Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
Факультет інформаційних технологій
Кафедра інформаційних технологій та комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня *магістра*

Студента Хари Германа Леонідовича

академічної групи *126 – 22М – 1*

спеціальності *126 «Інформаційні системи та технології»*

на тему: *Розробка інформаційної системи тестування веб-додатків*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		Рейтинговою	**Інституційною**	
кваліфікаційної роботи	*к.т.н., доц. Соколова Н.О.*			
розділів:				
Рецензент	*к.т.н., доц. Клименко А.В.*			
Нормоконтролер	*д.т.н., проф. Коротенко Г.М.*			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри
Інформаційних технологій та комп'ютерної інженерії
(повна назва)
_____ *д.т.н., проф. Гнатушенко В.В.*
(підпис) (прізвище, ініціали)
«_____» _____2023року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *магістра*

студенту Харі Г. Л. **академічної групи** *126-22М-1*
спеціальності: *126 «Інформаційні системи та технології»*
на тему «Розробка інформаційної системи тестування веб-додатків»
затверджену наказом ректора НТУ «Дніпровська політехніка»від *№1227-с від 09.10.2023*

Розділ	Зміст	Терміни виконання
Розділ 1.	*Аналіз стану предметної області рішення задачі*	9.10.2023 – 24.10.2023
Розділ 2.	*Визначення основних підходів до вибору інструментів для створення інформаційної системи тестування веб-додатків*	24.10.2023 – 11.11.2023
Розділ 3.	*Розробка інформаційної системи тестування веб-додатків*	12.11.2023 – 1.12.2023

Завдання видано _____ *доц. Соколова Н.О.*
(підпис) (прізвище, ініціали)

Дата видачі: *09.10.2023*
Дата подання до екзаменаційної комісії:_____

Прийнято до виконання _____ _____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 94 с., 17 рис., 9 табл., 2 додатки, 13 джерел.

Об’єкт дослідження: розробка інформаційної системи тестування веб-додатків.

Предмет дослідження: тести та їх ефективність при розробці веб додатків.

Мета роботи полягає в аналізі підходів розробки інформаційної системи тестування веб-додатків.

Використані методі дослідження: теоретичний метод - моделювання, аналіз, порівняння та абстрагування, а також емпіричний метод дослідження.

У вступі подано стан проблеми, сформульована мета і підхід до виконання даної кваліфікаційної роботи, а також обґрунтована її актуальність та сформовані задачі, які потрібно вирішити.

У першому розділі розглянуто основи тестування, його класифікацію, різноманітні типи тестів та методології їх застосування, було прийнято рішення щодо розробки інформаційної системи для тестування веб-додатків.

У другому розділі були розглянуті необхідні інструменти для розробки інформаційної системи тестування веб додатків. Були розглянути підходи до побудови архітектури системи, а також для написання тестів. Також було описано предмет тестування.

У третьому розділі були розроблені тести, що відповідають технічним вимогам, що дозволяє перевірити різні аспекти функціональності та надійності веб-додатку. Також було підключено систему звітування.

ТЕСТУВАННЯ, SELINIUM, TESTNG, PAGE OBJECT, XPATH, ALURE, MAVEN, WEBDRIVER, JAVA.

ABSTRACT

Explanatory note: 94 pages, 17 figures, 9 tables, 2 appendixs, 13 sources.

Research object: Development of an information system for testing web applications.

Subject of consideration: tests and their effectiveness in developing web add-ons.

The aim of the work is to analyze the approaches to developing an information system for testing web applications.

Research methods used: theoretical methods - modeling, analysis, comparison, abstraction, as well as empirical research methods.

The introduction outlines the problem statement, formulates the goal and approach to completing this qualification work, justifies its relevance, and defines the tasks to be addressed.

The first section covers the fundamentals of testing, its classification, various types of tests, and methodologies for their application, leading to the decision to develop an information system for testing web applications.

The second section discusses necessary tools for developing an information system for testing web applications. Approaches to system architecture construction and test writing are examined, along with a description of the testing subject.

The third section involves the development of tests conforming to technical requirements, allowing verification of different aspects of a web application's functionality and reliability. Additionally, a reporting system has been integrated.

TESTING, SELENIUM, TESTNG, PAGE OBJECT, XPATH, ALLURE, MAVEN, WEBDRIVER, JAVA.

ЗМІСТ

ВСТУП

Актуальність роботи. Кожного дня з’являються нові веб додатки у самих різних напрямках та різних сферах: бізнес, освіта, медицина, спорт або розваги, але для того щоб додаток приносив користь, прибуток або задоволення він повинне відповідати вимогам і саме для цього кожен програмний продукт повинен прийти через етап тестування.

Розробка інформаційної системи тестування веб-додатків є важливим етапом життєвого циклу програмного забезпечення. Архітектура повинна задовольняти сучасним вимогам, підлаштовуватися під веб додатки та бути гнучкою аби тестування було максимально ефективним.

Тестування веб додатків визначається необхідністю гарантувати високу якість, ефективність та продуктивність під час їх використання.

Одним із ключових аспектів є створення модульної структури, що дозволяє легко розширювати систему для адаптації до різних видів тестувань. Це дозволяє охопити більшу частину функціоналу. Використання автоматизованих тестів дозволяє ефективно виявляти помилки та забезпечує швидке внесення змін для поліпшення продукту. Інтеграція із засобами ведення версій та засобами звітності забезпечує зручний моніторинг та аналіз результатів, що в свою чергу позитивно впливає на розробку продукту.

Також тестування має й економічний влив. Виявлення помилок на ранніх етапах проектування або розробки дозволяє зменшити витрати ресурсів та коштів, так як їх виправляння на ранніх етапах більше дешевша.

Предметом дослідження є тести та їх ефективність при розробці веб додатків.

Об’єкт дослідження: розробка інформаційної системи тестування веб-додатків.

Мета роботи полягає в аналізі підходів розробки інформаційної системи тестування веб-додатків.

Для досягнення поставленої мети необхідно виконати наступний комплекс задач:

1. Визначення поняття тестування
2. Огляд методологій тестування
3. Огляд рівнів тестування
4. Опис технологічних рішень
5. Опис предмету тестування
6. Розробка інформаційної системи тестування веб-додатків

Наукова новизна полягає у реалізації концепції інформаційної системи для тестування веб-додатків. Цей проект орієнтований на створення передового інструментарію для забезпечення високої ефективності, точності та автоматизації процесу тестування веб-додатків.

Сутність наукової новизни полягає у використанні передових підходів і технологій, що спрямовані на покращення якості та швидкості тестування. Ця інформаційна система включає у себе сучасні інструменти для автоматизації тестів, методи валідації функціональності та безпеки, а також аналіз результатів для виявлення та виправлення помилок на ранніх етапах розробки.

Основною метою цього дослідження є створення інформаційної системи, яка не лише спрощує тестування веб-додатків, а й допомагає виявляти недоліки та оптимізувати процес розробки, підвищуючи загальну якість програмного забезпечення.

Практичне значення. Висвітлення досліджень у цій роботі дозволяє існуючим та майбутнім розробникам веб-додатків зрозуміти, як вдосконалити процес тестування для забезпечення високої якості продукту. Отримані знання можуть бути використані в освітніх програмах для навчання принципам та практиці тестування веб-додатків. Дані результати також можуть стати корисними для компаній та комерційних розробників, сприяючи покращенню їхніх тестових стратегій та продуктів.

РОЗДІЛ 1

АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧ

1.1 Аналіз предметної області

Тестування є одним із етапів життєвого циклу розробки програмного забезпечення (SDLC).

Життєвий цикл розробки програмного забезпечення (англійською повна назва, software development lifecycle) – це економічний та швидкий процес, який використовують групи розробників для проектування та створення високоякісного програмного забезпечення. Мета SDLC – мінімізувати проектні ризики за рахунок попереднього планування, внаслідок чого програмне забезпечення відповідатиме очікуванням клієнтів під час виробництва та на інших етапах[1].

Життєвий цикл розробки програмного забезпечення (SDLC) описує декілька завдань, необхідних створення програмного додатка. Процес розробки відбувається через кілька етапів, коли розробники додають нові функції та виправляють помилки у програмному забезпеченні[1].

Деталі процесу SDLC є різними для різних команд. Тим не менш, нижче описано деякі загальні етапи SDLC.

Планування.

Етап планування зазвичай передбачає виконання таких завдань, як аналіз витрат та вигод, складання розкладу, оцінка та розподіл ресурсів. Команда розробників збирає вимоги від кількох зацікавлених сторін, таких як клієнти, внутрішні та зовнішні експерти та менеджери, щоб створити документ специфікації вимог до програмного забезпечення[1].

Документ встановлює очікування та визначає спільні цілі, які допомагають у плануванні проекту. Команда оцінює витрати, складає графік та розробляє докладний план досягнення поставленої мети[1].

Проектування.

На етапі проектування інженери-програмісти аналізують вимоги та визначають найкращі рішення для створення програмного забезпечення. Наприклад, вони можуть розглянути можливість інтеграції вже існуючих модулів, зробити вибір технології та визначити засоби розробки. Вони розглянуть, як найкраще інтегрувати нове програмне забезпечення в існуючу IT-інфраструктуру організації[1].

Впровадження.

На етапі застосування команда розробників кодує товар. Вони аналізують вимоги, щоб визначити дрібніші завдання кодування, які можна виконувати щодня для досягнення кінцевого результату[1].

Тестування.

Команда розробників поєднує автоматизацію та ручне тестування для перевірки програмного забезпечення на наявність помилок. Аналіз якості передбачає тестування програмного забезпечення на наявність помилок та перевірку його відповідності вимогам замовника. Оскільки багато команд відразу ж тестують написаний ними код, етап тестування часто проходить паралельно з етапом розробки[1].

Розгортання.

Коли команди розробляють програмне забезпечення, вони виконують кодування та тестування на копії програмного забезпечення, відмінну від тієї, до якої мають доступ користувачі. Програмне забезпечення, яке використовують клієнти, називається виробничим, у той час як інші копії, як правило, знаходяться в середовищі складання, або тестуванні[1].

Наявність окремих середовищ збирання та виробництва гарантує, що клієнти зможуть і надалі використовувати програмне забезпечення навіть у процесі його зміни чи оновлення. Етап розгортання передбачає виконання кількох завдань щодо переміщення останньої копії складання у виробниче середовище, таких як упаковка, конфігурація середовища та встановлення[1].

Обслуговування.

На етапі обслуговування, окрім інших завдань, команда виправляє помилки, вирішує проблеми клієнтів та керує змінами програмного забезпечення. Крім того, команда стежить за загальною продуктивністю системи, безпекою та зручністю роботи користувачів, щоб визначити нові способи покращення існуючого програмного забезпечення[1].

І хоча тестування відокремлюються окремим етапом, проте воно присутнє на кожному етапі від планування до обслуговування.

1.2 Принципи тестування

Тестування показує наявність дефектів:

Тестування може показати наявність дефектів в програмі, але не довести їх відсутність. Тим не менш, важливо складати тест-кейси, які будуть знаходити якомога більше багів. Таким чином, при належному тестовому покритті, тестування дозволяє знизити вірогідність наявності дефектів в програмному забезпеченні. В той же час, навіть якщо дефекти не були знайдені в процесі тестування, не можна стверджувати, що їх немає[2].

Вичерпне тестування. Неможливо провести вичерпне тестування, яке би покривало всі комбінації користувацького вводу та станів системи, за виключенням найбільш примітивних випадків. Замість цього необхідно використовувати аналіз ризиків та розташування пріоритетів, що дозволить більш ефективно розподілити зусилля по забезпеченню якості ПЗ[2].

Раннє тестування. Тестування повинне починатися якомога раніше в життєвому циклі розробки програмного забезпечення, і його зусилля повинні бути сконцентровані на визначених цілях[2].

Скупчення дефектів. Різні модулі системи можуть містити різну кількість дефектів – тобто, щільність скупчення дефектів в різних елементах програми може відрізнятися. Зусилля по тестуванню повинні розподілятися пропорційно фактичній щільності дефектів. В основному, більшу частину

дефектів знаходять в обмеженій кількості модулів. Це прояв *принципу Парето*: 80% проблем містяться в 20% модулів[2].

Парадокс пестициду. Проганяючи одні й ті ж тести знову та знову, тестувальник стикається з тим, що вони знаходять все менше нових помилок. Оскільки система еволюціонує, багато із раніше знайдених дефектів виправляють і старі тест-кейси більше не спрацьовують[2].

Щоб подолати цей парадокс, необхідно періодично вносити зміни в набори тестів, що використовуються, рецензувати та коригувати їх з тим, щоб вони відповідали новому стану системи та дозволяли знаходити якомога більшу кількість дефектів[2].

Тестування залежить від контенту. Вибір методології, техніки та типу тестування буде напряму залежати від природи самої програми. Наприклад, програмне забезпечення для медичних цілей потребує більш суворої та ретельної перевірки, ніж, скажімо, комп'ютерна гра. З тих же міркувань, сайт із великою відвідуваністю повинен пройти через серйозне тестування продуктивності, щоб показати можливості роботи в умовах великого навантаження[2].

Міф про відсутність помилок. Той факт, що тестування не виявило дефектів, ще не означає, що програма готова до релізу. Знаходження та виправлення дефектів будуть не важливі, якщо система виявиться незручною у використанні, та не буде задовольняти очікуванням та вимогам користувача[2].

1.3 Методології тестування

Існують основні три типи методологій тестування:

- Тестування чорної скриньки
- Тестування білої скриньки
- Тстування сірої скриньки

1.3.1Тестування чорної скриньки

Тестування чорної скриньки — це тип тестування програмного забезпечення, у якому функціональність програмного забезпечення невідома. Тестування проводиться без внутрішнього знання продуктів. Його також називають функціональним тестуванням. Тестування чорної скриньки зосереджується на зовнішніх атрибутах і поведінці програмного забезпечення. Цей тип тестування розглядає очікувану поведінку програмного забезпечення програми з точки зору користувача[3].

Приклад: тестувальник проводить тестування веб-сайту, не знаючи особливостей його реалізації, використовуючи тільки передбачені розробником поля введення та кнопки. Джерело очікуваного результату – специфікація[4].

Оскільки це тип тестування, за визначенням він може включати інші його види. Тестування чорної скриньки може бути як функціональним, так і нефункціональним. Функціональне тестування передбачає перевірку роботи функцій системи, а нефункціональне – відповідно, загальні характеристики програми[4].

Техніка чорної скриньки застосовна на всіх рівнях тестування (від модульного до приймаючого), для яких існує специфікація. Наприклад, при здійсненні системного або інтеграційного тестування, вимоги або функціональна специфікація будуть основою для написання тест-кейсів[4].

Техніки тест-дизайну, засновані на використання чорної скриньки, включають:

- Класи еквівалентності.
- Аналіз граничних значень.
- Таблиці рішень.
- Діаграми зміни стану.
- Тестування всіх пар.

Переваги [4]:

- Тестування проводиться з позиції кінцевого користувача і може допомогти виявити неточності і протиріччя в специфікації.
- Тестувальнику немає необхідності знати мови програмування і заглиблюватися в особливості реалізації програми.
- Тестування може проводитися фахівцями, незалежними від відділу розробки, що допомагає уникнути упередженого ставлення.
- Можна починати писати тест-кейси, як тільки готова специфікація.

Недоліки [4]:

- Тестується тільки дуже обмежена кількість шляхів виконання програми.
- Без чіткої специфікації (а це швидше реальність на багатьох проектах) досить важко скласти ефективні тест-кейси.
- Деякі тести можуть виявитися надмірними, якщо вони вже були проведені розробником на рівні модульного тестування.

Вибір чорної скриньки (Black Box testing) має свої переваги у тестуванні програмного забезпечення:

Незалежність від реалізації. Тестування чорного скриньки дозволяє ігнорувати внутрішню реалізацію програми, фокусуючись лише на вхідних та вихідних даних. Це дає можливість тестувати систему незалежно від того, як вона була розроблена.

Орієнтованість на користувача. Тестування чорного скриньки орієнтоване на зовнішнє поведінку системи, що дозволяє перевірити, чи відповідає програма вимогам користувача та бізнес-потребам.

Комплексне тестування. Використання чорного скриньки дозволяє проводити тести на всіх рівнях, включаючи функціональність, узгодження даних, ефективність та безпеку.

Необхідність реального взаємодії. При чорному ящику тести робляться з позиції зовнішнього користувача системи, що відображає реальні сценарії взаємодії із програмою.

Універсальність тестування. Чорна скринька може використовуватися на будь-якому етапі розробки, незалежно від стану проекту.

1.3.2. Тестування білої скриньки

Тестування білої скриньки — це техніка тестування програмного забезпечення, яка перевіряє програмне забезпечення на основі знань про внутрішні структури даних, фізичний логічний потік і архітектуру на рівні вихідного коду. Це тестування працює, дивлячись на тестування з точки зору розробника. Це тестування також відоме як тестування скляної коробки, тестування прозорої коробки, структурне тестування або нефункціональне тестування[3].

Приклад: тестувальник, який, як правило, є програмістом, вивчає реалізацію коду поля введення на веб-сторінці, визначає всі передбачені (як правильні, так і неправильні) і не передбачені користувальницькі вводи, і порівнює фактичний результат виконання програми з очікуваним. При цьому очікуваний результат визначається саме тим, як повинен працювати код програми[4].

Тестування методом білої скриньки схоже на роботу механіка, який вивчає двигун машини, щоб зрозуміти, чому вона не заводиться[4].

Техніка білої скриньки застосовна на різних рівнях тестування – від модульного до системного, але головним чином застосовується саме для реалізації модульного тестування компонента його автором[4].

Переваги [4]:

- Тестування може проводитися на ранніх етапах: немає необхідності чекати створення користувальницького інтерфейсу.
- Можна провести більш ретельне тестування, з покриттям великої кількості шляхів виконання програми.

Недоліки [4]:

- Для виконання тестування білого скриньки необхідно велика кількість спеціальних знань.
- При використанні автоматизації тестування на цьому рівні, підтримка тестових скриптів може виявитися достатньо накладною, якщо програма часто змінюється.

1.3.3. Тестування сірої скриньки

Тестування сірої скриньки — це комбінація методів тестування чорної скриньки та методу тестування білої скриньки в тестуванні програмного забезпечення. Тестування сірої скриньки передбачає вхідні та вихідні дані програми для цілей тестування, але дизайн тесту перевіряється за допомогою інформації про код. Тестування сірої скриньки добре підходить для тестування веб-додатків, оскільки воно враховує середовище проектування високого рівня та умови взаємодії[3].

Приклад: тестувальник вивчає код програми з тим, щоб краще розуміти принципи її роботи і вивчити можливі шляхи її виконання. Таке знання допоможе написати тест-кейс, який напевно буде перевіряти певну функціональність[4].

Техніка сірої скриньки може застосовуватися на різних рівнях тестування – від модульного до системного, але головним чином застосовується на інтеграційному рівні для перевірки взаємодії різних модулів програми[4].

Детальне порівняння різних методологій представлено в таб.1.1[5].

Таблиця 1.1

Порівняння методологій тестування

Чорна скринька	**Біла скринька**	**Сіра скринька**
Він підходить для функціонального або бізнес-тестування.	Він підходить для глибокого функціонального або бізнес-доменного тестування.	Використовується для всіх.
Це тестування передбачає перевірку вихідних даних для заданих вхідних даних, причому програма тестується як метод чорної скриньки.	Тут ми маємо кращий вибір вхідних даних і можливість отримувати результати тестування з бази даних для порівняння з очікуваними результатами.	Він передбачає структурне тестування та забезпечує логічне покриття, рішення тощо в коді.
Це також називається тестуванням непрозорого боксу, тестуванням закритого боксу, тестуванням введення-виведення, тестуванням на основі даних, поведінковим, функціональним тестуванням	Це також називається напівпрозорим тестуванням	Це також називається тестуванням Glass-box, тестуванням Clear-box, тестуванням на основі дизайну, тестуванням на основі логіки, структурним тестуванням і тестуванням на основі коду.
Він підходить для функціонального або бізнес-тестування.	Він підходить для глибокого функціонального або бізнес-доменного тестування.	Використовується для всіх.

Таблиця 1.1 (продовження)

Чорна скринька	**Біла скринька**	**Сіра скринька**
Це тестування передбачає перевірку вихідних даних для заданих вхідних даних, причому програма тестується як метод чорної скриньки.	Тут ми маємо кращий вибір вхідних даних і можливість отримувати результати тестування з бази даних для порівняння з очікуваними результатами.	Він передбачає структурне тестування та забезпечує логічне покриття, рішення тощо в коді.
Це також називається тестуванням непрозорого боксу, тестуванням закритого боксу, тестуванням введення-виведення, тестуванням на основі даних, поведінковим, функціональним тестуванням	Це також називається напівпрозорим тестуванням	Це також називається тестуванням Glass-box, тестуванням Clear-box, тестуванням на основі дизайну, тестуванням на основі логіки, структурним тестуванням і тестуванням на основі коду.
Це тестування має низьку деталізацію.	Це тестування має середній рівень деталізації.	Це тестування має високий рівень деталізації.
Це роблять кінцеві користувачі, а також тестувальники та розробники.	Це виконується кінцевими користувачами (називається тестуванням прийнятності користувача), а також тестувальниками та розробниками.	Зазвичай це роблять тестувальники та розробники.

Продовження табл. 1.1

Чорна скринька	**Біла скринька**	**Сіра скринька**
Це тестування має низьку деталізацію.	Це тестування має середній рівень деталізації.	Це тестування має високий рівень деталізації.
Це роблять кінцеві користувачі, а також тестувальники та розробники.	Це виконується кінцевими користувачами (називається тестуванням прийнятності користувача), а також тестувальниками та розробниками.	Зазвичай це роблять тестувальники та розробники.
Тут не потрібно знати внутрішні дані.	Тут відомі внутрішні елементи, що стосуються тестування.	Тут відомий внутрішній код програми та бази даних.
Ймовірно, він буде менш вичерпним, ніж два інших.	Це щось посередині.	Найбільш вичерпний серед усіх трьох.
Він заснований на вимогах і тестових випадках на функціональних специфікаціях, оскільки внутрішні елементи невідомі.	Він забезпечує кращу різноманітність/глибину тестових випадків завдяки високому рівню знань внутрішніх органів.	Він може використовувати код із різними даними.

Продовження табл. 1.1

Чорна скринька	Біла скринька	Сіра скринька
Методи розробки тестів чорної скриньки: • Тестування таблиці рішень • Тестування всіх пар • Розбиття еквівалентності Помилка вгадування	Техніка проектування тесту сірої скриньки: • Матричне тестування • Регресійне тестування • Тестування шаблону Тестування ортогонального масиву	Методи проектування тестів білої скриньки: •Тестування контрольного потоку •Тестування потоку даних Тестування гілок
Тестування Black Box забезпечує стійкість і захист від вірусних атак.	Тестування Grey Box не забезпечує стійкість і захист від вірусних атак.	Тестування White Box не забезпечує стійкість і безпеку проти вірусних атак.

1.4. Рівні тестування

Концепція тестової піраміди

Майк Кон запропонував цю концепцію у своїй книзі «Успіх за допомогою Agile» [6]. Це чудова візуальна метафора, яка говорить вам подумати про різні рівні тестування. Він також повідомляє вам, скільки тестування потрібно зробити на кожному шарі. Модель піраміди можна побачити на рис. 1.1.

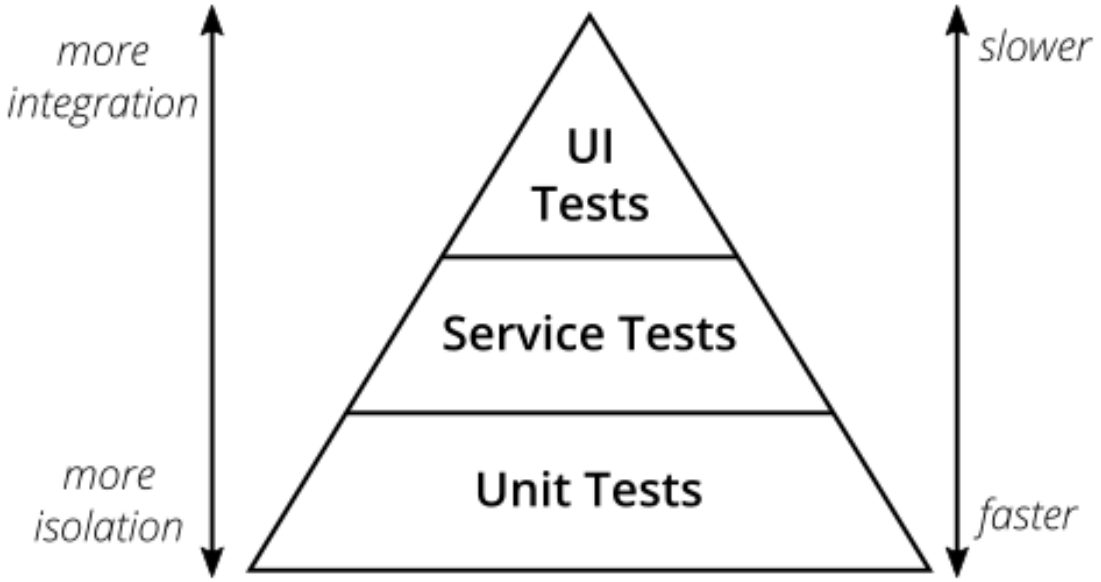

Рисунок 1.1 - Концепція тестової піраміди

Оригінальна тестова піраміда Майка Кона складається з трьох рівнів, з яких повинен складатися ваш набір тестів (знизу вгору):

- Модульні тести
- Сервісні тести
- Тести інтерфейсу користувача

З сучасної точки зору тестова піраміда виглядає надто спрощеною і тому може ввести в оману[6].

Проте завдяки своїй простоті суть тестової піраміди є хорошим емпіричним правилом, коли справа доходить до створення власного набору тестів. Найкраще запам’ятати дві речі з оригінальної тестової піраміди Кона[6]:

- Необхідно писати тести з різною деталізацією[6]
- Чим вищий рівень ви отримаєте, тим менше тестів вам доведеться проходити[6]

Дотримуйтеся пірамідальної форми, щоб створити здоровий, швидкий і зручний набір тестів: напишіть багато маленьких і швидких модульних тестів.

Напишіть кілька більш грубих тестів і дуже мало тестів високого рівня, які перевіряють вашу програму від кінця до кінця[6].

1.4.1 Модульні тести (Unit)

Модульне тестування — це метод тестування програмного забезпечення, у якому створюються модулі, тобто невеликі частини програми, поведінка кожного з яких перевіряється окремо. Модульне тестування виконується на етапі розробки програми. Модулем може бути будь-що, наприклад, процедура або функція[7].

В своїй більшості вони виконуються з боку програміста, а не тестувальника. Модульне тестування поділяється на ручне та автоматичне. Автоматизація модульних тестів – хороша практика, її також можна виконувати вручну. При ручному підході використовується документ із покроковими інструкціями.

Складові автоматизованого підходу[7]:

- Розробник пише модульні тести, щоб перевірити функціональність конкретної частини програми. Вони закоментовані та будуть видалені пізніше, після успішного розгортання програми[7].
- Функція має бути ізольована, щоб її можна було перевірити ретельніше. Найкраща практика unit-тестування — копіювати та вставляти код у тестове середовище, замість роботи у природному середовищі. Ізольований код допомагає виявити та усунути залежності між кодом, що тестується, і просторами даних[7].
- Існує середовище модульного тестування розробки автоматизованих тестових випадків. Це середовище автоматизації допомагає писати код і перевіряє, чи правильно написаний код. Під час виконання модульних тестів платформа реєструє статус тестових випадків. Залежно від серйозності збоїв, структура може зупинити подальше тестування[7].

- Робочий процес модульного тестування поділено на чотири категорії: це створення тестових прикладів, огляд, базовий рівень та виконання тестових прикладів[7].

1.4.2 Тестування сервісів (Service Testing)

Якість послуги, яку надає веб-додаток, можна визначити, включаючи всі його атрибути, такі як функціональність, продуктивність, надійність, зручність використання, безпека тощо[8].

Однак тут відокремлюємо три конкретні цілі сервісу, які підпадають під пильну увагу того, що називатимемо «тестуванням сервісу»[8].

Ці цілі:

Продуктивність: сервіс повинна реагувати на потреби користувачів, одночасно витримуючи навантаження, які на неї покладаються[8].

Надійність: якщо сервіс розроблений таким чином, щоб бути стійким до збоїв, він повинен бути надійним та/або продовжувати надавати послугу навіть у разі збою[8].

Керованість: сервісом має бути можливість керувати, налаштовуватися чи змінюватися без погіршення якості сервісу, помітного для кінцевих користувачів. Керованість або операційне тестування має на меті продемонструвати, що процедури адміністрування системи, керування, резервного копіювання та відновлення працюють ефективно[8].

У всіх трьох випадках потрібно змоделювати навантаження користувача для ефективного проведення тестів. Цілі продуктивності, надійності та керованості існують у контексті реальних клієнтів, які використовують сайт для ведення бізнесу[8].

Швидкість реагування (у цьому випадку час, необхідний одному системному вузлу для відповіді на запит іншого) сайту безпосередньо пов’язана з ресурсами, доступними в технічній архітектурі[8].

Чим більше клієнтів користуються програмним продуктом, тим менше технічних ресурсів доступно для обслуговування запитів кожного користувача, а час відповіді зменшиться[8].

Очевидно, що сервіс, який мало завантажений, має менше шансів на збій. Велика частина складності програмного та апаратного забезпечення існує для того, щоб відповідати вимогам до ресурсів у межах технічної архітектури, коли сайт сильно завантажений[8].

Коли сайт завантажується (або перевантажується), конфліктними запитами на ресурси повинні керувати різні компоненти інфраструктури, такі як серверні та мережеві операційні системи, системи управління базами даних, веб-серверні продукти, брокери запитів на об’єкти, проміжне програмне забезпечення тощо[8].

Ці компоненти інфраструктури зазвичай надійніші, ніж спеціально створений код програми, який вимагає ресурсів, але збої можуть виникати в будь-якому[8]:

Компоненти інфраструктури виходять з ладу, оскільки код програми (через поганий дизайн або впровадження) висуває надмірні вимоги до ресурсів[8].

Компоненти програми можуть вийти з ладу, оскільки ресурси, які їм потрібні, не завжди доступні (вчасно) [8].

Моделюючи типові та незвичайні виробничі навантаження протягом тривалого періоду, тестувальники можуть виявити недоліки в дизайні або реалізації системи. Коли ці недоліки будуть усунені, ті самі тести продемонструють, що система є стійкою. Контролери якості можуть скористатися перевагами інструментів навантажувального тестування для виконання багатьох процесів, визначених нижче[8].

У всіх сервісах зазвичай існує низка критичних процесів керування, які необхідно виконати, щоб забезпечити безперебійну роботу сервісу. Можливо, можна вимкнути сервісу для планового обслуговування в неробочий час, але більшість онлайн-сервісів працюють 24 години на добу[8].

Робочий день сервісу не закінчується ніколи. Неминуче процедури керування повинні виконуватися, поки сервіс активний, а користувачі знаходяться в системі. Ці процедури потрібно тестувати під час навантаження на систему, щоб переконатися, що вони не впливають негативно на живий сервіс, тобто тестування продуктивності[8].

1.4.3 Тести користувацького інтерфейсу (UI Tests)

Більшість додатків мають певний інтерфейс користувача. Зазвичай говорять про веб-інтерфейс у контексті веб-додатків. Варто не забувати, що REST API або інтерфейс командного рядка є так само інтерфейсом користувача, як і вишуканий веб-інтерфейс[9].

UI тести перевіряють правильність роботи інтерфейсу додатка. Введення користувача повинно викликати потрібні дії, дані повинні відображатися користувачу, стан інтерфейсу повинен змінюватися, як очікувалося[9].

Для традиційних веб-додатків тестування інтерфейсу можливе за допомогою інструментів, таких як Selenium[9].

З веб-інтерфейсами перевіряться декілька аспектів навколо Інтерфейсу користувача: поведінку, оформлення, зручність використання або відповідність корпоративному дизайну та інші [9].

Перевірка функціонування інтерфейсу користувача є достатньо простою. Для цього виконуються клікінг та введення даних у відповідних полях, очікуючи зміни стану інтерфейсу відповідно до введених дій. Сучасні фреймворки для односторінкових додатків, такі як React, Vue.js, Angular та інші, часто мають вбудовані інструменти і допоміжні засоби для детального тестування цих взаємодій на рівні модульних перевірок. Навіть у випадку, коли розробник створює власний фронтенд за допомогою чистого JavaScript, можна використовувати звичайні інструменти тестування, такі як Jasmine або Mocha. У традиційних серверних додатках використання тестів на основі Selenium є оптимальним варіантом[9].

Перевірка того, що макет веб-додатка залишається цілим, трохи складніша. Залежно від додатка та потреб користувачів, необхідно переконатися, що зміни коду не руйнують макет веб-сайту випадково[9].

Як інструменти, які допомагають автоматизувати перевірку дизайну веб-додатка під час збірки, часто використовують Selenium для відкриття додатка у різних браузерах, створення знімків екрану та порівняння їх зі зразками. Якщо виявляються різниці, інструмент попереджає про це[9].

Однак для оцінки зручності використання та "вигляду" продукту потрібні більш детальні методи. Тут корисно використовувати дослідницькі тести, оцінку зручності і демонстрації користувачам для з'ясування їх задоволеності використанням продукту[9].

UI-тестування має свої переваги через орієнтацію на візуальну частину програми та взаємодію з користувачем:

Повне покриття функціональності: UI-тести перевіряють, чи працює програма так, як очікує користувач, тобто перевіряють весь функціонал програми зовнішнього інтерфейсу.

Візуальна перевірка. UI-тести дозволяють перевірити, чи відповідає веб-сторінка або додаток заданим дизайну та вимогам до візуальної частини.

Користувацький досвід. UI-тести перевіряють зручність користування програмою, реакцію на дії користувача та спілкування з системою.

Крос-платформність. Правильно написані UI-тести можуть використовуватись на різних платформах, що дозволяє перевірити роботу програми на різних пристроях та браузерах.

Запобігання помилок. Виявлення проблем на ранніх етапах розробки дозволяє уникнути серйозних помилок та забезпечує високу якість продукту.

Однак, важливо розуміти, що UI-тести не завжди можуть покрити всі аспекти програми. Вони можуть бути більш крихкими та залежними від змін у веб-сторінці або додатку. Комбінація UI-тестів з іншими типами тестів, такими як unit-тести та integration-тести, дозволить забезпечити більш повне тестування всіх компонентів програми.

1.5. Висновки до першого розділу

Розглянувши основи тестування, його класифікацію, різноманітні типи тестів та методології їх застосування, було прийнято рішення щодо розробки інформаційної системи для тестування веб-додатків. Це рішення призначене покращити якість та надійність програмного продукту, забезпечивши його більш вичерпним, систематичним та ефективним тестуванням. Нова інформаційна система створюється для автоматизації процесів тестування веб-додатків, що дозволить забезпечити більш швидке виявлення помилок та забезпечить високу якість програмного забезпечення.

РОЗДІЛ 2

ПРОЕКТНІ РІШЕННЯ

2.1. Інструмент керування проектом

Для керування проектом було обрано Apache Maven.

Apache Maven — це інструмент управління та розуміння програмного проекту. Базуючись на концепції об’єктної моделі проекту (POM), Maven може керувати збіркою проекту, звітністю та документацією з центральної частини інформації[10].

Основна мета Maven — дозволити розробнику зрозуміти повний стан розробки за найкоротший період часу. Щоб досягти цієї мети, Maven має справу з кількома проблемними сферами[10]:

1. Полегшення процесу створення[10]
2. Забезпечення єдиної системи побудови[10]
3. Надання якісної інформації про проект[10]
4. Заохочення кращих практик розвитку[10]
5. Полегшення процесу створення[10]

Хоча використання Maven не виключає необхідність знань про базові механізми, він вберігає розробників від багатьох деталей[10].

Надання однорідної системи збірки:

Maven будує проект, використовуючи свою об'єктну модель проекту (POM) та набір плагінів. Після ознайомлення з одним проектом Maven, ви розумієте, як будуються всі Maven-проекти. Це економить час при роботі з багатьма проектами[10].

Надання якісної інформації про проект:

Maven надає корисну інформацію про проект, яка частково береться з вашого POM та частково генерується з джерел проекту. Наприклад, Maven може надати[10]:

1. Журнал змін, створений безпосередньо з джерела керування версіями [10]
2. Перехресні посилання на джерела[10]
3. Списки розсилки, керовані проектом[10]
4. Залежності, використовувані проектом[10]
5. Звіти про модульне тестування, включаючи покриття[10]
6. Також плагіни Maven сторонніх продуктів аналізу коду додають свої звіти до стандартної інформації, наданої Maven[10].
7. Надання рекомендацій для розвитку кращих практик[10]
8. Maven має на меті зібрати поточні принципи кращих практик розробки та спростити керівництво проектом у цьому напрямку[10].

Наприклад, специфікація, виконання та звітування модульних тестів є частиною звичайного циклу збирання Maven. Поточні кращі практики модульного тестування використовуються як рекомендації[10]:

- Зберігання вихідного коду тестів у окремому, але паралельному дереві вихідних кодів [10]
- Використання умов іменування тестових випадків для пошуку та виконання тестів[10]
- Установка тестових випадків для підготовки свого середовища замість налаштування збірки[10]
- Maven також допомагає в робочому процесі проекту, наприклад, у випусках та управлінні проблемами [10]

Також Maven надає деякі вказівки щодо структури каталогів проекту. Оволодівши цією структурою, можна орієнтуватися в інших проектах, що використовують Maven[10].

Хоча Maven має вибірковий підхід до структури проекту, деякі проекти можуть не відповідати цій структурі з історичних причин. Незважаючи на те, що Maven розроблений для гнучкості у використанні для потреб різних проектів, він не може задовольнити кожну ситуацію, не посягаючи на свої цілі[10].

Якщо проект має незвичну структуру збірки, яку неможливо реорганізувати, можливо, доведеться відмовитися від деяких можливостей або взагалі не використовувати Maven[10].

Основні особливості Maven у кількох словах:

- Просте налаштування проекту, яке відповідає кращим практикам - розпочніть новий проект або модуль за кілька секунд[10].
- Однакове використання в усіх проектах - це означає відсутність часу на адаптацію для нових розробників, які приходять на проект[10].
- Високоякісне керування залежностями, включаючи автоматичне оновлення, закриття залежностей (також відомі як транзитивні залежності) [10].
- Здатність легко працювати з кількома проектами одночасно[10].
- Великий і зростаючий репозиторій бібліотек і метаданих для використання вже готових рішень, а також узгодження з найбільшими проектами з відкритим вихідним кодом для доступності їх останніх релізів в реальному часі[10].
- Розширюваність, з можливістю легко написати плагіни на Java або скриптових мовах[10].
- Миттєвий доступ до нових можливостей з мінімальним або без додаткової конфігурації[10].
- Засновані на моделі збірки: Maven може зібрати будь-яку кількість проектів у попередньо визначені типи виходу, такі як JAR, WAR або розподіл на основі метаданих проекту без потреби у скриптах у більшості випадків[10].
- Згуртована сторінка інформації про проект: З використанням тих самих метаданих, що й для процесу збірки, Maven може створювати веб-сайт або PDF із будь-якою додатковою документацією та стандартними звітами про стан розвитку проекту[10].
- Управління випусками та публікація розподілу: Без значної додаткової конфігурації Maven інтегрується з вашою системою керування

версіями (наприклад, Subversion або Git) та керує випуском проекту на основі певного тегу. Також він може публікувати це у розподільному місці для використання іншими проектами[10].

- Керування залежностями: Maven сприяє використанню центрального репозиторію JAR-файлів та інших залежностей. Maven постачає механізм, який клієнти вашого проекту можуть використовувати для завантаження будь-яких JAR, необхідних для збірки вашого проекту, з центрального репозиторію JAR, подібно до CPAN в Perl. це дозволяє користувачам Maven використовувати JAR-файли у різних проектах та сприяє спілкуванню між проектами для забезпечення роботи сумісності версій[10].

Дуже корисним інструментом є плагіни. Плагіни - це набір функціональних модулів, які можна включати до процесу збірки та різноманітних фаз життєвого циклу проекту в Maven. Вони розширюють базовий функціонал Maven, дозволяючи робити такі дії, як генерація звітів, робота зі залежностями, компіляція, тестування, розгортання тощо. Основні переваги використання плагінів Maven включають наступне:

- Розширення функціональності. Плагіни дозволяють вам виконувати різноманітні завдання під час збирання, тестування та розгортання проекту.
- Конфігурування фаз життєвого циклу. Плагіни можуть бути налаштовані для виконання на певних етапах життєвого циклу проекту.
- Реюзабельність та розширюваність. Maven має широкий спектр вбудованих плагінів, а також можливість створювати власні плагіни або використовувати сторонні для вирішення конкретних завдань.

Наприклад, ось кілька популярних плагінів Maven:

1. Maven Compiler Plugin: Компіляція програмного коду проекту.
2. Maven Surefire Plugin: Запуск тестів у проекті.
3. Maven Clean Plugin: Видалення зібраного вихідного коду або файлів проекту.

4. Maven Install Plugin: Встановлення пакетів або артефактів в локальний репозиторій Maven.

5. Maven Deploy Plugin: Розгортання артефактів у віддалений репозиторій.

6. Maven Site Plugin: Генерація документації проекту

2.1. Фреймоврк тестування

TestNG — це фреймворк тестування, розроблена для спрощення широкого діапазону потреб у тестуванні, від модульного тестування (тестування класу ізольовано від інших) до інтеграційного тестування (тестування цілих систем, що складаються з кількох класів, кількох пакетів і навіть кількох зовнішніх фреймворків, таких як сервери додатків)[11].

Написання тесту зазвичай складається з трьох етапів:

1. Написання бізнес-логіки свого тесту та вставка анотацій TestNG у свій код[11].
2. Додавання інформації про свій тест (наприклад, назва класу, групи, які ви хочете запустити тощо) у файл testng.xml або build.xml[11].
3. Запуск TestNG[11].

Цей фреймворк використовує анотації - спеціальні мітки у коді, що дозволяють розробникам вказувати конфігурацію тестів, установлювати порядок виконання, групувати тести, робити параметризовані тести та використовувати багато інших функціональних можливостей[11].

Основні можливості TestNG включають:

Використання анотацій для визначення різних аспектів тестів, таких як @Test, @BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest тощо.

Групування тестів: Можливість групувати тести за різними критеріями, щоб вони виконувалися в окремих контекстах.

Параметризація тестів: Здатність виконувати однакові тести з різними вхідними параметрами.

Підтримка залежностей між тестами: Встановлення залежностей між тестами, де один тест може виконуватися тільки після успішного завершення іншого.

Паралельне виконання: Запуск тестів паралельно для збільшення швидкості виконання.

Генерація звітів: Автоматична генерація детальних звітів після виконання тестів, що дозволяє зручно аналізувати результати.

Головним інструментом TestNG є анотації. Опис анотацій зазначено в таблиці 2.1.

Таблиця 2.1

Анотації для впорядкування запуску тестів

Анотація	**Опис**
@BeforeSuite	Анотований метод буде запущено до виконання всіх тестів у цьому пакеті.
@AfterSuite	Анотований метод буде запущено після виконання всіх тестів у цьому пакеті.
@BeforeTest	Анотований метод буде запущено до запуску будь-якого тестового методу, що належить до класів усередині тегу <test>.
@AfterTest	Анотований метод буде запущено після виконання всіх тестових методів, що належать до класів усередині тегу <test>.
@BeforeGroups	Список груп, перед якими буде запущено цей метод налаштування. Цей метод гарантовано запуститься незадовго до виклику першого тестового методу, який належить до будь-якої з цих груп.

Продовження таблиці 2.1

@AfterGroups	Список груп, після яких запускатиметься цей метод налаштування. Цей метод гарантовано запуститься незабаром після виклику останнього тестового методу, який належить до будь-якої з цих груп.
@BeforeClass	Анотований метод буде запущено перед викликом першого тестового методу в поточному класі.
@AfterClass	Анотований метод буде запущено після виконання всіх тестових методів у поточному класі.
@BeforeMethod	Анотований метод запускатиметься перед кожним тестовим методом.
@AfterMethod	Анотований метод запускатиметься після кожного тестового методу.Поведінка анотацій у суперкласі класу TestNG

Наведені вище анотації також будуть враховані (успадковані) при розміщенні в суперкласі класу TestNG. Це корисно, наприклад, для централізованого налаштування тестування для кількох тестових класів у спільному суперкласі[11].

У цьому випадку TestNG гарантує, що методи «@Before» виконуються в порядку успадкування (спочатку найвищий суперклас, потім ланцюжок успадкування), а методи «@After» — у зворотному порядку (вгору ланцюжком успадкування)[11].

Також для анотацій упорядкування існують й свої параметри, які зазначено в таблиці 2.2[11].

Таблиця 2.2

Параметри анотацій

Параметр	**Опис**
alwaysRun	Виконує метод незалежно від груп, до яких він належить, перед викликом тестів або після їх виконання.
dependentOnGroups	Список груп, від яких залежить цей метод.
dependentOnMethods	Список методів, від яких залежить цей метод.
enabled	Позначає, чи є методи в цьому класі/методі активними.
groups	Список груп, до яких належить цей клас/метод.
inheritGroups	Якщо true, цей метод буде включатися до груп, зазначених у анотації @Test на рівні класу.
onlyForGroups	Тільки для анотацій @BeforeMethod і @AfterMethod: Метод буде викликано лише якщо відповідний метод тестування належить до однієї з перелічених груп.

Необхідно також виділити анотації, які позначають метод як джерело даних для тестового методу, вони зазначені в табл. 2.3 [11].

Таблиця 2.3

Анотації, які позначають метод як джерело даних

Анотація	**Опис**
@DataProvider	Позначає метод як постачальника даних для тестового методу. Анотований метод повинен повертати Object[][], де кожен Object[] може бути призначений списку параметрів тестового методу. Тестовий метод @Test, який хоче отримати дані від цього DataProvider, повинен використовувати ім'я dataProvider, рівне імені цієї анотації.
name	Назва цього постачальника даних. Якщо не вказано, ім'я цього постачальника даних автоматично встановлюється на ім'я методу.
parallel	Якщо встановлено значення true, тести, створені за допомогою цього постачальника даних, виконуються паралельно. Значення за замовчуванням - false.
@Factory	Позначає метод як фабрику, яка повертає об'єкти, які будуть використані TestNG як тестові класи. Метод повинен повертати Object[].
@Listeners	Визначає слухачів у тестовому класі. value - масив класів, які розширюють org.testng.ITestNGListener.
@Parameters	Описує, як передавати параметри до методу @Test. value - список змінних, які використовуються для заповнення параметрів цього методу.

Ну а також параметри для анатоції @Test яка позначає метод тесту, вони зазачені в таблиці 2.4[11].

Таблиця 2.4

Параметри для анотації @Test

Анотація	**Опис**
alwaysRun	Якщо встановлено значення true, цей тестовий метод завжди буде запускатись, навіть якщо він залежить від методу, що завершився неуспішно.
dataProvider	Назва постачальника даних для цього тестового методу.
dataProviderClass	Клас, де потрібно шукати постачальника даних. Якщо не вказано, постачальник даних буде шукатись у класі поточного тестового методу або в одного з його базових класів. Якщо цей атрибут вказано, метод постачальника даних має бути статичним у вказаному класі.
dependsOnGroups	Список груп, від яких залежить цей метод.
dependsOnMethods	Список методів, від яких залежить цей метод.
description	Опис цього методу.
enabled	Чи активовані методи в цьому класі/методі.
expectedExceptions	Список винятків, які тестовий метод має викинути. Якщо не виняток або викинуто інший виняток, тест буде відзначено як неуспішний.
groups	Список груп, до яких належить цей клас/метод.
invocationCount	Кількість разів, які цей метод повинен бути викликаним.

Продовження таблиці 2.4

Анотація	**Опис**
invocationTimeOut	Максимальний час у мілісекундах, який повинен займати тест для кумулятивного часу всіх invocationCount. Цей атрибут буде ігнорований, якщо не вказано invocationCount.
priority	Пріоритет для цього тестового методу. Методи з меншим пріоритетом будуть запускатись першими.
successPercentage	Відсоток успішності, який очікується від цього методу.
singleThreaded	Якщо встановлено значення true, всі методи в цьому тестовому класі гарантовано будуть виконуватись в одному потоці, навіть якщо тести виконуються з параметром parallel="methods". Цей атрибут можна використовувати тільки на рівні класу і буде ігнорований, якщо використовується на рівні методу. Примітка: цей атрибут раніше називався sequential (тепер застаріло).
timeOut	Максимальний час у мілісекундах, який цей тестовий метод повинен займати.
threadPoolSize	Розмір пула потоків для цього методу. Метод буде викликатися з кількох потоків, як вказано параметром invocationCount. Примітка: цей атрибут ігнорується, якщо не вказано invocationCount.

2.2 Інструмент для автоматизованого тестування веб-проектів

Selenium WebDriver – це гнучкий інструмент для автоматизованого тестування веб-проектів на базі набору бібліотек для різних мов програмування, таких як Java, .Net (C#), Python, Ruby, PHP, Perl, JavaScript[12].

Даний інструмент підтримує роботу на базі Windows, macOS та Linux, а також найпоширеніші браузери Google Chrome, Firefox, Safari, Edge, Internet Explorer та навіть деякі браузери без графічного інтерфейсу. Використовується Selenium WebDriver найчастіше з такими видами тестування, як регресійне та функціональне[12].

Для того, щоб зрозуміти логіку роботи Selenium WebDriver потрібно розібратися з основними компонентами для взаємодії[12].

Архітектура Selenium WebDriver складається з 4-х компонентів:

- Драйвер конкретного браузера;
- Клієнтська бібліотека Selenium;
- Браузер;
- Протокол JSON Wire (JavaScript Object Notation).

Графічне зображення представлено на рис. 2.1.

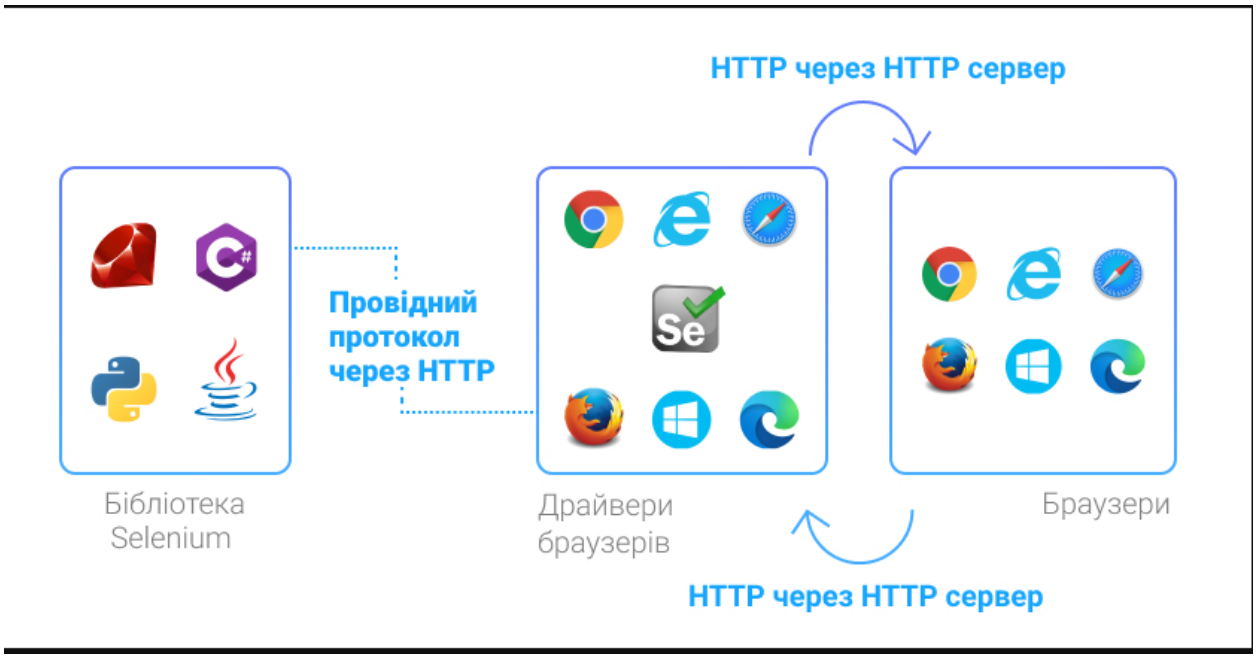

Рисунок 2.1 – Графічне зображенні архітектури Selenium WebDriver

Щоб не повторювати одноманітних операцій в браузері при тестуванні, використовується webdriver браузера, який звертається до браузера, драйвер якого використано, та виконує послідовність дій через певний скрипт з бібліотеки Selenium. Прикладами дій можуть бути команди по знаходженню елементів, перехід за посиланнями, збір великих об'ємів даних (парсинг), натискання кнопок та ін. Взаємодія бібліотеки з веб-драйвером відбувається через JSON Wire Protocol[12].

Основні сутності та методи Selenium WebDriver:

WebDriver – це найважливіша сутність, яка відповідає за управління діями браузера[12].

Webelement – друга важлива сутність, яка представляє собою абстракцію над конкретним веб-елементом (посиланням, кнопкою, полем для вводу та ін.)[12].

Локатор – це тип веб-елементу, який потрібно знайти та над яким виконуватиме дії веб-драйвер[12].

By – це абстракція над локатором веб-елемента, клас необхідний для ідентифікації веб-елементів який має такий синтаксис By.локатор[12].

Нижче наведено список найбільш поширених типів локаторів та їх синтаксис[12]:

- 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 за атрибутом 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 елементу або групи елементів (якщо, це клас) буде застосовано стиль;
- By.XPath – пошук елемента по атрибуту XML path.

Переваги та недоліки Selenium WebDriver

Даний інструмент є потужним та досить популярним серед тестувальників, проте, як будь-яке інше програмне забезпечення, він має ряд переваг та недоліків[12].

Переваги:

- Інтеграція з великою кількістю мов програмування та кросплатформність;
- Простий набір команд та легкість створення скриптів за допомогою бібліотек;
- Виконання авто-тестів можна здійснювати без участі людини та у будь-який час;
- Безкоштовний продукт з відкритим вихідним кодом.

Недоліки:

- Необхідність володіння навичками програмування;
- Обмеженість функціоналу в порівнянні з платними аналогами;
- Не може бути використаний для тестування графічних елементів та Flash-об’єктів;
- Наявність дефектів у самих бібліотеках.

2.3 Шаблон проектування

Модель сторінки (Page Object) - це шаблон проектування, який став популярним у тестуванні автоматизації для покращення підтримки тестів та зменшення дублювання коду. Модель сторінки представляє собою об'єктно-орієнтований клас, який служить інтерфейсом до сторінки вашого веб-додатку. Тести використовують методи цього класу моделі сторінки, коли

потрібно взаємодіяти з його інтерфейсом. Основна перевага полягає в тому, що якщо інтерфейс змінюється для сторінки, то тестам не потрібно змінюватися, а лише потрібно змінювати код в класі моделі сторінки. Всі зміни для підтримки нового інтерфейсу знаходяться в одному місці[13].

Переваги:

- Чистий розподіл між кодом тестів та кодом, специфічним для сторінки, таким як локатори або їх використання, якщо ви використовуєте UI Map, а також виведення[13].
- Один репозиторій для послуг або операцій, які пропонує сторінка, замість того, щоб мати ці послуги розпорошеними по всіх тестах[13].
- Завдяки цьому будь-які модифікації, необхідні через зміни в інтерфейсі, можуть бути внесені в одному місці[13].

PageObjects виконують функцію обличчя, що орієнтоване у дві сторони. З одного боку, вони відображають послуги, що пропонує певна сторінка, інтерфейс для розробника тестів. З іншого боку, PageObjects єдині, хто має глибоке розуміння структури HTML сторінки (або частини сторінки). Методи Page Object можна сприймати як "сервіси", які сторінка надає, замість розкриття деталей та механіки самої сторінки[13].

Як приклад, уявіть скриньку веб-пошти будь-якої системи електронної пошти. Серед послуг, які вона пропонує, є можливість створення нового листа, вибір читання одного листа та перелік тем листів у скриньці. Як саме ці функції реалізовані - не повинно впливати на тест[13].

Розробники тестів міркують про сервіси, з якими вони взаємодіють, а не про конкретну реалізацію. Тому PageObjects мало коли повинні розкривати основний екземпляр WebDriver. Для цього методи PageObject повертають інші PageObjects. Це дозволяє ефективно моделювати подорож користувача через додаток. Такий підхід також означає, що зміна взаємозв'язку сторінок призведе до невдалих спроб компіляції тестів, просто змінивши відповідний метод. Іншими словами, можна передбачити, які тести відмовлять без їх

запуску, коли змінюємо зв'язок між сторінками, і відобразити це у PageObjects[13].

2.4 Предмет тестування

Предметом тестування виступає веб додаток з організації навчального процесу. Він містить наступні сторінки:

- Реєстрація
- Секретері
- Курси
- Заняття
- Учні
- Користувачі без ролей
- Входу в систему
- Ментори

В залежності від ролі користувача він має окремі права на редагування різних сутностей. Права менторів можна побачити на рисунку 2.2.

Кожна з ролей можна редагувати, це в свою чергу створює нові вимоги до редагування полів даних.

Кожне поле має свої вимоги повідомлення, якщо якесь с полів заповнено без дотримання вимог.

Це добре можна побачити на вимогах для сторінці реєстрації, вона має вимоги в таблиці 2.5-2.8.

Description	Admin	Secretary	Mentor	Student
Assign account to mentor	✓	✓	✗	✗
Get filter list of lessons for mentor	✗	✗	✓	✗
Get only active mentors	✓	✓	✓	✗
Get mentor information by mentor ID	✓	✓	✓	✗
Disabling of mentor account	✓	✓	✗	✗
Gets list of all mentors	✓	✓	✗	✗
Update of mentor	✓	✓	✗	✗
Gets all of the mentor's study group	✓	✓	✓	✗
Gets all of the mentor's courses	✓	✓	✓	✗
Returns list of lessons for mentor	✓	✓	✓	✗

Рисунок 2.2. Права користувачів менторів.

Таблиця 2.5

Технічні вимоги для полів Ім'я/Прізвище

Поле "Ім'я/Прізвище"	Опис помилок
Поле порожнє, і користувач натиснув кнопку "Зареєструватися" або після першої події onBlur	Поле виділене червоним контуром, і під полем відображено червоний текст помилки "Це поле обов'язкове"
Поле містить одну літеру	Під полем відображено червоний текст помилки "Занадто коротке"
Поле містить більше п'ятдесяти літер	Під полем відображено червоний текст помилки "Занадто довге"
Поле починається або закінчується пробілом або дефісом	Під полем відображено червоний текст помилки "Неправильне значення"
Поле містить принаймні один символ або цифру	Під полем відображено червоний текст помилки "Неправильне значення"

Таблиця 2.6

Технічні вимоги для поля "Електронна пошта"

Поле "Електронна пошта"	Опис помилок
Поле "Електронна пошта" порожнє, і користувач натиснув кнопку "Зареєструватися" або після першої події onBlur	Поле виділене червоним контуром, і під полем відображено червоний текст помилки "Це поле обов'язкове"
Поле "Електронна пошта" заповнене неправильною адресою	Під полем відображено червоний текст помилки "Неправильна адреса електронної пошти"

Таблиця 2.7

Технічні вимоги для поля " Підтвердження пароля "

Поле "Підтвердження пароля"	Опис помилок
Поле "Підтвердження пароля" порожнє, і користувач натиснув кнопку "Зареєструватися" або після першої події onBlur	Поле виділене червоним контуром, і під полем відображено червоний текст помилки "Це поле обов'язкове"
Поле "Підтвердження пароля" відрізняється від поля "Пароль" вище	Під полем відображено червоний текст помилки "Ви повинні підтвердити свій пароль"

Таблиця 2.8

Технічні вимоги для поля " Пароль "

Поле "Пароль"	Опис помилок
Поле "Пароль" порожнє, і користувач натиснув кнопку "Зареєструватися" або після першої події onBlur	Поле виділене червоним контуром, і під полем відображено червоний текст помилки "Це поле обов'язкове"

Поле "Пароль"	Опис помилок
Довжина поля "Пароль" менше вісьми символів	Під полем відображено червоний текст помилки "Пароль повинен містити принаймні 8 символів"
Довжина поля "Пароль" більше шестнадцяти символів	Під полем відображено червоний текст помилки "Пароль повинен містити максимум 16 символів"
Поле "Пароль" містить неправильні дані	Під полем відображено червоний текст помилки "Пароль повинен містити принаймні одну велику літеру, одну малу літеру, одну цифру та один спеціальний символ (!@#$%^&*()_+-=)"

Такі вимоги є як для нових користувачів так и для користувачів які вже зареєстровані й вимагають редагування.

Також на сторонках доступний наступний функціонал:

- Сортування за ім’ям, прізвищем та поштою
- Пошук по трем критеріям
- Редагування користувачів
- Додавання нових користувачів

2.5 Висновки до другого розділу

Розглядаючи інструменти та технології тестування, вибір Apache Maven для керування проектом є важливим кроком у забезпеченні систематичності та ефективності у процесі розробки. TestNG, як фреймворк тестування, разом із Selenium як інструментом для автоматизації тестування веб-проектів, дозволять створити надійну інфраструктуру для тестування.

Шаблон проектування Page Object робить тестування більш модульним та зрозумілим, спрощуючи взаємодію з елементами веб-сторінок. Обравши веб-додаток для організації навчального процесу як предмет тестування, вибрали широкий та значущий простір для застосування цих технологій, що сприятиме покращенню якості та надійності програмного забезпечення.

РОЗДІЛ 3

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

3.1 Структура інформаційної системи

Загальна структура інформаційної системи подана на рис. 3.1

Рисунок 3.1 – Структура інформаційної системи

Тестова стратегія: Це визначення загального плану дій для тестування веб-додатку. Вона включає в себе обрані методи, види тестів, обсяг і порядок їх виконання.

Тестова документація: Це набір документів, які містять вимоги, плани, сценарії тестування, звіти.

Тестові скрипти та сценарії: Це конкретні інструкції, що визначають послідовність дій для виконання тестів. Вони можуть охоплювати сценарії взаємодії з веб-додатком, випадки використання.

Інструменти тестування: Вони включають у себе автоматизовані та ручні засоби для виконання тестів, перевірки функціональності, безпеки, продуктивності та інших аспектів веб-додатку.

Управління дефектами: Система для відстеження, документування та вирішення виявлених проблем у веб-додатку.

Звіти та аналітика: Інструменти для аналізу результатів тестування, які дозволяють зрозуміти якість додатку, виявлені проблеми та їх рівень критичності.

Ці блоки співпрацюють між собою для забезпечення ефективного та комплексного тестування веб-додатків. Важливо мати чіткий план тестування, який охоплює всі аспекти функціональності, безпеки та продуктивності додатку.

3.2.Розробка архітектури

Для розробки тестів був обраний патерн Paje Object.

Базуючись на технічних вимогах були розроблені наступні класи базових елементів: BaseElement, Header, SideBar, Pagination, Page.

Структура класів подана на рис. 3.2.

Абстрактний клас BaseElement описую основні методи, які будуть використовуватися в усіх тестах:

- Конструктор BaseElement приймає об'єкт WebDriver, ініціалізує елементи сторінки за допомогою PageFactory та створює об'єкт SoftAssert для м'яких перевірок.
- Метод fillField заповнює поле введення текстом, якщо воно не є відключеним або тільки для читання.
- Метод getItemFromMenu приймає список елементів webElementList та шукає певний елемент за назвою.
- clickElement клікає на переданий веб-елемент.
- hardClear виконує "жорстке" видалення тексту з поля введення.

- Метод assertAll викликає assertAll з об'єкта SoftAssert, щоб зробити всі м'які перевірки і вивести результати.

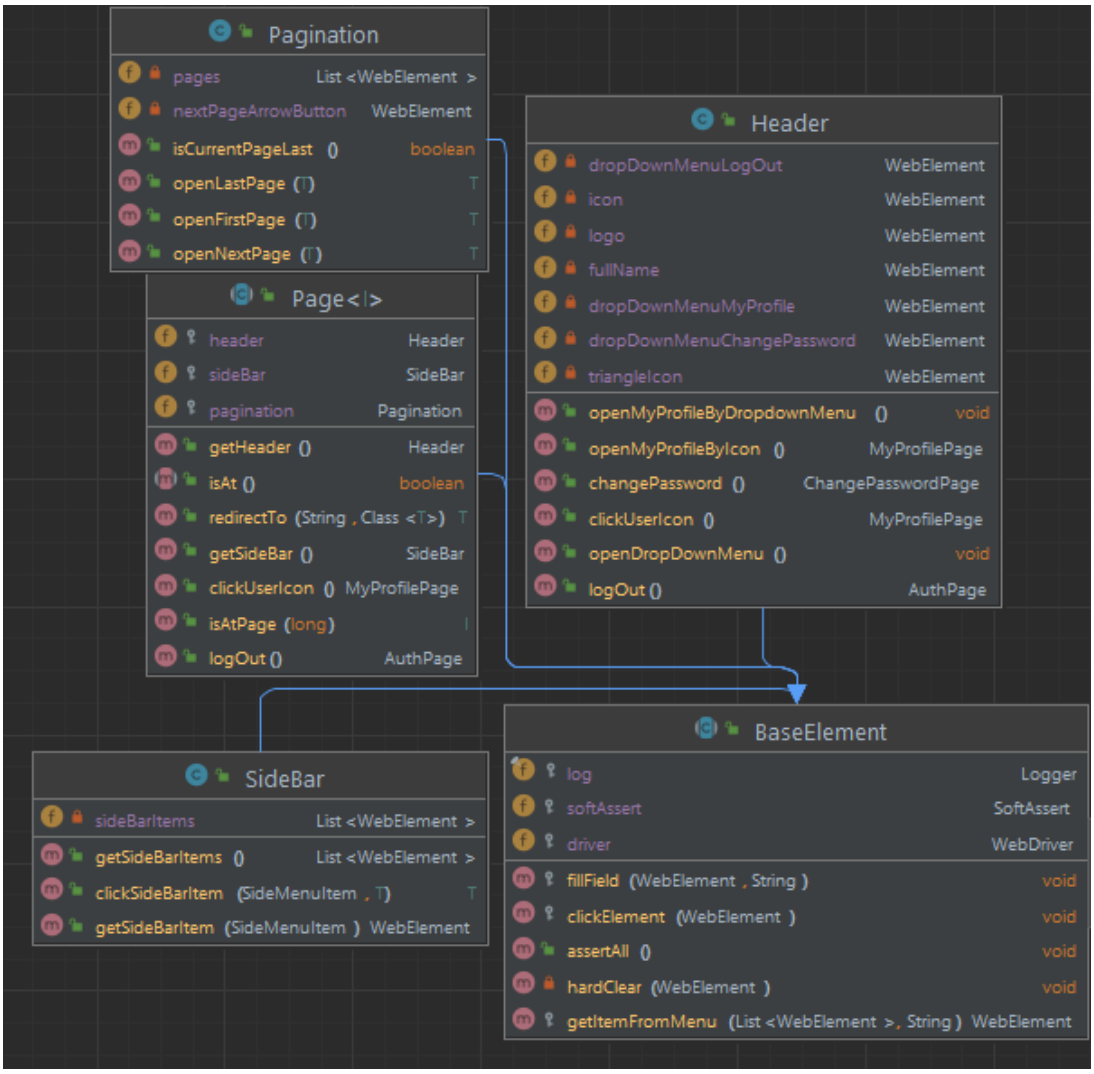

Рисунок 3.2 – Архітектура базових класів

Також необхідно придивитися до класу Page, так як він стане основою для всіх подальших класів для опису сторінок. Конструктор Page: Ініціалізує базові елементи сторінки, такі як header, sideBar та pagination.

- Метод redirectTo: Перенаправляє користувача за вказаною URL-адресою. Очікує, доки користувач не опиниться на цій сторінці.
- Метод getHeader: Повертає об'єкт Header, щомістить елементи верхнього колонтитулу сторінки.

- Метод clickUserIcon: Викликає дію натискання на іконку користувача, можливо, для переходу на сторінку особистого профілю (MyProfilePage).
- Метод getSideBar: Повертає об'єкт SideBar, що містить елементи бічної панелі.
- Метод isAt: Використовується для перевірки, чи користувач фактично перебуває на поточній сторінці.
- Метод isAtPage: Використовує очікування з awaitility, щоб перевірити, чи перебуває користувач на поточній сторінці протягом певного часу.
- Метод logOut: Виконує вихід з системивикликаючи функціонал виходу з об'єкта Header та повертає сторінку авторизації (AuthPage).

3.3 Пошук елементів

Для пошуку елементів використовувались XPath. Для їх пошуку використовувалась Selenium IDE, а також інструменти браузера. Користування XPath для локалізації елементів на сторінці, записаних та перевірених через Selenium IDE, а також за допомогою інструментів браузера, допомагає створювати надійні тести для автоматизованого тестування. Це дозволяє забезпечити стабільність тестових сценаріїв при змінах у веб-додатках.

Для пошуку елементів запускалось Selinum IDE , а потім вказувалась необхідна сторінка. На рисунку можна побачити як виглядаю процес запису дій у Selinum IDE на сторінці реєстрації, яка зображена на рис.3.3.

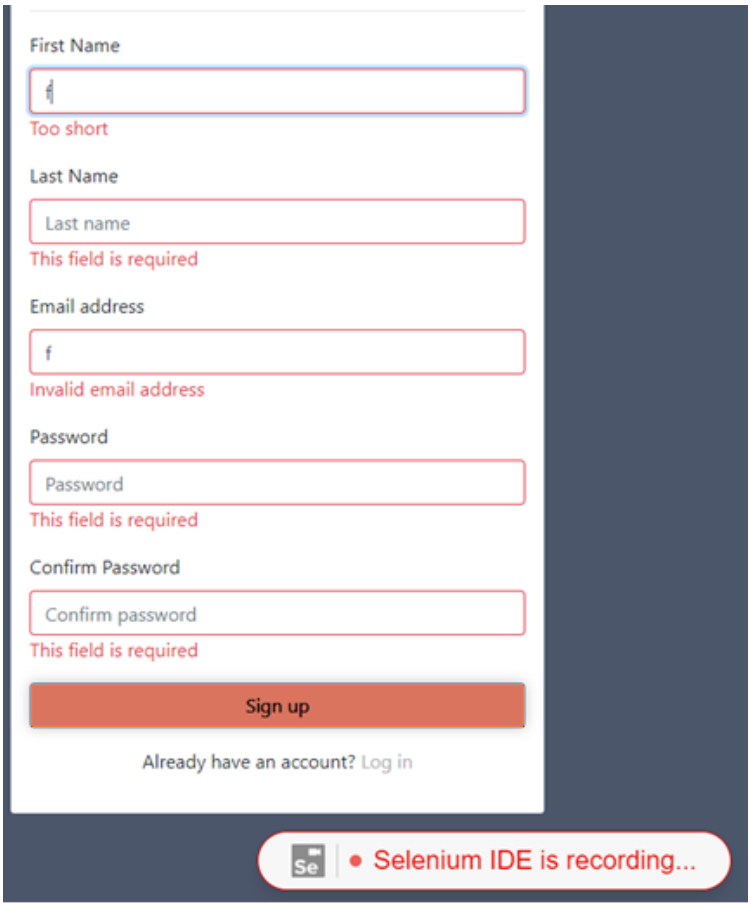

Рисунок 3.3 – Сторінка реєстрації

Цей інструментарій записує усі дії, а також дає шляхи необхідних елементів. В даному випадку це поля вводу та помилки, які можна побачити на рис. 3.4.

http://localhost:8080

	Command	Target	Value
1	open	/registration	
2	set window size	854x1010	
3	click	id=firstName	
4	type	id=firstName	f
5	mouse down	id=lastName	
6	mouse up	css=.form-group:nth-child(4) > label	
7	click	css=.form-group:nth-child(4)	
8	click	css=.form-group:nth-child(3) > .text-danger	
9	click	id=email	
10	click	id=firstName	
11	type	id=email	f
12	mouse down	id=password	
13	mouse up	css=.form-group:nth-child(6) > label	
14	click	css=.form-group:nth-child(6)	
15	mouse down	id=confirm-password	
16	mouse up	css=.form-group:nth-child(7) > label	
17	click	css=.form-group:nth-child(7)	
18	click	css=.form-group:nth-child(7)	

Рисунок 3.4 – Результат Selenium IDE

Також для пошуку використовувались інструменту розробника та навігації по істотному коду. Цей інструмент наявний майже в кожному браузері. Приклад такого підходу можна побачити на рис. 3.5.

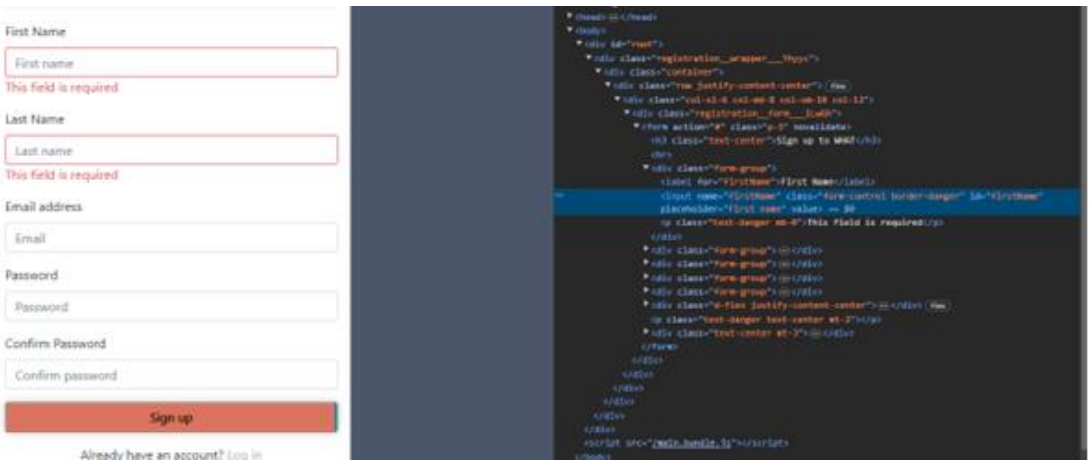

Рисунок 3.5 – Пошук елементів в браузері

Всі найдені елементи були розміщенні в класі Locators. Цей клас складається з набору інтерфейсів. Елементи цього класу відображені на рис.3.6.

Цей клас, Locators, є збірником локацій (адреси) різних елементів на веб-сторінках вашого додатка. Він містить статичні рядки, які представляють шляхи XPath до елементів на різних сторінках додатка.

Основна структура класу Locators розділена на інтерфейси для кожного типу сторінки або групи елементів. Наприклад, Auth, Lessons, Secretaries тощо. У межах кожного інтерфейсу зібрані різні локації (XPath, CSS селектори) для конкретних елементів, які використовуються на відповідних сторінках.

Цей підхід дозволяє зберігати всі локації в одному місці, що полегшує підтримку та редагування в майбутньому. Якщо змінити шлях до якогось елемента на сайті, можна оновити цей шлях лише в одному місці - відповідному інтерфейсі у класі Locators, щоб внести зміни у всьому додатку, де цей шлях використовується.

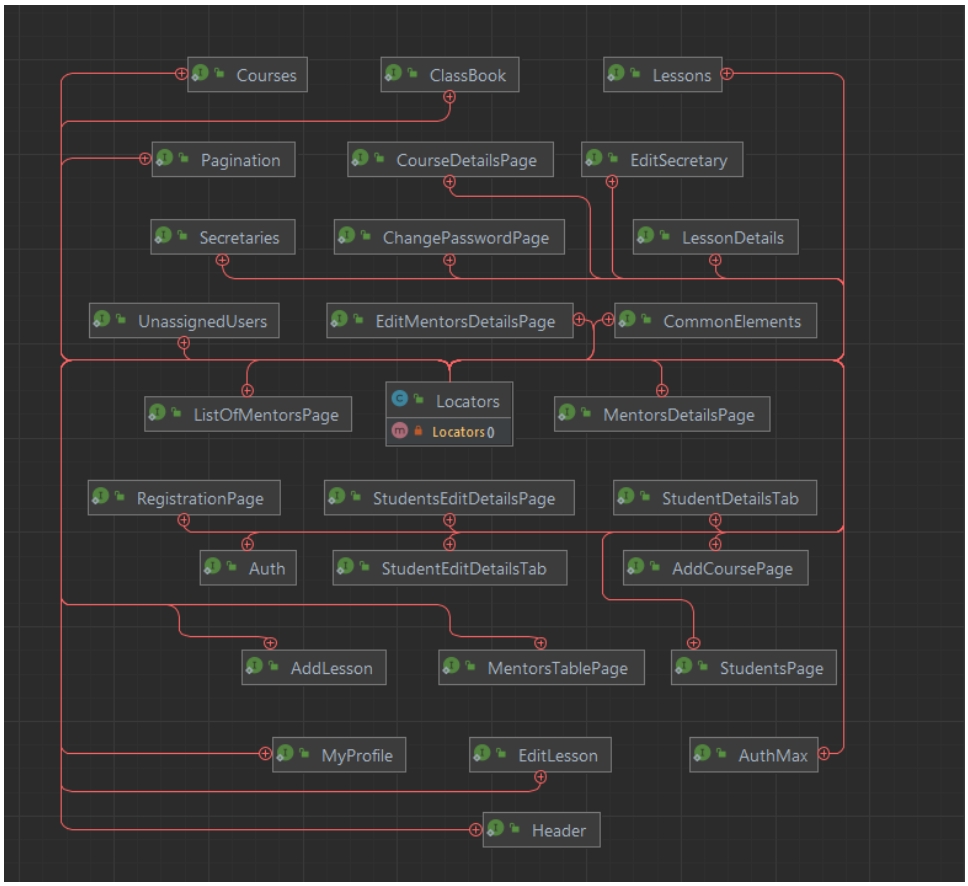

Рисунок 3.6 –Структура класу Locators

Наприклад, в Lessons маємо локації для різних елементів, таких як кнопки "Додати урок", іконки редагування уроків, таблиця з уроками та інші.

Такий підхід до організації локацій допомагає підтримувати чистий та структурований код для подальшого використання у тестах чи скриптах. Детальніше побудову Інтерфейсів можна подивитися на основі RegistrationPage.

Він має наступний перелік локаторів:

- FIRST_NAME_INPUT_FIELD_XPATH = "//input[@name='firstName']"; : XPath для поля введення імені.
- LAST_NAME_INPUT_FIELD_XPATH = "//input[@name='lastName']"; : XPath для поля введення прізвища.
- EMAIL_INPUT_FIELD_XPATH = "//input[@name='email']"; : XPath для поля введення електронної пошти.

- PASSWORD_INPUT_FIELD_XPATH = "//input[@name='password']"; : XPath для поля введення пароля.
- CONFIRM_PASSWORD_INPUT_FIELD_XPATH = "//input[@name='confirmPassword']"; : XPath для поля підтвердження пароля.
- FIRST_NAME_ERROR_XPATH = "//input[@name='firstName']//following-sibling::p"; : XPath для відображення помилки, пов'язаної з полем введення імені.
- LAST_NAME_ERROR_XPATH = "//input[@name='lastName']//following-sibling::p"; : XPath для відображення помилки, пов'язаної з полем введення прізвища.
- EMAIL_ERROR_XPATH = "//input[@name='email']//following-sibling::p"; : XPath для відображення помилки, пов'язаної з полем введення електронної пошти.
- PASSWORD_ERROR_XPATH = "//input[@name='password']//following-sibling::p"; : XPath для відображення помилки, пов'язаної з полем введення пароля.
- CONFIRM_PASSWORD_ERROR_XPATH = "//input[@name='confirmPassword']//following-sibling::p"; : XPath для відображення помилки, пов'язаної з полем підтвердження пароля.
- SIGN_UP_BUTTON_XPATH = "//button[text()= 'Sign up']"; : XPath для кнопки реєстрації.
- LOG_IN_XPATH = "//a[@href='/auth']"; : XPath для посилання на сторінку авторизації.
- MODAL_WINDOW_XPATH = "//*[contains(@class,'modal-content')]"; : XPath для модального вікна на сторінці.
- MODAL_WINDOW_BUTTON_XPATH = "//button[text()='Back']"; : XPath для кнопки "Back" в модальному вікні.

3.4 Опис сторінок об’єктів

Як і було зазначено в розділі, для опису було вибрано патерн PageObject.

Це означає, що для кожної сторінки були розроблений окремий клас. Ці класи містять поля типу WebElement.

Ці поля містять наступну конструкцію:

```
@FindBy(xpath = MENTORS_DETAILS_TAB_XPATH)
protected WebElement mentorsDetailsTab;
```

За допомоги анотації @FindBy та константи MENTORS_DETAILS_TAB_XPATH знаходиться необхідні елементи на сторонці веб додатку.

Для кожного елементу були розроблено методи для взаємодії:

- Ведення даних
- Вибір необхідного поля
- Очищення даних

Прикладом цього функціоналу буде метод inputFirstName:

```
public EditMentorsDetailsPage inputFirstName(String name) {
        fillField(firstNameInputField, name);
        return this;    }
```

Цей метод є частиною класу EditMentorsDetailsPage і він призначений для введення значення в поле для імені ментора на сторінці редагування даних ментора.

Функція fillField викликається для заповнення введеного поля. Вона приймає два параметри: firstNameInputField, який вказує на поле для введення імені ментора на веб-сторінці і name, яке є значенням, яке ми хочемо ввести у це поле. Після введення значення метод повертає об'єкт EditMentorsDetailsPage, що дозволяє використовувати цей метод для подальших дій чи послідовних запитів.

Методи взаємодії повинні повертати об’єкти свого ж класу, так як це необхідність того, щоб тести могли бути написані через ланцюжок викликів.

Цей підхід дозволяє здійснювати ланцюжкові виклики методів, які використовуються для взаємодії з елементами на сторінці та збереження коду.

Було окремо відділено пошук строкових значень для подальшого порівняння.

Також було розроблені й ряд методів, які займаються пошуком строкових значень, які потім будуть використовуватися для порівняння. Одним з таких методів є getFirstName():

```
public String getFirstName() {
        return firstNameInputField.getAttribute("value");
    }
```

Для всіх полів були розроблені подібні методи. Структура класів для ролі Ментори подана на рис. 3.7, для користувачі без ролей на рис.3.8.

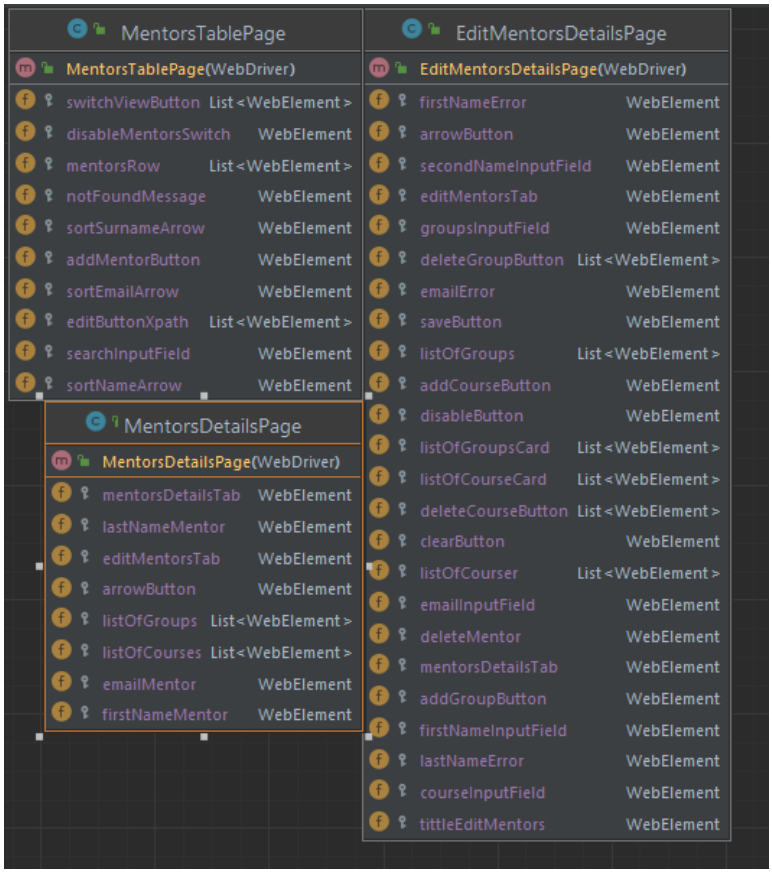

Рисунок 3.7 – Структура класів користувачі-менторів

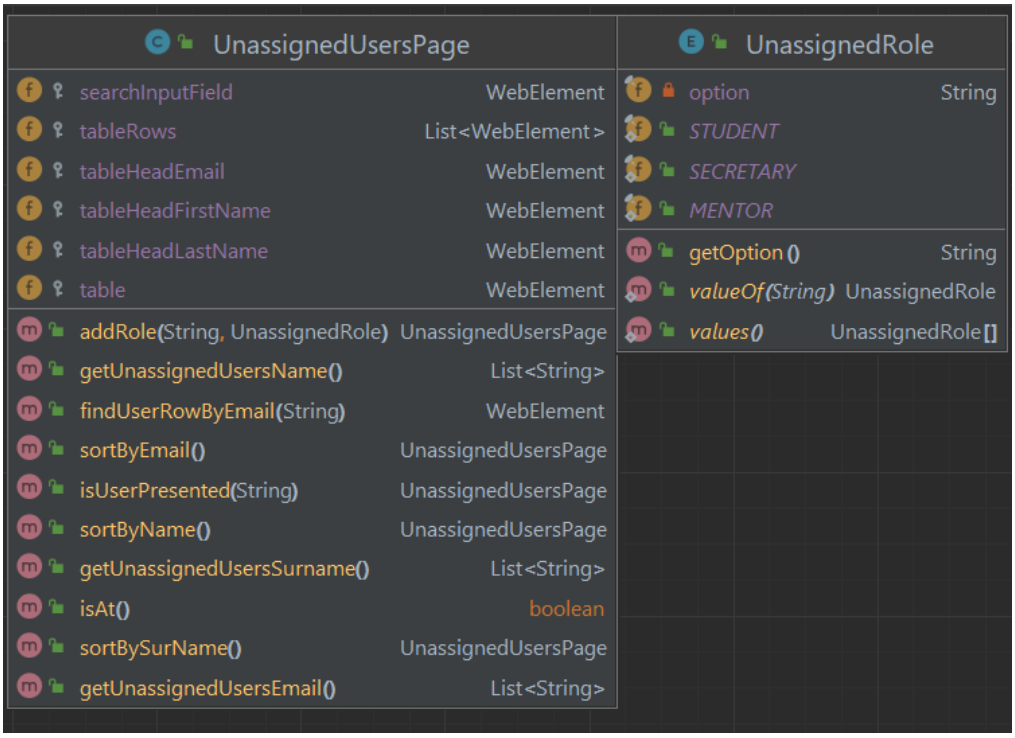

Рисунок 3.8– Структура класів користувачі без ролей

3.5 Тестування

Як основа всіх тестів було розроблено клас BaseTest, кожен тест буде его нащадком.

BaseTest має наступний код:

```
public class BaseTest {

    protected final Logger log = Logger.getLogger(getClass());
    protected WebDriver driver;
    protected long waitTime;

    @BeforeSuite
    protected void setup() {
        WebDriverManager.chromedriver().setup();
        log.info("Chromedriver is set up");
    }

    @BeforeClass
    protected void setupTest() {
        driver = new ChromeDriver();
```

```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));
                driver.manage().window().maximize();
                waitTime = 5;
                driver.get(Endpoints.BASE_URL);
                log.info("Initialise web driver");
            }
            @AfterClass
            protected void teardown() {
                driver.quit();
                log.info("Quit");
            }
        }
```

Цей код містить методи setup, setupTest і teardown, які викликаються перед тестами, перед кожним окремим тестом і після завершення тестування відповідно. Короткий опис кожного методу:

- setup: Викликається перед усіма тестами (в @BeforeSuite). Цей метод налаштовує ChromeDriver і встановлює логер.
- setupTest: Викликається перед кожним тестом (в @BeforeClass). Цей метод ініціалізує об'єкт WebDriver, налаштовує тайм-аути, максимізує вікно браузера, встановлює час очікування і переходить на основну адресу сайту.
- teardown: Викликається після закінчення усіх тестів (в @AfterClass). Цей метод закриває веб-драйвер.

Написані тести можна поділити на наступні категорії:

1. Редагування користувачів з коректними даними
2. Перевірка повідомлень помилок при некоректних даних
3. Тестування пошуку
4. Тестування сортування
5. Тестування надання ролі
6. Тестування спеціального функціоналу користувача

3.5.1 Редагування користувачів з коректними даними

До такого тесту можна віднести EditMentorDetailsPage_VerifyEditMentorDetails_CorrectData_AdminRole(), цей тест перевіряю редагування користувача з Ментор з коректними даними.

Цей тест має наступний код:

```
      public                                                    class
EditMentorDetailsPage_VerifyEditMentorDetails_CorrectData_AdminRole    extends
BaseTest {
          String newNameOfMentors;
          String newSurNameOfMentors;
          String newEmailOfMentors;
          UnassignedUser mentor;
          EditMentorsDetailsPage editMentor;
          String editMentorURL;
          public
EditMentorDetailsPage_VerifyEditMentorDetails_CorrectData_AdminRole() {
              mentor= UnassignedUser.getUnassignedUser();
              newNameOfMentors=mentor.getFirstName()+"new";
              newSurNameOfMentors=mentor.getLastName()+"new";
              newEmailOfMentors=mentor.getEmail()+"new";
          }
          @BeforeClass
          public void setUp() throws IOException {
              editMentor= AuthPage.init(driver)
                      .clickRegistrationLink()
                      .registerUser(mentor)
                      .logInAs(Role.ADMIN, StudentsPage.class)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.UNASSIGNED_USERS,
UnassignedUsersPage.class)
                      .isAtPage(waitTime)
                      .addRole(mentor.getEmail(), UnassignedRole.MENTOR)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.MENTORS, MentorsTablePage.class)
                      .inputSearchMentor(mentor.getFirstName()+"
"+mentor.getLastName())
                      .editMentors(0)
                      .isAtPage(waitTime);
              editMentorURL=driver.getCurrentUrl();
```

```
        }
        @Test(description = "126-22-1м")
        public void VerifyEditMentorDetails_CorrectData(){
            editMentor.inputFirstName(newNameOfMentors)
                    .inputSurname(newSurNameOfMentors)
                    .verifyInputSurName(newSurNameOfMentors)
                    .inputEmail(newEmailOfMentors)
                    .verifyInputEmail(newEmailOfMentors)
                    .saveMentor()
                    .isAtPage(waitTime)
                    .inputSearchMentor(newNameOfMentors+"
"+newSurNameOfMentors)
                    .verifyInputSearchField(newNameOfMentors+"
"+newSurNameOfMentors)
                    .editMentors(0)
                    .isAtPage(waitTime)
                    .verifyInputName(newNameOfMentors)
                    .verifyInputSurName(newSurNameOfMentors)
                    .verifyInputEmail(newEmailOfMentors)
                    .assertAll();
        }
    }
```

EditMentorDetailsPage_VerifyEditMentorDetails_CorrectData_AdminRole: Це конструктор класу, який встановлює нові дані для ментора, що будуть перевірятися під час тестування.

setUp: Цей метод викликається перед початком всіх тестів (в @BeforeClass). Він налаштовує тестовий середовище, реєструє нового користувача, входить в систему як адміністратор, додає роль ментора, переходить на сторінку редагування менторів і отримує URL цієї сторінки.

VerifyEditMentorDetails_CorrectData: Цей тест перевіряє функціонал редагування даних ментора. Він виконує декілька кроків:

1. Вводить нові дані ментора (ім'я, прізвище, електронну пошту).
2. Перевіряє, чи були введені правильні дані.
3. Зберігає зміни.
4. Перевіряє, чи збережені дані відображаються правильно в таблиці менторів.

5. Перевіряє правильність введених даних під час редагування.

Загалом, цей клас виконує тестування редагування даних менторів для адміністратора. Це перевіряє правильність функціоналу зміни даних ментора через веб-інтерфейс.

3.5.2 Перевірка повідомлень помилок при некоректних даних

До цієї категорії відноситься EditMentorsDetailsPage _VerifyEditMentors _ IncorrectData_ AdminRole з наступним кодом:

```
      public                                                                class
EditMentorsDetailsPage_VerifyEditMentors_IncorrectData_AdminRole       extends
BaseTest {
          InvalidData[] data;
          UnassignedUser mentor;
         EditMentorsDetailsPage editMentorsPage;

          public
EditMentorsDetailsPage_VerifyEditMentors_IncorrectData_AdminRole()      throws
IOException {
              mentor= UnassignedUser.getUnassignedUser();

data=InvalidData.getData(PathsToFiles.Mentors.EDIT_MENTOR_ERRORS);
          }

          @BeforeClass
          public void setUp() throws IOException {

              editMentorsPage = AuthPage.init(driver)
                      .clickRegistrationLink()
                      .registerUser(mentor)
                      .logInAs(Role.ADMIN, StudentsPage.class)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.UNASSIGNED_USERS,
UnassignedUsersPage.class)
                      .isAtPage(waitTime)
                      .addRole(mentor.getEmail(), UnassignedRole.MENTOR)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.MENTORS, MentorsTablePage.class)
```

```
                        .inputSearchMentor(mentor.getFirstName()   +   "   "   +
mentor.getLastName())
                        .editMentors(0)
                        .isAtPage(waitTime);
            }

            @DataProvider(name = "errors")
            public Object[][] provide() {
                Object[][] list = new Object[data.length][1];
                for (int i = 0; i < data.length; i++) {
                    list[i][0] = data[i];
                }
                return list;
            }

            @Test(description = "khara", dataProvider = "errors")
            public void verifyEditMentors_IncorrectData(InvalidData errors) {

                editMentorsPage
                        .isAtPage(waitTime)
                        .clearFields()
                        .inputFirstName(errors.getName())
                        .verifyInputName(errors.getName())
                        .loseFocus()
                        .verifyFirstNameErrors(errors.getErrorName())
                        .inputSurname(errors.getLast_name())
                        .verifyInputSurName(errors.getLast_name())
                        .verifyLastNameErrors(errors.getErrorSurname())
                        .inputEmail(errors.getEmail())
                        .verifyInputEmail(errors.getEmail())
                        .verifyEmailErrors(errors.getErrorEmail())
                        .assertAll();
            }
        }
```

EditMentorsDetailsPage_VerifyEditMentors_IncorrectData_AdminRole: Конструктор класу, який викликається для встановлення помилкових даних ментора та помилок, які очікуються під час тестування.

setUp: Метод, який викликається перед усіма тестами (в @BeforeClass). Цей метод налаштовує тестове середовище, реєструє нового користувача,

входить в систему як адміністратор, додає роль ментора, переходить на сторінку редагування менторів.

provide: Метод, який надає дані для тестів з використанням анотації @DataProvider. Він формує двовимірний масив об'єктів InvalidData, які містять помилкові дані для редагування менторів.

verifyEditMentors_IncorrectData: Тест, який виконує перевірку можливості редагування даних ментора з помилковими даними. Він проводить такі кроки:

1. Очищує поля вводу.
2. Вводить помилкові дані у поля.
3. Перевіряє введені дані та очікувані помилки для кожного поля.
4. Виконує перевірку помилок.

Окремо необхідно виділити цей кусок кода:

```
@DataProvider(name = "errors")
public Object[][] provide() {
    Object[][] list = new Object[data.length][1];
    for (int i = 0; i < data.length; i++) {
        list[i][0] = data[i];
    }
    return list;
}
```

Анотація @DataProvider у TestNG використовується для постачання тестам даних, які потрібні для виконання. Метод provide використовується для надання даних для тесту verifyEditMentors_IncorrectData.

Отже, метод provide повертає двовимірний масив об'єктів, де кожен рядок представляє окремий тестовий набір даних. Для цього використовується об'єкт InvalidData, який містить різні поля, що представляють помилкові дані для редагування менторів, а також самі повідомлення помилок.

Цей @DataProvider створює кілька наборів таких об'єктів InvalidData, і кожен набір передається у тест verifyEditMentors_IncorrectData. Під час

виконання цього тесту він використовує надані дані для вводу у відповідні поля, а потім перевіряє правильність введених даних та очікувані помилки.

Самі дані були передані з файлу PathsToFiles.Mentors.EDIT_MENTOR_ERRORS, який зберігається у форматі JSON і має наступний вигляд:


```
[
  {
    "name": "",
    "errorName" : "This field is required",
    "last_name": "",
    "errorSurname": "This field is required",
    "email": "",
    "errorEmail": "This field is required"


  },
  {
    "name": "a",
    "errorName" : "Too short",
    "last_name": "S",
    "errorSurname": "Too short",
    "email": "",
    "errorEmail": "This field is required"

  },
  {
    "name": "namenamenamenamenamenamenamenamenamenamenamenam",
    "errorName" : "Too long",
    "last_name": "namenamenamenamenamenamenamenamenamenamenamenamenam",
    "errorSurname": "Too long",
    "email": "",
    "errorEmail": "This field is required"
  },
  {

    "name": "-name",
    "errorName" : "Invalid first name",
    "last_name": "n2me",
    "errorSurname": "Invalid last name",
```

```
    "email": "email",
    "errorEmail": "Invalid email address"

  }
]
```

Саме ці значення передаються у об’єкти класу InvalidData, структуру якого класу можна побачити на рис 3.9.

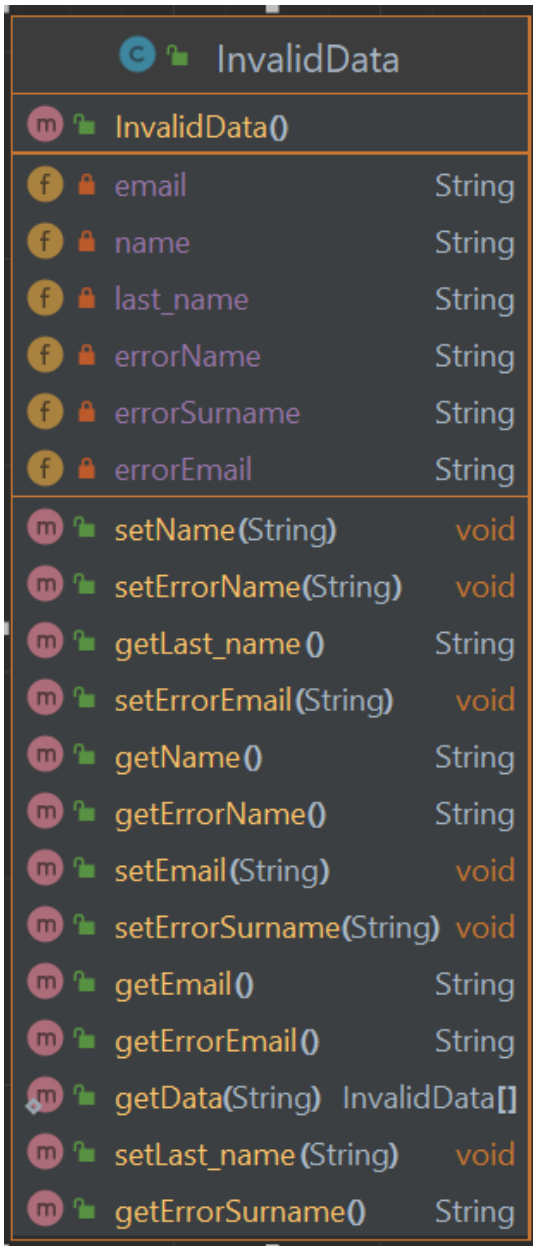

Рисунок 3.9. – Структура класу InvalidData

Поля цього класу співпадають з даними файла JSON.

Це дає можливість створювати тести з різними наборами вхідних даних, що допомагає виявити помилки та перевірити поведінку програми при різних умовах.

3.5.3 Тестування пошуку

Це можна наглядно побачити на класі MentrorsTablePage_VerifySearchingOfActiveMentors_AdminRole:

```
      UnassignedUser mentor;
          MentorsTablePage mentorsTablePage;

          public  MentrorsTablePage_VerifySearchingOfActiveMentors_AdminRole()
throws IOException {
              mentor = UnassignedUser.getUnassignedUser();
          }

          @BeforeClass
          public void precondition() throws IOException {

              mentorsTablePage = AuthPage.init(driver)
                      .clickRegistrationLink()
                      .registerUser(mentor)
                      .logInAs(Role.ADMIN, StudentsPage.class)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.UNASSIGNED_USERS,
UnassignedUsersPage.class)
                      .isAtPage(waitTime)
                      .addRole(mentor.getEmail(), UnassignedRole.MENTOR)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.MENTORS, MentorsTablePage.class);
          }

          @Test(description = "126-22-1м")
          public void verifySearchingOfActiveMentors() {
              String invalidData="6";
              List<String> expectFound = Arrays.asList(mentor.getFirstName(),
mentor.getLastName(), mentor.getEmail());
              mentorsTablePage.inputSearchMentor(mentor.getFirstName() + " " +
mentor.getLastName())
```

```
                        .verifyInputSearchField(mentor.getFirstName() + " " +
mentor.getLastName())
                        .verifyMentorsDataInTheTable(expectFound)
                        .inputSearchMentor(invalidData)
                        .verifyInputSearchField(invalidData)
                        .verifyNotFoundResult()
                        .assertAll();
            }
        }
```

MentorsTablePage_VerifySearchingOfActiveMentors_AdminRole: Конструктор класу, який встановлює дані ментора для тестування.

precondition: Метод, який викликається перед усіма тестами (в @BeforeClass). Цей метод налаштовує тестове середовище, реєструє нового користувача, входить в систему як адміністратор, додає роль ментора і переходить на сторінку таблиці менторів.

verifySearchingOfActiveMentors: Тест, який виконує перевірку можливості пошуку активних менторів. Він виконує наступні кроки:

1. Вводить ім'я та прізвище ментора у поле пошуку.
2. Перевіряє правильність введених даних та відображення цих даних у таблиці менторів.
3. Вводить недійсні дані у поле пошуку.
4. Перевіряє, чи відображається відповідне повідомлення про відсутність результатів пошуку.

Цей тест перевіряє функціонал пошуку активних менторів за допомогою відповідних даних та перевіряє відповідність результатів, що виводяться на сторінці з очікуваними даними.

3.5.4 Тестування сортування

Сортування відбувається за полями Ім’я, Прізвище та почтова адреса. Ці тести доволі схожі за змістом, тому вони зроблені як один тест MentorsTablePage_VerifySortingOfActiveMentors_AdminRole:

```
public class MentorsTablePage_VerifySortingOfActiveMentors_AdminRole
extends BaseTest {
    MentorsTablePage mentorsTablePage;

    public MentorsTablePage_VerifySortingOfActiveMentors_AdminRole() {
        mentorsTablePage = new MentorsTablePage(driver);
    }

    @BeforeClass
    public void setUp() throws IOException {
        mentorsTablePage = AuthPage
                .init(driver)
                .logInAs(Role.ADMIN, StudentsPage.class)
                .isAtPage(waitTime)
                .redirectTo(Endpoints.MENTORS, MentorsTablePage.class)
                .isAtPage(waitTime);
    }

    @Test(description = "DP213-67")
    public void verifySortingOfActiveMentors() {
        mentorsTablePage
                 .sortByName()
                .verifySoftByNameASC()
                .sortByName()
                .verifySoftByNameDEC()
                .sortBySurname()
                .verifySoftBySurNameASC()
                .sortBySurname()
                .verifySoftBySurNameDEC()
                .sortByEmail()
                .verifySoftByEmailASC()
                .sortByEmail()
                .verifySoftByEmailDEC()
                .assertAll();
    }
}
```

Конструктор класу:

Створює об'єкт MentorsTablePage для взаємодії зі сторінкою таблиці менторів.

Метод setUp:

Увійшовши в систему як адміністратор, переходить на сторінку з таблицею менторів. Переконується, що потрібна сторінка завантажена.

Тест verifySortingOfActiveMentors:

Сортування за ім'ям:

1. Спочатку сортує менторів за ім'ям у порядку зростання.
2. Перевіряє правильність сортування за ім'ям в алфавітному порядку.
3. Потім сортує менторів за ім'ям у порядку спадання.
4. Перевіряє правильність сортування за ім'ям у зворотному алфавітному порядку.
5. Сортування за прізвищем:
6. Сортує менторів за прізвищем у порядку зростання.
7. Перевіряє правильність сортування за прізвищем в алфавітному порядку.
8. Сортує менторів за прізвищем у порядку спадання.
9. Перевіряє правильність сортування за прізвищем у зворотному алфавітному порядку.
10. Сортування за електронною поштою:
11. Сортує менторів за електронною поштою у порядку зростання.
12. Перевіряє правильність сортування за електронною поштою в алфавітному порядку.
13. Сортує менторів за електронною поштою у порядку спадання.
14. Перевіряє правильність сортування за електронною поштою у зворотному алфавітному порядку.

Метод assertAll():

Завершує тест і виконує усі перевірки, щоб переконатися, що всі очікувані ствердження (assertions) виконані.

Цей тест виконує широкий спектр перевірок сортування таблиці менторів за різними критеріями та перевіряє правильність сортування даних на сторінці.

Тут окреме хотілось би виділити метод verifySoftByNameASC()

```
public MentorsTablePage verifySoftByNameASC() {
        List<String> actualResult = getMentorsName();
        List<String> expectResult = new ArrayList<>(actualResult);
        Collections.copy(expectResult,actualResult);
        Collections.sort(expectResult);
        softAssert.assertEquals(actualResult, expectResult);
        return this;
    }
```

Цей метод verifySoftByNameASC() виконує перевірку сортування таблиці менторів за ім'ям у порядку зростання. Ось детальний опис його роботи:

Отримання даних для порівняння:

Метод getMentorsName() отримує список імен менторів, які відображаються у таблиці. Ця функція повертає реальні імена менторів на сторінці.

Створення очікуваних даних для порівняння:

Тут створюється копія фактичних даних для майбутнього порівняння. Копіювання виконується за допомогою Collections.copy(), проте цей метод потребує, щоб expectResult був пустим або мав відповідний розмір для копіювання.

Сортування очікуваних даних:

Фактичні дані expectResult сортуються за алфавітом за допомогою Collections.sort(). Це створює список очікуваних даних у порядку зростання за ім'ям.

Порівняння фактичних та очікуваних даних:

Метод softAssert.assertEquals() порівнює фактичні дані actualResult із відсортованими очікуваними даними expectResult. Це дозволяє визначити, чи відображаються імена менторів у порядку зростання.

Повернення об'єкту сторінки:

Метод повертає об'єкт this (тобто об'єкт MentorsTablePage) для можливості подальших послідовних викликів методів цього класу.

Цей метод порівнює фактичні дані з очікуваними для перевірки сортування таблиці менторів за ім'ям у порядку зростання.

3.5.5Тестування надання ролі

Це можна побачити в коді тесту MentorsTablePage_VerifyAddMentortFunction_AdminRole:

```
      public class MentorsTablePage_VerifyAddMentortFunction_AdminRole extends
BaseTest {
          UnassignedUser mentor;
          MentorsTablePage mentorsTablePage;

          public MentorsTablePage_VerifyAddMentortFunction_AdminRole() {
              mentor = UnassignedUser.getUnassignedUser();
          }

          @BeforeClass
          public void setUp() throws IOException {
              mentorsTablePage = AuthPage.init(driver)
                      .clickRegistrationLink()
                      .registerUser(mentor)
                      .logInAs(Role.ADMIN, StudentsPage.class)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.MENTORS, MentorsTablePage.class)
                      .isAtPage(waitTime);
          }

          @Test(description = "DP213-159")
          public void verifyAddMentorFunction() {
              List<String>  exceptData  =  Arrays.asList(mentor.getFirstName(),
mentor.getLastName(), mentor.getEmail());
              mentorsTablePage
                      .addMentor()
                      .isAtPage(waitTime)
                      .addRole(mentor.getEmail(), UnassignedRole.MENTOR)
                      .redirectTo(Endpoints.MENTORS, MentorsTablePage.class)
                      .isAtPage(waitTime)
```

```
                        .inputSearchMentor(mentor.getFirstName()   +   "   "   +
mentor.getLastName())
                        .verifyInputSearchField(mentor.getFirstName()  +  "  "  +
mentor.getLastName())
                        .verifyMentorsDataInTheTable(exceptData)
                        .assertAll();

            }
      }
```

Конструктор MentorsTablePage_VerifyAddMentortFunction_AdminRole:

Створюється об'єкт класу UnassignedUser для роботи з даними ментора.

Метод setUp():

Виконує попередні умови для тестування, включаючи створення нового користувача-ментора, його реєстрацію, вхід в систему як адміністратор, та перехід на сторінку таблиці менторів.

Тест verifyAddMentorFunction():

1. Підготовка даних для перевірки:
2. Створюється список із даними ментора (exceptData), що містить його ім'я, прізвище та електронну пошту.
3. Виконання дій на сторінці:
4. Додається новий ментор.
5. Надається роль ментора для електронної адреси, яку використовував новий ментор.
6. Перевіряється наявність нового ментора в таблиці за допомогою пошуку його імені та перевірки відповідності даних у таблиці з очікуваними exceptData.
7. Завершується перевірка за допомогою assertAll().

Цей тест створений для перевірки коректності функції додавання нового ментора на сторінці таблиці менторів у режимі адміністратора.

3.5.6 Тестування спеціального функціоналу користувача

Ментор може додавати групи та курси, а також видаляти їх за потреби, тому для перевірки цього функционалу був розробленний тест EditMentorDetailsPage_VerifyAddDeleteGroupCourse_AdminRole:

```
      public class EditMentorDetailsPage_VerifyAddDeleteGroupCourse_AdminRole
extends BaseTest {
          UnassignedUser mentor;
          EditMentorsDetailsPage editMentorsPage;

          public  EditMentorDetailsPage_VerifyAddDeleteGroupCourse_AdminRole()
{
              mentor= UnassignedUser.getUnassignedUser();
          }
          @BeforeClass
          public void setUp() throws IOException {

              editMentorsPage = AuthPage.init(driver)
                      .clickRegistrationLink()
                      .registerUser(mentor)
                      .logInAs(Role.ADMIN, StudentsPage.class)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.UNASSIGNED_USERS,
UnassignedUsersPage.class)
                      .isAtPage(waitTime)
                      .addRole(mentor.getEmail(), UnassignedRole.MENTOR)
                      .isAtPage(waitTime)
                      .redirectTo(Endpoints.MENTORS, MentorsTablePage.class)
                      .inputSearchMentor(mentor.getFirstName()+"
"+mentor.getLastName())
                      .editMentors(0)
                      .isAtPage(waitTime)
                      .choseGroup(0)
                      .addGroup()
                      .choseCourse(0)
                      .addCourse()
                      .saveMentor()
                      .isAtPage(waitTime)
                      .inputSearchMentor(mentor.getFirstName()+"
"+mentor.getLastName())
                      .editMentors(0)
```

```
                        .isAtPage(waitTime);
            }
            @Test(description = "DP213-159")
            public void verifyAddMentorFunction(){
                List<String>emptyList=new ArrayList<>();
                List<String>listGroupCard=
Arrays.asList(editMentorsPage.getValueOfGroupCard(0));

List<String>listCourseCard=Arrays.asList(editMentorsPage.getValueOfCourseCard
(0));
                editMentorsPage
                        .deleteGroup(0)
                        .deleteCourse(0)
                        .verifyListOfGroupsCard(emptyList)
                        .verifyListOfCourseCard(emptyList)
                        .saveMentor()
                        .isAtPage(waitTime)
                        .inputSearchMentor(mentor.getFirstName()+"
"+mentor.getLastName())
                        .editMentors(0)
                        .choseGroup(0)
                        .addGroup()
                        .verifyListOfGroupsCard(listGroupCard)
                        .choseCourse(0)
                        .addCourse()
                        .verifyListOfCourseCard(listCourseCard)
                        .saveMentor()
                        .isAtPage(waitTime)
                        .inputSearchMentor(mentor.getFirstName()+"
"+mentor.getLastName())
                        .editMentors(0)
                        .verifyListOfGroupsCard(listGroupCard)
                        .verifyListOfCourseCard(listCourseCard)
                        .assertAll();
            }

        }
```

Конструктор EditMentorDetailsPage_VerifyAddDeleteGroupCourse_AdminRole:

Створює об'єкт класу UnassignedUser для роботи з даними ментора.

Метод setUp():

Виконує попередні умови для тестування, включаючи створення нового користувача-ментора, його реєстрацію, вхід в систему як адміністратор, перехід на сторінку редагування менторів, додавання групи та курсу ментору та їх збереження.

Тест verifyAddMentorFunction():

Підготовка даних для перевірки:

1. Створюється порожній список emptyList.
2. Створюються списки listGroupCard і listCourseCard зі значеннями групи та курсу на картці ментора.

Виконання дій на сторінці:

3. Видаляється група та курс, перевіряється, що списки порожні.
4. Зберігаються зміни, перевіряється, чи вони відображені на сторінці.
5. Повторно відкривається карта ментора, додаються група та курс, перевіряється відображення даних.
6. Зберігаються зміни, перевіряється, чи вони відображені на сторінці.
7. Знову відкривається карта ментора, перевіряється, чи збереглися дані.

Цей тест призначений для перевірки коректності функціоналу додавання та видалення групи та курсу на сторінці редагування даних ментора в режимі адміністратора.

3.6 Формування звітності

Для формування звітності було використано Allure-Testng, цей плагін підключається за допомогою файла pom.xml та наступного коду:

```
<dependency>
            <groupId>io.qameta.allure</groupId>
            <artifactId>allure-testng</artifactId>
            <version>2.15.0</version>
        </dependency>
```

Результат кожного тесту формується в файлі JSON та зберігається у директорії allure-results.

Для подальшого перегляду тестів використовувалось все сама програма allure. Для її запуску необхідно через командний рядок виконати наступну команду:

```
/g/allure-2.24.1/bin/allure.bat serve
```

Це запустить локально сервісу allure і відкриє звіт.

Цю команду необхідно виконувати с директорії де знаходяться всі звіти.

На рис. 3.10. Можна переглянути загальну статистику всіх тестів, що виконувалися.

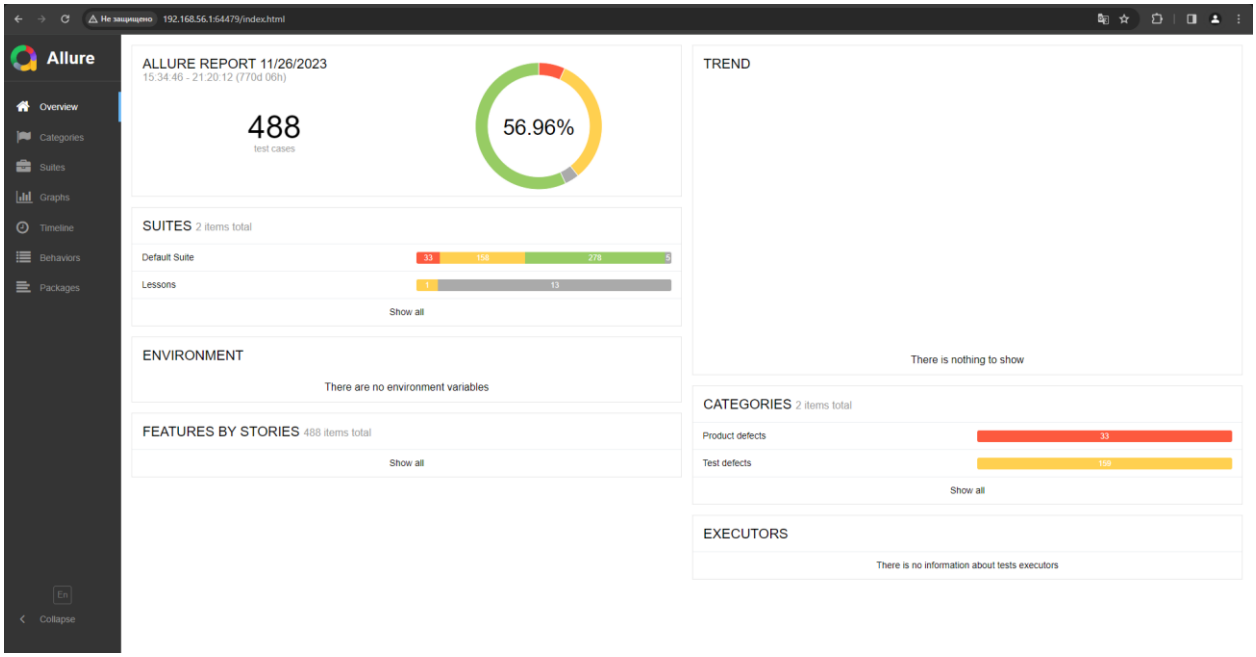

Рисунок 3.10 – Загальна статистика всіх тестів

Рис. 3.11 демонструє тести Product Defects, які не пройшли перевірку, а елементи веб додатку не відповідають вимогам.

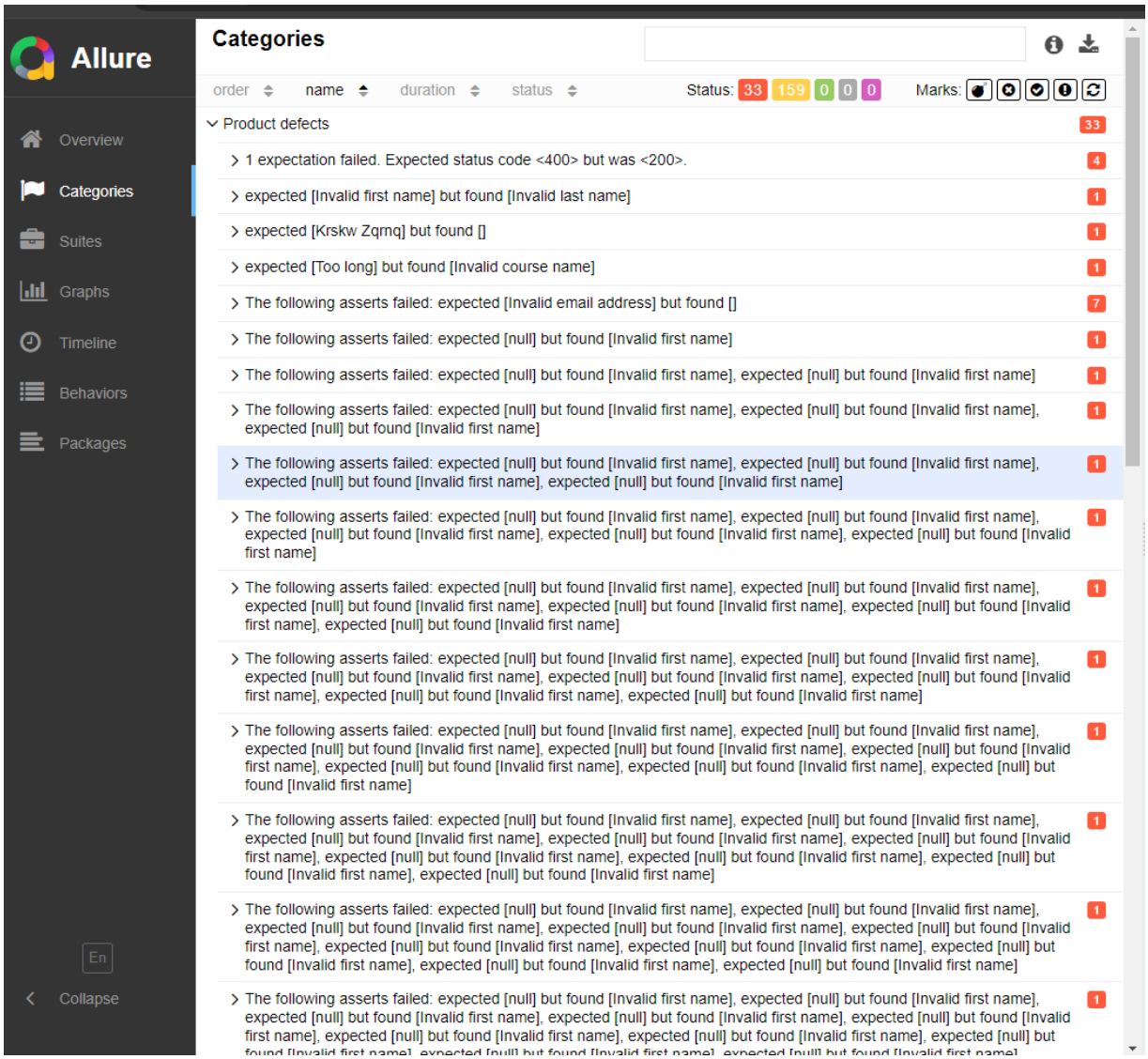

Рисунок. 3.11 - Product Defects тести

Рис. 3.12 демонструє нам тести із категорії Test defects, тобто це ті тести, що не пройшли через логіку самого тести, а не через помилки в розробці додатку.

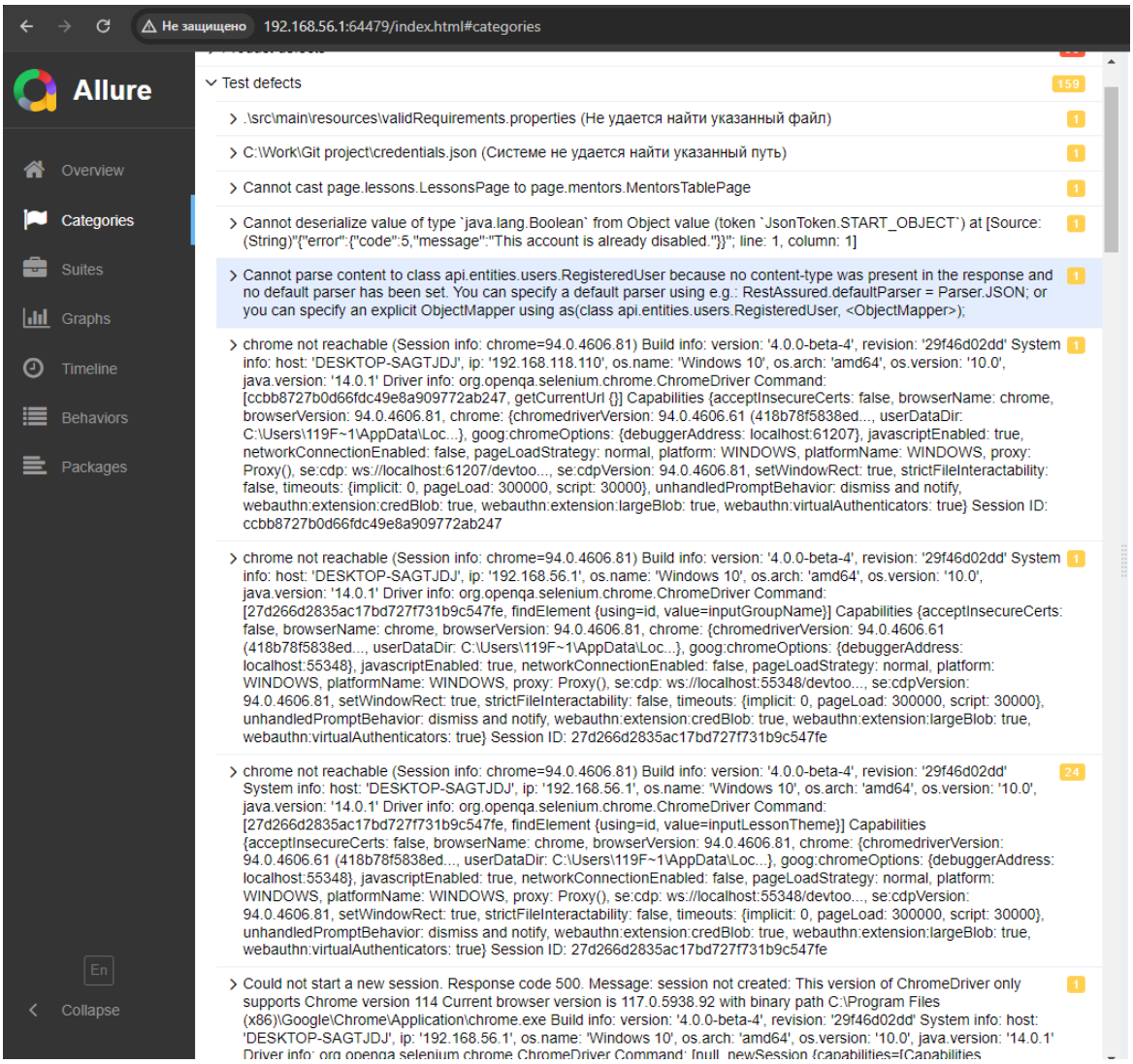

Рисунок 3.12 - Test defects тести

Звичайно можна переглянути якийсь конкретний тест, які перевірки він пройшов, а які були завалені. На рис.3.13 можна побачити тест EditMentorsDetailsPage_VerifyEditMentors_IncorrectData_AdminRole.

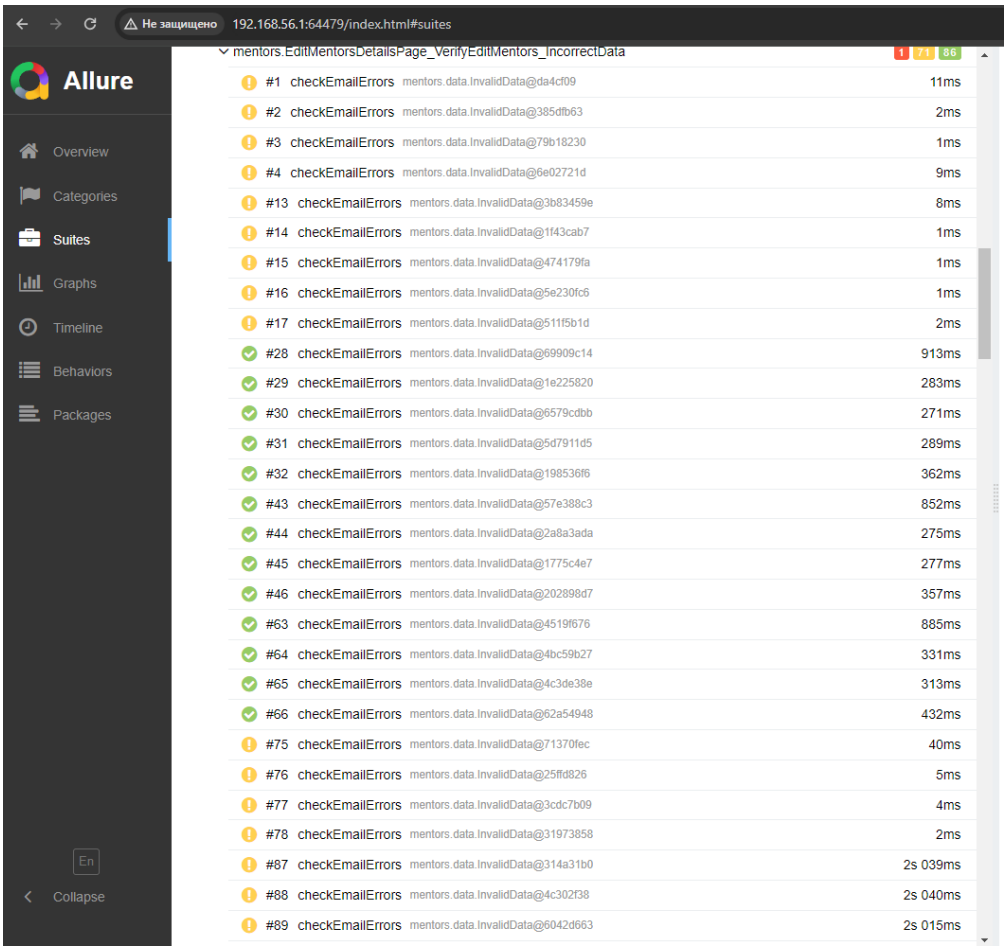

Рисунок 3.13 – Тест EditMentorsDetailsPage_VerifyEditMentors_IncorrectData_AdminRole

На рис. 3.14 можна побачити інформацію зі сторінки Graphs, яка містить графіки співвідношення результатів тестів та час їх виконання.

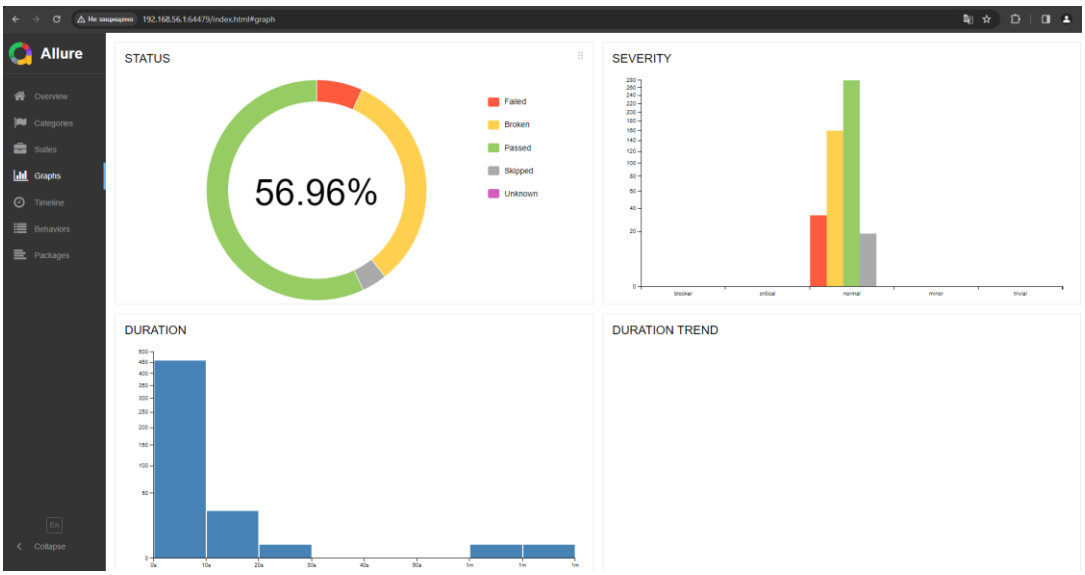

Рисунок 3.14 – Сторінка Graphs

3.7 Висновки до третього розділу

У цьому розділі була розроблена інформаційна система для тестування веб-додатків, де використовувалися різні технології та підходи.

Для взаємодії з елементами веб-сторінок були використані шляхи елементів (XPath), що дозволяє точно вибирати та взаємодіяти з елементами під час автоматизованого тестування. Описано та впроваджено шаблон проектування Page Object, який спрощує управління тестами та забезпечує зрозумілу структуру для взаємодії користувачів із сторінками веб-додатку.

Розроблені тести відповідають технічним вимогам, що дозволяє перевірити різні аспекти функціональності та надійності веб-додатку. Підключено систему звітування Allure TestNG, що сприяє зручному та інформативному аналізу результатів тестування, дозволяючи легко виявляти проблемні моменти та робити висновки про якість програмного продукту.

ВИСНОВКИ

В ході даної кваліфікаційної роботи:

1. Проведено аналіз основ тестування та його різноманітних аспектів. Результатом цього аналізу стало прийняття рішення щодо розробки інформаційної системи для тестування веб-додатків. Це рішення спрямоване на покращення якості та надійності програмного продукту шляхом систематичного й ефективного тестування.

2. Оцінюючи інструменти та технології тестування, було обрано Apache Maven для керування проектом, визначено TestNG та Selenium як ключові інструменти для автоматизації веб-тестування, що дозволило створити надійну та стійку інфраструктуру для тестування.

3. Розроблена інформаційна система призначена для автоматизації процесів тестування веб-додатків, що сприятиме оперативному виявленню помилок та забезпечить високу якість програмного забезпечення.

4. Реалізація шаблону проектування Page Object спрощує управління тестами та забезпечує легку структуру для взаємодії з користувачами на сторінках веб-додатку. Використання Page Object спростило взаємодію з елементами веб-сторінок, роблячи процес тестування більш модульним та зрозумілим.

5. Взаємодія з елементами веб-сторінок була здійснена за допомогою шляхів елементів (XPath), що сприяє точній взаємодії та вибору елементів під час автоматизованого тестування.

6. Як предмет тестування був обраний веб-додаток для організації навчального процесу.

7. Розроблені тести відповідають технічним вимогам, що дозволяє перевірити різні аспекти функціональності та надійності веб-додатку. Підключення системи звітування Allure TestNG є важливим кроком у зручному та інформативному аналізі результатів тестування, сприяючи виявленню проблем та забезпеченню якості програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке ЖЦП (Життєвий Цикл Розробки Програмного Забезпечення)? aws.amazon.com. URL: https://aws.amazon.com/what-is/sdlc/?nc1=h_ls (дата відвідування: 8 вересня 2023).
2. Принципи тестування. qalight.ua. URL: https://qalight.ua/baza-znaniy/printsipi-testuvannya/ (дата відвідування: 9 вересня 2023).
3. Відмінність між тестуванням на чорну, білу та сіру скриньку. geeksforgeeks.org. URL: https://www.geeksforgeeks.org/difference-between-black-box-vs-white-vs-grey-box-testing/ (дата відвідування: 14 вересня 2023).
4. Тестування на білу, чорну та сіру скриньку. qalight.ua. URL: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/ (дата відвідування: 15 вересня 2023).
5. Відмінність між тестуванням на чорну, білу та сіру скриньку. geeksforgeeks.org. URL: https://www.geeksforgeeks.org/difference-between-black-box-vs-white-vs-grey-box-testing/ (дата відвідування: 14 вересня 2023).
6. Практична Тестова Піраміда. martinfowler.com. URL: https://martinfowler.com/articles/practical-test-pyramid.html (дата відвідування: 20 вересня 2023).
7. Тестування Одиниць. highload.today. URL: https://highload.today/unit-testing/ (дата відвідування: 22 вересня 2023).
8. Тестування Сервісів. theqalead.com. URL: https://theqalead.com/test-management/service-testing/ (дата відвідування: 25 вересня 2023).
9. Практична Тестова Піраміда - UI Тести. martinfowler.com. URL: https://martinfowler.com/articles/practical-test-pyramid.html#UiTests (дата відвідування: 27 вересня 2023).

10. Maven - Особливості. maven.apache.org. URL: https://maven.apache.org/maven-features.html (дата відвідування: 30 вересня 2023).
11. TestNG - Анотації. testng.org. URL: https://testng.org/doc/documentation-main.html#annotations (дата відвідування: 3 жовтня 2023).
12. Selenium WebDriver. training.qatestlab.com. URL: https://training.qatestlab.com/blog/technical-articles/selenium-webdriver/ (дата відвідування: 5 жовтня 2023).
13. Моделі Сторінок в Selenium. selenium.dev. URL: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/ (дата відвідування: 8 жовтня 2023).

ДОДАТОК А

Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кільк. аркушів	Примітки
1				
2		Документація		
3				
4	**ІТКІ.КР.22.11.ДА.ПЗ**	Пояснювальна записка	94	Формат А4
5				
6		Презентація		
7				
8		Диск CD-R з презентацією	1	Диск CD-R
9				

Зм.	Ар-куш	№ докум.	Підпис	Дата	**ІТКІ.КР.22.11.ДА.ПЗ.**			
Розроб.					**Матеріали кваліфікаційної роботи**	Літ.	Аркуш	Арку-шів
Керівник		Соколова Н.О.				Н	1	1
Рецензент		Клименко А.В				НТУ «ДП», 12; 126-22м-1		
Н.контр.		Коротенко Г.М.						
Зав. каф.		Гнатушенко В.В.						

ДОДАТОК Б

Лістинг коду

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>
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.11.0</version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>
        <dependency>
            <groupId>io.qameta.allure</groupId>
            <artifactId>allure-testng</artifactId>
            <version>2.15.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>4.5.13</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.13.0</version>
        </dependency>
        <dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>webdrivermanager</artifactId>
            <version>5.4.1</version>
```

```
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <version>3.21.0</version>
        </dependency>
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>rest-assured</artifactId>
            <version>4.4.0</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.4.0</version>
        </dependency>
        <dependency>
            <groupId>org.awaitility</groupId>
            <artifactId>awaitility</artifactId>
            <version>4.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.awaitility</groupId>
            <artifactId>awaitility-proxy</artifactId>
            <version>3.1.6</version>
        </dependency>
    </dependencies>
    <build>

    <plugins>
        <plugin>
            <artifactId>maven-resources-plugin</artifactId>
            <version>3.0.2</version>
            <configuration>
                <resources>
                    <resource>
                        <directory>src/main/resources</directory>
                        <includes>
                            <include>**/*.json</include>
                            <include>**/*.xml</include>
                            <include>**/*.properties</include>
```

```
                </includes>
                <filtering>false</filtering>
            </resource>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.xml</include>
                </includes>
            </resource>
        </resources>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.2.0</version>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>test-jar</goal>
            </goals>
        </execution>
    </executions>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.1.1</version>
    <configuration>
        <descriptors>
            <descriptor>src/main/resources/assembly.xml</descriptor>
        </descriptors>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
```

```
                <archive>
                    <manifest>
                        <mainClass>org.testng.TestNG</mainClass>
                    </manifest>
                </archive>
            </configuration>
            <executions>
                <execution>
                    <id>make-assembly</id>
                    <phase>package</phase>
                    <goals>
                        <goal>single</goal>
                    </goals>
                </execution>
            </executions>

        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.0</version>
            <configuration>
                <suiteXmlFiles>

<suiteXmlFile>src/test/resources/testng.xml</suiteXmlFile>
                </suiteXmlFiles>
            </configuration>
        </plugin>

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

Main:

```
public class Main {
    public static void main(String[] args) throws URISyntaxException,
IOException {
```

```
        TestNG testng = new TestNG();
        InputStream                          inputStream                          =
Main.class.getClassLoader().getResourceAsStream("testng.xml");
        SuiteXmlParser parser = new SuiteXmlParser();
        XmlSuite suite = parser.parse("testng.xml", inputStream, true);
        System.out.println(suite);
        testng.setXmlSuites(new ArrayList<>(Collections.singleton(suite)));
        testng.run();
    }
}
```

MentorsDetailsPage:

```
public class MentorsDetailsPage extends Page<MentorsDetailsPage> {
    public MentorsDetailsPage(WebDriver driver) {
        super(driver);
    }

    @Override
    public boolean isAt() {
        return false;
    }

    @FindBy(xpath = MENTORS_DETAILS_TAB_XPATH)
    protected WebElement mentorsDetailsTab;
    @FindBy(xpath = EDIT_MENTOR_TAB_XPATH)
    protected WebElement editMentorsTab;
    @FindBy(xpath = ARROW_BUTTON_XPATH)
    protected WebElement arrowButton;
    @FindBy(xpath = MENTORS_FIRST_NAME_XPATH)
    protected WebElement firstNameMentor;
    @FindBy(xpath = MENTORS_LAST_NAME_XPATH)
    protected WebElement lastNameMentor;
    @FindBy(xpath = MENTOR_EMAIL_NAME_XPATH)
    protected WebElement emailMentor;
    @FindBy(xpath = LIST_OF_GROUP_LINK)
    protected List<WebElement> listOfGroups;
    @FindBy(xpath = LIST_OF_COURSE_LINK)
    protected List<WebElement> listOfCourses;


    public MentorsTablePage backToMentorsTable() {
```

```
    clickElement(arrowButton);
    return new MentorsTablePage(driver);
}

public EditMentorsDetailsPage viewEditMentorsTab() {
    clickElement(editMentorsTab);
    return new EditMentorsDetailsPage(driver);
}

public MentorsDetailsPage viewMentorsDetailTab() {
    clickElement(mentorsDetailsTab);
    return this;
}

public MentorsDetailsPage verifyFirstName(String firstName) {
    softAssert.assertEquals(firstNameMentor.getText(), firstName);
    return this;
}

public MentorsDetailsPage verifyLastName(String lastName) {
    softAssert.assertEquals(lastNameMentor.getText(), lastName);
    return this;
}

public MentorsDetailsPage verifyEmail(String email) {
    softAssert.assertEquals(emailMentor.getText(), email);
    return this;
}

public MentorsDetailsPage verifyGroups(List<String> expectGroups) {
    List<String> actualGroups = new ArrayList<>();
    for (int i=0;i<listOfGroups.size();i++){
        actualGroups.add(listOfGroups.get(i).getText());
    }
    softAssert.assertEquals(actualGroups,expectGroups);
    return this;
}
public MentorsDetailsPage verifyCourse(List<String> expectCourse) {
    List<String> actualCourse = new ArrayList<>();
    for (int i=0;i<listOfCourses.size();i++){
        actualCourse.add(listOfCourses.get(i).getText());
    }
```

```
        softAssert.assertEquals(actualCourse,expectCourse);
        return this;
    }
```

Page:

```
public abstract class Page<I extends Page<I>> extends BaseElement {

    protected Header header;
    protected SideBar sideBar;
    protected Pagination pagination;

    public Page(WebDriver driver) {
        super(driver);
        sideBar = new SideBar(driver);
        header = new Header(driver);
        pagination = new Pagination (driver);
    }

    public <T> T redirectTo(String url, Class<T> type) {
        driver.get(url);
        await().until(() -> driver.getCurrentUrl().equals(url));
        return type.cast(Endpoints.getPages(driver).get(url));
    }

    public Header getHeader() {
        return header;
    }

    public MyProfilePage clickUserIcon() {
        return header.clickUserIcon();
    }


    public SideBar getSideBar() {
        return sideBar;
    }

    public abstract boolean isAt();

    public I isAtPage(long timeout) {
        try {
```

```
            await().atMost(timeout, TimeUnit.SECONDS)
                    .ignoreExceptions()
                    .until(() -> isAt());
            return (I) this;
        } catch (Exception e) {
            return null;
        }
    }

    public AuthPage logOut() throws IOException {
        return header.logOut();
    }
}
```

BaseTest

```
public class BaseTest {

    protected final Logger log = Logger.getLogger(getClass());
    protected WebDriver driver;
    protected long waitTime;

    @BeforeSuite
    protected void setup() {
        WebDriverManager.chromedriver().setup();
        log.info("Chromedriver is set up");
    }

    @BeforeClass
    protected void setupTest() {
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));
        driver.manage().window().maximize();
        waitTime = 5;
        driver.get(Endpoints.BASE_URL);
        log.info("Initialise web driver");
    }

    @AfterClass
    protected void teardown() {
        driver.quit();
        log.info("Quit");
    }
```

```
}
```

UnassignedUsersPage_VerifyAddMentorsRole:

```
public class UnassignedUsersPage_VerifyAddMentorsRole extends BaseTest {
    private UnassignedUser mentor;

    @BeforeClass
    public void setUp () {
        mentor = UnassignedUser.getUnassignedUser();
    }
    @Test
    public void verifyAddMentorRole() throws IOException {
        List<String> actualData=AuthPage.init(driver)
                .clickRegistrationLink()
                .registerUser(mentor)
                .logInAs(Role.ADMIN, StudentsPage.class)
                .isAtPage(waitTime)
                .redirectTo(Endpoints.UNASSIGNED_USERS,
UnassignedUsersPage.class)
                .addRole(mentor.getEmail(), UnassignedRole.MENTOR)
                .redirectTo(Endpoints.MENTORS, MentorsTablePage.class)
                .inputSearchMentor("a")
                .inputSearchMentor(mentor.getFirstName()+"                    "+
mentor.getLastName())
                .getMentorsData();
        List<String>                                                exceptData=
Arrays.asList(mentor.getFirstName(),mentor.getLastName(),mentor.getEmail());
        Assert.assertEquals(actualData,exceptData);
    }
```

InvalidData:

```
public class InvalidData {
    private String name;
    private String errorName;
    private String last_name;
    private String errorSurname;
    private String email;
    private String errorEmail;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
    public String getErrorName() {
        return errorName;
    }
    public void setErrorName(String errorName) {
        this.errorName = errorName;
    }
    public String getLast_name() {
        return last_name;
    }

    public void setLast_name(String last_name) {
        this.last_name = last_name;
    }

    public String getErrorSurname() {
        return errorSurname;
    }

    public void setErrorSurname(String errorSurname) {
        this.errorSurname = errorSurname;
    }

    public String getEmail() {
        return email;
    }

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

    public String getErrorEmail() {
        return errorEmail;
    }

    public void setErrorEmail(String errorEmail) {
        this.errorEmail = errorEmail;
    }
    public static InvalidData[] getData(String filename) throws IOException {
        return      new      ObjectMapper().readValue(new      File(filename),
InvalidData[].class);
    }
```