

МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: “Розробка фреймворку для автоматичного тестування веб додатків
на базі Selenium WebDriver та TestNG”

Здобувача вищої освіти другого (магістерського) рівня,
II курсу, групи КМІ-22-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»



(підпис)

Прибитька Євгенія
Анатолійовича
(прізвище, ім'я, по батькові)

Науковий керівник



(підпис)

Редич Олександр
Володимирович, к.е.н.,
доцент
(прізвище, ім'я, по батькові, науковий
ступінь, вчене звання)

Керівник
освітньо-професійної
програми



(підпис)

Горбовий Артур Юліанович,
д.т.н. професор
(прізвище, ім'я, по батькові, науковий
ступінь, вчене звання)

Допущено до захисту за рішенням кафедри
протокол від «28» грудня 2023 № 8

Завідувач кафедри



(підпис)

Філоненко Михайло
Миколайович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові,
науковий ступінь, вчене звання)

Ірпінь – 2024

Реєстрація № 2

29.01.2024
(дата)


(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

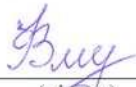
29.01.2024 
(дата) (підпис)

Результат захисту

5 (відмінно)
(оцінка)

12.02.2024 
(дата) (підпис)

Голова комісії:


(підпис)

В.М. Франчук

Члени комісії:


(підпис)

А.Ю. Горбовий


(підпис)

О.В. Редич


(підпис)

В.О. Ніжегородцев

БЛАНК ОЦІНЮВАННЯ ВИПУСКНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		12.02.2024 Франчук В.М. Горбовий А.Ю. Редич О.В. Ніжегородцев В.О. Суховерхівка Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	9		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	5		
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	47		

Оцінювання кваліфікаційної роботи науковим керівником - 43 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підписи)
90	A	5 (відмінно)	12.02.2024	Франчук В.М. Горбовий А.Ю. Редич О.В. Ніжегородцев В.О. Суховерхівка Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 122 «Комп'ютерні науки»

Освітня програма: «Інформаційні управляючі системи та технології в економіці»

Затверджую

Завідувач кафедри  Філоненко М.М.

« 04 » 04 2023

Завдання на кваліфікаційну роботу
здобувачу вищої освіти

Прибильку Євгенію Анатолійовичу

1. Тема дипломної роботи

Розробка фреймворку для автоматичного тестування веб долатків на базі Selenium WebDriver та TestNG

Затверджена наказом ректора від 04 квітня 2023 р. №362

2. Строк здачі закінченої роботи: 20 грудня 2023

3. Мета роботи: провести дослідження технологій розробки фреймворків для автоматичного тестування програмного забезпечення

Об'єкт дослідження: технології розробки фреймворків для автоматичного тестування програмного забезпечення

Предмет дослідження: інструментальні засоби та методологічні підходи до розробки фреймворку для автоматизації тестування веб-сайтів Selenium WebDriver та TestNG

Орієнтовна база дослідження Відкриті ресурси та технології розробки засобів тестування ПО, веб ресурси ДПУ

4. Вихідні дані до роботи: Відкриті дані про інструментальні засоби та методологічні підходи до розробки застосовань для автоматизованого тестування

5. Перелік графічного матеріалу: рисунки, формули, що пояснюють суть застосованих методів, засобів та технологій; Структурні схеми та, таблиці, графіки, результати апробації та тестування.

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

Розділ 1. Автоматизоване тестування як етап в процесі забезпечення якості продукту

1.1.Поняття автоматизованого тестування. Переваги і недоліки автоматизованого тестування

1.2.Огляд існуючих інструментів автоматизованого тестування

1.3. Критерії вибору компонентів фреймворку

Розділ 2. Складові частини фреймворку

2.1. Архітектура фреймворку. його компоненти

2.2. Огляд Selenium та його можливостей

2.3. Огляд TestNG та його місце у побудові фреймворку

2.4. Поняття та засоби Continuous Integration

Розділ 3. Практичне застосування фреймворку

3.1. Патерни побудови логіки тестів

3.2. Програмні компоненти та процес побудови фреймворку

3.3. Застосування принципів ООП та TestNG в розробці фреймворку

3.4. Вибір тестових даних та сценаріїв проходження тестів

7. Календарний план виконання роботи

№ п/п	Назва етапів випускної кваліфікаційної (дипломної) роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури ДР	До 27.12.2022 р.	26.12.2022 р.
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26. 06. 2023 р.	26.06.2023 р.
3	II розділ (аналіз експериментальної частини про і методики досліджень)	23. 10.2023 р.	20.10.2023 р.
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	30. 11.2023 р.	30.11.2023 р.
5	Висновки	10. 12.2023 р.	10.12.2023 р.
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	20.01.2024 р.	20.01.2024 р.
7	Отримання відгуку керівника	29.01.2024 р.	29.01.2024 р.
8	Отримання зовнішньої рецензії	01.02.2024р.	31.01.2024 р.

Дата видачі завдання « 04 » 04 2023 р.

Керівник дипломної роботи

 О.В. Редич
(підпис)

Керівник освітньо-професійної
програми

 А.Ю. Горбовий

Завдання до виконання отримав
здобувач вищої освіти

(підпис)
 С.А. Прибисько
(підпис)

АНОТАЦІЯ

Розробка фреймворку для автоматичного тестування веб додатків на базі Selenium WebDriver та TestNG

Мета роботи - автоматизація процесу тестування веб-продукту через розробку власного фреймворку.

Завданнями роботи є аналіз існуючих рішень для автоматизації тестування, проектування архітектури фреймворку, розробку необхідних модулів та компонентів, а також проведення тестування та оцінку ефективності розробленого фреймворку.

Об'єктом дослідження є процес розробки фреймворку для автоматизації тестування інтерфейсу веб-сайту, а предметом дослідження - інструментальні засоби та методологічні підходи до розробки фреймворку для автоматизації тестування веб-сайтів.

Одержані результати можуть бути використані для побудови готового рішення по автоматизації тестування веб-додатків.

Кваліфікаційна робота містить: 76 сторінок, 1 таблицю, 8 рисунків, список використаних джерел містить 41 найменування, 1 додаток.

Ключові слова: тестування, автоматизація, фреймворк, веб додаток, selenium webdriver, maven, testng, паттерн, перевірка, очікування, репорт, архітектура.

ABSTRACT

Development of a framework for automated testing of web applications based on Selenium WebDriver and TestNG.

The aim of the work is to automate the testing process of a web product through the development of a custom framework.

The tasks of the work include analyzing existing solutions for test automation, designing the framework architecture, developing necessary modules and components, as well as conducting testing and evaluating the effectiveness of the developed framework.

The research object is the process of developing a framework for automating the testing of a website's interface, and the subject of the research is instrumental tools and methodological approaches to developing a framework for automating website testing.

The obtained results can be utilized to construct a ready-made solution for automating the testing of web applications.

The qualification work consists of 76 pages, 1 table, 8 figures, a list of used sources with 41 titles, and 1 appendice.

Keywords: testing, automation, framework, web application, selenium webdriver, maven, testng, pattern, verification, expectation, report, architecture

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. Автоматизоване тестування як етап в процесі забезпечення якості продукту.....	8
1.1. Поняття автоматизованого тестування. Переваги і недоліки автоматизованого тестування.....	8
1.2. Огляд існуючих інструментів автоматизованого тестування.....	9
1.3. Критерії вибору компонентів фреймворку.....	12
РОЗДІЛ 2. Складові частини фреймворку.....	14
2.1. Архітектура фреймворку, його компоненти.....	14
2.2. Огляд Selenium та його можливостей.....	16
2.3. Огляд TestNG та його місце у побудові фреймворку.....	20
2.4. Поняття та засоби Continuous Integration.....	21
РОЗДІЛ 3. Практичне застосування фреймворку.....	25
3.1. Патерни побудови логіки тестів.....	25
3.2. Програмні компоненти та процес побудови фреймворку.....	28
3.3. Застосування принципів ООП та TestNG в розробці фреймворку.....	36
3.4. Вибір тестових даних та сценаріїв проходження тестів.....	41
ВИСНОВКИ.....	53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	54
ДОДАТКИ.....	58

ВСТУП

Динамічний розвиток сучасних веб-додатків вимагає ефективних і надійних засобів для забезпечення їхньої якості та функціональності. Існує два основних підходи для забезпечення якості - ручне та автоматизоване тестування програмних продуктів. Якщо ручне тестування передбачає застосування більших зусиль насамперед під час його виконання, то автоматичне тестування концентрує зусилля на етапі підготовки та написання самих тестів.

Автоматизація тестування не лише прискорює процес верифікації та виявлення дефектів, але й відіграє ключову роль у забезпеченні якості продукту. Вона гарантує стабільну роботу та відповідність заявленим вимогам. Автоматизація розв'язує ряд стратегічних завдань, включаючи прискорення та оптимізацію тестування, систематичність повторюваних перевірок, виявлення помилок на ранніх етапах розробки, а також забезпечення безперервної інтеграції. Лідери в галузі розуміють важливість інвестування значних ресурсів у впровадження автоматизації, реалізуючи, що це є важливою умовою для швидкого виходу нових релізів і одночасного збереження високої якості продукту.

У світі автоматизованого тестування, розробка надійних та гнучких фреймворків стає ключовим компонентом для швидкого та ефективного виконання тестів. Слід зазначити, що автоматизоване тестування підходить далеко не всім проектам, наприклад, стартапи можуть забезпечувати якість продукту на перших етапах розробки тільки за допомогою ручного тестування.

Завдання забезпечення якості програмного забезпечення стає тим більш важливим у світлі швидкого циклу розробки та впровадження. Програмний продукт має відповідати вимогам сучасного ринку, де швидкість, точність та широка покриття тестами стають критичними факторами. Розробка ефективного фреймворку для автоматичного тестування веб-додатків є важливим етапом у забезпеченні якості відповідно до найвищих стандартів.

Метою цього проекту є автоматизація процесу тестування веб-продукту через розробку власного фреймворку. Цей фреймворк призначений для прискорення

тестування веб-сайту, забезпечуючи ефективне виявлення помилок та перевірку його правильності.

Основні завдання проекту включають в себе аналіз існуючих рішень для автоматизації тестування, проектування архітектури фреймворку, розробку необхідних модулів та компонентів, а також проведення тестування та оцінку ефективності розробленого фреймворку.

Об'єктом дослідження є процес розробки фреймворку для автоматизації тестування інтерфейсу веб-сайту, а предметом дослідження - інструментальні засоби та методологічні підходи до розробки фреймворку для автоматизації тестування веб-сайтів.

Для вирішення цих завдань будуть використовуватися різноманітні методи тестування, такі як позитивні та функціональні тестові сценарії, інструментальні засоби, такі як Selenium WebDriver та TestNG, та інформаційні технології, зокрема, мова програмування Java.

Очікується, що розроблений фреймворк спростить процес тестування обраного веб-сайту, зменшить витрати часу та ресурсів і забезпечить високу якість та надійність розробленого веб-продукту.

Дослідження базується на практичному використанні розробленого фреймворку для тестування реальних веб-додатків з різних галузей.

Одержані результати можуть служити основою для поліпшення процесу розробки та тестування веб-додатків, а також сприяти розвитку сучасних підходів до автоматизованого тестування.

РОЗДІЛ 1. Автоматизоване тестування як етап в процесі забезпечення якості продукту.

1.1. Поняття автоматизованого тестування. Переваги і недоліки автоматизованого тестування.

Тестування програмного забезпечення (ПЗ) є одним з найважливіших етапів в процесі розробки такого забезпечення для забезпечення його якості. В свою чергу поняття якості - це міра якою ПЗ задовольняє потреби користувачів та відповідає своєму призначенню [1]. Отже, продукт вважається якісним, якщо він відповідає чітким критеріям, встановленим зацікавленими сторонами, користувачами або замовниками. Для того щоб програмний продукт визнавався якісним, йому необхідно відповідати конкретним очікуванням і стандартам. Під терміном "помилки" або "дефекти" в програмному забезпеченні розуміють недоліки в розробленому програмному продукті, які призводять до невідповідності очікуваним результатам виконання програмного продукту та фактично отриманим результатам [2].

Тестування може проводитися як вручну спеціалістами з тестування, так і автоматично за допомогою спеціальних програмних засобів. Процес верифікації програмного продукту, який включає етапи тестування, такі як запуск, ініціалізація, виконання, аналіз і видача результату, може бути автоматизованим за допомогою інструментів автоматизованого тестування і називається автоматизованим тестуванням програмного забезпечення [3].

У автоматизованого тестування є свої переваги та недоліки. До переваг автоматизованого тестування відносять:

- висока швидкість виконання, що перевершує можливості людини;
- відсутність впливу "людського фактора" (не ознайомленість, втома);
- можливість багаторазового виконання тестів і зниження витрат ресурсів;
- виконання тест-кейсів особливої складності, недоступних людині;
- здатність зберігати та аналізувати великі обсяги даних;
- здатність виконувати низько-рівневі дії з додатком, операційною системою та ін.

Недоліки автоматизованого тестування включають:

- необхідність залучення висококваліфікованого персоналу для автоматизації;
- витрати на засоби автоматизації, розробку і підтримку тестів;
- фінансові витрати та ризики, пов'язані з великою кількістю наявних інструментів автоматизації і складністю їх вибору;
- старіння тестів при змінах вимог і переробці інтерфейсів продуктів.

Автоматизація тестування не виключає використання ручного тестування при розробці програмного забезпечення, але допомагає суттєво знизити його обсяг, уникнути рутинних дій і покращити якість розроблюваного програмного забезпечення. Тестування є трудомісткою фазою розробки програмного забезпечення, але ідея автоматизації процесу тестування стає все більш актуальною і важливою, особливо при зростанні складності програмних продуктів і збільшенні вимог до їх якості. Автоматизоване тестування вирішує проблему нестачі часу для якісного тестування, покращуючи ефективність та швидкість процесу.

1.2. Огляд існуючих інструментів автоматизованого тестування.

По-перше, розглянемо популярні мови програмування, що можуть використовуватись для написання автотестів.

Python – це сучасна мова програмування високого рівня, яка знаходить застосування у різних галузях. Її використовують для розробки веб-додатків, аналізу даних, машинного навчання, напрямках DevOps, автоматизації тестування та навіть розробці ігор. Багато початківців обирають Python для вивчення програмування, оскільки вона мінімалістична, проста у вивченні основ та дозволяє зосередитися на розв'язанні завдань, не втрачаючи часу на складний синтаксис. Крім того, Python має велику спільноту та історію, і знаходження відповіді на питання є легким завданням завдяки безлічі книг, форумів та курсів, призначених саме для Python.

Якщо говорити про використання Python у сфері автоматизації тестування, то ця мова стає дуже популярною, особливо серед новачків. Мануальні тестувальники,

які бажають підвищити свою кваліфікацію, починають вивчати Python та спробувати свої сили в написанні власних проектів.

Java, як мова програмування загального призначення, використовується для написання серверної частини веб-додатків, мобільного програмного забезпечення та промислового програмного забезпечення. Програми, написані на Java, можна запустити на будь-якій платформі, якщо на ній встановлено середовище виконання Java (JRE, Java Runtime Environment). Крім того, у Java реалізований механізм управління пам'яттю, відомий як збирач сміття або *garbage collector* [4].

У сфері автоматизації тестування Java є однією з найпопулярніших мов. Розвиток автоматизації саме з неї почався, і наразі для цієї мови існує найбільше кількість бібліотек і фреймворків, які полегшують написання автотестів.

JUnit є базовим фреймворком для модульного тестування, забезпечуючи інструменти для запуску тестів, що входять до категорії тест-раннерів. На основі JUnit був створений TestNG, який є простішим у використанні та призначений для написання більш високорівневих тестів. Він володіє більшим набором анотацій, порівняно з JUnit, що дозволяє запускати окремі методи перед і після автотесту або цілого тестового набору. TestNG також підтримує залежні тести.

JavaScript представляє собою динамічну інтерпретовану мову програмування, яку використовують для написання фронтенд-частини веб-додатків, а також може виконувати роль бекенду за допомогою Node.js. Ця мова програмування дозволяє створювати браузерні ігри і розробляти мобільні програми з використанням React. В сфері автоматизації тестування JavaScript широко використовується для написання автотестів як для веб-додатків, так і для мобільних додатків. Її найбільшою популярністю користуються саме у веб-середовищі, а також вона може бути використана для тестування програм для ПК, якщо вони написані мовою Electrone.

В області тестування JavaScript є відносно новою мовою, але швидко набирає популярність, оскільки веб-розробка також стає все більш популярною. При виборі мови для тестування новачки часто обирають ту, яка вже використовується кимось у їхній команді, особливо при розробці фронтенд-частини, де застосовується JavaScript. Для цієї мови вже створено кілька обгортки для Selenium WebDriver, а

також існує безліч додаткових бібліотек. Однак через свою новизну не всі ці проекти можуть вважатися стабільними.

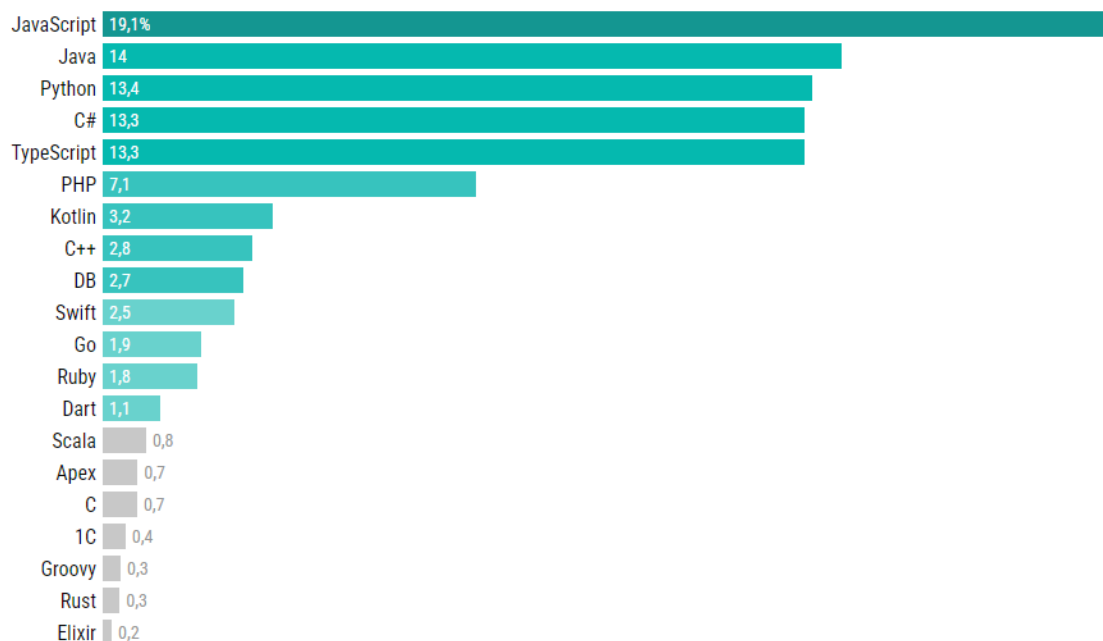


Рис. 1. 1. Рейтинг мов програмування у 2023 році.

Звернемося до рейтингу мов програмування у 2023 році. На Рис 1. 1. представлено рейтинг мов в Україні за даними сайту спільноти програмістів [5], де ми бачимо що лідируючі позиції займають JavaScript, Java, Python. Це пояснюється широкими можливостями цих мов, а також значною підтримкою спільноти та великою кількістю готових рішень.

Перейдемо тепер до розгляду поняття фреймворку та інструментів для його побудови на сьогоднішній момент.

Фреймворк (англ. Framework, каркас, платформа, структура, інфраструктура[6]) — інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проєкту та написання коду.

Розглянемо популярні інструменти для автоматизованого тестування.

Selenium: Відомий фреймворк для автоматизованого тестування веб-додатків. Підтримує різні мови програмування. Початково розроблений в ThoughtWorks як внутрішній інструмент, став відкритим програмним забезпеченням. Здобув

популярність завдяки можливості тестування веб-додатків на різних браузерах та операційних системах. На сьогодні залишається одним із найпоширеніших інструментів для автоматизованого тестування веб-додатків.

Appium: Фреймворк для автоматизованого тестування мобільних додатків на Android та iOS. Створений як відкритий інструмент для тестування мобільних додатків на платформах Android та iOS. Широко використовується в індустрії мобільного тестування, отримавши велику підтримку спільноти. Продовжує залишатися важливим інструментом для тестування мобільних додатків.

JUnit та TestNG: Популярні фреймворки для тестування Java-додатків.

JUnit спочатку був створений як розширення для мови Java для підтримки тестування. Встановив високий стандарт для тестування Java-програм, регулярно оновлюється та розширюється. TestNG створений як конкурент JUnit та отримав популярність завдяки своїм можливостям і підтримці широкого числа анотацій, що значно спрощує роботу з тестовими сценаріями та налаштуванням всього фреймворку.

Playwright: відносно новий фреймворк автоматизованого тестування, розроблений компанією Microsoft, що активно розвивається. Має переваги у швидкодії проходження тестів та підтримці кросбраузерності.

1.3. Критерії вибору компонентів фреймворку.

Вибір фреймворку для автоматизації тестування веб-додатку — це важливий етап, що може визначити ефективність та легкість тестування вашого проекту. Загальний підхід, який можна використовувати при виборі фреймворку може спиратись на наступні критерії:

1. Аналіз вимог. Потрібно ретельно розглянути вимоги проекту та типи тестів, які планується виконувати. Приділити увагу особливостям веб-додатка, таким як технології, використовувані на клієнті та сервері, і архітектурні особливості.

2. Підтримка технологій. Необхідно бути впевненим, що обраний фреймворк підтримує технології, використовувані в вашому веб-додатку (наприклад, JavaScript-бібліотеки, фреймворки, RESTful API).
3. Легкість вивчення та використання. Цей критерій дуже важливий на початковому етапі введення автоматизованого тестування на проєкті. Якщо у команді немає досвіду з автоматизацією, то легкість вивчення фреймворку стає особливо значущою. Потрібно розглянути доступність документації, навчальних матеріалів та розвиток спільноти, яка підтримує фреймворк.
4. Інтеграція з іншими інструментами. Фреймворк повинен легко інтегруватися з іншими інструментами, такими як засоби збору звітів, системи керування конфігурацією, інструменти CI/CD тощо.
5. Спільнота та підтримка. Потрібно оцінити активність спільноти фреймворку, частоту випуску нових версій та наявність великої кількості користувачів.
6. Масштабованість. Ще однією вимогою може бути те, як фреймворк розрахований на масштабування, особливо якщо проєкт передбачає розростання в масштабах.
7. Тестування безпеки та стійкості. Опціональний фактор вибору фреймворку якщо тестування безпеки та стійкості важливо для проєкту, потрібно впевнитись, що фреймворк підтримує відповідні інструменти.

Як бачимо з наведених критеріїв та наявних сучасних компонентів для побудови фреймворку для нашого проєкту найбільше підходить зв'язка Selenium WebDriver та TestNG аннотацій. Цей вибір має переваги перед іншими насамперед у його потужних можливостях управління браузером та здатністю до масштабування, тобто підключення додаткових модулів у разі розвитку нашого проєкту. TestNG модуль дозволяє більш розширено управляти тестовими даними завдяки широкому спектру аннотацій, а також більш гнучко налаштовувати тестові сценарії.

РОЗДІЛ 2. Складові частини фреймворку.

2.1. Архітектура фреймворку, його компоненти.

Архітектура фреймворку автоматизації тестування визначає структуру, організацію та взаємодію компонентів для забезпечення ефективного написання, виконання та обслуговування автоматизованих тестів. Існують різні архітектурні моделі, і конкретна архітектура може залежати від вимог конкретного проекту та вибраного фреймворку.

До загальних компонентів можна віднести:

1. Модуль тестового скрипту. Це частина фреймворку, де розробляються тестові скрипти. Його завдання — взаємодія з елементами тестованого додатка та виконання необхідних перевірок.
2. Мова програмування. Фреймворки для автоматизованого тестування можуть використовувати різні мови програмування (наприклад, Java, Python, C#). Вибір мови залежить від уподобань команди розробників та технічних вимог проекту.
3. Драйвер тестування (Test Driver). Цей компонент взаємодіє з тестовими скриптами та ініціює виконання тестових кейсів. Драйвер тестування може контролювати виконання тестів та збирати результати.
4. Бібліотека тестових скриптів (Test Script Library). Бібліотека містить готові функції та методи, які можна використовувати в тестових скриптах для виконання стандартних дій.
5. Менеджер конфігурацій (Configuration Manager). Деякі фреймворки мають компонент для управління конфігураціями, який дозволяє налаштовувати параметри тестового середовища та інші конфігураційні параметри.
6. Засоби реєстрації та звітності (Logging and Reporting Tools). Для відстеження прогресу виконання тестів та аналізу результатів важливо мати засоби реєстрації та звітності.

7. Менеджер тестових даних. Для забезпечення незалежності тестів та повторюваності результатів, важливо мати можливість ефективно керувати тестовими даними.

Важливо враховувати, що конкретні компоненти та їх взаємодія можуть відрізнятися в залежності від типу додатка, вимог проекту та вибраного фреймворку.

Якщо розглянути взаємодію процесів між компонентами всередині самого тестового фреймворку, то цю взаємодію можна представити у вигляді схеми Рис 2.1

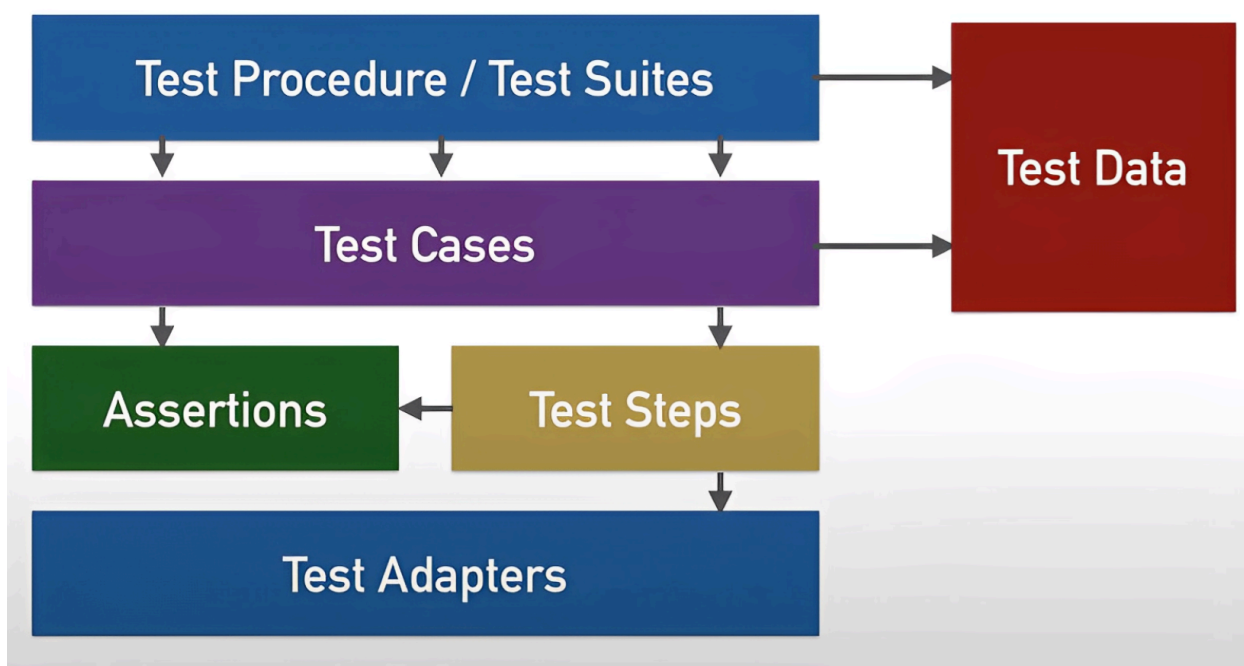


Рис. 2. 1. Схема зв'язків компонентів тестового фреймворку.

Це загальна схема компонентів із яких мінімально повинен складатися тестовий фреймворк [7].

Почнемо розгляд з шару Test Procedure/Test Suites. Це шар який управляє вибіркою і поведінкою тестів під час конкретного запуску і має здатність генерувати тест кейси динамічно при користуванні Test Data модулем. Test Data це модуль, який визначає вхідні параметри для тестових сценаріїв. Зрозуміло, що тестові дані можуть надходити у різних форматах та з різних джерел. Тому у тестовому фреймворку повинні бути відповідні бібліотеки та адаптери для роботи з цими вхідними даними.

Класичний приклад такого модуля - це генератор параметрів для тестів REST API на основі додатка Swagger. Оскільки Swagger містить опис типів полів API, їх

обмеження та формат, то можна легко проаналізувати його і на основі цього згенерувати необхідні параметри для тестів.

Не менш важливим модулем є наступний великий шар, що складається з таких елементів: Test Cases, Test Steps та Assertions. Test Cases це набір тестових сценаріїв за якими буде проходити перевірка додатка. Він складається з певних, окремих кроків (Test Steps), а також з перевірок виконання (Assertions). Test Steps також можуть використовувати Assertions, оскільки деякі загальні перевірки успішного виконання повинні бути взяті з кроків, а не "розкидані" по всіх тестах. В іншому випадку, при зміні кроку, доведеться змінювати перевірку по всьому проекту.

Самий нижній шар представлено у вигляді технічного модуля, що загально можна назвати як тестові адаптери (Test Adapters). Цей шар забезпечує взаємодію з тестованою системою. Під час аналізу системи приймається рішення про те, через які інтерфейси та обхідні шляхи тести будуть відправляти та отримувати дані, змінювати стан системи та інше. Потім всі виклики обертаються адаптерами, щоб абстрагувати себе від системи, що тестується та протоколів взаємодії з нею. До таких інтерфейсів, зазвичай, відносять технічні протоколи, такі як HTTP, SSH і т.д. Саме сюди відноситься Selenium WebDriver та інші модули, такі як Mobile, Desktop, Image Recognition і т.д.

2.2. Огляд Selenium та його можливостей.

Selenium WebDriver представляє собою гнучкий інструмент для автоматизованого тестування веб-проектів, який базується на наборі бібліотек для різних мов програмування, таких як Java, .Net (C#), Python, Ruby, PHP, Perl, JavaScript. Цей інструмент підтримує операційні системи Windows, macOS та Linux, а також найпоширеніші браузерери, такі як Google Chrome, Firefox, Safari, Edge, Internet Explorer, а навіть деякі браузерери без графічного інтерфейсу. Використовується Selenium WebDriver переважно для регресійного та функціонального тестування.

Щодо інструментів Selenium, окрім WebDriver, існують кілька інших. Selenium Standalone Server - це інструмент для віддаленого запуску Selenium, відомий також як Selenium Grid. Selenium IDE - це плагін для браузера Firefox, який працює за принципом "record-and-play", практично не використовується в реальній практиці тестування. Selenium RC - застаріла версія Selenium Driver, іноді вказують на неї як на Selenium 1.0, в той час як Selenium WebDriver - це Selenium 2.0 і вище. Основним продуктом в екосистемі Selenium є WebDriver.

Selenium WebDriver - це драйвер для управління браузером. Замість того, щоб виконувати дії в браузері вручну, можна написати програму, яка буде виконувати ці дії за допомогою Selenium WebDriver. Таким чином, можна сказати, що Selenium дозволяє симулювати дії користувачів у браузері. Він може використовуватися не лише для автоматизації тестування, але й для автоматизації різних рутинних дій. Також WebDriver може запускатись локально або на віддаленому сервері за допомогою Selenium Grid.

Для розуміння принципів роботи Selenium WebDriver важливо розглянути основні компоненти для взаємодії. Архітектура Selenium WebDriver включає 4 компоненти (Рис. 2. 2):

1. Драйвер конкретного браузера.
2. Клієнтська бібліотека Selenium.
3. Браузер.
4. Протокол JSON Wire (JavaScript Object Notation).

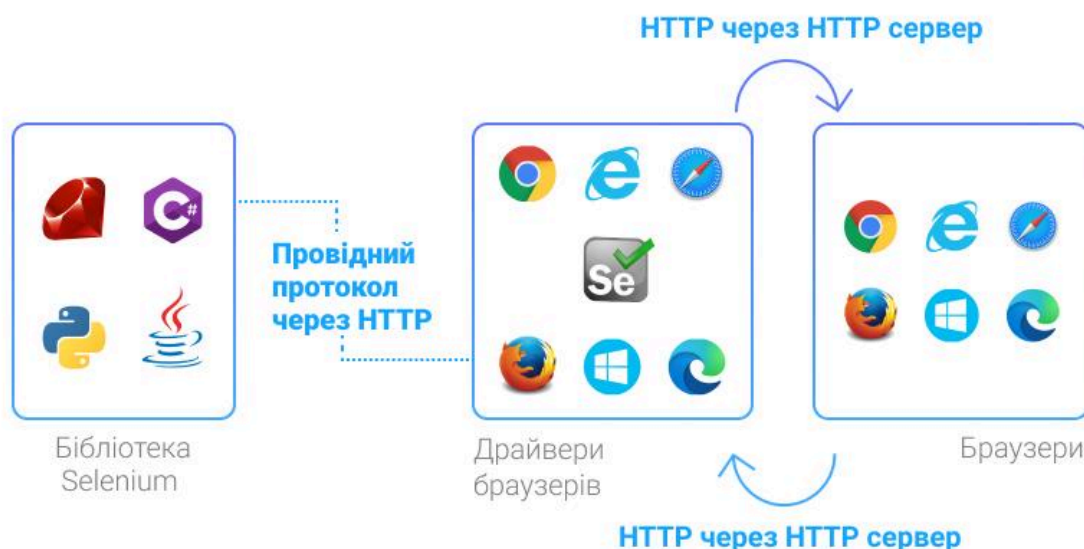


Рис. 2.2. Основні компоненти Selenium WebDriver

З одного боку у нас є веб-додаток, який потрібно протестувати. З іншого боку - тестовий фреймворк, який є інструментом або набором інструментів для тестування. Оскільки ми працюємо з веб-додатком, застосунок відкривається в веб-браузері, і наш тестовий фреймворк взаємодіє з серверною та бекендною частинами браузера за допомогою Selenium Driver, здійснюючи різні дії, такі як кліки, заповнення полів і перевірка результатів цих дій.

Основні сутності та методи Selenium WebDriver

1. WebDriver – це найважливіша сутність, яка відповідає за управління діями браузера.
2. WebElement – друга важлива сутність, яка представляє собою абстракцію над конкретним веб-елементом (посиланням, кнопкою, полем для вводу та ін.).
3. Локатор – це тип веб-елементу, який потрібно знайти та над яким виконуватиме дії веб-драйвер.
4. By – це абстракція над локатором веб-елемента, клас необхідний для ідентифікації веб-елементів який має такий синтаксис By.локатор.

Нижче наведено список найбільш поширених типів локаторів та їх синтаксис:

- `By.id` – пошук елемента по атрибуту `id`;
- `By.name` – пошук елемента по атрибуту `name`;
- `By.className` – пошук елемента по назві класу, використовується коли на веб-сторінці відображається набір однотипних елементів;
- `By.tagName` – пошук елемента по атрибуту назви тега;
- `By.LinkText` – пошук елемента по конкретному тексту, якщо він точно відомий;
- `By.PartialLinkText` – пошук елемента по частковому збігу тексту;
- `By.cssSelector` – пошук елемента по CSS-селектору, який визначає до якого елемента або групи елементів (якщо, це клас) буде застосовано стиль;
- `By.XPath` – пошук елемента по атрибуту XML path.

Тепер перейдемо до основних дій, що може робити `Driver` по відношенню до елементів. Загальний список дій дуже великий і всі розглянути не має можливості, але найбільш використовувані наведено нижче.

- `click()` - клік на елемент.

Можна натискати майже на будь-який об'єкт, такий як кнопка, радіо-кнопка, прапорець, текстове поле, розкривний список та інші елементи.

- `sendKeys()` - вставка тексту.

Це виконується за допомогою методу, який приймає рядок `sendKeys()`. Ми викликаємо метод `sendKeys()` і передаємо туди певний рядок. Такий `sendKeys()` можна використовувати для текстового поля, поля вводу, щоб написати в ньому який-небудь текст. Ще одна функція `sendKeys()` емулює натискання клавіш клавіатури, як окремих, так і їх комбінацій. Наприклад, можна викликати `sendKeys(Keys.return)`, що емулює натискання клавіші `Enter`.

- `getText()` - отримання тексту елемента.

Звісно, перед тим як прочитати текст елемента, необхідно переконатися, що він дійсно присутній у даному елементі.

- `getAttribute()` - читати значення атрибутів у тегах.

Метод `getAttribute()` приймає рядок як параметр. В якості параметра потрібно передати назву атрибута, і метод поверне його значення.

2.3. Огляд TestNG та його місце у побудові фреймворку.

TestNG – це тестовий фреймворк, створений Cédric Beust, допомагає задовольнити значну кількість потреб у тестуванні. А саме, TestNG дає розробнику можливість створювати більш гнучкі та потужні тести. TestNG широко використовується разом з Selenium.

Сама назва TestNG (Next Generation) означає - “Наступне покоління”. Мається на увазі, що TestNG це покращена версія попереднього тестового фреймворка JUnit

Особливості TestNG, що надають йому перевагу серед інших:

- Потужні і різноманітні анотації для підтримки тест-кейсів.
- TestNG використовує більше можливостей Java і ООП
- Розділяє тестовий код часу компіляції і інформацію про конфігурацію даних часу виконання.
- Паралельне виконання тестів, використання залежностей між тестами. Скажімо тестування навантаження і часткової відмови.
- Гнучкість виконання тестів з різними наборами даних, через файл TestNG.xml або через концепцію постачальників даних (data-provider).
- Угрупування і пріоритизація тест-кейсів.
- Генерація HTML-звітів, встановлення додаткового ПЗ з різних плагінів.
- Особливо важливо гнучкий плагін API.
- Генерація логів виконання тестів.
- Легка інтеграція з Eclipse, Maven, Jenkins і ін.

Перейдемо від особливостей TestNG до його основних можливостей, що стануть нам у пригоді під час розробки власного фреймворку.

- набір тверджень (assertions), котрі знаходяться в класі Assert, для написання різноманітних перевірок.

Наприклад:

assertEquals/assertNotEquals - перевірка на відповідність (тотожність).

assertTrue/assertFalse - перевірка на логічні значення True/False.

assertNull/assertNotNull - перевірка на порожність/непорожність.

- набір анотацій для гнучкості налаштування середовища і тестів на різних етапах виконання.

Наприклад:

@BeforeSuite/AfterSuite - анотований метод буде виконано перед тим/після того, як всі тести в цьому наборі будуть запущені.

@BeforeTest/AfterTest - анотований метод буде виконано перед/після будь-яким тестовим методом, що належить класам всередині тегу <test>.

@BeforeClass/AfterClass - анотований метод буде виконано перед/після тим, як буде викликаний перший тестовий метод у поточному класі.

@BeforeGroups/AfterGroups - список груп, перед/після яких буде виконаний цей метод конфігурації. Цей метод гарантовано буде виконаний невдовзі перед/після того, як буде викликано перший тестовий метод, що належить будь-якій з цих груп.

@BeforeMethod/AfterMethod - анотований метод буде виконано перед кожним/після тестовим методом.

2.4. Поняття та засоби Continuous Integration.

Неперервна інтеграція (Continuous Integration) - це методика розробки програмного забезпечення, яка включає в себе регулярне автоматизоване збирання проекту для найшвидшого виявлення та вирішення проблем інтеграції. У традиційних проектах, де розробники працюють над різними частинами системи незалежно, стадія інтеграції є завершальною і може непередбачувано затримувати завершення робіт. Впровадження неперервної інтеграції дозволяє зменшити трудомісткість процесу інтеграції та зробити його більш передбачуваним шляхом виявлення та усунення помилок та конфліктів найраніше можливо.

Так описується поняття Continuous Integration в процесі розробки програмного продукту, але цей же самий підхід можна і необхідно впроваджувати під час

розробки фреймворка автоматизованого тестування. Завдяки Continuous Integration можна налаштувати процес збірки, запуску тестів, створення звітності для тестів, зробити автоматичний запуск ваших тестів і так далі. Іншими словами, можна автоматизувати робочий процес, автоматизувати процес, пов'язаний з запуском автоматизованих тестів. Чому для автоматизації тестування варто налаштовувати таке середовище Continuous Integration і запускати в ній тести? Тому що середовище розробки служить для створення та підтримки тестів, а для запуску та аналізу тестів воно не дуже зручне. Наприклад, в середовищі розробки не зберігається історія запусків, артефакти перезаписуються, і результат буде доступний лише вам локально на вашому комп'ютері. Щоб запустити тести, вам потрібна відповідна конфігурація, належним чином налаштоване середовище розробки, а іноді ці налаштування доводиться змінювати під час роботи. Таким чином, це досить не зручний процес, якщо потрібно запускати тести саме з метою тестування додатків, а не для розробки тестів, то рекомендовано не використовувати середовище розробки, а звернутись до одного з наявних середовищ Continuous Integration.

Тут важливо сказати, що тести можна запускати також з консолі, наприклад, командою `mvn clean test`. І відповідно для запуску тестів з консолі можливо написати скрипт, який би збирав історію, результати тестів, звіти і навіть запускав тести в певний час. Але, на сьогодні, існують інструменти Continuous Integration, такі як Jenkins або TeamCity. Існує безліч інструментів Continuous Integration, але, на мій погляд, ці два одні з найбільш популярних. Jenkins - це open-source інструмент, він безкоштовний і дуже популярний, для нього існує безліч плагінів. TeamCity - це також хороший якісний інструмент від творців IntelliJ IDEA. Безкоштовна версія TeamCity має обмеження за кількістю агентів, різні агенти дозволяють вам запускати процеси паралельно. Jenkins ж не має жодних обмежень, а в разі TeamCity, щоб мати необмежені можливості, вам знадобиться мати також ліцензію.

Для нашого проекту з побудови фреймворку розглянемо безкоштовну версію Jenkins.

Jenkins – система з відкритим вихідним кодом, тобто продукт доступний для перегляду, вивчення та зміни. Він створений на базі Java. Надає сотні плагінів для

підтримки створення, розгортання та автоматизації будь-якого проекту. Створений ще у 2008 році.

Jenkins дозволяє автоматизувати частину процесу розробки програмного забезпечення без участі людини. Ця система призначена для забезпечення процесу безперервної інтеграції програмного забезпечення.

Переваги Jenkins:

1. Безперервна інтеграція та безперервна доставка.

Як розширений сервер автоматизації, Jenkins можна використовувати як простий сервер CI або перетворити на центр безперервної доставки для будь-якого проекту.

2. Легкий монтаж

Jenkins — це автономна програма на основі Java, готова до запуску «з коробки» з пакетами для Windows, Linux, macOS та інших операційних систем, подібних Unix.

3. Легке налаштування

Jenkins можна легко налаштувати за допомогою веб-інтерфейсу, який включає перевірку помилок на льоту та вбудовану довідку.

4. Плагіни

Завдяки сотням плагінів у Центрі оновлення, Дженкінс інтегрується практично з кожним інструментом у ланцюзі інструментів безперервної інтеграції та безперервної доставки.

5. Розширення

Jenkins можна розширити за допомогою архітектури плагінів, надаючи майже нескінченні можливості для роботи.

6. Поширення

Jenkins може легко розподіляти роботу між кількома машинами, допомагаючи швидше створювати, тестувати та розгортати на кількох платформах

Зараз Jenkins використовується практично в будь-якій сучасній компанії, де є необхідність в автоматичному деплоїменті (розгортанні) додатків, а також у зручному управлінні різноманітних завдань.

РОЗДІЛ 3. Практичне застосування фреймворку.

3.1. Патерни побудови логіки тестів.

Перед тим як перейти до написання коду потрібно ознайомитись з паттернами, тобто підходами у побудові логіки тестів.

Патерн - це концептуальне рішення для розв'язання загальних проблем, що виникають під час проектування програм. В контексті автоматизації тестування існує понад 80 патернів. Ці патерни розроблені з урахуванням таких ключових характеристик, як надійність, зрозумілість, гнучкість та стабільність, які важливі для автоматизованих тестів. Коротко розглянемо найпоширеніші з них, а також їх переваги.

До першої групи віднесемо структурні патерни:

- “Page Object” патерн використовується для автоматизації інтерфейсу веб-додатків. Кожна сторінка має свій власний клас, де зберігаються елементи та методи, що взаємодіють із цією сторінкою. Переваги: Спрощує підтримку, уникнення дублікатів і зменшує заплутаність коду. Забезпечує легше змінення та підтримку.
- “Page Element” патерн пропонує виносити логіку роботи із загальними елементами для кількох сторінок веб-додатка. Переваги: Дозволяє централізовано управляти елементами, що знаходяться на різних сторінках. Зменшує дублювання коду та підвищує його зрозумілість.

Ці структурні патерни спрощують розробку автотестів, роблять код більш зрозумілим і легко підтримуваним, що особливо важливо в автоматизованому тестуванні.

Наприклад, у випадку верхнього меню, яке відображається на кожній сторінці та з яким потрібно взаємодіяти з ним у різних автотестах, ефективним є винесення методів роботи з меню в окремий клас. Це дозволяє істотно скоротити дубльований код та прискорити написання нових тестів, оскільки логіка взаємодії з елементами вже описана.

- Патерн "Loadable Component".

Проблема непередбачуваного завантаження елементів на сторінці призводить до використання "waiters" - методів очікування. "Loadable Component" вирішує цю проблему, успадковуючи опис сторінок та конкретизуючи очікування для кожної сторінки. Це дозволяє приховати логіку очікувань в одному місці та забезпечує налаштування очікувань для кожної сторінки.

- Патерн "Strategy"

Застосовується, коли є кілька реалізацій однієї дії, і вибір конкретної залежить від контексту. Наприклад, реєстрацію користувача можна виконати через UI або HTTP запит до API. "Strategy" створює узагальнену стратегію для досягнення результату та може містити декілька реалізацій.

Перейдемо тепер до групи патернів даних. Вони націлені на максимальне відокремлення даних від логіки тестів, щоб уникнути зайвого коду та полегшити розуміння структури автотесту.

- Паттерн "Value Object".

Цей патерн використовується для об'єднання логічно пов'язаних об'єктів, вводячи додатковий "Value Object", який містить цю інформацію. Наприклад, у сутності користувача може бути набір параметрів, таких як ім'я, прізвище, вік, телефон, які можна структурувати та агрегувати в "Value Object", названий, наприклад, "User". Цей об'єкт є незмінним і представляє собою простий засіб передачі даних від одного місця до іншого. Використання такого підходу дозволяє легко організовувати параметри відповідно до їхньої належності до сутності, забезпечуючи чітку читабельність коду.

- Патерн "Builder".

Патерн "Builder" спрощує роботу з об'єктами, які можуть мати велику кількість різних змінних характеристик. Замість того, щоб кожного разу створювати новий конструктор для нової конфігурації, застосовується "Builder" об'єкта, який містить методи зі змінними характеристиками. Ці методи викликаються лише для необхідних

значень, що дозволяє передавати тільки ті параметри, які потрібно змінити, зберігаючи залишок їхньої стандартної конфігурації.

- Патерн "Data Registry".

"Data Registry" використовується тоді, коли потрібно отримувати унікальні дані для кожного тесту, при цьому гарантуючи, що унікальний набір даних використовується лише в межах конкретного тесту і не використовується в інших тестах. Цей патерн винесення логіки генерації даних в "Data Registry," де може бути визначений алгоритм рандомізації. При такому підході, при необхідності нових даних, їх можна запитувати у "Data Registry" і немає потреби вручну перевіряти їхню унікальність, оскільки це гарантує логіка рандомізатора.

- Патерн "Data Provider":

"Data Provider" реалізує підхід Data-driven Testing, де один тест проходить з використанням різних наборів даних. Дані завантажуються з зовнішнього файлу чи сервісу, що дозволяє тестувати різні сценарії використання за рахунок різноманітних вхідних даних.

Також слід окремо виділити технічні патерни побудови фреймворка.

- Патерн "Decorator".

Патерн "Decorator" використовується при взаємодії з технічними драйверами, такими як веб-драйвер, коли потрібно додати до нього додатковий функціонал, такий як логування, кешування або інші додаткові можливості. Головна мета полягає в тому, щоб внести ці зміни, не змінюючи логіку тестів, оскільки автотести не повинні залежати від нових доповнень. Для досягнення цього створюється додатковий обгортковий шар навколо основного драйвера, де реєструється слухач (listener), і тим самим автотести можуть без змін використовувати інтерфейс драйвера.

- Патерн "Proxy".

Патерн "Proxy" дозволяє втручатись у процес, що відбувається між клієнтом та сервером, для впровадження будь-якої логіки, не торкаючись жодної сторони. Один із способів використання цього патерну - використання HTTP проху для запуску тестів. Це дозволяє встановлювати додаткову логіку або проводити перевірки на

етапі "спілкування" між автотестом і програмою, що тестується. Наприклад, можна паралельно з тестуванням збирати HTTP-трафік або кешувати необхідні дані.

Ці патерни технічної реалізації застосовуються для відокремлення технічних деталей від тестової логіки, забезпечуючи кращий контроль над аспектами фреймворку.

Для задач нашої реалізації найбільш підходять патерни комбіновані з декількох видів. А саме: Page Object, Page Element та Data Provider.

3.2. Програмні компоненти та процес побудови фреймворку.

Для роботи над побудовою фреймворку автоматизації нам необхідні наступні компоненти:

1. Середовище розробки. Для зручності роботи та налаштувань підходить середовище IntelliJ Idea від компанії JetBrains. можна скористатись безкоштовною версією "Community", що дозволяє повністю задовольнити наші потреби.
2. Java JRE/JDK. Оскільки розроблюваний фреймворк буде написаний на мові програмування Java, то нам буде необхідний комплект розробника програмного забезпечення від компанії Oracle.
3. Apache Maven - це засіб автоматизації роботи з програмними проєктами. Використовується для керування (management) та складання (build) програм.

Встановлення, налаштування та способи використання компонентів не будемо розглядати в межах цієї роботи.

Структура проєкту в середовищі програмування та підключення необхідних бібліотек

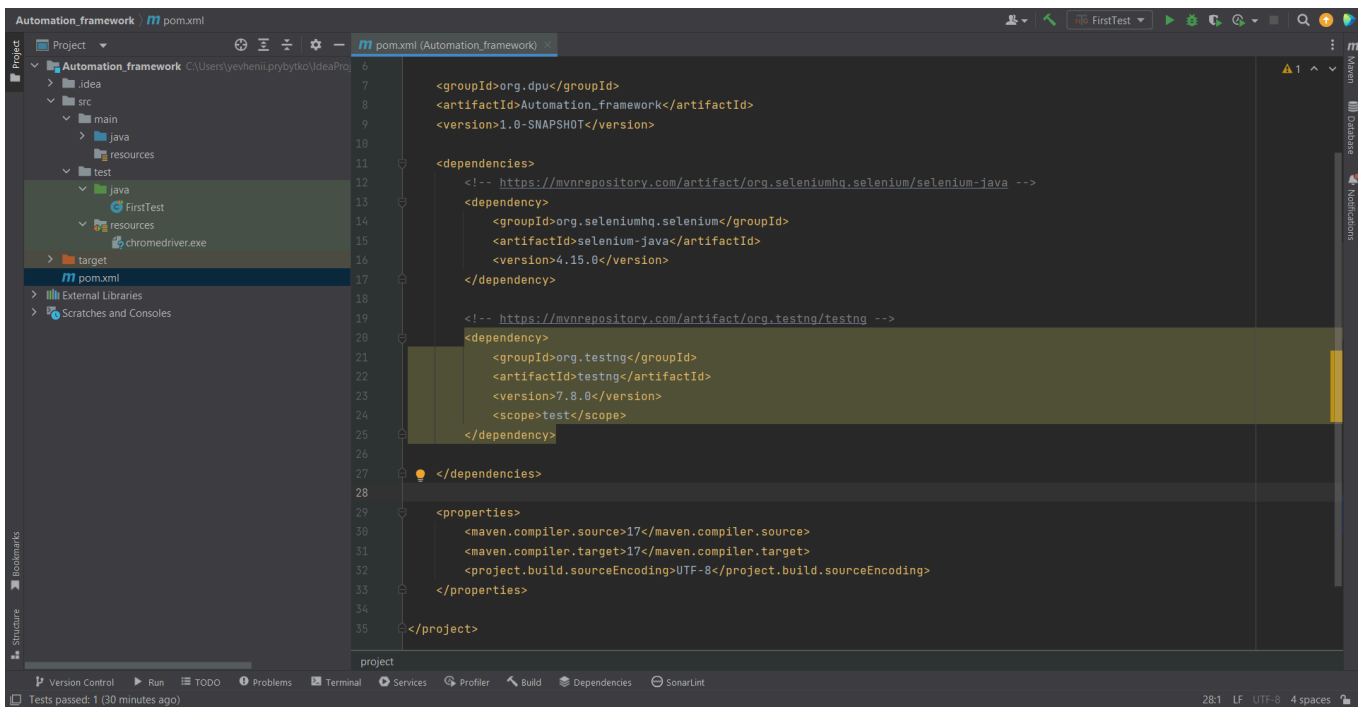


Рис. 3.1. Структура проєкту в середовищі програмування

Проект створено за допомогою засобу Maven, що значно полегшує з управлінням та налаштуванням взаємодії підключених бібліотек, а також задає початкову структуру проєкту. Перейдемо до огляду основних розділів та елементів проекту.

Структура проєкту для автоматизації тестування має особливі відмінності. Проект складається з двох модулів: main та test, як показано на прикладі (Рис. 3.1).

Модуль main зберігає інформацію коли йде розробка програми а не набір тестів, тому його ми не будемо використовувати в нашому проєкті.

Модуль test містить у собі пакети з UI наборами тестів. Зазвичай, набори тестів поділяють певною логікою. Наприклад, для UI набори тестів можна розділити за назвами сторінок програми. У цьому модулі зібрані лише автотести та ресурси (файли з даними, тестові файли), необхідні для їх виконання. Також в модулі test зберігається розділ з ресурсами resources, де знаходяться самі файли драйверів для Selenium, наприклад, файл chromedriver.exe. Він призначений для управління саме браузером Chrome, також там можуть бути інші драйвери для інших браузерів. Це дуже зручно при запуску тестів для різних типів браузерів.

Розділ target зберігає структуровану інформацію про попередні запуски тестів та їх статус. Perezapisuetsya при кожному запуску автотестів.

Ще одним важливим елементом проєкту є файл `pom.xml`, що належить до структури Maven проєкту. В ньому зберігається інформація про підключені бібліотеки до проєкту та їх версії. Структура цього файлу показана на Рис. 3.1, де можна побачити підключені основні бібліотеки проєкту: `selenium-java` та `testng`.

Як простий приклад сценарію та демонстрацію початкових можливостей автоматизованого фреймворку розглянемо файл `FirstTest`. Це ще не повноцінний тестовий сценарій, тому що не містить перевірок результатів. Такі перевірки будуть розглянуті і додані при вивченні можливостей TestNG компоненти. В подальшому, при додаванні функціональних можливостей, фреймворк буде ускладнюватись та змінювати свою структуру як того вимагають сучасні методики побудови таких фреймворків. Будемо вважати поточний стан структури як відправну точку в побудові сучасного фреймворку автоматизації тестування.

```
public class FirstTest {
    @Test
    public void openGoogleComInChromeTest() {
        File file = new
            File("src/test/resources/chromedriver.exe");
        System. setProperty("webdriver.chrome.driver",
            file.getAbsolutePath());
        WebDriver driver = new ChromeDriver();
        driver.navigate().to( "https://www.google.com/");

        WebElement searchField = driver.findElement(By.name("q"));
        searchField.click();
        searchField.sendKeys("selenium java");
        searchField.sendKeys(Keys.RETURN);
        WebElement resultRow =
            driver.findElement(By.xpath("//div[@class='yuRUbf']/h3"));
        System.out.println(resultRow.getText());
        driver.quit();
    }
}
```

```
}  
}
```

Цей код є прикладом, що демонструє лише частину можливостей Selenium драйвера. Маємо наступну послідовність дій роботи драйвера з цього прикладу:

- 1) Відкриття браузера Chrome.
- 2) Відкриття в браузері початкової сторінки Google.
- 3) Ввід в пошукове поле запиту “selenium java”.
- 4) Натискання кнопки “Enter”.
- 5) На відкритій сторінці результатів пошуку знаходження першого елемента зі знайдених результатів.
- 6) Вивід на екран заголовка цього елемента.
- 7) Закриття браузера і завершення тесту.

Слід зазначити, що на даному етапі було застосовано анотацію TestNG фреймворку - `@Test`. Ця анотація позначає, що цей метод є тестом та дає змогу запускати такі методи з будь якого місця проєкті. В іншому випадку запуск коду можливий тільки з головного методу `main`. Інші, не менш важливі, анотації буде розглянуто дещо пізніше.

Перейдемо до розгляду верифікації результатів тестів. З точки зору використання в тестових фреймворках, асерти (від англ. “Assert” - твердження) - це метод, який дозволяє перевіряти результати виконання тестів. Найчастіше це порівняння очікуваних та реальних результатів тестів. Асерт діє за таким принципом. Він перевіряє, що переданий в нього вираз вірний, а якщо ні, то він генерує виняток (exception). Після того, як виняток виник, TestNG завершує тест зі статусом “failed” (англ. “не вдалося”), тобто тест не пройшов, впав з якоїсь причини. При автоматизації тестового сценарію, обов'язково повинен бути хоча б один асерт. В іншому випадку це буде просто послідовність виконуваних дій, тобто якийсь скрипт, а не повноцінний тест.

Якщо асерт не проходить, то при цьому виникає виняток і тест падає. Однак якщо асерт проходить успішно, то він нічого не робить і тест йде далі. Якщо тест містить кілька асертів і всі вони проходять, то тест завершується у статусі "passed" (англ. "пройшов").

```

WebDriver driver = new ChromeDriver();

driver.navigate().to("https://www.google.com");

WebElement feelingLuckyButton =
driver.findElement(By.name("btnI"));

assertTrue(feelingLuckyButton.isEnabled(), "I'm Feeling Lucky
button is disabled!");

assertEquals(feelingLuckyButton.getAttribute("value"), "I'm Feeling
Lucky", "Wrong text has been displayed!");

```

Actual result
Error message
Expected result

Рис. 3.2. Приклади асертів.

Отже, у нашому проєкті ми вже використовуємо бібліотеку TestNG, тому для початку розглянемо саме TestNG асерти. На Рис 3.2. бачимо приклади таких асертів. Два найчастіше використовувані методи асертів це `assertTrue()` і `assertEquals()`. Почнемо з `assertTrue()`, який приймає два параметри. Перше - це умова, друге - це яке-небудь повідомлення. Обов'язково має бути передана умова - це який небудь вираз, який повертає або `true`, або `false`. Якщо ця умова виконалася, тобто повертає `true`, асерт проходить і не виводить при цьому жодних повідомлень. Якщо ж умова не виконалася і повернула `false`, асерт генерує виняток, в якому повідомляється про причини помилки. Щоб причина помилки була максимально зрозумілою при аналізі результатів тестів, настійно рекомендовано писати повідомлення про помилку в асерті.

Асерт `assertEquals()`. Він приймає аргументи Actual Result, Expected Result та Message. Цей асерт порівнює два значення і також має опціональний аргумент формі повідомлення про помилку. За допомогою нього можливо порівнювати дві

строки, два цілочисельні значення та інші типи даних. Необхідно пам'ятати, що можна порівнювати дані лише одного типу.

На рисунку показано аргумент `message` - це саме те саме повідомлення про помилку, і воно опціональне. Однак гарна практика - це додавати повідомлення про помилку в асерт. При цьому необхідно описати його зрозумілою мовою. Добре зроблені асерти в майбутньому можуть заощадити багато часу.

Крім перевірок, що нам пропонує TestNG модуль, існує ще досить багато інших, більш гнучких та функціональних модулів. Розглянемо один з таких, а саме AssertJ бібліотеку, яка дасть нам робити ланцюжки перевірок в наших тестах. Встановлення додаткової бібліотеки робимо через управління проектом Maven через додавання у файл залежностей `pom.xml`. AssertJ бібліотека перевірок дає нам змогу використовувати один уніфікований запис для виклику а множини подальших методів гнучких перевірок.

Наведемо приклад застосування такого запису. Для цього нам потрібно зробити статичний імпорт у наш клас тестів, а потім звернутись до бібліотеки напряму через імпорт:

```
import static org.assertj.core.api.Assertions.assertThat;

public class FirstTest {

    @Test
    public void openGoogleComInChromeTest() {
        ...
        WebElement resultRow =
            driver.findElement(By.xpath("//div[@class='yuRUbf']/h3"));
        assertThat(resultRow.isDisplayed).as("Element has not been
        displayed").isTrue();
        assertThat(resultRow.getText()).as("Wrong test has been
        displayed").isEqualTo("The Selenium Browser Automation Project");
    }
}
```

З прикладу бачимо, що в загальному синтаксис перевірки складається з декількох повторюваних пунктів. А саме, в першій частині аргументів перевірки ми вказуємо на об'єкт перевірки та дію над ним (`resultRow.getText()`), потім йде текст помилки (`as("Element has not been displayed")`), якщо перевірка не буде успішною. В кінці вказуємо саме метод перевірки. Можуть бути різні методи: на присутність, на рівність, на нерівність і т. д. Повний список можливих перевірок досить великий і може бути використаний для будь яких перевірок [8]. Якщо перевірка не проходить, то в результатах проходження тесту буде показано два результати: очікуваний та наявний. Порівнявши такі результати можна легко встановити чи помилка присутня в самому додатку, чи в самому написаному тесті.

Розглянемо тепер таку не менш важливу функціональну частину фреймворка, як очікування (англ. "Waits"). Очікування застосовуються для отримання часу для виконання певної дії браузером. Оскільки автоматизовані тести можуть виконуватись дуже швидко, то деякі елементи веб додатку можуть не встигати з'явитись в структурі сторінки або деякі дії не відбудуться до того часу коли вже буде задіяно перевірку тесту. Тому очікування це необхідний інструмент в роботі автотестів. Існують різні типи очікувань. Це неявне очікування ("implicit wait"), явне очікування ("explicit wait") та кастомні очікування ("fluent wait"). Ми більш докладно розглянемо саме неявне і явне очікування.

Неявне очікування — це один з видів очікувань. Зазвичай неявне очікування налаштовується в проекті один раз, хоча за необхідності його можна змінювати. При включенні неявного очікування воно працюватиме однаково для всіх елементів. Це означає, що доки елемент на сторінці веб додатку не стане видимим або не з'явиться, то драйвер не буде продовжувати свої дії. Також неявне очікування використовує тайм-аут. Якщо елемент не буде знайдений і тайм-аут вичерпано, настане виключення "no such element" (такого елемента немає). Приклад налаштування такого очікування показано нижче:

```
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
```

Налаштовується неявне очікування через метод драйвера `manage()`. Потім викликаємо метод `implicitlyWait()` і передаємо туди параметри. Спочатку час очікування, наприклад, 30, а потім ми вказуємо, в чому саме ми хочемо чекати. Наприклад, вказуємо, що це секунди.

Явне очікування — це індивідуальне, користувацько налаштоване очікування, яке підходить для конкретної ситуації. Явне очікування налаштовується гнучко. По-перше, це гнучке налаштування тайм-ауту. Наприклад, ми створюємо об'єкт `WebDriverWait` з тайм-аутом у 30 секунд.

```
WebDriverWait wait = new WebDriverWait( driver, 30);  
wait.until(ExpectedConditions.elementToBeClickable(searchButton));
```

Ми використовуємо клас `WebDriverWait` та ініціалізуємо нашу змінну за допомогою `new WebDriverWait` та передаємо в конструктор класу сам екземпляр класу `Driver` і значення 30 — це час очікування. За потреби можна створити об'єкт `WebDriverWait` з тайм-аутом, наприклад, 20 секунд, і будь-яким іншим. При цьому можливо створити стільки об'єктів, скільки потрібно, і використовувати їх в різних ситуаціях незалежно один від одного. Наприклад, для одного елемента потрібно чекати не більше 20 секунд, а для іншого — чекати до 30 секунд. Це перше, де проявляється гнучкість використання таких очікувань.

Наступний важливий момент — це події, які ми очікуємо при використанні `WebDriverWait`. Можна очікувати різні події, і це не обов'язково лише поява елемента, як у випадку з неявними очікуваннями. Явне очікування за допомогою методів класу `ExpectedConditions` може очікувати наявність елемента DOM, видимість елемента на сторінці, і, навпаки, можна чекати, поки елемент не припинить бути видимим на сторінці. Можна очікувати такі події, як завантаження веб-сторінки, поява сповіщення або навіть чекати, поки JavaScript поверне яке-небудь значення. Приклад в тексті вище — це очікування, доки кнопка не стане клікабельною, тобто, якщо кнопка не є клікабельною, тестовий драйвер буде чекати, поки вона не змінить свій статус, і на неї можна буде натискати. Якщо умова

очікування так і не виконалася, тоді у цьому випадку тест провалиться через помилку `TimeoutException`, тобто явне очікування викличе виняток через тайм-аут.

Що важливо розуміти при використанні вейтів, так це те що коли ви використовуєте очікування, потрібно визначити які саме очікування будуть використовуватися у фреймворку. Явні очікування не будуть працювати, якщо вже увімкнено неявні. Для того щоб вимкнути загальні очікування потрібно змінити конфігурацію встановивши час в нульове значення. Іншими словами, зробити час очікування рівним 0.

3.3. Застосування принципів ООП та TestNG в розробці фреймворку

Перейдемо до розгляду, як можна використовувати анотації `@Before` і `@After` компоненти TestNG розглянуті у розділі 2.3 на практиці. Для цього виділимо частину кроків в окремі методи і назовемо ці методи `Setup` і `Teardown`.

`@BeforeClass`

```
public void setUp() {  
    File file = new File("src/test/resources/chromedriver.exe");  
    System.setProperty("webdriver.chrome.driver", file.getAbsolutePath());  
    driver = new ChromeDriver();  
    driver.navigate().to("https://www.google.com");  
}
```

`@AfterClass`

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

Бачимо два таких методи, позначених анотаціями `Before` і `After` класу. Це метод `Setup` і метод `Teardown`. Давайте подивимося, що робить метод `Setup` і `Teardown`. У методі `Setup` ми створюємо екземпляр хром-драйвера і переходимо на

сторінку google.com. Я виділив ці дії в окремий клас, який буде виконуватися лише один раз. Після цього в тестовому фреймворку я буду виконувати кілька тестів уже на відкритому хром-драйвері та на сторінці google.com. Те саме можна сказати і про метод Teardown. У цьому методі ми закриваємо веб-драйвер, і робимо це також один раз, після того, як всі тести класу виконані. Приблизно так буде виглядати наш тест після того, як ми виділили з нього всі передумови та післяумови тесту, тобто підготовчі дії та дії по завершенню тесту.

@Test

```
public void openGoogleComInChromeTest() throws InterruptedException {
    WebElement searchField = driver.findElement(By.name("q"));
    searchField.click();
    searchField.sendKeys("selenium java");
    searchField.sendKeys(Keys.RETURN);
    WebElement resultRow = driver.findElement(By.xpath("//div[@class='yuRUbf']/h3"));
    assertThat(resultRow.isDisplayed()).as("Element has not been displayed").isTrue();
    assertThat(resultRow.getText()).as("Wrong text has been displayed").
        isEqualTo("The Selenium Browser Automation Project");
    System.out.println(resultRow.getText());
}
```

Звернемо увагу, як значно скоротилася кількість коду в тесті. Тепер тест набагато компактніший, і при написанні наступних тестів код не буде повторюватися. Усі дії, які ми виділили в окремі методи, будуть перевикористовуватися, згідно принципу "не повторюй себе". Далі ми продовжимо вдосконалювати тестовий фреймворк. Наступний крок — це не використання якихось спеціальних бібліотек, це просте використання успадкування, що є одним з принципів ООП (Об'єктно-Орієнтованого Програмування). Дуже важливо при побудові тестового фреймворку використовувати принципи ООП, тому давайте подивимося, як це буде виглядати на практиці. Створимо абстрактний клас BaseTest і перемістимо туди наші методи before і after. Після цього успадкуємо SearchTest від

BaseTest. Тобто створимо окремий клас, в якому будуть саме тестові методи, а BaseTest буде містити всілякі вспоміжні методи, які ми будемо використовувати в наших тестах. Отже, SearchTest успадковує setup і teardown з класу BaseTest. При цьому логіка самого тесту не змінилася. Нижче можемо побачити як буде виглядати клас BaseTest:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.DataProvider;
import java.io.File;

public abstract class BaseTest {
    protected WebDriver driver;

    @BeforeClass
    public void setUp() {
        File file = new File("src/test/resources/chromedriver.exe");
        System.setProperty("webdriver.chrome.driver", file.getAbsolutePath());
        driver = new ChromeDriver();
        driver.navigate().to("https://www.google.com");
    }

    @AfterClass
    public void tearDown() {
        driver.quit();
    }
}
```

По-перше, клас `BaseTest` є абстрактним, оскільки ми не збираємося створювати об'єкт даного класу, ми просто хочемо успадкувати від нього певну кількість методів. Саме клас містить поле `WebDriver`, і на даному етапі поле `WebDriver` є `protected`, оскільки нам знадобиться використовувати екземпляр `WebDriver` в наших тестових методах, і також він нам знадобиться безпосередньо в класі `BaseTest`. Ну і тут ви бачите ті самі методи `setup` і `teardown`, про які ми говорили раніше. Тут ніяких змін, просто вони зараз знаходяться в класі `BaseTest`. А от, саме, як буде виглядати клас `SearchTest`, який успадковує `BaseTest`. Тут ми маємо тестовий метод з анотацією `@Test`, і сам тест містить кроки і перевірки. Тобто він містить наступні дії: шукаємо потрібний веб-елемент, після того, як веб-елемент знайдений, ми робимо перевірку, у даному випадку “`assert equals`”, і перевіряємо, що потрібний текст був відображений на сторінці. Тобто виконуємо перевірку і порівнюємо реальні результати з очікуваними.

Саме за такими принципами і будується фреймворк в подальшому. Використовуючи `Page Object` патерн ми створюємо класи-сторінки веб-додатку як окремі сутності, додаємо туди всі необхідні елементи і дії, що знадобляться нам для тестування на цій, певній сторінці. А загальний: підготовчий та післявиконавчий функціонал самого фреймворку виноситься з загальний абстрактний клас від якого будемо успадковувати всі інші сутності.

Ще однією потужною функцією `TestNG` модуля є можливість проходження тих самих тестів з різними тестовими даними за один запуск автотестів. Це досягається за допомогою концепції постачальників даних, а саме анотації `@DataProvider`. Побудуємо такий модуль для нашого фреймворку.

Давайте розглянемо простий приклад використання цієї анотації, тобто спробуємо передати певні тестові дані через `@DataProvider`. Спочатку нам знадобиться сам `DataProvider`, і ми можемо його створити у класі `BaseTest` та використовувати його у різних тестах.

```
@DataProvider(name = "dataProvider")
public Object[][] dataProviderMethod() {
```

```
return new Object[][]{{"Selenium java"},"Selenium JavaScript"}};
}
```

Зверніть увагу, що `DataProvider` буде повертати тип `Object` і це буде двовимірний масив. Відповідно, створимо `return`, і тут ми можемо передати різні тестові дані, на яких ми хочемо тестувати. Наприклад, можемо вписати у поле `SearchField` слова "Selenium java". Проте, оскільки ми передаємо двовимірний масив, то передамо не лише один параметр в кожен тест, але й кілька. Один тест буде запусканий із параметром "Selenium java", інший тест вже буде запусканий наступного разу із параметром "Selenium JavaScript". Для того, щоб прив'язати `DataProvider` безпосередньо до наших тестів, де він буде використовуватися потрібно на рівні анотації `@Test`, додати параметр `DataProvider`. Тут також потрібно присвоїти йому якусь назву, наприклад, нехай це буде "dataProvider".

```
@Test(dataProvider = "dataProvider")
public void openGoogleComInChromeTestWithDataProvider(String text) {
    WebElement searchField = driver.findElement(By.name("q"));
    searchField.click();
    searchField.sendKeys(text);
    searchField.sendKeys(Keys.RETURN);
    WebElement resultRow = driver.findElement(By.xpath("//div[@class='yuRUbf']/h3"));
    assertThat(resultRow.isDisplayed()).as("Element has not been displayed").isTrue();
    assertThat(resultRow.getText()).as("Wrong text has been displayed").
        isEqualTo("The Selenium Browser Automation Project");
}
```

Отже, `TestNG` запустить декілька тестів. Перший тест буде із першим масивом, і другий тест вже буде із використанням наступного масиву, де зберігається "Selenium JavaScript". Це дуже зручно, і таких тестів ми можемо зробити скільки

завгодно, якщо є потреба. Ми розглянули найпростіший із можливих сценаріїв, але `DataProvider` можна використовувати, наприклад, щоб тестові дані брались із окремих файлів, таблиць.

3.4. Вибір тестових даних та сценаріїв проходження тестів.

Побудований тестовий фреймворк дозволяє нам проводити тестування користувацького інтерфейсу в автоматичному режимі, імітуючі реальні дії користувачів на сторінках веб додатка. Постає питання які саме тестові сценарії відібрати з усіх можливих для того, щоб їх покрити автотестами. Відповідь на це питання допоможе дати стратегія автоматизації тестування створена на основі класифікації Майка Кона подана у його роботі «Succeeding with Agile» [9]. Згідно цієї класифікації автоматизація тестування може бути поділена на рівні:

- модульне тестування;
- інтеграційне тестування;
- тестування користувацького інтерфейсу.

Ці рівні утворюють традиційну структуру тестування, відому як "класична піраміда тестування", яку можна побачити на рисунку 3.3. “Класична піраміда тестування” за Майком Коном. У цій піраміді кожен наступний рівень тестів, починаючи з модульних, розширює тестові сценарії нижчих рівнів і додає нові сценарії інтеграції частин системи. Кількість тестів на конкретному рівні залежить від його положення у піраміді. Однак особливістю піраміди є те, що разом із підвищенням рівня тестування зростають і витрати часу та зусиль на розробку та підтримку тестових сценаріїв. Тести вищого рівня характеризуються великою крихкістю, оскільки навіть незначні зміни в системі можуть призвести до їх невірності у великій кількості. Однак ці сценарії тестування узагальнюють і завершують системність тестування і, найчастіше, демонструють сценарії використання розробленої системи.



Рисунок 3.3. “Класична піраміда тестування” за Майком Коном

Модульне тестування є основою для структури "піраміди тестування" і гарантує, що найдрібніші компоненти програмного забезпечення, такі як функції, методи та класи, відповідають встановленим вимогам. Згідно з Майком Коном [9], ці тести мають становити ключову частину автоматизованого тестування, і кількість сценаріїв тестування на цьому рівні повинна бути максимальною. Зазвичай програмісти розробляють модульні тести без участі тестувальників, а сценарії цих тестів представляють собою програмний код, який тестує конкретний функціонал в ізоляції від інших компонентів системи. Модульні тести деталізовані, і за допомогою регресійного тестування ефективно виявляють дефекти вихідного коду. Через ізольованість тестування можна точно визначити місце, де був виявлений знайдений дефект.

Сервісне або інтеграційне тестування (англ. integration testing) – це процес тестування, в якому об’єктами тестування є інтерфейси системи та взаємодія між інтегрованими компонентами. Залежно від архітектури системи такими

взаємозалежними компонентами можуть бути як методи одного класу, так і ряд функцій, що об'єднані спільною бізнес-логікою. Оскільки розробка різноманітних модулів системи або підсистем виконується ізольовано та різними командами, виникає проблема подальшої інтеграції створених частин у єдину систему. Сутність інтеграційного тестування полягає в тестуванні взаємодії модулів як частини єдиної системи та в перевірці того, що окремо розроблені модулі працюють разом належним чином, а їхні інтерфейси відповідають дизайну архітектури системи.

Тести користувацького інтерфейсу, розташовані на вершині "класичної піраміди тестування", призначені для імітації дій користувача в інтерфейсі програми, таких як натискання кнопок миші, введення тексту і т.д. Очікується, що інтерфейс програми буде передбачувано реагувати на цю взаємодію. Загальна кількість сценаріїв тестування користувацького інтерфейсу повинна бути мінімальною, оскільки будь-які зміни у користувацькому інтерфейсі можуть призвести до недійсності цих тестів. Створення таких сценаріїв тестування є трудомісткою задачею, як зазначено Майком Коном [9].

Оскільки розроблюваний фреймворк автоматизованого тестування покликаний для тестування функціоналу веб додатка саме через користувацький інтерфейс, то написання тестових сценаріїв вимагає продуманості та логічної послідовності тестів. Ця частина роботи може бути виконана силами мануальних тестувальників, оскільки не передбачає наявності спеціальних знань з програмування та роботи фреймворка. Зазвичай в таких тестових сценаріях притримуються позитивного підходу до тестування, тобто такі сценарії тестують функціонал системи в роботі з правильними даними та етапами в роботі. З його допомогою ми можемо визначити, що система робить те, для чого і була створена. Негативний підхід передбачає, що система тестується в розрізі її поведінки при роботі з неправильними, неповними або неточними даними. Перш за все негативне тестування направлено на перевірку стійкості системи до різних впливів, валідації невірних даних, обробки виняткових ситуацій.

Для демонстрації працездатності побудованого фреймворку виберемо веб додаток для якого напишемо кілька тестових сценаріїв. Так як фреймворк

побудовано з урахуванням гнучкості в застосуванні, то він підходить для тестування більшості сучасних веб додатків. Тому для демонстрації нам підходить будь який сучасний додаток. Наприклад, сайт <https://taxer.ua/>, що спеціалізується на допомозі і підготовці комплекту документів для реєстрації фізичної особи – підприємця, відправлення електронної звітності до податкової служби та Пенсійного фонду, оплати підприємницьких податків. Цей веб додаток побудований з оглядом на сучасні тенденції розробки сайтів та має широку аудиторію користувачів серед підприємців України. Слід зазначити, що сайт має зручний користувацький інтерфейс та зрозумілу логіку роботи з ним.

В цій роботі не будуть розроблені та описані всі можливі тестові сценарії для тестування всього веб додатку Taxer, оскільки це займе багато часу і не є метою даної роботи. На тестовому сценарію буде показано роботу побудованого фреймворку автоматизації.

Крок взаємодії з веб додатком	Очікуваний результат
1. Відкрити веб додаток за адресою https://taxer.ua/ у веб браузері.	Веб додаток https://taxer.ua/ відкрито у вікні браузера
2. Відкрити вікно входу в користувацький кабінет, натиснувши кнопку “Вхід”	Відкрилось вікно “Вхід” поверх головної сторінки сайту.
3. Ввід коректних даних для входу та натиснення кнопки “Увійти в Таксер”	Кабінет користувача відкрито на закладці “Огляд”.
4. Перехід на вкладку “Події” - “Календар”	Вкладка “Календар” відкрита, присутні податкові події на поточний календарний рік.
5. Перехід на вкладку “Події” - “Поточні”	Інформація щодо сплати ЄП з'явилась в списку доступних подій.
6. Натиснути кнопку “Сформувати рахунок”	Вікно з назвою “Сплата ЄП (ФОП)” з'явилося поверх сторінки. Поля “Прізвище, ім'я та по батькові.”, “Ідентифікаційний номер.” та

	“Сума:” доступні до заповнення.
7. Заповнити доступні поля “Прізвище, ім’я та по батькові:”, “Ідентифікаційний номер:” та “Сума:” та натиснути кнопку “Зберегти”	Вікно з реквізитами та введеними даними з’явилося поверх попереднього вікна.
8. Відкрити збережений платіж та перевірити, що всі внесені дані коректно збереглися	Вікно збереженого платежу відкрито, дані заповнені коректно
9. Закрити вікно браузера з веб додатком	Вікно закрито, тест пройдено

Таблиця 1. Тестовий сценарій для перевірки працездатності фреймворку.

Події вказані в стовпцю “Очікуваний результат” також слугують критеріями для написання відповідних перевірок при проходженні тесту та виявлення можливих невідповідностей (помилки) роботі самого веб додатку. Наведений тестовий сценарій містить не дуже багато кроків, але він по максимуму розкриє функціонал роботи автоматизованого фреймворку. Сценарій вимагає від фреймворку наступних дій: відкриття сторінки, натискання відповідних кнопок та вкладок інтерфейсу, заповнення полів даними; а також різноманітних перевірок: наявності різноманітних елементів інтерфейсу та коректності введених даних. Звичайно, що один сценарій не може повністю перевірити всі функції фреймворку, оскільки останніх дуже велика кількість і не всі вони використовуються під час автоматизації тестування.

Як було вже зазначено в Розділі 3.3, однією з основних ознак підходу в патерні Page Object є розбиття сторінок веб додатку на окремі сутності та створення для цих сутностей різних класів. Переформатуємо структуру нашого фреймворку у відповідності до тестового сценарію. Він набуде такого вигляду:

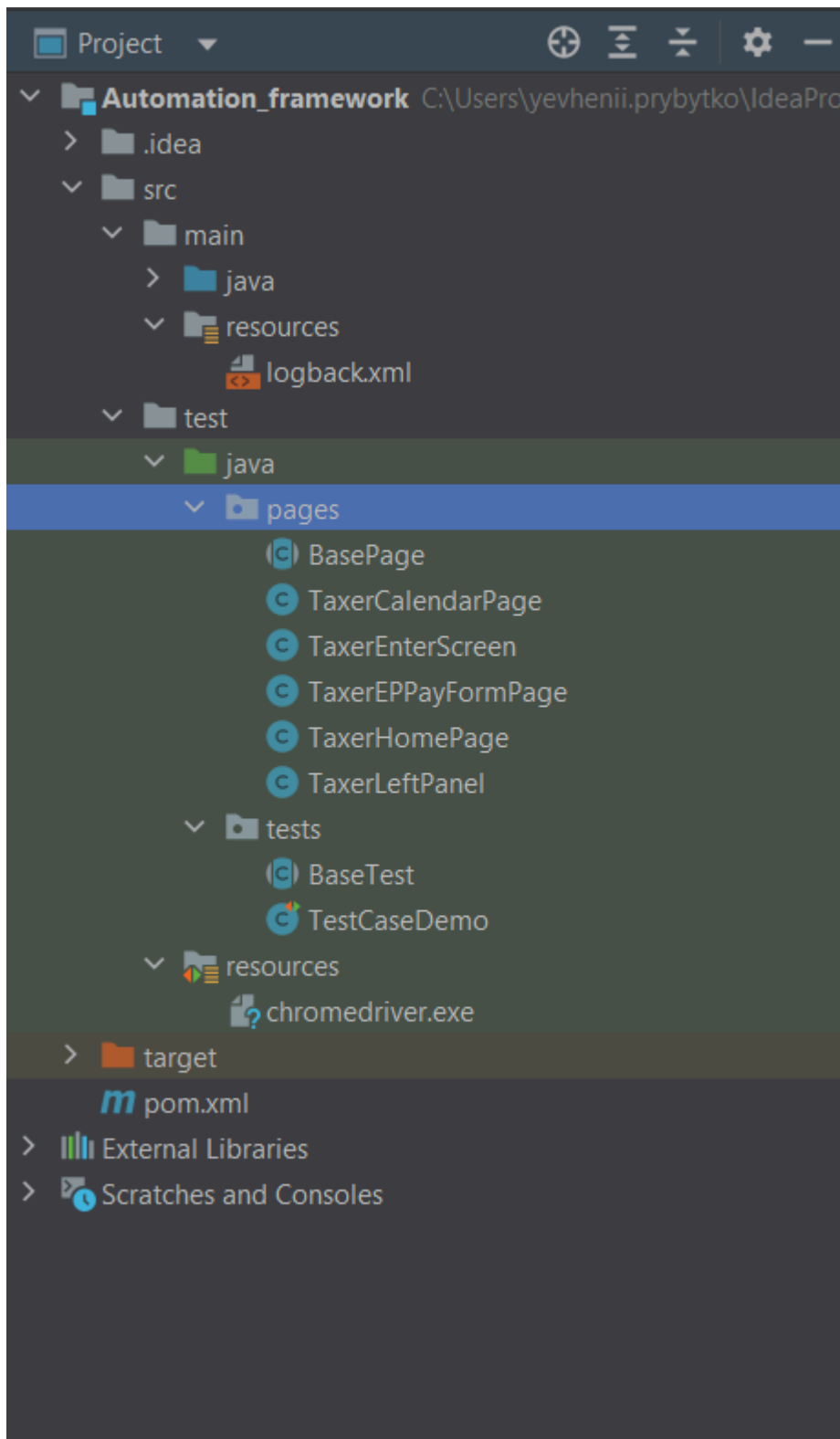


Рисунок 3.4. Структура файлів у відповідності до PageObject патерну.

Бачимо з рисунку що кожній сторінці до якої ми звертаємось під час тесту створено відповідний клас у структурі проекту. Так само в директорії tests містяться класи з самими кроками тестового сценарію. Це дозволяє зручно і зрозуміло

зберігати інформацію про тести у разі їх збільшення та чітко розмежовувати логіку самих тестів від логіки виконання цих тестів.

Наведемо сам код тесту, написаний під даний сценарій.

```
package tests;
import org.testng.annotations.Test;
import pages.*;

public class TestCaseDemo extends BaseTest {
    private String indCode = "1234567890";
    private String amount = "4200";
    private String amountAfterSave = "4 200,00 грн";
    private String PIB = "Микола Василенко";

    @Test(dataProvider = "emailAndPass")
    public void userFlowDemo(String email, String pass) {
        TaxerHomePage taxerHomePage = new TaxerHomePage();
        taxerHomePage.verifyWebAppLoaded(); // Крок 1
        taxerHomePage.clickEnterButton(); // Крок 2

        TaxerEnterScreen taxerEnterScreen = new TaxerEnterScreen();
        taxerEnterScreen.inputEmailAndPass(email, pass); // Крок 3
        taxerEnterScreen.pressLoginButton(); // Крок 3

        TaxerLeftPanel taxerLeftPanel = new TaxerLeftPanel();
        taxerLeftPanel.calendarButtonClick(); // Крок 4

        TaxerCalendarPage taxerCalendarPage = new TaxerCalendarPage();
        taxerCalendarPage.checkYear("2024"); // Крок 4
        taxerCalendarPage.checkEventPanel(); // Крок 4
```

```
taxerLeftPanel.currentEventsButtonClick(); // Крок 5
```

```
taxerLeftPanel.checkEPEvent(); // Крок 5
```

```
TaxerEPPayFormPage taxerEPPayFormPage = new TaxerEPPayFormPage();
```

```
taxerEPPayFormPage.formPaymentButtonClick(); // Крок 6
```

```
taxerEPPayFormPage.userValidDataPayment(PIB, indCode, amount); // Крок 7
```

```
taxerEPPayFormPage.clickSaveButton(); // Крок 7
```

```
taxerEPPayFormPage.openSavedButtonClick(); // Крок 8
```

```
taxerEPPayFormPage.checkEnteredUserDataIsCorrectInForm(PIB, indCode,  
amountAfterSave); // Крок 8
```

```
}
```

```
}
```

Клас тесту наслідувано від базового класу і має в собі всі функції цього класу. А саме: створення та запуск WebDriver для управління браузером, відкриття потрібної початкової сторінки у вікні браузера, тестові дані для передачі через DataProvider, а також закриття браузера після виконання тестів.

При написанні цього коду було використано декілька важливих функцій як самого WebDriver, так і оболонки TestNG. По-перше, це розмежування методів виконання по відповідним сторінкам веб додатку. Тобто перед виконанням дій на певній сторінці потрібно створити екземпляр цієї сторінки, а потім звертатись до її методів. По-друге, це використання DataProvider анотації, що робить, з однієї сторони, можливим збільшення тестових даних для внесення і перевірки у веб додаток, а з іншої сторони це відбувається без перезапуску всього тестового сценарію.

Також розглянемо написаний код для цього тестового сценарію, що міститься саме в класі певної сторінки. Для прикладу, візьмемо клас TaxerEnterScreen, що відповідає за управління імітації входу користувача до персонального кабінету.

```

public class TaxerEnterScreen extends BasePage {

    private static final Logger logger =
    LoggerFactory.getLogger(TaxerEnterScreen.class);

    JavascriptExecutor js = (JavascriptExecutor) driver;

    public TaxerEnterScreen() {
        super();
    }

    @FindBy(xpath =
    "/html/body/ui-popup/div/div/div/app-auth-popup/div/app-auth-popup-login/div/ui-site-po
    pup-content/div/div/div[1]/div[2]/form/ui-site-form-row[1]/div/div[1]/input")
    private WebElement emailField;

    @FindBy(xpath =
    "/html/body/ui-popup/div/div/div/app-auth-popup/div/app-auth-popup-login/div/ui-site-po
    pup-content/div/div/div[1]/div[2]/form/ui-site-form-row[2]/div/div[1]/ui-password/div/inp
    ut")
    private WebElement passwordField;

    @FindBy(xpath =
    "/html/body/ui-popup/div/div/div/app-auth-popup/div/app-auth-popup-login/div/ui-site-po
    pup-content/div/div/div[1]/div[2]/form/ui-site-form-row[4]/div/div[1]/div/ui-button/button
    ")
    private WebElement enterTaxerButton;

    @FindBy(xpath =
    "/html/body/app-root/div/aside/div/layout-drawer-auth/div/ul/li[8]/div/span")
    private WebElement exitButton;

    @FindBy(xpath =
    "/html/body/app-root/div/aside/div/layout-drawer-auth/div/ul/li[2]/a")

```

```
private WebElement eventButton;
```

```
public void inputEmailAndPass(String email, String pass){
    wait.until(ExpectedConditions.visibilityOf(emailField));
    wait.until(ExpectedConditions.visibilityOf(passwordField));
    emailField.sendKeys(email);
    passwordField.click();
    passwordField.sendKeys(pass);
    logger.info("Email and Pass entered correctly");
}
```

```
public void pressLoginButton() {
    wait.until(ExpectedConditions.visibilityOf(enterTaxerButton));
    enterTaxerButton.click();
    logger.info("Login button was clicked");
    wait.until(ExpectedConditions.visibilityOf(eventButton));
    js.executeScript("arguments[0].scrollIntoView(true);", exitButton);
    assertThat(exitButton.isDisplayed()).as("Element has not been
displayed").isTrue();
    logger.info("User is logged in");
}
```

В цьому прикладі коду слід звернути увагу на декілька речей. Клас `TaxerEnterScreen` наслідується від базового класу `BasePage`, в якому відбувається ініціалізація, тобто попереднє створення елементів веб сторінки, як тільки вона стає доступною браузерові. Це дозволяє вчасно звертатися до DOM елементів веб сторінки після їхнього створення, а не до їх появи, що призвело до помилок самого виконання тестів. Також в класи такого типу додано логування подій через вільний до розповсюдження модуль логування `slf4j`.

```
private static final Logger logger = LoggerFactory.getLogger(TaxerEnterScreen.class);
```

Він дозволяє виводити інформацію про проходження тестів саме під час їх виконання. Що дуже зручно для локалізації можливих проблем при написанні автотестів та їх виконанню.

Завдяки TestNG анотації `@FindBy` дуже зручно і лаконічно вказувати локатори до веб елементів, а потім ці елементи використовувати у відповідних методах через змінний параметр. Для підтримання автотестів це ще зручно і тим, що у випадку зміни локатора елемента у веб додатку знадобиться відповідно змінити його тільки в одному місці, при оголошенні, і не перевіряти, що всі методи, де він використовувався стануть непрацюючими.

Саме у методах, як видно з елементу коду, використовуються явні очікування. Вони дозволяють більш гнучко налаштовувати час очікування тієї чи іншої події та умови настання цих подій. Для проекту вказано максимальний час очікування в 10 секунд при не настанні будь якої події. Цього достатньо, оскільки веб додаток, що тестується має досить велику швидкодію і великих затримок у відображенні елементів на сторінці чи переходах між сторінками не має.

Також широко по проекту використовуються перевірки того що відбулось на сторінці. Без цих перевірок успішне тестування неможливе, оскільки саме на них спираються тестові сценарії.

Запустимо автоматизований тест та подивимось на його результати. Запуск та проходження тестового сценарію відбувається на локальному комп'ютері, тож можна слідкувати за його перебігом. Також доступно налаштувати Selenium WebDriver на запуск на віддаленому сервері використовуючи технологію Selenium Grid.

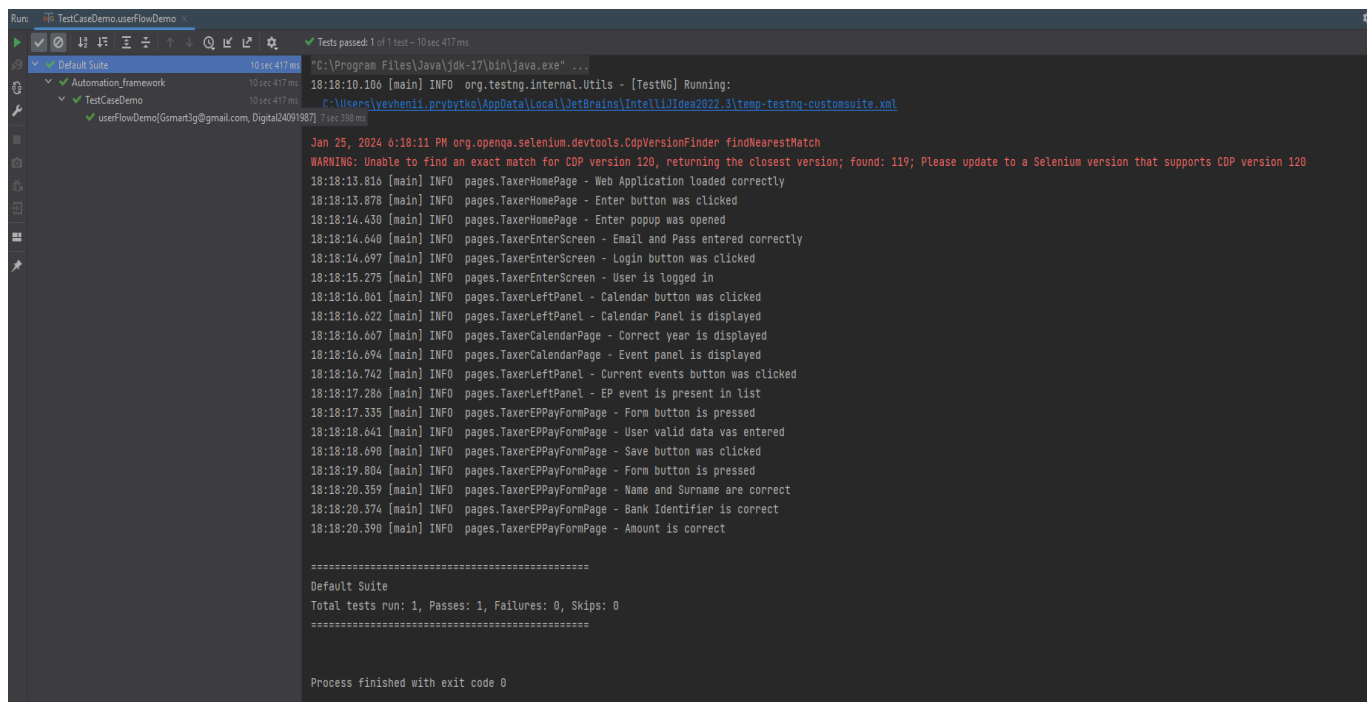


Рисунок 3.5. Результати запуску автоматизованого тестового сценарію.

Як бачимо з Рисунок 3.5. тест пройшов успішно та завдяки підключеному логуванню він вивів усі виконані кроки під час роботи. Слід зазначити, що виконання цього автоматизованого сценарію зайняло всього 10 секунд, що є дуже гарним показником. Завдяки такій швидкодії виконання автоматизоване тестування і має перевагу перед ручним. Така швидкість виконання недоступна людині. Якщо функціонал всього веб додатку покрити такими автотестами, то перевірка працездатності поточної версії сайту зайняла б десятки хвилин, на що у людини пішло б декілька днів.

Весь написаний код фреймворку автоматизованого тестування доступний у додатку А.

ВИСНОВКИ

Робота над проектом фреймворка автоматизованого тестування веб-додатків включала в себе докладний аналіз сучасних тенденцій та методів у цій галузі розробки. Під час дослідження було вивчено різноманітні підходи до побудови фреймворків автоматизованого тестування, включаючи використання різних патернів проектування, інструментів та технологій. Даний аналіз дозволив вибрати найбільш оптимальні та ефективні компоненти для подальшої реалізації фреймворку.

Однією з ключових складових успішної реалізації була модель схеми зв'язків компонентів всередині тестового фреймворку. Ця модель дозволила чітко визначити взаємозв'язки між різними елементами фреймворку та забезпечити його гнучкість та масштабованість. Використання логіки патерну Page Object було ключовим аспектом у підтримці чистоти та підтримки коду фреймворку.

Під час практичної реалізації була розроблена структура фреймворку, включаючи програмний код запуску, проходження та наповнення тестовими даними сценаріїв. Для валідації функціональності фреймворку був обраний сучасний веб-додаток, на якому проводились практичні тести. Ці тести дозволили перевірити працездатність та ефективність розробленого фреймворку в реальних умовах веб-додатку.

У результаті фреймворк автоматизованого тестування продемонстрував значні переваги та ефективність в процесі автоматизованого тестування. Його гнучкість, зручність в застосуванні та можливість масштабування роблять його придатним інструментом для автоматизації тестування веб-додатків. Результати практичних тестів підтвердили, що використання фреймворку може значно скоротити час та зусилля, необхідні для проведення ручних тестів, і сприяти підвищенню якості програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISTQB glossary. ISTQB Glossary. URL: https://glossary.istqb.org/en_US/ (date of access: 19.12.2023).
2. Copeland L. A practitioner's guide to software test design. Boston, Mass : Artech House, 2004. 294 p.
3. Автоматизоване тестування. URL: <https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya/> (date of access: 19.12.2023).
4. Sierra K. Head first Java. Sebastopol, CA : O'Reilly, 2003. 619 p.
5. Рейтинг мов програмування 2023. URL: <https://dou.ua/lenta/articles/language-rating-2023> (date of access: 19.12.2023).
6. Учасники проєктів Вікімедіа. Автоматизоване тестування – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Автоматизоване_тестування (дата звернення: 19.12.2023).
7. Test Automation Fundamentals: A Study Guide for the Certified Test Automation Engineer Exam * Advanced Level Specialist * Istqb Compliant / J. Hartner et al. Rocky Nook, 2022.
8. AssertJ - fluent assertions java library. Site not found · GitHub Pages. URL: <https://assertj.github.io/doc/#assertj-core-common-assertions> (date of access: 19.12.2023).
9. Cohn M. Succeeding with agile: Software development using Scrum. Upper Saddle River, NJ : Addison-Wesley, 2010. 475 p.
10. Selenium WebDriver як інструмент для автоматизованого тестування | Головна сторінка. URL: <https://training.qatestlab.com/blog/technical-articles/selenium-webdriver/> (дата звернення: 19.12.2023).
11. Учасники проєктів Вікімедіа. Програмний каркас – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Програмний_каркас (дата звернення: 19.12.2023).

12. Contributors to Wikimedia projects. TestNG - Wikipedia. Wikipedia, the free encyclopedia. URL: <https://en.wikipedia.org/wiki/TestNG> (date of access: 19.12.2023).
13. TestNG. URL: <https://testng.org/doc/documentation-main.html> (date of access: 19.12.2023).
14. Учасники проєктів Вікімедіа. Неперервна інтеграція – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Неперервна_інтеграція (дата звернення: 19.12.2023).
15. Що таке Jenkins?. QualityAssuranceGroup. URL: <https://qagroup.com.ua/publications/shcho-take-jenkins/> (дата звернення: 19.12.2023).
16. The Practical Test Pyramid. martinowler.com. URL: <https://martinfowler.com/articles/practical-test-pyramid.html> (date of access: 24.01.2024).
17. Rasmusson J. The Way of the Web Tester: A Beginner's Guide to Automating Tests. Pragmatic Bookshelf, 2016. 258 p.
18. Gundecha U., Avasarala S. Selenium WebDriver 3 Practical Guide: End-to-end automation testing for web and mobile browsers with Selenium WebDriver, 2nd Edition. Packt Publishing, 2018. 280 p.
19. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010. 512 p.
20. Richardson M. A. J. Java For Testers: Learn Java fundamentals fast. Compendium Developments Ltd, 2015. 436 p.
21. Dosaj C. Self-Taught Software Tester a Step by Step Guide to Learn Software Testing Using Real-Life Project. Independently Published, 2020.
22. Myers G. J. The Art of Software Testing. New York : John Wiley & Sons, Ltd., 2004.
23. Jones N., Rosenthal C. Chaos Engineering: System Resiliency in Practice. O'Reilly Media, Incorporated, 2020. 400 p.

24. Hendrickson E. Explore it!: Reduce risk and increase confidence with exploratory testing. 2013. 162 p.
25. Weinberg G. M. Perfect software--and other illusions about testing. New York, NY : Dorset House Pub., 2008.
26. Gregory J., Crispin L. Agile Testing: A Practical Guide for Testers and Agile Teams. Pearson Education, Limited.
27. Aniche M. Effective Software Testing: A Developer's Guide. Manning Publications Co. LLC, 2022.
28. Angelov A. Design Patterns for High-Quality Automated Tests: Clean Code for Bulletproof Tests. Independently Published, 2021. 323 p.
29. IEEE Standards Association. IEEE Standards Association. URL: <https://standards.ieee.org/ieee/829/1217/> (date of access: 26.01.2024).
30. Exploratory Testing - A Definitive Guide. DevQA.io - For Developers and QAs. URL: <https://devqa.io/exploratory-testing-definitive-guide/> (date of access: 26.01.2024).
31. WebDriver Implicit, Explicit and Fluent Wait Examples. DevQA.io - For Developers and QAs. URL: <https://devqa.io/webdriver-explicit-implicit-fluent-wait/> (date of access: 26.01.2024).
32. Selenium CSS Selectors Examples. DevQA.io - For Developers and QAs. URL: <https://devqa.io/selenium-css-selectors/> (date of access: 26.01.2024).
33. Selenium Tutorial - Learn Browser Automation with Selenium WebDriver. DevQA.io - For Developers and QAs. URL: <https://devqa.io/selenium-tutorial/> (date of access: 26.01.2024).
34. Page Object Model Framework with Java and WebDriver. DevQA.io - For Developers and QAs. URL: <https://devqa.io/page-object-framework-java-webdriver/> (date of access: 26.01.2024).
35. Sharma P. Selenium with Java – A Beginner's Guide: Web Browser Automation for Testing using Selenium with Java. BPB Publications, 2022. 206 p.
36. Kovalenko D. Selenium Design Patterns and Best Practices. Packt Publishing, Limited, 2014.

37. Avasarala S. Selenium WebDriver Practical Guide. Packt Publishing, 2014. 264 p.
38. Gundecha U. Selenium Testing Tools Cookbook. Packt Publishing, Limited, 2012.
39. Collin M. Mastering Selenium WebDriver. Packt Publishing, 2015. 280 p.
40. Gundecha U. Selenium Testing Tools Cookbook. Packt Publishing, Limited, 2012.
41. Chatham M. Selenium by Example - Volume III: Selenium Webdriver. Lulu Press, Inc., 2014.

ДОДАТОК А

Лістинг коду фреймворку автоматизованого тестування

Файл pom.xml

```

<?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.dpu</groupId>
    <artifactId>Automation_framework</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java
-->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.15.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.testng/testng -->
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.8.0</version>
            <scope>test</scope>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.assertj/assertj-core -->

```

```
<dependency>
  <groupId>org.assertj</groupId>
  <artifactId>assertj-core</artifactId>
  <version>3.24.2</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.14.1</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.14.1</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>2.0.11</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.4.14</version>
  <scope>test</scope>
</dependency>

</dependencies>
```

```

<properties>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
</project>

```

Файл logback.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%date{HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
    </encoder>
  </appender>

  <root level="info">
    <appender-ref ref="STDOUT"/>
  </root>
</configuration>

```

Файл BaseTest.java

```

package tests;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.DataProvider;

```

```

import java.io.File;

public abstract class BaseTest {

    private static WebDriver driver;

    public static WebDriver getDriver() {
        return driver;
    }

    @BeforeClass
    public void setUp() {
        File file = new File("src/test/resources/chromedriver.exe");
        System.setProperty("webdriver.chrome.driver", file.getAbsolutePath());
        driver = new ChromeDriver();
        driver.navigate().to("https://taxer.ua/");
        driver.manage().window().maximize();
    }

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

    @DataProvider(name = "emailAndPass")
    public Object[][] dataProviderMethod() {

        return new Object[][]{
            {"GsmartXX@gmail.com", "XXXXXXXX"}
        };
    }
}

```

```
}
```

Файл TestCaseDemo.java

```
package tests;
```

```
import org.testng.annotations.Test;
```

```
import pages.*;
```

```
public class TestCaseDemo extends BaseTest {
```

```
    private String indCode = "1234567890";
```

```
    private String amount = "4200";
```

```
    private String amountAfterSave = "4 200,00 грн";
```

```
    private String PIB = "Микола Василенко";
```

```
    @Test(dataProvider = "emailAndPass")
```

```
    public void userFlowDemo(String email, String pass) {
```

```
        TaxerHomePage taxerHomePage = new TaxerHomePage();
```

```
        taxerHomePage.verifyWebAppLoaded(); // Крок 1
```

```
        taxerHomePage.clickEnterButton(); // Крок 2
```

```
        TaxerEnterScreen taxerEnterScreen = new TaxerEnterScreen();
```

```
        taxerEnterScreen.inputEmailAndPass(email, pass); // Крок 3
```

```
        taxerEnterScreen.pressLoginButton(); // Крок 3
```

```
        TaxerLeftPanel taxerLeftPanel = new TaxerLeftPanel();
```

```
        taxerLeftPanel.calendarButtonClick(); // Крок 4
```

```
        TaxerCalendarPage taxerCalendarPage = new TaxerCalendarPage();
```

```
        taxerCalendarPage.checkYear("2024"); // Крок 4
```

```
        taxerCalendarPage.checkEventPanel(); // Крок 4
```

```
taxerLeftPanel.currentEventsButtonClick(); // Крок 5
```

```
taxerLeftPanel.checkEPEvent(); // Крок 5
```

```
TaxerEPPayFormPage taxerEPPayFormPage = new TaxerEPPayFormPage();
```

```
taxerEPPayFormPage.formPaymentButtonClick(); // Крок 6
```

```
taxerEPPayFormPage.userValidDataPayment(PIB, indCode, amount); // Крок
```

```
7
```

```
taxerEPPayFormPage.clickSaveButton(); // Крок 7
```

```
taxerEPPayFormPage.openSavedButtonClick(); // Крок 8
```

```
taxerEPPayFormPage.checkEnteredUserDataIsCorrectInForm(PIB, indCode,  
amountAfterSave); // Крок 8
```

```
}
```

```
}
```

Файл BasePage.java

```
package pages;
```

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

```
import org.openqa.selenium.support.PageFactory;
```

```
import org.openqa.selenium.support.ui.WebDriverWait;
```

```
import java.time.Duration;
```

```
import static tests.BaseTest.getDriver;
```

```
public abstract class BasePage {
```

```
    WebDriver driver;
```

```
WebDriverWait wait;
```

```
public BasePage() {
    this.driver = getDriver();
    PageFactory.initElements(driver, this);
    wait = new WebDriverWait(driver, Duration.ofSeconds(10));
}
}
```

Файл TaxerCalendarPage.java

```
package pages;
```

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import static org.assertj.core.api.Assertions.assertThat;
```

```
public class TaxerCalendarPage extends BasePage{
```

```
    public TaxerCalendarPage() {
        super();
    }
```

```
    private static final Logger logger =
    LoggerFactory.getLogger(TaxerCalendarPage.class);
```

```

    @FindBy(xpath =
"/html/body/app-root/div/main/events-calendar-events-calendar/ui-page/ui-section/s
ection/div/ui-overlay/div/div/calendar-calendar/ui-overlay/div/div/div[1]/div/calenda
r-filters/div/div[1]")
    private WebElement yearDropDown;

    @FindBy(xpath =
"/html/body/app-root/div/main/events-calendar-events-calendar/ui-page/ui-section/s
ection/div/ui-overlay/div/div/calendar-calendar/ui-overlay/div/div/div[1]/div/calenda
r-filters/div/div[2]")
    private WebElement typeDropDown;

    @FindBy(xpath =
"/html/body/app-root/div/main/events-calendar-events-calendar/ui-page/ui-section/s
ection/div/ui-overlay/div/div/calendar-calendar/ui-overlay/div/div/div[1]/div/div[1]/
div/calendar-canvas/canvas")
    private WebElement eventsPanel;

    public void checkYear (String year){
        wait.until(ExpectedConditions.visibilityOf(yearDropDown));
        assertThat(yearDropDown.getText()).as("Wrong year is
shown").isEqualTo(year);
        logger.info("Correct year is displayed");
    }

    public void checkEventPanel(){
        wait.until(ExpectedConditions.visibilityOf(eventsPanel));
        assertThat(eventsPanel.isDisplayed()).as("Element has not been
displayed").isTrue();
        logger.info("Event panel is displayed");
    }

```

```

    }
}
Файл TaxerEnterScreen.java
package pages;

import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import static org.assertj.core.api.Assertions.assertThat;

public class TaxerEnterScreen extends BasePage {

    private static final Logger logger =
LoggerFactory.getLogger(TaxerEnterScreen.class);

    JavascriptExecutor js = (JavascriptExecutor) driver;

    public TaxerEnterScreen() {
        super();
    }

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/app-auth-popup/div/app-auth-popup-login/div/ui-s
ite-popup-content/div/div/div[1]/div[2]/form/ui-site-form-row[1]/div/div[1]/input")
    private WebElement emailField;

```

```

        @FindBy(xpath =
"/html/body/ui-popup/div/div/div/app-auth-popup/div/app-auth-popup-login/div/ui-s
ite-popup-content/div/div/div[1]/div[2]/form/ui-site-form-row[2]/div/div[1]/ui-pass
word/div/input")
        private WebElement passwordField;

        @FindBy(xpath =
"/html/body/ui-popup/div/div/div/app-auth-popup/div/app-auth-popup-login/div/ui-s
ite-popup-content/div/div/div[1]/div[2]/form/ui-site-form-row[4]/div/div[1]/div/ui-b
utton/button")
        private WebElement enterTaxerButton;

        @FindBy(xpath =
"/html/body/app-root/div/aside/div/layout-drawer-auth/div/ul/li[8]/div/span")
        private WebElement exitButton;

        @FindBy(xpath =
"/html/body/app-root/div/aside/div/layout-drawer-auth/div/ul/li[2]/a")
        private WebElement eventButton;

    public void inputEmailAndPass(String email, String pass){
        wait.until(ExpectedConditions.visibilityOf(emailField));
        wait.until(ExpectedConditions.visibilityOf(passwordField));
        emailField.sendKeys(email);
        passwordField.click();
        passwordField.sendKeys(pass);

        logger.info("Email and Pass entered correctly");

    }

    public void pressLoginButton() {

```

```

        wait.until(ExpectedConditions.visibilityOf(enterTaxerButton));
        enterTaxerButton.click();
        logger.info("Login button was clicked");
        wait.until(ExpectedConditions.visibilityOf(eventButton));
        js.executeScript("arguments[0].scrollIntoView(true);", exitButton);
        assertThat(exitButton.isDisplayed()).as("Element has not been
displayed").isTrue();
        logger.info("User is logged in");
    }
}

```

Файл TaxerEPPayFormPage.java

```

package pages;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import static org.assertj.core.api.Assertions.assertThat;

public class TaxerEPPayFormPage extends BasePage {

    public TaxerEPPayFormPage() {
        super();
    }

    private static final Logger logger =
LoggerFactory.getLogger(TaxerEPPayFormPage.class);

```

```
private TaxerLeftPanel taxerLeftPanel = new TaxerLeftPanel();
```

```

    @FindBy(xpath =
"/html/body/app-root/div/main/events-events/ui-page/div/div/div[2]/events-event-pa
nel/ui-box-panel/div[2]/div[2]/div[2]/events-event-action/div/ui-button[1]/button")
    private WebElement formPaymentButton;

    @FindBy(xpath =
"/html/body/app-root/div/main/events-events/ui-page/div/div/div[2]/events-event-pa
nel/ui-box-panel/div[2]/div[2]/div[2]/events-event-action/div/ui-button[1]/button/sp
an")
    private WebElement openSavedPayment;

    @FindBy(xpath =
"/html/body/app-root/div/main/events-events/ui-page/div/div/div[2]/events-event-pa
nel/ui-box-panel/div[2]/div[1]/div[1]/ui-heading/app-payment-status")
    private WebElement draftStatus;

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-payment-form-popup/div/ui-form-popu
p-content/div/div[2]/div/payments-payment-form/div/div/form/ui-form-row[1]/div/d
iv[1]/div[2]/input")
    private WebElement PIBField;

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-payment-form-popup/div/ui-form-popu
p-content/div/div[2]/div/payments-payment-form/div/div/form/ui-form-row[2]/div/d
iv[1]/div[2]/input")
    private WebElement bankIdentifierCodeField;

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-payment-form-popup/div/ui-form-popu
p-content/div/div[2]/div/payments-payment-form/div/div/form/ui-form-row[4]/div/d
iv[1]/div[2]/div/input")
    private WebElement amountField;

```

```

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-payment-form-popup/div/ui-form-popu
p-content/div/div[3]/div/ui-form-popup-actions/div/ui-form-popup-button-group[1]/
div/div[2]/ui-form-popup-button-group-item/div/div[1]")
    private WebElement saveButton;

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-view-payment-popup/div/ui-popup-hea
der/div/div/h2/span")
    private WebElement savedPaymentPage;

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-view-payment-popup/div/ui-popup-cont
ent/div/div/div/div[1]/table/tr[1]/td[2]")
    private WebElement PIBSavedPayment;

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-view-payment-popup/div/ui-popup-cont
ent/div/div/div/div[1]/table/tr[2]/td[2]")
    private WebElement bankIdentifierSavedPayment;

    @FindBy(xpath =
"/html/body/ui-popup/div/div/div/payments-view-payment-popup/div/ui-popup-cont
ent/div/div/div/div[1]/table/tr[7]/td[2]")
    private WebElement amountSavedPayment;

    public void formPaymentButtonClick(){
        wait.until(ExpectedConditions.visibilityOf(formPaymentButton));
        formPaymentButton.click();
        logger.info("Form button is pressed");

    }public void openSavedButtonClick(){
        wait.until(ExpectedConditions.visibilityOf(openSavedPayment));
        openSavedPayment.click();
    }

```

```

        logger.info("Form button is pressed");
    }

```

```

    public void userValidDataPayment(String fullName, String indCode, String
amount){
        wait.until(ExpectedConditions.visibilityOf(saveButton));
        PIBField.click();
        PIBField.sendKeys(fullName);
        bankIdentifierCodeField.click();
        bankIdentifierCodeField.sendKeys(indCode);
        amountField.click();
        amountField.sendKeys(amount);
        logger.info("User valid data was entered");

    }

```

```

    public void clickSaveButton() {
        wait.until(ExpectedConditions.visibilityOf(saveButton));
        saveButton.click();
        logger.info("Save button was clicked");
        wait.until(ExpectedConditions.visibilityOf(draftStatus));

    }

```

```

    public void checkEnteredUserDataIsCorrectInForm(String fullName, String
indCode, String amount){
        wait.until(ExpectedConditions.visibilityOf(savedPaymentPage));
        wait.until(ExpectedConditions.visibilityOf(PIBSavedPayment));
        assertThat(PIBSavedPayment.getText()).as("Wrong Name and Surname is
shown").isEqualTo(fullName);
    }

```

```

        logger.info("Name and Surname are correct");
        assertThat(bankIdentifierSavedPayment.getText()).as("Wrong Bank Identifier
is shown").isEqualTo(indCode);
        logger.info("Bank Identifier is correct");
        assertThat(amountSavedPayment.getText()).as("Wrong amount is
shown").isEqualTo(amount);
        logger.info("Amount is correct");

    }
}

```

Файл TaxerHomePage.java

```

package pages;

import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.ui.ExpectedConditions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import static org.assertj.core.api.Assertions.assertThat;

public class TaxerHomePage extends BasePage{

    private static final Logger logger =
LoggerFactory.getLogger(TaxerHomePage.class);

```

```

@FindBy(xpath = "/html/body/app-root/div[1]/header/layout-header/div/div")
private WebElement siteHeader;

@FindBy(xpath = "/html/body/app-root/div[1]/footer/layout-footer/div[1]")
private WebElement siteFooter;

@FindBy(xpath =
"/html/body/app-root/div[1]/main/index-index/ui-page/ui-section[4]/section/div")
private WebElement mainCalendar;

@FindBy(xpath =
"/html/body/app-root/div[1]/header/layout-header/div/div/div[2]/div[1]/ui-button[1]/
button")
private WebElement enterButton;

@FindBy(xpath =
"/html/body/ui-popup/div/div/div/app-auth-popup/div/app-auth-popup-login/div/ui-s
ite-popup-content/div/div/div[1]")
private WebElement enterPopup;

```

```

JavascriptExecutor js = (JavascriptExecutor) driver;

```

```

public TaxerHomePage() {
    super();
}

```

```

public void verifyWebAppLoaded(){
    wait.until(ExpectedConditions.visibilityOf(siteHeader));
    js.executeScript("arguments[0].scrollIntoView(true);", mainCalendar);
    wait.until(ExpectedConditions.visibilityOf(mainCalendar));
    js.executeScript("arguments[0].scrollIntoView(true);", siteFooter);
    wait.until(ExpectedConditions.visibilityOf(siteFooter));
}

```

```

        assertThat(siteHeader.isDisplayed()).as("Element has not been
displayed").isTrue();
        assertThat(mainCalendar.isDisplayed()).as("Element has not been
displayed").isTrue();
        assertThat(siteFooter.isDisplayed()).as("Element has not been
displayed").isTrue();
        logger.info("Web Application loaded correctly");
    }

    public void clickEnterButton() {

        wait.until(ExpectedConditions.visibilityOf(enterButton));
        enterButton.click();
        logger.info("Enter button was clicked");
        wait.until(ExpectedConditions.visibilityOf(enterPopup));
        logger.info("Enter popup was opened");

    }
}

```

Файл TaxerLeftPanel.java

```

package pages;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import static org.assertj.core.api.Assertions.assertThat;

```

```

public class TaxerLeftPanel extends BasePage {

    public TaxerLeftPanel() {
        super();
    }

    private static final Logger logger =
LoggerFactory.getLogger(TaxerLeftPanel.class);

    @FindBy(xpath =
"/html/body/app-root/div/aside/div/layout-drawer-auth/div/ul/li[2]/ul/li[2]/a")
    private WebElement calendarButton;

    @FindBy(xpath =
"/html/body/app-root/div/main/events-calendar-events-calendar/ui-page/ui-section/s
ection")
    private WebElement calendarPanel;

    @FindBy(xpath =
"/html/body/app-root/div/aside/div/layout-drawer-auth/div/ul/li[2]/a")
    private WebElement eventButton;

    @FindBy(xpath =
"/html/body/app-root/div/aside/div/layout-drawer-auth/div/ul/li[2]/ul/li[1]/a")
    private WebElement currentEventsButton;

    @FindBy(xpath =
"/html/body/app-root/div/main/events-events/ui-page/div/div/div[2]/events-event-pa
nel/ui-box-panel/div[2]/div[1]/div[1]/ui-heading")
    private WebElement EPEvent;

    public void calendarButtonClick(){

```

```

        eventButton.click();
        wait.until(ExpectedConditions.visibilityOf(calendarButton));
        calendarButton.click();
        logger.info("Calendar button was clicked");
        wait.until(ExpectedConditions.visibilityOf(calendarPanel));
        assertThat(calendarPanel.isDisplayed()).as("Element has not been
displayed").isTrue();
        logger.info("Calendar Panel is displayed");
    }

    public void currentEventsButtonClick() {
        wait.until(ExpectedConditions.visibilityOf(currentEventsButton));
        currentEventsButton.click();
        logger.info("Current events button was clicked");

    }

    public void checkEPEvent() {
        wait.until(ExpectedConditions.visibilityOf(EPEvent));
        assertThat(EPEvent.isDisplayed()).as("Element has not been
displayed").isTrue();
        logger.info("EP event is present in list");
    }
}

```