МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Використання технології Selenium для автоматизації тестування Веб-сайту**

Виконав: студент 4 курсу, групи 6.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
Д. М. Харченко
(ініціали та прізвище)
Керівник к.ф.-м.н., доцент, доцент кафедри ЕІС та ПЗ, Г.П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус» Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т. В. Критська
"_01_" __березня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Харченку Данилу Михайловичу
(прізвище, ім'я, по батькові)

1.Тема роботи Використання технології Selenium для автоматизації тестування Веб-сайту
керівник роботи Коломоєць Геннадій Павлович, к.ф.-м.н. доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _29.12.2022 № 1893-с

2. Строк подання студентом кваліфікаційної роботи 14.06.2023
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження фреймворку Selenium та його функціоналу;
- огляд технології Selenium IDE та застосунку, який буде тестуватись;
- створення тестів та їх опис.

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

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	01.03 – 10.03.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.03 – 12.03.23	виконано
3	Аналіз існуючих методів рішення	13.03 – 14.03.23	виконано
4	Дослідження засобів тестування Веб-застосунків	15.03 – 20.03.23	виконано
5	Дослідження технології Selenium IDE та її функціональних можливостей	21.03 – 26.03.23	виконано
6	Вибір Веб-застосунку, який буде тестуватись	27.03 – 28.03.23	виконано
7	Огляд фреймворку Selenium, дослідження архітектури та функціональних можливостей	29.03 – 13.04.23	виконано
8	Розгортання оточення розробки для виконання тестів	14.04 – 16.04.23	виконано
9	Опис функціоналу Веб-застосунку	17.04 – 19.04.23	виконано
10	Розробка тестових сценаріїв	20.04 – 01.05.23	виконано
11	Запис тестових сценаріїв та експортування їх до JUnit 5 тестів	02.05 – 7.05.23	виконано
12	Рефакторинг JUnit 5 тестів	8.05 – 12.05.23	виконано
13	Оформлення звіту	13.05 – 14.06.23	виконано

Студент _____ Д.М. Харченко
(підпис) (прізвище та ініціали)

Керівник роботи _____ Г.П. Коломоєць
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок – 69,

Рисунків – 29,

Джерел – 8,

Таблиць – 5.

Харченко Д. М. Використання технології Selenium для автоматизації тестування Веб-сайту: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник Г. П. Коломоєць. Запоріжжя : ЗНУ, 2023. 69 с.

Для сучасних Веб-застосунків якість і надійність є пріоритетними, тому їх тестування є обов'язковим етапом життєвого циклу розробки програмного забезпечення. У роботі проведено аналіз основних підходів до тестування Веб-застосунків. В результаті порівняльного дослідження було виявлено, що фреймворк Selenium є одним з найбільш ефективних інструментів для автоматизації тестування Веб-застосунків.

Основна мета роботи полягає у дослідженні можливостей та переваг, які надає фреймворк Selenium та його інструменти при розробці тестів для Веб-застосунків.

Для тестування був обраний застосунок «Депозитний калькулятор» – це Веб-застосунок, який планувати доходи від розміщення депозитів у банках, задаючи різні параметри депозитів. Розроблено тести для Веб-застосунку, які тестують таку функціональність: розрахунок доходів від депозиту з отриманням доходу в кінці терміну депозиту та при капіталізації процентів щомісячно або щоквартально, збереження розрахунків у файлах різних форматів та виведення їх на друк.

Ключові слова: *Selenium, Selenium IDE, Selenium WebDriver, JUnit 5, Веб-застосунок, тестовий сценарій*.

ABSTRACT

Pages – 69,

Figures – 29,

Sources – 8,

Tables – 5.

Kharchenko D. M. Use of Selenium technology for Website automation testing: bachelor's thesis in speciality 121 "Software engineering" / supervisor H. P. Kolomoets. Zaporizhzhia: ZNU, 2023. 69 p.

For modern Web applications, quality and reliability are a priority, so their testing is a mandatory stage of the software development life cycle. This paper analyzes the main approaches to testing Web applications. As a result of a comparative study, it was found that the Selenium framework is one of the most effective tools for automating the testing of Web applications.

The main goal of the work is to investigate the possibilities and advantages provided by the Selenium framework and its tools in the development of tests for Web applications.

The application "Deposit Calculator" was chosen for testing – it is a Web application that plans income from placing deposits in banks, setting various parameters of deposits. Tests have been developed for the Web application, which test the following functionality: calculation of income from a deposit with receipt of income at the end of the term of the deposit and capitalization of interest monthly or quarterly, saving calculations in files of various formats and printing them out.

Keywords: Selenium, Selenium IDE, Selenium WebDriver, JUnit 5, Web application, test case.

ЗМІСТ

ВСТУП

Актуальність теми

Актуальність теми "Використання технологій Selenium для автоматизації тестування Веб-сайту" базується на зростаючій важливості тестування Веб-застосунків та потребі в ефективних та надійних інструментах для автоматизації цього процесу. Нижче розглянемо докладніше актуальність даної теми:

- Веб-розробка продовжує займати провідну позицію в індустрії програмного забезпечення. Зараз існує безліч Веб-сайтів та Веб-застосунків, що постійно розширюються та оновлюються. Забезпечення якості цих Веб-застосунків вимагає ефективного тестування, а використання технологій Selenium може сприяти автоматизації та поліпшенню процесу тестування.

- Сучасні Веб-застосунки стають все більш складними, з великою кількістю функцій та взаємодій з користувачем. Це створює виклик для тестувальників, оскільки вони повинні переконатися, що всі компоненти працюють належним чином. Використання Selenium дозволяє автоматизувати тестування різних аспектів Веб-застосунків, включаючи перевірку функціональності, навігації, взаємодії з формами та інше.

- Ручне тестування Веб-застосунків може бути часо- та ресурсозатратним процесом. Використання технологій Selenium дозволяє автоматизувати багато рутинних тестових сценаріїв, що допомагає значно збільшити швидкість та ефективність тестування. Відповідно, це дозволяє знизити час та зусилля, витрачені на виконання тестів, і забезпечує більш швидку поставку Веб-застосунків.

- Веб-застосунки повинні працювати на різних платформах та браузерах. Це вимагає тестування на різних конфігураціях, що може бути складним і трудомістким. Selenium надає можливість тестувати Веб-застосунки на різних платформах та браузерах, забезпечуючи високу переносимість тестових сценаріїв.

- Автоматизація тестування з використанням Selenium допомагає забезпечити високу якість та надійність Веб-застосунків. Автоматизовані тестові сценарії можуть бути повторно використані, що дозволяє швидко виявляти та виправляти помилки. Крім того, Selenium надає можливість проводити тестування на різних навантаженнях та умовах, що допомагає перевірити надійність та стабільність Веб-застосунків.

Отже, використання технологій Selenium для автоматизації тестування Веб-сайтів є актуальним завданням, що допомагає забезпечити ефективність, надійність та якість Веб-застосунків, зменшити зусилля та витрати на тестування, а також підтримувати швидку поставку Веб-застосунків.

Мета дослідження

Вивчення та аналіз можливостей, переваг та обмежень, пов'язаних з використанням Selenium для автоматизації тестування Веб-застосунків.

Завдання дослідження

Детально дослідити принципи та методології автоматизації тестування Веб-сайтів, включаючи основні поняття, підходи та інструменти, що використовуються.

Об'єкт дослідження

Вивчення основних компонентів Selenium, зокрема Selenium WebDriver та Selenium IDE. Дослідити їх функціонал, можливості та обмеження.

Предмет дослідження

Аналіз різних методів тестування, які можна застосовувати з використанням Selenium, включаючи методики класів еквівалентності та граничних значень.

Методи дослідження

Спостереження та аналіз: Проведення спостережень над процесом автоматизованого тестування Веб-сайтів з використанням Selenium. Аналіз результатів тестування, виявлення проблем, оцінка продуктивності та ефективності Selenium як інструменту для автоматизації тестування.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, щоб визначити, наскільки ефективно можна використовувати технологію Selenium для тестування Веб-застосунків. Розроблені тестові скрипти, їх аналіз та результати демонструють, які конкретні переваги та можливості можуть бути використані розробниками та тестувальниками для автоматизованого тестування.

Глосарій

Selenium – відкритий фреймворк для автоматизованого тестування Веб-застосунків. Включає інструменти для запису, редагування та відтворення тестових сценаріїв.

Автоматизація тестування – процес використання програмних засобів для автоматичного виконання тестів замість їх ручного виконання. Забезпечує ефективність, повторюваність та точність тестування.

Веб-сайт – колекція Веб-сторінок, пов'язаних між собою гіперпосиланнями. Використовується для представлення інформації, надання послуг та взаємодії з користувачами через Інтернет.

Тестовий сценарій – послідовність кроків, що описують поведінку об'єкта або системи, що тестується. Використовується для перевірки правильності функціонування програмного продукту.

Інтерфейс користувача (UI) – засіб взаємодії користувача з програмним продуктом через графічний чи текстовий інтерфейс.

Локатор – унікальний ідентифікатор елемента на Веб-сторінці, який використовується для його знаходження та взаємодії з ним під час тестування.

Selenium WebDriver – інструмент фреймворку Selenium, що надає можливість програмно взаємодіяти з Веб-елементами і керувати браузером для автоматизованого тестування.

Перехоплення елементів – процес отримання інформації про Веб-елемент на сторінці, такої як текст, значення атрибутів, розташування тощо, за допомогою Selenium.

Assertion – технологія перевірки очікуваного результату під час виконання тесту. Використовується для підтвердження, що фактичні результати співпадають з очікуваними.

Тестовий набір – колекція тестових сценаріїв, які виконуються разом як єдиний блок для перевірки повного функціонування системи або окремої її частини.

XPath – мова запитів, що використовується для вибору елементів на Веб-сторінці за допомогою шляхів до них в дереві DOM.

Запис тестових сценаріїв – процес створення тестових сценаріїв, за якими будуть виконуватись автоматизовані тести. Може включати запис дій користувача, введення даних та перевірку результатів.

JUnit - фреймворк для тестування Java-програм, який надає можливості для створення, виконання та перевірки тестів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

Технологія Selenium є потужним інструментом для автоматизації тестування Веб-сайтів. Вона надає зручний набір інструментів, які дозволяють розробникам і тестувальникам ефективно створювати та виконувати автоматизовані тестові сценарії. Ось деякі ключові аспекти використання Selenium для автоматизації тестування Веб-сайтів [1]:

Selenium WebDriver: Selenium WebDriver є основною складовою технології Selenium. Він дозволяє вам взаємодіяти з елементами Веб-сторінки, такими як кнопки, поля вводу, посилання тощо. За допомогою WebDriver ви можете виконувати різні дії, такі як клікання, введення тексту, наведення курсору, отримання даних тощо. WebDriver підтримує різні мови програмування, включаючи Java, Python, C#, Ruby і багато інших.

Selenium IDE: Selenium IDE – це інструмент для запису та відтворення дій на Веб-сторінці. Ви можете використовувати його для створення автоматизованих тестових сценаріїв без програмування. Записані тестові сценарії можна виконувати безпосередньо у браузері, а також експортувати у різні мови програмування для подальшого редагування та розширення.

Вибір елементів: Selenium надає потужні методи для вибору елементів на Веб-сторінці. Ви можете використовувати різні стратегії пошуку, такі як за ідентифікатором елемента, за класом, за CSS селектором, за XPath тощо. Це дозволяє точно визначати та взаємодіяти з потрібними елементами на сторінці під час виконання тестів.

Перевірка результатів: Selenium дозволяє здійснювати перевірку результатів тестування, що дозволяє автоматично порівнювати фактичні результати з очікуваними. Ви можете перевіряти текст, значення атрибутів, наявність елементів тощо. Це надає можливість автоматично виявляти помилки та проблеми на Веб-сайті.

Масштабованість: Selenium підтримує розподілене виконання тестів за допомогою Selenium Grid. За допомогою Selenium Grid ви можете запускати тести на різних браузерах та платформах паралельно, що дозволяє економити час та збільшувати покриття тестування.

Інтеграція зі засобами розробки: Selenium може бути легко інтегрований з іншими засобами розробки, такими як фреймворки тестування, засоби збирання інформації про покриття тестів, системи управління конфігурацією тощо. Це дозволяє вам створювати потужні тестові набори та використовувати Selenium в контексті вашого проекту.

Це лише загальний огляд використання технології Selenium для автоматизації тестування Веб-сайтів. Враховуючи те , що успішне використання Selenium вимагає глибоких знань технології, програмування та Веб-розробки.

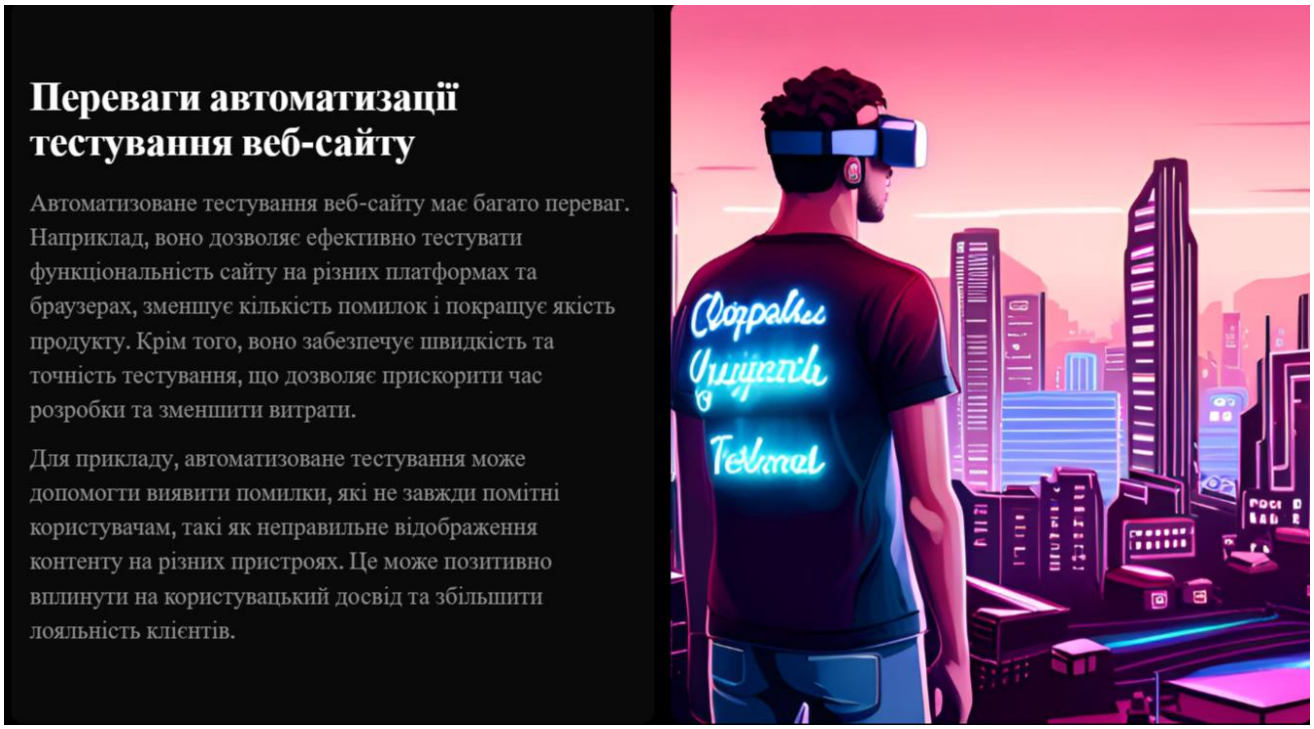


Рис. *1 Перевага автоматизації тестування Веб-сайту*

Автоматизація тестування Веб-сайтів (Рис. 1) має чисельні переваги, які сприяють покращенню ефективності тестування, забезпеченню високої якості продукту та зниженню витрат часу та ресурсів. Основні переваги автоматизації тестування Веб-сайту включають [2]:

Повторюваність: Автоматизовані тестові сценарії можуть бути легко повторені безпосередньо із зазначеними вхідними даними та очікуваними результатами. Це дозволяє ефективно перевіряти функціональність Веб-сайту при кожній зміні коду або впровадженні нових функцій.

Ефективність: Автоматизовані тестові сценарії виконуються швидше, ніж ручні тести, що забезпечує економію часу тестувальників. Операції, які вимагають багато часу, такі як заповнення форм або натискання кнопок, можуть бути автоматизовані для прискорення процесу тестування.

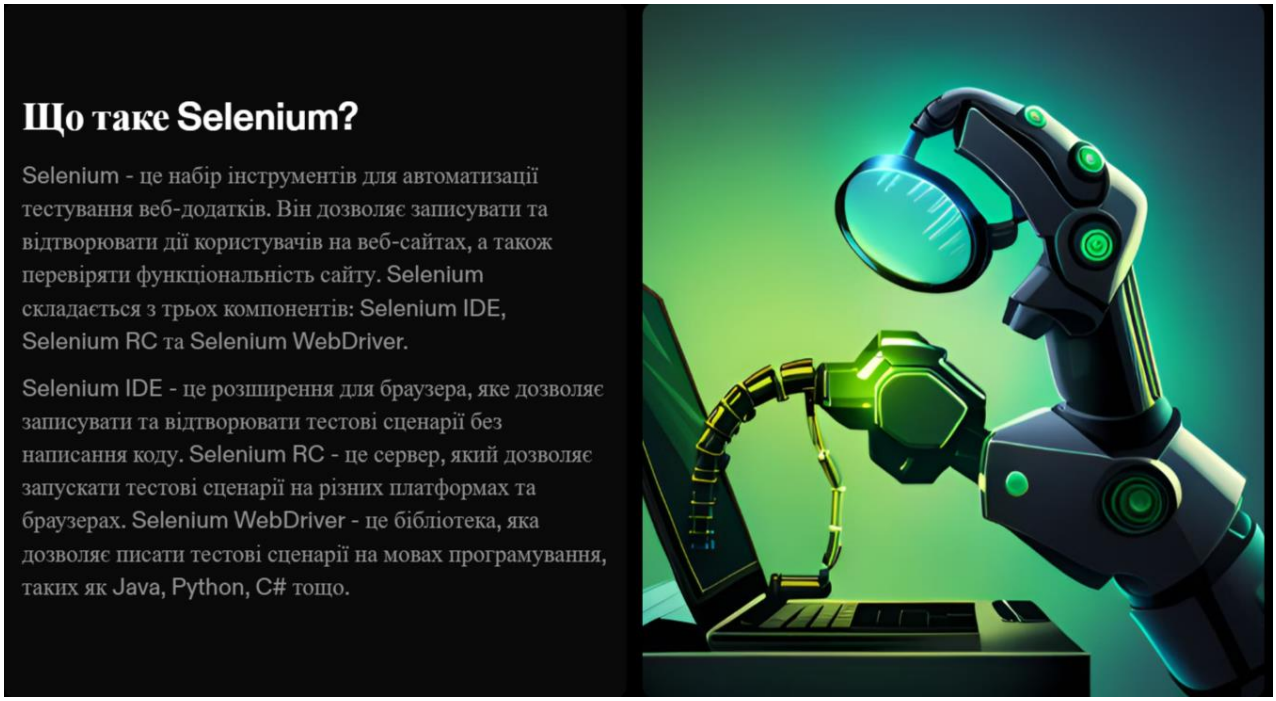


Рис. *2 Технологія Selenium*

Selenium (Рис. 2) також інтегрується з різними інструментами тестування та фреймворками, що дозволяє розширити його функціональність та зручність використання. Загалом, Selenium є потужним інструментом для автоматизації тестування Веб-сайтів, який допомагає забезпечити якість продукту, прискорити процес тестування і знизити його витрати.

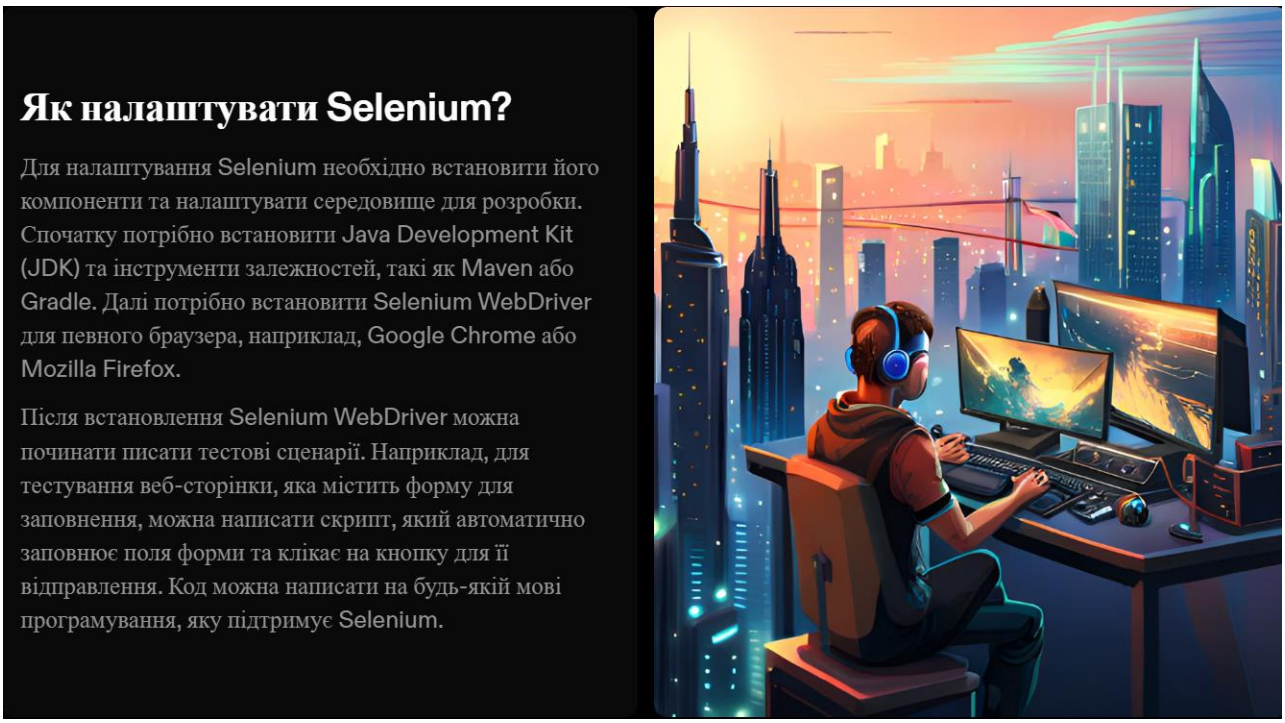


Рис. *3 Налаштування Selenium*

Для налаштування Selenium (Рис. 3) потрібно виконати наступні кроки [3]:

1. Встановіть Java Development Kit (JDK): Переконайтеся, що на вашому комп'ютері встановлений JDK. Ви можете завантажити його з офіційного Веб-сайту Oracle та встановити, дотримуючись інструкцій.

2. Завантажте Selenium WebDriver: Перейдіть на офіційний Веб-сайт Selenium (https://www.selenium.dev/downloads/) і завантажте відповідну версію Selenium WebDriver для вашої мови програмування та браузера.

3. Встановіть WebDriver для вашого браузера: Для використання Selenium з певним браузером вам потрібно завантажити відповідний драйвер браузера. Наприклад, для використання з Google Chrome потрібно завантажити ChromeDriver, для Firefox - GeckoDriver тощо. Завантажте відповідний драйвер з офіційного Веб-сайту браузера і додайте його до системного шляху або вказуйте шлях до нього під час виконання тестів.

4. Налаштуйте середовище розробки: Залежно від вибраної мови програмування, вам потрібно налаштувати середовище розробки, таке як Eclipse, IntelliJ IDEA або інше. Створіть новий проект і налаштуйте залежності до Selenium WebDriver.

5. Почніть використовувати Selenium: Імпортуйте необхідні пакети Selenium у вашому коді і почніть використовувати його функціональність для автоматизації тестування Веб-сайту. Ви можете здійснювати дії, такі як натискання кнопок, заповнення форм, перехід по сторінкам, отримання тексту та багато іншого.

Рис. *4 Тестування за допомогою Selenium*

Selenium дозволяє автоматизувати тестування різноманітних елементів і функціональності Веб-сайту (Рис. 4). Деякі з основних компонентів, які можна тестувати за допомогою Selenium, включають [3]:

1. Навігацію по сторінкам: Ви можете переходити з однієї сторінки на іншу та перевіряти правильність переходу.

2. Взаємодію з елементами Веб-сторінки: Ви можете взаємодіяти з елементами, такими як кнопки, текстові поля, чекбокси, радіокнопки та інші, шляхом заповнення форм, вибору елементів та виконання різних дій на сторінці.

3. Перевірку вмісту сторінки: Ви можете перевіряти наявність, текст, стилізацію та інші атрибути елементів на сторінці.

4. Перевірку функціональності: Ви можете перевіряти роботу різних функцій Веб-сайту, таких як авторизація, додавання товарів у кошик, розміщення замовлення тощо.

5. Перевірку поведінки на різних браузерах: Ви можете перевіряти, як Веб-сайт відображається та працює на різних браузерах, таких як Chrome, Firefox, Safari, Edge тощо.

6. Перевірку відповідності вимогам: Ви можете перевіряти, чи відповідає Веб-сайт вимогам, які були встановлені для нього, наприклад, перевіряти, чи відображаються всі необхідні елементи, чи працюють всі функції тощо.

Це лише кілька прикладів того, що можна тестувати за допомогою Selenium. Із допомогою цієї технології ви можете автоматизувати різноманітні сценарії тестування Веб-сайтів для забезпечення якості продукту та поліпшення користувацького досвіду.

Використання Selenium для автоматизації тестування Веб-сайту має значний вплив на ефективність, якість і надійність тестування. Зробивши огляд та аналізуючи переваги, можна зробити такі висновки:

- За допомогою Selenium можна автоматизувати велику кількість тест-кейсів і скриптів, що значно прискорює процес тестування та зменшує час, витрачений на ручне тестування.

- Автоматизовані тести, створені з використанням Selenium, забезпечують більшу точність та надійність, оскільки вони не піддаються помилкам, пов'язаним з людським фактором.

- Selenium забезпечує широкі можливості для взаємодії з різними елементами Веб-сторінок, такими як кнопки, форми, таблиці, що дозволяє ефективно тестувати різноманітні функції та взаємодіяти з ними.

- Багатофункціональний характер Selenium дозволяє тестувати Веб-сайти на різних платформах та браузерах, забезпечуючи сумісність та переносимість тестів.

- Використання Selenium дозволяє автоматизувати як позитивні, так і негативні тестові сценарії, зокрема за допомогою методик класів еквівалентності та граничних значень, що сприяє виявленню більш широкого спектру дефектів.

- Застосування Selenium сприяє покращенню продуктивності команди тестувальників, оскільки вони можуть фокусуватися на складніших тестових сценаріях та аналізі результатів, в той час як рутинні та повторювані завдання автоматизуються.

1.2 Постановка завдання

Метою кваліфікаційної роботи є розробка автоматизованих тестів для Веб-застосунку за допомогою технології Selenium.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Проаналізувати сучасні підходи до тестування Веб-застосунків та ознайомитись з основними проблемами, з якими стикаються розробники при реалізації тестування.
2. Виконати дослідження технології Selenium: вивчити принципи роботи та можливості технології, ознайомитись з документацією.
3. Обрати Веб-застосунок, який буде тестуватись за допомогою Selenium.
4. Розгорнути та налаштувати середовище розробки тестів.
5. Реалізувати автоматизоване тестування Веб-застосунку з використанням технології Selenium, використовуючи розроблену методологію та набір тестових сценаріїв.
6. Написати пояснювальну записку з описанням, використаних технологій та інструментів, обраного для тестування застосунку, процесу розробки тестових сценаріїв, аналізом результатів тестування та висновками і рекомендаціями щодо подальшого вдосконалення мобільного застосунку.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКІВ

2.1 Огляд технології Selenium

Якщо говорити простими словами то для того, 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*** [4].

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

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

Локатор – це тип Веб-елементу, який потрібно знайти та над яким виконуватиме дії Веб-драйвер.

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

Нижче наведено список найбільш поширених типів локаторів та їх синтаксис:

- By.id – пошук елемента по атрибуту id;
- By.name – пошук елемента по атрибуту name;
- By.className – пошук елемента по назві класу, використовується коли на Веб-сторінці відображається набір однотипних елементів;
- By.TagName – пошук елемента по атрибуту назви тега;
- By.LinkText – пошук елемента по конкретному тексту, якщо він точно відомий;
- By.PartialLinkText – пошук елемента по частковому збігу текста;

- 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.

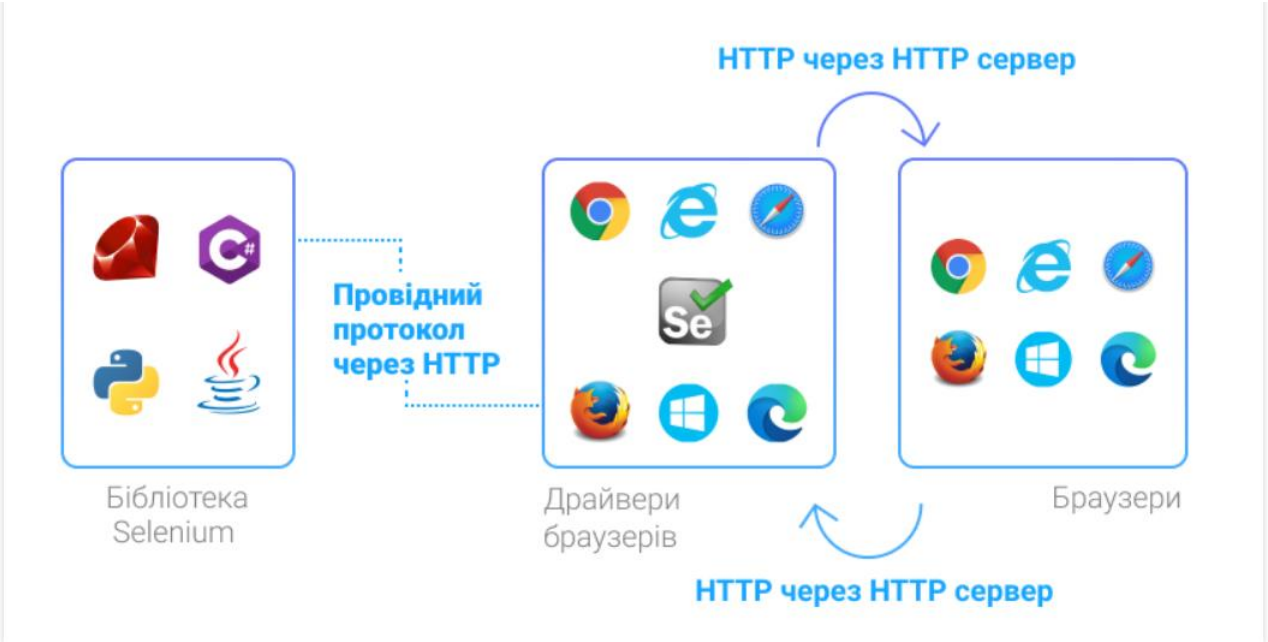


Рис. 5 *Архітектура Selenium*

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.

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.

Робота з Selenium WebDriver

Для початку роботи з Веб-драйвером попередньо потрібно встановити JDK 8 (Java Development Kit) і вище, якщо автоматизація буде здійснюватися на мові Java та будь-яку програму для програмної реалізації. Прикладом може бути IntelliJ IDEA – це інтегроване середовище розробки ПЗ, яке є доволі зручним при роботі з Веб-драйвером та є популярним інструментом в програмуванні. За допомогою нього можна налаштовувати та формувати зручні звіти по тестуванню та збір статистики.

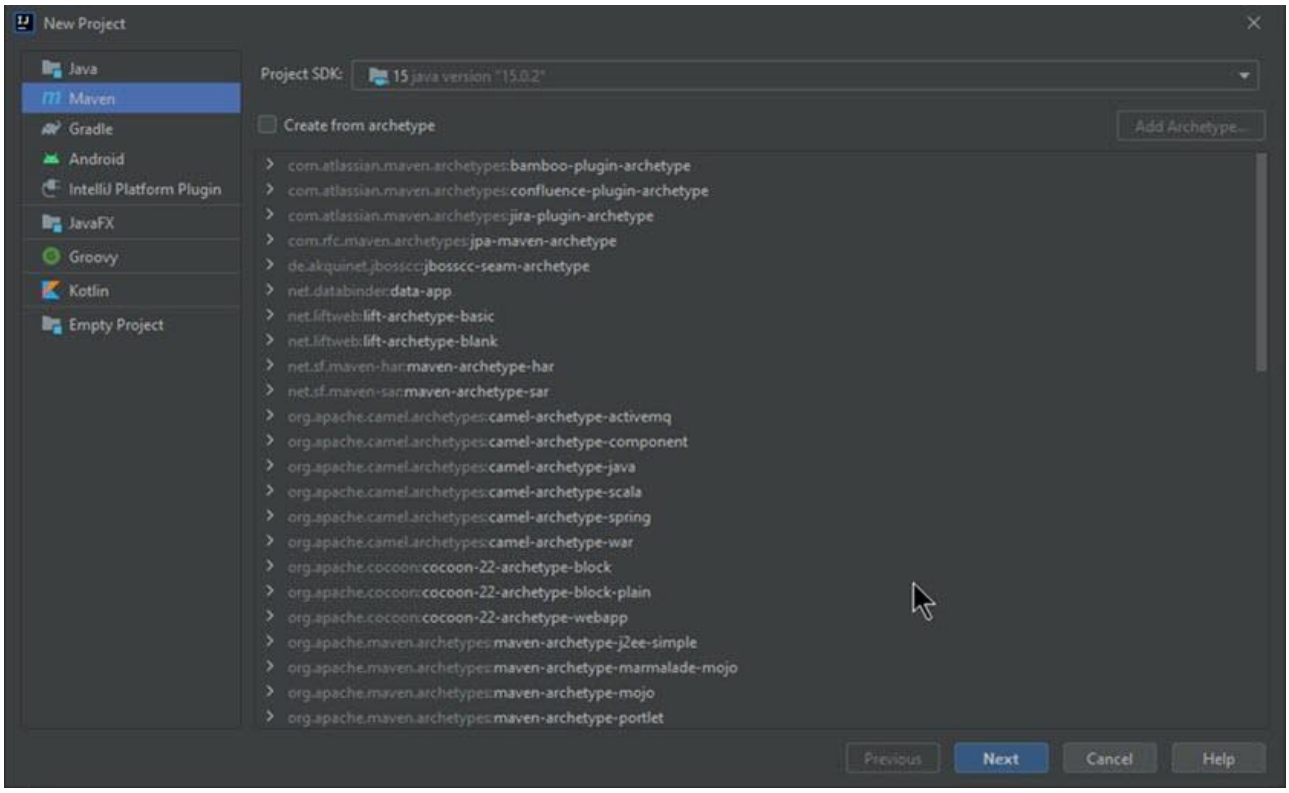


Рис. 6 *Створення проєкту в IntelliJ IDEA*

Перед створенням проєкту в IntelliJ IDEA користувачу необхідно вибрати платформу, як засіб автоматизації, на якій буде здійснюватися тестування. В IntelliJ IDEA є перелік платформ, таких як Java, Maven, Gradle, Groovy, Kotlin та ін.

В залежності від вибору платформи, будуть дещо різнитися скрипти бібліотек Selenium. Для прикладу нижче розглядається платформа Maven. Скрипти для Maven є дещо довшими ніж, наприклад, у інших платформ, таких, як Gradle або Kotlin, проте синтаксис є більш звичним.

Приклад скрипта для Maven:

```
<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
</dependency>
```

Приклад скрипта для Gradle:

```
// https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java
implementation group: 'org.seleniumhq.selenium', name: 'selenium-java', version: '3.141.59'
```

Приклад скрипта для Kotlin:

```
// https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java
implementation("org.seleniumhq.selenium:selenium-java:3.141.59")
```

Після створення проєкту користувач побачить наступне (Рис. 7):

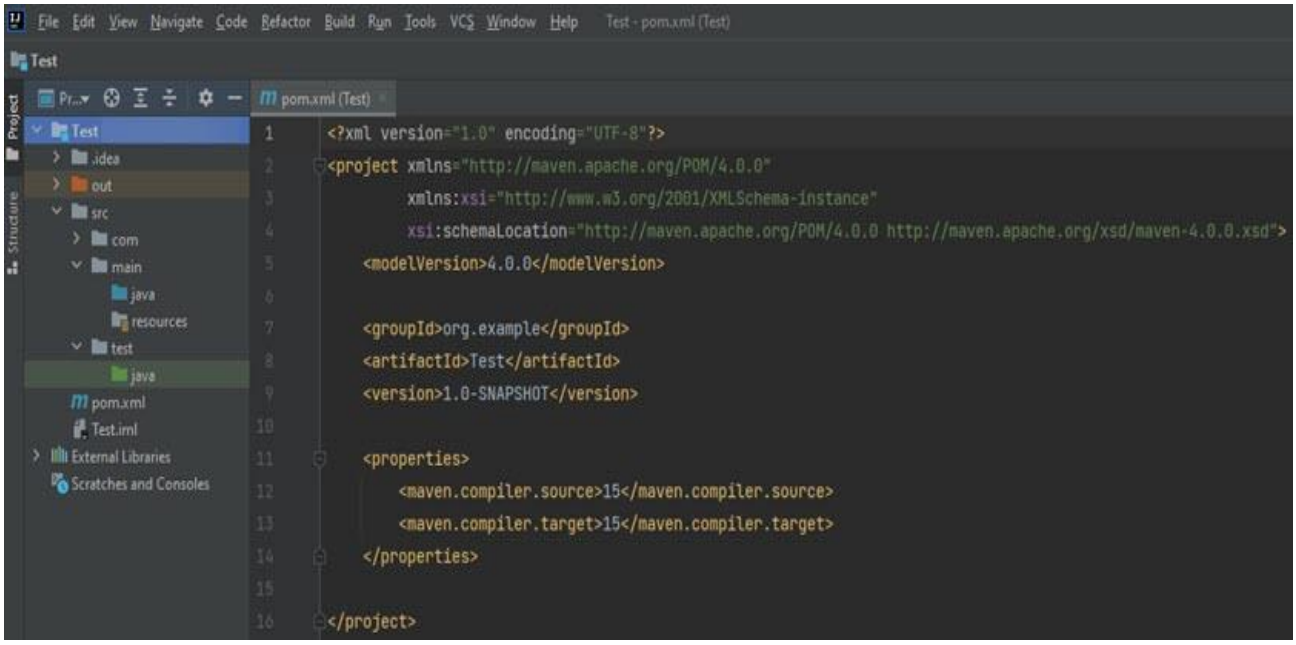


Рис. *7 Інтерфейс IntelliJ IDEA та створений проєкт*

Після створення проєкту на платформі Maven, необхідно встановити зв’язок, так звану залежність новоствореного проєкту з бібліотекою Selenium. Бібліотеки можна використовувати з сайту за посиланням https://mvnrepository.com/, копіюючи скрипти для відповідної платформи Maven. Скрипт залежності в проєкті буде виглядати наступним чином (Рис. 8):

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

    <groupId>com.qatestlab.net</groupId>
    <artifactId>testart</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>15</maven.compiler.source>
        <maven.compiler.target>15</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.141.59</version>
        </dependency>

    </dependencies>
</project>
```

Рис. *8. Скрипт залежності в проєкті Maven*

Також необхідно завантажити з мережі Інтернет сам драйвер браузера з відповідною версією. Посилання на сайти, де можна завантажити драйвери для браузерів наведено нижче:

- Chrome – http://chromedriver.chromium.org/downloads
- Firefox – https://github.com/mozilla/geckodriver/releases
- Microsoft Edge – https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
- IE – https://www.selenium.dev/documentation/ie_driver_server/
- Safari – https://developer.apple.com

Налаштування залежностей та Веб-драйвера браузера – це перше, що необхідно зробити, для того, щоб приступити до процесу автоматизації. Так як тестування буде здійснювати в браузері Google Chrome, потрібно завантажити відповідний драйвер «chromedriver.exe» з версією, аналогічною версії браузера. Далі завантажений файл «chromedriver.exe» потрібно додати в папку «resources», яка за замовчуванням є в новоствореному проекті (Рис. 9).

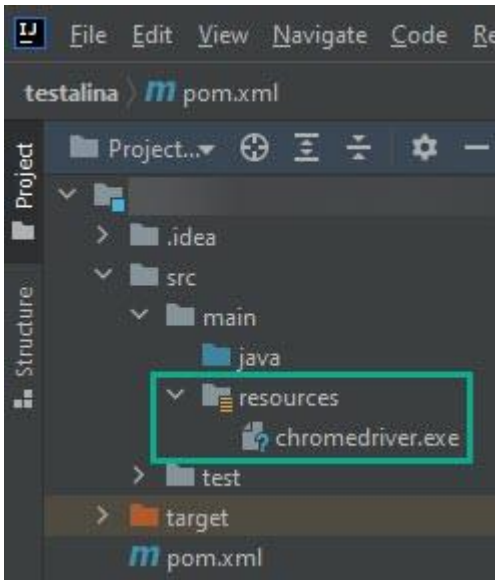


Рис. *9 Розміщення Веб-драйвера в проєкті*

Усі подальші дії, які будуть виконуватися над драйвером та безпосередньо елементами Веб-сайту здійснюються в окремому класі Java, який користувач також створює самостійно (Рис. 10).

В цьому класі важливо вказати шлях до драйвера та властивості, без цього скрипт (Рис. 11) не буде працювати та неможливо буде виконати авто-тест. Шлях можна вказувати як відносний, так і абсолютний, якщо пам’ятаєте куди саме помістили файл «chromedriver.exe» в каталозі папок.

Після всіх налаштувань проєкту можна приступати до виконання основної частини – написання авто-тесту для автоматизації порівняння вартості товару на сайті та в кошику після додавання товару користувачем для здійснення покупки. Варто звернути увагу, що в наведеному прикладі нижче, виконується подвійне

додавання одного й того ж товару, а відповідно і перевірка вартості товару в кошику повинна враховувати подвійну вартість.

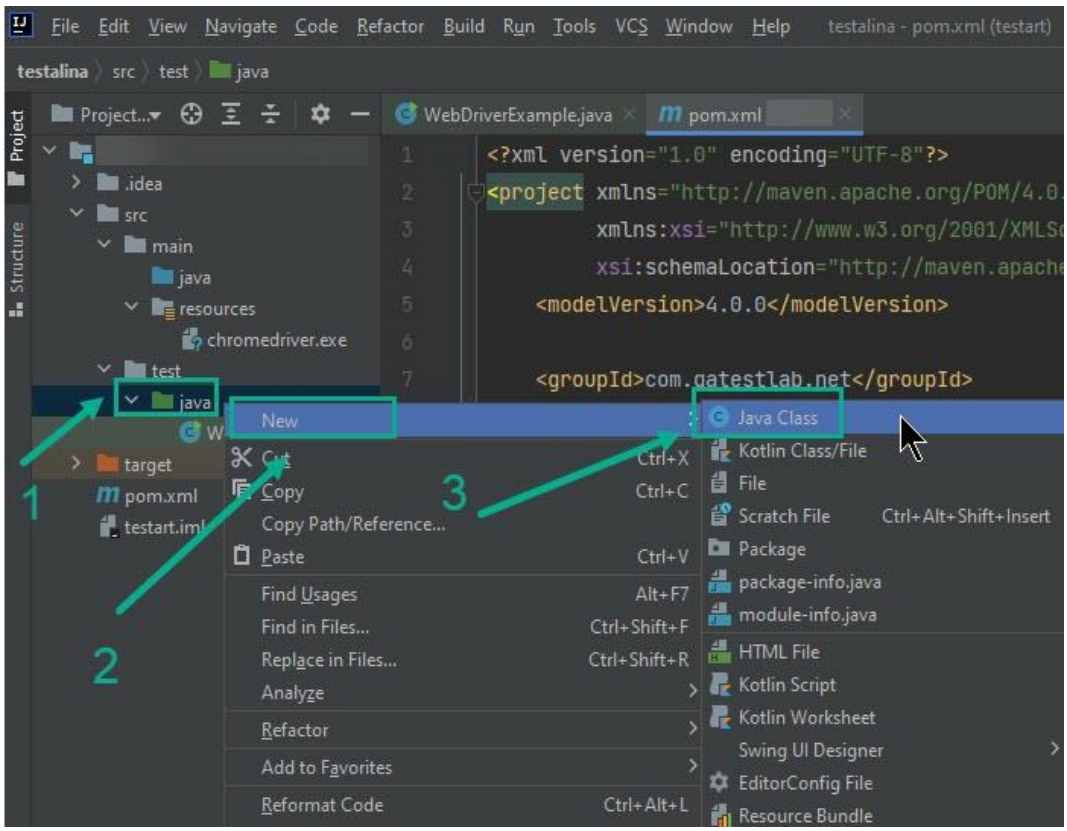


Рис. *10 Створення класу Java*

```
public static WebDriver getDriver() {
    System.setProperty("webdriver.chrome.driver", "src/main/resources/chromedriver.exe")
    return new ChromeDriver();
```

Рис. *11 Скрипт вказування відносного шляху до Веб-драйвера*

Отже, для початку, необхідно знайти ряд елементів, над якими виконуватимуться певні дії, здебільшого кліки або ж логічні операції порівняння. Це такі елементи, як конкретний товар на сторінці сайту, ціна товару, кнопка «Додати в кошик», кнопка «Кошик», випадаюче меню кошика, кнопка «Перейти до кошика» та вартість товару в кошику. Пошук елементів на сайті здійснюється за допомогою локаторів By.XPath. Синтаксис запису елементів в HTML-дереві, які перелічено вище, можна знайти та скопіювати через Chrome DevTools.

В кінці скрипта здійснюється порівняння вартості за допомогою оператора умови if...else. А в результаті запуску даного тесту отримуємо або успішну відповідність вартості товару, або знаходимо баг у вигляді відповіді в консолі IntelliJ IDEA (Рис. 12).

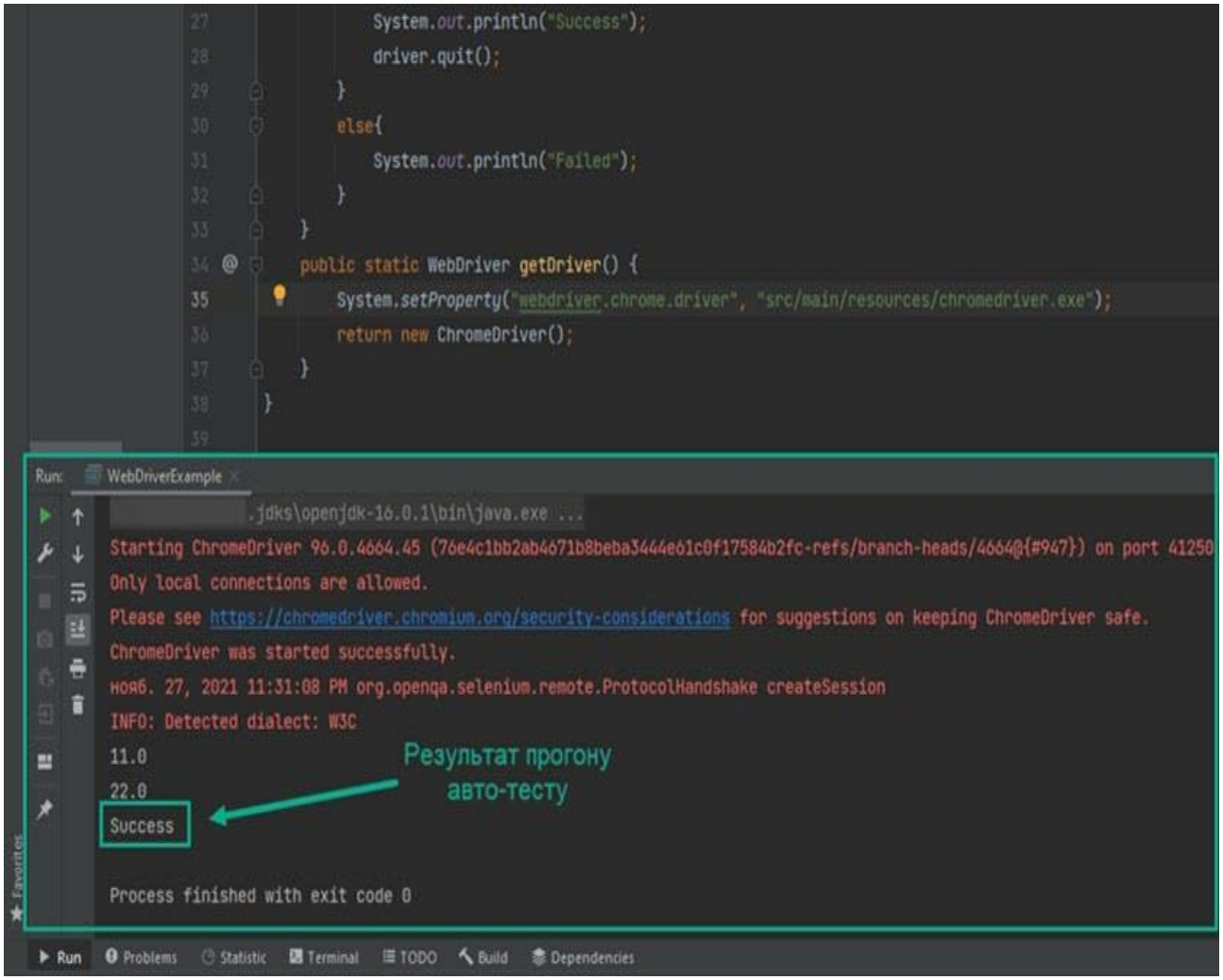


Рис. *12 Консоль результатів тестування IntelliJ IDEA*

2.2 Вибір мови програмування

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

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

Хоча програми Java компілюються безпосередньо, Python інтерпретується, що сповільнює програми Python під час виконання. Визначення типу змінної, що виникає під час виконання, збільшує навантаження на інтерпретатор. Крім того, запам'ятовування типи об'єктів, отриманих з об'єктів-контейнерів, сприяє використанню пам'яті. Виправлення помилок у Python проти Java. У Python будь-які помилки, внесені програмістом, не будуть знайдені, доки не буде запущено цей рядок коду. Це може призвести до збою в роботі та подовжити час виконання. Хоча Python залишає об’єкти вразливими до мутацій, в Java мутації об’єктів неможливі. Це веде до безпечної розробки програмного забезпечення.

Ключові різниці між мовами програмування Java та Python перераховані далі. Java – це скомпільована плюс інтерпретована мова, тоді як Python – це інтерпретована мова.

Java типізується статично, тоді як Python — динамічно. Java має складну криву навчання, тоді як Python легко вивчати та використовувати.

Java є багатоплатформною, об'єктно-орієнтованою та мережевою мовою програмування, тоді як Python є високорівневою об'єктноорієнтованою мовою програмування. Java потребує 10 рядків коду для зчитування з файлу, тоді як Python потребує лише 2 рядків коду. Мова Java використовує фігурні дужки для визначення початку та кінця кожної функції та визначення класу, тоді як Python використовує відступ для розділення коду на окремі блоки. У Java множинне успадкування частково здійснюється через інтерфейси, тоді як Python підтримує як одинарне, так і множинне успадкування. Java пропонує обмежені функції,

пов’язані з рядками, тоді як Python пропонує багато функцій, пов’язаних з рядками.

Програма Java може працювати на будь-якому комп’ютері або мобільному пристрої, який може запускати JVM, тоді як програмам Python потрібен інтерпретатор, встановлений на цільовій машині для перекладу коду Python. У порівнянні з Java, Python менш переносимий. Java найкраще підходить для настільних програм із графічним інтерфейсом користувача, систем вбудовування, служб Веб-застосунків, тоді як Python відмінно підходить для наукових і числових обчислень, програм машинного навчання тощо. Java пропонує стабільне підключення, тоді як Python пропонує слабке підключення.

3 ОПИС ЗАСТОСУНКУ ТА ВИЗНАЧЕННЯ ЙОГО ФУНКЦІОНАЛУ, ЩО БУДЕ ТЕСТУВАТИСЬ

3.1 Опис функціоналу мобільного застосунку

Розглянемо використання Selenium IDE для системного тестування Веб-застосунку «Депозитний калькулятор» https://fin-calc.org.ua/ua/deposit/calculate/, який дозволяє розраховувати прибуток від депозиту, що нараховується кожен місяць, та сумарний прибуток за весь термін депозиту (Рис. 13, Рис. 14).

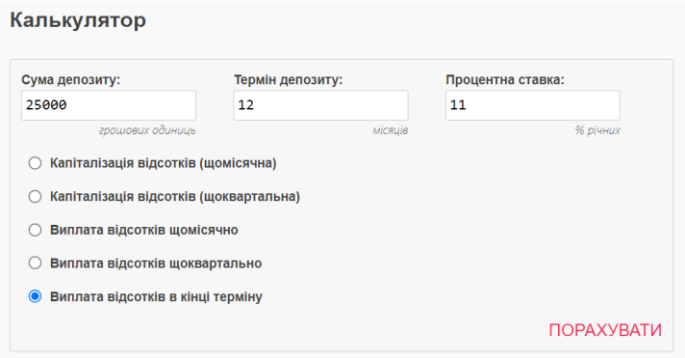
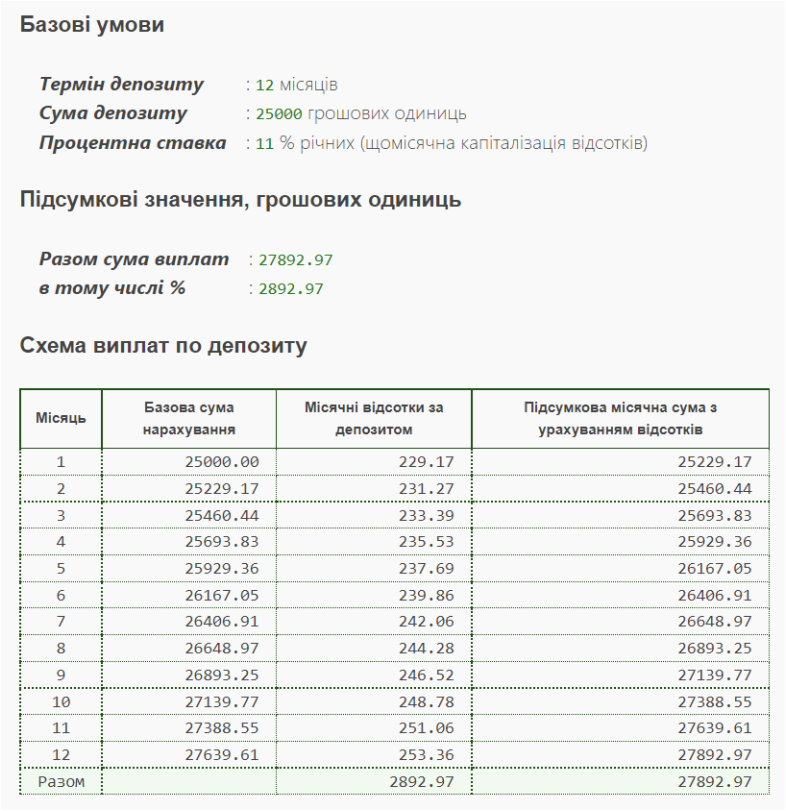


Рис. *13 Веб-застосунок, який тестується*

З інтерфейсу системи зрозуміло, що передбачається 5 шляхів, один з яких користувач може обрати за допомогою групи радіокнопок. У той же час апробація застосунку показала, що результат розрахунків при виборі будь-якої з трьох нижніх радіокнопок не відрізняється. Таким чином, кількість можливих шляхів тесту зводиться до трьох: капіталізація відсотків (щомісячна), капіталізація відсотків (щоквартальна) і виплата відсотків (з будь-яким періодом).



Базові умови

Термін депозиту : 12 місяців
Сума депозиту : 25000 грошових одиниць
Процентна ставка : 11 % річних (щомісячна капіталізація відсотків)

Підсумкові значення, грошових одиниць

Разом сума виплат : 27892.97
в тому числі % : 2892.97

Схема виплат по депозиту

Місяць	Базова сума нарахування	Місячні відсотки за депозитом	Підсумкова місячна сума з урахуванням відсотків
1	25000.00	229.17	25229.17
2	25229.17	231.27	25460.44
3	25460.44	233.39	25693.83
4	25693.83	235.53	25929.36
5	25929.36	237.69	26167.05
6	26167.05	239.86	26406.91
7	26406.91	242.06	26648.97
8	26648.97	244.28	26893.25
9	26893.25	246.52	27139.77
10	27139.77	248.78	27388.55
11	27388.55	251.06	27639.61
12	27639.61	253.36	27892.97
Разом		2892.97	27892.97

Рис. *14 Результати розрахунку доходів від депозиту*

Далі слід проаналізувати можливі діапазони значень суми депозиту, терміну депозиту, процентної ставки та визначити відповідні класи еквівалентності. Можливі обмеження вхідних даних було б доцільно представити на сторінці з інструкцією використання депозитного калькулятора, до якої веде посилання Як розрахувати відсотки по депозиту. Проте таких обмежень на ній не вказано (очевидно, покладаючись на розумність вибору користувача). Методом підбору були з'ясовані допустимі значення для поля Термін депозиту – цілі числа від 1 до 255, для поля Процентна ставка – числа з плаваючою комою від 0.000001 до 1000.0. Для поля Сума депозиту вдалося з'ясувати тільки нижню межу – 0, тип чисел – тільки цілі (що взагалі дивно), а також, що при досить великих значеннях результат обчислення прибутку виводиться як INF (нескінченість). Визначені класи еквівалентності, обрані вхідні тестові значення, відповідні граничним умовам, та типи тестових сценаріїв наведені у таблиці 1.

Табл. *1*

Класи еквівалентності та пропоновані вхідні дані для тестів

Тип вхідних даних	Тип даних	Клас еквівалентності	Вхідні дані для тесту	Тип сценарію
Сума депозиту	int	[0, ∞]	0, 1, 25000	позитивний
Сума депозиту	int	[-∞, -1]	-1	негативний
Термін депозиту	int	[1, 255]	1, 12, 255	позитивний
Термін депозиту	int	[-∞, 0]	-1, 0	негативний
Термін депозиту	int	[256, ∞]	256	негативний
Процентна ставка	double	[0.000001, 1000.0]	0.000001, 0.0000011, 11.0, 1000.0	позитивний
Процентна ставка	double	[-∞,0.00000099...]	-1.0, 0.0, 0.000000999999999999999	негативний
Процентна ставка	double	[1000.000...1, ∞]	1000.0000000000001	негативний

Для незалежного обчислення очікуваних результатів виконаємо розробку відповідної таблиці у табличному процесорі (наприклад, Microsoft Excel) (вона буде тестовим оракулом), заклавши у неї відповідні формули для розрахунку банківських процентів [5]. У таблиці 2 наведений приклад розрахунку очікуваних значень прибутку від депозиту при вхідних даних: сума депозиту – 25000 грн; термін депозиту – 12 місяців; процентна ставка – 11%.

Табл. *2*

Розрахунок очікуваних значень

Місяць	Щомісячна капіталізація %		Поквартальна капіталізація %		Без капіталізації %	
	сума на рахунку	дохід	сума на рахунку		сума на рахунку	дохід
1	25000,00	229,17	25000,00	229,17	25000,00	229,17
2	25229,17	231,27		229,17	25000,00	229,17
3	25460,44	233,39		229,17	25000,00	229,17
4	25693,83	235,53	25687,51	235,47	25000,00	229,17
5	25929,36	237,69		235,47	25000,00	229,17
6	26167,05	239,86		235,47	25000,00	229,17
7	26406,91	242,06	26393,92	241,94	25000,00	229,17
8	26648,97	244,28		241,94	25000,00	229,17
9	26893,25	246,52		241,94	25000,00	229,17
10	27139,77	248,78	27119,74	248,60	25000,00	229,17
11	27388,55	251,06		248,60	25000,00	229,17
12	27639,61	253,36		248,60	25000,00	229,17
Очікуваний:	**27892,97**	**2892,97**	**27865,54**	**2865,54**	**27750,04**	**2750,04**
Фактично:	**27892,97**	**2892,97**	**27865,54**	**2865,54**	**27750,04**	**2750,04**

3.2 Налаштування та проведення тестування в Selenium IDE

Для системного тестування Веб-застосунків з використанням засобів Selenium необхідно встановити плагін Selenium IDE для браузеру Chrome (або Firefox, або Edge), посилання на сторінки плагінів є на офіційному сайті Selenium https://selenium.dev/downloads/. Після встановлення плагіну на панелі інструментів з'явиться піктограма запуску Selenium IDE (Рис. 1.5).

Рис. *15 Піктограма запуску Selenium ID*

Виконаємо розробку тесту для визначеного у Табл. 2 тестового сценарію. Для цього запустимо Selenium IDE, на першому екрані оберемо команду Record a New Test in a New Project (Запис нового тесту у новому проєкті). У наступних екранах, що відкриваються, вкажемо назву проєкту та URL-адресу застосунку, що тестується. Після натискання на кнопку START RECORDING (ПОЧАТИ ЗАПИС) починається запис скрипту – завантажується сторінка сайту у браузері та періодично з'являється спливаюче вікно, яке вказує, що відбувається запис .

На сторінці калькулятора вводимо до відповідних полів значення суми депозиту (25000), терміну депозиту (12) та процентної ставки (11), після чого виконаємо клік по радіокнопці Виплата відсотків в кінці терміну, а потім – по кнопці ПОРАХУВАТИ (для того, щоб Selenium IDE зафіксував введення, потрібно повторно ввести дані у тестові поля та виконати клік по радіокнопці). Після цього зупинимо запис натисканням на кнопці запису у вікні Selenium IDE, при цьому з'явиться вікно, у якому необхідно вказати назву тестового сценарію.

Дії користувача відображаються у Selenium IDE як список команд (Рис. 1.8). Зверніть увагу, що записується кожна дія користувача, і навіть деякі параметри браузера, тому команди, які записали випадкові дії користувача, або

неважливі 9 команди можна видалити обравши Delete із контекстного меню, яке відкривається після кліку на команді правою кнопкою миші.

Кожна команда (Command) має параметри: мета (Target), значення (Value) та опис (Description). Наприклад, команда open в якості мети має URL сторінки, що тестується, а команда type в якості мети має локатор HTML-елементу сторінки, а в якості значення – значення, що вводиться до цього елементу (Рис. 16).

За відсутності необхідної команди можна додати її вручну. Для цього необхідно виділити у списку команду, перед якою необхідно вставити нову команду, виконати клік по ній правою кнопкою миші і обрати із контекстного меню Insert new command (Вставити нову команду). Далі у полі Command внизу вікна необхідно ввести команду (при цьому відкривається список доступних команд, а при виборі команди – відповідна довідка), мету (Target) та значення (Value).

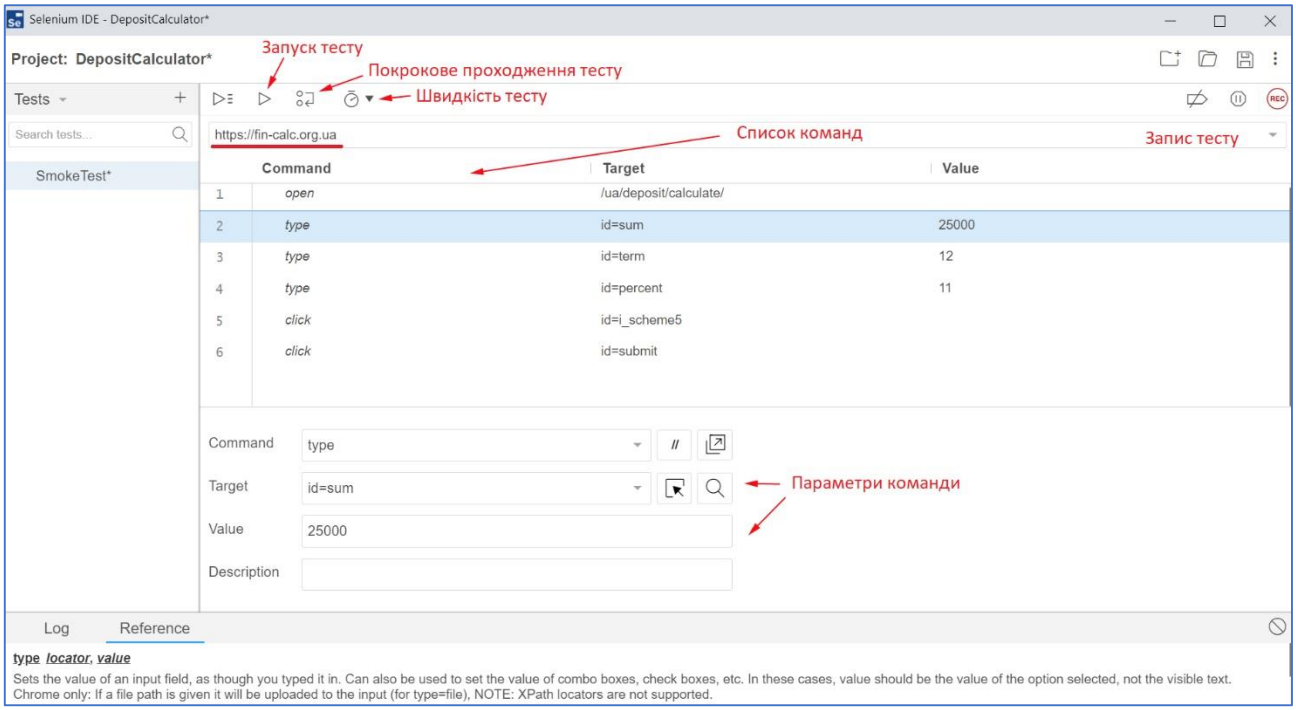


Рис. *16 Вікно Selenium IDE з командами тестового сценарію*

Усі команди поділяються на:

- команди-дії (Actioins);

- команди, що здійснюють доступ до змінних (Accessors);

- команди, що здійснюють верифікацію даних (Assertions);

- команди, що здійснюють очікування появи елемента (Waiters).

Команди-дії (Actioins) змінюють стан застосунку (вони очікують завершення повного завантаження сторінки перед виконанням наступної команди). Прикладами команд-дій є команди:

- open – завантаження Веб-сторінки за вказаним URL;

- type – введення тексту до вказаного HTML-елементу;

- select – вибір певного елементу зі вказаного списку;

- check – перевіряє, чи встановлений елемент чекбокса, радіокнопки, кнопки, що перемикається;

- click – імітує клік на вказаному посиланні, кнопці, чекбоксі або радіокнопці.

Команди, що здійснюють доступ до змінних (Accessors), визначають стан застосунку та зберігають данні у змінних. Прикладами таких команд є:

- store – зберігає рядок з поля Target у змінну, вказану у полі Value;

- store text – зберігає текстове значення вказаного у полі Target HTMLелементу до вказаної у полі Value змінної;

Збережені у змінній значення можуть використовуватися у подальших командах (як їх параметри) у вигляді ${ім'я_змінної} (приклад використання змінних буде наведений далі).

Команди, що здійснюють верифікацію даних (Assertions), виконують порівняння результату роботи застосунку з очікуваним результатом. Ці команди можна поділити на два підтипи, що розпочинаються з префіксів assert та verify:

- команди, що починаються з assert, провалюють тест та припиняють його виконання у разі неспівпадіння фактичного та очікуваного результату;

- команди, що починаються з verify, виводять повідомлення про помилку у разі неспівпадіння фактичного та очікуваного результату, але продовжують виконання тесту.

В якості прикладів команд, що здійснюють верифікацію даних, можна привести:

- assert element present - виконує перевірку, чи знаходиться вказаний у полі Target Веб-елемент на сторінці, провалюючи тест за його відсутності;

- verify element present – виконує перевірку, чи знаходиться вказаний у полі Target Веб-елемент на сторінці, виводячи повідомлення про помилку при його відсутності, але продовжуючи виконання тесту;

Команди, що здійснюють очікування появи елемента на сторінці (Waiters), забезпечують можливість призупинення виконання тестового скрипту до появи 11 визначеного як параметр елементу сторінки, який буде тестуватись наступними командами.

В якості прикладів команд, що здійснюють очікування появи елемента на сторінці, можна привести:

- команди, що починаються з wait for, чекають вказаного стану HTMLелементу, який ідентифікується локатором у полі Target, впродовж часу тайм-ауту (в мілісекундах), вказаного у полі Value.

- wait for element present – чекає появи на сторінці вказаного у полі Target Веб-елементу впродовж часу тайм-ауту, встановленого у полі Value. Якщо за цей час елемент не з'являється, тест провалюється.

Документація щодо усіх команд Selenium IDE доступна за посиланням https://www.selenium.dev/selenium-ide/docs/en/api/commands.

Поле мети (Target) призначене для зазначення локаторів HTMLелементів, по відношенню до яких виконується поточна команда, або значення змінної при записі цього значення у змінну або імені змінної при виведенні значення змінної командою echo або URL-адреси команди open (Рис. 17).

Command	Target	Value
open	https://ui.vision/	
store	12345	myvar
echo	${myvar}	
type	id=phone	${myvar}

Рис. *17 Приклади використання поля Target*

Загальний синтаксис локаторів Веб-елементів має такий вигляд:

тип_локатора_HTML-елементу = значення_локатора

Наприклад, для текстового поля введення суми депозиту, HTML-код якого можна побачити, виконавши у браузері Chrome клік правою кнопкою миші по Веб-сторінці або її елементу та обравши команду Перевірити (у браузері Firefox аналогічно – команду Дослідити):

- у разі використання ідентифікатора елементу input – атрибуту id поле мети (Target) виглядатиме:

id=sum

- у разі використання атрибуту name елементу input:

name=sum

- у разі використання CSS-селектора:

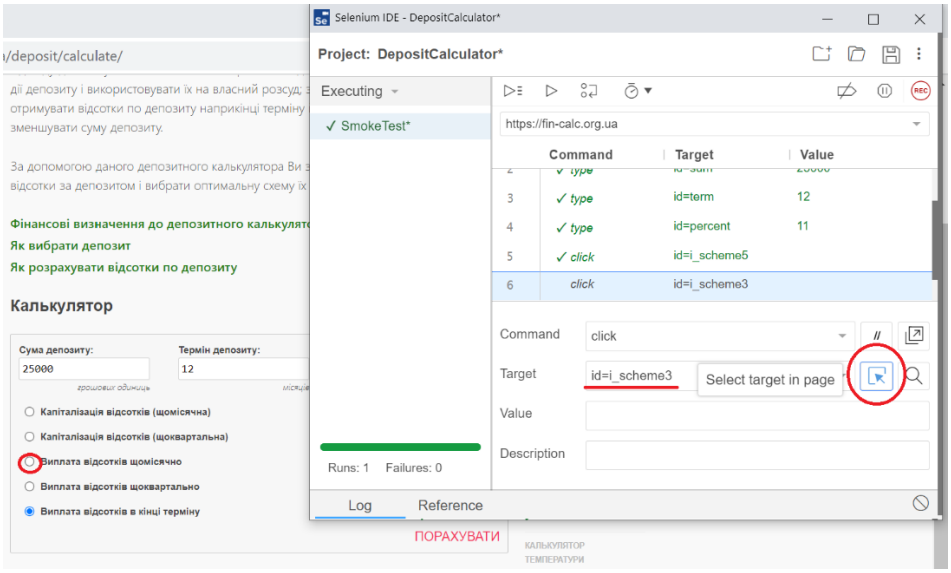
css=#sum

- у разі використання XPath

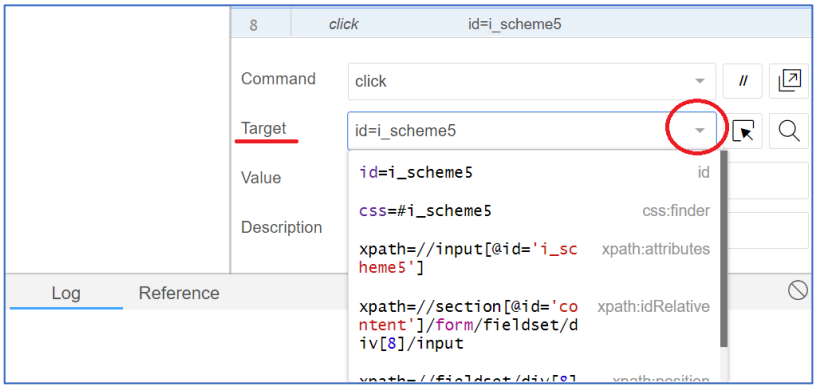
xpath= //input[@id='sum']

Останні два локатори зручно отримувати з вікна дослідження HTML-коду браузера командою Copy контекстного меню (для Chrome) .

Існує спосіб визначення для команди локатора HTML-елементу у полі мети (Target) безпосередньо в Selenium IDE. Для цього необхідно виконати клік по кнопці Вибір мети на сторінці (Select target in page) і виділити на HTML-сторінці необхідний елемент кліком по ньому. При цьому в поле мети (Target) вписується оптимальний локатор вказаного елементу, а зі списку у цьому полі можна обрати доступний локатор іншого типу (Рис. 18).



а)



б)

Рис. *18 Визначення локатора HTML- елементу*

Клік по кнопці Знайти мету на сторінці (Find target in page) після введення до цього поля локатора HTML-елементу підсвічує цей елемент на сторінці, що тестується. Така функція надає можливість переконатися, що команда буде застосована до бажаного елементу (Рис. 19).

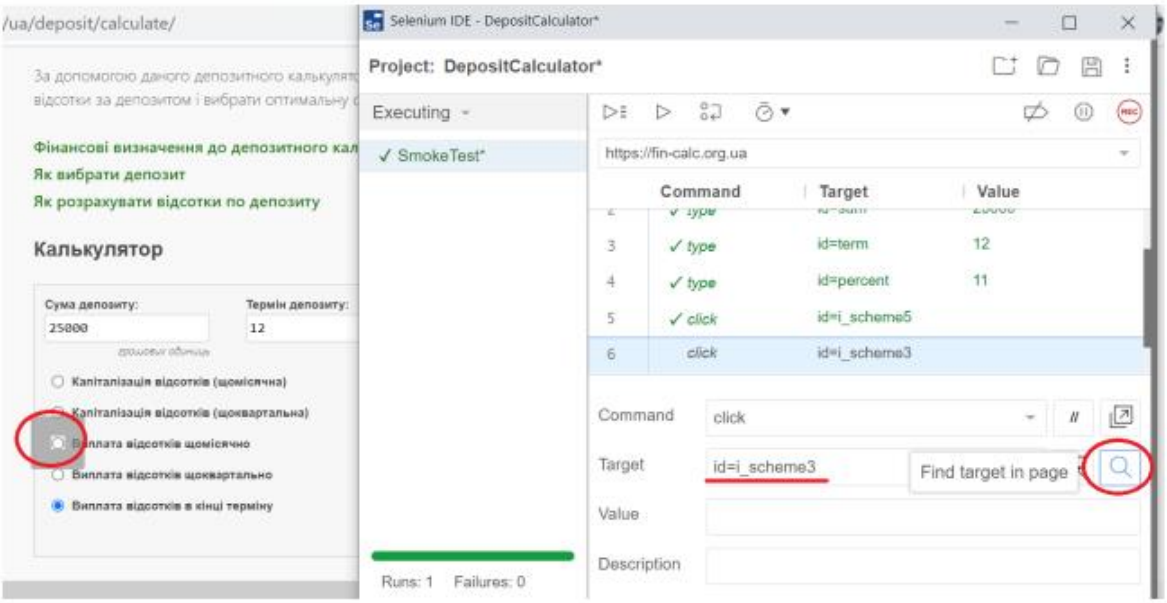


Рис. *19 Пошук HTML-елемента по локатору*

Запустити скрипт можна кліком по піктограмі запуску , на цій же панелі доступна піктограма регулювання швидкості виконання скрипту . У вікні браузеру буде видно, як повторно завантажується сторінка зі значеннями полів за замовчуванням, як поля заповнюються вхідними даними та виконується їх відправка на сервер кліканням по елементу ПОРАХУВАТИ. У нижній частині вікна на вкладці Журнал (Log) виводяться повідомлення про виконання команд. Вкладка Довідка (Reference) містить опис команди тестового скрипта, яка виділена у верхній частині вікна. Правіше від вкладок розташована кнопка Очистити журнал (Clear log) , що видаляє записи журналу.

Після розрахунку прибутку від депозиту, ініційованого кліканням по елементу ПОРАХУВАТИ, необхідно у частині сторінки з результатами виявити значення розрахованих (фактичних) сумарних відсотків за депозитом та загальної суми виплат і ввести до тестового скрипту команди перевірки цих значень. Для цього у вікні Selenium IDE виділіть рядок нижче останньої команди скрипту, увімкніть режим запису кліканням по кнопці , після чого перейдіть на Веб-сторінку, що тестується, та виконайте клік правою кнопкою миші на значенні розрахованої суми відсотків за депозитом. З контекстного меню оберіть

команду VerifyText. До скрипту буде додана команда верифікації з автоматичним вибором мети

`(Target) css=.finale > .right:nthchild(3)` (Рис. 20).

Повторіть введення команди на цей раз `assert text` для значення суми депозиту разом з нарахованими відсотками.

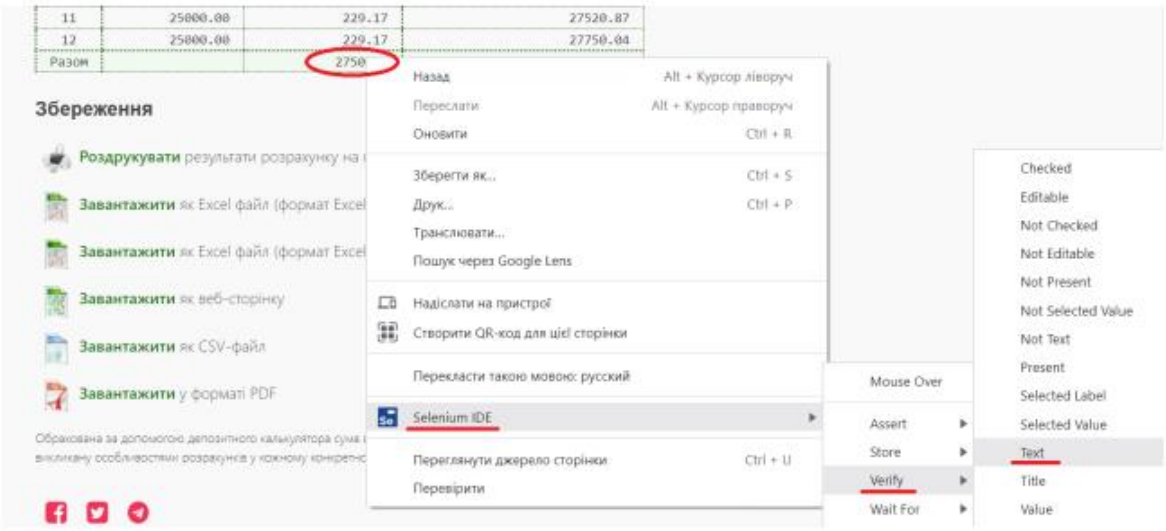
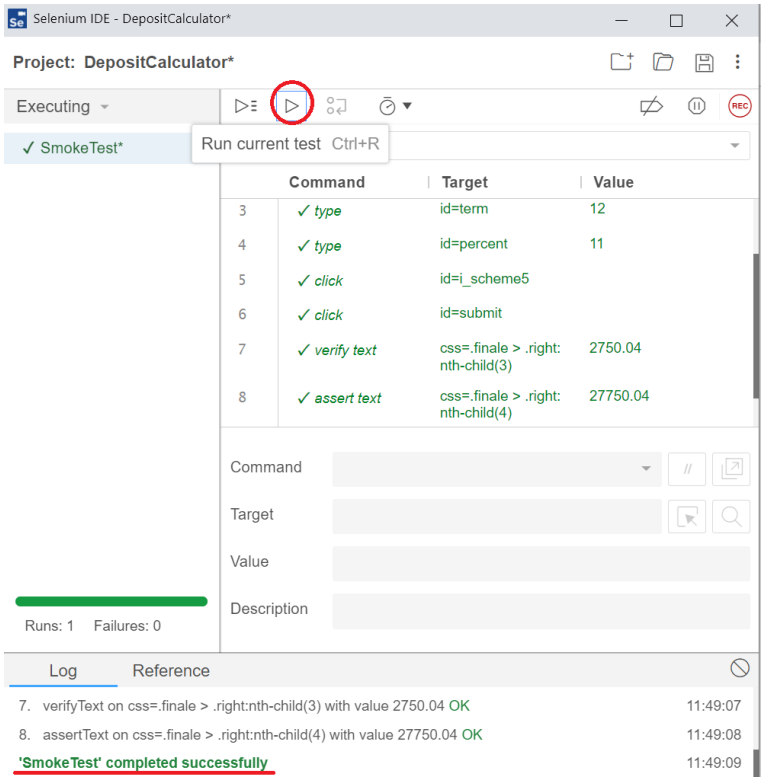


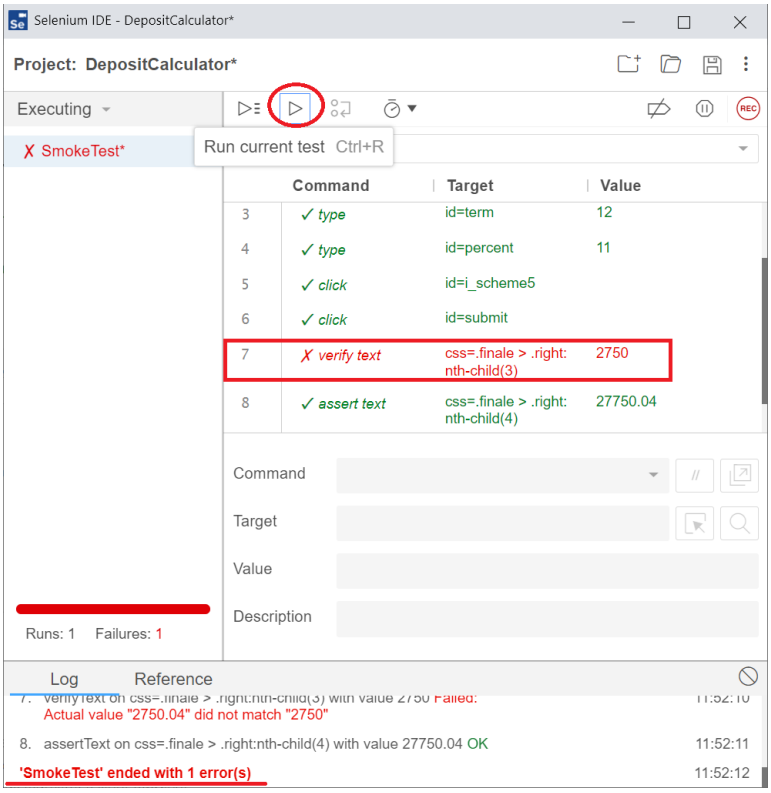
Рис. *20 Додання команди перевірки значення, що повертається*

Зауважте, що у полі мети (Target) команд verify text та assert text повинне вказуватися очікуване значення, розраховане незалежним способом (у цьому прикладі – за допомогою табличного процесора). Якби ці значення відрізнялися, нам необхідно було б встановити поле Target саме в очікуване значення.

Тепер розробка тестового сценарію завершена. Запустіть його кліком по кнопці ▷, після успішного виконання його команд вони підсвічуються шрифтом зеленого кольору з встановленням мітки (Рис. 21а), що говорить про успішне проходження тесту. На Рис. 21,б продемонстровано проходження тесту з помилкою команди верифікації (було змінене очікуване значення у полі значення (Value) з 2750.04 на 2750, шрифт червоного кольору сигналізує про помилку). Оскільки не пройшла команда верифікації, тест було подовжено та завершено із повідомленням про одну помилку. У разі, якщо не пройшла б команда ствердження, тест би був на ній зупинений.



а)



б)

Рис. *21 Результати виконання тестового скрипту: а) успішне, б) неуспішне*

3.3 Визначення функціональних вимог застосунку та розробка тест-вимог

У разі наявності функціональних вимог до системи, має сенс розробка тест-вимог з подальшою розробкою тест-плану, що описує яким чином виконуватиметься тестування та включає набір тестових сценаріїв (TestCase). Функціональні вимоги описують поведінку системи як "чорної скриньки", тобто виключно з позицій того, що повинна робити система у різних ситуаціях. Іншими словами, функціональні вимоги визначають реакцію системи на різні вхідні дії. Оскільки нам недоступні формальні функціональні вимоги до депозитного калькулятора, виконаємо припущення відносно них, виходячи з інтерфейсу Веб-застосунку та доступних для нього довідок.

Функціональні вимоги до Веб-застосунку «Депозитний калькулятор»:

1. Вхід:

1.1.Значення поля Сума депозиту (у грошових одиницях) – ініціюється значенням 25000 та дозволяє вводити цілі значення від 0 до дуже великих значень, коли результат обчислення прибутку виводиться як INF.

1.2.Значення поля Термін депозиту (у місяцях) – ініціюється значенням 12 та дозволяє вводити цілі числа від 1 до 255.

1.3.Значення поля Процентна ставка (у % річних) ініціюється значенням 11 та дозволяє вводити значення як числа з плаваючою комою від 0.000001 до 1000.0 (розрахунок виконується при використанні як коми так і крапки у якості роздільника цілої та добової частини, після розрахунку кома змінюється на крапку).

1.4.Опція, що визначає схему нарахування відсотків – ініціюється значенням Виплата відсотків в кінці терміну і може бути встановлена в одне зі значень.

1.4.1. Капіталізація відсотків (щомісячна);

1.4.2. Капіталізація відсотків (щоквартальна);

1.4.3. Виплата відсотків щомісячно;

1.4.4. Виплата відсотків щоквартально;

1.4.5. Виплата відсотків в кінці терміну

2. Вихід:

2.1.У підблоці Базові умови блоку Результати:

2.1.1. Термін депозиту – виводиться введене значення у місяцях;

2.1.2. Сума депозиту – виводиться введене значення в грошових одиницях;

2.1.3. Процентна ставка – виводиться введене значення в % річних та схема нарахування відсотків.

2.2.У підблоці Підсумкові значення, грошових одиниць блоку Результати:

2.2.1. Разом сума виплат – виводиться розрахована сума виплат, яка дорівнюється первинній сумі депозиту плюс сума нарахованих за увесь термін депозиту відсотків;

2.2.2. в тому числі % – виводиться розрахована сума нарахованих за увесь термін депозиту відсотків.

2.3.У блоці Схема виплат по депозиту для кожного місяця депозиту виводяться:

2.3.1. Порядковий номер місяця, починаючи з початку депозиту;

2.3.2. Базова сума нарахування (у грошових одиницях), яка для першого місяця депозиту дорівнює первинній сумі депозиту, а для подальших місяців:

2.3.2.1. У разі нарахування відсотків без капіталізації базова сума нарахування дорівнює первинній сумі депозиту для кожного місяця;

2.3.2.2. У разі нарахування відсотків з капіталізацією базова сума нарахування дорівнює базовій сумі попереднього періоду капіталізації плюс сума нарахованих за цей період капіталізації відсотків.

2.3.3. Місячні відсотки за депозитом – нараховані відсотки за депозитом за кожен місяць (у грошових одиницях);

2.3.4. Підсумкова місячна сума з урахуванням відсотків (у грошових одиницях) – базова сума нарахування плюс сума нарахованих відсотків за кожен місяць;

2.3.5. У схемі виплат по депозиту також наводяться:

2.3.5.1. Підсумкова сума нарахованих відсотків (дорівнює значенню у п. 2.2.2);

2.3.5.2. Підсумкова сума виплат (дорівнює значенню у п. 2.2.1).

2.4. У підблоці Збереження блоку Результати виводиться перелік піктограм та посилань, які надають можливість друку або збереження у файлах різних форматів результатів розрахунку:

2.4.1. Роздрукувати результати розрахунку на принтері;

2.4.2. Завантажити як Excel файл (формат Excel 5);

2.4.3. Завантажити як Excel файл (формат Excel 2007);

2.4.4. Завантажити як Веб-сторінку;

2.4.5. Завантажити як CSV-файл;

2.4.6. Завантажити у форматі PDF.

Відповідно до функціональних вимог складемо тест-вимоги: Тест-вимоги до Веб-застосунку «Депозитний калькулятор»:

1. Перевірка ініціалізації застосунку (після первинного завантаження сторінки):

1.1.Перевірити, що значення поля Сума депозиту дорівнює 25000 (п 1.1 функціональних вимог).

1.2.Перевірити, що значення поля Термін депозиту дорівнює 12 (п 1.2 функціональних вимог).

1.3.Перевірити, що значення поля Процентна ставка дорівнює 11 (п 1.3 функціональних вимог).

1.4.Перевірити, що обрана опція розрахунку схеми нарахування відсотків – Виплата відсотків в кінці терміну (п 1.4 функціональних вимог).

2. Перевірка розрахунку відсотків за депозитом та суми виплат:

2.1.Перевірити, що у підблоці Базові умови блоку Результати виводяться:

2.1.1. У полі Термін депозиту однакові значення з введеними до текстового поля Термін депозиту (п 2.1.1 функціональних вимог);

2.1.2. У полі Сума депозиту однакові значення з введеними до текстового поля Сума депозиту (п 2.1.2 функціональних вимог);

2.1.3. У полі Процентна ставка однакові значення з введеними до текстового поля Процентна ставка (п 2.1.3 функціональних вимог).

2.2.Перевірити, що у підблоці Підсумкові значення, грошових одиниць блоку Результати виводяться:

2.2.1. Правильне значення підсумкової суми виплат – залежно від обраної схеми нарахування відсотків (п 2.2.1 функціональних вимог);

2.2.2. Правильне значення підсумкової суми нарахованих за увесь термін депозиту відсотків – залежно від обраної схеми нарахування відсотків (п 2.2.2 функціональних вимог).

2.3.Перевірити, що у блоці Схема виплат по депозиту для кожного місяця депозиту виводяться:

2.3.1. Правильні значення порядкових номерів місяця, починаючи з початку депозиту (п 2.3.1 функціональних вимог);

2.3.2. Правильні значення базових сум нарахувань, які для першого місяця депозиту дорівнюють первинній сумі депозиту (п 2.3.2 функціональних вимог), а для подальших місяців:

2.3.2.1. У разі нарахування відсотків без капіталізації базова сума нарахування дорівнює первинній сумі депозиту для кожного місяця (п 2.3.2.1 функціональних вимог)

2.3.2.2. У разі нарахування відсотків з капіталізацією базова сума нарахування дорівнює базовій сумі попереднього періоду капіталізації плюс сума нарахованих за цей період капіталізації відсотків (п 2.3.2.2 функціональних вимог).

2.3.3. Правильні значення нарахованих відсотків за депозитом за кожен місяць (п 2.3.3 функціональних вимог);

2.3.4. Правильне значення підсумкової щомісячної суми з урахуванням відсотків, яка дорівнює базовій сумі нарахування плюс сума нарахованих відсотків за кожен місяць (п 2.3.4 функціональних вимог)

2.3.5. Правильні підсумкові значення:

2.3.5.1. Правильне значення підсумкової суми нарахованих відсотків (дорівнює значенню параметра п. 2.2.2) (п 2.3.5.2 функціональних вимог);

2.3.5.2. Правильне значення підсумкової суми виплат (дорівнює значенню параметра п. 2.2.1) (п 2.3.5.1 функціональних вимог).

2.4. Перевірити, що при обранні у підблоці Збереження блоку Результати:

2.4.1. Посилання Роздрукувати результати розрахунку на принтері (п 2.4.1 функціональних вимог):

2.4.1.1. Виконується перехід на сторінку з URL https://fincalc.org.ua/ua/deposit/printer/[термін_депозиту]/[сума_депозиту]/[процентна_ставка]/[номер_опції_схеми_відсотків], де замість параметрів в квадратних скобках вказуються їх значення;

2.4.1.2. Заголовок сторінки містить текст "Розрахунок доходності депозиту і схема виплат відсотків".

2.4.2. Посилання Завантажити як Excel файл (формат Excel 5) (п 2.4.2 функціональних вимог) – виконується завантаження файлу calculationon-deposit.xls у каталог Downloads поточного користувача;

2.4.3. Посилання Завантажити як Excel файл (формат Excel 2007) (п 2.4.3 функціональних вимог) – виконується завантаження файлу calculationon-deposit.xlsx у каталог Downloads поточного користувача;

2.4.4. Посилання Завантажити як Веб-сторінку (п 2.4.4 функціональних вимог) – виконується завантаження файлу calculation-on-deposit.html у каталог Downloads поточного користувача; 2.4.5. Посилання Завантажити як CSV-файл (п 2.4.5 функціональних вимог) – виконується завантаження файлу calculation-on-deposit.csv у каталог Downloads поточного користувача; 2.4.6. Посилання Завантажити у форматі PDF (п 2.4.6 функціональних вимог) – виконується завантаження файлу calculation-on-deposit.pdf у каталог Downloads поточного

користувача. Зауважимо, що у тест-вимогах рекомендується виконувати посилання на відповідні пункти функціональних вимог.

3.4 Розробка додаткових тестових сценаріїв та об’єднання їх у тестовий набір

Виконаємо розробку ще одного тестового сценарію для перевірки ініціалізації полів застосунку при запуску (п. 1 тест-вимог). Для цього зі списку Tests заголовку бокової панелі вікна Selenium IDE оберіть команду Tests. При цьому в заголовку панелі з’явиться піктограма додання нового тестового сценарію (Рис. 22), при натисканні на яку відкриється діалогове вікно для введення імені нового тестового сценарію. Додайте новий тестовий сценарій, задавши йому ім'я, наприклад, InitTest та введіть необхідні команди для перевірки початкових значень за допомогою автозапису дій (альтернативно, команди можуть бути скопійовані з першого тестового сценарію до поточного стандартними командами Copy і Paste контекстного меню). Вимоги тест-плану пп. 1.1-1.3 щодо перевірки початкових значень полів введення можуть бути реалізовані через збереження командою store value вмісту полів, вказаних їх локаторами у полі Target, до відповідних змінних, імена яких вказуються у полі Value, та порівняння командою Verify значень змінних, вказаних у полі Target, з очікуваними значеннями, вказаними у полі Value (Рис. 23).

У разі необхідності перевірки встановлення радіокнопки необхідно використовувати команду verify checked з локатором у полі Target, наприклад, id=i_scheme5 та значенням Value on для радіокнопки Виплата відсотків в кінці терміну (що реалізовує перевірку п. 1.4 тест-плану) (Рис. 23).

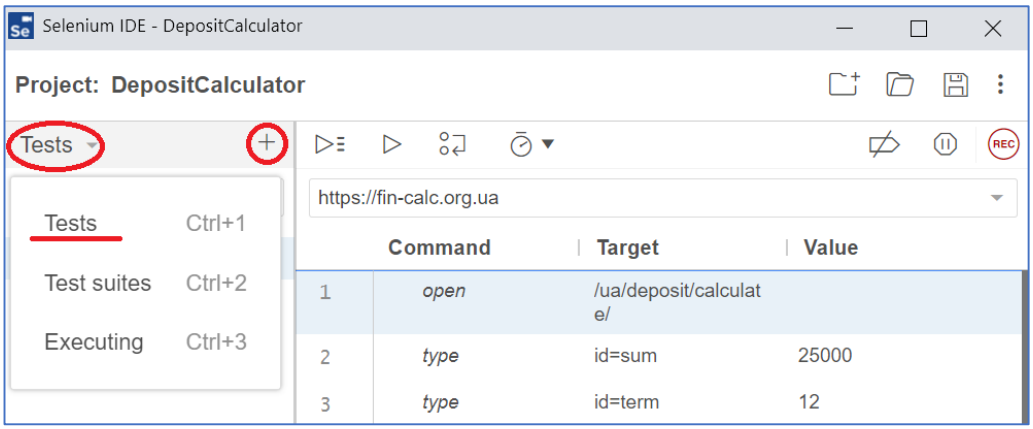


Рис. 22. Додання нового тестового сценарію

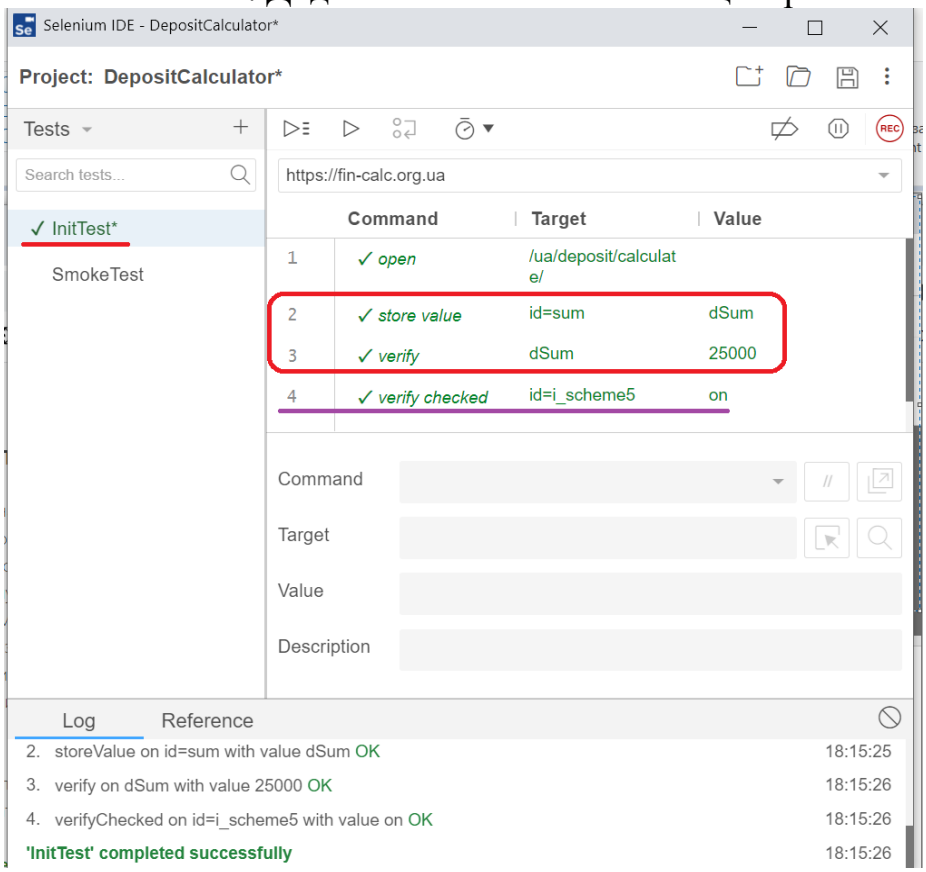


Рис. *23 Перевірка початкових значень у текстовому полі та радіокнопці*

Перевірку однакових значень у декількох Веб-елементах сторінки (пп. 2.1.1- 2.1.3 тест-плану) можна здійснити збереженням текстових значень у двох змінних з подальшим їх порівнянням. Наприклад, для збереження значення, виведеного у підблоці Базові умови блоку Результати у полі Сума депозиту, необхідно виконати клік правою кнопкою миші по цьому полю і обрати із контекстного меню команду Store–Text (Рис. 24). При цьому у діалоговому вікні,

яке з’явиться, необхідно вказати ім'я змінної, до якої збережеться значення з цього Веб-елементу (наприклад, dSumRes).

Далі командою verify потрібно порівняти значення змінної dSumRes зі значенням попередньо збереженої змінної dSum з початковою сумою депозиту з поля введення даних (Рис. 25). Ім'я змінної dSumRes у полі Target цієї команди є посиланням, з якого автоматично береться значення, а у полі Value повинно вказуватися саме значення змінної, його можна отримати, записавши змінну у вигляді ${dSum}.

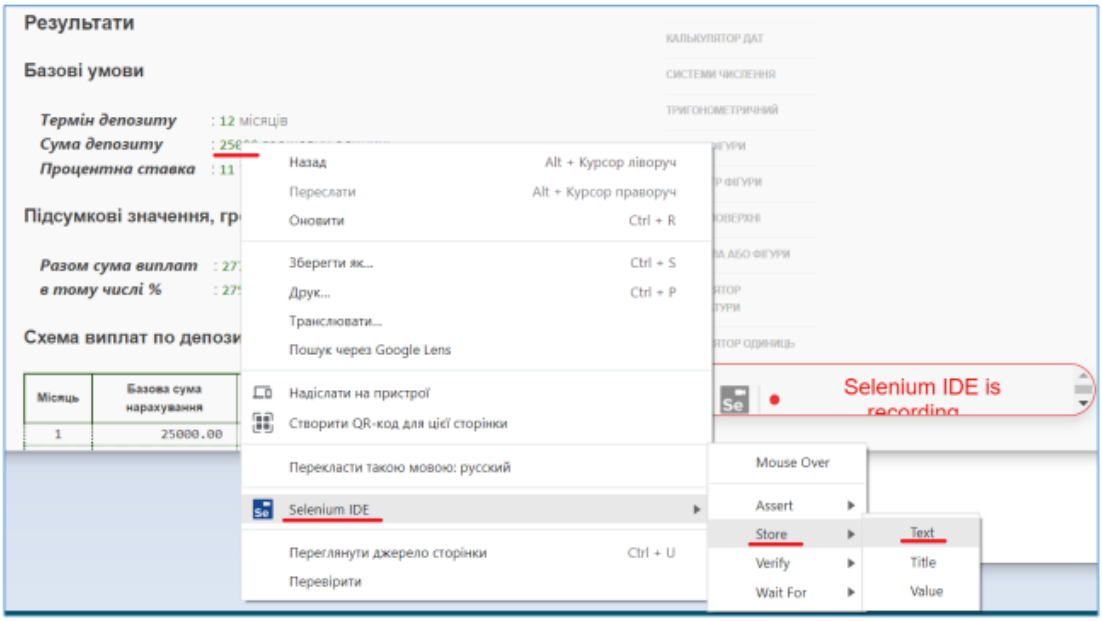


Рис. *24 Збереження текстового значення у змінну та порівняння значень двох змінних*

Значення змінної можна виводити, наприклад для перевірки, до вікна журналу командою echo, в полі Target відповідної команди необхідно ввести необхідне повідомлення та вивести значення змінної вище вказаним способом, наприклад, Сума депозиту складає ${dSum} (Рис. 25).

Для перевірки функціональності друку результатів розрахунку (п. 2.4.1 тест-вимог) створимо новий тестовий сценарій. Його можна створити дублюванням попереднього (командою Duplicate контекстного меню) з

подальшим редагуванням назви, наприклад, у PrintTest та видаленням/редагуванням команд скрипту.

Для перевірки відкриття додаткової сторінки (п. 2.4..1.1 тест-вимог) можна використати команду click зі значенням Target, що дорівнює значенню-рядку тега сторінки, яка відкривається (вона введеться автоматично у режимі запису).

Перевірку відкритої сторінки можна здійснити перевіркою за допомогою команди verify text наявності у певному елементі (тег <h1>) визначеного у тест-вимозі 2.4.1.2 текстового рядка, але ми продемонструємо можливість використання для цього команди verify title зі значенням Target, що дорівнює відповідному значенню тега <title>.

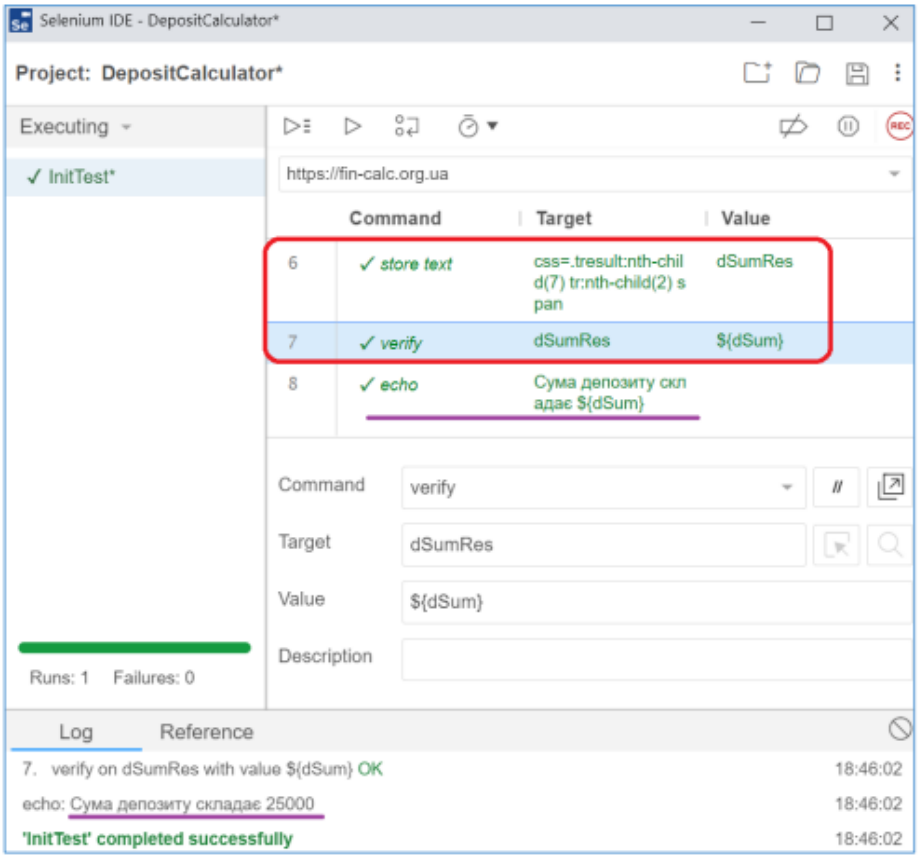


Рис. *25 Команди збереження текстового значення у змінну, порівняння значень двох змінних та виведення значення змінної у журнал*

Для вибору такої команди необхідно у режимі запису виконати клік правою кнопкою миші на довільному місці сторінки та обрати з контекстного меню команду Verify–Title. До скрипту буде додана команда verify title зі

значенням у полі Target тега <title> поточної сторінки (Рис. 26). Зверніть увагу на команду select window зі значенням у полі Target handle=${win7346}, яка обирає вікно за допомогою його локатора (handle) (при записі його значення визначається автоматично). Після вибору вікна такою командою всі подальші команди скрипту будуть стосуватися цього вікна.

Створені тестові сценарії можуть бути об'єднані як тестовий набір (Test Suite), для створення якого необхідно обрати на панелі зі списку команду Test suites (Рис. 27, а), після чого виконати клік на трьох вертикальних крапках біля назви тестового набору за замовчуванням Default Suite. При цьому відкриється вікно, яке дозволяє додавати розроблені тестові сценарії до тестового набору (Рис. 27, б).

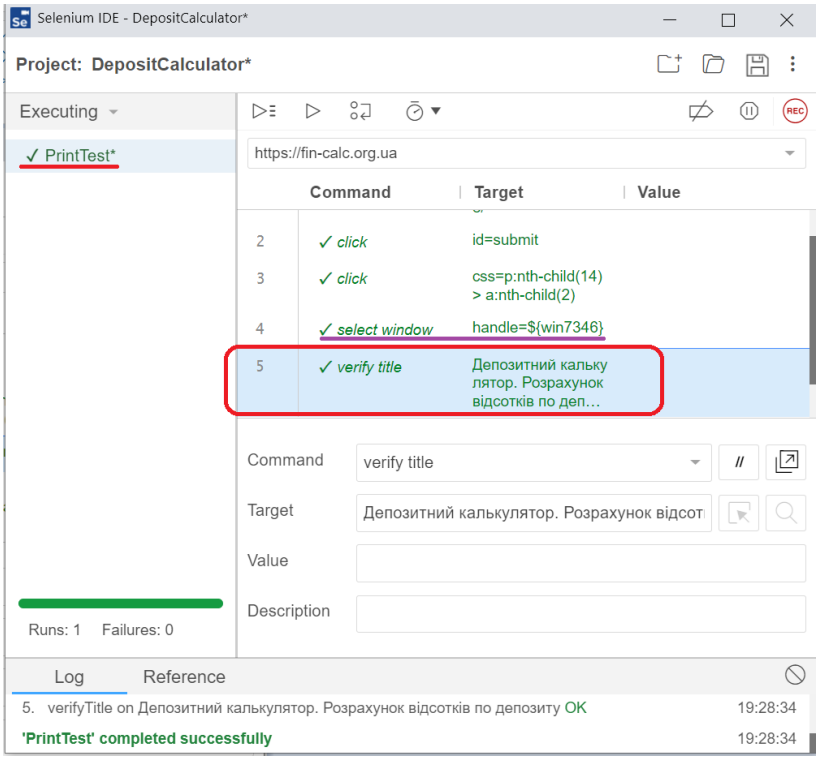
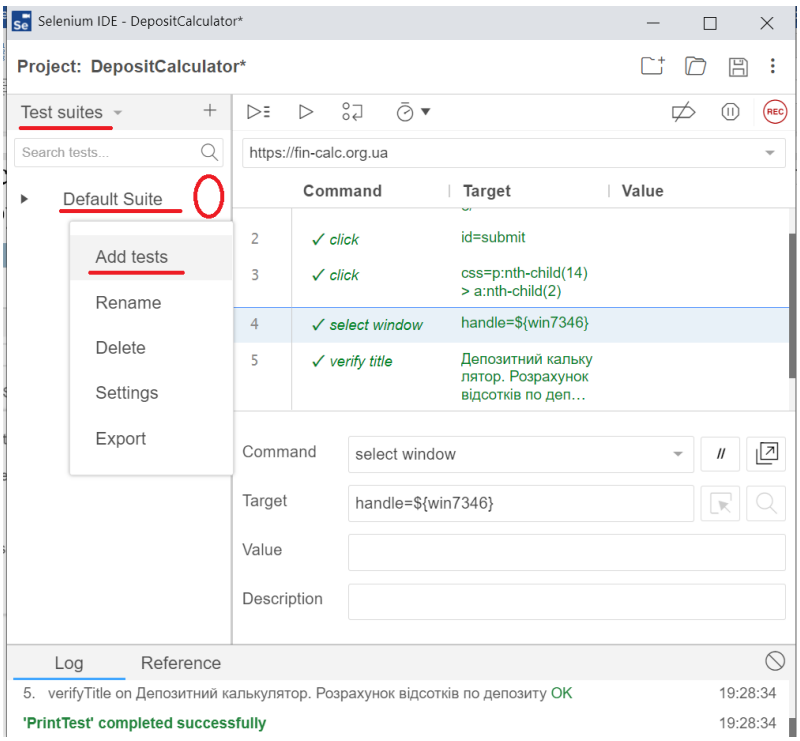
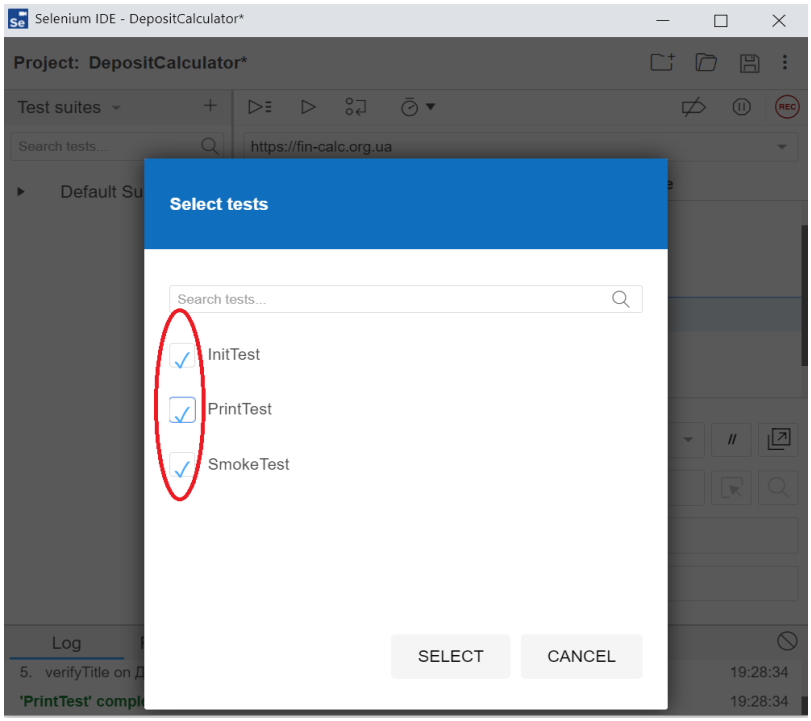


Рис. *26 Відкриття нової сторінки та перевірка її заголовку (<title>)*



а)



б)

Рис. *27 Створення тестового набору*

Змінити назву тестового набору за замовчуванням можливо командою Rename контекстного меню. Запустити тестовий набір можна кнопкою Run all

tests (Запустити всі тести) в режимі роботи з тестовими наборами (Test suites) (Рис. 28).

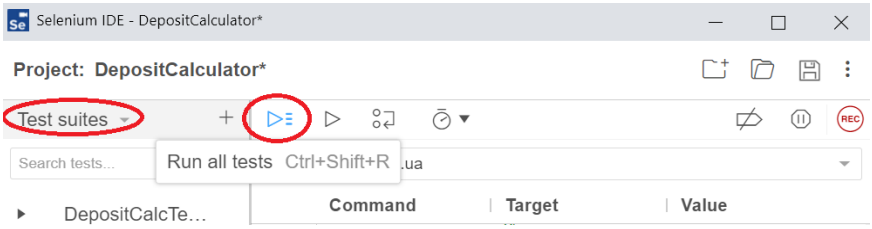
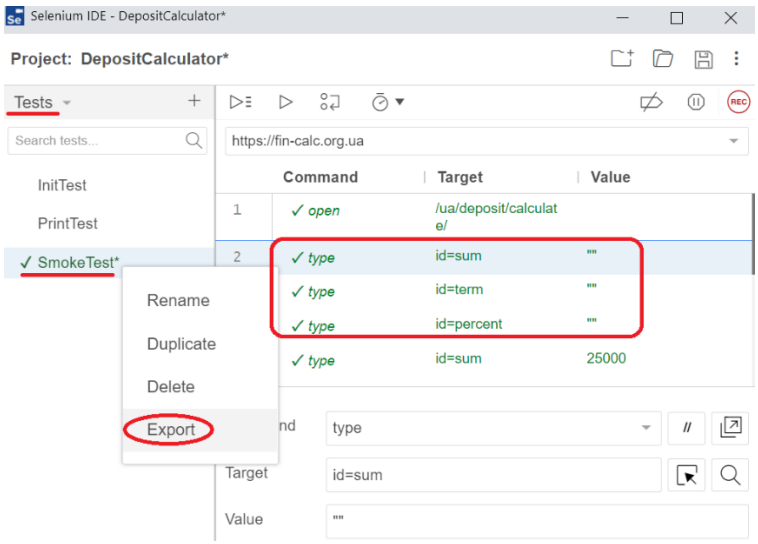


Рис. *28 Запуск тестового набору*

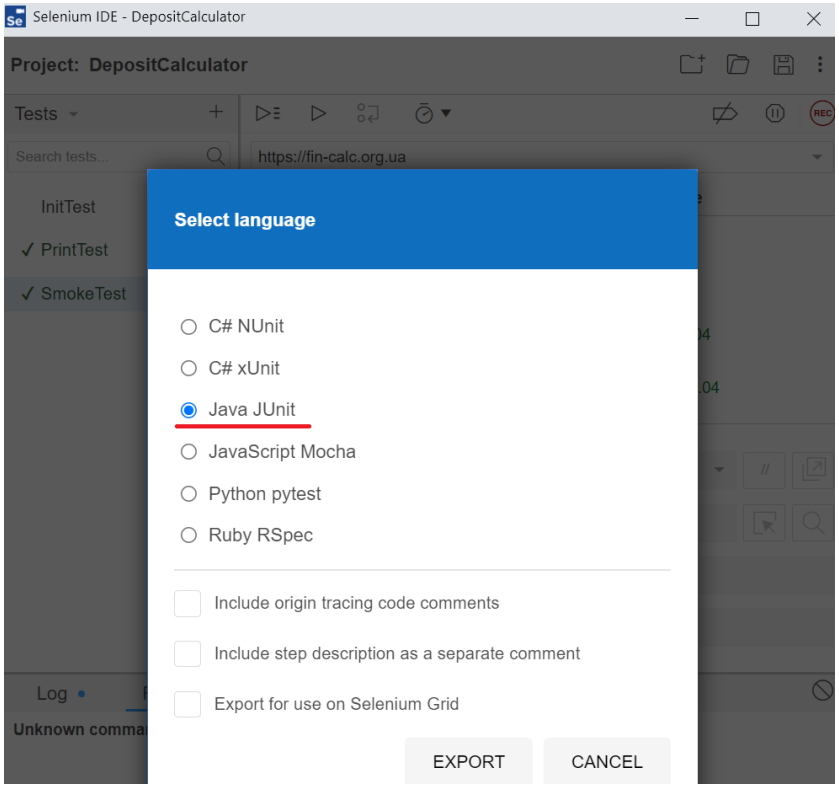
3.5 Експортування тестового скрипту Selenium IDE до Maven-проекту із модульними тестами JUnit 5 із використанням Selenium WebDriver API

Для експорту тестового скрипту Selenium IDE необхідно обрати команду Export із контекстного меню скрипту (Рис. 29, а). На Рис. 29, б показані підтримувані для експорту мови програмування та тестові фреймворки. Після клікання на кнопці EXPORT з'явиться вікно збереження файлу із вихідним кодом на Java та використанням засобів JUnit (ми обрали для нього ім'я SmokeTest.java). Зверніть увагу, що до скрипту додані команди, які попередньо очищують поля введення суми, терміну та проценту за депозитом.

Експортований файл буде доданий до проєкту, який необхідно створити. Нехай це буде проєкт Maven з назвою DepositCalculator, створимо його у Intellij IDEA. Для цього необхідно створити каталог з назвою проєкту та після запуску Intellij IDEA обрати команду New Project (або File–New Project), далі обрати у списку типів проєктів – Maven, погодившись з пропонованою версією JDK, або обравши зі списку необхідну, та клікнути по кнопці Next. У наступному вікні вкажіть назву та оберіть каталог проєкту, а також зазначте дані артефакту, після чого зробіть клік по кнопці Finish.



а)



б)

Рис. *29 Експорт тестового скрипту до файлу з вихідним кодом JUnit*

Додайте до файлу pom.xml проєкту розділи залежностей та плагінів та наступні залежності та плагіни, показані у лістингу:

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

```
    <groupId>ua.edu.znu</groupId>
    <artifactId>DepositCalculator</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <junit.version>5.8.2</junit.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.1.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>io.github.bonigarcia</groupId>
            <artifactId>webdrivermanager</artifactId>
            <version>5.1.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <!--Use for Test Suite-->
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-suite-engine</artifactId>
            <version>1.8.2</version>
            <scope>test</scope>
        </dependency>
        <!--Use for Parameterized Tests-->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.36</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Use for tests run from command prompt by mvn clean install -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M5</version>
            </plugin>
        </plugins>
    </build>
</project>
```

Залежність selenium-java додає до проєкту бібліотеку з засобами Selenium на мові програмування Java, залежність webdrivermanager – розроблену Boni Garcia бібліотеку для автоматичного управління (тобто завантаження, налаштування та обслуговування) драйверами браузерів, необхідними для Selenium WebDriver (наприклад, chromedriver, geckodriver, msedgedriver тощо) [7], залежність slf4jsimple (реалізація Simple Logging Facade for Java) – є дочірньою залежністю Selenium WebDriver та використовується для організації внесення повідомлень у робочий журнал (log) проєкту. Інші залежності та плагін були розглянуті у попередніх змістових модулях.

Наступним кроком необхідно у каталозі src/test/java проєкту створити пакет Java (команда New Package контекстного меню), наприклад, ua.edu.znu.depositcalculator та скопіювати у цей пакет файл з вихідним кодом тесту, експортованого з Selenium IDE.

На жаль, на цей час Selenium IDE виконує експорт тестового скрипту з використанням засобів JUnit версії 4.12, тому необхідно виконати деякі зміни для підвищення версії фреймворку до JUnit 5, а також для використання можливостей WebDriver Manager:

```
package ua.edu.znu.depositcalculator;

// Generated by Selenium IDE
import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

import java.util.HashMap;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class SmokeTest {

    private WebDriver driver;
    private Map<String, Object> vars;
    JavascriptExecutor js;

    @BeforeAll
    public static void setupClass() {
```

```
        WebDriverManager.chromedriver().setup();
    }

    @BeforeEach
    public void setUp() {
        driver = new ChromeDriver();
        js = (JavascriptExecutor) driver;
        vars = new HashMap<>();
    }

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

    @Test
    public void smokeTest() {
        driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");
        WebElement sum = driver.findElement(By.id("sum"));
        WebElement term = driver.findElement(By.id("term"));
        WebElement interest = driver.findElement(By.id("percent"));
        sum.clear();
        term.clear();
        interest.clear();
        sum.sendKeys("25000");
        term.sendKeys("12");
        interest.sendKeys("11");
        driver.findElement(By.id("i_scheme5")).click();
        driver.findElement(By.id("submit")).click();
        String interestAmount = driver.findElement(By.cssSelector(".finale >
.right:nth-child(3)")).getText();
        String sumWithInterst = driver.findElement(By.cssSelector(".finale >
.right:nth-child(4)")).getText();
        assertEquals("2750.04", interestAmount);
        assertEquals("27750.04", sumWithInterst);
    }
}
```

Розглянемо код тесту. У якості змінних класу використовуються WebDriver driver – драйвер браузера, у якому буде тестуватись застосунок, Map vars – колекція для пар ключ-значення, які використовуються у тесті, JavascriptExecutor js – об’єкт, що може виконувати код JavaScript.

У методі setupClass(), який запускається перед виконанням всіх тестів класу (анотація @BeforeAll), використовуються можливості WebDriverManager для пошуку в Інтернет необхідної версії драйвера браузера (у даному випадку Chrome), яка буде працювати з версією браузера, встановленого на комп’ютері, завантаження та підключення до проєкту такого драйвера.

У методі setUp(), який запускається перед кожним тестом класу (анотація @BeforeEach), створюються об’єкти змінних класу, знову ж для управління

браузером Chrome, а у методі tearDown(), який запускається після кожного тесту класу (анотація @AfterEach), об’єкт драйвера видаляється.

Метод smokeTest() містить тест основної функціональності, експортований з Selenium IDE. Команда open Selenium IDE перетворена у виклик методу WebDriver void get(String url):

driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");

Команди Selenium IDE, які виконували очищення полів введення даних, були замінені викликом метода WebDriver WebElement findElement(By by), який приймає як аргумент об’єкт By, що визначає тип локатора Веб-елементу:

driver.findElement(By.id("sum")).sendKeys("\"\"");

але WebDriver має метод void clear(), який виконує очищення елементу Веб-сторінки, і є, на наш погляд, більш доречним.

Для уникнення дублювання коду пошуку елементів Веб-сторінки (об’єктів WebElement) методами WebElement findElement(By by), були створені відповідні локальні змінні. Команда type Selenium IDE при експорті перетворена у виклик методу void sendKeys(CharSequence... keysToSend) об’єкта WebElement, а команда click – у виклик однойменного методу void click() об’єкта WebElement. Також наявні часто використовувані методи void submit() – використовується для Веб-елемента форма та відправляє дані форми, та String getText() – повертає текст, що знаходиться у Веб-елементі, String getAttribute(String name) – повертає значення атрибуту name Веб-елемента.

Фактичні результати – сума відсотків за депозитом та сума депозиту з урахуванням відсотків розміщена у комірках HTML-таблиці, які не мають ідентифікаторів або імен, тому для пошуку цих елементів використаний метод пошуку за селектором CSS: driver.findElement(By.cssSelector(String cssSelector). Завершується код тесту двома ствердженнями JUnit 5, які порівнюють очікуваний результат розрахунку з фактичним.

Selenium WebDriver підтримує вісім основних типів локаторів Веб-елементів, їх назви, методи об’єкта By, що їх забезпечують та відповідні команди Selenium IDE наведені у таблиці 3 [2].

Табл. *3*

Типи локаторів Веб-елементів, які підтримує Selenium

№	Тип локатора	Опис	Метод об'єкта By	Команда Selenium IDE
1	Ім'я тега	Ім'я HTML тега	`By.tagName(String tagName)`	
2	Текст посилання	Повне текстове значення, яке відображається посиланням (тегом <a>)	`By.linkText(String linkText)`	`link="Print"`
3	Частина тексту посилання	Частина текстового значення, яке відображається посиланням (тегом <a>)	`By.partialLinkText(String partialLinkText)`	
4	Ім'я	Значення атрибуту `name` різних тегів	`By.name(String name)`	`name="sum"`
5	Ідентифікатор	Значення атрибуту `id` різних тегів	`By.id(String id)`	`id="sum"`
6	Ім'я класу	Значення атрибуту `class` різних тегів	`By.className(String className)`	
7	Селектор CSS	Шаблони CSS, які відповідають рекомендаціям W3C Selectors	`By.cssSelector(String cssSelector)`	`css=tag#id` `css=tag.class`
8	Значення XPath	XPath – це стандартна мова запитів W3C для вибору вузлів із XML-подібного документа	`By.xpath(String xpathExpression)`	`xpath=//input[@id='i_5']`

Додамо до проєкту ще один тестовий клас – InitTest.java (він реалізує тест-вимоги пп. 1.1-1.4), код якого отримаємо експортом з Selenium IDE з подальшими правками, аналогічними вище описаним.

```
package ua.edu.znu.depositcalculator;

// Generated by Selenium IDE
import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

import java.util.HashMap;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class InitTest {

    private WebDriver driver;
    private Map<String, Object> vars;
    JavascriptExecutor js;

    @BeforeAll
```

```
    public static void setupClass() {
        WebDriverManager.chromedriver().setup();
    }

    @BeforeEach
    public void setUp() {
        driver = new ChromeDriver();
        js = (JavascriptExecutor) driver;
        vars = new HashMap<String, Object>();
    }

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

    @Test
    public void initTest() {
        driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");
        vars.put("dSum",
                 driver.findElement(By.id("sum")).getAttribute("value"));
        assertEquals(vars.get("dSum").toString(), "25000");
        assertTrue(driver.findElement(By.id("i_scheme5")).isSelected());
        driver.findElement(By.id("submit")).click();
        vars.put("dSumRes", driver.findElement(By
                 .cssSelector(".tresult:nth-child(7) tr:nth-child(2)
span"))
                 .getText());
        assertEquals(vars.get("dSumRes").toString(),
                     vars.get("dSum").toString());
        System.out.println("Сума депозиту складає "
                    + vars.get("dSum").toString());
    }
}
```

Команда Selenium IDE store text перетворена у виклик метода put, який додає до колекції з парами ключ-значення vars ім’я змінної, що зберігається та значення параметру команди Value для Веб-елемента поля введення суми депозиту, знайденого за ідентифікатором. Тест демонструє використання колекції vars для збереження та використання змінних, введених до тестового скрипту.

Виконаємо розробку ще одного тесту, який буде виконувати тестування за тест-вимогами підпунктів п. 2 з врахуванням обраних вхідних значень, наведених у таблиці 2. У таблиці 4 наведені вхідні дані та очікувані результати.

Табл. *4*

Вхідні дані та очікувані результати для тестування

Вхідні дані	**Сума**	25000	-1	0	-1	1	-1	25000	-1
	Термін	12	1	-1	0	255	12	256	-1
	Процент	11	0.000001	0.0000011	1000.0	-1.0	0	0.000000999999999999999	1000.0000000000001
Капіталізація відсотків (щомісячна)	**Кількість виплат на рік**	12	css=#errors > p Введені значення для розрахунку некоректні. Поля з помилками виділені кольором. Виправте дані в зазначених полях і повторіть розрахунок.						
	Сума з відсотками	**27892.97**							
	Сума відсотків	**2892.97**							
Капіталізація відсотків (щоквартальна)	**Кількість виплат на рік**	4							
	Сума з відсотками	**27865.54**							
	Сума відсотків	**2865.54**							
Без капіталізації	**Кількість виплат на рік**	12							
	Сума з відсотками	27750.04							
	Сума відсотків	2750.04							

Частина вхідних даних побудована відповідно до методики таблиць рішень для 3 умов – значень полів суми, терміну та проценту за депозитом побудовані 2x3 = 8 стовпців зі значеннями, що приводять до виконання розрахунків (аналог значення "True"), – вони виділені зеленим кольором та значеннями, що приводять до виведення повідомлення про помилку (аналог значення "False"). При цьому значення послідовно обиралися із граничних умов, наведених у таблиці 4.

2. Аналіз таблиці 4 показує, що для тестування достатньо одного параметризованого позитивного тесту із 3 наборами параметрів та очікуваних результатів та одного параметризованого негативного тесту з 7 наборами параметрів та однаковим очікуваним результатом – виведенням повідомлення про помилку. Додаймо до проєкту тестовий клас DepositCalculatorTest та реалізуємо у ньому необхідні тести:

```
package ua.edu.znu.depositcalculator;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.HashMap;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class DepositCalculatorTest {

    private WebDriver driver;
    private Map<String, Object> vars;
    JavascriptExecutor js;

    WebElement sum;
    WebElement term;
    WebElement interest;

    @BeforeAll
    public static void setupClass() {
        WebDriverManager.chromedriver().setup();
    }

    @BeforeEach
    public void setUp() {
        driver = new ChromeDriver();
        js = (JavascriptExecutor) driver;
        vars = new HashMap<>();

        driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");
        sum = driver.findElement(By.id("sum"));
        term = driver.findElement(By.id("term"));
        interest = driver.findElement(By.id("percent"));
        sum.clear();
        term.clear();
        interest.clear();
    }

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

    @ParameterizedTest
    @CsvSource({
            "25000, 12, 11, i_scheme1, 2892.97, 27892.97",
            "25000, 12, 11, i_scheme2, 2865.54, 27865.54",
            "25000, 12, 11, i_scheme5, 2750.04, 27750.04"
    })
    public void positiveTest(String dSum, String dTerm, String dInterest,
        String schemeId, String interestAmounExpResult,
        String sumWithInterestExpResult) {
        sum.sendKeys(dSum);
        term.sendKeys(dTerm);
```

```
        interest.sendKeys(dInterest);
        driver.findElement(By.id(schemeId)).click();
        driver.findElement(By.id("submit")).click();
        String interestAmount = driver.findElement(By.cssSelector(".finale >
.right:nth-child(3)")).getText();
        String sumWithInterest = driver.findElement(By.cssSelector(".finale >
.right:nth-child(4)")).getText();
        assertEquals(interestAmounExpResult, interestAmount);
        assertEquals(sumWithInterestExpResult, sumWithInterest);
    }

    @ParameterizedTest
    @CsvSource({
            "-1, 1, 0.000001, i_scheme1",
            "0, -1, 0.0000011, i_scheme2",
            "-1, 0, 1000.0, i_scheme5",
            "1, 255, -1, i_scheme1",
            "-1, 12, 0, i_scheme2",
            "25000, 256, 0.000000999999999999999, i_scheme5",
            "-1, -1, 1000.0000000000001, i_scheme1",
    })
    public void negativeTest(String dSum, String dTerm, String dInterest,
        String schemeId) {
        sum.sendKeys(dSum);
        term.sendKeys(dTerm);
        interest.sendKeys(dInterest);
        driver.findElement(By.id(schemeId)).click();
        driver.findElement(By.id("submit")).click();
        String expErrorMessage = "Введені значення для розрахунку некоректні. "
            + "Поля з помилками виділені кольором. Виправте дані в зазначених "
            + "полях і повторіть розрахунок.";
        String errorMessage = driver.findElement(By.cssSelector("#errors > p"))
                                    .getText();
        assertEquals(expErrorMessage, errorMessage);
    }
}
```

Оскільки обидва тести мають однакову частину операторів – завантаження сторінки, очищення полів введення даних, вони були винесені до методу setUp(), який викликається перед кожним з тестів (анотація @BeforeEach). Тепер залишилось перевірити функціональність друку та збереження у файлах різного типу результатів розрахунку (пп. 2.4.1 – 2.4.6 тестових вимог). Для цього виконаємо експорт з Selenium IDE тестового скрипту PrintTest.

```
package ua.edu.znu.depositcalculator;

// Generated by Selenium IDE
import io.github.bonigarcia.wdm.WebDriverManager;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
```

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class PrintSaveTest {

    private WebDriver driver;
    private Map<String, Object> vars;
    JavascriptExecutor js;

    @BeforeAll
    public static void setupClass() {
        WebDriverManager.chromedriver().setup();
    }

    @BeforeEach
    public void setUp() {
        driver = new ChromeDriver();
        js = (JavascriptExecutor) driver;
        vars = new HashMap<String, Object>();
        driver.manage().window().maximize();
    }

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


    public String waitForWindow(int timeout) {
        try {
            Thread.sleep(timeout);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Set<String> whNow = driver.getWindowHandles();
        Set<String> whThen = (Set<String>) vars.get("window_handles");
        if (whNow.size() > whThen.size()) {
            whNow.removeAll(whThen);
        }
        return whNow.iterator().next();
    }

    @Test
    public void printTest() {
        driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");
        driver.findElement(By.id("submit")).click();
        vars.put("window_handles", driver.getWindowHandles());
        driver.findElement(By.cssSelector("p:nth-child(14) > a:nth-
child(2)")).click();
        vars.put("win7346", waitForWindow(2000));
        driver.switchTo().window(vars.get("win7346").toString());
        assertEquals("Депозитний калькулятор. Розрахунок відсотків по депозиту",
driver.getTitle());
    }

    @ParameterizedTest
```

```
    @CsvSource({
            "'p:nth-child(15) > a:nth-child(2)', calculation-on-deposit.xls",
            "'p:nth-child(16) > a:nth-child(2)', calculation-on-deposit.xlsx",
            "'p:nth-child(17) > a:nth-child(2)', calculation-on-deposit.html",
            "'p:nth-child(18) > a:nth-child(2)', calculation-on-deposit.csv",
            "'p:nth-child(19) > a:nth-child(2)', calculation-on-deposit.pdf"
    })
    public void SaveTest(String cssLocator, String filename) {
        Path downloadPath = Paths.get(System.getProperty("user.home"),
"Downloads", filename);
        driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");
        driver.findElement(By.id("submit")).click();
        vars.put("window_handles", driver.getWindowHandles());
        driver.findElement(By.cssSelector(cssLocator)).click();
        vars.put("win7346", waitForWindow(2000));
        driver.switchTo().window(vars.get("win7346").toString());
        assertTrue(Files.exists(downloadPath));
    }
}
```

У наведеному тестовому класі (він перейменований у PrintSaveTest) розроблені два тестових методи. Перший – printTest() – отриманий експортом з Selenium IDE – не зазнав жодних змін. Зверніть увагу, яким чином замінена команда скрипту select window з параметром handle (ідентифікатором вікна браузера) а також на метод введення затримки у 2 секунди для завантаження нового вікна браузера (це все створено автоматично при експорті скрипту із Selenium IDE).

Другий тест – параметризований SaveTest(String cssLocator, String filename) був доданий вручну. Він перевіряє наявність завантажених файлів у каталозі Downloads профілю користувача усіх підтримуваних типів із 37 розрахунками доходів від депозиту. Єдина вимога – перед запуском тесту потрібно впевнитись, що файлів з ім’ям calculation-on-deposit у цьому каталозі немає.

Зверніть увагу на доданий автоматично при експорті метод String waitForWindow(int timeout), який, зокрема, забезпечує призупинку потоку виконання на час тайм-ауту (ця частина коду виділена жирним шрифтом). Таке штучно введене очікування часто є необхідним для клієнт-серверних систем, оскільки у разі перевантаження серверу, браузер може ще не завантажити певний Веб-елемент, з яким працюють оператори кода тесту (зазвичай при цьому виникає помилка org.openqa.selenium.NoSuchElementException: Unable to locate element). Тим не менш, використаний при експорті підхід не вважається

оптимальним, оскільки Selenium WebDriver забезпечує можливості використання інших, більш ефективних стратегій очікування: неявне (implicit), явне (explicit) та плавне (fluent) очікування (waits) [6].

Неявне (implicit) очікування забезпечується викликом ланцюжка методів: driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(timeout)) і цей один оператор може замінити виділений код у методі String waitForWindow(int timeout). Однак ця стратегія очікування не рекомендується, оскільки, по-перше, вона працює лише при пошуку елементів, по-друге, її реалізація залежить від драйвера (тобто залежна від типу браузера). Нарешті, оскільки неявні очікування застосовуються глобально, перевірка відсутності Веб-елементів зазвичай збільшує час виконання всього сценарію.

Стратегія явного (explicit) очікування дозволяє призупинити виконання тесту, доки не відбудеться певна умова впродовж часу тайм-ауту. Ця стратегія передбачає створення об’єкту WebDriverWait, використовуючи об’єкт WebDriver як перший аргумент конструктора і екземпляр Duration як другий аргумент (щоб вказати максимальний час очікування). Після цього пошук Вебелементу можна виконувати методом T until(Function isTrue) інтерфейса Wait, що приймає об’єкт ExpectedCondition, який шукає Веб-елемент за тим, чи іншим типом локатора впродовж часу, що не перевищує тайм-аут, заданий для об’єкта WebDriverWait [8].

Продемонструємо використання цієї стратегії на прикладі тесту SmokeTest. Дійсно, після клікання на посиланні ПОРАХУВАТИ завантажується додаткова частина сторінки із результатами розрахунку і, у разі великого завантаження сервера, може скластись ситуація, що Веб-елементи з сумою відсотків та з загальною сумою депозиту разом з відсотками, які перевіряються, можуть бути не завантаженими. Тому виконаємо рефакторинг методу smokeTest(), щоб передбачити таку ситуацію:

```
@Test
public void smokeTest() {
    driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");
    WebElement sum = driver.findElement(By.id("sum"));
    WebElement term = driver.findElement(By.id("term"));
    WebElement interest = driver.findElement(By.id("percent"));
```

```
        sum.clear();
        term.clear();
        interest.clear();
        sum.sendKeys("25000");
        term.sendKeys("12");
        interest.sendKeys("11");
        driver.findElement(By.id("i_scheme5")).click();
        driver.findElement(By.id("submit")).click();

        /*Explicit waits*/
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(2));
        WebElement interestAmount = wait.until(ExpectedConditions
                .presenceOfElementLocated(By.cssSelector(".finale > .right:nth-
child(3)")));
        WebElement sumWithInterest = wait.until(ExpectedConditions
                .presenceOfElementLocated(By.cssSelector(".finale > .right:nth-
child(4)")));

//        String interestAmount = driver.findElement(By.cssSelector(".finale >
.right:nth-child(3)")).getText();
//        String sumWithInterest = driver.findElement(By.cssSelector(".finale >
.right:nth-child(4)")).getText();
        assertEquals("2750.04", interestAmount.getText());
        assertEquals("27750.04", sumWithInterest.getText());
    }
```

Тепер при завантаженні сервер пошук Веб-елементів із результатами розрахунку буде призупинений, поки вони не завантажаться до DOM-графу елементів Веб-сторінки (або поки не сплине 2 секунди).

Існує декілька типів об’єктів ExpectedCondition: окрім ExpectedCondition, який використовується для очікування Веб-елементів, знайдених за локатором, існують ExpectedCondition – повертає об’єкт WebDriver, який буде очікувати певних умов, наприклад, завантаження фрейму перед перемиканням на нього, ExpectedCondition – повертає true або false перевіряючи клікабельність елементів, чи є елемент виділеним, наявність певного тексту у елементі тощо та ExpectedCondition – повертає об’єкт Alert (вікно сповіщення, що виникає), якщо на сторінці є вікно типу Alert та перемикає об’єкт WebDriver перемикається у вікно сповіщення. Класифікація типів ExpectedCondition разом з найчастіше використовуваними прикладами їх використання наведена у [8].

Стратегія плавного (fluent) очікування, як і явне (explicit) очікування, призупиняє тест до виконання певних умов, але, крім того, надає можливості додаткової конфігурації очікування, основні методи об’єкта FluentWait, використовувані для цього наведені у таблиці 5.

Табл. *5*

Основні методи конфігурації об'єкта FluentWait

Метод	Опис
`FluentWait<T> withTimeout(Duration timeout)`	Встановлює, скільки часу потрібно чекати, доки оцінювана умова стане істинною
`FluentWait<T> pollingEvery(Duration interval)`	Встановлює, як часто потрібно оцінювати стан (500 мс за замовчуванням)
`FluentWait<T> withMessage(final String message)`	Встановлює повідомлення, яке відображатиметься після закінчення часу
`FluentWait<T> ignoring(Class<? extends Throwable> exceptionType)`	Ігнорує зазначене виключення під час очікування умови
`<V> V until(Function<? super T, V> isTrue)`	Забезпечує очікування умови

Виконаємо ще раз рефакторинг методу smokeTest() тестового класу SmokeTest для демонстрації використання плавного (fluent) очікування із конфігурацією його параметрів:

```
    @Test
    public void smokeTest() {
        driver.get("https://fin-calc.org.ua/ua/deposit/calculate/");
        WebElement sum = driver.findElement(By.id("sum"));
        WebElement term = driver.findElement(By.id("term"));
        WebElement interest = driver.findElement(By.id("percent"));
        sum.clear();
        term.clear();
        interest.clear();
        sum.sendKeys("25000");
        term.sendKeys("12");
        interest.sendKeys("11");
        driver.findElement(By.id("i_scheme5")).click();
        driver.findElement(By.id("submit")).click();

        /*Explicit wait
        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(2));
        */

        /*Fluent wait*/
        Wait<WebDriver> wait = new FluentWait<>(driver)
                .withTimeout(Duration.ofSeconds(2))
                .pollingEvery(Duration.ofSeconds(1))
                .ignoring(NoSuchElementException.class);

        WebElement interestAmount = wait.until(ExpectedConditions
                .presenceOfElementLocated(By.cssSelector(".finale > .right:nth-
child(3)")));
        WebElement sumWithInterest = wait.until(ExpectedConditions
                .presenceOfElementLocated(By.cssSelector(".finale > .right:nth-
child(4)")));

//        String interestAmount = driver.findElement(By.cssSelector(".finale >
.right:nth-child(3)")).getText();
//        String sumWithInterest = driver.findElement(By.cssSelector(".finale >
.right:nth-child(4)")).getText();
        assertEquals("2750.04", interestAmount.getText());
        assertEquals("27750.04", sumWithInterest.getText());
    }
```

Створений об’єкт очікування FluentWait сконфігурований на час очікування до 2 секунд, упродовж якого через секунду буде перевірятись виконання умови і, якщо Веб-елемент при цьому не буде знайдений, буде ігноруватись виникнення виключення NoSuchElementException.

Останнім кроком додамо усі розроблені тести, окрім SmokeTest, до тестового набору DepositCalulatorTestSuite:

```
package ua.edu.znu.depositcalculator;

import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.Suite;

@Suite
@SelectClasses({InitTest.class, DepositCalculatorTest.class,
PrintSaveTest.class})
public class DepositCalulatorTestSuite {
}
```

Запуск цього тестового набору призведе до повного тестування Веб-застосунку відповідно до тестових вимог.

ВИСНОВКИ

1. У процесі виконання кваліфікаційної роботи були опрацьовані джерела інформації за темою тестування Веб-застосунків та технологія Selenium.
2. Вивчений склад фреймворку Selenium та його архітектурні рішення.
3. Проведено дослідження Веб-застосунку «Депозитний калькулятор» та визначені його функціональні вимоги.
4. Розроблені тест-вимоги із використанням методики класів еквівалентності та аналізу граничних значень.
5. На основі тест-вимог виконаний запис тестових сценаріїв у Selenium IDE з подальшим експортом до проєкту Java з тестовим фреймворком JUnit 5.
6. Виконана прогонка тестів, які тестують функціональні можливості застосунку, такі як розрахунок доходів за депозитом та збереження/дпук результатів розрахунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Garcia B. Hands-on Selenium WebDriver with Java. O'Reilly Media, 2022. 647 p.
2. Zhimin Zhan. Selenium WebDriver Recipes in Java. The problem solving guide to Selenium WebDriver in Java. Lean Publishing, 2021. 169 p.
3. The Selenium Browser Automation Project. Documentation. URL: https://www.selenium.dev/documentation/ (дата звернення 25.05.2023).
4. WebDriver. W3C Working Draft. URL: https://www.w3.org/TR/webdriver/ (дата звернення 25.05.2023).
5. Складні відсотки. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Складні_відсотки. (дата звернення 15.06.2023).
6. Control Flow. Selenium IDE Docs. URL: https://www.selenium.dev/selenium-ide/docs/en/introduction/control-flow . (дата звернення 17.06.2023).
7. Boni García. WebDriverManager. URL: https://bonigarcia.dev/webdrivermanager/ (дата звернення 19.06.2023).
8. Sheth H. What Is ExpectedConditions In Selenium (With Examples). LambdaTest Site. URL: https://www.lambdatest.com/blog/expected-conditions-in-selenium-examples/ (дата звернення 19.06.2023).