БАКАЛАВРСЬКА РОБОТА

БР.КІ-01.00.00.000 ПЗ

Група КІ-21-1

Бабчук Ірина

2025

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

Бабчук Ірина Сергіївна

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Розробка пакету автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium

Комп'ютерна інженерія
(назва освітньої програми)

F7 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____*Бабчук І.С.*_____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____*Мануляк І.З., к.т.н., доцент*_____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри КСМ

*д.т.н.,професор*_____*С.І. Мельничук*
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025 рік

Івано-Франківський національний технічний університет нафти і газу

Факультет *Інформаційних технологій*

Кафедра *Комп'ютерних систем і мереж*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *F7 – Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ:

Зав. кафедрою КСМ

С.І. Мельничук

« 5 » червня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Бабчук Ірині Сергіївні

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Розробка пакету автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium*

керівник проекту (роботи) *Мануляк Ірина Зіновіївна, к.т.н., доцент*.

затверджена наказом вищого навчального закладу від *05.05.2025 № 275/7*

2. Строк подання студентом проекту (роботи) *12 червня 2025 р.*

3. Вихідні дані до роботи *Методичні вказівки, технічна література*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *Аналіз предметної області. Розробка підходів до тестування. Реалізація пакету тестування.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____.

6. Консультанти розділів роботи

Розділ	Консультант	Підпис, дата
Нормоконтроль		
Антиплагіат		

7. Дата видачі завдання *29 січня 2025 р.* .

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Постановка задачі*	*Лютий, 2025*	
2	*Аналіз предметної області*	*Березень, 2025*	
3	*Розробка підходів до тестування*	*Квітень, 2025*	
4	*Реалізація пакету тестування*	*Травень, 2025*	
5	*Оформлення пояснювальної записки*	*Червень, 2025*	

Студент _____ *Бабчук І.С.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Мануляк І.З.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Під час виконання бакалаврської роботи було розроблено пакет автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium

В першому розділі проведено порівняльний аналіз методик ручного та автоматизованого тестування web-додатків. В результаті проведеного аналізу було обрано Selenium як інструмент тестування web-додатків.

В другому розділі виконано аналіз фронтенд частини інформаційної системи університету та функціональних вимог до неї. Також було вибрано стратегії тестування та структуру пакету тестування та тест-кейсів.

В третьому розділі організовано середовище для тестування, розроблено пакет автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium, запущено тестування та здійснено аналіз його результатів.

Ключові слова: тестування, ручне тестування, автоматизоване тестування, програмне забезпечення, web-додаток, Selenium, Java.

ABSTRACT

During the bachelor's thesis, a package of automatic tests for the functioning of the web application of the IFNTUNG information system using Selenium was developed.

The first section provides a comparative analysis of the methods of manual and automated testing of web applications. As a result of the analysis, Selenium was selected as a tool for testing web applications.

The second section provides an analysis of the front-end part of the university's information system and its functional requirements. Testing strategies and the structure of the testing package and test cases were also selected.

In the third section, a testing environment is organized, a package of automatic tests for the functioning of the web application of the IFNTUNG information system using Selenium was developed, testing was launched and its results were analyzed.

Keywords: testing, manual testing, automated testing, software, web application, Selenium, Java.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Інтерфейс прикладного програмування;

IDE – Інтегроване середовище розробки;

RC – Дистанційне керування.

ВСТУП

Актуальність теми дослідження. Важливість процесу тестування програмного забезпечення зумовила виникнення тенденцій використання технічних засобів для перевірки його якості. Основним напрямком є впровадження систем автоматизованого тестування. Для автоматизації тестування є багато інструментів, рішень і засобів. Автоматизація тестування не тільки сприяє заощадженню часу на розробку, а й підвищує безпеку і надійність створюваних програмних продуктів.

Для автоматизованого тестування часто використовують спеціалізовані фреймворки, які дозволяють економити час та матеріальні засоби в процесі контролю якості розробленого web-додатка. В зв'язку з вищевказаним в даний час при розробці великої кількості web-додатків використовують спеціальні тестові бібліотеки.

Використання автоматизованих тестів створює умови для автоматичного надання розробникам звітів про стан розробленого програмного забезпечення, багаторазового запуску тестів, зменшення впливу людини на процес тестування.

Таким чином, в даний час актуальним є застосування інструментів автоматизованого тестування, і розробка пакету автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ.

Об'єкт дослідження. Процес розробки пакету тестів для тестування клієнтської частини web-додатка інформаційної системи ІФНТУНГ за допомогою Selenium.

Предмет дослідження. Методи і засоби тестування клієнтської частини web-додатків.

Мета та завдання роботи. Метою даної бакалаврської роботи є розробка пакету автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ. Для досягнення поставленої мети бакалаврської роботи необхідно виконати наступні завдання:

– здійснити порівняльний аналіз методик ручного та автоматизованого тестування web-додатків;

– проаналізувати Selenium як інструмент тестування web-додатків;

– виконати аналіз фронтенд частини інформаційної системи університету та функціональних вимог до неї;

– обрати стратегію тестування;

– розробити структуру пакету тестування та тест-кейсів;

– організувати середовище для тестування;

– розробити пакет автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium;

– запустити тестування та виконати аналіз його результатів;

– оцінити результати тестування.

Методи дослідження. Використано метод порівняльного аналізу під час дослідження методик ручного та автоматизованого тестування web-додатків.

Практичне значення. Розроблено пакет автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium, що дозволить скоротити час тестування web-додатка, створить умови для багаторазового запуску тестів, мінімізує вплив людини на процес тестування.

Впровадження результатів. Результати бакалаврської роботи впроваджені в Івано-Франківському національному технічному університеті нафти і газу, що підтверджено актом впровадження.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Порівняльний аналіз методик ручного та автоматизованого тестування web-додатків

Тестування web-додатків необхідне, щоб уникнути потрапляння неякісного продукту до кінцевих споживачів. Попереднє ручне тестування дозволяє усувати помилки заздалегідь та передбачати їх появу у програмному забезпечені на етапі формування завдань, створення дизайну та розробки продукту.

Ручне тестування дозволяє оцінити працездатність розробленого програмного забезпечення. Даний етап розробки допомагає виявити помилки, які були зроблені під час створення програмного забезпечення. Після цього інформація про виявлені помилки направляється розробникам для їх усунення. Потім програмне забезпечення повертається для повторного тестування на предмет перевірки чи були усунені всі помилки, які були виявлені на першому етапі, та чи не було допущено нових помилок [1].

Головні особливості ручного тестування:

– гнучкість;

– людиноцентричність.

Гнучкість полягає в тому, що під час ручного тестування, тестувальники можуть підлаштовувати підхід відносно змін у програмному забезпеченні.

Людиноцентричність полягає в тому, що під час ручного тестування, тестувальники можуть оцінювати комфорт при користуванні та виявляти проблеми у кінцевих користувачів.

На ранніх стадіях розробки помилки можна знаходити швидше, а їх вчасне усунення зекономить фінанси для компанії у майбутньому.

Дослідницьке тестування є різновидом ручного тестування, при якому попередньо тести не створюються, а web-додаток вивчається та перевіряється у

самому процесі. Дослідницьке тестування є звичайним компонентом ручного тестування.

Розпочинати ручне тестування можна на початку життєвого циклу розробки програмного забезпечення, навіть до того, як програма повністю розроблена [2].

Під час тестування складних сценаріїв, які важко автоматизувати, ручне тестування добре працює. Тестери можуть оцінити взаємодію між складними робочими процесами та різними компонентами.

Тестування вручну є хорошим способом оцінити інтерфейс користувача програми. З точки зору користувача тестери можуть оцінити загальний дизайн, зовнішній вигляд та зручність.

Ручне тестування програмного забезпечення включає в себе декілька фаз, що мають своє призначення і під час яких відбувається виявлення дефектів та перевірка системи на відповідність вимогам [3]. Основними етапами є:

– модульне тестування (Unit Testing);

– інтеграційне тестування (Integration Testing);

– системне тестування (System Testing);

– приймальне тестування (Acceptance Testing).

Модульне тестування є першим етапом тестування, коли перевіряють модулі програми, такі як функції, методи чи класи [4]. Головне впевнитися, що кожен компонент працює правильно згідно поставлених вимог. Тому створюють тестові сценарії, які містять різні можливі варіанти використання модуля. Спеціальні тестові фреймворки зазвичай застосовуються розробниками для проведення модульного тестування.

Інтеграційне тестування здійснюється після успішного модульного тестування і полягає в перевірці взаємодії між модулями або компонентами системи. Це необхідно для того, щоб переконатися, що об’єднані частини програмного забезпечення коректно взаємодіють одна з одною, а передача даних і виконання функцій відбуваються без помилок. Інтеграційне тестування

допомагає виявити такі проблеми, як несумісність між модулями, помилки у взаємодії з базою даних або помилки у викликах API.

На етапі системного тестування перевіряється робота всього програмного забезпечення як єдиного цілого. Тестується функціональність, продуктивність, безпека та зручність використання системи. Основна мета системного тестування полягає в тому, щоб переконатися, що всі компоненти працюють разом відповідно до вимог, і система готова до використання. Системне тестування виконується в середовищі, максимально наближеному до реальних умов експлуатації.

Приймальне тестування є завершальним етапом тестування, яке дозволяє оцінити, чи відповідає програмне забезпечення бізнес-вимогам та потребам користувачі до продукту. На цьому етапі, при участі замовників або представників користувачів, перевіряють систему, чи відповідає вона потребам користувачів. Продукт можна запускати у промислову експлуатацію, якщо він успішно проходить приймальне тестування.

На рисунку 1.1 наведено кроки процесу ручного тестування:

Рисунок 1.1 – Процес ручного тестування

Аналіз вимог включає ознайомлення з документацією проекту програмного забезпечення, посібниками та програмою, що тестується.

Далі виконується створення плану тестування, який охоплює всі вимоги.

Після цього відбувається створення тестових кейсів, які відповідають усім вимогам, що описані у документації.

Виконання тестових випадків проводиться, коли було переглянуто та створено базові тестові випадки разом із керівником групи та клієнтом.

У журналі дефектів фіксуються баг-репорти та через трекінгову систему про них повідомляється розробникам.

Коли помилки виправлено, знову потрібно виконати невдалі тестові випадки, щоб переконатися, що вони пройшли.

Переваги ручного тестування:

– цей вид тестування виявляє майже кожну помилку в програмному забезпеченні та використовується для тестування динамічно змінюваних дизайнів графічного інтерфейсу, таких як макет, текст тощо;

– потребує менших витрат, оскільки не вимагає жодних навичок високого рівня чи певного типу інструменту;

– під час використання методу тестування чорного ящика не потрібні знання програмування. Це легко освоїти для нових тестувальників;

– є ефективним для незапланованих змін у програмі, оскільки його можна легко адаптувати.

Недоліки ручного тестування:

– ручне тестування є менш надійним, оскільки воно забезпечує тестування аспектів, які тільки доступні для користувачів web-додатків;

– щоразу потрібно створювати та проходити окремі тест кейси вручну із залученням тестувальників. Відтворення не автоматизується, для кожного нового програмного забезпечення необхідно розробляти нові тест кейси;

– кожне ручне виконання тесту є незалежним. Нове виконання вимагає від тестувальників повторити всі кроки спочатку. Це призводить до неузгоджених результатів – різні тестери можуть дотримуватись дещо різних підходів, що призводить до відмінностей у результатах тестування; проблем з

масштабованістю: зі збільшенням кількості тестів потрібно більше тестувальників, що робить його неефективним для великомасштабних проектів;

– для ручного тестування потрібні численні людські ресурси, і є деякі завдання, які неможливо виконати вручну;

– часто для перевірки web-додатків потрібно мати широкий досвід та приклади рішень подібних задач. Від цього залежить покриття програми тестами та кількість виявлених помилок;

– оскільки у ручному тестуванню залучені безпосередньо люди, сам процес потребує значних витрат часу;

– ручне тестування вимагає від тестувальників виконання кожного кроку вручну. Це добре для невеликих проектів, але для тих, хто має сотні чи тисячі тестів, ручне тестування просто непродуктивне;

– люди не можуть обробляти дані так швидко, як машини, і люди схильні до втоми та відволікання. Якщо сталася помилка, тестери повинні повторити кроки, ще більше подовжуючи час тестування;

– попри те, що для ручного тестування не потрібні спеціальні інструменти чи сценарії, довгострокові витрати можуть бути високими через потребу в більшій кількості тестувальників, довших циклах тестування та повторних зусиль для тих самих тестів. Під час масштабування процесу тестування компанії часто недооцінюють ці витрати.

Ручне тестування займає більше часу, щоб виявити дефекти та повідомити про них. Оскільки виконання повільніше, помилки можуть бути виявлені на пізній стадії циклу розробки, що збільшує зусилля, необхідні для їх налагодження та виправлення. Ця затримка може вплинути на графік випуску та якість програмного забезпечення.

Автоматизоване тестування є важливим етапом у забезпеченні якості програмного забезпечення, що дозволяє значно підвищити ефективність перевірки системи [5].

Автоматизоване тестування є технікою тестування, яка автоматизує процес перевірки функціональності програмного забезпечення та забезпечує

його відповідність вимогам перед випуском у виробництво [6]. Завдяки автоматизованому тестуванню організація може швидше запускати певні тести програмного забезпечення без тестувальників.

Процес впровадження автоматизації тестування складається з кількох ключових етапів:

– вибір інструментів автоматизації: на початковому етапі необхідно підібрати відповідні інструменти для тестування. Наприклад, для тестування web-додатків підходять Selenium або Cypress, а для мобільних додатків – Appium, а для API-тестування – Postman [7]. Вибір інструментів повинен відповідати потребам проєкту та кваліфікації команди;

– розробка архітектури тестового фреймворку, який є основою для автоматизованих тестів. Важливо створити добре структуровану архітектуру, яка включає організацію тестових файлів, механізми обробки тестових даних та підтримку різних тестових середовищ. Це спрощує написання, виконання та підтримку тестів;

– розробка тестових сценаріїв: на цьому етапі створюються тестові скрипти, які перевіряють конкретні функції додатку. Тести повинні бути зрозумілими, добре задокументованими та включати механізми обробки помилок, щоб полегшити аналіз у разі невдалого тесту;

– виконання автоматизованих тестів: автоматизоване виконання тестів дозволяє скоротити час перевірки програмного забезпечення. Важливо правильно налаштувати тестове середовище та забезпечити можливість паралельного запуску тестів для підвищення продуктивності;

– аналіз результатів тестування: автоматизовані тести фіксують відхилення фактичної поведінки системи від очікуваної. Помилки та збої реєструються у вигляді звітів, що дозволяє швидко виявляти дефекти;

– створення тестових звітів: для ефективного аналізу результатів необхідно формувати детальні звіти про тестування. У звітах відображається інформація про перевірені функції, успішно виконані та провалені тести, а також можливі причини збоїв;

– інтеграція з CI/CD: інтеграція тестів у процес безперервної інтеграції та розгортання (CI/CD) дозволяє автоматично виконувати тести при внесенні змін у код. Це забезпечує раннє виявлення помилок і стабільність програмного забезпечення [8];

– підтримка тестів: автоматизовані тести потребують регулярного оновлення відповідно до змін у програмному забезпеченні. Важливо видаляти застарілі тести та виправляти нестабільні сценарії;

– моніторинг якості тестування: контроль за ефективністю тестів допомагає виявити проблеми, такі як часті збої або тривалий час виконання. Регулярний моніторинг дозволяє підтримувати тестову систему на високому рівні якості;

– оновлення фреймворку: автоматизована система тестування повинна розвиватися разом із додатком. Регулярне оновлення фреймворку, додавання нових тестів та покращення продуктивності є важливими складовими довгострокової стратегії тестування [9].

Таким чином, успішна автоматизація тестування є безперервним процесом, який потребує регулярного вдосконалення та адаптації до змін у проєкті. В таблиці 1.1 представлено порівняння ручного та автоматизованого тестування [10].

Таблиця 1.1 – Порівняння ручного та автоматизованого тестування

Критерій	Ручне тестування	Автоматизоване тестування
Точність	Схильне до людських помилок, проте ефективне для складних тестів	Висока точність у повторюваних тестах, але можливі помилки у випадку неправильно написаних скриптів
Вартість	Дешевше для поодиноких або складних тестів	Вигідніше для великих обсягів тестів, проте вимагає початкових інвестицій
Надійність	Добре підходить для дослідницького тестування	Надійне для повторюваних тестів
Охоплення тестів	Гнучке, проте менш ефективне для великих проєктів	Високе охоплення при великій кількості тестових сценаріїв
Час виконання	Потребує більше часу на виконання тестів	Виконується швидко після налаштування
Оцінка UI/UX	Найкраще підходить для оцінки взаємодії користувача	Обмежене в оцінці користувацького досвіду
Потреби в ресурсах	Не потребує програмування, але вимагає досвіду тестувальника	Потребує знань мов програмування (Python, Java, JavaScript)

Ручне тестування підходить для:

– дослідницького тестування, де важливий людський фактор [11];

– тестування UI/UX, де необхідна оцінка зручності використання;

– невеликих або рідко змінюваних проєктів, де автоматизація недоцільна.

Автоматизоване тестування ефективне у:

– регресійному тестуванні, де необхідне повторюване виконання тестів;

– навантажувальному тестуванні, коли потрібно симулювати велику кількість користувачів;

– великих проєктах, де необхідно швидко тестувати великі обсяги коду.

1.2 Selenium як інструмент тестування web-додатків

У сучасній розробці програмного забезпечення автоматизація тестування є ключовим етапом, що дозволяє підвищити якість продукту та скоротити час на тестування. Серед найпопулярніших засобів для автоматизованого тестування виділяють Selenium, TestComplete, Katalon Studio, Cypress та Playwright. Кожен із них має свої переваги та обмеження, що важливо враховувати при виборі інструмента для конкретного проєкту.

Selenium WebDriver є основним компонентом Selenium, який забезпечує взаємодію з web-браузерами [12]. WebDriver виконує сценарії тестування шляхом безпосереднього зв’язку з браузером через інтерфейс програмування додатків (API). WebDriver сумісний із низкою браузерів, включаючи Chrome, Firefox, Safari та Edge, із окремим драйвером для кожного браузера. Підтримує різні мови програмування (Java, Python тощо).

Основні характеристики Selenium:

– Selenium підтримує багато мов програмування: C#, Java, JavaScript, Node.js, Python, PHP, .NET, Groovy, Perl, Ruby, Scala. Сумісний з усіма сучасними операційними системами: Linux, Windows, MacOS. Найпоширеніші браузери, що дозволяють працювати з Selenium – це Google Chrome, Firefox, Safari, Edge, Internet Explorer та навіть деякі браузери без графічного інтерфейсу;

– застосовують Selenium: розробники, тестувальники, фахівці з кібербезпеки, системні адміністратори для автоматизації різноманітних задач у браузері, наприклад web-скрапінгу та парсингу, автоматичного виконання скриптів та сценаріїв;

– використовується Selenium WebDriver найчастіше з такими видами тестування, як регресійне та функціональне;

– Selenium можна інтегрувати з різними інструментами, такими як TestNG, JUnit, Maven, Jenkins тощо.

Автоматизоване тестування за допомогою Selenium забезпечує легке масштабування, щоб охопити широкий спектр тестових випадків, сценаріїв і взаємодій з користувачем [18]. Така масштабованість забезпечує максимальне тестове покриття функціональності програми.

Selenium дозволяє тестувальникам створювати багаторазові тестові сценарії, які можна використовувати в різних тестових випадках і проектах. Така можливість багаторазового використання економить час і зусилля на створення та обслуговування тестового сценарію.

Selenium підтримує паралельне виконання тестів, що дозволяє виконувати кілька тестів одночасно [19]. Це допомагає скоротити загальний час тестування, роблячи процес розробки ефективнішим.

Selenium надає докладні журнали виконання тестів і звіти, що полегшує відстеження результатів тестування та визначення областей, які потребують уваги.

Selenium може моделювати взаємодію та поведінку користувача, дозволяючи тестувальникам оцінити досвід користувача та переконатися, що програма є інтуїтивно зрозумілою та зручною для користувача.

Selenium можна інтегрувати в конвеєри CI/CD для автоматизації тестування кожної зміни коду. Ця інтеграція допомагає виявляти та вирішувати проблеми на ранніх етапах циклу розробки, дозволяючи швидші та надійніші випуски.

TestComplete, як інструмент тестування web-додатків, пропонує зручний інтерфейс для створення автоматизованих тестів як у графічному режимі (без коду), так і за допомогою скриптів. Його сильна сторона – підтримка як веб, так і десктопних та мобільних застосунків. Основним недоліком є висока вартість ліцензії.

TestComplete розроблений компанією SmartBear. Він дозволяє створювати тести для веб, десктопних та мобільних застосунків, що робить його універсальним рішенням для підприємств.

Основні можливості:

– TestComplete дозволяє тестувати застосунки, створені з використанням технологій .NET, Java, WPF, Delphi, HTML5, Android та iOS;

– завдяки зручному запису дій користувача, тестування може виконувати навіть людина без досвіду програмування;

– для досвідчених користувачів доступна можливість написання тестів мовами JavaScript, Python, VBScript, JScript, DelphiScript та C++Script;

– TestComplete легко інтегрується з Jenkins, Azure DevOps, Git, TestRail та іншими популярними інструментами DevOps;

– інструмент забезпечує надійну ідентифікацію елементів інтерфейсу, навіть якщо їхні властивості змінюються;

– можливість запускати тести паралельно на декількох машинах або віртуальних середовищах.

Переваги:

– SmartBear TestComplete пропонує можливості кросплатформної автоматизації, включаючи веб, UNIX та інші системи;

– він забезпечує інтеграцію з Azure DevOps для безперервної інтеграції та розгортання;

– простий в підтримці, через такі функції, як самовідновлення, що зменшує потребу в постійному обслуговуванні;

– TestComplete підтримує різні мови програмування, що полегшує керування проектами з бібліотеками на Python, JavaScript та C#;

– SmartBear TestComplete має потужні можливості звітності та дозволяє створювати та виконувати масштабовані набори тестів.

Недоліки:

– SmartBear TestComplete стикається з проблемами стабільності та періодичними збоями, що впливає на надійність під час використання;

– функції обробки помилок та тестування API потребують значного вдосконалення для покращення управління тестуванням;

– існують проблеми інтеграції, зокрема з Jenkins, Git та іншими інструментами управління тестуванням, що обмежує безперебійність робочих процесів;

– ціна SmartBear TestComplete висока, що створює обмеження для потенційних користувачів, які шукають економічно ефективні варіанти;

– файл журналу TestComplete Executor важко читати, коли він великий, що ускладнює аналіз журналу тестування для користувачів.

TestComplete підходить у випадках, коли:

– потрібно автоматизувати десктопні додатки;

– команда має обмежені навички програмування – можна використовувати тестування на основі запису дій;

– проєкт має достатній бюджет на ліцензії та потребує інтегрованого рішення “все в одному”;

– потрібна офіційна техпідтримка з гарантіями.

TestComplete використовується для створення та автоматизації багатьох різних типів тестів програмного забезпечення. Запис та відтворення тестів записує виконання тестування вручну тестом і дозволяє відтворювати та підтримувати його знову і знову як автоматизований тест. Записані тести можуть бути змінені пізніше тестувальниками для створення нових тестів або покращення існуючих тестів за допомогою додаткових варіантів використання.

Katalon Studio підтримує автоматизацію веб, API, мобільного тестування. Він базується на платформі Selenium і Appium, але надає більш дружній інтерфейс.

Katalon Studio дозволяє тестувати мобільні застосунки (через Appium), RESTful і SOAP API, тестування десктопу (обмежено, через Katalon Platform), зокрема web-додатки (через Selenium).

Katalon Studio дає можливість створювати тести без навичок програмування через запис дій. Єдиною мовою, що підтримує Katalon Studio є Groovy, що є розширенням Java.

Вбудоване середовище для розробки тестів (IDE):

– містить автодоповнення коду, інтегрований перегляд об’єктів (Object Repository), менеджер тест-кейсів і планувальник;

– інтеграція з CI/CD: Підтримка Jenkins, Azure DevOps, Git, Docker, GitLab, Bamboo та інших;

– звітність і аналітика: Генерація звітів, підтримка Katalon TestOps для розширеного моніторингу, дашбордів, логів та історії запусків.

Переваги:

– Katalon містить широкий спектр зручних інтеграцій, тому не потребує додаткових розширень для запуску тестів. Користувачі також можуть легко інтегруватися з іншими платформами управління SDLC (життєвим циклом розробки програмного забезпечення), такими як JIRA, TestRail, qTest та TestLink;

– Katalon Studio також підтримує інтеграцію з платформами для співпраці (Git, Microsoft Teams та Slack) та платформами виконання (SauceLabs, BrowserStack, Selenium Grid та Kobiton);

– вбудований модуль тестування на основі даних дозволяє користувачам аналізувати тести на основі записаних сценаріїв. Katalon відображає аналітичні результати у вигляді вбудованих звітів, які можна експортувати у формати PDF, HTML, Excel або CSV. Звіти візуально інтуїтивно зрозумілі та прості у використанні;

– Katalon підтримує всі популярні браузери та дозволяє запускати автоматизовані тести в Chrome, Firefox, Internet Explorer, Safari, Edge, віддаленому, безголовому та користувацьких середовищах;

– інтерфейс користувача Katalon пропонує насичену графіку з деревоподібним представленням, таблицями та меню, що дозволяє користувачеві легко керувати артефактами тестування;

– Katalon Studio славиться своєю простотою як встановлення, так і використання. Його легко налаштувати завдяки численним інтегрованим пакетам. Він підходить як початківцям, так і досвідченим користувачам, які можуть використовувати всю IDE та писати складні скрипти.

Недоліки:

– Katalon має повнофункціональну безкоштовну версію, що більшість користувачів вважають перевагою. Однак версія Enterprise пропонує більше функцій та доступ до всіх плагінів Katalon і офлайн-ліцензії, тому її рекомендується використовувати в командах і масштабованих проектах, тоді як безкоштовна базова версія пропонується для індивідуального використання;

– на відміну від Selenium та TestComplete, єдиною мовою сценаріїв, яку підтримує Katalon, є Groovy. Ця мова сценаріїв належить до родини Java, тому будь-хто, хто знає Java, може нею користуватися;

– інструмент має закритий вихідний код, що призводить до меншої кількості розробників у спільноті. Selenium, один з головних конкурентів Katalon, – це інструмент з відкритим вихідним кодом, який дозволяє інженерам налаштовувати його або використовувати пакети, створені спільнотою;

– однак Katalon Studio має деякі відкриті компоненти, фреймворк Katalium з відкритим вихідним кодом та магазин Katalon Store з відкритим вихідним кодом та плагінами, які ми описали вище;

– користувачі повідомляють про деякі помилки, які перешкоджають тестуванню, уповільнюючи його. Наприклад, іноді інструмент зависає або може почати лагати, перевірка тексту та об'єктів у iframe є проблематичною. Мобільне тестування займає більше часу через необхідність захоплення та
написання коду.

Cypress – це сучасний фреймворк для тестування web-додатків, орієнтований на JavaScript-розробників. Він працює у браузері, має швидкий

зворотний зв'язок і просту установку. Cypress не підтримує багатобраузерність так само добре, як Selenium, і має обмеження щодо тестування застосунків поза браузером.

Cypress – це сучасний open-source інструмент для автоматизованого тестування web-додатків, створений спеціально для JavaScript-розробників. Його головна особливість – тісна інтеграція з браузером і самим застосунком, що забезпечує швидке, стабільне та зручне тестування.

Ключові особливості Cypress:

– тести виконуються безпосередньо в браузері – це дає можливість отримувати миттєвий зворотний зв’язок та бачити, як кожен тест взаємодіє з DOM;

– автоматичний запис виконання тестів – у вигляді скріншотів або відео, що дуже зручно для відлагодження помилок;

– End-to-End тестування – Cypress дозволяє перевіряти додаток з точки зору користувача, взаємодіючи з UI як справжній браузер;

– підтримка мокінгу, спайів і тест-дублерів – це дозволяє тестувати навіть тоді, коли бекенд ще не готовий;

– автоматичне оновлення при зміні коду – жива перезагрузка тестів під час розробки.

Плюси:

– Cypress.io надає пріоритет простоті та досвіду розробника, пропонуючи інтуїтивно зрозумілий API та зрозумілу документацію, що дозволяє розробникам швидко та ефективно писати тести.

– тести запускаються безпосередньо у браузері, що забезпечує неперевершену швидкість порівняно з традиційними фреймворками для тестування. Ця ефективність має вирішальне значення для підтримки швидких циклів зворотного зв'язку, особливо в конвеєрах безперервної інтеграції.

– Cypress.io може похвалитися вичерпною документацією та активною спільнотою, надаючи ресурси та підтримку розробникам будь-якого рівня знань.

– Cypress.io постачається з вбудованими твердженнями та утилітами, що зменшує потребу в зовнішніх бібліотеках та спрощує процес тестування.

Обмеження та недоліки:

– хоча Cypress.io підтримує кілька браузерів, його основна увага залишається зосереджена на Chrome. Підтримка інших браузерів може бути не такою надійною, що потенційно обмежує можливості кросбраузерного тестування.

– Cypress.io працює в одному вікні браузера, не маючи вбудованої підтримки для тестових сценаріїв, що включають кілька вкладок або вікон, що може бути обмеженням для певних програм.

– опанування розширених функцій Cypress.io, таких як користувацькі команди або інтеграція з іншими інструментами, може вимагати додаткового часу та зусиль через крутішу криву навчання.

– підтримка лише JavaScript/TypeScript – неможливо використовувати інші мови програмування;

– обмежена кросбраузерність – підтримуються лише Chromium-браузери, Firefox та Edge; Safari підтримується обмежено, Internet Explorer – взагалі ні;

– менше гнучкості у взаємодії з елементами DOM на рівні API, ніж у Selenium.

Playwright – відносно новий інструмент від Microsoft, який підтримує тестування у кількох браузерах (включаючи Chrome, Firefox і Safari) та має API, схожий на Cypress. Його переваги – швидкість, підтримка багатьох мов програмування, можливість запису відео виконання тестів. Проте Playwright ще не має такої великої спільноти, як Selenium.

Playwright – це open-source фреймворк для автоматизованого тестування web-додатків, створений розробниками з Microsoft, які раніше працювали над Puppeteer. Він став серйозним конкурентом як Selenium, так і Cypress, поєднуючи у собі високу продуктивність, кросбраузерність та потужні інструменти для написання стабільних тестів.

Ключові можливості Playwright:

– підтримка кількох браузерів:

– Chromium (Chrome, Edge);

– Firefox;

– WebKit (Safari) – на відміну від Cypres;

– кросплатформеність: підтримка Windows, macOS, Linux;

– End-to-End тестування з реалістичною взаємодією – Playwright може імітувати введення з клавіатури, миші, сенсору (тач), геолокацію, камеру, файли тощо;

– підтримка паралельного виконання – вбудована багатопотокова інфраструктура (Test Runner);

– знімки екрану, відео та трасування – автоматичне збереження всіх дій тесту, що спрощує відлагодження;

– імітація мережевих умов – можлива емуляція слабкого інтернету, помилок сервера, заглушки API.

Переваги Playwright:

– одна з найбільших проблем автоматизації тестування – це нестабільні тести, і це особливо впливає на Selenium. У Selenium вам доводиться витрачати значну кількість часу на створення тестів, щоб переконатися, що вони очікують видимості або клікабельності елементів. І навіть якщо вважати, що успішно досягнуто цього під час їх написання, коли вони виконуються в конвеєрі, вони все одно можуть іноді завершуватися невдачею, оскільки налаштовані очікування не завжди працюють в іншому середовищі.

– Playwright вирішує це (здебільшого) за допомогою автоматичних вбудованих очікувань, що спрощує написання тестів та робить їх стабільнішими під час виконання. У Playwright все ще бувають випадки, коли очікування потрібно налаштовувати, але їх набагато менше.

– Playwright підтримує такі мови програмування, як JavaScript, Python, C# та Java, хоча його основний API спочатку був написаний на Node.js.

– під час використання іншої системи автоматизації тестування, можливо, також буде необхідно вибрати, встановити та інтегрувати окремий засіб

виконання тестів, такий як NUnit або MSTest, систему тверджень, таку як Jasmine або Mocha, окремий інструмент звітності, такий як TestNG або Allure, та сторонній інструмент візуального тестування. Playwright має власні вбудовані версії всіх цих інструментів, тому його набагато легше встановити, і вони більш інтегровані.

– однією з головних проблем із Selenium було завантаження та підтримка правильних версій драйверів браузера, особливо забезпечення використання тих самих версій під час створення тестів локально та в конвеєрі CI/CD. Playwright вирішує цю проблему, беручи на себе управління драйверами браузера.

Недоліки:

– більшість фреймворків для автоматизації тестування переходять на асинхронний стиль кодування, тому це зрештою стосуватиметься всіх. На жаль, на мою думку, асинхронний код пропонує мало, якщо взагалі пропонує, переваг для автоматизації тестування та може створювати проблеми для менш досвідчених інженерів з автоматизації тестування. На щастя, IDE забезпечують кращу підтримку асинхронного кодування та все більше допомагають долати проблеми, які воно створює.

– документація, розміщена на офіційному сайті Playwright, є доволі складною для орієнтації та розуміння, особливо для початківців або користувачів, які переходять з ручного тестування. Матеріали на Stack Overflow часто не приносять користі, оскільки здебільшого стосуються застарілих версій Playwright, які суттєво відрізняються від актуальної. У порівнянні з підтримкою Selenium або загальним програмуванням, підтримка Playwright у спільноті виглядає менш ефективною. Очікується, що з часом ці ресурси будуть удосконалені.

– порівняно з Selenium, зокрема, який є простим і зрозумілим фреймворком, Playwright часом може здаватися надто складним, маючи багато способів зробити одне й те саме. Як наслідок, його набагато складніше опанувати. Це також змушує архітекторів тестів впроваджувати дуже круті

складні фреймворки для вашого проекту, які можуть стати кошмаром для всіх інших.

Playwright – чудовий вибір, якщо:

– потрібне кросбраузерне тестування, включаючи Safari;

– проєкт використовує JavaScript, TypeScript, Python або C#;

– команді важлива висока швидкість виконання тестів та гнучкість налаштування середовища;

– потрібно масштабувати тестування на кілька середовищ одночасно (CI/CD, Docker, хмара);

– обхідно поєднати UI, API і функціональні тести в одному фреймворку.

Playwright поєднує гнучкість та швидкість, яку раніше пропонували лише окремі інструменти. Завдяки підтримці WebKit і можливості писати тести різними мовами, Playwright – це вибір багатьох сучасних команд.

В таблиці 1.2 представлено порівняння інструментів тестування web-додатків.

Таблиця 1.2 – Порівняння інструментів тестування web-додатків

Критерій	Selenium	TestComplete	Katalon Studio	Cypress	Playwright
Тип	Open-source	Комерційний	Частково безкоштовний	Open-source	Open-source
Підтримка мов	Java, C#, Python, JS, Ruby тощо	JavaScript, Python, VBScript	Java, Groovy	JavaScript, TypeScript	JavaScript, Python, C#, Java
Підтримка браузерів	Chrome, Firefox, Safari, Edge, Opera	Chrome, Firefox, IE, Edge	Chrome, Firefox, Edge	Chrome, Firefox, Edge (обмежена Safari)	Chrome, Firefox, WebKit (Safari)
Підтримка мобільних додатків	Через Appium	Пряма підтримка	Через Appium	Ні	Частково (експериментально)
Підтримка десктоп-додатків	Ні	Так	Ні	Ні	Ні
Простота у використанні	Середня	Висока (GUI + скрипти)	Висока (інтерфейс)	Висока (для JS-розробників)	Висока
Інтеграція з CI/CD	Так (через Jenkins, GitLab CI т.д)	Так	Так	Так	Так

Кінець таблиці 1.2

Швидкість виконання тестів	Середня	Висока	Середня	Висока	Висока
Спільнота / Підтримка	Дуже велика	Обмежена (платна підтримка)	Середня	Швидко зростає	Зростає
Ціна	Безкоштов-ний	Платний (дорогий)	Частково безкоштовний, Enterprise – платно	Безкоштов ний	Безкоштовний

Хоча сучасні інструменти, такі як Cypress чи Playwright, мають інтуїтивні інтерфейси та кращу продуктивність, Selenium залишається найбільш універсальним і гнучким засобом для автоматизації тестування web-застосунків. Його підтримка багатьох мов програмування, браузерів і операційних систем, а також активна спільнота, роблять його оптимальним вибором для довгострокових та масштабних проєктів. Саме тому в рамках цього дослідження було обрано Selenium як основний інструмент автоматизації.

Висновок до розділу

За результатами проведеного аналізу різних джерел [1-42] було прийнято рішення про доцільність використання автоматизованого тестування функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium, що дозволить скоротити час тестування web-додатка, створить умови для багаторазового запуску тестів, мінімізує вплив людини на процес тестування.

2 РОЗРОБКА ПІДХОДІВ ДО ТЕСТУВАННЯ

2.1 Аналіз фронтенд частини інформаційної системи університету та функціональні вимоги

Єдина інформаційна система ІФНТУНГ Alma Mater була запущена в жовтні 2024 року як інструмент для заповнення індивідуальних планів роботи викладача. При проєктуванні система задумувалася як єдиний агрегатор інформації щодо кадрового забезпечення, навчального та дослідницького процесу, та супутніх процесів в ІФНТУНГ. На даний момент, впроваджено також реєстр шаблонів документації ІФНТУНГ та систему керування відвідуванням басейну, що знаходиться в спорткомплексі ІФНТУНГ. Наступними напрямками розвитку системи планується:

– заповнення інформаційних довідок викладача;

– заповнення та формування звітності про наукову, навчальну, методичну та організаційну діяльність;

– автоматизоване формування рейтингових показників викладачів;

– систематизація інформації про методичне забезпечення навчального процесу.

Інформаційна система ІФНТУНГ реалізується як два незалежні web-додатки – клієнтська та серверна частина, що взаємодіють між собою по API з авторизацією через Bearer Token.

Серверна частина базується на стеку технологій PHP та MySQL. Для систематизації програмного коду був розроблений власний фреймворк, що передбачає використання декількох рівнів:

– рівень моделей – для забезпечення цілісності даних;

– рівень сервісів – для взаємодії з базою даних;

– рівень контролерів – для опрацювання HTTP запитів з клієнтської частини та видачі відповіді;

Крім того, засобами серверної частини здійснюється перевірка авторизації користувача, контроль доступу та маршрутизація запитів по ендпойнтах. Із сторонніх сервісів у серверній частині використовується бібліотека PHPOffice (для читання та запису файлів Microsoft Office).

Клієнтська частина реалізовується засобами технологій HTML, CSS та Javascript. Для реалізації вибрано client-side рендеринг, де вся функціональна архітектура користувацької сторінки будується засобами Javascript. Із сторонніх бібліотек використовується бібліотека vanilla-Datatables (для формування користувацького інтерфесу таблиць) та choices (для реалізації інтерактивного випадаючого списку). Доступ до платформи здійснюється через єдину сторінку, роутинг та переміщення по навігації на якій реалізовано засобами Javascript. Для функціонального розмежування використовується модульний підхід до js-файлів.

Система функціонує з використанням web-сервера Apache для обох компонент – клієнтської та серверної.

Тестований програмний модуль «Індивідуальні плани роботи викладача» реалізовує функціонал із планування та звітування про різні види роботи науково-педагогічними працівниками ІФНТУНГ. Нормативною базою, на якій базується даний програмний модуль, є Положення про планування та облік роботи науково-педагогічних працівників Івано-Франківського національного технічного університету нафти і газу, введене в дію наказом Ректора №266 від 29 вересня 2023 року.

Функціональні вимоги до програмного модуля включають наступні характеристики:

– створення нових індивідуальних планів;

– імпорт навчальної роботи із програми Деканат;

– заповнення планованої навчальної, методичної, та організаційної роботи;

– погодження планованої роботи;

– звітування по виконаній роботі;

– затвердження звіту.

Модуль передбачає наступні ролі:

– викладач – має право створювати план роботи, імпортувати навчальну роботу із файлу експорту ПЗ Деканат, коригувати навчальну роботу, заповнювати плановану методичну, наукову та організаційну роботу, відправляти план на погодження, заповнювати фактичну роботу, відправляти на погодження фактичний план, друкувати план;

– завідувач кафедри – має право погоджувати або відхиляти план та фактичне виконання викладачів кафедри, а також аналогічний викладачу функціонал (як науково-педагогічний працівник;

– директор інституту/декан факультету – має право погоджувати або відхиляти план та фактичне виконання плану викладачами інституту/факультету, фіналізує погодження методичної та організаційної роботи;

– начальник навчального відділу має право погоджувати або відхиляти план та фактичне виконання плану викладачами університету, фіналізує погодження навчальної роботи;

– директор НДІ має право погоджувати або відхиляти план та фактичне виконання плану викладачами університету, фіналізує погодження наукової роботи;

– проректор з наукової роботи та проректор з науково-педагогічної роботи – мають право здійснювати моніторинг та погоджувати або відхиляти план та фактичне виконання плану викладачами університету.

2.2 Вибір стратегії тестування

Для комплексної перевірки системи можна використати комбінований підхід, що включає:

– функціональне тестування, де проводиться перевірка основних сценаріїв роботи (створення плану, завантаження файлів);

– юзабіліті-тестування містить оцінку зручності інтерфейсу для викладачів.

– тестування безпеки перевіряє роботу рівнів доступу та захисту даних;

– інтеграційне тестування важливе у тестуванні взаємодії з підсистемами (електронний документообіг, бази даних).

Функціональне тестування включає перевірку основних сценаріїв роботи системи, таких як створення, редагування та збереження навчальних планів, а також валідацію Excel-файлів для імпорту та експорту даних, щоб забезпечити коректність їх обробки. Окрему увагу приділяється перевірці обмежень доступу для різних ролей користувачів, щоб усі дії виконувалися згідно з наданими правами.

Юзабіліті-тестування спрямоване на оцінку зручності інтерфейсу, зокрема форм введення даних та навігації по системі, щоб користувачі могли працювати без зайвих труднощів. Також перевіряється коректність відображення помилок та наявність зрозумілих підказок, які допомагають уникнути неправильних дій.

Тестування безпеки охоплює перевірку авторизації та контролю доступу, щоб різні групи користувачів мали лише ті права, які їм надані. Додатково проводиться валідація вхідних даних на предмет потенційних загроз, таких як XSS-атаки чи SQL-ін’єкції, що допомагає запобігти вразливостям системи.

Інтеграційне тестування забезпечує коректну взаємодію з Excel, зокрема перевіряє формат файлів та правильність передачі даних, а також синхронізацію з іншими модулями, наприклад, системою обліку навантаження, щоб усі процеси працювали узгоджено.

Далі встановлено критерії тестування, які керують, коли починати та припиняти тестування. Тут встановлено два критичні критерії: вхід та вихід.

Вхідним критерієм тестування є виправлення всіх критичних дефектів.

Вихідним критерієм є успішно пройдені основні сценарії використання.

За результатами аналізу інтерфейсу web-додатку інформаційної системи ІФНТУНГ було сформовано план тестування web-додатку:

– для проведення тестування створення плану:

– користувач може вибрати рік;

– план зберігається у системі;

– редагування аудиторного навантаження:

– можливість завантажити Excel-файл;

– коректний розподіл годин (основна/додаткова ставка);

– редагування неаудиторного навантаження:

– форма валідує введені дані;

– дані зберігаються коректно;

– погодження плану:

– різні ролі мають відповідні права (наприклад, деканат може підтверджувати).

У таблиці 2.1 відображена оцінка ризиків.

Таблиця 2.1 – Оцінка ризиків

Ризик	Мітигація
Некоректний імпорт Excel	Тестувати різні формати файлів
Втрата даних при збереженні	Перевірка backup-механізмів
Доступність для не авторизованих користувачів	Тестування безпеки

Основними компонентами для перевірки є:

– форма входу та авторизація;

– розділ "Мої індивідуальні плани";

– функціонал створення та редагування планів (аудиторне/неаудиторне навантаження);

– імпорт даних з Excel-файлів;

– робота з формами для ручного введення даних;

– процеси погодження планів (для адміністраторів та методистів).

Критерії тестування:

– валідність даних: коректність обробки введених годин, розподіл між основною та додатковою ставкою;

– стабільність: відсутність збоїв під час завантаження файлів або відправки на погодження;

– відповідність вимогам: усі функції повинні працювати згідно з технічним завданням;

– крос-браузерна сумісність: робота в Chrome, Firefox, Edge.

Стратегія дозволить виявити критичні дефекти на ранніх етапах, забезпечити стабільність системи та підвищити рівень довіри користувачів. Усі знайдені помилки будуть документовані та виправлені до релізу.

2.3 Структура пакету тестування та тест-кейсів

Для тестування правильності створення плану аудиторного навантаження в індивідуальному плані роботи викладача сформовано два тест-кейси.

В першому тест-кейсі “Створення плану” виконуються кроки:

– відкрити сторінку створення плану;

– вибрати рік з випадаючого списку;

– вибрати тип плану (наприклад, "Основна ставка");

– натиснути "Зберегти".

Очікуваний результат виконання тест-кейсу “Створення плану”:

– план успішно створюється та зберігається в системі;

– система відображає повідомлення про успішне збереження.

В другому тест-кейсі “Редагування аудиторного навантаження” виконуються кроки:

– створити план (як у Тест-кейсі 1);

– перейти у розділ "Мої індивідуальні плани роботи".

Очікуваний результат виконання тест-кейсу “Редагування аудиторного навантаження”:

– створений план відображається у списку.

Для тестування правильності редагування аудиторного навантаження в індивідуальному плані роботи викладача сформовано два тест-кейси (третій та

четвертий).

В третьому тест-кейсі “Завантаження Excel-файлу” виконуються кроки:

– відкрити сторінку аудиторного навантаження;

– натиснути “Завантажити Excel”;

– вибрати коректний Excel-файл (відповідно до шаблону);

– завантажити файл.

Очікуваний результат виконання тест-кейсу “Завантаження Excel-файлу”:

– система приймає файл та відображає дані;

– немає помилок валідації.

В четвертому тест-кейсі “Перевірка розподілу годин (основна/додаткова ставка)” виконуються кроки:

– завантажити Excel-файл (як у Тест-кейсі 3);

– перевірити, чи правильно розподілено години між ставками.

Очікуваний результат виконання тест-кейсу “Перевірка розподілу годин (основна/додаткова ставка)”:

– години розподілено коректно.

Для тестування правильності редагування неаудиторного навантаження в індивідуальному плані роботи викладача сформовано два тест-кейси (п’ятий та шостий).

В п’ятому тест-кейсі “Валідація введених даних” виконуються кроки:

– відкрити сторінку неаудиторного навантаження;

– ввести некоректні дані (наприклад, літери замість цифр у полі "Години");

– натиснути “Зберегти”.

Очікуваний результат виконання тест-кейсу “Валідація введених даних”:

– система виводить помилку валідації.

В шостому тест-кейсі “Коректне збереження даних” виконуються кроки:

– ввести коректні дані;

– натиснути “Зберегти”;

– перевірити у БД або на сторінці перегляду.

Очікуваний результат виконання тест-кейсу “Коректне збереження даних”:

– дані збережені та відображаються правильно.

На рисунку 2.1 зображено графічне представлення описаних тест-кейсів.

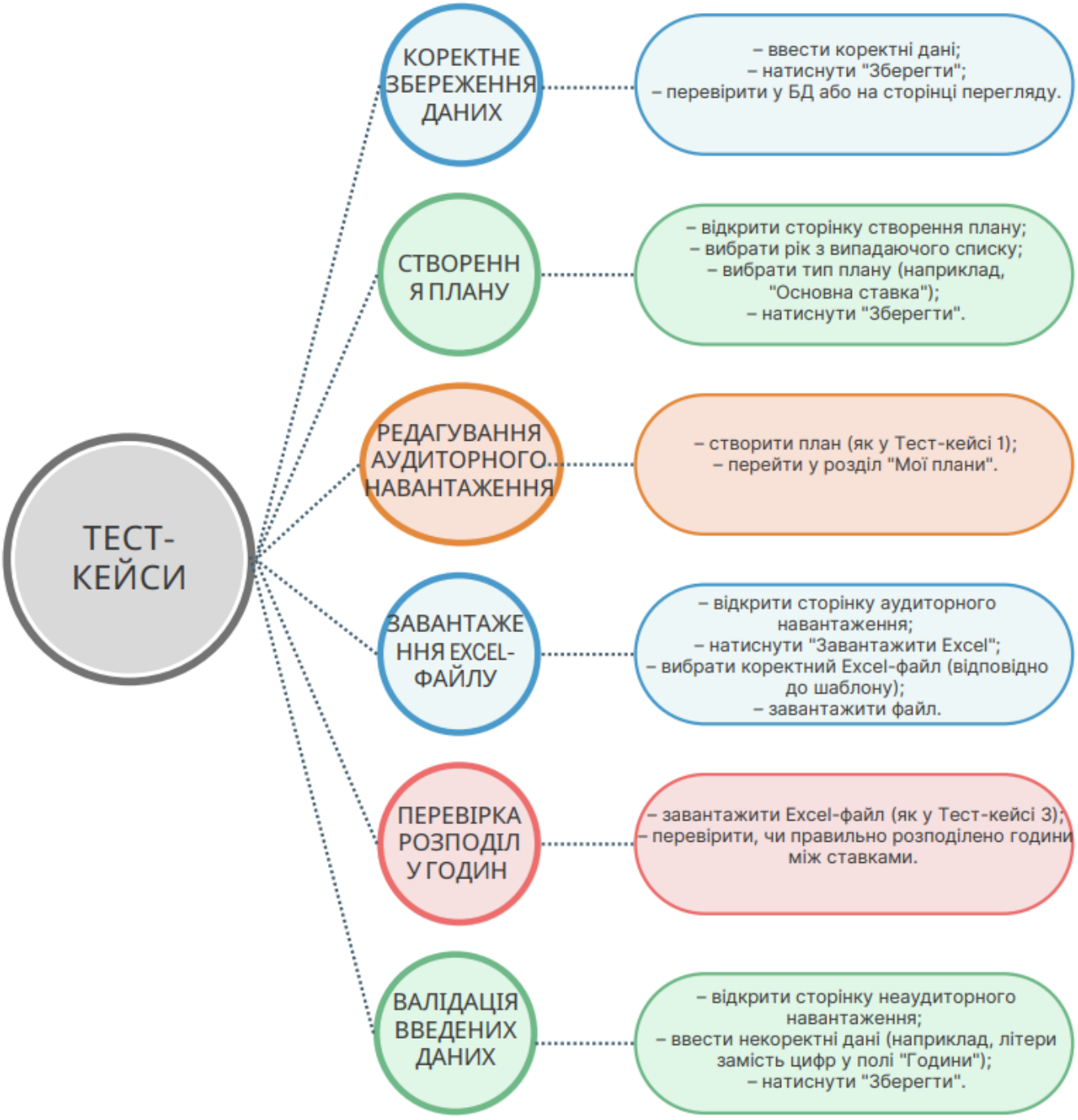

Рисунок 2.1 – Пакет тест-кейсів для автоматизації

На вищепредставленому рисунку можна переглянути 6 тест-кесів, на основі яких було написано автоматизовані тести.

Висновок до розділу

У даному розділі було проведено аналіз фронтенд-частини інформаційної системи університету, визначено функціональні вимоги до неї, обрано стратегію тестування та розроблено структуру пакету тестування з тест-кейсами.

Розроблено тест-кейси для основних сценаріїв: створення та збереження плану аудиторного навантаження, завантаження та обробка Excel-файлів, валідація та збереження даних неаудиторного навантаження.

3 РЕАЛІЗАЦІЯ ПАКЕТУ ТЕСТУВАННЯ

Для проведення тестування web-додатків необхідно попередньо організувати відповідне середовище. У даній роботі буде використовуватися наступний підхід до налаштування:

– встановлюється інтегроване середовище розробки IntelliJ IDEA Community Edition, яке надає зручний інтерфейс для роботи з Java-проектами;

– завантажується та інсталюється Java Development Kit (JDK), необхідний для компіляції та виконання Java-коду. Для коректної роботи JDK налаштовуються змінні середовища, що дозволяє системі виконувати Java-команди з будь-якої директорії;

– створюється новий Maven-проект у IntelliJ, до якого додається остання версія бібліотеки Selenium, яка є основним інструментом для автоматизації тестування web-додатків.

Для взаємодії з браузером Chrome завантажується ChromeDriver – спеціальний драйвер, який забезпечує зв’язок між Selenium та браузером. Шлях до драйвера вказується у коді, що дозволяє автоматично запускати тести у браузері. Фінальним етапом налаштування є перевірка коректності встановлених змінних середовища та запуск тестового сценарію для відкриття web-сторінки.

Таким чином, створюється повноцінне середовище, готове для проведення автоматизованого тестування.

3.1 Розробка пакету автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium

У рамках реалізації функціонального автотестування вебзастосунку було використано фреймворк Selenium WebDriver з підтримкою TestNG для організації життєвого циклу тестів. Було створено тестовий клас SmokeTest, який імітує базовий сценарій використання вебпорталу, що включає вхід до

системи через Google акаунт, вибір ролі, відкриття модуля індивідуального планування, а також створення нового плану. У роботі було застосовано як WebDriver API, так і інструмент Robot, який дозволяє взаємодіяти з елементами, недоступними через DOM (наприклад, системні попапи).

Спочатку здійснено імпорт необхідних бібліотек:

– org.openqa.selenium.* – основний інтерфейс взаємодії з браузером;

– org.openqa.selenium.chrome.ChromeDriver та ChromeOptions – реалізація для браузера Google Chrome;

– WebDriverWait та ExpectedConditions – елементи явного очікування;

– java.awt.Robot – інструмент симуляції натискань клавіш та миші на рівні ОС;

– TestNG анотації @Test, @BeforeClass, @AfterClass – для структурування тестів.

У методі setUpClass() було виконано попередню конфігурацію драйвера Chrome. Зокрема, шлях до ChromeDriver було задано через System.setProperty, після чого створено об’єкт ChromeOptions, до якого додано опцію --disable-features=ChromeBrowserCloudManagement з метою відключення деяких політик керування браузером, що можуть заважати автоматизації. Надалі у методі setup() створюється новий екземпляр ChromeDriver вже безпосередньо у глобальне поле driver.

```
    @BeforeClass
    public void setUpClass() {
        System.setProperty("webdriver.chrome.driver",
"C:\\chromedriver\\chromedriver.exe"); // Replace with correct path
or use WebDriverManager
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--disable-
features=ChromeBrowserCloudManagement");
        WebDriver driver = new ChromeDriver(options);
    }
```

У методі setup() було реалізовано відкриття браузера та перехід на головну сторінку тестованого ресурсу. Було встановлено тайм-аут на неявне очікування (implicitlyWait) тривалістю 10 секунд, а також приведено вікно браузера до повного розміру. Відкриття ресурсу здійснено за URL.

```
    @Test(priority = 1)
    public void setup() {
        driver = new ChromeDriver();

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
        driver.manage().window().maximize();
        driver.get("https://nung.edu.ua/");
    }
```

У методі loginGmail() реалізовано сценарій авторизації через акаунт Google. За допомогою явного очікування (WebDriverWait) було знайдено кнопку входу за ідентифікатором loginButton, після чого здійснено клік.

```
    @Test(priority = 2)
    public void loginGmail() {
        WebDriverWait wait = new WebDriverWait(driver,
    Duration.ofSeconds(15));

        // Клік по кнопці входу
        WebElement loginButton =
    wait.until(ExpectedConditions.elementToBeClickable(By.id("logi
    nButton")));
        loginButton.click();
```

Далі було послідовно введено email та пароль користувача. Поля введення знаходились за допомогою XPath-виразів, що враховують атрибути type або aria-label.

```
    // Введення email
    WebElement emailField =
wait.until(ExpectedConditions.visibilityOfElementLocated(
            By.xpath("//*[contains(@type, 'email')]")
    ));
    emailField.sendKeys("iryna.babchuk-ki211@nung.edu.ua");
```

Кнопка "Далі" також шукалася за вмістом тексту. Введення облікових даних завершувалося переходом до наступної сторінки. Було зазначено, що пароль бажано передавати через параметр, а не в явному вигляді, що є правильною рекомендацією щодо безпеки.

```
        // Клік по кнопці "Далі"
        WebElement nextButtonAfterEmail =
     wait.until(ExpectedConditions.elementToBeClickable(
                By.xpath("//*[contains(text(),'Далі')]")
        ));
        nextButtonAfterEmail.click();

        // Введення пароля
        WebElement passwordField =
     wait.until(ExpectedConditions.visibilityOfElementLocated(
                By.xpath("//*[contains(@aria-label,'Введіть
     пароль')]")
        ));
        passwordField.sendKeys(yourPassword); // краще передавати
     через змінну або параметр

        // Клік по кнопці "Далі" після пароля
        WebElement nextButtonAfterPassword =
     wait.until(ExpectedConditions.elementToBeClickable(
                By.xpath("//*[contains(text(),'Далі')]")
        ));
        nextButtonAfterPassword.click();
     }
```

У методі dismissChromeSystemPopup() було реалізовано обробку системного попапу Chrome, який не входить до складу DOM дерева сторінки. Було спочатку здійснено спробу закрити alert (в разі, якщо він з'являється як JavaScript-спливаюче вікно), з обробкою винятку TimeoutException. Якщо ж такий alert не виявлено, далі застосовано клас Robot, за допомогою якого переміщено курсор миші до визначених координат та здійснено симульоване натискання лівої кнопки миші. Було вказано координати mouseMove(990, 550), що вимагає уточнення залежно від роздільної здатності екрану та розташування попапу.

```
@Test(priority = 3)
public void dismissChromeSystemPopup() throws AWTException,
InterruptedException {
    try {
        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(5));
        Alert alert =
wait.until(ExpectedConditions.alertIsPresent());
        alert.dismiss(); // натискає "Скасувати"
    } catch (TimeoutException e) {
        System.out.println("Попап не з'явився або це не
alert.");
    }
    Robot robot = new Robot();
    Thread.sleep(3000); // Дати час на появу вікна

    robot.mouseMove(990, 550);
    robot.mousePress(InputEvent.BUTTON1_DOWN_MASK);
    robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK);
}
```

Метод chooseRole() відповідає за вибір ролі користувача після входу в систему. Здійснюється пошук елемента за id='selectRoleLink' та клік по ньому.

Після цього очікується доступність кнопки з data-id='25', яка символізує конкретну роль, та виконується її активація.

```
@Test(priority = 4)
public void chooseRole() {
    WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));

    // Клік на кнопку вибору ролі
    WebElement chooseRole =
wait.until(ExpectedConditions.elementToBeClickable(
            By.xpath("//a[id='selectRoleLink']")
    ));
    chooseRole.click();

    // Очікуємо появу кнопки з конкретною роллю (data-id='25')
    WebElement roleButton =
wait.until(ExpectedConditions.elementToBeClickable(
            By.xpath("//a[@data-id='25']")
    ));
    roleButton.click();
}
```

У методі openIndividualPlans() виконується перехід до розділу "Мої індивідуальні плани роботи" через клік по відповідному посиланню, яке ідентифікується за частковим збігом тексту у тегу a.

```
@Test(priority = 5)
public void openIndividualPlans() {
    WebElement plansButton =
driver.findElement(By.xpath("//a[contains(text(),'Мої
індивідуальні плани роботи')]"));
    plansButton.click();
```

```
        System.out.println("Opened 'Your Individual Plans'");
    }
```

У методі createIndividualPlan() реалізується створення нового індивідуального плану.

```
    @Test(priority = 6)
    public void createIndividualPlan() {
        WebElement createPlanBtn =
    driver.findElement(By.xpath("//a[@data-id='createPlan']"));
        createPlanBtn.click();
    }
```

Після натискання на кнопку створення плану, здійснюється заповнення полів заголовка (id="planTitle") та опису (id="planDescription"), а також збереження через кнопку saveButton.

На завершення, метод closeDriver() із позначкою @AfterClass забезпечує закриття браузера після завершення всіх тестів, звільняючи ресурси.

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

У даній реалізації набору тестів було виконано автоматизоване модульне тестування основного користувацького сценарію входу до системи через Google акаунт, вибору ролі, відкриття модуля індивідуальних планів та створення нового плану. Окрему увагу було приділено обробці системного попапу, що виникає під час авторизації. Попри деякі технічні неточності (наприклад, повторна ініціалізація драйвера та помилка в XPath), структура тестів є логічно

завершеною, а підхід – наближеним до промислових стандартів тестування вебзастосунків.

3.2 Запуск тестів та аналіз результатів

Для проведення комплексного тестування було розроблено детальний план, що охоплює основні функціональні сценарії роботи системи. Тестування виконано з метою перевірки коректності роботи ключових модулів та виявлення потенційних дефектів.

Виконано перевірку процесу створення навчального плану. Протестовано функціонал вибору року. Підтверджено, що система коректно зберігає створений план у базі даних та відображає його у відповідному розділі інтерфейсу. Додатково перевірено обробку помилок при спробі зберегти план без обов'язкових полей (рис.3.1).

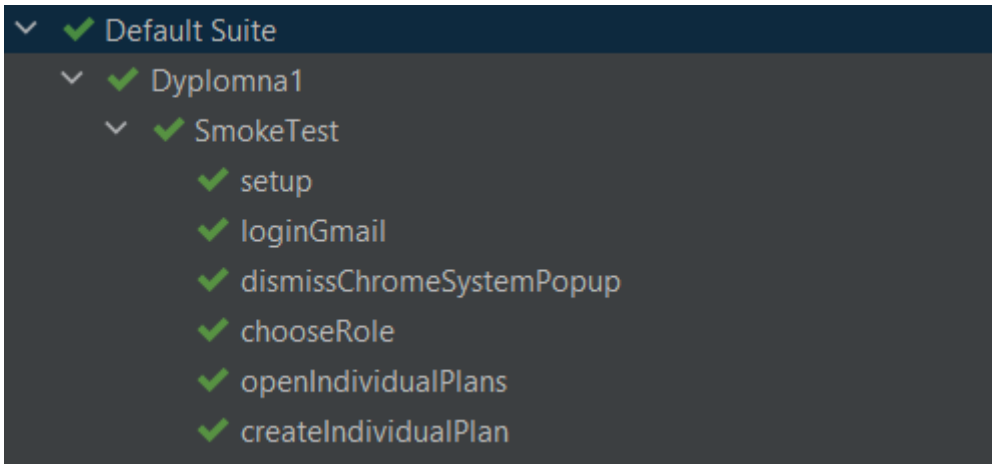

Рисунок 3.1 – Створення плану

Реалізовано перевірку завантаження Excel-файлів з даними аудиторного навантаження. Протестовано різні формати файлів, включаючи коректні та пошкоджені (рис.3.2). Підтверджено, що система правильно розпізнає та імпортує дані, а також коректно розподіляє години між основною та додатковою ставками.

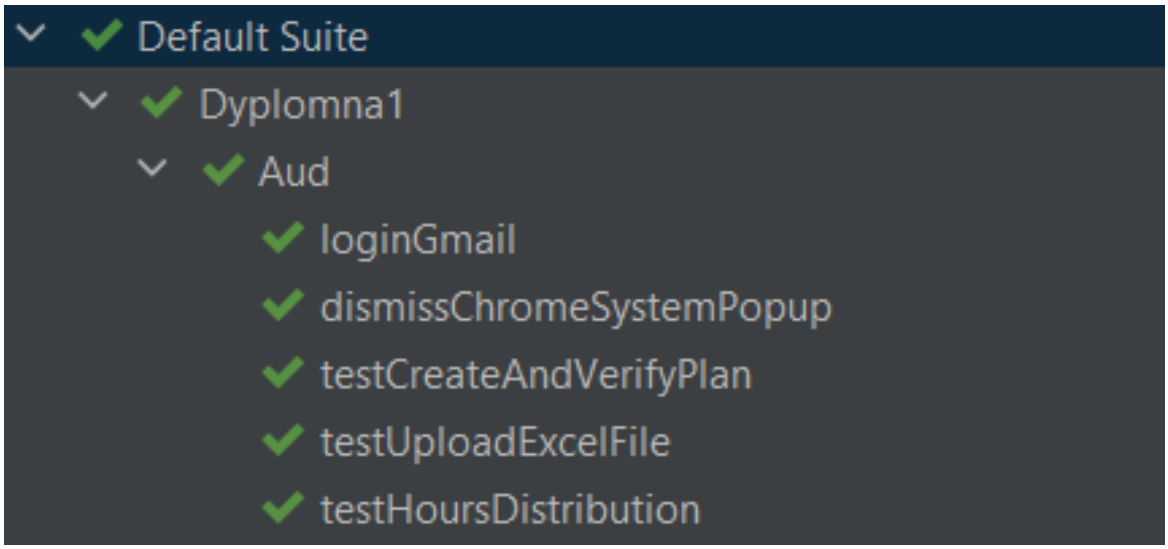

Рисунок 3.2 – Редагування аудиторного навантаження

Виконано перевірку роботи форми введення неаудиторного навантаження. Протестовано валідацію різних типів даних, включаючи коректні та некоректні значення (рис.3.3). Підтверджено, що система правильно перевіряє вхідні дані та блокує збереження при виявленні помилок. Перевірено коректність збереження даних у системі після успішного заповнення форми.

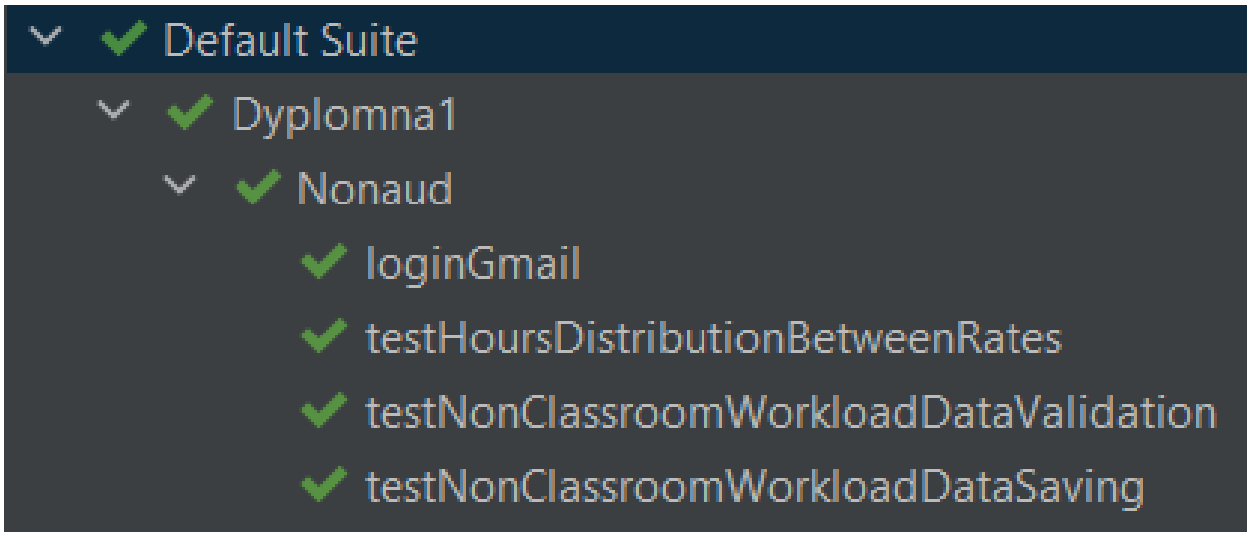

Рисунок 3.3 – Редагування неаудиторного навантаження

Реалізовано перевірку роботи механізму погодження планів для різних ролей користувачів. Результат перевірки зображено на рисунку 3.4.

Протестовано доступність функціоналу підтвердження для деканату та обмеження прав для інших ролей.

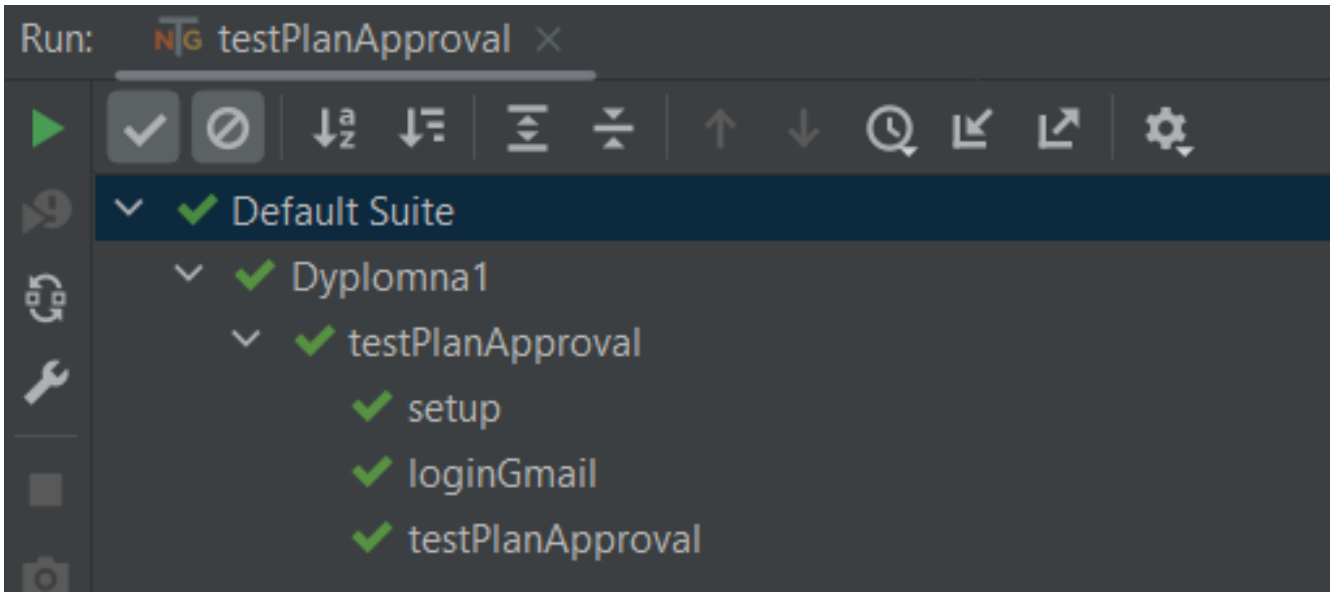

Рисунок 3.4 – Процес погодження плану

З рисунка 3.4 видно, що система правильно обмежує доступ до функцій погодження відповідно до ролей користувачів.

3.3 Оцінка результатів тестування

Після завершення комплексного тестування було проведено детальний аналіз отриманих результатів з метою оцінки якості роботи системи та виявлення напрямів для її оптимізації.

На основі проведених тестів було підтверджено стабільну роботу основних модулів системи. Зокрема, процес створення та збереження навчального плану виконано без критичних помилок, проте механізми валідації даних не завжди забезпечили коректне внесення інформації. Також у модулі редагування аудиторного навантаження виявлено незначні затримки при імпорті великих Excel-файлів, що може вказувати на необхідність оптимізації алгоритмів обробки даних.

Тестування неаудиторного навантаження показало не повну ефективність системи у виявленні некоректних даних, оскільки було зафіксовано окремі випадки, коли помилкові значення приймаються системою, як допустимі на рисунку 3.5. Це може спричинити спотворення інформації у заповненому плані.

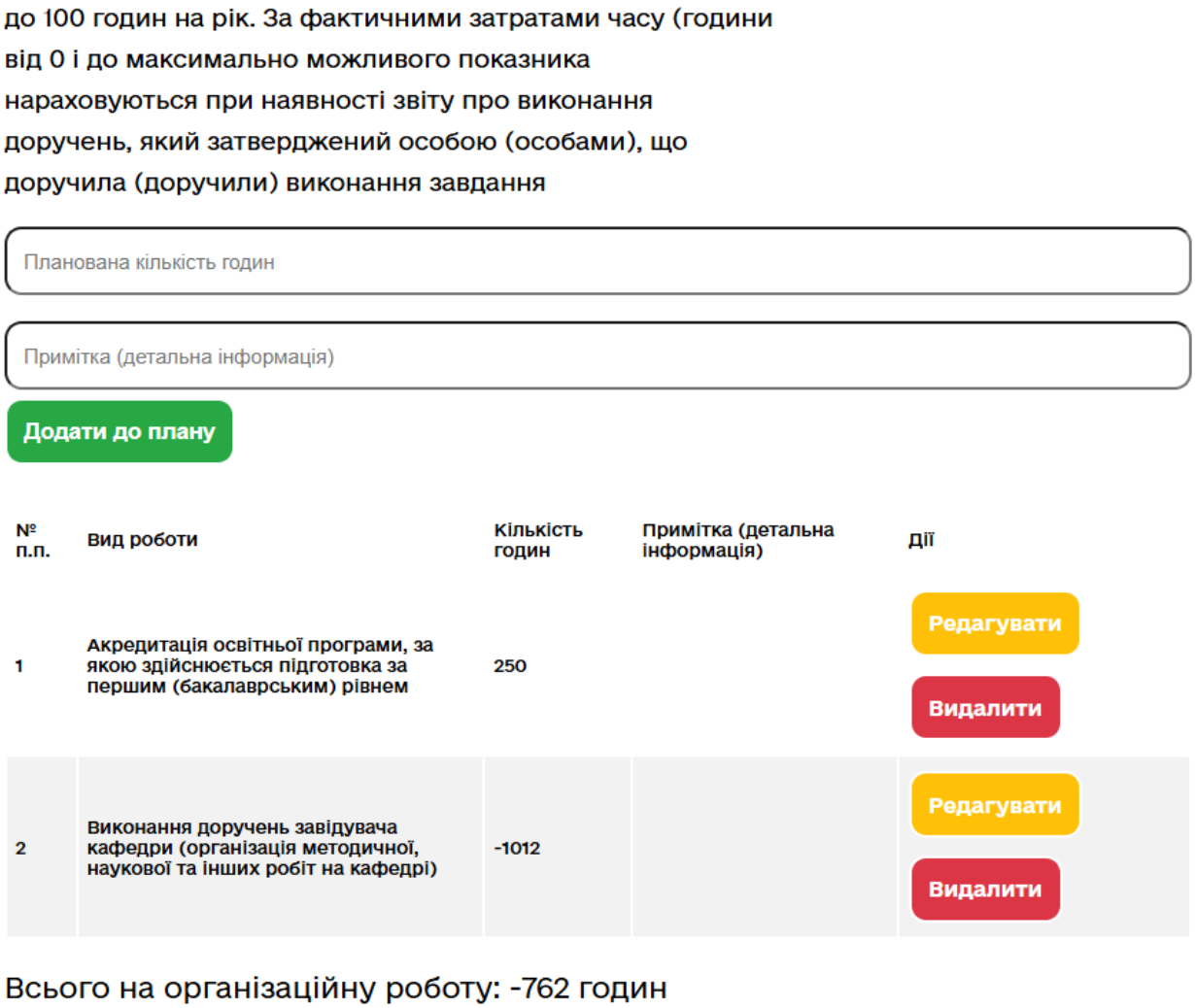

Рисунок 3.5 – Помилка введення годин відведених на наукову роботу

Наступна помилка полягає в тому, що web-додаток під час заповнення фактичного навантаження допускає недопустимі значення при перевірці коректності формату введених годин.

Якщо користувач вводить години у неправильному форматі (наприклад, десяткові числа з помилками, негативні значення тощо), то система зберігає ці некоректні значення. Тобто дані потрапляють у систему, як на рисунку 3.6.

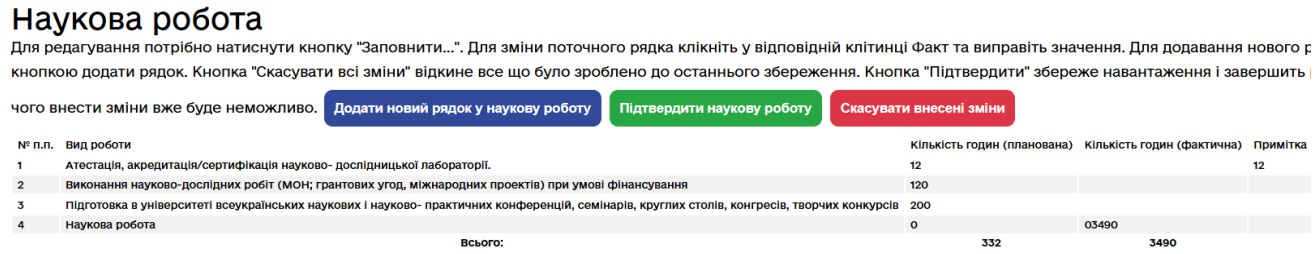

Рисунок 3.6 – Помилка введення фактичних годин відведених на наукову роботу

У випадку на рисунку 3.7, коли помилкове значення не приймається, форма не повідомляє користувача, що сталося – немає жодного повідомлення про помилку або підсвічування поля.

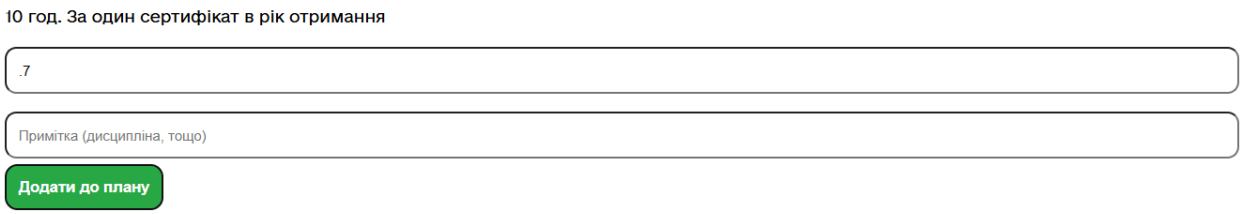

Рисунок 3.7 – Форма не повідомляє про помилкове значення

Також після того як користувач натискає кнопку “Відправити неаудиторне навантаження на погодження”, сайт переходить на сторінку “Індивідуальні плани роботи НПП”. Але коли натиснути кнопку «Назад», замість повернення на попередню головну сторінку, користувач залишається на тій самій сторінці (рис.3.8) – тобто кнопка фактично не виконує свою функцію переходу назад.

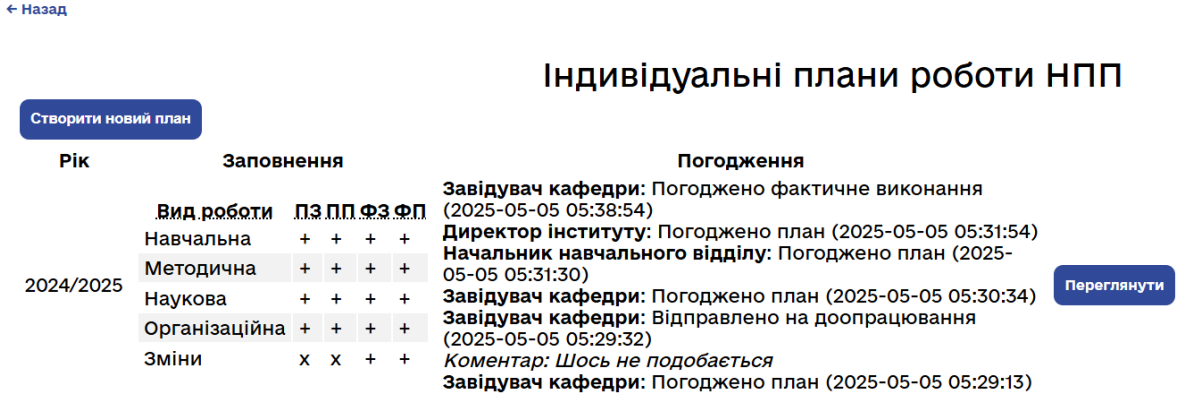

Рисунок 3.8 – Відображення попередньої сторінки

Помилка, що виникає після додавання фактичних годин, проявляється у вигляді напису на рисунку 3.9: “Обсяг наукового навантаження: 333 годин (підтверджено NaN)”. Це свідчить про те, що система не змогла правильно обробити введені дані і замість очікуваного числового значення відображає технічне повідомлення.

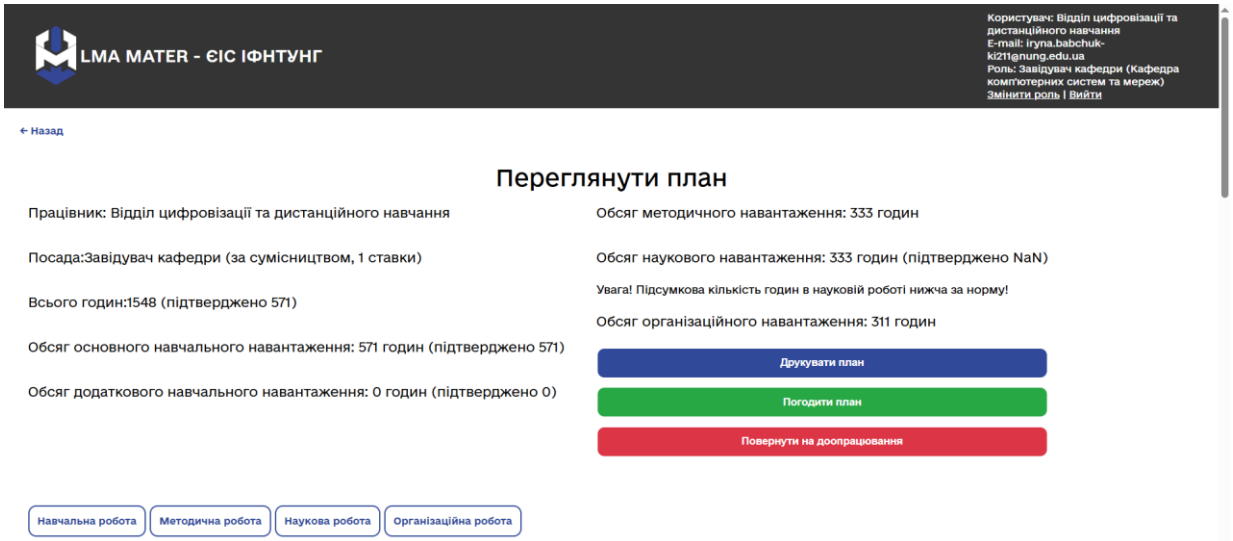

Рисунок 3.9 – Додавання фактичного навантаження

При фіксуванні фактичного навантаження промт web-додатку дозволяє ввести довільні дані (рис.3.10) і не повідомляє про помилку, як на рисунку 3.10.

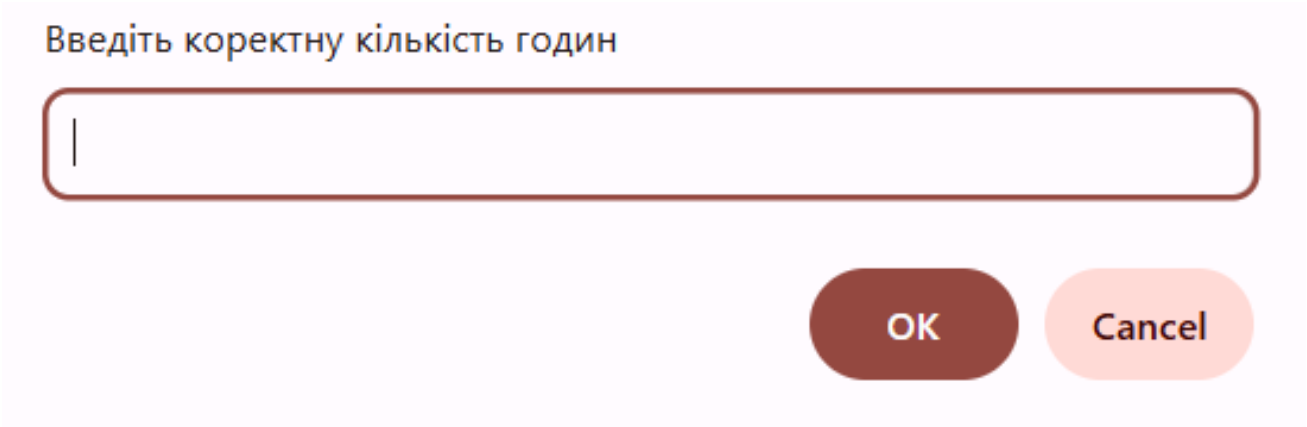

Рисунок 3.10 – Промт для введення коректної кількості годин фактичного навантаження

Сторінка на рисунку 3.11 після внесення неправильних даних не змінилася та не реагує на неправильно внесену інформацію попередньо.

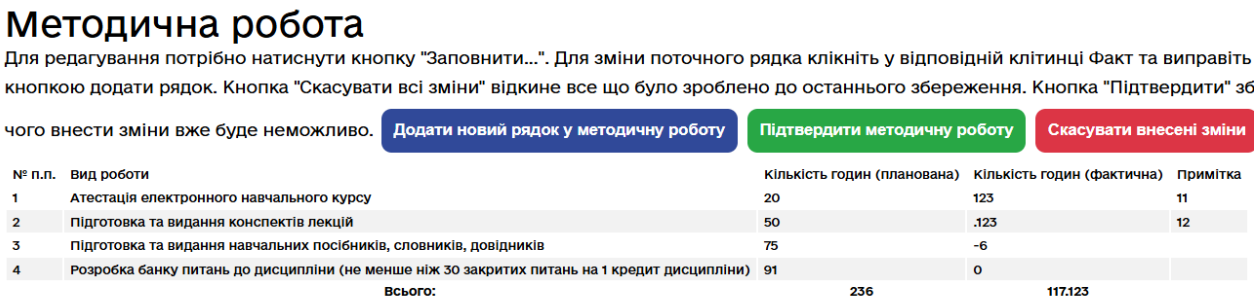

Рисунок 3.11 – Додавання фактичного навантаження

Під час тестування інтерфейсу системи було виявлено, що деякі елементи керування (наприклад, кнопки підтвердження або відміни дій) потребують більш чіткого візуального розмежування. Для підвищення зручності роботи з системою запропоновано:

– оптимізувати швидкодію;

– вдосконалити перевірку вхідних даних;

– покращити візуальну індикацію помилок.

Щодо безпеки, система коректно обмежує доступ відповідно до ролей користувачів, але для підвищення рівня захисту рекомендовано додати двофакторну аутентифікацію для адміністративних облікових записів.

На основі проведеного аналізу сформовано наступні рекомендації:

– оптимізація швидкодії: провести ревізію коду для усунення “вузьких” місць у роботі з даними;

– покращення інтерфейсу: додати інтерактивні підказки;

– розширення тестування: включити додаткові сценарії роботи для покриття крайових випадків;

– оновлення документації: оновити інструкції для користувачів з урахуванням виявлених особливостей роботи.

Результати роботи дають змогу сформувати підхід до тестування системи в цілому і масштабувати розроблені тест-кейси та стратегію для тестування всіх модулів клієнтської частини інформаційної системи університету.

Висновок до розділу

У цьому розділі було відображено організацію процесу тестування функціонування web-додатку інформаційної системи ІФНТУНГ, починаючи з налаштування тестового середовища та реалізації тестових сценаріїв і закінчуючи запуском тестів, аналізом отриманих результатів та їх оцінкою.

На етапі організації тестового середовища тестування функціонування web-додатку інформаційної системи ІФНТУНГ було визначено необхідні інструменти та конфігурації, що дозволяють імітувати робочу систему.

Проведено тестування критичних функцій системи, таких як створення навчального плану, завантаження даних через Excel-файли та валідація форм. Кожен сценарій включав як позитивні, так і негативні кейси, що дозволило перевірити не лише стандартну поведінку системи, а й її реакцію на некоректні вхідні дані.

Запуск тестів та аналіз результатів показав, наскільки важливою є автоматизація процесу тестування для швидкого виявлення дефектів.

На завершальному етапі оцінки результатів тестування було проаналізовано знайдені помилки, їх критичність та вплив на роботу системи. На основі отриманих даних сформульовано рекомендації щодо покращення якості коду та запобігання подібним проблемам у майбутньому.

ВИСНОВКИ

Під час виконання бакалаврської роботи було розроблено пакет автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium

В першому розділі проведено порівняльний аналіз методик ручного та автоматизованого тестування web-додатків. В результаті проведеного аналізу було обрано Selenium як інструмент тестування web-додатків.

В другому розділі виконано аналіз фронтенд частини інформаційної системи університету та функціональних вимог до неї. Також було вибрано стратегії тестування та структуру пакету тестування та тест-кейсів.

В третьому розділі організовано середовище для тестування, розроблено пакет автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium, запущено тестування та здійснено аналіз його результатів.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Try QA. Посібник із сертифікації ISTQB Foundation Level і Agile Tester. Київ, 2023. URL: https://tryqa.com/istqb-certification-the-definitive-guide/ (дата звернення: 05.05.2025).
2. Якість та тестування інформаційних систем. Навч. посібник. Київ : ННІІТ ДУТ, 2020. – 128 с.
3. ISTQB. Foundations of Software Testing. 2022. URL: https://www.utcluj.ro/media/page_document/78/Foundations%20of%20software%20testing%20-%20ISTQB%20Certification.pdf (дата звернення: 05.05.2025).
4. ISTQB. Standard Glossary of Terms Used in Software Testing. 2023. URL: https://glossary.istqb.org/ (дата звернення: 05.05.2025)
5. SmartBear. What is Automated Testing? 2024. URL: https://smartbear.com/learn/automated-testing/what-is-automated-testing/ (дата звернення: 05.05.2025).
6. ISTQB. Термін: Test Automation. 2023. URL: https://glossary.istqb.org/en_US/term/test-automation (дата звернення: 05.05.2025).
7. Redstone. Автоматизація тестування web-додатків: інструменти та стратегії. Київ, 2023. URL: https://redstone.agency/blog/avtomatyzatsiia-testuvannia-veb-dodatkiv-instrumenty-ta-stratehii/ (дата звернення: 05.05.2025).
8. JUTI Journal. Comparison of Automation Testing Tools. 2022. URL: http://juti.if.its.ac.id/index.php/juti/article/view/1021/464 (дата звернення: 05.05.2025).
9. Preprints. Automation Testing in Software Engineering. 2024. URL: file:///C:/Users/38099/Downloads/preprints202404.0911.v1.pdf (дата звернення: 05.05.2025).
10. TestRail. Manual vs. Automated Testing. 2024. URL: https://www.testrail.com/blog/manual-vs-automated-testing/ (дата звернення: 05.05.2025).

11. Graham D., Black R., Veenendal E. Foundations of Software Testing: ISTQB Certification. – 4th ed. – United Kingdom : EMEA, 2018. – 273 p.

12. QATestLab. Selenium WebDriver. 2024. URL: https://training.qatestlab.com/blog/technical-articles/selenium-webdriver/ (дата звернення: 05.05.2025).

13. BrowserStack. Selenium Testing. 2024. URL: https://www.browserstack.com/selenium (дата звернення: 05.05.2025).

14. Конференції ВНТУ. Автоматизоване Тестування: Вступ до Selenium. 2022. URL: https://conferences.vntu.edu.ua (дата звернення: 05.05.2025).

15. IOPScience. Use of Automation Testing in Web Applications. 2020. URL: https://iopscience.iop.org/article/10.1088/1757-899X/801/1/012136/pdf (дата звернення: 05.05.2025).

16. BrowserStack. Selenium RC: Differences from Webdriver. 2024. URL: https://www.browserstack.com/ (дата звернення: 05.05.2025).

17. KR Labs. Працюємо з Selenium для автоматизації задач у web-браузері. 2023. URL: https://kr-labs.com.ua/blog/pratsyuyemo-z-selenium-dlya-avtomatyzatsiyi-zadach-u-veb-brauzeri/ (дата звернення: 05.05.2025).

18. Apidog. Selenium Automation Testing Tool Review. 2024. URL: https://apidog.com/blog/selenium-automation-testing-tool-review/ (дата звернення: 05.05.2025).

19. GlobalLogic. Автоматизація тестування: як уникнути поширених помилок. 2023. URL: https://www.globallogic.com/ua/insights/blogs/qa-automation-2/ (дата звернення: 05.04.2025).

20. JavaTpoint. Selenium WebDriver. 2024. URL: https://www.javatpoint.com/selenium-webdriver (дата звернення: 05.04.2025).

21. ResearchGate. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. 2021. URL: https://www.researchgate.net/publication/300080121_The_Impacts_of_Test_Automation_on_Software%27s_Cost_Quality_and_Time_to_Market (дата звернення: 05.04.2025).

22. QA Group. Що таке Selenium? 2023. URL: https://qagroup.com.ua/publications/shcho-take-selenium (дата звернення: 05.05.2025).

23. Wikipedia. Selenium. 2024. URL: https://uk.wikipedia.org/wiki/Selenium (дата звернення: 05.05.2025).

24. Global Market Insights. Software Testing Market Size By Component. 2023. URL: https://www.gminsights.com/industry-analysis/software-testing-market (дата звернення: 05.04.2025).

25. Satisfice. Testing and Checking Refined. 2022. URL: https://www.satisfice.com/blog/archives/856 (дата звернення: 05.04.2025).

26. Джумаєв С. Розробка технології перевірки на уразливості web-ресурсів. – 2023. – 150 с.

27. Gapon A. O., Fedorchenko V. M. Методи та засоби статичного та динамічного аналізу коду. – К. : Радіотехніка, 2023. – 20 с.

28. He T., Li Z. A model and method of information system security risk assessment based on MITRE ATT&CK. – IEEE, 2021. – 6 с.

29. Сайківська Л. Ф., Копиця А. В. Можливість використання метрик для оцінювання надійності інформаційних систем. – Київ : КНУКіМ, 2018. – С. 46-47.

30. Blancaflor E. et al. Vulnerability Assessment on Cross-site scripting attack in a simulated E-commerce platform using BeEF and XSStrike. – IEEE, 2023. – С. 1-6.

31. Baklizi M. et al. A technical review of SQL injection tools and methods: a case study of SQLMap. – International Journal of Intelligent Systems and Applications in Engineering, 2022. – Т. 10, № 3. – С. 75-85.

32. Ashari I. F. A. et al. Security Audit for Vulnerability Detection and Mitigation of UPT Integrated Laboratory (ILab) ITERA Website Based on OWASP Zed Attack Proxy (ZAP). – Jurnal JTIK, 2023. – Т. 7, № 1. – С. 24-34.

33. Karaçay L. et al. Guarding 6G use cases: a deep dive into AI/ML threats in AllSenses meeting. – Annals of Telecommunications, 2024. – С. 1-15.

34. Zhang L. et al. A risk-level assessment system based on the STRIDE/DREAD model for digital data marketplaces. – International Journal of Information Security, 2022. – С. 1-17.

35. ISTQB. Термін: Black-box Testing. 2023. URL: https://glossary.istqb.org/en_US/term/black-box-testing (дата звернення: 05.05.2025).

36. ISTQB. Термін: White-box Testing. 2023. URL: https://glossary.istqb.org/en_US/term/white-box-testing (дата звернення: 05.05.2025).

37. Milosev V. Educational Softwares in the Function of Improving Education of Children with Special Needs. 2021. URL: https://www.researchgate.net/... (дата звернення: 05.05.2025).

38. Medium. Client-side Rendering (CSR) vs. Server-side Rendering (SSR) in Automation Testing. 2023. URL: https://medium.com/@merisstupar11/client-side-rendering-csr-vs-server-side-rendering-ssr-in-automation-testing-e5390b4e3f13 (дата звернення: 05.05.2025).

39. Dynamic Yield. Client-side Testing and Personalization. 2024. URL: https://www.dynamicyield.com/lesson/client-side-testing-and-personalization/ (дата звернення: 05.05.2025).

40. Medium. Client-side Testing Tools. 2023. URL: https://medium.com/design-bootcamp/client-side-testing-tools-4b3027ba845c (дата звернення: 05.05.2025).

41. DW. Причини аварії Boeing 737 MAX в Індонезії. 2023. URL: https://www.dw.com/uk/оприлюднено-причини-аварії-літака-boeing-737-max-в-індонезії/a50988595 (дата звернення: 05.04.2025).

42. Google Scholar. Тестування client-side rendered додатків. 2022. URL: https://scholar.googleusercontent.com/scholar?q=cache:IiYSuLlh4agJ... (дата звернення: 05.05.2025).

ДОДАТКИ

ДОДАТОК А

Лістинг SmokeTest.java

```
import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

import java.awt.*;
import java.awt.event.InputEvent;
import java.time.Duration;

public class SmokeTest {

    public static String readPasswordFromFile(String filePath)
throws IOException {
        return
Files.readAllLines(Paths.get(filePath)).get(0).trim();
    }

    WebDriver driver;
    String yourPassword ="";
    String path = "password.txt"; // Задай шлях до файлу з паролем
```

```
    @BeforeClass
    public void setUpClass() {
        System.setProperty("webdriver.chrome.driver",
"C:\\chromedriver\\chromedriver.exe"); // Replace with correct path
or use WebDriverManager
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--disable-
features=ChromeBrowserCloudManagement");


        try {
            yourPassword = readPasswordFromFile(path);
        } catch (IOException e) {
            System.err.println("Помилка читання пароля: " +
e.getMessage());
        }


    }


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


    @Test(priority = 1)
    public void setup() {
        driver = new ChromeDriver();

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
        driver.manage().window().maximize();
        driver.get("https://happywife.pp.ua/");
        System.out.println("Opened browser and navigated to site");
```

```
    }

    @Test(priority = 2)
    public void loginGmail() {
        WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(15));

        // Клік по кнопці входу
        WebElement loginButton =
wait.until(ExpectedConditions.elementToBeClickable(By.id("loginButt
on")));
        loginButton.click();

        // Введення email
        WebElement emailField =
wait.until(ExpectedConditions.visibilityOfElementLocated(
                By.xpath("//*[contains(@type, 'email')]")
        ));
        emailField.sendKeys("iryna.babchuk-ki211@nung.edu.ua");

        // Клік по кнопці "Далі"
        WebElement nextButtonAfterEmail =
wait.until(ExpectedConditions.elementToBeClickable(
                By.xpath("//*[contains(text(),'Далі')]")
        ));
        nextButtonAfterEmail.click();

        // Введення пароля
        WebElement passwordField =
wait.until(ExpectedConditions.visibilityOfElementLocated(
                By.xpath("//*[contains(@aria-label,'Введіть
пароль')]")
```

```
        ));
        passwordField.sendKeys(yourPassword); // краще передавати через змінну або параметр


        // Клік по кнопці "Далі" після пароля
        WebElement nextButtonAfterPassword = wait.until(ExpectedConditions.elementToBeClickable(
                By.xpath("//*[contains(text(),'Далі')]")
        ));
        nextButtonAfterPassword.click();
    }


    @Test(priority = 3)
    public void dismissChromeSystemPopup() throws AWTException, InterruptedException {
        try {
            WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(5));
            Alert alert = wait.until(ExpectedConditions.alertIsPresent());
            alert.dismiss(); // натискає "Скасувати"
        } catch (TimeoutException e) {
            System.out.println("Попап не з'явився або це не alert.");
        }
        Robot robot = new Robot();
        Thread.sleep(3000); // Дати час на появу вікна


        robot.mouseMove(990, 550);
        robot.mousePress(InputEvent.BUTTON1_DOWN_MASK);
        robot.mouseRelease(InputEvent.BUTTON1_DOWN_MASK);
```

```
        System.out.println("Скасовано додавання акаунта через системний попап Chrome.");
    }

    @Test(priority = 4)
    public void chooseRole() {
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(20));

        // Клік на кнопку вибору ролі
        WebElement chooseRole = wait.until(ExpectedConditions.elementToBeClickable(
                By.xpath("//a[@id='selectRoleLink']")
        ));
        chooseRole.click();

        // Очікуємо появу кнопки з конкретною роллю (data-id='25')
        WebElement roleButton = wait.until(ExpectedConditions.elementToBeClickable(
                By.xpath("//a[@data-id='25']")
        ));
        roleButton.click();
    }

    @Test(priority = 5)
    public void openIndividualPlans() {
        WebElement plansButton = driver.findElement(By.xpath("//a[contains(text(),'Мої індивідуальні плани роботи')]"));
        plansButton.click();
        System.out.println("Opened 'Your Individual Plans'");
```

```
    }

    @Test(priority = 6)
    public void createIndividualPlan() {

        WebElement createPlanBtn =
driver.findElement(By.xpath("//a[@data-id='createPlan']"));
        createPlanBtn.click();
        System.out.println("Created new individual plan");
    }
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: Розробка пакету автоматичних тестів функціонування web-додатку інформаційної системи ІФНТУНГ засобами Selenium

Обсяг пояснювальної записки 53 аркушів:
3 таблиці;
13 рисунків;
1 додаток.

Дата завершення роботи: *12 червня 2025р*.

Підпис студента- _____*Бабчук І.С.*