOMATION TESTING, WEB APPLICATION, JAVA, MAVEN.

The object of study is web application testing.

The subject of research is the automation of web application testing.

The purpose of the work is to develop software for automating web application testing.

To achieve this goal it is necessary to solve the following tasks:

- perform research on existing testing methods and development tools;
- develop the project structure;
- develop a software algorithm;
- develop software.

Materials, methods and technical means: object-oriented programming, Java programming language, IntelliJ IDEA development environment, Selenium WebDriver.

Results. A transparent software for testing a web application has been developed, which allows speeding up the testing of functional units of testing by automating test scenarios.

Field of use - testing web applications.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
ВСТУП	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ	11
1.1 Переваги та недоліки автоматизованого тестування перед ручним..	11
1.2 Огляд можливостей автоматизації тестів	13
1.3 Види автоматизованого тестування	14
1.4 Висновки за розділом та постановка задачі	16
2 ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ СИСТЕМ	18
2.1 Вибір мови програмування	18
2.2 Вибір середовища розробки та додатковий інструментарій	22
2.3 Висновки за розділом 2	27
3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ	29
3.1 Проєктування архітектури програмного продукту	29
3.2 Розробка структури програмного проєкту	29
3.3 Діаграма послідовності виконання програми	30
3.4 Проєктування класів та методів	32
3.5 Висновки до розділу 3	33
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	35
4.1 Інструкція по експлуатації програми.....	35
4.2 Функції системи	38
4.3 Результати тестування	39
4.6 Висновки до розділу 4	40
ВИСНОВКИ.....	41
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	43
ДОДАТОК А Текст програми.....	45

ДОДАТОК Б Слайди презентації	55
------------------------------------	----

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE	Integrated Development Environment
JIT	Just-In-Time
JVM	Java Virtual Machine
QA	Quality Assurance

ВСТУП

Тестування є частиною циклу розробки продукту, крім того дуже важливою для того, щоб розроблюваний продукт був таким, яким потрібно, та згідно з заданих вимог. Тестування є останньою у циклі перед самим випуском продукту до ринку, таким чином саме проведене тестування надає відповідь потрапляє продукт на ринок, відповідає він усім вимогам та виконує поставлені задачі. Або продукт ще треба доробити чи виправити знайдені помилки у роботі.

Автоматизоване тестування є методом тестування програмного забезпечення, який виконується з допомогою спеціальних програмних засобів, які, своєю чергою необхідні виконання набору тестових прикладів. Навпаки, ручне тестування виконується людиною, яка сидить перед комп'ютером і старанно виконує кожен крок тесту «руками».

Деякі команди просто не мають часу чи ресурсів, щоб вручну тестувати програмне забезпечення. Таким чином проваджується автоматизація. Це може значно скоротити час, необхідний для тестування продуктів, оскільки працює швидко та ефективно. У наш час тестування стало більш складним і розширеним. Почали використовувати автоматизоване тестування, щоб полегшити навантаження на решту команди та забезпечити чіткі та точні результати. Це економить час розробників і менеджерів, які можуть спрямувати свої зусилля на інші аспекти проєкту. В результаті це може значно підвищити продуктивність роботи.

Програмне забезпечення для автоматизації тестування також може вводити тестові дані у тестове середовище, порівнювати очікувані та фактичні результати та створювати докладні звіти про тести. Як правило, автоматизація тестування потребує значних вкладень грошей та ресурсів.

Автоматизоване тестування може скоротити життєві цикли розробки, усунути можливість людських помилок і автоматизувати звичайні й монотонні завдання.

Використання технології автоматизації також означає, що тестування можна проводити частіше, покращуючи загальну функціональність. Цикли розробки програмного забезпечення вимагають повторного тестування, часто одного і того ж тесту знову і знову. Тестування автоматизації робить це можливим, не відриваючи членів команди від іншої роботи. Він також може дати більш точні та надійні результати, ніж лише ручне тестування. Подальше забезпечення готовності продукту до ринку або переходу на наступний етап розробки. Ця перевірка дає команді поштовх для продовження розвитку.

Перед тим як розробляти автотести необхідно визначитись з видами тестування, які потрібно використати у проєкті, що саме можна перевести в автотести, а в чому немає необхідності. Таким чином робота присвячена актуальній темі, аналізу різновидів автоматизованого тестування, та розробці програмного забезпечення для автоматизованого тестування вебзастосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Переваги та недоліки автоматизованого тестування перед ручним

Автоматизація тестування є процесом тестування програмного забезпечення та інших технологічних продуктів, щоб переконатися, що воно відповідає поставленим вимогам. По суті, це тест, щоб ще раз перевірити, чи обладнання чи програмне забезпечення виконує саме те, для чого було розроблено. Воно перевіряє помилки, дефекти та будь-які інші проблеми, які можуть виникнути під час розробки продукту. [1]

Хоча деякі типи тестування, такі як регресійне або функціональне тестування, можна виконувати вручну, є більші переваги від автоматичного виконання. Автоматичне тестування можна проводити в будь-який час доби. Він використовує сценарні послідовності для перевірки програмного забезпечення. Потім він повідомляє про те, що було знайдено, і цю інформацію можна порівняти з попередніми тестовими запусками. [2]

Багато компаній, що займаються програмним забезпеченням, матимуть призначеного тестувальника автоматизації QA (quality assurance – забезпечення якості). Вони розробляють і пишуть тестові сценарії на початкових етапах. Тестер автоматизації QA працюватиме з інженерами з тестування автоматизації та розробниками продуктів, щоб фактично перевірити програмне забезпечення та продукти. Вони сформуєть команду та контролюватимуть ініціативи з автоматизації тестування, а також використовуватимуть різні типи платформ автоматизації тестування, щоб встановити найкращу для успішної автоматизації тестування. [2]

Деякі з найбільших переваг використання автоматизованого тестування для розробки програмного забезпечення представлені в таблиці 1.1.

Таблиця 1.1 – Переваги та недоліки автоматизованого тестування

Переваги	Недоліки
Прискорює та спрощує тестування	Витрачений час на написання та налагодження автотестів.
В довгострокових проєктах витрачені інвестиції на автотести повністю себе окупають.	Великі інвестиції на самому початку проєкту.
Зменшує людський фактор при перевірці.	При зміні логіки проєкту необхідна також повна зміна логіки автотестів.
Можлива краща звітність та більш повне оформлення.	

Можливості детальної звітності – автоматизоване тестування використовує добре розроблені тестові випадки для різних сценаріїв. Ці сценарії послідовності можуть бути неймовірно глибокими та надавати докладні звіти, які просто не були б можливими, коли б їх виконувала людина. Не кажучи вже про те, щоб забезпечити їх за короткий проміжок часу.

Покращене виявлення помилок . Однією з основних причин тестування продукту є виявлення помилок та інших дефектів. Автоматичне тестування спрощує цей процес. Він також здатний аналізувати ширше охоплення тестами, ніж люди.

Спрощує тестування. Тестування є рутинною частиною операцій більшості SaaS і технологічних компаній. Важливо зробити це максимально простим. Використання автоматизації надзвичайно вигідно. Під час автоматизації інструментів тестування тестові сценарії можна використовувати повторно. Ручне тестування, тим часом, вимагає написання одного рядка коду для одного тестового прикладу кожного разу, коли його потрібно запускати.

Прискорює процес тестування – машини та автоматизовані технології працюють швидше, ніж люди. Поряд із покращеною точністю, тому можна використовувати їх. У свою чергу, це скорочує цикли розробки програмного забезпечення. [3]

Зменшує людське втручання — тести можна проводити в будь-який час дня, навіть вночі, без потреби контролю за цими людьми. Крім того, коли воно виконується автоматично, це також може знизити ризик людської помилки.

Економія часу та грошей - тестування може зайняти багато часу. Хоча автоматизація може вимагати початкових інвестицій, вона може заощадити гроші в довгостроковій перспективі, щоб стати більш економічно ефективним. Члени команди використовують свій час в інших сферах, і їм більше не потрібно проводити ручне тестування в багатьох ситуаціях. Це покращує їхній робочий процес.

1.2 Огляд можливостей автоматизації тестів

Неможливо автоматизувати всі тести та тестові сценарії відразу, і взагалі в цьому немає жодної потреби. Отже, потрібно проаналізувати всі тести та вирішити, які тести автоматизувати в першу чергу, а які і зовсім немає необхідності автоматизувати (табл. 1.2).

Нижче наведено типи тестів, які можуть бути автоматизованими та їх автоматизація надає переваги [1]:

- тести, які можуть призвести до відмови через людську помилку;
- повторювані та монотонні тести;
- розширені тести, які вимагають кількох наборів даних;
- тести, які неможливо виконати вручну;
- тести, які займають багато часу вручну;
- тести високого ризику;
- тести, які потрібно виконувати на кількох різних апаратних та/або програмних платформах.

Тестові сценарії які слід автоматизувати наведені можна вибрати за такими критеріями:

- тестові сценарії, які регулярно повторюються;

- тестові сценарії, які дуже складні та стомлюючі для виконання вручну;
- тестові сценарії, що забирають багато часу.

Таблиця 1.2 – Тести які підходять і не підходять для автоматизації

Тести які підходять	Тести які не підходять
Тести, які регулярно або часто повторюються.	Тести що не пройшли ручне тестування жодного разу.
Тести які стомлюючі та важкі для виконання людиною.	Тести логіка яких дуже часто підпадає під зміни.
Тести на проведення яких необхідно багато часу.	Тести які виконуються лише одноразово.

Наступна категорія тестових випадків не підходить для автоматизації [4]:

- нові тестові сценарії, які не виконувались вручну хоча б один раз;
- сценарії тестування, вимоги до яких часто змінюються;
- тестові приклади, що виконуються на разовій основі.

1.3 Види автоматизованого тестування

Як показало дослідження [1] існує п'ять основних типів автоматизованого тестування. Кожен з них можна використовувати в різних обставинах, залежно від тестованого застосування. Можна проаналізувати кожен з них, щоб побачити, що буде найкращим для конкретного продукту, або можна перевірити їх за допомогою пробного періоду. Іноді це найкращий спосіб дізнатися, який інструмент необхідно використовувати у проєкті. Основні види інструментів автоматизованого тестування (рис. 1.1):

- аналіз коду;
- модульні тести;
- інтеграційні тести;
- автоматизовані прийняті тести;

— димові тести.

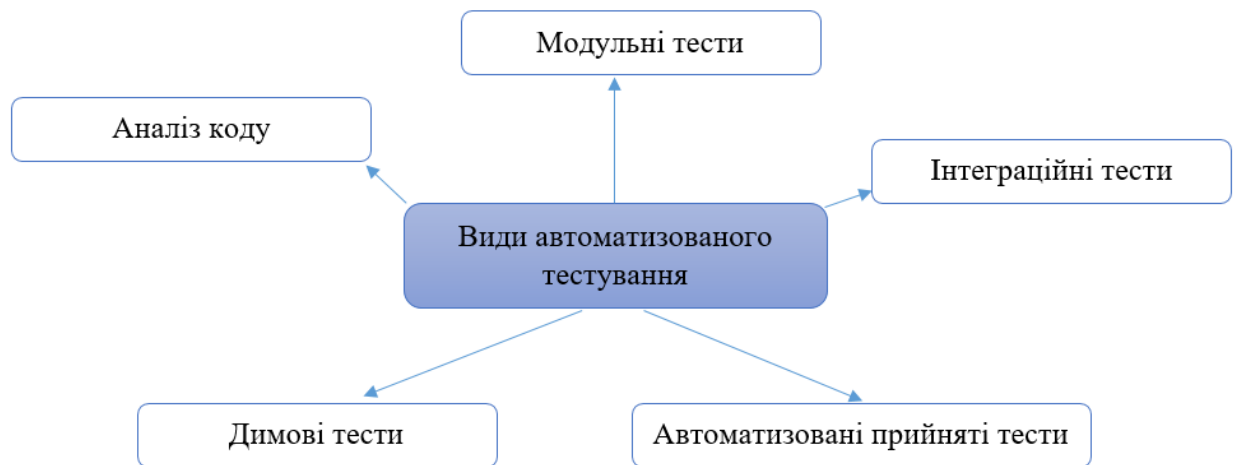


Рисунок 1.1 – Види автоматизованого тестування

Аналіз коду складається з різних інструментів тестування, включаючи динамічний аналіз і статичний аналіз. Ви можете застосувати різні для виконання окремих тестів. Наприклад, деякі шукають можливі недоліки безпеки, а інші перевіряють зручність використання. Щоб виконати ці тести, розробнику потрібно написати код. Однак, як тільки це буде зроблено, у решті процесу тестування не буде втручання людини. [5]

Модульне тестування полягає в перевірці окремих компонентів програмного забезпечення або продукту, як і розробка технології iOS для iPhone або Android. Це означає, що кожен елемент програмного забезпечення повністю тестується перед випуском готової версії. Ці тести можуть бути написані розробниками, але тепер, коли автоматизоване тестування вступило в гру, у цьому немає потреби. Підприємства зазвичай використовують такі типи тестів на етапі розробки програмного забезпечення виробничого процесу.

Інтеграційні тести, також відомі як наскрізні тести, часто складніше налаштувати, ніж деякі інші тести. Моделі додатків інтегруються та тестуються як група. Це означає, що зв'язок між кожним модулем можна перевірити, щоб з'ясувати, наскільки добре вони працюють в цілому.

Автоматизовані прийняті тести схожі на розробку, керовану поведінкою і автоматизовану розробку, керовану тестами прийняття. Тест приймання створюється перед розробкою нової функції. Він створює прецедент для відповідності функції і зазвичай пишеться розробниками, бізнесом та органами забезпечення якості разом. У майбутньому вони також можуть використовуватися як регресійні тести.

Димове тестування використовується, щоб перевірити, чи є продукт стабільним чи ні. Якщо він нестабільний, він повертається розробникам, позначеним як «нестабільна збірка». Потім вони можуть провести додаткові тести, якщо необхідно, щоб визначити основну причину проблеми. [6]

1.4 Висновки за розділом та постановка задачі

В даному розділі були розглянуті переваги та недоліки впровадження автоматизованого тестування в розроблюваний проєкт. Як показав аналіз розробка та впровадження автотестів в проєкт надає переваг більше ніж недоліків. Але варто дивитись на сам проєкт, якщо це аутсорсінгова компанія, в яку зайшов невеличкій проєкт на створення веб-сторінки профілю то впровадження автоматизації тестування не виправдає свого застосування. Але впровадження в середні та великі проєкти надає такі переваги як прискорення та спрощення тестування. Ці фактори стало основними перевагами для того, щоб автоматизувати тестування у наданому для тестів вебзастосунку.

В розділі було досліджено та проаналізовано які саме тестові сценарії потребують автоматизації, а які немає необхідності автоматизувати.

Також було виконано огляд сидів автоматизованого тестування та зроблено висновок, що на початку проєктування саме модульне тестування є доречним, оскільки є простим для початківця в автоматизації, бо не вимагає необхідності повного розуміння коду розробників або написання складних за логікою тестів для інтеграційного тестування.

Таким чином, завданням даної роботи є автоматизувати тестові сценарії для вебзастосунку, які необхідно постійно перевіряти, для звільнення часу тестувальника для виконання іншого виду роботи. Таким чином, буде застосоване так зване модульне тестування окремих компонентів вебзастосунку, які потребують частотої перевірки.

2 ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ СИСТЕМ

2.1 Вибір мови програмування

Дослідження у роботі [7] говорить про те, що серед поширених мов у сфері автоматизації варто виділити Java та Python, які також вважаються легшими для вивчення. Крім того, розробники автоматизації також пишуть на C#, JavaScript, а деякі також пишуть на Ruby.

Різниця в швидкості та ефективності між Python і Java. Ефективність Java значною мірою забезпечується її компілятором Just-In-Time (JIT) і підтримкою паралельності. Компілятор JIT є частиною середовища виконання Java. Він покращує продуктивність програм Java шляхом компіляції байт-кодів у рідний машинний код «точно вчасно» для виконання. Віртуальна машина Java (JVM) викликає безпосередньо скомпільований код. Оскільки код не інтерпретується, компіляція не вимагає часу процесора та використання пам'яті. Теоретично це може зробити програму Java такою ж швидкою, як і рідну програму. [8]

Хоча програми Java компілюються безпосередньо, Python інтерпретується, що сповільнює програми Python під час виконання. Визначення типу змінної, що виникає під час виконання, збільшує навантаження на інтерпретатор. Крім того, запам'ятовування типи об'єктів, отриманих з об'єктів-контейнерів, сприяє використанню пам'яті.

Виправлення помилок у Python проти Java. У Python будь-які помилки, внесені програмістом, не будуть знайдені, доки не буде запущено цей рядок коду. Це може призвести до збою в роботі та подовжити час виконання. Хоча Python залишає об'єкти вразливими до мутацій, в Java мутації об'єктів неможливі. Це веде до безпечної розробки програмного забезпечення. [9]

Ключові різниці між мовами програмування Java та Python перераховані далі. [10]

Java – це скомпільована плюс інтерпретована мова, тоді як Python – це інтерпретована мова.

Java типізується статично, тоді як Python — динамічно.

Java має складну криву навчання, тоді як Python легко вивчати та використовувати.

Java є багатоплатформною, об'єктно-орієнтованою та мережевою мовою програмування, тоді як Python є високорівневою об'єктно-орієнтованою мовою програмування.

Java потребує 10 рядків коду для зчитування з файлу, тоді як Python потребує лише 2 рядків коду.

Мова Java використовує фігурні дужки для визначення початку та кінця кожної функції та визначення класу, тоді як Python використовує відступ для розділення коду на окремі блоки.

У Java множинне успадкування частково здійснюється через інтерфейси, тоді як Python підтримує як одинарне, так і множинне успадкування.

Java пропонує обмежені функції, пов'язані з рядками, тоді як Python пропонує багато функцій, пов'язаних з рядками.

Програма Java може працювати на будь-якому комп'ютері або мобільному пристрої, який може запускати JVM, тоді як програмам Python потрібен інтерпретатор, встановлений на цільовій машині для перекладу коду Python. У порівнянні з Java, Python менш переносимий. [8]

Java найкраще підходить для настільних програм із графічним інтерфейсом користувача, систем вбудовування, служб веб-застосунків, тоді як Python відмінно підходить для наукових і числових обчислень, програм машинного навчання тощо.

Java пропонує стабільне підключення, тоді як Python пропонує слабе підключення.

В таблиці 2.1 наведене порівняння Java та Python за основними параметрами.

Таблиця 2.1 – Порівняння мов програмування Java та Python

Параметр	Java	Python
Вивчання	Порівняно складніша.	Легко навчатися.
Дужки, відступи	Він використовує фігурні дужки для визначення початку та кінця кожної функції та визначення класу.	Python використовує відступи для розділення коду на блоки коду.
Швидкість	Програма Java працює повільно в порівнянні з Python.	Програми на Python працюють швидше, ніж на Java.
Портативність	Будь-який комп'ютер або мобільний пристрій, який може запускати віртуальну машину Java, може запускати програму Java	Програмам Python потрібен інтерпретатор, встановлений на цільовій машині для перекладу коду Python. У порівнянні з Java, Python менш переносимий.
Відомі компанії використовують цю технологію	Airbnb, Netflix, Spotify, Instagram.	Uber, Technologies, Dropbox, Google.
Найкращі характеристики	Чудові бібліотеки Широко використовуваний Відмінна оснастка Доступна величезна кількість документів.	Читаний код Швидкий розвиток Гарний код
Найкраще використовувати для	Java найкраще підходить для настільних програм із графічним інтерфейсом користувача, систем вбудовування, служб веб-застосунків тощо.	Python чудово підходить для наукових і числових обчислень, програм машинного навчання тощо.
Підтримка бази даних	Java забезпечує стабільне підключення	Python пропонує слабке підключення.
Компанії, які його використовують	Docker, Android SDK, Spring Boot, Senty тощо.	Django, Flask, Pycharm, CircleCI тощо.

Ось переваги використання Java [8]:

- доступна детальна документація;
- доступний велике коло кваліфікованих розробників;
- величезний набір сторонніх бібліотек;

- він дозволяє формувати стандартні програми та багаторазовий код;
- це багатопотокове середовище, яке дозволяє виконувати багато завдань одночасно в програмі;

- відмінна продуктивність;
- зручна навігація по бібліотеках.

Недоліками Java є:

- компілятор JIT робить програму порівняно повільною;
- Java має високі вимоги до пам'яті та обробки, тому вартість обладнання зростає;
- він не забезпечує підтримку низькорівневих конструкцій програмування, таких як покажчики;
- користувач не має жодного контролю над збиранням сміття, оскільки Java не пропонує таких функцій, як `delete()`, `free()`.

Недоліками Python є [10]:

- використовується на меншій кількості платформ;
- Python інтерпретується, тому він набагато повільніший, ніж його аналоги;
- слабкий у мобільних комп'ютерах, тому не використовується в розробці застосунків;
- Science Python є динамічним, більше помилок з'являється під час виконання;
- нерозвинений і примітивний рівень доступу до бази даних;
- відсутність комерційної підтримки.

Основна відмінність між Java та Python полягає в тому, що Java – це статично типізована та компільована мова, яка пропонує обмежені функції, пов'язані з рядками, а Python – це динамічно типізована та інтерпретована мова, яка пропонує багато функцій, пов'язаних із рядками.

Python є ідеальним вибором для початківців, оскільки мова є більш інтуїтивно зрозумілою, а її синтаксис дуже схожий на англійську мову. Його

природа з відкритим кодом сприяє створенню безлічі нових інструментів, які покращують його.

З іншого боку, Java – це мова програмування загального призначення, яка була розроблена з конкретною метою дозволити розробникам писати після запуску в будь-якому місці. Вибір однієї мови серед них в кінцевому підсумку зводиться до характеру та вартості проекту розробки.

2.2 Вибір середовища розробки та додатковий інструментарій

Як свідчить дослідження [7] у світі тестування та автоматизації тестування є купа фреймворків для модульного тестування. Ці фреймворки дозволяють нам взяти код, який ми написали, і перетворити його в код, який можна перевірити, використовуючи функції метаданих, такі як: анотації або атрибути. Крім того, за допомогою цих фреймворків ми можемо вставити функції тестування, різні механізми виконання, звіти тощо.

У світі IDE (Integrated Development Environment) Java існує багато імен, які можуть надати основні інструменти та програмні застосунки розробникам, яким потрібно писати та тестувати програмне забезпечення.

Завдяки вбудованим редакторам вихідного коду, інструментам автоматизації збірки, інтелектуальним інструментам автоматизації, підтримці кількох мов та безлічі інших захоплюючих функцій і переваг. [11]

Як розробника, робота стає дуже вимогливою, енерговитратною та тривалою без відповідних інструментів. Однак, з правильним інструментом, це стає набагато легше і менш виснажливим. Отже, багато розробників покладаються на найкращі Java IDE (інтегровані середовища розробки), щоб виконувати роботу без великого стресу. Нижче наведені переваги та недоліки двох найпопулярніших середовищ розробки (табл. 2.2).

IntelliJ IDEA [12] є однією з професійних Java IDE для розробників, які шукають надійне, інтелектуальними та ергономічне середовище розробки для розробки Java-застосунків для веб- та мобільних пристроїв. Завдяки

розширеному інтелекту та готовому досвіду IntelliJ IDE пропонує розробникам важливі та комплексні інструменти, а також надає допомогу в кодуванні, щоб допомогти вам зробити більше. [12]

IntelliJ – це потужна і розширена IDE з вбудованими інструментами, які допоможуть вам залишатися продуктивними та керувати своїми проєктами від початку до кінця кращим та ергономічним способом, ніж раніше. Він підтримує веб- та мобільну розробку для різних мов, таких як HTML5, JavaScript, Angular JS, React, Android, PhoneGap і Node.js з широким спектром плагінів та інтегрованими елементами керування версіями.

Таблиця 2.2 – Порівняння IntelliJ IDEA та Eclipse як Java IDE

Характеристика	IntelliJ IDEA	Eclipse
Налагодження	Натиснути в будь-якому місті і система сама зрозуміє що необхідно запускати для налагодження.	Необхідно виділяти весь фрагмент коду, не більше не менше, щоб середовище зрозуміло що саме запускати.
Автозаповнення	Середовище розуміє що саме далі необхідно у коді (змінна, метод чи щось інше) та автоматично пропонує список при вводі	Середовище пропонує все що починається на введений символ в проєкті. Для того щоб побачити варіанти використовується комбінація клавіш.
Зміна коду, редагування	При заміні функціоналу, середовище пропонує, що, де і навіть як назвати змінну.	Трохи поступається, точно не пропонує назв новим змінним.

IntelliJ IDEA намагається бути інтелектуальним інтегрованим середовищем розробки. IntelliJ використовує методи, що дозволяють автоматизувати деякі завдання, і пропонує дії, які можуть полегшити роботу розробника. Завдяки цьому IntelliJ забезпечує готову інтеграцію з популярними фреймворками, які використовуються розробниками Java. Ця вбудована інтеграція різко зменшує кількість помилок і забезпечує чудову

синергію між різними технологіями, які використовуються для створення проєкту.

IntelliJ IDE пропонує важливі та комплексні інструменти для розробників, а також надає допомогу в кодуванні, щоб допомогти зробити більше. IntelliJ — це потужна та розширена IDE з вбудованими інструментами, які допоможуть залишатися продуктивними та керувати своїми проєктами від початку до кінця кращим і ергономічним способом, ніж раніше.

Плюси використання IntelliJ IDE [12]:

- він пропонує багато плагінів, які пропонують різні мови програмування;
- і дизайн, і розробка можуть бути виконані з легкістю;
- команди швидкого доступу насправді дуже корисні, коли необхідно працювати над дуже великою кодовою базою з кількома учасниками;
- IntelliJ допомагає з більшістю розумного завершення та переспрямування коду, коли працюєте з інтерфейсними або бекендовими (java) фреймворками;
- він має всі функції, які можна побажати, включаючи швидкий глобальний пошук файлів і класів, автозаповнення, інструменти рефакторингу та акуратний інтерфейс користувача для фреймворка UT;
- просте налаштування для налагодження;
- розумний дизайн інтерфейсу, включаючи комбінації клавіш та елегантне завершення коду.

При цьому мінусами IntelliJ є [12];

- повільне індексування, коли проєкт великий;
- UI Designer створює код Java, який залежить від бібліотек IntelliJ;
- підтримка мов, що не належать до JVM, є регулярною або відсутня;
- повільний час запуску.

IntelliJ випускається у версії «Ultimate» для корпоративних розробників, а також у «Індивідуальній» версії. Обидва вони оснащені

розширеними функціями продуктивності та розробки та потужними графічними інтерфейсами для створення надійних веб-застосунків. Eclipse IDE також є безкоштовною та з відкритим вихідним кодом з багатою бібліотекою, величезною спільнотою та величезною екосистемою плагінів і розширень.

Eclipse – це IDE OpenSource, яку можна використовувати для розробки програмного забезпечення будь-якою мовою, а не лише Java. Eclipse – надзвичайно гнучка IDE. Це одна з найбільш поширених ідей для розробки Java у світі. Якщо ми порівнюємо його з іншими IDE, його дуже легко використовувати. Розробка в Eclipse проста. Крім того, його інструмент для налагодження досить потужний. Eclipse має розширення та плагіни майже для всього. Оскільки він відкритий, то можна навіть при необхідності розробити власне розширення. [13]

Проект Eclipse має одночасний випуск щороку в кінці червня. У червні 2017 року була випущена версія Eclipse 4.7 (Oxygen). На даний момент IDE Eclipse може запропонувати багато. Завдяки таким функціям, як підтримка Gradle, контроль версій Git, редагування XML, інтеграція Apache Maven та інструмент списку завдань Mylyn, вбудований в Eclipse IDE, розробники завжди можуть бути в курсі оновлених бібліотек, а також функції для створення програм на основі Java, а також підтримка HTML5, Node.js, AJAX, CSS, PHP, JavaScript, Typescript тощо.

Декілька проєктів і компаній розширили IDE Eclipse або створили окремі програми (Eclipse Rich Client Platform) поверх фреймворка Eclipse. деякі з них включають

Плюсами використання Eclipse при розробці є [14, 15]:

- вбудований інкрементний компілятор, який надає вам звіти про помилки в режимі реального часу;
- багато плагінів – Devstyle, SVN, Maven тощо;
- інтеграція сервера - запустить Tomcat, JBoss всередині Eclipse;
- варіанти налаштування на будь-який смак;

- редактор ярликів, ви можете налаштувати відповідно до потреб;
- можна програмувати на більшості мов за допомогою Eclipse;
- помилки виправляються відносно швидко;
- інструмент налагодження досить потужний;
- Rich Client Platform: можна легко створювати багаті клієнтські програми за допомогою Eclipse, які сумісні з рідними програмами.

Мінуси [14,15]:

- це вимагає великих обсягів системних ресурсів, що іноді призводить до збою та втрати роботи;
- існує багато плагінів для одного і того ж, і стає важко вибрати один;
- немає сенсорного інтерфейсу.

Веб-автоматизація є найпопулярнішою в індустрії тестування автоматизації та з найбільшою екосистемою.

Провідною платформою є Selenium WebDriver, не дивно, що інструменти веб-автоматизації нового покоління, такі як TestProject, побудовані на основі Selenium. Хоча останнім часом ми бачимо все більше і більше інструментів веб-автоматизації, які не засновані на Selenium, наприклад: Cypress, Puppeteer, Nightwatch.js і TestCafe.

У будь-якому випадку, багато інструментів і рішень для автоматизації тестування (не тільки для Інтернету) засновані на протоколі WebDriver (раніше відомому як JSON Wire Protocol), інтерфейси WebDriver і WebElement, зрозуміти що таке RemoteWebDriver і всі функції, які їх реалізують: локатори, синхронізація, перемикачі, дії, об'єкти сторінки, взаємодіючі браузерери, JSExecutor, прослуховувачі подій. [16]

Інфраструктура автоматизації тестування це, мабуть, найважливіша частина будь-якої ролі розробника автоматизації (разом із вивченням мови програмування).

Інвестиції в надійну інфраструктуру автоматизації тестування є найдовшими і найважливішими для будь-якого проєкту, і включають в себе

модульні компоненти (так само, як створення головоломки або з'єднання деталей Lego).

Це інструменти, які потрібно буде вивчити, щоб написати інфраструктуру автоматизації тестування: сховище об'єктів, механізм збоїв, відновлення, система звітності, знімки екрана, перетворення екрана, зовнішні конфігураційні файли, підтримка кількох клієнтів, паралельні виконання, система реєстрації, підтримка бази даних, тестування драйверів даних, тестування на основі ключових слів, розробка на основі поведінки, планувальники часу.

Процес складання програмного забезпечення є найважливішою частиною процесу, і він відповідає за збірку всіх наших джерел (коду) і створення з них програмного забезпечення, наприклад, файлу .exe.

Існує чимало інструментів автоматизації збірки, найпоширенішими є Maven для Java.

При написанні проєкту автоматизації також потрібно створити таке програмне забезпечення, щоб в кінцевому підсумку виконання відбувалося з одного джерела (наприклад, за допомогою командного рядка або виконання файлу з розширенням bat. [16]

2.3 Висновки за розділом 2

Як показало дослідження в цьому розділі Python і Java є двома найпопулярнішими та надійними мовами програмування. Java, як правило, швидша та ефективніша, ніж Python, оскільки це скомпільована мова. Як інтерпретована мова, Python має простіший, більш стислий синтаксис, ніж Java. Він може виконувати ту ж функцію, що й Java, у меншій кількості рядків коду.

Також в розділі було порівняно і обрано середовище розробки автотестів, а саме IntelliJ IDE, через його розповсюдження у використанні саме для автоматизації тестування, та вже налагоджену і просту інтеграцію з

необхідними і найпопулярнішими інструментарієм Selenium WebDriver а також Maven, необхідними для побудови модульного проєкту та виконання тестових сценаріїв.

.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

3.1 Проєктування архітектури програмного продукту

На рисунку 3.1 наводиться архітектура взаємодії основних компонентів розроблюваного програмного продукту.

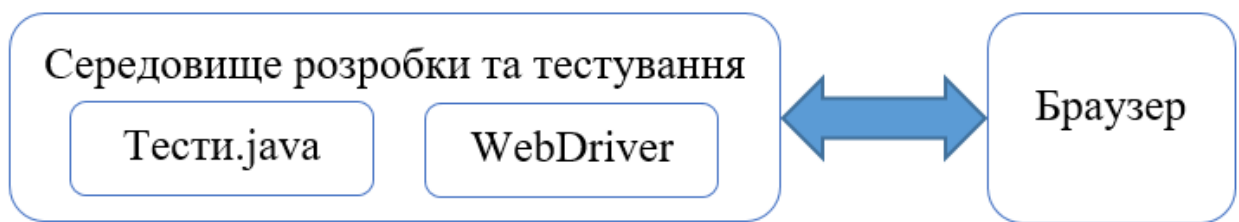


Рисунок 3.1 – Архітектура програмного продукту

Середовище IntelliJ IDEA виступає в якості середовища розробки і тестування. Дане середовище включає в себе тести написані мовою Java, а також підключений драйвер браузера Selenium WebDriver, який представляє собою бібліотеку, яка містить інструменти, які дозволяють керувати поведінкою браузера. І сам браузер в нашому проєкті для тестування візьмемо браузер Chrome.

3.2 Розробка структури програмного проєкту

Структура розроблюваного проєкту будується на основі структури тестованого веб-застосунку. Вебзастосунок містить 3 сторінки: сторінка Home, Login, Dashboard, та сторінка Allert. Кожна веб-сторінка програми розглядається як файл класу. Кожен файл класу містить лише відповідні елементи веб-сторінки. Таким чином структура програмного забезпечення тестування вебзастосунку виглядає схожою на сам вебзастосунок. Структура програмного проєкту зображена на рисунку 3.2.

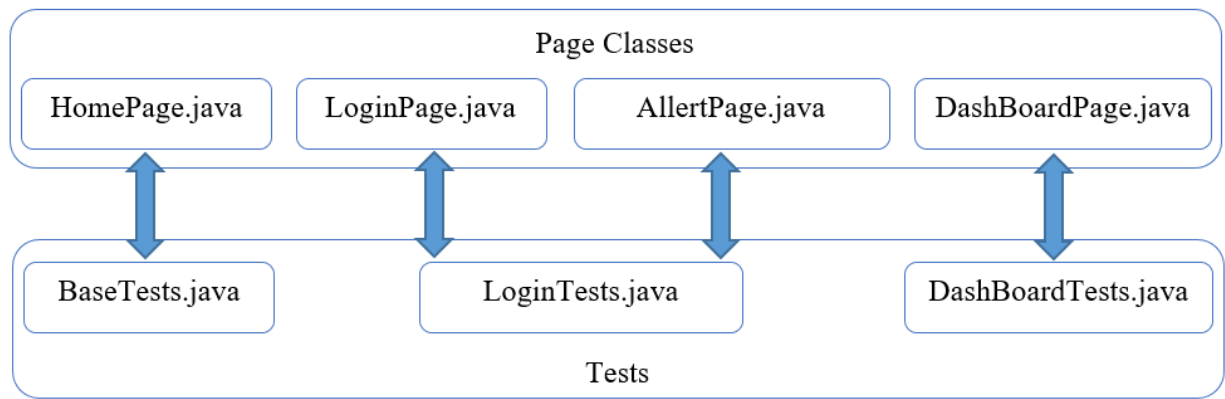


Рисунок 3.2 – Структура програмного проєкту

Кожен із файлів (класів), які знаходяться у блоці Page Classes містить елементи інтерфейсу вебсторінки або об'єкти, які присутні на цих екранах. Він також містить операції, які потрібно виконати над цими елементами.

Тести з блоку Tests використовують методи цього класу об'єктів сторінки, коли їм потрібно взаємодіяти з інтерфейсом користувача цієї сторінки.

3.3 Діаграма послідовності виконання програми

Для того щоб приблизно розуміти, яка саме структура проєкту буде розроблятися та які класи та методи необхідно розробити, спершу було розроблено алгоритм роботи програми, що представляє собою сукупність виконуваних тестів. На рисунку 3.3 зображено алгоритм роботи програми у вигляді діаграми послідовності дій.

Крок 1. Імпорт бібліотек. Імпорт бібліотеки Selenium WebDriver для роботи з браузером, TestNG для перевірки правильності роботи метода, наприклад такі методи як `assertEquals()`.

Крок 2. Налаштування браузера для роботи. На цьому кроці визначається параметри браузера та URL самої початкова вебсторінка для тестування.

Крок 3. Перехід зі сторінки Home на сторінку Login.

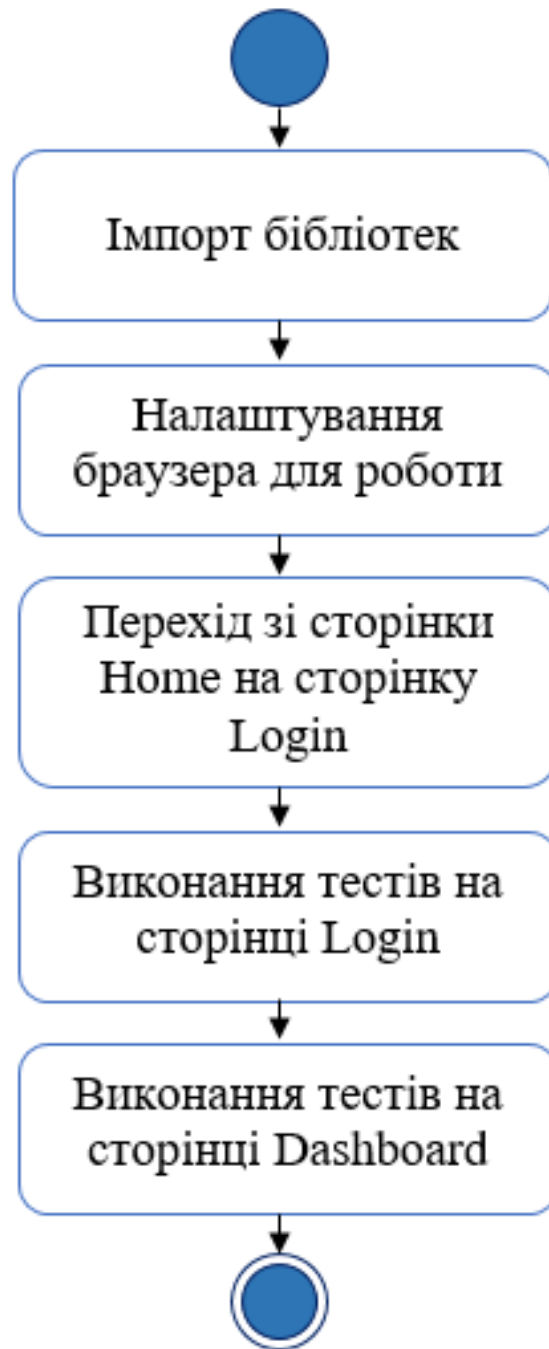


Рисунок 3.3 – Алгоритм роботи програми

Крок 4. Виконання тестів на сторінці Login. Перевірка успішного і негативного сценарію логіна на сторінці Login.

Крок 5. Виконання тестів на сторінці Dashboard. Перевірка існування заданого користувача на сторінці Dashboard з випадającego списку користувачів.

3.4 Проектування класів та методів

Для всіх класів було спроектовані методи (функції). Всі методи та опис того що вони виконують наводяться у таблиці 3.1-3.6.

Таблиця 3.1 – Методи класу BaseTests

Метод	Опис
public void SetUp()	Відкриття браузера, відкриття початкової вебсторінки для тестування, налаштування вигляду вебсторінки, максимізація на весь екран.
public void testDown()	Вихід та закриття вебсторінки та браузера.

Таблиця 3.2 – Методи класу LoginTests

Метод	Опис
public void testSuccessful Login()	Введення валідних параметрів в поле логіну та пароля та натискання клавіши Login. Та перевірка успішності завантаження відповідної сторінки після логіна.
public void testUnsuccessful Login()	Введення не валідних параметрів в поле логіну та пароля та натискання клавіши Login. Та перевірка, що користувач залишився на відповідній сторінці Login.

Таблиця 3.4 – Методи класу HomePage

Метод	Опис
public LoginPage clickFormLogin()	Визначення змінною за якою необхідно шукати посилання на сторінку Login, щоб в подальшому перейти на неї.
private void clickLinck(String linkText)	Саме дія, моделювання натискання правою кнопкою на посилання на сторінку Login.

Таблиця 3.3 – Методи класу DashBoardTests

Метод	Опис
public void testSelectUser()	Перевірка чи існує шуканий користувач в випадяючому списку на сторінці Dashboard.

Таблиця 3.5 – Методи класу LoginPage

Метод	Опис
public void setUsername(String username)	Пошук поля для введення ім'я користувача, та введення відповідного імені у це поле.
public void setPsswrд(String psswrд)	Пошук поля для введення пароля, та введення відповідного пароля у це поле.
public String getLoginPageText()	Перевірка чи знаходиться користувач на сторінці Login шляхом пошуку елементу назви сторінки.
public AlertAreaPage clickLoginButtn()	Імітації дії натискання правої кнопки миші на кнопку Login.

Таблиця 3.6 – Методи класу DashBoardPage

Метод	Опис
public void selectFromDashBoard(String user)	Знаходить заданого користувача в випадяючому списку.
public List<String> getSelectedUser()	Повертає список користувачів з заданим ім'я з випадяючого списку на сторінці.
private Select findDashBoardUser()	Пошук поля з випадяючим списком користувачів.

3.5 Висновки до розділу 3

В цьому розділі було спроектовано загальну архітектуру програмного продукту. Спроектована структура програмного проєкту, яка складається з класів сторінок та класів тестів. Кожен тест використовує методи класу сторінки для перевірки необхідних даних.

Розроблено алгоритм роботи програми у вигляді діаграми послідовності дії, яка описує послідовність роботи тесту.

Також наводяться методи розроблених класів та детальний опис того, що саме ці методи виконують.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Інструкція по експлуатації програми

Першим кроком в експлуатації програми є запуск середовища та налаштування, які необхідно зробити.

Після завантаження та відкриття середовища розробки IntelliJ IDEA, необхідно розархівувати проєкт програми TestDashbordSL.rar. Далі у середовищі розробки відкрити цей проєкт, та перевірити щоб у файлі pom.xml були визначені залежності як на рисунку 4.1. А також щоб в папці resources знаходився файл драйвер для браузера Chrome під назвою chromedriver.

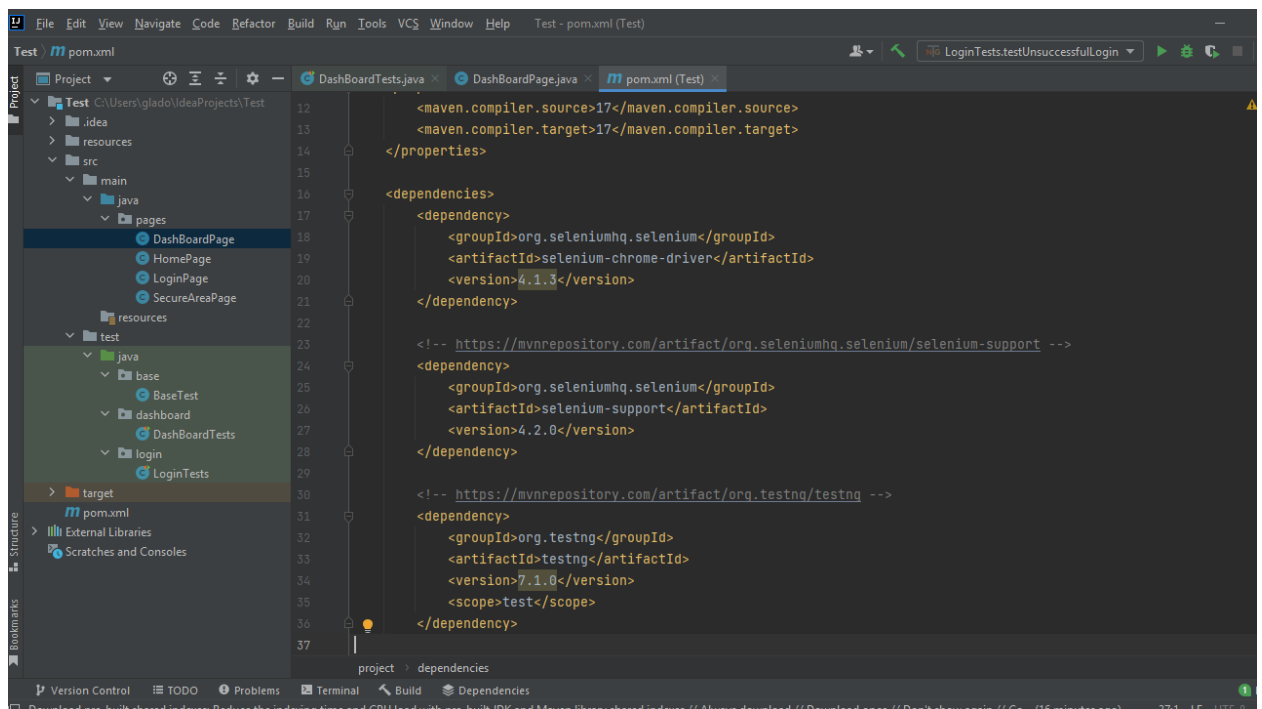


Рисунок 4.1 – Налаштування залежностей для проєкту

Для тестування сторінки Login яка зображена на рисунку 4.2 необхідно відкрити файл проєкту, який називається LoginTests, який знаходиться за шляхом test->java->login->LoginTests.java, в правій частині вікна середовища IntelliJ (рис. 4.3). Для виконання успішного і негативного тесту необхідно біля кожного тесту натиснути зелену кнопку Run test.

welcome to
SPOTLOGIC™

Email *

Password *

Select Server *

Heroku

Login

[Forgot password?](#)

support@spotlogic.com

Рисунок 4.2 – Сторінка тестування Login

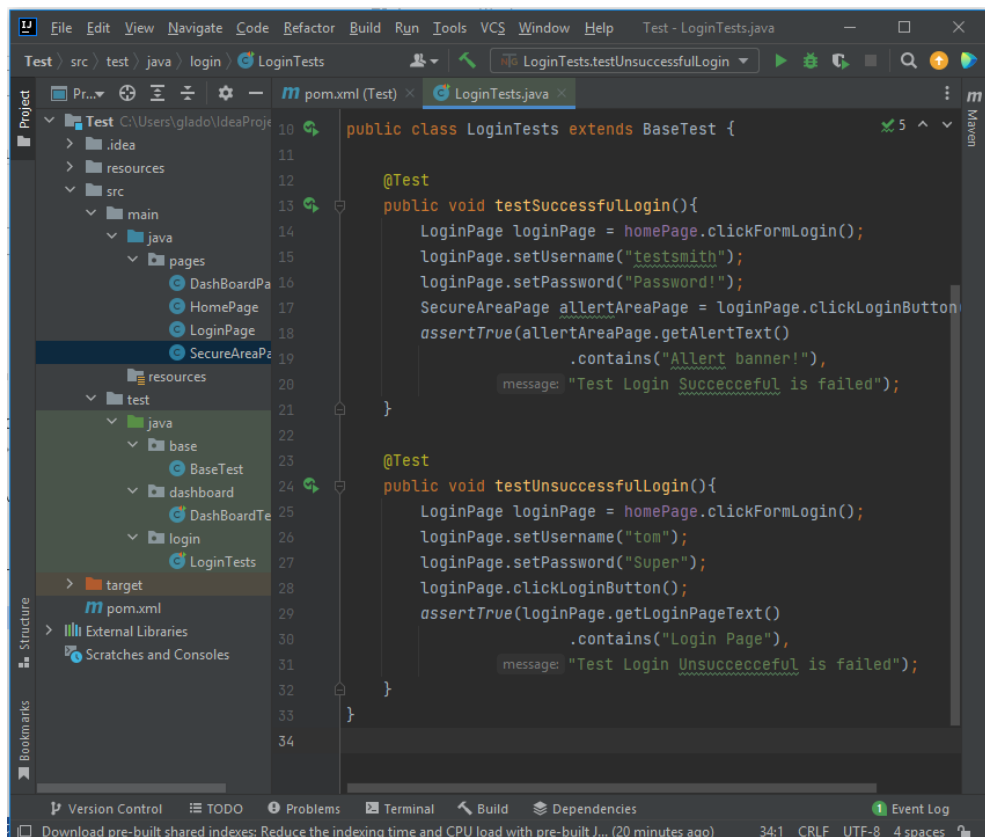


Рисунок 4.3 – Тест LoginTests

Після успішного сценарію тестування сторінки Login далі переходимо до наступної сторінки. Для тестування наступної сторінки Dashboard, а саме поля з випадаючим списком користувачів, яка зображена на рисунку 4.4.

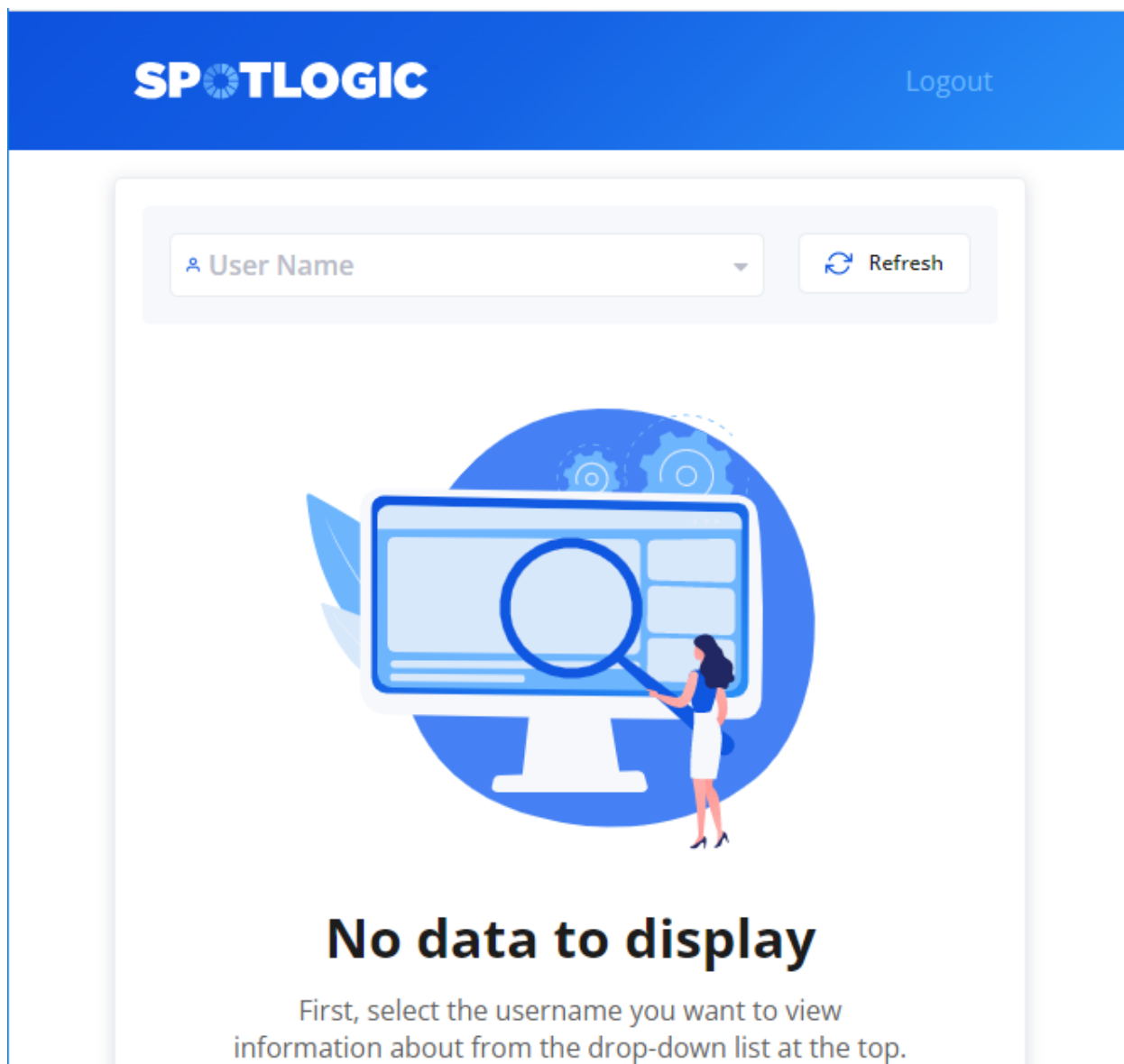


Рисунок 4.4 – Сторінка для тестування Dashboard

Для цього необхідно запустити .java файл тест під назвою DashboardTests, який розміщений за шляхом test->java->dashboard->DashboardTests.java (рис. 4.5).

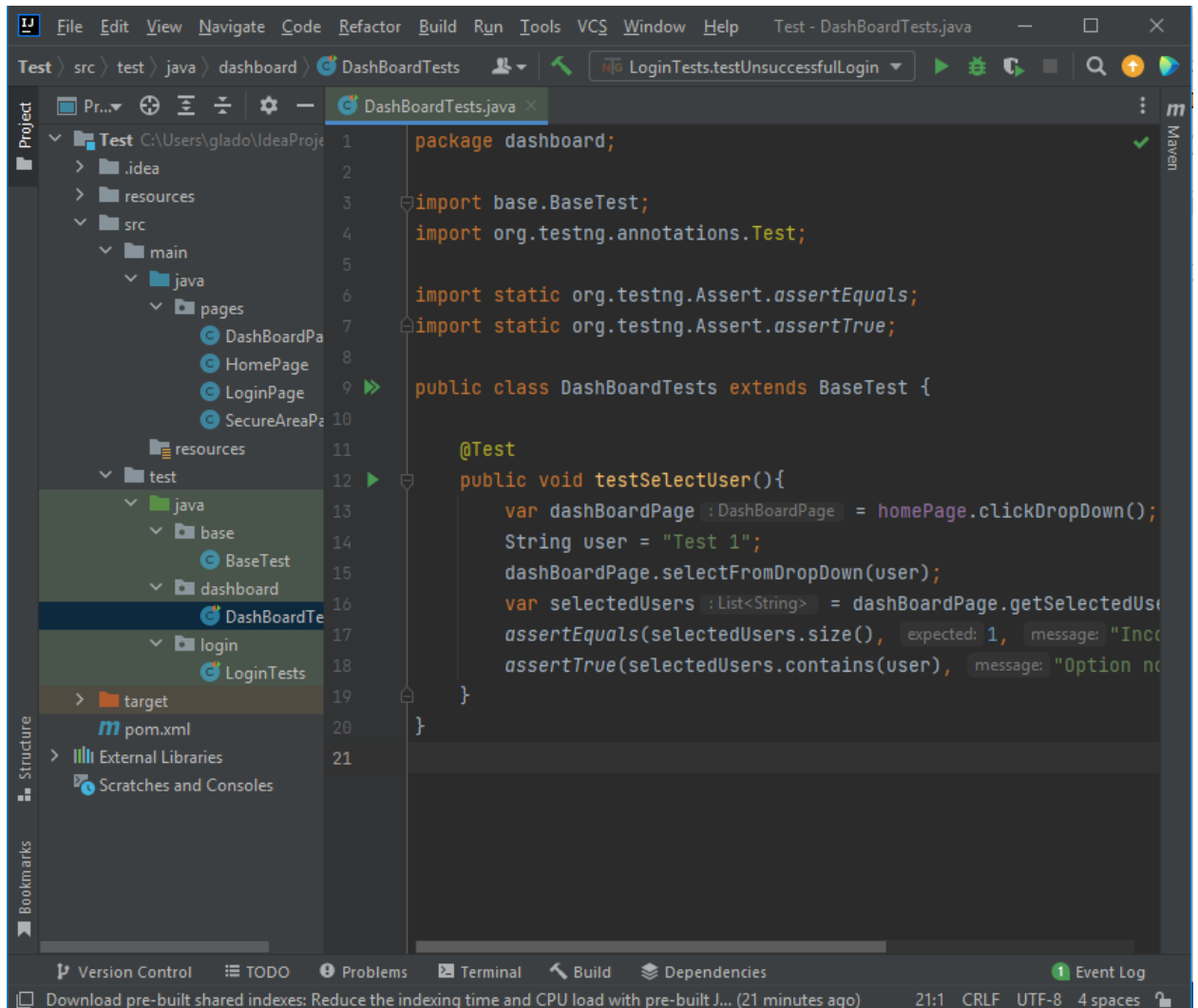


Рисунок 4.5 – Клас тесту DashBoardTests

4.2 Функції системи

Програмний продукт призначений для виконання таких функцій:

- автоматично запускати браузер із зазначеної сторінки;
- виконувати тестові сценарії успішного та негативного тестів авторизації на сайті;
- виконувати пошук та перевірку необхідного користувача з випадаючого списку;
- виводити повідомлення та результати тестування.

4.3 Результати тестування

Після виконання кожного тесту середовище відображає результати та повідомлення користувачеві. Крім того, можна подивитися час виконання тесту. За результатами тесту які зображені на рисунках 4.6 – 4.8 видно що загалом на виконання цих тестів було витрачено 15 секунд. В свою чергу виконання таких тестів вручну тестувальником зайняло би в 4-8 раз більше часу.

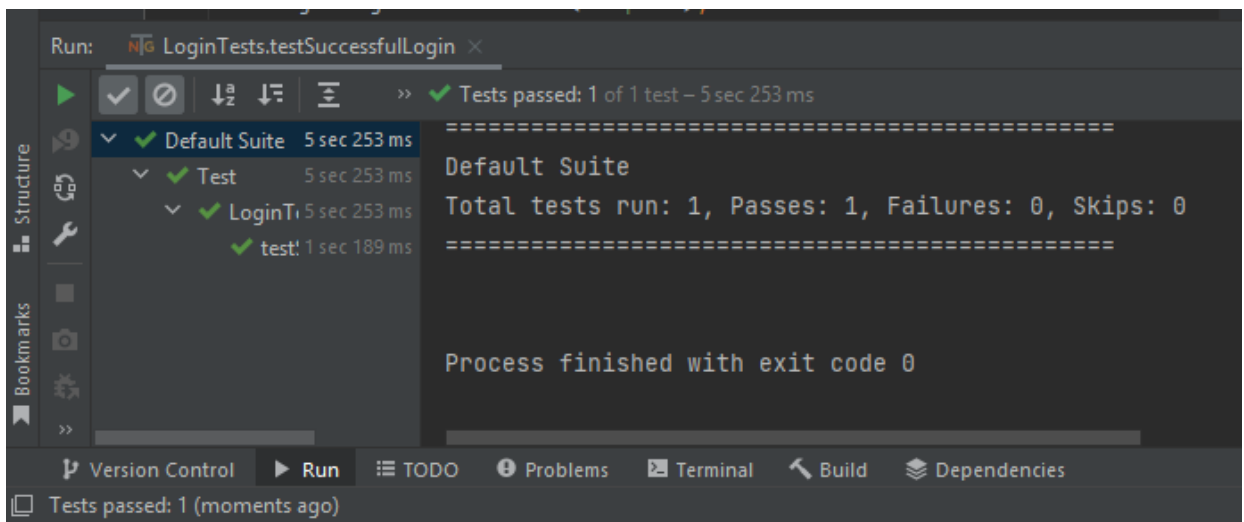


Рисунок 4.6 – Результати виконання позитивного тесту LoginTests

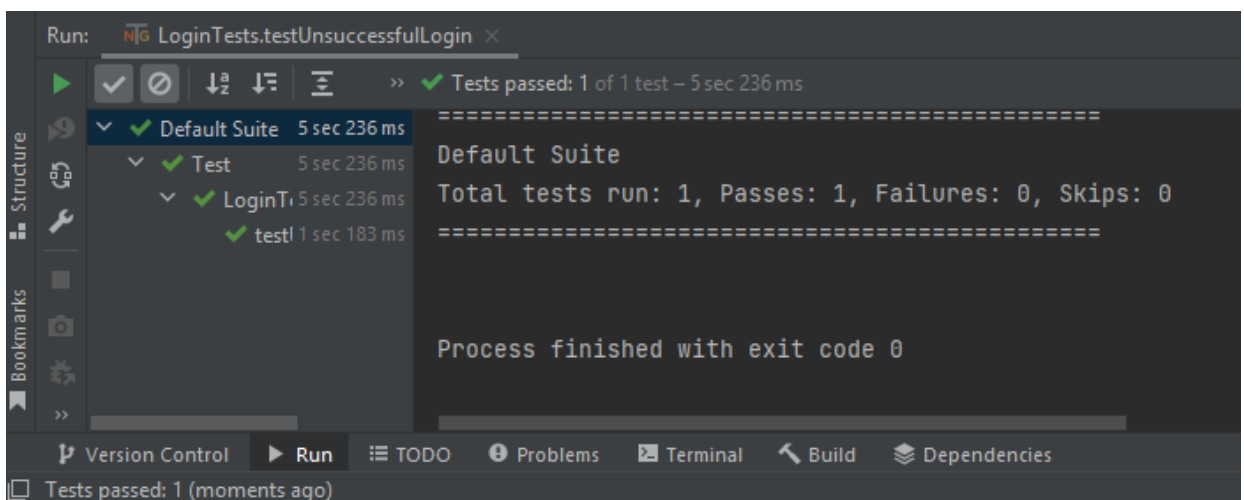


Рисунок 4.7 – Результати виконання негативного тесту LoginTests

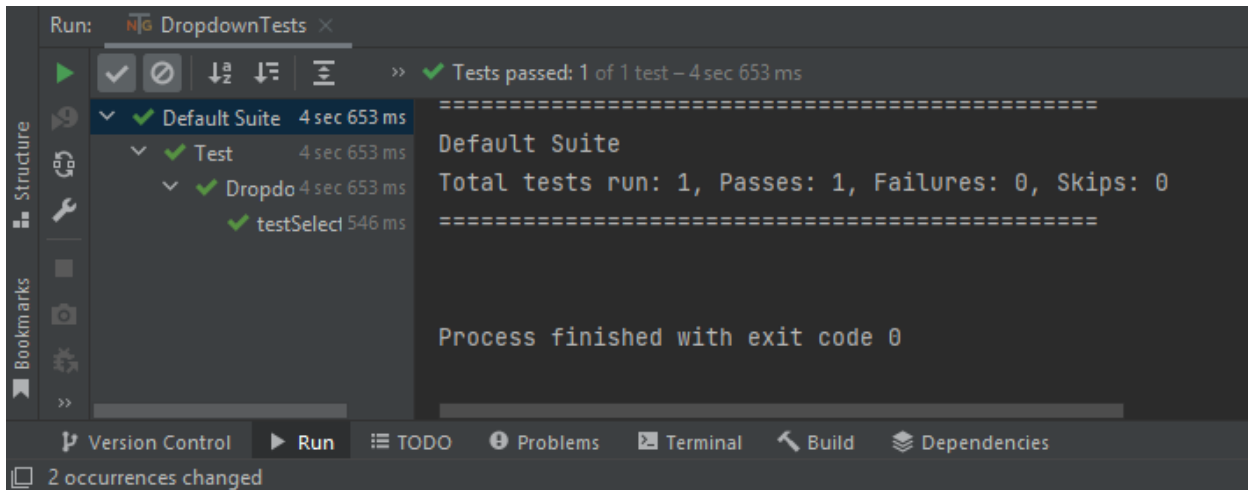


Рисунок 4.7 – Результати виконання тесту DashBoardTests

4.6 Висновки до розділу 4

У даному розділі описано детальну інструкцію з запуску та експлуатації програми. Описано основні функції призначення програми.

Крім того наводяться результати тестування за якими можна зробити висновок, що впровадження засобів автоматизації дозволяю прискорити сам процес тестування до 8 разів.

ВИСНОВКИ

У міру ускладнення коду, додавання нового функціоналу та виправлення багів, процес тестування займатиме все більше часу. Команда мануального тестування фізично зможе встигати за ритмом команди розробників. Адже перевірка нового функціоналу – це не лише функціональні тести, а й ресурсномісткі регресійні прогони. Використання автоматизованого тестування дозволяє зняти з команди мануального тестування досить об'ємну частину регресійних прогонів. Але слід пам'ятати, що повна автоматизація, ні стовідсоткове ручне тестування не дадуть гарантії ідеального результату. Оптимальним підходом у разі є розумний баланс.

Завдання автоматизації — у створенні автоматизованих сценаріїв, а й у безпосередньої інтеграції у процес розробки ПЗ.

Використання зв'язки автоматизованого та ручного тестування та тісна взаємодія з командою розробників від початку процесу роботи над проєктом до його реалізації підвищить якість кінцевого продукту.

Проаналізовано переваги та недоліки впровадження автоматизації в процес тестування, згідно з яким автоматизація вимагає значних витрат на початку створення тестів, але в процесі виконання тестів має більше переваг перед річним тестуванням.

Виконано аналіз існуючих методів тестування. Обрано інструментів розробки, а саме мова програмування Java та середовище розробки IntelliJ Idea. Вона дозволяє швидше та якісніше писати та змінювати код, підказує відповідні імена, знаходить відповідні методи. Не вимагає точно виділити вираз, а тому, де знаходиться курсор, вгадує, що користувач хоче зробити і як хоче це назвати. IDEA передбачає і нагадує.

Була спроектована архітектура програмного продукту та структура програмного проєкту, яка включає в себе дві категорії файлів, а саме класи сторінок та класи тестів які використовують методи класів сторінок для виконання тестування.

Розроблено алгоритм роботи програмного забезпечення, та на основі цього алгоритму та структури програми були розроблені класи та методи програмного забезпечення.

За результатами проведеного тестування автоматизації тестових сценаріїв вебзастосунку дозволило виконати саме тестування до 8 разів швидше ніж якби ці тести виконувались вручну.

В результаті виконання кваліфікаційної роботи бакалавра було розроблено програмне забезпечення автоматизації тестування вебзастосунку, що дозволяє виконувати тестування більш ефективніше та прискорити цей процес тестування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 What is automation testing? [Electronic source]. – Access mode: <https://www.globalapptesting.com/blog/what-is-automation-testing>
- 2 Как и когда внедрять автотесты: преимущества и недостатки автоматизации тестирования [Электрон. ресурс]. – Режим доступа: <https://quality-lab.ru/blog/как-и-когда-внедрять-автотесты-преиму/>
- 3 Преимущества и недостатки автоматизированного тестирования [Электрон. ресурс]. – Режим доступа: <https://qaat.ru/preimushhestva-i-nedostatki-avtomatizirovannogo-testirovaniya/>
- 4 Автоматизированное тестирование: что это? Краткое учебное пособие. [Электрон. ресурс]. – Режим доступа: https://logrocon.ru/news/automation_testing
- 5 Методы автоматизированного тестирования [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/post/489424/>
- 6 Автоматизированное тестирование [Электрон. ресурс]. – Режим доступа: <https://gist.github.com/codedokode/a455bde7d0748c0a351a>
- 7 Automation Testing for Beginners – The Ultimate Guide [Electronic source]. – Access mode: <https://blog.testproject.io/2020/03/26/automation-testing-for-beginners-ultimate-guide>
- 8 Почему программисты продолжают использовать многословный Java, хотя есть лаконичный Python [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/company/skillbox/blog/443412/>
- 9 Python vs. Java Performance [Electronic source]. – Access mode: <https://www.snaplogic.com/glossary/python-vs-java-performance>
- 10 Java vs Python: Key Differences Between Java and Python? [Electronic source]. – Access mode: <https://www.guru99.com/java-vs-python.html>
- 11 7 Best Java IDEs – Compared [Electronic source]. – Access mode: <https://www.dunebook.com/5-best-ide-for-java-development/>

12 IntelliJ IDEA: функциональная и эргономичная IDE для разработки на Java от JetBrains [Электрон. ресурс]. – Режим доступа: <https://www.jetbrains.com/ru-ru/idea/>

13 Eclipse IDE for Java Developers, Eclipse Packages [Electronic source]. – Access mode: <https://www.eclipse.org/downloads/packages/release/oxygen/3a/eclipse-ide-java-developers>

14 IDE Eclipse: за и против от ведущих программистов [Электрон. ресурс]. – Режим доступа: <https://proglib.io/p/ide-eclipse-za-i-protiv-ot-vedushchih-programmistov-2019-11-02>

15 Обзор Eclipse Java: среда разработки “под себя” [Электрон. ресурс]. – Режим доступа: <https://javarush.ru/groups/posts/2359-obzor-eclipse-java-sreda-razrabotki-pod-sebja>

16 Почему IDEA лучше Eclipse [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/post/112749/>

ДОДАТОК А
Текст програми

```
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.Select;

import java.util.List;
import java.util.stream.Collectors;

public class DashBoardPage {

    private WebDriver driver;
    private By ddownfield = By.id("DashBoard");

    public DashBoardPage(WebDriver driver){
        this.driver = driver;
    }

    public void selectFromDashBoard(String user){
        findDashBoardUser().selectByVisibleText(user);
    }

    public List<String> getSelectedUser(){
        List<WebElement> selectedUsers =
            findDashBoardUser().getAllSelectedUsers();
        return selectedUsers.stream().map(e->e.getText()).collect(Collectors.toList());
    }
}
```

```

private Select findDashBoardUser(){
    return new Select(driver.findUser(ddownfield));
}
}

```

```

package pages;

```

```

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

```

```

public class HomePage {
    private WebDriver driver;

    public HomePage(WebDriver driver) {
        this.driver = driver;
    }

```

```

    public LoginPage clickFormLogin(){
        clickLinck("Form Login");
        return new LoginPage(driver);
    }

```

```

    public DashBoardPage clickDashBoard(){
        clickLinck("DashBoard");
        return new DashBoardPage(driver);
    }

```

```

    private void clickLinck(String linkText){
        driver.findUser(By.linkText(linkText)).click();
    }

```

```

    }

}

package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class LoginPage {
    private WebDriver driver;
    private By psswrdfield = By.id("psswrdfield");
    private By loginButton = By.cssSelector("#loginbtn");
    private By textLoginPage = By.id("example");
    private By usernameField = By.id("username");

    public LoginPage(WebDriver driver){
        this.driver = driver;
    }

    public void setUsername(String username){
        driver.findUser(usernameField).sendKeys(username);
    }

    public void setPassword(String password){
        driver.findUser(passwordField).sendKeys(password);
    }

    public String getLoginPageText(){
        return driver.findUser(textLoginPage).getText();
    }
}

```

```

    }
    public AllertAreaPage clickLoginButtn(){
        driver.findUser(loginButtn).click();
        return new AllertAreaPage(driver);
    }
}

```

```
package pages;
```

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
```

```

public class AllertAreaPage {

    private WebDriver driver;
    private By statysAlert = By.id("flash");

    public AllertAreaPage(WebDriver driver){
        this.driver = driver;
    }

    public String getAlertText(){
        return driver.findUser(statysAlert).getText();
    }
}

```

```
package base;
```

```
import pages.HomePage;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;

public class BaseTest {
    private WebDriver driver;
    protected HomePage homePage;

    @BeforeClass
    public void setUp() {

        System.setProperty("webdriver.chrome.driver",
"resources/chromedriver.exe");
        driver = new ChromeDriver();
        driver.get("https://spotlogic-dealdashboard-prod.herokuapp.com/");

        driver.manage().window().maximize();
        System.out.println(driver.getTitle());

        homePage = new HomePage(driver);
    }

    @AfterClass
```

```

    public void tearDown(){
        driver.quit();
    }
}

package DashBoard;

import base.BaseTest;
import org.openqa.selenium.WebDriver;
import org.testng.annotations.Test;

import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertTrue;

public class DashBoardTests extends BaseTest {

    @Test
    public void testSelectUser(){

        var DashBoardPage = homePage.clickDashBoard();
        String user = "User 1";

        DashBoardPage.selectFromDashBoard(user);
        var selectedUsers = DashBoardPage.getSelectedUser();

        assertEquals(selectedUsers.size(), 1, "Incorrect number of selections");
        assertTrue(selectedUsers.contains(user), "User not selected");
    }
}

```

```

package login;

import base.BaseTest;
import org.testng.annotations.Test;
import pages.LoginPage;
import pages.AlertAreaPage;

import static org.testng.Assert.*;

public class LoginTests extends BaseTest {

    @Test
    public void testSuccessfulLogin(){

        LoginPage loginPage = homePage.clickFormLogin();
        loginPage.setUsername("testsmith");
        loginPage.setPsswrд("SuperSecretPsswrд!");
        AlertAreaPage alertAreaPage = loginPage.clickLoginButtn();

        assertTrue(alertAreaPage.getAlertText()
            .contains("Alert banner!"),
            " Test Login Successful is failed ");
    }

    @Test
    public void testSuccessfulLogin(){

        LoginPage loginPage = homePage.clickFormLogin();
        loginPage.setUsername("tom");
    }

```

```

loginPage.setPsswrд("Super");
loginPage.clickLoginButtn();

assertTrue(loginPage.getLoginPageText()
            .contains("Login Page"),
            "Test Login Unsuccessful is failed");
}

```

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>Test</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>

```

```
<artifactId>selenium-chrome-driver</artifactId>
<version>4.1.3</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-
support -->
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-support</artifactId>
  <version>4.2.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.testng/testng -->
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.1.0</version>
  <scope>test</scope>
</dependency>

</dependencies>

</project>
```

ДОДАТОК Б
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

кваліфікаційна робота бакалавра

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ
DEVELOPMENT OF WEB APPLICATION TESTING SOFTWARE

Виконав: ст.гр. КНТ-218

Керівник: к.т.н., доц.

Авраменко І.Ю.

Гладкова О.М.

1

Рисунок Б.1 – Слайд 1

Мета та завдання

Об'єкт дослідження - тестування вебзастосунків.

Предмет дослідження - автоматизація тестування вебзастосунків.

Мета роботи - розробити програмне забезпечення для автоматизації тестування вебзастосунку.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- ▶ виконати дослідження існуючих методів тестування та інструментів розробки;
- ▶ розробити структуру проєкту;
- ▶ розробити алгоритм роботи програмного забезпечення;
- ▶ розробити програмне забезпечення.

2

Рисунок Б.2 – Слайд 2

Переваги та недоліки автоматизованого тестування

Переваги	Недоліки
Прискорює та спрощує тестування	Витрачений час на написання та налагодження автотестів.
В довгострокових проєктах витрачені інвестиції на автотести повністю себе окупають.	Великі інвестиції на самому початку проєкту.
Зменшує людський фактор при перевірці.	При зміні логіки проєкту необхідна також повна зміна логіки автотестів.
Можлива краща звітність та більш повне оформлення.	

3

Рисунок Б.3 – Слайд 3

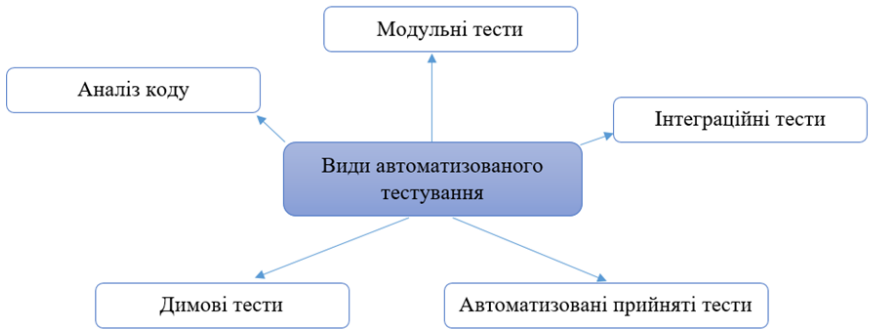
Тести які підходять і не підходять для автоматизації

Тести які підходять	Тести які не підходять
Тести, які регулярно або часто повторюються.	Тести що не пройшли ручне тестування жодного разу.
Тести які стомлюючі та важкі для виконання людиною.	Тести логіка яких дуже часто підпадає під зміни.
Тести на проведення яких необхідно багато часу.	Тести які виконуються лише одноразово.

4

Рисунок Б.4 – Слайд 4

Види автоматизованого тестування



5

Рисунок Б.5 – Слайд 5

Вибір мови програмування

Параметр	Java	Python
Вивчення	Порівняно складніша.	Легко навчатися.
Дужки, відступи	Він використовує фігурні дужки для визначення початку та кінця кожної функції та визначення класу.	Python використовує відступи для розділення коду на блоки коду.
Швидкість	Програма Java працює повільно в порівнянні з Python.	Програми на Python працюють швидше, ніж на Java.
Портативність	Будь-який комп'ютер або мобільний пристрій, який може запускати віртуальну машину Java.	У порівнянні з Java, Python менш переносимий.
Найкращі характеристики	Чудові бібліотеки. Широко використовуваний. Відмінна оснастка Доступно багато документів.	Читаний код Швидкий розвиток Гарний код
Найкраще використовувати для	Java найкраще підходить для настільних програм із графічним інтерфейсом користувача, систем числових обчислень, програм машинного вбудовування, служб веб-додатків тощо.	Python чудово підходить для наукових і навчання тощо.
Підтримка бази даних	Java забезпечує стабільне підключення	Python пропонує слабке підключення.

6

Рисунок Б.6 – Слайд 6

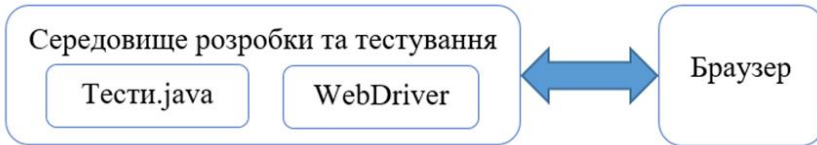
Вибір середовища розробки

Характеристика	IntelliJ IDEA	Eclipse
Налагодження	Натиснути в будь-якому місті і система сама зрозуміє що необхідно запускати для налагодження.	Необхідно виділяти весь фрагмент коду, не більше не менше, щоб середовище зрозуміло що саме запускати.
Автозаповнення	Середовище розуміє що саме далі необхідно у коді (змінна, починається на введений метод чи щось інше) та символ в проєкті. Для того щоб автоматично пропонує список при вводі	Середовище пропонує все що побачити варіанти використовується комбінація клавіш.
Зміна коду, редагування	При заміні функціоналу, середовище пропонує, що, де і навіть як назвати змінну.	Трохи поступається, точно не пропонує назв новим змінним.

7

Рисунок Б.7 – Слайд 7

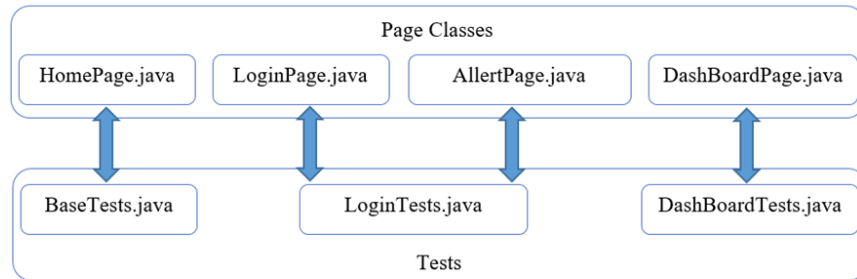
Архітектура програмного продукту



8

Рисунок Б.8 – Слайд 8

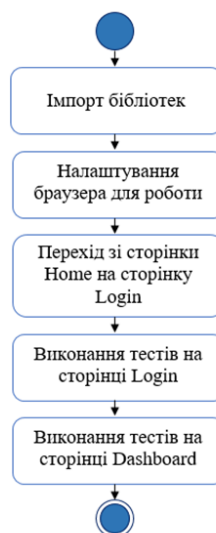
Структура програмного проєкту



9

Рисунок Б.9 – Слайд 9

Діаграма алгоритму роботи програми



10

Рисунок Б.10 – Слайд 10

Розроблені методи

Методи класу LoginPage.java

Метод	Опис
public void setUsername(String username)	Пошук поля для введення ім'я користувача, та введення відповідного імені у це поле.
public void setPassword(String password)	Пошук поля для введення пароля, та введення відповідного пароля у це поле.
public String getLoginPageText()	Перевірка чи знаходиться користувач на сторінці login шляхом пошуку елемента назви сторінки.
public SecureAreaPage clickLoginButton()	Імітації дії натискання правої кнопки миші на кнопку Login.

Методи класу LoginTests.java

Метод	Опис
public void testSuccessful Login()	Введення валідних параметрів в поле логіну та пароля та натискання клавіші Login. Та перевірка успішності завантаження відповідної сторінки після логіна.
public void testUnsuccessful Login()	Введення не валідних параметрів в поле логіну та пароля та натискання клавіші Login. Та перевірка, що користувач залишився на відповідній сторінці Login.

Рисунок Б.11 – Слайд 11

Експлуатація програми

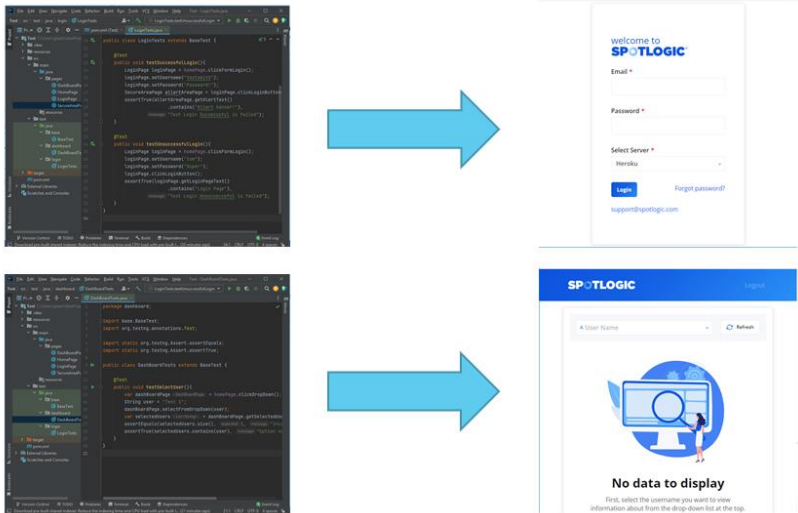
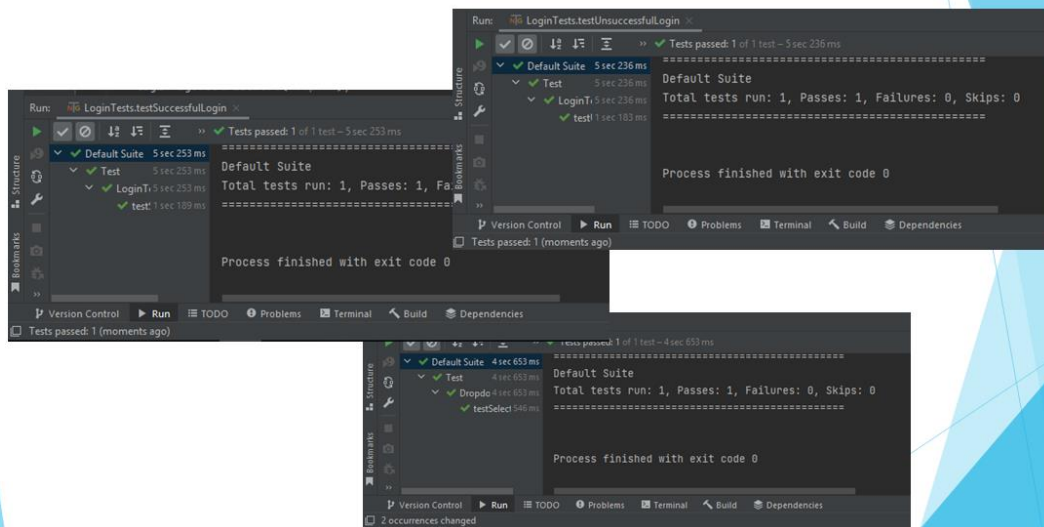


Рисунок Б.12 – Слайд 12

Результати тестування



13

Рисунок Б.13 – Слайд 13

Висновки

В результаті виконання кваліфікаційної роботи бакалавра було:

- ▶ Проаналізовано переваги та недоліки впровадження автоматизації в процес тестування.
- ▶ Виконано аналіз існуючих методів тестування.
- ▶ Обрано інструментів розробки.
- ▶ Була спроектована архітектура програмного продукту та структура програмного проєкту.
- ▶ Розроблено алгоритм роботи програмного забезпечення, та на основі цього алгоритму та структури програми були розроблені класи та методи програмного забезпечення.
- ▶ За результатами проведеного тестування розроблене програмне забезпечення автоматизації тестування вебзастосунку дозволило виконати саме тестування до 8 разів швидше ніж якби ці тести виконувались вручну.

14

Рисунок Б.14 – Слайд 14