НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Фреймворк для автоматизованого тестування програмного забезпечення»

Виконав:
студент IV курсу, групи КП-12,
Якубишин Анатолій Сергійович _____

Керівник:
доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри СПіСКС, к.т.н., доцент,
Тарасенко-Клятченко Оксана Володимирівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент_____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Якубишину Анатолію Сергійовичу

1. Тема проєкту «Фреймворк для автоматизованого тестування програмного забезпечення», керівник проєкту Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджені наказом по університету № 1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - огляд та аналіз існуючих рішень;
   - обґрунтування вибору засобів реалізації;
   - структурно-алгоритмічна організація системи;
   - особливості програмної реалізації фреймворку.

5. Перелік обов'язкового графічного матеріалу:
   - функціональність фреймворку (креслення);
   - алгоритм ініціалізації драйверу (креслення);
   - архітектура фреймворку (плакат);
   - діаграма взаємозв'язків між сутностями (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	16.11.2024	
2.	Розроблення та узгодження технічного завдання	27.11.2024	
3.	Розроблення структури фреймворку	15.12.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	28.12.2024	
5.	Розроблення дизайну вікон та графічних елементів	05.02.2025	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2025	
7.	Програмна реалізація фреймворку	15.03.2025	
8.	Тестування фреймворку	18.03.2025	
9.	Підготовка матеріалів третього розділу дипломного проєкту	27.03.2025	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	29.04.2025	
11.	Підготовка графічної частини дипломного проєкту	15.05.2025	
12.	Оформлення документації дипломного проєкту	27.05.2025	

Студент Анатолій ЯКУБИШИН

Керівник проєкту Тетяна ЗАБОЛОТНЯ

АНОТАЦІЯ

У дипломному проєкті розроблено фреймворк для автоматизованого тестування програмного забезпечення, що поєднує у собі консольний інтерфейс та графічну оболонку, створену за допомогою JavaFX та основну частину фреймворку. Основна частина фреймворку містить набір засобів для тестування WEB, Android та API систем з використанням принципів чистого коду та перевикористанням логіки. Основною метою є підвищення ефективності аналізу результатів тестування та виявлення спільних помилок, які можуть повторюватися в різних сценаріях.

Фреймворк дозволяє виконувати операції через командний рядок, що забезпечує гнучкість налаштування та можливість інтеграції з інструментами безперервної інтеграції. Графічна частина застосунку надає зручний інтерфейс для перегляду результатів виконання тестів і автоматизованого групування помилок за схожими ознаками. Це дозволяє суттєво зменшити час на діагностику дефектів та полегшує роботу інженерів з якості програмного забезпечення.

У рамках роботи спроєктовано архітектуру системи, реалізовано базові модулі для обробки та візуалізації тестів, а також розроблено структуру зберігання інформації про виконання тестів та виявлені помилки. Передбачено можливість подальшого розширення функціоналу та інтеграції з зовнішніми сервісами.

Результатом є багатофункціональний інструмент, здатний підвищити продуктивність тестування та забезпечити глибший аналіз стабільності програмного продукту.

ABSTRACT

This diploma project presents the development of a framework for automated software testing, which combines a command-line interface, a graphical interface built with JavaFX, and the core testing functionality. The core of the framework provides a set of tools for testing web, Android, and API systems, and enables writing tests that follow clean code principles and promote logic reuse. The main objective is to improve the efficiency of analyzing test results and identifying common errors that may appear across different scenarios.

The framework allows operations through the command line, offering flexible configuration and integration with continuous integration (CI) tools. The graphical part of the application provides a convenient interface for viewing test execution results and automatically grouping similar errors. This significantly reduces the time required for defect diagnosis and streamlines the work of software quality engineers.

Within the scope of the project, the system architecture was designed, core modules for processing and visualizing tests were implemented, and a structure for storing test execution data and detected errors was developed. The framework also supports future functionality extensions and integration with external services.

The result is a multifunctional tool capable of increasing testing productivity and providing deeper insights into software product stability.

ДП.045490-01-90 Фреймворк для автоматизованого тестування програмного забезпечення. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045490-02-91	Фреймворк для	5	
	автоматизованого		
	тестування програмного		
	забезпечення. Технічне		
	завдання		
ДП.045490-03-81	Фреймворк для	69	
	автоматизованого		
	тестування програмного		
	забезпечення. Пояснювальна		
	записка		
ДП.045490-04-51	Фреймворк для	4	
	автоматизованого		
	тестування програмного		
	забезпечення. Програма та		
	методика тестування		
ДП.045490-05-34	Фреймворк для	17	
	автоматизованого		
	тестування програмного		
	забезпечення. Керівництво		
	користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045490-06-99	Фреймворк для	1	
	автоматизованого		
	тестування програмного		
	забезпечення.		
	Функціональність		
	фреймворку. UML-діаграма		
	прецедентів		
ДП.045490-07-99	Фреймворк для	1	
	автоматизованого		
	тестування програмного		
	забезпечення. Алгоритм		
	ініціалізації драйверу.		
	Блок-схема алгоритму		
ДП.045490-08-98	Фреймворк для	1	
	автоматизованого		
	тестування програмного		
	забезпечення. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ФРЕЙМВОРК ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Технічне завдання

ДП.045490-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анатолій ЯКУБИШИН

2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: фреймворк для автоматизованого тестування програмного забезпечення.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання ІТ-компаніями для зменшення часу розробки, поліпшення якості тестування. Також ця розробка допоможе забезпечувати актуальність розроблюваного продукту шляхом частих випусків його нових версій.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Фреймворк повинен забезпечувати такі основні функції:

1) підтримка автоматизованого тестування для Web, API, Mobile застосунків;
2) створення, редагування та запуску тестових сценаріїв;
3) генерування звітів про результати тестування;
4) підтримка інтеграції з CI/CD (наприклад, GitHub Actions, Jenkins);
5) можливість зберігання результатів тестів на сервері;

6) серверний аналіз результатів тестування для пошуку спільних помилок в автоматизованих тестах;
7) можливість написання кросплатформних тестів для WEB та Mobile;
8) підтримка шаблону POM з використанням вкладених компонентів;
9) підтримка багатопотокового запуску тестів;
10) підтримка запуску тестів у headless режимі;
11) можливість запускати тести через браузерну ферму (Selenium grid);
12) можливість запускати тести на Google Chrome, FireFox, Edge – на найновіших версіях браузера без Selenium Grid.

Додаткові вимоги:

1) наявність графічного інтерфейсу;
2) можливість створення тестів, що керуються даними;
3) захист від порушення цілісності;
4) захист від несанкціонованого доступу.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   – «Функціональність фреймворку»;
   – «Алгоритм ініціалізації драйверу».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи.....16.11.2024

Розроблення та узгодження технічного завдання 27.11.2024

Розроблення структури фреймворку..... 15.12.2024

Розроблення дизайну вікон та графічних елементів 05.02.2025

Програмна реалізація фреймворку..... 15.03.2025

Тестування фреймворку 18.03.2025

Підготовка матеріалів текстової частини проєкту 29.04.2025

Підготовка матеріалів графічної частини проєкту 15.05.2025

Оформлення технічної документації проєкту..... 27.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2025 р.

ФРЕЙМВОРК ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Пояснювальна записка

ДП.045490-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анатолій ЯКУБИШИН

2025

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення.

HTTP – HyperText Transfer Protocol «протокол передачі гіпертексту» – це протокол прикладного рівня гіпермедійних документів [22].

JS – мова програмування JavaScript.

Newman – це інструмент, котрий надає можливість запускати та тестувати колекції Postman безпосередньо з командного рядка, що дозволяє інтегрувати його з серверами постійної інтеграції [23].

DDT – тести, що керуються даними (Data-driven tests).

ORM (Object-Relational Mapping) – це техніка програмування, яка дозволяє використовувати об'єктно-орієнтовану модель для роботи з реляційними базами даних [24].

JRE – Java Runtime Environment (JRE) – це програмне забезпечення, необхідне для коректної роботи програм Java [28].

Рефакторинг – це процес, який передбачає редагування та очищення раніше написаного програмного коду без зміни його функцій [29].

Юніт-тести (модульні тести) – тестування кожної атомарної функціональності додатку окремо, в штучно створеному середовищі [30].

CRUD – абревіатура, що означає create, read, update, delete.

REST – Representational State Transfer.

БД – база даних.

СКБД – система керування базами даних.

JSON – це текстовий формат обміну даними між комп'ютерами.

XML – стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками.

ВСТУП

У сучасному світі інформаційних технологій ефективне тестування програмного забезпечення відіграє ключову роль у забезпеченні якості та надійності інформаційних продуктів. Традиційні методи ручного тестування виявляються недостатньо ефективними, особливо в умовах багатоплатформенного середовища, де необхідно підтримувати різні типи інтерфейсів – вебзастосунки, мобільні клієнти та API-сервіси. Саме тому автоматизація тестування стає важливою складовою сучасного процесу розробки. Зростання складності програмних систем, необхідність швидкого випуску нових версій, а також підтримка мультиплатформенності (веб, мобільні пристрої, API) обумовлюють актуальність впровадження автоматизованого тестування.

Автоматизоване тестування дозволяє значно зменшити час перевірки працездатності програмних продуктів, забезпечити повторюваність і відтворюваність тестів, а також знизити ризик людських помилок. Завдяки цьому зростає продуктивність команд розробників і тестувальників, а якість кінцевого продукту підвищується. Особливої актуальності набуває створення універсальних рішень, які б забезпечували можливість тестування різних компонентів системи у єдиному середовищі.

У межах цього дипломного проєкту було розроблено фреймворк для автоматизованого тестування, що дозволяє здійснювати тестування вебінтерфейсів, API та мобільних застосунків. Основною метою є не лише спрощення процесу створення та запуску якісних тестів, в основі яких лежать принципи модульності та «чистоти» коду, але й надання можливості їх гнучкого налаштування, зберігання результатів виконання, формування звітів та інтеграції з іншими інструментами розробки, зокрема системами безперервної інтеграції.

1. ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Загальні положення та аналіз предметної області

Автоматизоване тестування програмного забезпечення відіграє ключову роль у забезпеченні якості продукції на сучасному етапі розвитку IT-індустрії. Складність програмних систем зростає з кожним роком, що, в свою чергу, вимагає ефективних, масштабованих і гнучких інструментів тестування. Особливої актуальності набуває автоматизоване тестування веб, API та мобільних додатків, а також інтеграція тестування в процеси CI/CD.

Автоматизоване тестування передбачає використання спеціалізованих програмних засобів для виконання тестових сценаріїв без постійної участі людини. Це має наступні переваги: підвищення швидкості тестування, повторюваність перевірок і оперативне виявлення дефектів. Це має особливе значення у контексті сучасної розробки програмного забезпечення, коли оновлення програмного продукту відбуваються щодня, автоматизація стає не лише бажаною, а й необхідною складовою процесу забезпечення якості.

Але, попри всі переваги, автоматизоване тестування має низку викликів. При зміні логіки або інтерфейсу додатків тести часто виходять з ладу і вимагають значних доопрацювань. Це вимагає постійного оновлення та підтримки тестів, що може бути трудомістким процесом. Крім того, вибір відповідного фреймворку або інструменту для тестування часто ускладнюється специфікою застосунку, його архітектурою та технологічним стеком.

Сьогодні існує велика кількість засобів автоматизації. Кожен з них має переваги та недоліки та охоплює певні аспекти тестування, проте жоден з них не є універсальним рішенням, щоб забезпечувало ефективну автоматизацію для всіх типів програмних продуктів. Це створює потребу в фреймворках, що б об’єднували можливості декількох інструментів,

забезпечували перевикористання логіки для тестів, що перевіряють різні типи додатків та підтримували адаптацію тестів до змін.

Беручи до уваги актуальність задачі, постає потреба у розробці власного фреймворку, що буде не тільки забезпечувати запуск та виконання тестів, але й моніторити результати та групувати помилки для швидкого оновлення тестових сценаріїв.

1.2. Аналіз існуючих рішень

У цьому розділі буде проведено огляд найпоширеніших інструментів та фреймворків, які застосовуються в автоматизованому тестуванні програмного забезпечення. Кожен з них має свої особливості, переваги та обмеження, які варто враховувати при виборі оптимального рішення для конкретного проєкту.

1.2.1. Cypress

Cypress – це фреймворк для автоматизації тестування вебдодатків на мовах JS/TS [1]. З його допомогою користувачі можуть створювати інтерфейсні end-to-end тести та тестувати серверну частину за допомогою API викликів.

Розглянемо функціональність:

– створення end-to-end тестів для web UI;

– створення тестів для API;

– створення тестів, що комбінують API виклики з web UI взаємодією;

– підключення плагінів;

– групування тестової логіки у команди, тобто винесення повторюваної логіки на рівень вище;

– покроковий дебагінг з переглядом вебсторінок після кожної дії;

– запуск тестів в headless режимі.

Платні підписки додатково дозволяють: запускати тести паралельно, аналізувати аномалії в тестах та їх результатах, запускати тести в CI/CD в

cypress cloud та переглядати покрокове відображення сторінок для полегшеного дебагу.

Деякі версії підписок дають змогу інтегруватися з Jira, пріоритезувати тести. На рис. 1.1 наведено стартову сторінку графічного інтерфейсу Cypress.

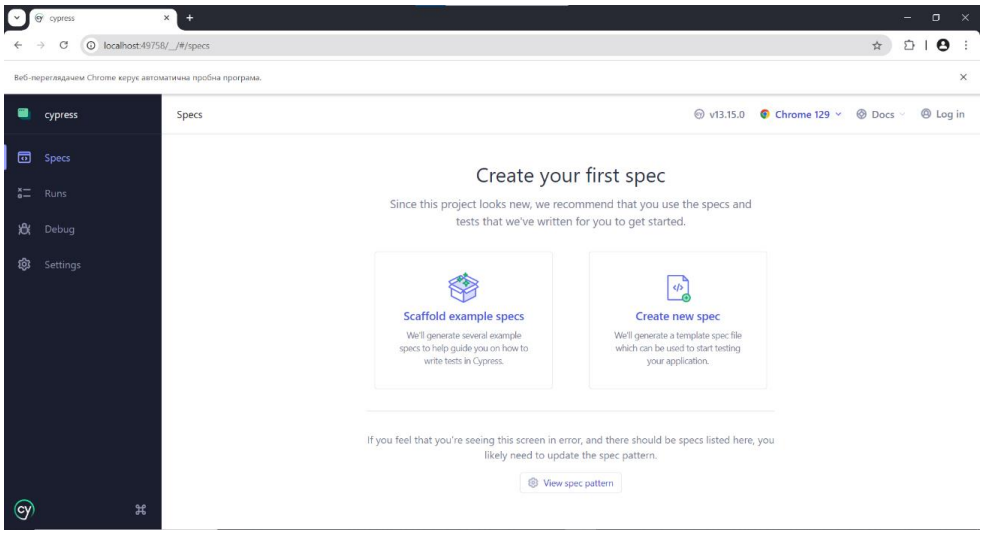


Рис. 1.1. Вікно Cypress для ініціалізації проєкту

На рис. 1.2 наведено вебсторінку головного меню Cypress.

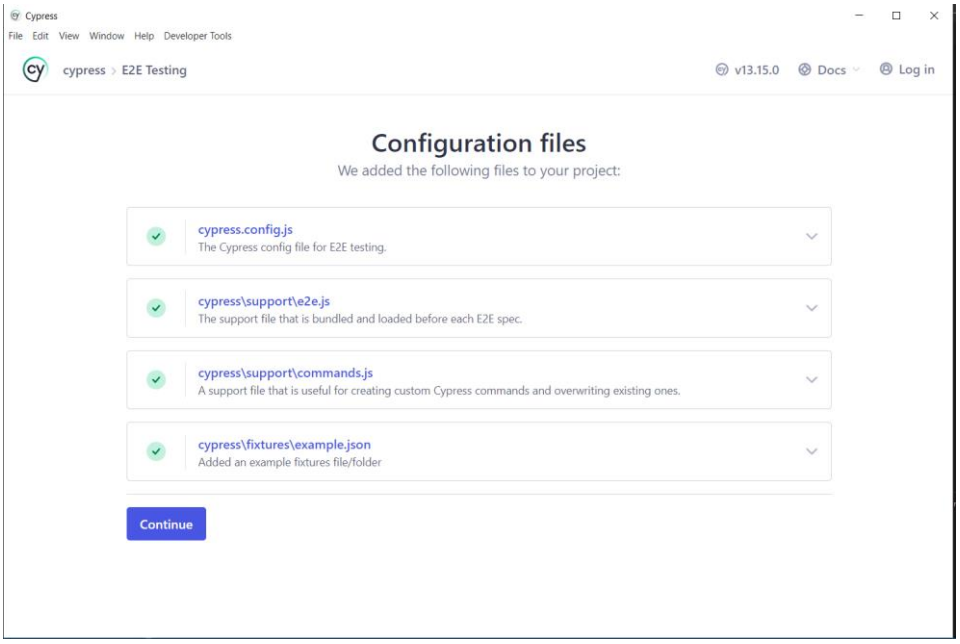


Рис. 1.2. Вебсторінка головного меню Cypress

На рис. 1.3 Наведено сторінка Cypress runner, що містить функціональність покрокового відлагодження тестів.

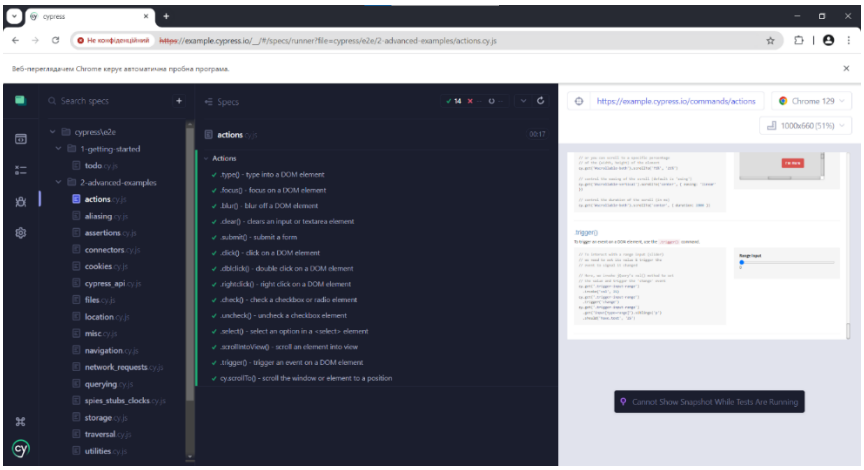


Рис. 1.3. Сторінка Cypress runner для покрокового debug тестових скриптів

Також тести у Cypress можуть запускатися через командний рядок.

1.2.2. Selenide

Selenide – це фреймворк для тестування UI на Java [2]. Він заснований на використанні бібліотеки Selenium та містить спрощений API для автоматизації дій в браузері за допомогою Java.

Фреймворк не містить графічного інтерфейсу. Інтегрується з інструментами юніт-тестування для Java такими як Junit та TestNG або будь-якими іншими. Тому запуск тестів налаштовується відповідно до бібліотек юніт- тестування.

Також фреймворк виправляє проблему з роботою динамічних сторінок, яка присутня при використанні бібліотеки Selenium, надає зручний спосіб конфігурації: затримки, адреса віддаленого драйверу (Selenium grid) тип браузера.

Цей фреймворк не містить строгих рамок: можна використовувати різні архітектурні стилі та інтегруватися з різними інструментами, але не підтримує кросплатформені тести: обмежується роботою з браузером.

1.2.3. Playwright

Фреймворк для автоматизації тестування вебдодатків створений компанією Microsoft [3].

Користувачі можуть:

– використовувати headless та headed mode для запуску тестів;

– робити кросбраузерні тести;

– запускати тести в ci/cd середовищі;

– використовувати улюблену мову програмування для написання клієнтської частини;

– займатися автоматизацією рутинних задач в браузері;

– реалізовувати web-scrapping.

На відміну від selenium-based фреймворк використовує іншу архітектуру. Замість JSON-WIRE protocol використовуються підключення через сокети. На рис.1.4 наведено узагальнену архітектуру Playwright.

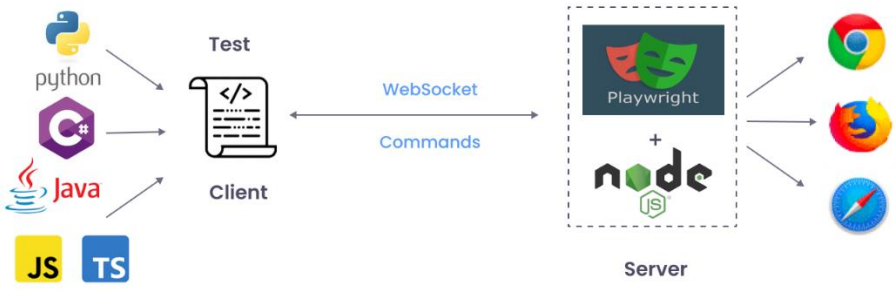


Рис. 1.4. Узагальнена архітектура Playwright

Оскільки це інструмент для роботи з браузером, Playwright не містить користувацького інтерфейсу, і запуск тестів виконується через інтерфейс бібліотеки юніт-тестування, яку обере користувач самостійно.

1.2.4. Postman

Postman – це платформа для проєктування та тестування API [4].

Користувачі можуть:

– надсилати HTTP запити;

– групувати запити;

– створювати тести на JS;

– запускати у CLI за допомогою Newman;

– створювати DDT тести на основі JSON, CSV;

– спільно працювати над проєктуванням та тестуванням;

– створювати змінні для використання в запитах на різних рівнях організації проєкту.

Postman також дозволяє створювати документацію API прямо на основі колекцій, імпортувати/експортувати запити, проводити моніторинг, створювати середовища тестування та підтримує інтеграції з CI/CD пайплайнами.

Однак слід враховувати, що Postman орієнтований виключно на роботу з API – він не призначений для повноцінного тестування вебінтерфейсів, мобільних застосунків чи системного ПЗ.

В Postman шаблони запитів можна групувати в колекції та теки. Приклад групування запитів в колекцію наведено на рис. 1.5. Також за допомогою Postman можна задавати різні типи автентифікації.

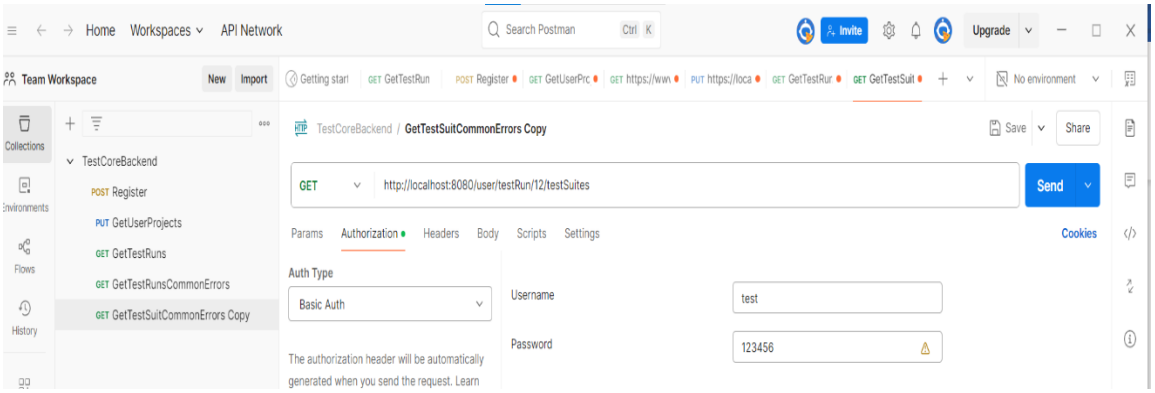


Рис. 1.5. Графічний інтерфейс Postman

На рис. 1.6 використовується Basic Auth.

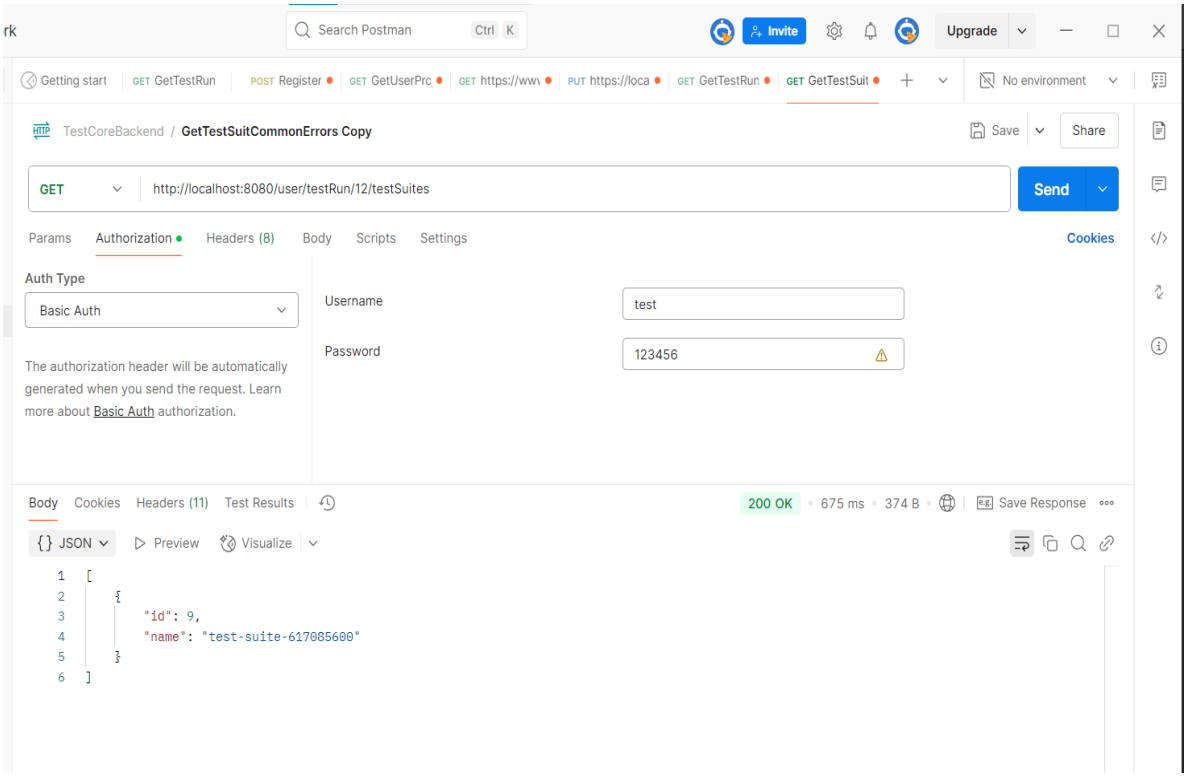


Рис. 1.6. Postman. Ілюстрація процесу надсилання запиту з Basic Auth та отримання відповіді

На рис. 1.7 наведено приклад тесту написаного мовою JS у Postman.

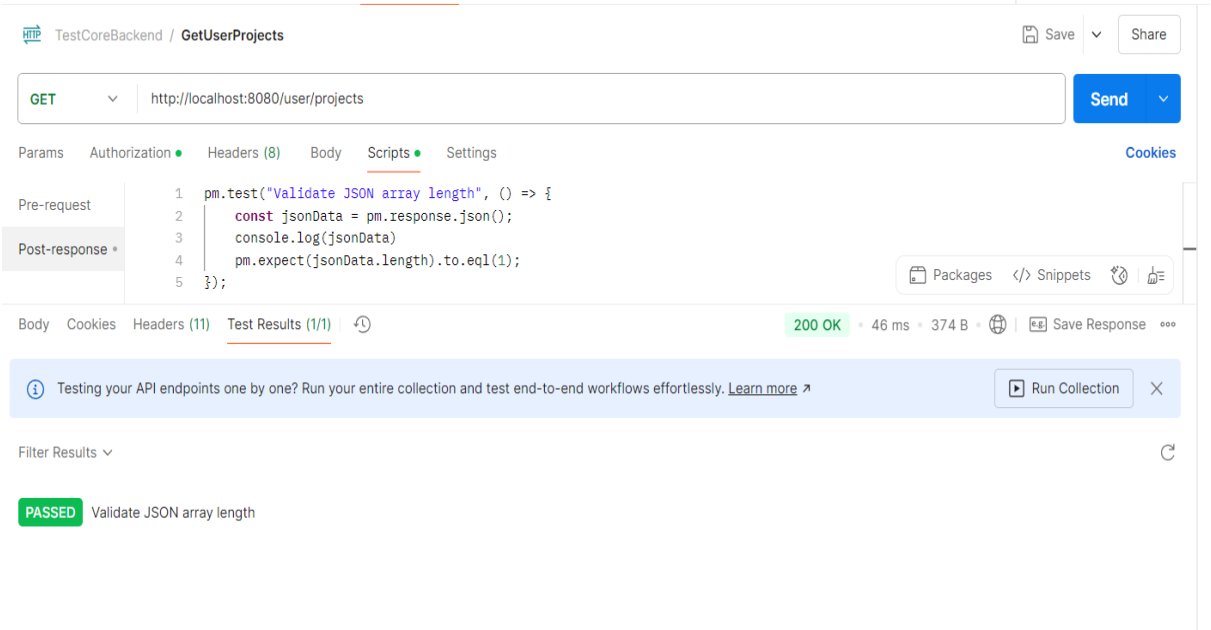


Рис. 1.7. Postman. Приклад тесту на JS

На рис. 1.8 наведено приклад використання Newman в комнадній оболонці, схожий вивід можна отримати в системах CI/CD.

Рис. 1.8. Використання Newman

Всі проаналізовані програмні рішення характеризуються перевагами та певними недоліками, які систематизовано та подано у таблиці 1.1. Також наведено переваги власного продукту над кожним з конкурентів та проаналізовано схожу та відмінну функціональність.

Таблиця 1.1

Результат аналізу конкурентів

Назва	Доля ринку	Конкурентні переваги	Схоже та відмінне	Переваги власного продукту
Postman	Лідер у сфері тестування API	Інтуїтивно зрозумілий інтерфейс, підтримка автоматизованого тестування, співпраця з CI/CD, можливість створення API документації	Схоже: тестування API, підтримка інтеграції з іншими інструментами. Відмінне: відсутність підтримки мобільного тестування, веб-інтерфейс без окремого додатку для автоматизації тестів.	Підтримка автоматизації для Web, API та Mobile в одному проєкті, зручність роботи з кількома сервісами одночасно.
Cypress	Популярний у вебтестуванні	Швидкість виконання тестів, простота налаштування, інтеграція з CI/CD, підтримка JavaScript для написання тестів	Схоже: автоматизація веб-тестування, інтеграція з CI/CD. Відмінне: відсутність підтримки мобільного тестування, акцент лише на JavaScript. Відсутність DDT реалізації	Можливість мобільного тестування та кросплатформеного тестування, створення тестів, що керуються даними.

Продовження табл. 1.1

Playwright	Швидко зростаючий конкурент у вебтестуванні	Підтримка тестування на всіх основних браузерах, кросбраузерне тестування, зручні інструменти дебагінгу	Схоже: кросбраузерне тестування, підтримка інтеграції з CI/CD. Відмінне: відсутність підтримки мобільного тестування.	Універсальність: підтримка Web, API, Mobile, вбудовані звіти та простий інтерфейс для управління тестами.
Selenide	Використовується у Java-спільноті, особливо в корпоративних проєктах	Простота API, зручна обробка динамічних елементів DOM, автоматичне очікування елементів	Схоже: підтримка UI-тестування, сумісність із Selenium, можливість інтеграції з CI/CD Відмінне: не підтримує мобільне тестування, відсутність UI-інтерфейсу	підтримка Web, API та Mobile у межах єдиного рішення, централізована обробка результатів і оновлення тестів через сервер, наявність графічного інтерфейсу.

1.3. Загальні вимоги до системи

Виходячи з наведених вище відомостей про існуючі програмні рішення для автоматизованого тестування, можна сформулювати вимоги до розроблюваного програмного забезпечення. Наведемо їх нижче.

1. Підтримка автоматизованого тестування для Web, API, Mobile застосунків.
2. Можливість створення, редагування та запуску тестових сценаріїв.
3. Генерування звітів про результати тестування.
4. Підтримка інтеграції з CI/CD (наприклад, GitHub Actions, Jenkins).
5. Можливість зберігання результатів тестів на сервері.
6. Серверний аналіз результатів тестування для пошуку спільних помилок в автоматизованих тестах.
7. Можливість написання кросплатформних тестів для WEB та Mobile.
8. Підтримка шаблону POM з використанням вкладених компонентів.
9. Підтримка багатопотокового запуску тестів.
10. Підтримка запуску тестів у headless режимі.
11. Можливість запускати тести через браузерну ферму (Selenium grid).
12. Можливість запускати тести на Google Chrome, FireFox, Edge – на найновіших версіях браузера без Selenium Grid.
13. Підтримка Data driven testing.

1.4. Вимоги до безпеки

З розвитком інформаційних технологій та збільшенням кількості кібератак забезпечення безпеки програмного забезпечення набуває особливої актуальності. Через це до розроблюваного програмного забезпечення було висунуто наступні вимоги до безпеки:

1. Авторизація при запуску тестів через CLI. Необхідно реалізувати механізм авторизації користувачів, які запускають тести з командного рядка. Це дозволить контролювати, хто має право

запускати тести, обмежити доступ тільки авторизованим особам і забезпечити відстеження дій кожного користувача.

2. Хешування паролів із додаванням «солі». Зберігання паролів у відкритому вигляді є критичною вразливістю. Цей підхід ускладнює використання rainbow-таблиць для зворотного відновлення паролів, навіть якщо зловмисник отримає доступ до бази даних.
3. Захист API від несанкціонованого використання. Система має реалізовувати механізми аутентифікації та авторизації при зверненні до API.
4. Запровадити політику сильних паролів. Для забезпечення захищеності облікових записів користувачів необхідно ввести політику формування складних паролів.
5. Використання ORM – для забезпечення захисту від SQL-ін’єкцій. ORM дозволяє взаємодіяти з базами даних на високому рівні, автоматично захищаючи від SQL-ін’єкцій. Замість створення запитів вручну за допомогою конкатенації рядків, ORM забезпечує використання параметризованих запитів та перевірку введених даних, що суттєво знижує ризики, пов’язані з ін’єкціями.

1.5. Висновки до розділу

У ході підготовки матеріалів до першого розділу дипломного проєкту було виконано низку аналітичних і дослідницьких задач, спрямованих на обґрунтування доцільності створення власного фреймворку автоматизованого тестування програмного забезпечення. Зокрема, були реалізовані наступні кроки:

1. Проаналізовано сучасний стан ринку засобів автоматизованого тестування Web, API та Mobile застосунків.
2. Визначено ключові проблеми, з якими стикаються користувачі існуючих фреймворків та інструментів, включаючи складність

налаштування, обмежену підтримку певних типів додатків та недостатню гнучкість.

3. Виявлено найбільш популярні програмні продукти у сфері автоматизованого тестування, зокрема Postman, Cypress, Playwright, Selenide.
4. Сформовано перелік критеріїв, за якими було здійснено порівняння існуючих рішень, включаючи функціональні можливості, зручність інтеграції, підтримку різних платформ, інтерфейс тощо.
5. Проведено порівняльний аналіз згаданих інструментів, визначено їхні сильні та слабкі сторони, що дозволило виявити недоліки, які можна усунути при створенні власного рішення.
6. Побудовано узагальнену таблицю з результатами аналізу конкурентів, що наочно демонструє рівень функціонального покриття кожного з інструментів.
7. На основі зібраної інформації сформовано перелік функціональних можливостей, які повинні бути реалізовані у межах дипломного проєкту, щоб забезпечити ефективність, гнучкість та універсальність розроблюваної системи.
8. Результати цього етапу стали підґрунтям для формування вимог до майбутнього програмного продукту, що викладені в підрозділах 1.3 та 1.4, і забезпечили логічну основу для початку проєктування власної архітектури фреймворку.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Перш за все, треба зазначити, що розроблюваний проєкт складається з клієнтської та серверної частини, тобто має архітектуру клієнт-сервер.

На стороні сервера проходить аналіз результатів регресії для виправлення автотестів, зберігання шаблонів проєктів.

Клієнт розроблений для взаємодії з сервером та взаємодії з шаблоном проєкта (фреймворка) для автоматизованого тестування, що завантажується з серверу.

Варто зауважити, що наведені частини є допоміжними до роботи. Звичайно, основна частина буде складатися з самого засобу для створення автоматизованих тестів. Надалі буде позначено, як основна частина.

Вибір засобів реалізації є важливим, тому що його результати впливають на швидкість створення додатку та продуктивність самої системи. Також значущим є і вибір СКБД.

2.1. Вибір технологій для розроблення основної частини фреймворку

Основна частина фреймворку складається з наступних частин: з платформи тестування (конфігурація, запуск, отримання результатів тестування), системи збірки, мови програмування, бібліотек для взаємодії з браузером, мобільними пристроями та API.

2.1.1. JAVA

Java [5] – це одна з найбільш універсальних мов програмування, яка відіграє ключову роль у створенні сучасного програмного забезпечення. Її активно використовують для розроблення корпоративних застосунків, мобільних застосунків на платформі Android, серверних систем, а також у сфері автоматизованого тестування. У межах розроблення власного

фреймворку Java було обрано як основну мову через її зрілість, продуктивність та багату екосистему бібліотек.

Однією з ключових переваг Java є її багатоплатформність – принцип «write once, run anywhere» дозволяє розгорнути рішення як на Windows, так і на Unix-подібних системах без змін у коді. Крім того, вона має статичну типізацію, що дозволяє виявляти багато помилок ще на етапі компіляції.

Java також має ефективну інтеграцію з системами побудови (Maven, Gradle), тестовими фреймворками (JUnit, TestNG), інструментами для роботи з базами даних (Hibernate, JPA), бібліотеками для логування (Log4j, SLF4J), що дозволяє створювати надійні та гнучкі автоматизовані рішення. Більше того, середовище виконання Java (JVM) забезпечує високий рівень безпеки, що є критично важливим у багатокористувацьких та CI/CD-середовищах.

Для реалізації фреймворку було обрано саме Java, оскільки більшість популярних бібліотек для тестування (Selenium, Appium, Rest-Assured) мають повну підтримку саме цієї мови. Також Java має чудову інтеграцію з системами CI/CD (GitHub Actions [25], Jenkins), контейнеризацією (Docker [26]), що дозволяє ефективно автоматизувати процеси тестування.

2.1.2. TestNG

TestNG – це бібліотека для модульного, інтеграційного та end-to-end тестування, який став стандартом у багатьох Java-проєктах [6]. Його основна перевага – гнучкість при організації тестів і можливість налаштування складних сценаріїв запуску.

Одним із головних функціональних блоків TestNG є створення test suite – тобто об'єднання тестів у групи, які можна запускати за конкретним шаблоном. Наприклад, можна створити окремі групи для smoke, regression та integration тестів і запускати їх у відповідності до потреб релізного циклу.

Крім того, TestNG дозволяє налаштувати паралельне виконання тестів, що значно скорочує час проходження повного тестового запуску.

Через XML-конфігурацію або анотації в коді можна гнучко керувати запуском, повторним запуском, пропуском або залежностями між тестами.

Інтеграція з DataProvider дозволяє створювати параметризовані тести, що є основою підходу Data Driven Testing (DDT), де вхідні дані для тестів зберігаються окремо (наприклад, у JSON, CSV або базі даних).

TestNG також підтримує власну систему слухачів (listeners), які можуть реагувати на події під час виконання тестів. Слухачі зазвичай використовуються для логування, збору метрик, створення кастомних звітів або знімків екрану при падінні тестів.

Інструмент чудово інтегрується з Maven та CI-системами. Враховуючи вищезазначене, TestNG є логічним вибором для розроблення фреймворку, орієнтованого на масштабоване і підтримуване автоматизоване тестування.

2.1.3. Maven

Apache Maven є стандартом у світі Java-проєктів, коли йдеться про управління залежностями, компіляцію, збірку, виконання тестів та розгортання [7]. Його ключова перевага – централізоване управління всіма аспектами життєвого циклу проєкту через файл pom.xml.

Механізм POM (Project Object Model) дозволяє визначити структуру проєкту, під'єднати потрібні бібліотеки (Selenium, Appium, Log4j тощо), а також налаштувати фази збірки. Maven автоматично завантажує всі необхідні бібліотеки з центрального репозиторію, що значно спрощує налаштування середовища для інших учасників команди.

Інструмент підтримує запуск тестів у рамках фази test, що дозволяє інтегрувати його з фреймворками TestNG або JUnit. Також за допомогою плагінів Maven можна генерувати звіти (наприклад, Surefire або Allure), підключати статичний аналіз коду, виконувати перевірки якості.

Для CI/CD-середовищ Maven є надзвичайно зручним, завдяки чітко визначеній структурі: він дозволяє легко запускати всі фази (clean, compile,

test, package, verify, deploy) за допомогою одного рядка команди. Це мінімізує кількість помилок та забезпечує передбачуваність збірки.

У межах даного фреймворку Maven виконує роль координатора всіх компонентів, а також інструмента, через який здійснюється централізоване оновлення версій бібліотек, побудова артефактів і запуск тестів.

2.1.4. Selenium

Selenium [8] – це відкритий інструмент, який дозволяє автоматизувати тестування вебінтерфейсів. Він підтримує роботу з різними браузерами (Chrome, Firefox, Edge тощо) та операційними системами, що робить його універсальним рішенням для QA-інженерів.

Основна частина Selenium – WebDriver, який дає змогу програмно керувати браузером:

– відкривати вебсторінки;

– взаємодіяти з елементами;

– перевіряти наявність даних на сторінці;

– робити знімки екрану;

– виконувати JavaScript-код.

Тести можна писати на Java, Python, C# або JavaScript. Однак, Java – найпоширеніший варіант, оскільки добре інтегрується з іншими інструментами (TestNG, JUnit, Maven).

Переваги Selenium:

– підтримка багатьох мов і платформ – можна вибрати зручну мову програмування;

– гнучкість – підходить як для простих, так і для складних тестів;

– інтеграція з хмарними сервісами (BrowserStack, Sauce Labs) для паралельного запуску тестів;

– велика спільнота – багато готових рішень і прикладів.

– Selenium часто застосовують для:

– регресивного тестування;

– перевірки UI під час розроблення;

– написання скриптів для автоматизації рутинних дій.

Цей інструмент став стандартом у вебтестуванні, завдяки своїй надійності та широким можливостям. Він дозволяє максимально наблизити автоматизовані тести до реальних дій користувача.

2.1.5. Appium

Appium [9] – це платформа автоматизації з відкритим вихідним кодом, яка надає можливості автоматизації на основі WebDriver для широкого спектру різних платформ мобільних пристроїв, настільних ПК і IoT. Appium є модульним і розширюваним, а також підтримує кілька мов програмування.

Тому це оптимальний вибір для кросплатформеного тестування. Також він підтримує взаємодію з гібридними додатками, тобто тими, що містять нативну та веб частину, шляхом переключення контексту. Також за допомогою Appium можна робити дії не тільки з додатком, але й із самим девайсом/емулятором: натискати кнопки навігації, включати геолокацію тощо.

2.1.6. Альтернативи

У процесі вибору технологій для розроблення основної частини фреймворку також розглядалися альтернативні варіанти, що наведено нижче.

– Замість Java могли бути використані такі мови, як Python або JavaScript (Node.js). Однак, Java має значні переваги в тестуванні завдяки великій кількості стабільних бібліотек, сильній екосистемі, високій продуктивності та широкій підтримці корпоративними рішеннями. Крім того, вона забезпечує кращу інтеграцію з такими інструментами, як Appium, Selenium, Maven та TestNG.

– Замість TestNG ми могли б використати JUnit, інший популярний фреймворк для тестування на Java. Однак, TestNG пропонує більш

гнучку систему конфігурації тестів, включаючи групування, залежності між тестами, зручний DataProvider та зручне розпаралелювання, що є важливою вимогою для масштабованого фреймворку.

– Як систему збірки, Gradle можна розглядати замість Maven, яка є більш сучасною і забезпечує швидшу збірку завдяки інкрементальній компіляції. Однак, Maven має простішу структуру POM, краще підходить для навчальних або дипломних проєктів, завдяки більшій кількості документації та підтримки.

2.2. Вибір технологій для розробки клієнтської частини фреймворку

Вибір технологій для клієнтської частини напряму впливає на ефективність розробки, легкість підтримки коду та можливість інтеграції з основною частиною фреймворку та серверною частиною. У цьому розділі описано ключові міркування, що враховувалися під час вибору, та обґрунтування прийнятих рішень.

2.2.1. JavaFX

JavaFX – це платформа клієнтських програм нового покоління з відкритим кодом для настільних, мобільних і вбудованих систем, побудованих на Java. Це результат спільних зусиль багатьох осіб і компаній з метою створення сучасного, ефективного та повнофункціонального інструментарію для розроблення багатофункціональних клієнтських програм [10].

Цей засіб було вибрано через кросплатформність застосунків, створених за допомогою нього. Це дозволяє захопити більшу аудиторію користувачів.

Також разом з JavaFX буде використовуватися програмний засіб Scene Builder для створення вікон та сцен через графічний інтерфейс та перетворення його в xml формат для імпорту в JavaFX.

2.2.2. Альтернативи

Серед інших можливих технологій для розроблення клієнтської частини фреймворку розглядалися також:

1. Swing [11] – стара, але стабільна бібліотека для створення графічного інтерфейсу на Java. Попри те, що Swing має широку підтримку та документацію, вона застаріла й обмежена в плані створення сучасного UI. Візуальні компоненти виглядають застарілими, а підтримка анімацій та стилізації суттєво поступається JavaFX.
2. Electron [12] – кросплатформна технологія для створення настільних застосунків на базі JavaScript, HTML і CSS. Попри сучасний вигляд і багаті можливості інтерфейсу, Electron споживає більше ресурсів, оскільки запускає браузер у кожному вікні. Для дипломного проєкту, де важлива легкість та продуктивність, це суттєвий недолік.
3. Qt [13] – платформа для створення настільних застосунків. Однак використання її вимагає або інтеграції з іншими мовами програмування, або складної конфігурації, що ускладнює підтримку та зменшує гнучкість. До того ж, вона не є частиною екосистеми Java.
4. Web-based UI (наприклад, React або Angular у поєднанні з Java через REST API) – це рішення могло б забезпечити кросплатформність та доступність через браузер, однак це значно ускладнило б архітектуру, розроблення й розгортання, особливо якщо враховувати потребу в локальній взаємодії з файловою системою та зовнішніми бібліотеками для тестування.

У порівнянні з цими технологіями, JavaFX має низку суттєвих переваг у контексті реалізації дипломного проєкту:

– нативна інтеграція з Java, що дозволяє легко поєднувати клієнтську частину з ядром фреймворку без необхідності в складних адаптерах або сторонніх бібліотеках;
– підтримка FXML та Scene Builder, що спрощує створення інтерфейсу навіть для користувачів без досвіду програмування – це дозволяє зменшити час на розробку та забезпечити швидке прототипування інтерфейсів;
– гнучка стилізація через CSS, можливість анімації, обробки подій та побудови складних UI-компонентів;
– кросплатформність – програми на JavaFX працюють на Windows, Linux та macOS без значних змін у коді;
– активна підтримка спільноти та оновлення через проєкт OpenJFX.

Таким чином, JavaFX було вибрано як оптимальне рішення, що поєднує в собі зручність розроблення, ефективність виконання, сучасний вигляд та простоту розгортання, що повністю відповідає вимогам до клієнтської частини автоматизованого фреймворку.

2.3. Вибір технологій для розроблення серверної частини фреймворку

Для серверної частини важливо підібрати такі технології, які забезпечать швидку обробку даних, надійну роботу з базами даних і можливість масштабування системи в майбутньому. Далі наведено огляд вибраних рішень та пояснення, чому саме вони були обрані.

2.3.1. Express

Node.js [14] – це серверна платформа на основі JavaScript, яка дозволяє будувати ефективні додатки. Вона використовує модель подій, що дозволяє обробляти багато запитів одночасно без блокування основного потоку.

Express.js [27] – це простий, але потужний фреймворк для розроблення веб-серверів та API.

Переваги:

– один JavaScript для всього – одна мова для фронтенду та бекенду спрощує розроблення;
– асинхронність – добре підходить для реального часу (наприклад, чатів чи ігор);
– велика кількість бібліотек – через npm можна легко під'єднати потрібні інструменти;
– швидкий старт – не потрібна компіляція, код виконується одразу.

Недоліки:

– динамічна типізація – JavaScript може призводити до помилок, тому іноді доводиться використовувати TypeScript;
– мінімалістичність Express – багато функцій (безпека, робота з даними) потрібно додавати самостійно;
– обмеження у важких обчисленнях – складні операції можуть уповільнити роботу сервера.

2.3.2. Django

Django [15] – це повноцінний фреймворк для створення вебдодатків та API. Він включає в себе все необхідне для швидкої розробки: ORM, систему аутентифікації, маршрутизацію, адмін-панель та багато іншого.

Перевагами Django є такі факти:

– не потрібно шукати додаткові бібліотеки для базових функцій;
– Python дозволяє писати чистий і зрозумілий код;
– забезпечення швидкого старту – ідеально підходить для MVP та CRUD-додатків;
– гнучкість – можна використовувати як для монолітів, так і для мікросервісів.

Недоліки Django:

– менша продуктивність під навантаженням – для високонавантажених систем Java або Go можуть бути кращим вибором;
– залежність від Python – якщо інші частини системи написані на Java, інтеграція може бути складною;
– складніше налаштування CI/CD – якщо інфраструктура орієнтована на Java, знадобиться додаткова робота;

Цей фреймворк лежить в основі таких а, як YouTube, Pinterest та Mozilla. Він також підходить для тестування, наприклад, у зв'язці з PyTest, Allure та REST API.

Django є чудовим вибором для швидкої розробки вебзастосунків, проте для побудови високонавантажених систем може виникнути потреба у використанні інших технологій.

2.3.3. Spring Boot

Spring Boot – це фреймворк для створення готових до випуску вебдодатків на Java з мінімальною конфігурацією [16] . Він інтегрує всі компоненти Spring (MVC, Data, Security тощо) та дозволяє швидко створювати RESTful сервіси.

Переваги:

– автоматична конфігурація;
– потужна система безпеки (Spring Security);
– вбудований Tomcat сервер;
– підтримка JPA/Hibernate для роботи з базами даних;
– логування, моніторинг, інтеграція з Swagger, Jenkins, Docker тощо;
– стандартизація – Spring є промисловим стандартом у Java-середовищі.

Недоліки:

– порівняно висока вхідна межа для новачків;

– великий розмір jar-файлу.

2.3.4. Результати проведеного аналізу

Під час проведеного порівняльного аналізу було розглянуто кілька альтернатив до Spring Boot для реалізації серверної частини фреймворку. Враховувалися такі критерії, як продуктивність, масштабованість, зручність розроблення, інтеграція з іншими сервісами, наявність модульності, підтримка баз даних, безпеки та відповідність обраній мові програмування на клієнті.

У порівнянні з іншими рішеннями, Spring Boot забезпечує:

– стабільну і продуктивну роботу у багатокористувацькому середовищі завдяки потужному стеку Java та багатопотоковому обробленню запитів;
– чітку архітектуру з розділенням обов’язків: контролери, сервіси, DAO, що відповідає принципам MVC та Clean Architecture;
– повноцінну підтримку безпеки, логування, баз даних (Hibernate), валідації, міжнародалізації тощо – значна частина функціональності доступна «з коробки»;
– широкі можливості розширення і модульності через використання Spring Boot Starter’ів, власних конфігурацій, dependency injection;
– інтеграцію з Docker, Jenkins, GitHub Actions, MySQL, Swagger, Testcontainers тощо – завдяки активній підтримці в open-source-спільноті;
– сумісність із Java-екосистемою, що забезпечує єдність технічного стеку у проєкті (включаючи клієнтську частину, розроблену на JavaFX).

Щобільше, використання Spring Boot дозволяє дотримуватись принципів чистої архітектури та Domain-Driven Design, що є важливою умовою для побудови масштабованого, підтримуваного та розширюваного фреймворку. Це дає можливість у майбутньому легко інтегрувати нові

клієнтські застосунки (наприклад, мобільний застосунок, веб-інтерфейс, REST API для сторонніх сервісів).

2.4. Вибір СКБД

Під час розроблення ПЗ з’явилася необхідність у виборі СКБД, що б забезпечувала ефективне, надійне та масштабоване зберігання інформації, що генерується під час виконання тестів. Нижче розглянемо варіанти існуючих рішень.

2.4.1. PostgreSQL

PostgreSQL [17] – це об’єктно-реляційна СКБД із відкритим вихідним кодом, використовується у багатьох проєктах з високими вимогами до надійності та стабільності.

Основні переваги PostgreSQL:

– безкоштовність – система доступна для безоплатного використання без функціональних обмежень;
– розширена підтримка форматів даних – PostgreSQL дозволяє працювати з JSON, XML, CSV, геоданими та іншими структурами;
– відповідність ACID-вимогам – система гарантує цілісність, узгодженість, ізольованість та довговічність операцій;
– надійне відновлення після збоїв – забезпечується внаслідок механізмів журналювання транзакцій;
– масштабованість – можливість працювати з великими обсягами даних без зниження продуктивності;
– розширюваність – розробник може додавати нові типи даних, оператори, функції та індекси;
– підтримка кількох видів реплікації – асинхронна, потокова, логічна тощо.

Недоліки PostgreSQL:

– складність налаштування – початкове конфігурування потребує глибших технічних знань;
– великий розмір файлів – база даних може займати значні ресурси зберігання;
– відсутність вбудованого кластерування – для реалізації кластера потрібні сторонні інструменти.

2.4.2. MongoDB

MongoDB [18] – одна з найпопулярніших документо-орієнтованих СКБД, що зберігає дані у гнучкому форматі BSON, подібному до JSON. Вона розроблена для зручної роботи з неструктурованими даними.

Переваги MongoDB:

– підтримка індексів – що забезпечує швидкий пошук та оброблення даних;
– гнучка структура зберігання – документи у BSON-форматі дозволяють зберігати різноманітну інформацію без жорсткої схеми;
– підтримка реплікації – включаючи Replica Sets для підвищення відмовостійкості;
– журналювання транзакцій – можливість відновлення даних після збою;
– висока продуктивність – система ефективно працює з великими обсягами даних;
– простота у використанні – інтерфейс і синтаксис MongoDB нагадують JavaScript, що є зручним для багатьох розробників.

Недоліки MongoDB:

– неповна відповідність ACID-принципам – транзакції мають обмежену підтримку, а консистентність і ізольованість не завжди гарантуються;

– відсутність операцій JOIN – ускладнює виконання запитів до пов’язаних об’єктів;
– вартість масштабування – потребує серйозних ресурсів і складних рішень при побудові кластерів;
– менша зручність при роботі з реляційними структурами – не підходить для чітко структурованих даних з фіксованими зв'язками.

2.4.3. MySQL

MySQL [19] – одна з найвідоміших реляційних СКБД з відкритим вихідним кодом, що отримала широке застосування завдяки своїй стабільності, зручності та продуктивності. MySQL активно використовується в проєктах різного масштабу – від невеликих сайтів до великих корпоративних застосунків.

Основні переваги MySQL:

– повна відповідність ACID-вимогам – забезпечується підтримкою транзакцій, ізольованості та узгодженості;
– підтримка реплікації – реалізовано кілька механізмів (master-slave, master-master);
– інтуїтивно зрозумілий інтерфейс – численні клієнтські утиліти спрощують керування БД;
– ефективна робота з індексами – що дозволяє пришвидшити обробку запитів;
– гнучка система безпеки – підтримка прав доступу, шифрування, SSL-з’єднань;
– широка спільнота підтримки – велика база документації та активна участь спільноти.

Недоліки MySQL:

– обмежена підтримка XML, слабка підтримка JSON – не підходить для гнучких форматів даних;

– вразливість до атак – за умов недотримання безпекових практик можливе викрадення даних;
– складнощі при масштабуванні – ефективність знижується при роботі з дуже великими обсягами даних;
– повільна швидкість запису – при великих потоках даних запис може бути вузьким місцем;
– повільне впровадження нових функцій.

2.4.4. Результати аналізу та вибір СКБД

У результаті порівняльного аналізу трьох систем – PostgreSQL, MongoDB та MySQL – було виявлено, що кожна з них має свої сильні сторони, проте вибір бази даних має бути обґрунтованим з урахуванням потреб конкретного проєкту.

PostgreSQL продемонструвала найвищу відповідність ключовим вимогам до СКБД, але цю систему не було обрано через її надмірну складність.

MongoDB добре підходить для роботи з гнучкими структурами даних, але відсутність повної відповідності ACID-принципам є серйозною вадою, особливо для систем, де критичною є точність і узгодженість інформації.

У підсумку, вибір було зроблено на користь СКБД MySQL, яка забезпечує баланс між надійністю, продуктивністю, простотою у використанні та безкоштовністю. Незважаючи на деякі недоліки, MySQL повністю відповідає вимогам до зберігання, пошуку та обробки інформації.

2.5. Висновки до розділу

Під час роботи над другим розділом пояснювальної записки виконано ряд завдань:

1) здійснено аналіз існуючих технологій для розроблення серверної частини, клієнтської частини, частини засобу тестування (основна частина) та бази даних;

2) розглянуто найпопулярніші безкоштовні засоби реалізації;
3) обґрунтовано вибір технологій для розроблення частин проєкту та освітлено взаємодію цих частин;
4) сформульовано висновки за результатами аналізу та обрано оптимальні технології для розроблення кожної складової застосунку.

У результаті проведеної роботи закладено основу для подальшої практичної реалізації програмного продукту. Отримані результати дозволяють забезпечити відповідність обраних рішень вимогам до продуктивності, надійності, масштабованості та зручності підтримки системи. Ретельний вибір технологій на цьому етапі є запорукою ефективного розроблення та успішної експлуатації майбутнього ПЗ.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ПЗ

3.1. Архітектура розробленого фреймворку

В основі розробленого фреймворку лежить архітектура клієнт-сервер. Клієнтом є CLI та графічний інтерфейс JavaFX. JavaFX-клієнт в свою чергу взаємодіє з серверною частиною через виклики CLI. CLI комунікує з бекендом за допомогою REST API.

Архітектура фреймворку представлена нижче на рис. 3.1.

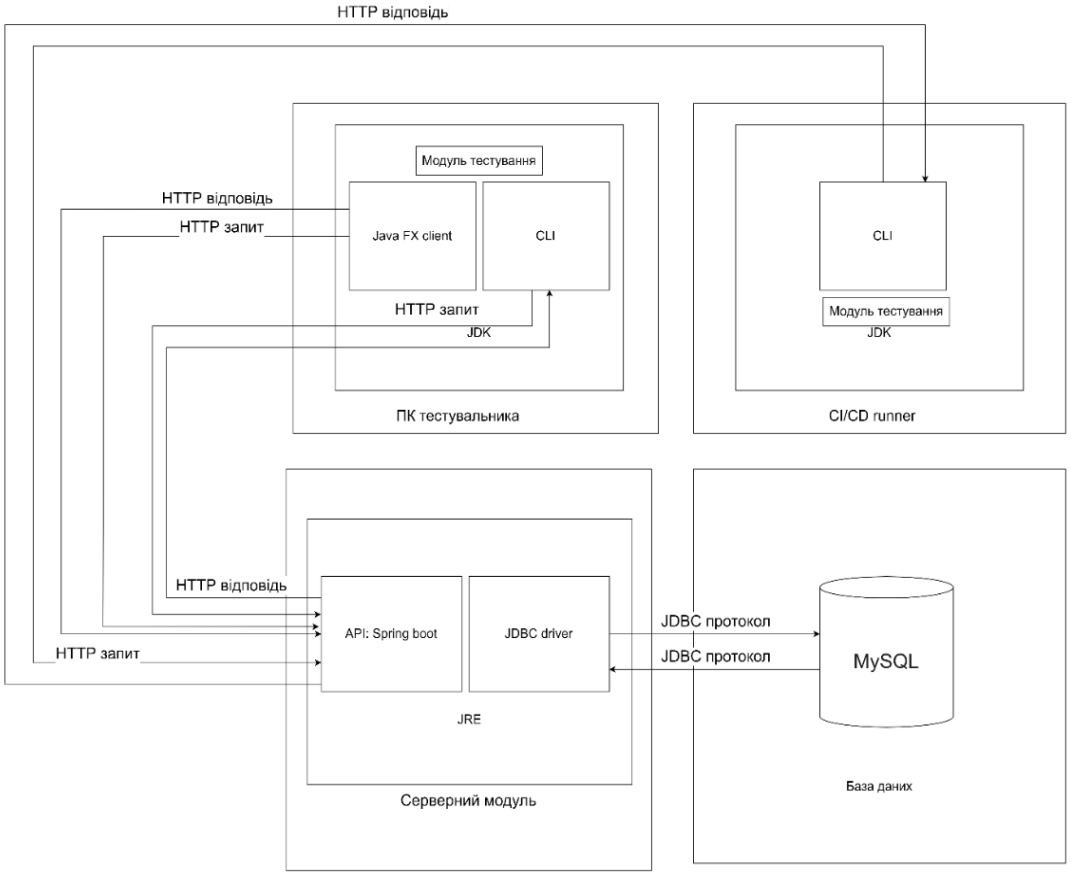


Рис. 3.1. Архітектура фреймворку

CLI – необхідний засіб для процесів CI/CD. Для локального відлагодження тестів тестувальником зручніше користуватися графічним інтерфейсом, хоча використання CLI в локальному середовищі теж можливе і важливе для відлагодження процесів CI/CD.

Графічна частина клієнта є кросплатформним настільним додатком, що підтримує ОС на яких встановлено JRE та JavaFX.

Серверна частина відповідає за зберігання шаблону проєкту для автоматизованого тестування, зберігання даних про користувачів та результатів їх тестових запусків по тестових проєктах, виконання парсингу XML файлу результатів тестування у реляційну структуру БД, реалізація логіки групування тестів за спільними помилками в межах тестових наборів, тестових запусків та тестових проєктів. Тестовий проєкт включає в себе тестові запуски, тестові запуски – тестові набори відповідно. Серверна частина має свою архітектуру – багатошарову. На рис. 3.2 наведено архітектуру серверної частини.

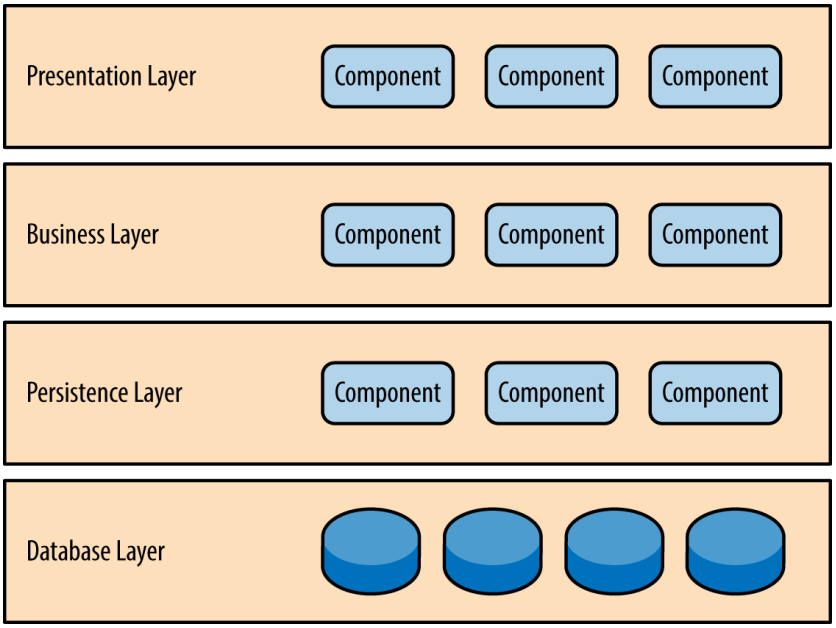


Рис. 3.2. Архітектура серверної частини

Серверна частина компілюється в jar файл і запускається на сервері.

Пакетна організація відповідає багатошаровій архітектурі. Тобто було логічно розбито модулі під кожен з шарів багатошарової архітектури. На рис. 3.3 наведено знімок екрану з структури файлів проєкту.

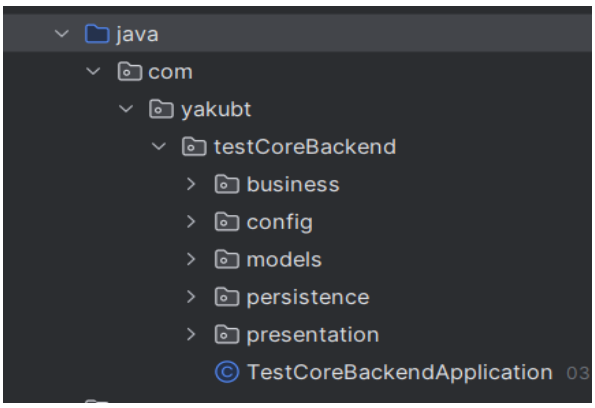


Рис 3.3. Пакетна організація серверної частини

Persistence шар – містить логіку для взаємодії з БД засобами ORM. Цей шар містить опис інтерфейсів JPA.

Presentation шар – містить набір REST контролерів, який забезпечує програмний інтерфейс для взаємодії з серверною частиною за допомогою HTTP запитів.

Основна частина фреймворку також компілюється як jar файл і потім імпортується як бібліотека у шаблоні тестового проєкту, що завантажується запитом до серверної частини.

Основна частина має пакетну організацію відповідно до об’єку тестування. На рис. 3.4 наведено пакетну організацію основної частини фреймворку.

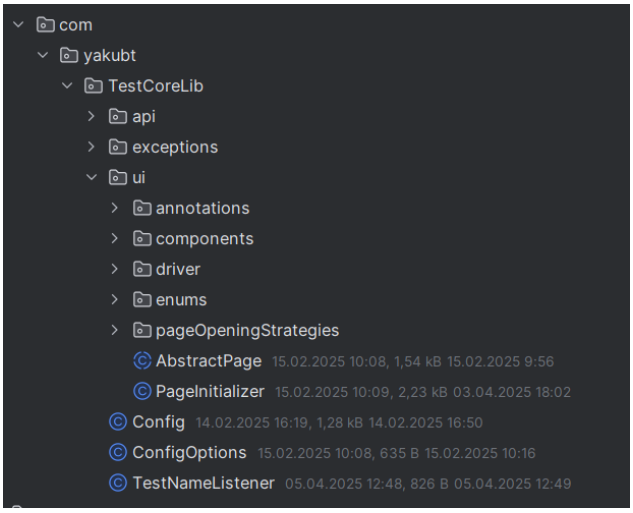


Рис 3.4. Пакетна організація основної частини фреймворку

UI пакет містить засоби для тестування web та mobile та містить логіку кросплатформного тестування цих частин, тобто тестів, що запускаються для Web та Mobile без зміни коду тестів.

В середині пакета «driver» знаходяться модулі для налаштування запуску тестів на емуляторі та різних типах браузерів.

Пакет компонент використовується для тестування графічного інтерфейсу шляхом розбиття вікон/сторінок на дрібніші структурні елементи.

Модуль конфігурації використовується для читання даних з файлу конфігурацій у форматі properties.

Api пакет містить модулі для надсилання HTTP-запитів, конвертації JSON рядка в об’єкт класу, читання JSON файлів та шаблонізації запитів.

3.2. Опис функцій розробленого ПЗ

Перед тим, як переходити до опису функцій фреймворку, необхідно звернути увагу на визначені раніше у п. 1.3 загальні вимоги до системи.

Розроблений фреймворк для автоматизованого тестування програмного забезпечення реалізує підтримку широкого спектра завдань, серед яких створення тестів для перевірки API, вебзастосунків та мобільних застосунків. Автоматизація тестування для мобільних та вебзастосунків ґрунтується на підході розділення графічних інтерфейсів на окремі візуальні компоненти, що забезпечує гнучкість побудови тестових сценаріїв і дозволяє створювати кросплатформені рішення. Передбачена функціональність оновлення тестових наборів відповідно до змін у програмному продукті, а також збору й подальшого аналізу результатів запусків. Система підтримує як послідовний, так і паралельний запуск тестів із можливістю конфігурації параметрів середовища, таких як режими роботи браузера (headless/headed) чи застосування емуляторів мобільних пристроїв. Крім того, передбачено інтеграцію з платформами безперервної інтеграції за допомогою інструменту командного рядка.

На рис. 3.5 наведено діаграму прецедентів, що описують основні функції фреймворку та їх взаємозв’язок з акторами та між собою.

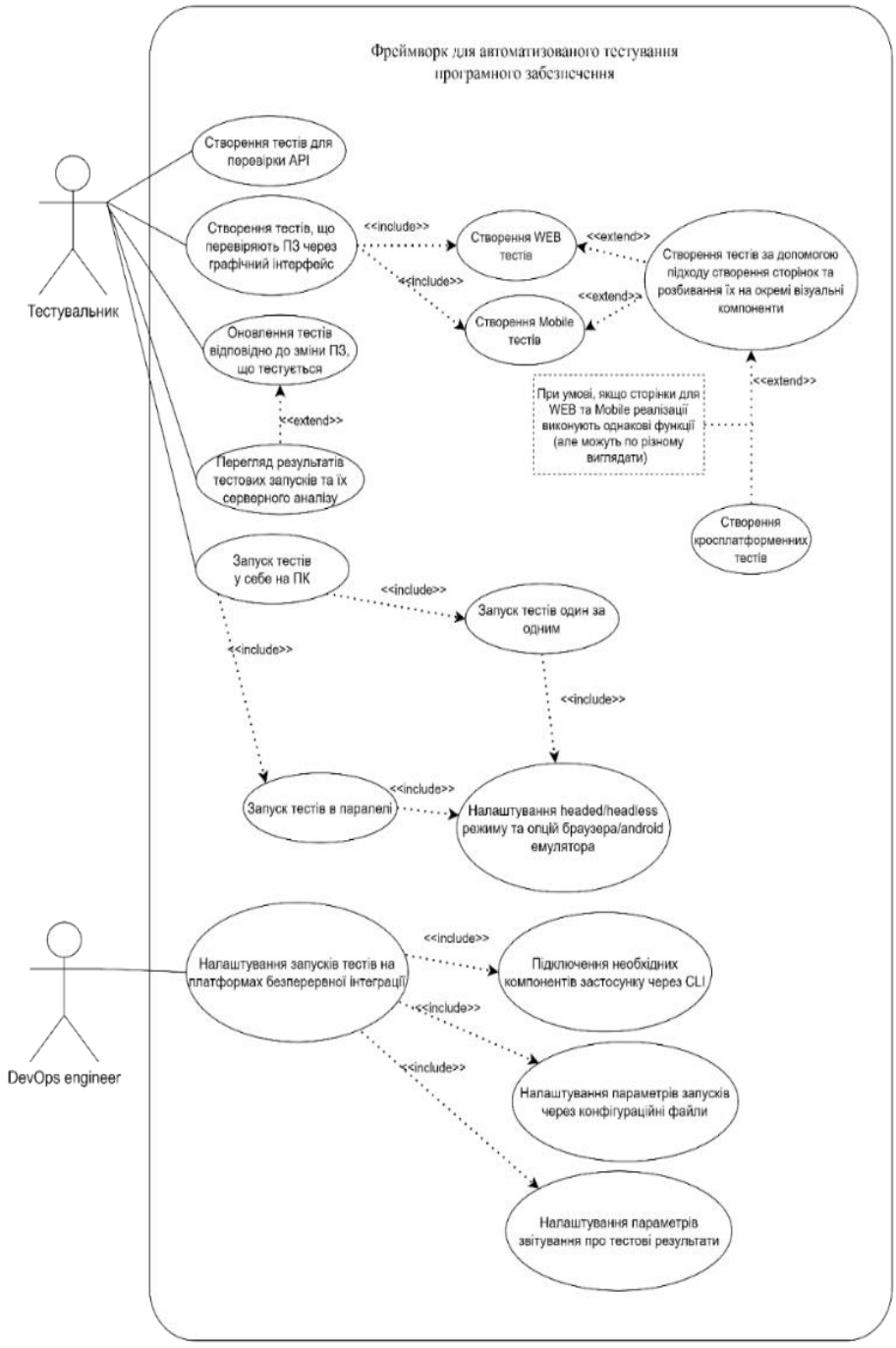


Рис. 3.5. Діаграма прецедентів

Для детального опису функціональності ПЗ було вибрано сценарії використання. Вони дозволяють, глибше зрозуміти функції, що були

поверхово розглянуті на діаграмі прецедентів. В таблицях 3.1 – 3.4 наведено сценарії виростання для деяких функцій фреймворку.

Таблиця 3.1

UC-01: Вхід користувача у систему

UC-01	Вхід користувача у систему
Актори	Тестувальник, DevOps-інженер
Початкові умови	Наявне підключення до інтернету, користувач не авторизований
Очікуваний результат	Користувач увійшов у систему
Основний потік	1. Користувач відкриває командну оболонку та вводить «testcoregui». 2. Користувач натискає кнопку з написом «Login». 3. Користувач вводить свій логін та пароль. 4. Користувач натискає кнопку Login.
Альтернативний потік	1. Користувач відкриває командну оболонку та вводить «testcoregui». 2. Користувач вводить «testcorecli login <ім’я> <пароль>.

Таблиця 3.2

UC-02: Реєстрація користувача у системі

UC-02	Реєстрація користувача у системі
Актори	Тестувальник, DevOps-інженер
Початкові умови	Наявне підключення до інтернету, користувач не авторизований
Очікуваний результат	Користувач зареєструвався у системі

Продовження табл. 3.2

Основний потік	1. Користувач відкриває командну оболонку та вводить «testcoregui». 2. Користувач натискає кнопку з написом «Register». 3. Користувач вводить не пусте ім’я. 4. Користувача вводить пароль. 4-a. Пароль, що має містити щонайменше 8 літер, цифру, принаймні одну велику літеру, принаймні одну малу літеру та спеціальний символ. 4-b. Пароль, що не задовільняє вимогами 4-a. 5. Підтверджує пароль та натискає кнопку Register.
Альтернативний потік	1. Користувач відкриває командну оболонку та вводить «testcorecli». 2. Користувач вводить «testcorecli register <ім’я> <пароль>». пароль, що задовільняє властивостям вказаним у основному потоці.
	4-b. Користувач вводить пароль, що не задовільняє вимогам. Система виводить повідомлення про слабкий пароль та вимоги до нього.

Таблиця 3.3

UC-03: Створення тесту для API

UC-03	Створення тесту для API
Актори	Тестувальник, Backend-система
Початкові умови	Backend-система розгорнута й готова для взаємодії, користувач завантажив шаблон проєкту через CLI.
Очікуваний результат	Користувач створив тест для перевірки API

Продовження табл. 3.3

Основний потік	1. Користувач оголошує тестовий клас. 2. Користувач створює тестовий метод засобами TestNG. 3. Користувач задає endpoint. 4. Користувач викликає метод «sendHttpResponse» задаючи endpoint, HTTP-метод, тіло та заголовки. 4-а. Можливе використання класу JSONReader для перевикористання тіла запиту та його параметризації. 5. Користувач робить перевірку відповідно до свого завдання. 5-а. Можливе використання класу JSONReader для перевикористання тіла запиту та його параметризації.

Таблиця 3.4

UC-04: Завантаження шаблону проєкту для автоматизованого тестування програмного забезпечення

UC-04	Завантаження шаблону проєкту для автоматизованого тестування програмного забезпечення
Актори	Тестувальник, DevOps-інженер
Початкові умови	Підключення до інтернету, користувач попередньо зробив вхід у систему.
Очікуваний результат	Папка тестового проєкту створена
Основний потік	1. Вводимо у командній оболонці: testcorecli download_java_template <назва проєкту>.

3.3. Опис структур даних системи

Як вже було описано в п. 3.2 після запуску тестів генерується xml файл звітів, поля якого заносяться у реляційну БД. Також серверна частина використовує Spring Data JPA [20] для об’єктно-реляційного відображення.

На рис. 3.6 наведено діаграму взаємозв’язків між сутностями.

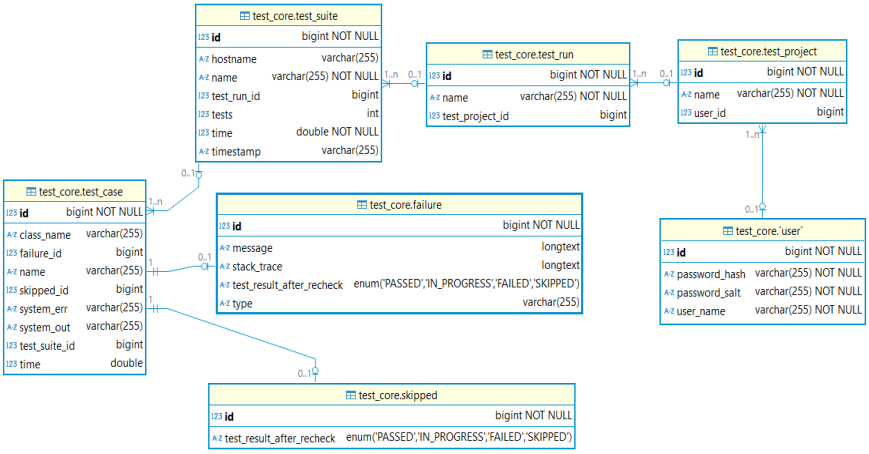


Рис. 3.6. Діаграма взаємозв’язків між сутностями

Далі наведена більш детальна інформація про таблиці, що зберігаються у базі даних.

Таблиця user зберігає дані про зареєстрованих користувачів системи. Для забезпечення вимог до безпеки визначених у п. 1.4 використовується хешування з сіллю.

В таблиці 3.5 наведено інформацію про поля таблиці БД user.

Таблиця 3.5

Таблиця user

User		
Ім’я поля	Тип даних	Характеристики
Id	BIGINT	Первинний ключ

Продовження табл. 3.5

password_hash	VARCHAR(255)	Обов’язкове поле
password_salt	VARCHAR(255)	Обов’язкове поле
user_name	VARCHAR(255)	Унікальне, обов’язкове поле

Таблиця test_project зберігає дані про тестові проєкти користувача. Тестовий проєкт – це найбільша структурна одиниця, що охоплює дрібніші елементи для зберігання результатів запуску тестів. Користувач може мати декілька тестових проєктів. При використанні звітування назва тестового проєкту формується автоматично та має назву проєкту в якому зберігаються автоматизовані тести.

В таблиці 3.6 наведено інформацію про поля таблиці БД test_project.

Таблиця 3.6

Таблиця test_project

test_project		
Ім’я поля	Тип даних	Характеристики
Id	BIGINT	Первинний ключ
name	VARCHAR(255)	Обов’язкове поле
user_id	BIGINT	Зовнішній ключ

Також для цієї таблиці існує обмеження на унікальність пари властивостей user_id та name.

Таблиця test_run зберігає дані про тестові запуски. Тестовий проєкт може містити багато тестових запусків. В таблиці 3.7 інформацію про наведено поля таблиці БД test_run.

Таблиця 3.7

Таблиця test_run

test_run		
Ім’я поля	Тип даних	Характеристики
Id	BIGINT	Первинний ключ
name	VARCHAR(255)	Обов’язкове поле
test_project_id	BIGINT	Зовнішній ключ

Також для цієї таблиці існує обмеження на унікальність пари властивостей test_project_id та name.

Таблиця test_suite зберігає дані про тестові набори. Один тестовий запуск може містити багато тестових наборів. Тести групуються в тестові набори для логічного відокремлення тестів за різними критеріями, такими як: функціональність яка тестується, кількість потоків в яких запускати тести, вид застосунку який тестується тощо.

В таблиці 3.8 наведено інформацію про поля таблиці БД test_suite.

Таблиця 3.8

Таблиця test_suite

test_suite		
Ім’я поля	Тип даних	Характеристики
Id	BIGINT	Первинний ключ
host_name	VARCHAR(255)	Необов’язкове поле
name	VARCHAR(255)	Обов’язкове поле
test_run_id	BIGINT	Зовнішній ключ
tests (кількість)	INT	Необов’язкове поле
time	DOUBLE	Обов’язкове поле

Продовження табл. 3.8

timestamp	VARCHAR(255)	Необов’язкове поле

Таблиця test_case містить дані про результат запуску окремого тесту. Атрибут class_name визначається приналежність тесту до певного класу Java. Тобто це місце, де зберігається автоматизований тест з точки зору коду.

В таблиці 3.9 наведено інформацію про поля таблиці БД test_case.

Таблиця 3.9

Таблиця test_case

test_case		
Ім’я поля	Тип даних	Характеристики
Id	BIGINT	Первинний ключ
class_name	VARCHAR(255)	Необов’язкове поле
failure_id	BIGINT	Зовнішній ключ
name	VARCHAR(255)	Необов’язкове поле
skipped_id	test_project_id	Зовнішній ключ
system_err	VARCHAR(255)	Необов’язкове поле
test_result_after_recheck	ENUM(‘PASSED’, ’IN_PROGRESS’, ‘FAILED’, ‘SKIPPED’)	Значення за замовчуванням: IN_PROGRESS
type	VARCHAR(255)	Необов’язкове поле
system_out	VARCHAR(255)	Необов’язкове поле
test_suite_id	BIGINT	Зовнішній ключ
time	DOUBLE	Необов’язкове поле

Таблиця failure містить дані про тести, які не завершились успіхом. Ця таблиця містить повідомлення про помилки, за якими групуються тести. В таблиці 3.10 наведено інформацію про поля таблиці БД failure.

Таблиця 3.10

Таблиця failure

Failure		
Ім’я поля	Тип даних	Характеристики
Id	BIGINT	Первинний ключ
message	LONGTEXT	Необов’язкове поле
stack_trace	LONGTEXT	Необов’язкове поле
test_result_after_recheck	ENUM(‘PASSED’, ’IN_PROGRESS’, ‘FAILED’, ‘SKIPPED’)	Значення за замовчуванням: IN_PROGRESS
type	VARCHAR(255)	Необов’язкове поле

В таблиці 3.11 міститься інформація про поля таблиці БД skipped.

Таблиця 3.11

Таблиця skipped

Skipped		
Ім’я поля	Тип даних	Характеристики
Id	BIGINT	Первинний ключ
test_result_after_recheck	ENUM(‘PASSED’, ’IN_PROGRESS’, ‘FAILED’, ‘SKIPPED’)	Значення за замовчуванням: IN_PROGRESS

Таблиця skipped використовується для зберігання даних про тести, що не були пропущені. Тести можуть бути пропущені, якщо передумова запуску тестів була виконана не успішно.

3.4. Опис реалізованих алгоритмів функціонування системи

Основним розробленим алгоритмом у фреймворку для автоматизованого тестування програмного забезпечення є алгоритм ініціалізації драйвера для автомитизації дій в браузері або в Андроїд-емуляторі.

Цей алгоритм використовується кожного разу при запуску тестів для перевірки графічного інтерфейсу. Параметри ініціалізації драйверу застосовуються відповідно до конфігураційного файлу.

На рис. 3.7 наведено блок-схему алгоритму для ініціалізації драйвера.

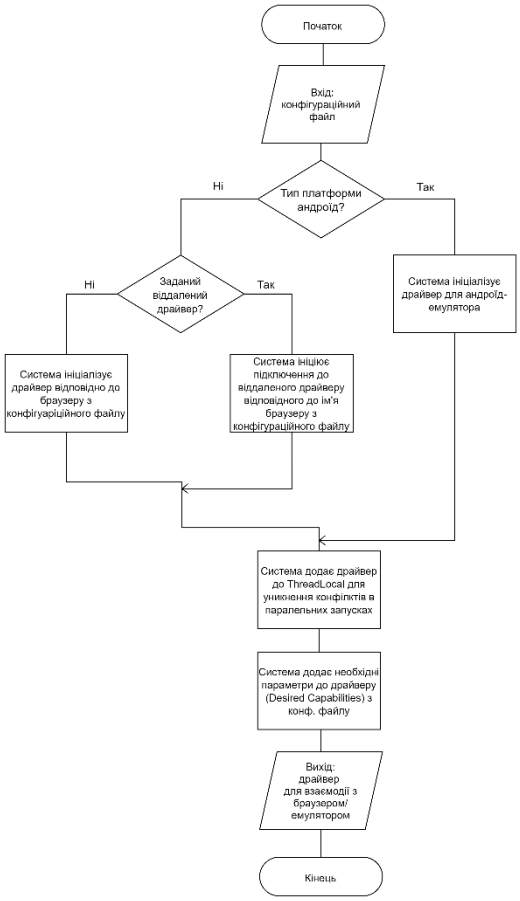


Рис. 3.7. Блок-схема алгоритму ініціалізації драйверу

Залежно від типу платформи з конфігураційного файлу, ініціалізується відповідний драйвер. Також в залежності від того чи вказана адреса віддаленого драйверу (браузерної ферми) чи не вказана, створюється драйвер для браузерної ферми чи для браузеру встановленого локально відповідно до назви браузеру в конфігураційному файлі.Для локального запуску необхідно оновити браузери до останньої версії. Після цього драйвер завантажується в ThreadLocal. ThreadLocal – це спеціальна колекція у Java, що забезпечує потокозахищеність. Останнім етапом додаються Desired capabilities. Для Андроїд-емулятора – це параметри емулятора, його id, пакет застосунку, що треба запустити тощо. Для браузерного драйверу – це headless/headed mode, розмір вікна тощо.

3.5. Висновки до розділу

У результаті написання третього розділу було зроблено наступне:

1) розглянуто архітектуру фреймворку;
2) опис функцій розробленого фреймворку;
3) описано структуру БД;
4) побудовано модель зв’язків сутностей БД;
5) розглянуто механізм ORM на прикладі серверної частини фреймворку;
6) розглянуто пакетну структуру серверної та основної частини;
7) було описано алгоритм для ініціалізації драйверу для запуску тестів на Web та Android платформах.

4. ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ФРЕЙМВОРКУ

У цьому розділі поставлено за мету описати особливості реалізації серверної частини, основної частини (TestCoreLib), CLI (testcorecli) та графічного клієнту, особливості тестування і можливі шляхи покращення фреймворку для автоматизованого тестування програмного забезпечення, загальний опис розгортання.

4.1. Особливості реалізації

Фреймворк для автоматизованого тестування програмного забезпечення було написано з використанням найкращих практик «чистої» архітектури та коду, а саме: уникання повторюваних фрагментів коду, використання шаблонів проєктування, використання зрозумілих змінних, відсутність антипатернів, рефакторинг та написання юніт-тестів. Такий підхід дозволяє додавати нові модулі до системи та запроваджувати зміни без значних зусиль та з мінімальним ризиком появи дефектів.

4.1.1. Серверна частина фреймворку

Серверна частина фреймворку повинна мати змогу взаємодіяти з БД та реалізовувати серверну логіку для групування результатів тестів за спільними помилками. Така взаємодія може відбуватися за допомогою прямих SQL-запитів через JDBC до БД, або за допомогою використання ORM, що забезпечує безпечність такої взаємодії та зменшення трудових затрат на реалізацію persistence шару. В якості ORM було вибрано Spring Data JPA, що є частиною екосистеми Spring Boot. Загальна архітектура серверної частина описана в п. 3.2 разом з пакетною організацією. Нижче наведено особливості реалізації кожного шару серверної частини.

Persistence шар складається з інтерфейсів репозиторіїв, що дозволяють робити базові CRUD операції. Всі інтерфейси наслідують JpaRepository, що

є частиною ORM Spring Data JPA і при необхідності містять інші методи для пошуку за полями сутностей в БД.

Шар бізнес-логіки реалізовує основну логіку серверної частини та звертається до persistence шару для взаємодії з БД через нього.

Шар представлення містить REST контролери, що є API, тобто інтерфейсом взаємодії з усією серверною частиною через HTTP.

4.1.2. Основна частина фреймворку

Основна частина фреймворку компілюється у JAR файл та підключається як бібліотека до шаблону проєкту автоматизованого тестування, що, у свою чергу, вивантажується запитом до серверної частини.

Цей складник фреймворку містить основні засоби для створення тестів, кросплатформного тестування, тестування API, конфігурації тестових запусків, роботи з page object model структурою з можливістю створення вкладених компонент.

В процесі реалізації даної частини використовувалися шаблони проєктування для забезпечення «чистоти» коду. Серед них «Компонувальник», «Стратегія», «Одинак» та «Фабричний метод».

Для забезпечення кросплатформності Web та Android тестів використовується модуль PageInitializer з статичним методом initPage, що вибирає реалізацію абстрактного класу відповідного page object на основі файлу конфігурації, що задається як файл з розширенням properties та анотації класу наслідника для взаємодії з відповідними графічними елементами. Такий гнучкий підхід можливий, завдяки механізму рефлексії у Java.

Для тестування API використовувався пакет Jackson. Розроблено сервіси для тестування на основі шаблонів запитів та відповідей, де пропущені дані задаються у вигляді знаку питання і використовуються для підстановки значень. На рис. 4.1 наведено прикладу даного шаблону.

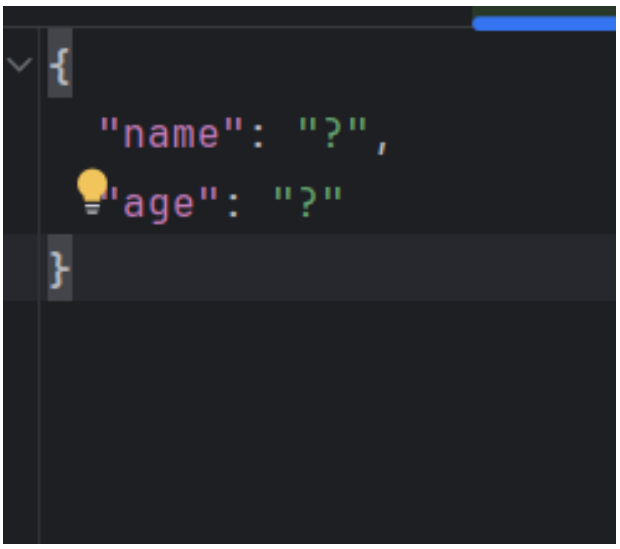


Рис. 4.1. Приклад файлу шаблону для запиту/відповіді

Для забезпечення можливості запусків тестів у декількох потоках без конфліктів з драйверами браузера використовується ThreadLocal обгортка в поєднані з використанням класу DriverHelper, що є «Одинаком».

Також за допомогою класу DesiredCapabilites loader реалізована логіка завантаження опцій браузера та емулятора, таких як: headless режим, розмір вікна, унікальний індефікатор емулятора та інші його властивості. Для користувача ж цей механізм працює «під капотом», все, що потрібно задати – це параметри в конфігураційному файлі.

4.1.3. Інструмент командного рядка та графічний інтерфейс

Інструмент командного рядка (CLI) дозволяє взаємодіяти із серверною частиною за допомогою команд. При цьому команди та їх аргументи передаються через параметри виклику, що дозволяє використовувати CLI в CI/CD конвейєрах.

В основі реалізації інструменту командного рядка лежить шаблон проєктування «Стратегія», де на основі першого вхідного аргументу, що визначає тип операції, використовується необхідна реалізація/поведінка. Для надсилання запитів на серверну частину та отримання відповіді використовується пакет okhttp3 [21].

CLI було скомпільовано у jar файл і створено скрипт для інсталяції інструменту командного рядка. Було створено powershell та bash скрипти відповідно до різних ОС, що дозволяють викликати командний рядок з будь-якої директорії файлової системи.

CLI складається з двох шарів: шару представлень (presentation layer) та бізнес-логіки.

Графічний інтерфейс реалізовано за допомогою JavaFX. Він є обгорткою до CLI. Тобто для проведення операцій з серверною частиною використовуються виклики до CLI, що загальнодоступний у системі.

4.2. Опис реалізованого користувацького інтерфейсу

Розроблене ПЗ є комлексним, тобто таким, що складається з багатьох складників. Користувацький інтерфейс складається з CLI та настільного кросплатформного додатку.

4.2.1. Опис реалізованого інструмента командного рядка

Для взаємодії з інструментом командного рядка використовується наступний формат команд: testcorecli *args.

Нижче розглянемо можливі аргументи командного рядка.

Перший по порядку аргумент визначає тип операції. Важливим моментом є використання логіну та паролю для виклику команд: якщо користувач був авторизований раніше, його облікові дані зберігаються на боці клієнтської частини для повторного використання, що дозволяє не передавати облікові дані для кожної операції.

Для отримання довідки по всім можливим командам використовується testcorecli help. Також в довідці зазначено, що для всіх команд окрім логіну та реєстрації використання логіну та паролю не обов’язкове. Використання логіну та паролю для окремої операції має сенс, якщо якась специфічна операція має бути здійснена під іншим користувачем, при чому основний користувач при цьому не змінюється.

На рис. 4.2 наведено приклад виконання команди для отримання довідки користувача.

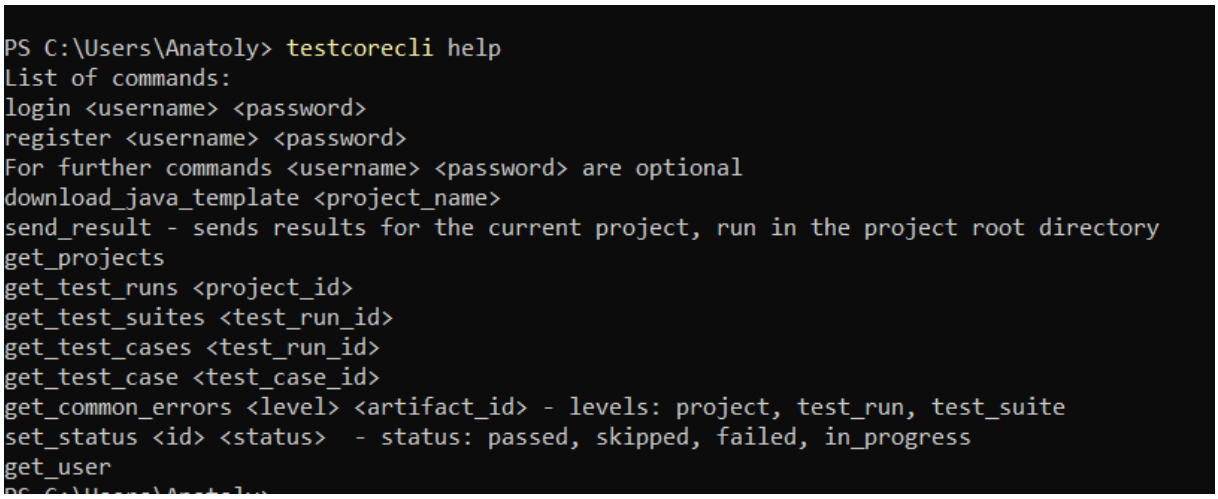


Рис. 4.2. Приклад виконання команди для отримання довідки користувача

Для реєстрації за допомогою CLI потрібно виконати: testcorecli register <username> <password>. Пароль має задовольняти вимогам описаним у табл. 3.2 раніше. Також ім’я користувача має бути унікальни, тобто таким, якого ще не було в системі.

Приклад процесу реєстрації наведено на рис. 4.3.

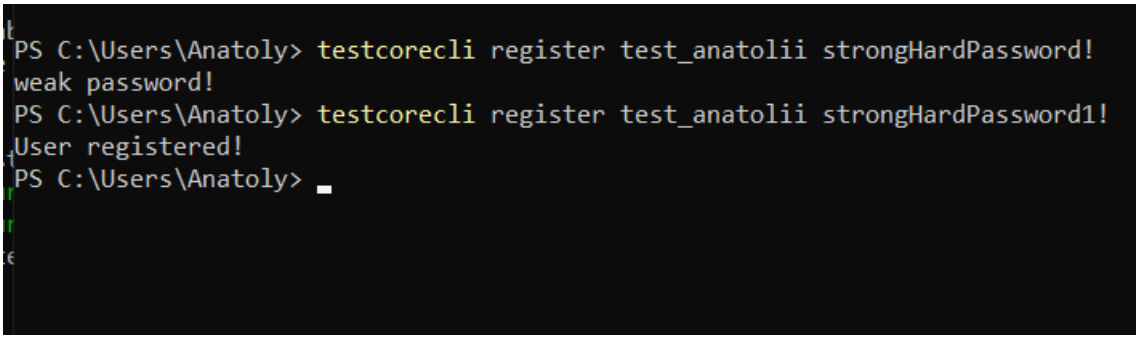


Рис. 4.3. Приклад процесу реєстрації користувача у системі за допомогою використання інструменту командного рядка

Для входу користувача у систему використовується команда testcorecli login <username> <password>. Також дана команда може

використовуватися для зміни користувача. На рис. 4.4 продемонстровано успішну спробу входу.

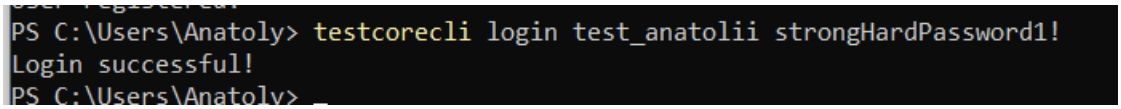


Рис. 4.4. Приклад процесу авторизації користувача у системі за допомогою використання інструменту командного рядка

На рис. 4.5 наведено приклад, що демонструє необов'язковість використання логіну та паролю для виконання команд.

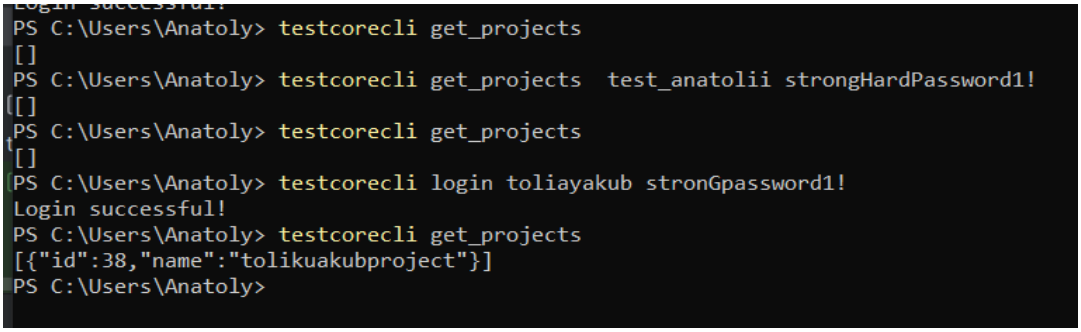


Рис. 4.5. Приклад отримання проєктів користувача за допомогою використання інструменту командного рядка

Для того, щоб ініціалізувати проєкт для автоматизованого тестування використовується команда testcorecli download_java_template <назва проєкту >. На рис. 4.6 наведено приклад ініціалізації тестового проєкту з шаблону.

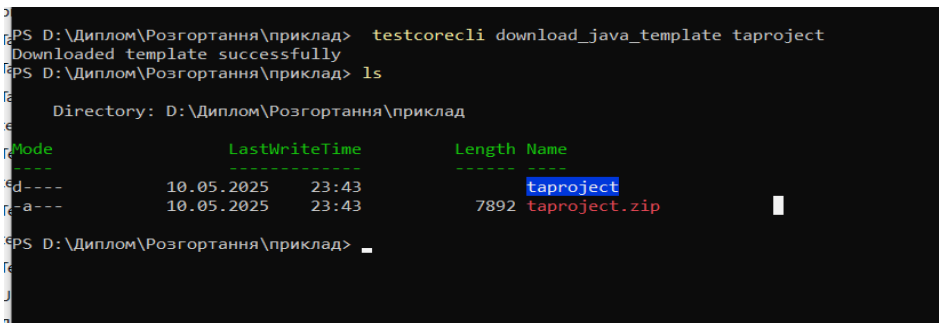


Рис. 4.6. Приклад ініціалізації тестового проєкту з шаблону

На стороні CLI відбувається розпаковування архіву та перейменування теки відповідно до заданого аргументу.

Для отримання імені поточного користувача потрібно виконати команду testcorecli get_user без жодних параметрів, як на рис 4.7.

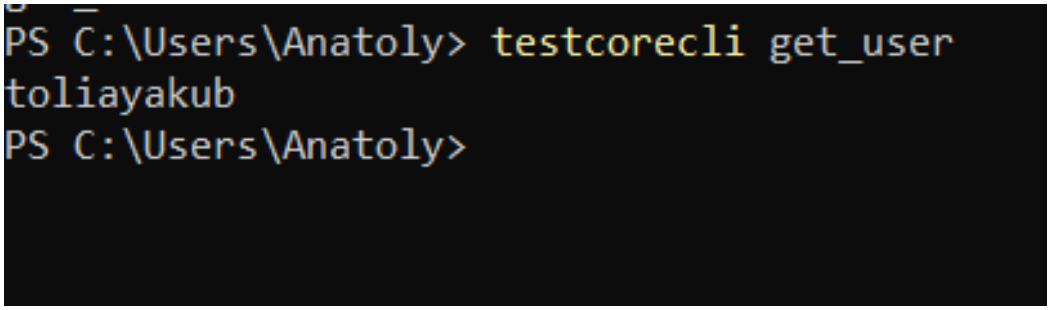


Рис. 4.7. Приклад виконання команди для отримання ім’я поточного користувача

Більш специфічні особливості CLI описано у керівництві користувача.

4.2.2. Опис реалізованого графічного інтерфейсу

При першому запуску програми користувача буде направлено на вікно атворизації. При чому, якщо був виконаний вхід з командного рядка чи вхід з графічного інтерфейсу, то користувач буде перенаправлений одразу на вікно проєктів. Звичайно, наявна можливість змінити користувача і також потрапити на вікно авторизації.

Групування помилок працює окремо на всіх рівнях. Тобто для рівня проєкту, шукаються спільні помилки в результатах тестів, що були здійснені за весь час в середині цього проєтку. Для тестових запусків знаходяться спільні помилки по всім тестовим наборам в середині тестового запуску і групуються в межах груп спільних помилок без повторів. Для тестового проєкту також групування відбувається без повторів назв тестів.

Вікно авторизації містить напис привітання. Та дві кнопки: для логіну та реєстрації. При наведенні на якусь з кнопок вони збільшують розмір, у

них з’являється тінь та курсор миші перетворюється на вказівний. Знімок екрана вікна авторизації наведений на рис. 4.8.

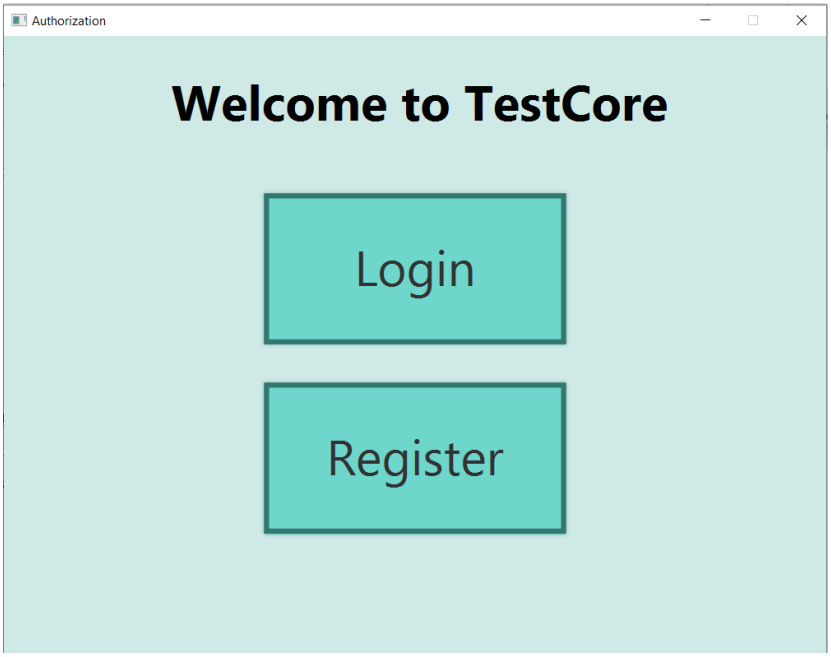


Рис. 4.8. Вікно авторизації

На рис. 4.9 наведено зображення вікна реєстрації користувача.

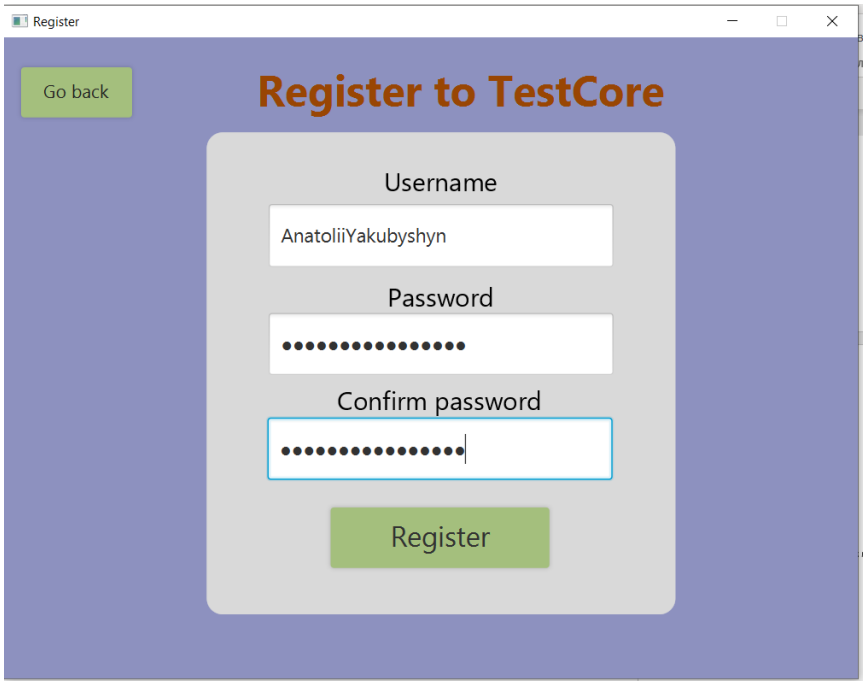


Рис. 4.9. Вікно реєстрації користувача

Вікно реєстрації містить у верхньому лівому куті кнопку для повернення назад до вікна авторизації. По центру зверху містить текст, що дає зрозуміти користувачу на якій сторінці він знаходиться. Основну частину вікна займає форма реєстрації. Форма реєстрації містить поле для вводу для ім’я користувача та два поля з прихованим відображенням вмісту для введення паролю та підтвердження паролю. Знизу форми міститься кнопка для завершення процесу реєстрації (відправки форми).

Як і на сторінці реєстрації у вікні логіну в лівому верхньому куті міститься кнопка для повернення на вікно авторизації. По центру зверху міститься текст червоного кольору – заголовок вікна. Основний вміст вікна займає форма для входу. Форма містить поля для ім’я користувача та паролю та кнопку «Login». Вміст поля для введення паролю не видимий для користувача, кожен символ маркується.

На рис. 4.10 зображено вікно логіну.

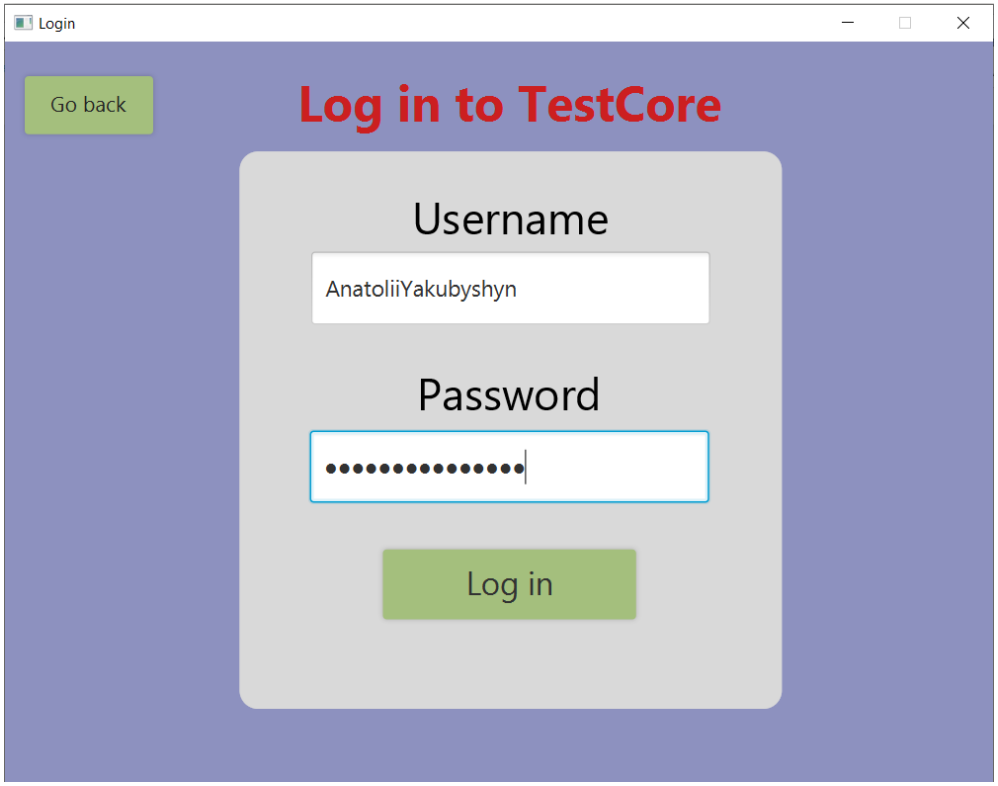


Рис. 4.10. Вікно логіну

Вікно проєктів містить у правому верхньому кутку кнопку для зміни користувача. Також по центру зверху вікна міститься привітання користувача. Під ним іде список проєктів.

На рис. 4.11 наведено знімок екрану вікна проєктів.

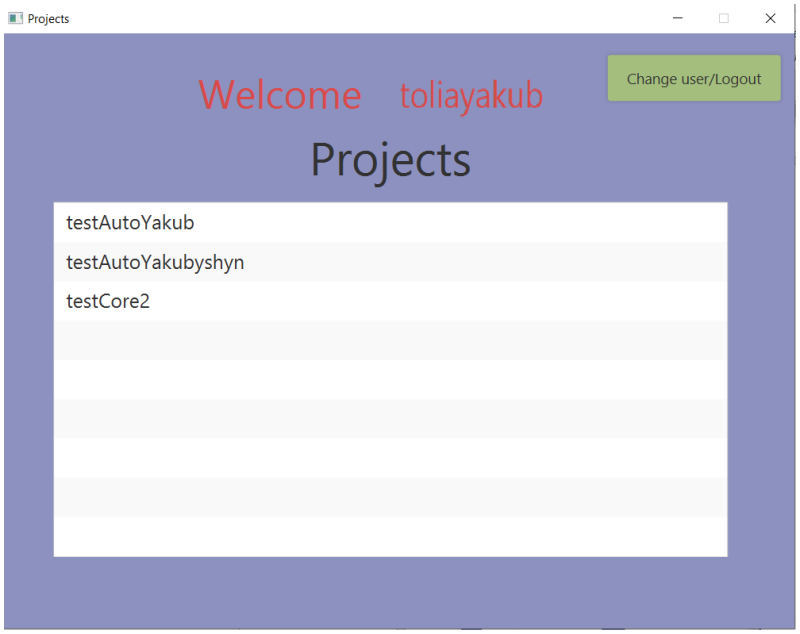


Рис. 4.11. Вікно проєктів

На рис. 4.12 наведено приклад відображення інформації про тестовий проєкт.

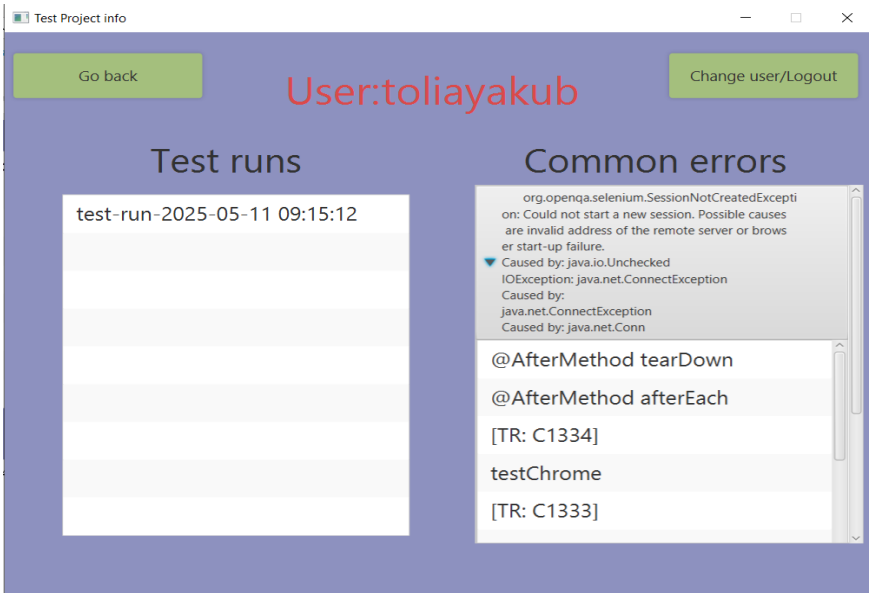


Рис. 4.12. Вікно виведення інформації про проєкт

Вікно містить кнопку повернення на попередній рівень зліва зверху. Справа зверху міститься кнопка для зміни користувача. Також зверху по центру є текст, який допомагає зрозуміти, під якими користувачем відбувся вхід. Основний вміст сторінки складається з таблиці тестових запусків та таблиці групування тестів за спільними помилками.

Кожну групу спільних помилок можна згорнути та розгорнути. Вміст групи має динамічний розмір завдяки смуги прокручування.

Сторінка інформації про тестові запуски майже ідентична до попередньої. Єдина відмінність, що замість таблиці тестових запусків сторінка містить таблицю тестових наборів. На рис. 4.13 наведено приклад вікна інформації про тестовий запуск.

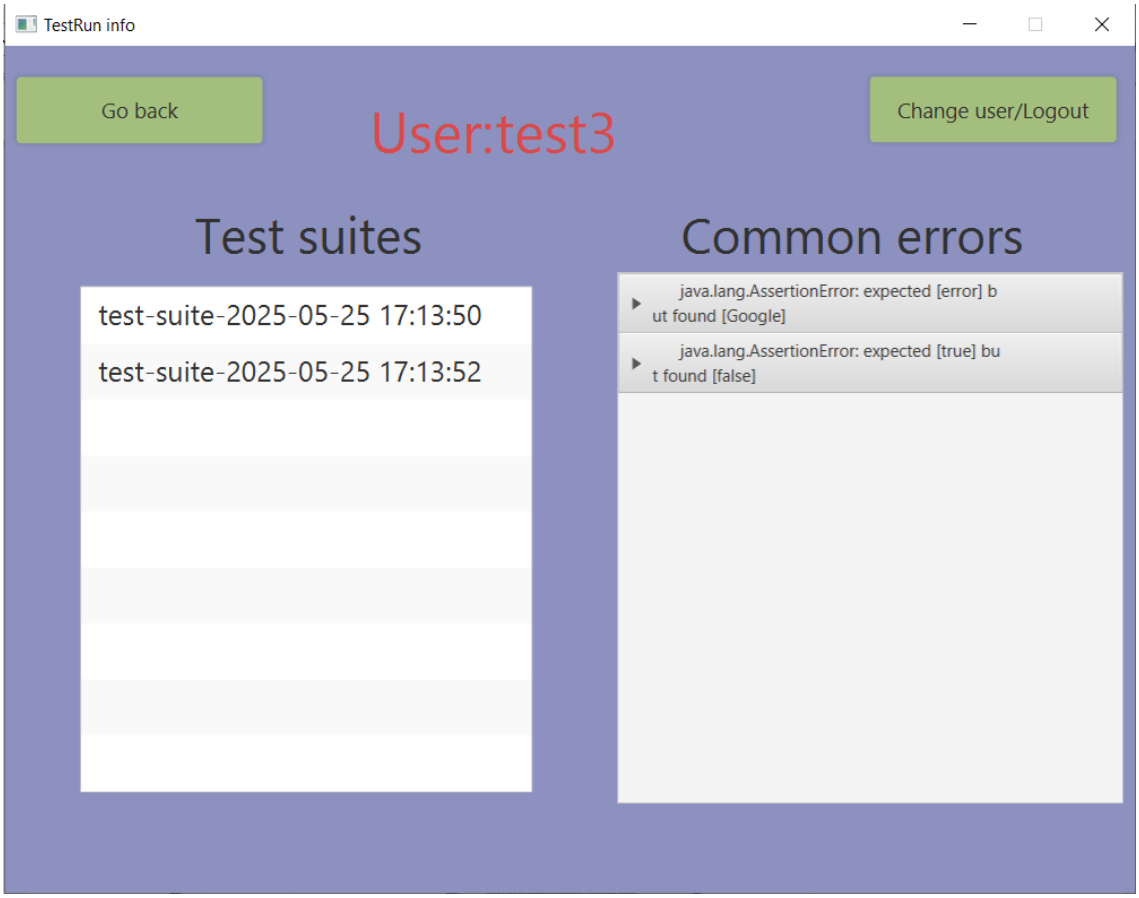


Рис. 4.13. Вікно виведення інформації про testrun

Сторінка інформації про тестовий набір містить ті самі елементи, що сторінка про інформацію про тестовий запуск, але замість таблиці тестових

наборів містить таблицю тестових результатів. Таблиця тестових результатів містить наступні колонки: id, ім’я, статус після повторної перевірки тесту. Статус після повторної перевірки можна змінити натиснувши на відповідну клітинку та вибрати необхідну опцію. На рис. 4.14 наведено вікно виведення інформації про тестовий набір.

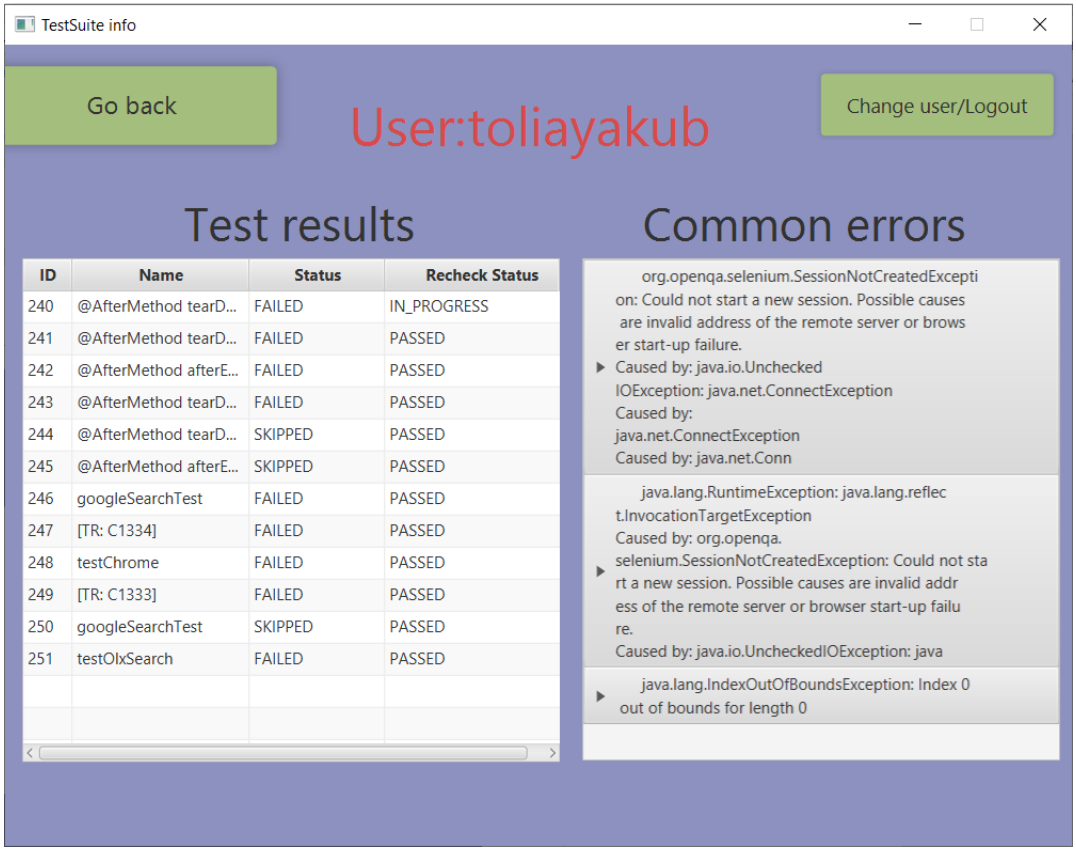


Рис. 4.14. Вікно виведення інформації про TestSuite

4.3. Тестування фреймворку

В ході розроблення фреймворку тестування відіграло важливу роль у забезпеченні якості кінцевого продукту. Було написано юніт та інтеграційні тести для основної частини фреймворку, CLI та серверної частини. Це дозволило робити зміни в існуючій функціональності та додавати нові характеристики до ПЗ часто. Окрім юніт тестування було також проведено наскрізне тестування.

За допомогою наскрізного тестування відбувалася перевірка функціональності від початку до кінця зі сторони кінцевого користувача.

Тому основний фокус у наскрізному тестуванні саме припав на тестування через графічний інтерфейс та CLI.

На рис. 4.15 подано результати запуску модульних та інтеграційних тестів для серверної частини. Було написано 11 тестів з яких всі пройшли успішно у фінальній версії ПЗ.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  15.889 s
[INFO] Finished at: 2025-05-14T19:01:59+03:00
[INFO] ------------------------------------------------------------------------
[WARNING] The requested profile "with-http-repo" could not be activated because it does not exist.
PS D:\Диплом\testCoreBackend>
```

Рис. 4.15. Результати автоматизованого тестування серверної частини фреймворку

На рис. 4.16 подано результати автоматизованого тестування основної частини. Було написано 22 модульних та інтеграційних тести, що пройшли успішно.

```
[INFO] Results:
[INFO]
[INFO] Tests run: 22, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  55.053 s
[INFO] Finished at: 2025-05-14T19:19:57+03:00
[INFO] ------------------------------------------------------------------------
```

Рис. 4.16. Результати автоматизованого тестування основної частини фреймворку

Було написано 7 модульних тестів для інструменту командного рядка, що успішно пройшли у фінальній версії ПЗ. Результати наведен на рис. 4.17.

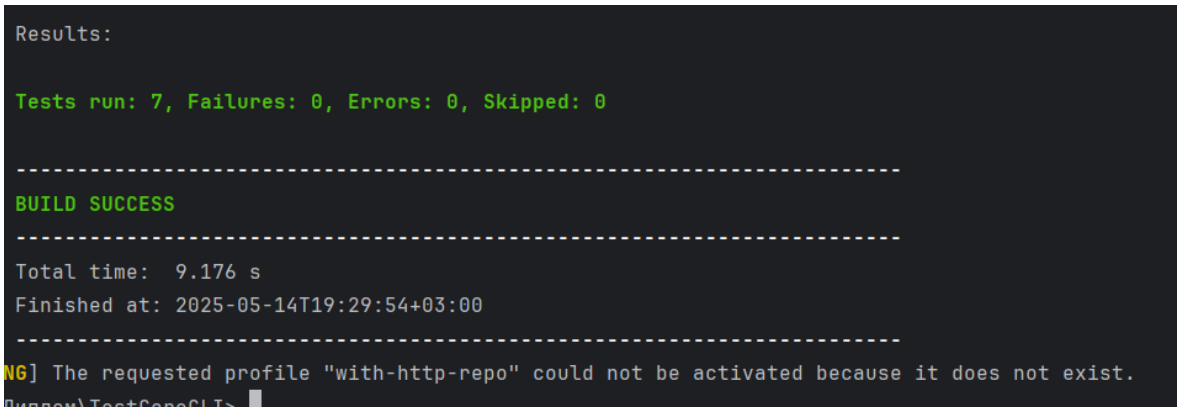


Рис. 4.17. Результати автоматизованого тестування інструменту командного рядка

Ручне тестування було проведене для виявлення помилок інтеграції окремих компонент у загальну систему. Було проведено димне тестування, тому тест-кейсів написано не багато.

У таблиці 4.1 наведено тест-кейси для димного наскрізного тестування.

Таблиця 4.1

Таблиця тест-кейсів для димного наскрізного тестування

№	Назва	Передумова	Кроки виконання	Очікуваний результат
1	Валідація виведення помилки про слабкий пароль (1)	Відкрита сторінка реєстрації	1. Ввести в username – user1234 2. Ввести пароль: HardPa! 3. Ввести в поле Confirm password: HardPa!	Інформаційний банер про слабкий пароль з’являється на екрані

Продовження табл. 4.1

2	Перевірка повідомлення про пустий userName	Відкрита сторінка реєстрації	1. Ввести пароль: HardPa! 2. Ввести в поле Confirm password: HardPa!	Інформаційний банер про пустий username з'являється на екрані
3	Валідація виведення помилки про слабкий пароль (2)	Відкрита сторінка реєстрації	1. Ввести в username – user1234 2. Ввести пароль: hardpass1! 3. Ввести в поле Confirm password: hardpass1!	Інформаційний банер про слабкий пароль з'являється на екрані
4	Валідація виведення помилки про слабкий пароль (3)	Відкрита сторінка реєстрації	1. Ввести в username – user1234 2. Ввести пароль: HARDPASSWORD1! 3. Ввести в поле Confirm password: HARDPASSWORD1!	Інформаційний банер про слабкий пароль з'являється на екрані
5	Валідація виведення помилки про неспівпадіння паролей	Відкрита сторінка реєстрації	1. Ввести в username – user1234 2. Ввести пароль: hARDPASSWORD1! 3. Ввести в поле Confirm password: HARDpASSWORD1!	Інформаційний банер про неспівпадіння паролей з'являється на екрані

Тест-кейс номер 2 допоміг виявити дефект у програмному продукті, який було згодом виправлено.

4.4. Рекомендації щодо подальшого вдосконалення

У результаті аналізу створеного фреймворку для автоматизованого тестування програмного забезпечення було визначено потенційні вдосконалення системи.

Одна з них це можливість автоматизованого тестування настільних додатків. Це б допомогло захопити більшу аудиторію користувачів, оскільки фреймворк виконував б більше задач.

Також для отримання фінансової вигоди з програмного продукту потрібно додати модуль платіжної системи. Це може бути або платна підписка, або платна реєстрація з безлімітним використанням продукту.

Третім покращенням є створення Low code/No code підтримки для написання автоматизованих тестів. Це б дозволило б фахівцям з нижчим технічним рівнем автоматизовувати процес тестування.

Четвертим покращенням є функціональність порівняння вікна браузера з очікуваним зображуванням. Це може допомогти виявити дефекти у стилях вебсайтів, послідовності розміщення елемнтів на сторінці або вікні.

Зрозуміло, що потрібно слідкувати за новими тенденціями в галузі інформаційних технологій та відповідно адаптувати ПЗ до них.

4.5. Висновки до розділу

У ході написання цього розділу описано особливості реалізації фреймворку, описано реалізований користувацький інтрфейс, що поділяється на дві частини: інструмент командного рядка та графічний інтерфейс у вигляді настільного додатку, процес тестування та рекомендації щодо подальшого вдосконалення.

Під час опису інструменту командного рядка було розглянуто основний принцип роботи з ним, особливості передачі аргументів та наведено приклади запусків декількох команд.

Під час опису графічного інтерфейсу було розглянуто всі вікна настільного клієнта та візуальні елементи всередині них.

Розглянуто автоматизоване та ручне тестування складників системи. Наведено кількісну статистику по результатам запусків тестів.

Наведено потенційні напрямки для подальшого вдосконалення.

ВИСНОВКИ

Метою даного дипломного проєкту є розроблення фреймворку для написання якісних автоматизованих тестів для перевірки коректності роботи ПЗ та їх подальшого оновлення. Під час виконання дипломного проєкту було розроблено фреймворк для автоматизованого тестування програмного забезпечення, що дозволило досягнути поставлену мету.

У процесі виконання дипломного проєкту було проаналізовано предметну область, проаналізовано існуючи рішення та обґрунтовано доцільність розробки даного ПЗ.

На етапі проєктування ПЗ було обрано засоби реалізації. Було обґрунтовано доцільність використання мови Java та БД MySQL. Також вибрано платформу для запуску тестів TestNG та Maven як систему збірки. Розглянуто й інші альтернативні засоби реалізації.

ПЗ було створено на основі комплексної архітектури в основі якої лежить архітектура клієнт-сервер.

ПЗ містить наступні частини:

1) основна частина фреймворку;

2) серверна частина;

3) настільний графічний інтерфейс;

4) інструмент командного рядка.

ПЗ було протестовано ручними перевірками та за допомогою автоматизованих тестів.

ПЗ відповідає висунутим до нього загальним вимогам та вимогам до безпеки.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Cypress. Why Cypress [Електронний ресурс]. – Режим доступу: https://docs.cypress.io/app/get-started/why-cypress – Дата доступу: 27.05.2025 р.
2. Selenide. Documentation [Електронний ресурс]. – Режим доступу: https://selenide.org/documentation.html – Дата доступу: 27.05.2025 р.
3. Playwright. Java [Електронний ресурс]. – Режим доступу: https://playwright.dev/Java/ – Дата доступу: 27.05.2025 р.
4. Postman. Overview [Електронний ресурс]. – Режим доступу: https://learning.postman.com/docs/introduction/overview/ – Дата доступу: 27.05.2025 р.
5. AWS. What is Java [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/what-is/java – Дата доступу: 27.05.2025 р.
6. TestNG [Електронний ресурс]. – Режим доступу: https://testng.org/ – Дата доступу: 27.05.2025 р.
7. Maven [Електронний ресурс]. – Режим доступу: https://maven.apache.org – Дата доступу: 27.05.2025 р.
8. Selenium. [Електронний ресурс]. – Режим доступу: https://www.selenium.dev – Дата доступу: 27.05.2025 р.
9. Appium [Електронний ресурс]. – Режим доступу: https://github.com/appium/appium?tab=readme-ov-file – Дата доступу: 27.05.2025 р.
10. OpenJFX [Електронний ресурс]. – Режим доступу: https://openjfx.io – Дата доступу: 27.05.2025 р.
11. GeeksforGeeks. Introduction to Java Swing [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/introduction-to-java-swing/ – Дата доступу: 27.05.2025 р.
12. Electron [Електронний ресурс]. – Режим доступу: https://www.electronjs.org/ – Дата доступу: 27.05.2025 р.

13. Qt [Електронний ресурс]. – Режим доступу: https://www.qt.io/ – Дата доступу: 27.05.2025 р.
14. Node.js [Електронний ресурс]. – Режим доступу: https://nodejs.org/uk – Дата доступу: 27.05.2025 р.
15. Django [Електронний ресурс]. – Режим доступу: https://www.djangoproject.com – Дата доступу: 27.05.2025 р.
16. Spring Boot [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-boot – Дата доступу: 27.05.2025 р.
17. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/ – Дата доступу: 27.05.2025 р.
18. MongoDB Documentation [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/docs/ – Дата доступу: 27.05.2025 р.
19. W3Schools. MySQL [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/MySQL/default.asp – Дата доступу: 27.05.2025 р.
20. Spring Data JPA [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-data-jpa#overview – Дата доступу: 27.05.2025 р.
21. OkHttp [Електронний ресурс]. – Режим доступу: https://square.github.io/okhttp/ – Дата доступу: 27.05.2025 р.
22. QATestLab. HTTP protocol: what and where to test [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/http-protocol-what-and-where-to-test/ – Дата доступу: 27.05.2025 р.
23. IT Notes Wiki. Postman Newman run collections [Електронний ресурс]. – Режим доступу: https://www.it-notes.wiki/tools/postman-newman-run-collections/ – Дата доступу: 27.05.2025 р.
24. IT-Rating. Оптимізація роботи з базами даних за допомогою ORM [Електронний ресурс]. – Режим доступу: https://it-

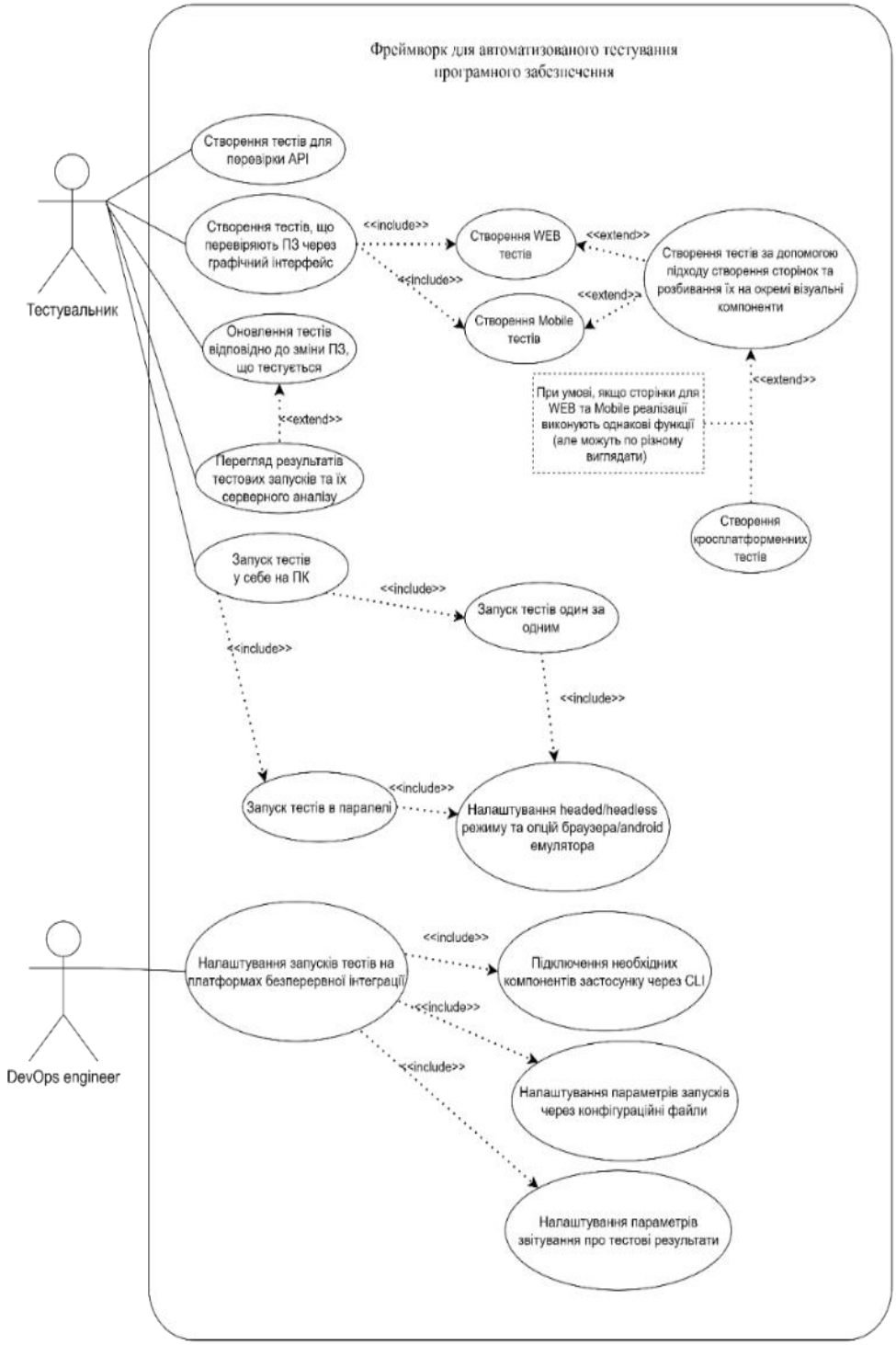
rating.ua/optimizatsiya-roboti-z-bazami-danih-za-dopomogoyu-orm-object-relational-mapping – Дата доступу: 27.05.2025 р.

25. GitHub. Actions [Електронний ресурс]. – Режим доступу: https://github.com/features/actions – Дата доступу: 27.05.2025 р.
26. AWS. Docker [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/docker/ – Дата доступу: 27.05.2025 р.
27. Express.js [Електронний ресурс]. – Режим доступу: https://expressjs.com/uk/ – Дата доступу: 27.05.2025 р.
28. AWS. What is Java Runtime Environment [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/what-is/java-runtime-environment/ – Дата доступу: 27.05.2025 р.
29. QATestLab. What is code refactoring and why is it important for testers [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/what-is-code-refactoring-and-why-is-it-important-for-testers/ – Дата доступу: 27.05.2025 р.
30. QALight. Модульне тестування [Електронний ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/modulne-testuvannya/ – Дата доступу: 27.05.2025 р.

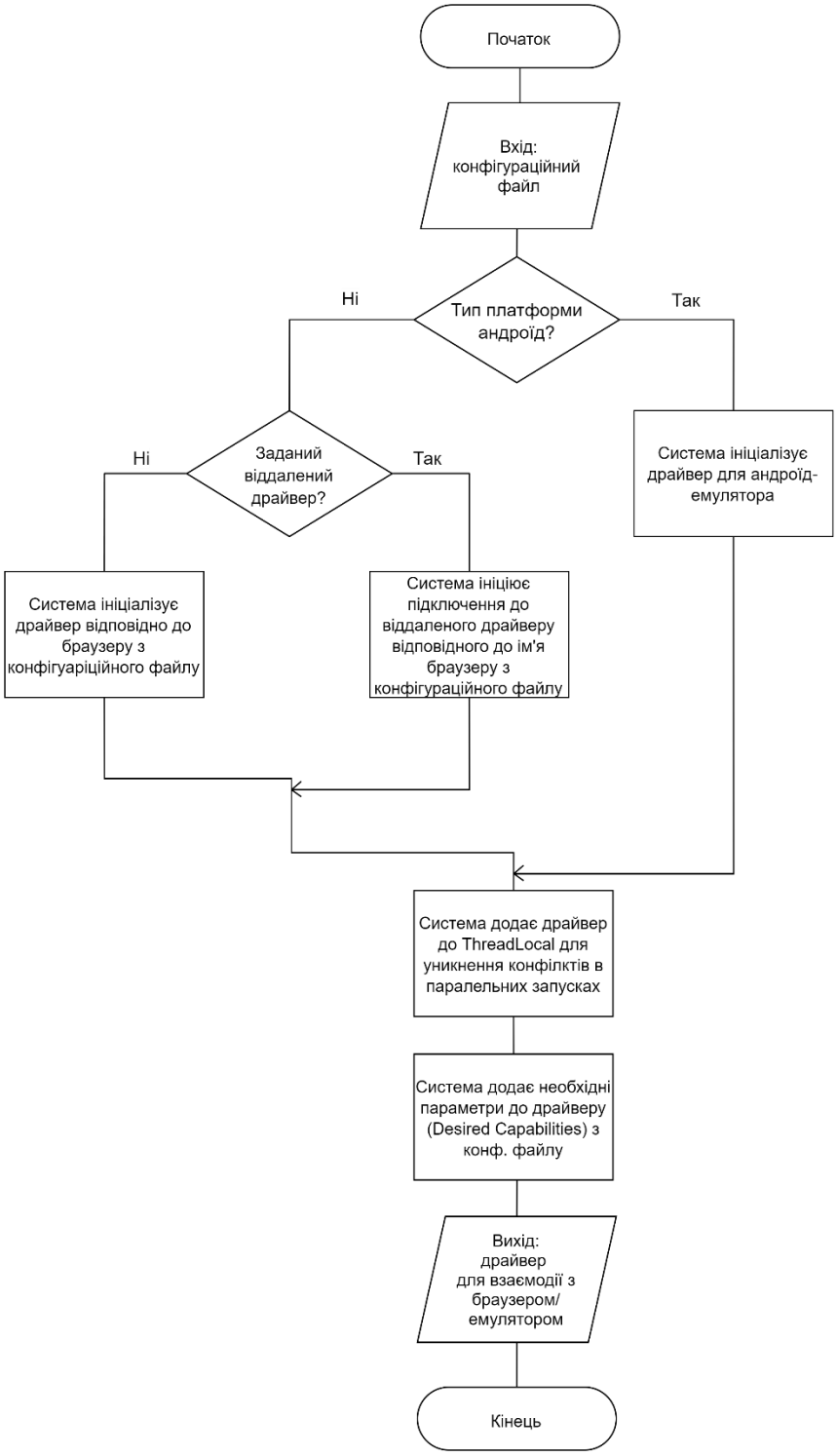
ДОДАТКИ

Додаток 1

Копії графічних матеріалів

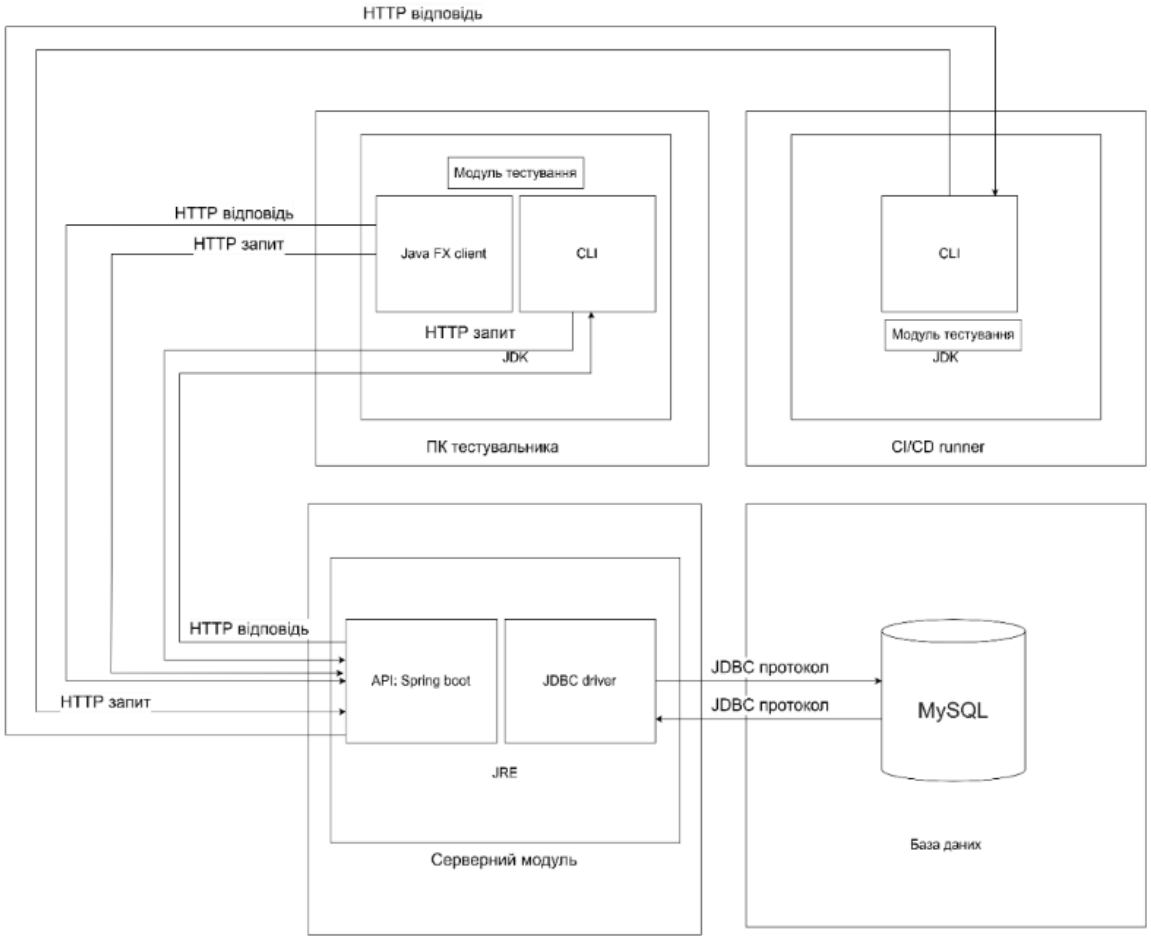


ДП.045490-06-99
Фреймворк для автоматизованого тестування програмного забезпечення. Функціональність фреймворку. UML-діаграма прецедентів



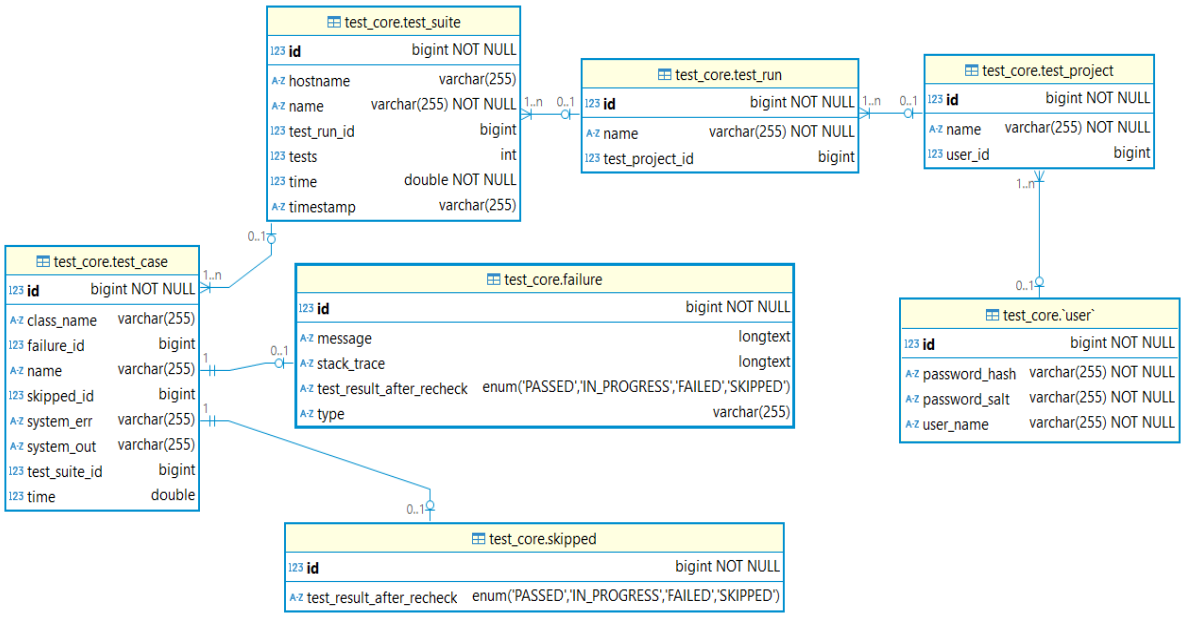
ДП.045490-07-99

Фреймворк для автоматизованого тестування програмного забезпечення. Алгоритм ініціалізації драйверу. Блок-схема



Архітектура фреймворку

Якубишин А.С., група КП-12



Діграма зв’язків між сутностями

Якубишин А.С., група КП-12

Додаток 2
Лістинг програми

```
package com.yakubt.testCoreBackend.presentation;

import com.yakubt.testCoreBackend.business.auth.RegisterStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.yakubt.testCoreBackend.business.auth.AuthService;

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

@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private AuthService authService;

    private      final      Map<RegisterStatus,      ResponseEntity<String>>
registerResponse = Map.of(
            RegisterStatus.CREATED, new ResponseEntity<>("User registered!",
HttpStatus.CREATED),
            RegisterStatus.USERNAME_ALREADY_EXISTS, new ResponseEntity<>(
                    "userName already exists!", HttpStatus.BAD_REQUEST),
            RegisterStatus.WEAK_PASSWORD, new ResponseEntity<>(
                    "Weak password! Password should be at least 8 letters
length" +
                            " ,contains digit, at least one capital letter,
at least " +
                            "one  small  letter  and  a  special  character",
HttpStatus.BAD_REQUEST)
    );

    @PostMapping("/register")
    public  ResponseEntity<String>  register(@RequestParam  String  userName,
@RequestParam String password) {
        return          registerResponse.get(authService.register(userName,
password));
    }

    @PostMapping("/login")
    public  ResponseEntity<String>  login(@RequestParam  String  userName,
@RequestParam String password) {
        boolean success = authService.login(userName, password);
        return   success   ?   new   ResponseEntity<>("Login   successful!",
HttpStatus.OK) :
                new        ResponseEntity<>("Invalid        credentials!",
HttpStatus.FORBIDDEN);
    }
}

package                              com.yakubt.testCoreBackend.presentation;

import                                                  java.io.IOException;
import                                       java.nio.charset.StandardCharsets;
import                                                 java.time.LocalDateTime;

import              org.springframework.beans.factory.annotation.Autowired;
import                              org.springframework.http.ResponseEntity;
import                    org.springframework.security.core.Authentication;
import                            org.springframework.web.bind.annotation.*;
```

```
import                    org.springframework.web.multipart.MultipartFile;

import
com.yakubt.testCoreBackend.business.resultparse.XmlTestResultsParser;
import            com.yakubt.testCoreBackend.business.SaveTestSuiteService;

@RestController
@RequestMapping("/report")
public                class                ReportController                {

    @Autowired
    private            XmlTestResultsParser            xmlTestResultsParser;

    @Autowired
    private            SaveTestSuiteService            saveTestSuiteService;


    @PostMapping(value    =    "/upload",    params    =    {"testProjectName",
"testRunName",                                              "testSuiteName"})
    public ResponseEntity<String> uploadSurefireReport(@RequestParam("file")
MultipartFile                                                              file,
                                                    @RequestParam  String
testProjectName,
                                                    @RequestParam  String
testRunName,
                                                    @RequestParam  String
testSuiteName,
                                                    Authentication
authentication)                                                            {
        String                xmlContent                =                null;
        try                                                                {
            xmlContent         =         new         String(file.getBytes(),
StandardCharsets.UTF_8);
        }            catch            (IOException            e)            {
            throw                  new                  RuntimeException(e);
        }
        try                                                                {
            saveTestSuiteService.saveTestSuite(xmlContent,
authentication.getName(),           testProjectName,           testRunName,
                    testSuiteName);
            return  ResponseEntity.ok("Report  received  and  processed!");
        }            catch            (Exception            e)            {
            return   ResponseEntity.badRequest().body("Bad   request:   "   +
e.getMessage());
        }
    }

    @PostMapping(value    =    "/upload",    params    =    {"testProjectName",
"testRunName"})
    public ResponseEntity<String> uploadSurefireReport(@RequestParam("file")
MultipartFile                                                              file,
                                                    @RequestParam  String
testProjectName,
                                                    @RequestParam  String
testRunName,
                                                    Authentication
authentication)                                                            {
        return   uploadSurefireReport(file,   testProjectName,   testRunName,
                "test-suite-"        +        LocalDateTime.now().getNano(),
authentication);
    }
}
```

```
package com.yakubt.testCoreBackend.presentation;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import org.springframework.core.io.InputStreamResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("download")
public class TemplateDownloadController {
    @GetMapping("java_template")
    public ResponseEntity<InputStreamResource> downloadJavaZip() throws
IOException {

        File zipFile = new File("java_template.zip");

        InputStreamResource resource = new InputStreamResource(new
FileInputStream(zipFile));

        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION,
"attachment;filename=" + zipFile.getName())
                .contentLength(zipFile.length())
                .contentType(MediaType.parseMediaType("application/zip"))
                .body(resource);
    }
}

package com.yakubt.testCoreBackend.presentation;

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

import com.yakubt.testCoreBackend.models.Failure;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import com.yakubt.testCoreBackend.models.DTO.TestCaseDTO;
import com.yakubt.testCoreBackend.business.TestCaseService;
import com.yakubt.testCoreBackend.models.enums.TestResultAfterRecheck;

@RestController
@RequestMapping("user")
public class TestCaseController {

    @Autowired
    private TestCaseService testCaseService;

    @GetMapping("/testsuite/{id}/testcases")
    public ResponseEntity<List<TestCaseDTO>> getTestCases(@PathVariable Long
id) {
        List<TestCaseDTO> dtos =
testCaseService.getTestCasesBySuiteId(id).stream().map(
                el -> {
```

```
                    Failure emptyFailure = new Failure();
                    if (el.getFailure() != null) {
                        TestResultAfterRecheck   testResultAfterRecheck   =
el.getFailure()
                                .getTestResultAfterRecheck();

emptyFailure.setTestResultAfterRecheck(testResultAfterRecheck);
                    }

                        el.setFailure(emptyFailure);
                        return el;
                    }
        ).collect(Collectors.toList());
                    return ResponseEntity.ok(dtos);
                }

        @GetMapping("/testcase/{id}")
        public ResponseEntity<TestCaseDTO> getTestCase (@PathVariable Long
id){
            TestCaseDTO                         tdo                         =
testCaseService.getTestCaseDTOByID(id).orElse(null);
            return testCaseService.getTestCaseDTOByID(id)
                    .map(ResponseEntity::ok)
                    .orElse(ResponseEntity.notFound().build());
        }

        @PutMapping("/testcase/{id}")
        public ResponseEntity<Boolean> changeTestCaseStatus (@PathVariable
Long id,
                @RequestParam("status")
                        String status){
            Map<String, TestResultAfterRecheck> mapTestResults = Map.of(
                    "passed", TestResultAfterRecheck.PASSED,
                    "skipped", TestResultAfterRecheck.SKIPPED,
                    "failed", TestResultAfterRecheck.FAILED,
                    "in_progress", TestResultAfterRecheck.IN_PROGRESS);
            TestResultAfterRecheck         testResultAfterRecheck         =
mapTestResults.getOrDefault(status,
                    TestResultAfterRecheck.IN_PROGRESS);
            return
ResponseEntity.ok(testCaseService.setTestCaseStatusAfterRecheck(id,
testResultAfterRecheck));
        }
    }

package com.yakubt.testCoreBackend.presentation;

import java.util.Collections;
import java.util.List;
import java.util.Map;

import jakarta.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.yakubt.testCoreBackend.models.DTO.TestRunDTO;
import com.yakubt.testCoreBackend.models.TestProject;
```

```
import com.yakubt.testCoreBackend.persistence.TestProjectDao;
import
com.yakubt.testCoreBackend.business.commonerrors.GetCommonErrorsFromTestRun
Service;
import com.yakubt.testCoreBackend.models.TestRun;
import com.yakubt.testCoreBackend.persistence.TestRunDao;

@RestController()
@RequestMapping("user/projects")
public class TestRunsController {

    @Autowired
    private TestProjectDao testProjectDao;


    @Autowired
    private TestRunDao testRunDao;

    @Autowired
    private                                GetCommonErrorsFromTestRunService
getCommonErrorsFromTestRunService;

    @GetMapping("/{projectID}/testruns")
    @Transactional
    public ResponseEntity<List<TestRunDTO>> getProjectsByUser(@PathVariable
Long projectID,

Authentication authentication) {
        TestProject                        testProject                        =
testProjectDao.read(projectID).orElse(null);
        if (testProject == null || !testProject.getUser()
                .getUserName().equals(authentication.getName())) {
            return
ResponseEntity.badRequest().body(Collections.emptyList());
        }
        return ResponseEntity.ok(testProject.getTestRuns().
                stream().map(testRun -> new TestRunDTO(testRun.getId(),
                        testRun.getName())
                ).toList());
    }

    @GetMapping("/testruns/{testRunID}/common_errors")
    @Transactional
    public       ResponseEntity<List<Map.Entry<String,       List<String>>>>
getCommonErrors(
            @PathVariable Long testRunID,
            Authentication authentication) {
        TestRun testRun = testRunDao.read(testRunID).orElseThrow();
        if (testRun == null || !testRun.getTestProject().getUser()
                .getUserName().equals(authentication.getName())) {
            return
ResponseEntity.badRequest().body(Collections.emptyList());
        }
        return
ResponseEntity.ok(getCommonErrorsFromTestRunService.getCommonErrors(testRun
));

    }
}
package com.yakubt.testCoreBackend.presentation;

import java.util.Collections;
import java.util.List;
```

```
import java.util.Map;

import com.yakubt.testCoreBackend.models.DTO.TestSuiteDTO;
import com.yakubt.testCoreBackend.models.TestSuite;
import com.yakubt.testCoreBackend.persistence.TestSuiteDao;
import jakarta.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import
com.yakubt.testCoreBackend.business.commonerrors.GetCommonErrorsFromTestRun
Service;
import com.yakubt.testCoreBackend.models.TestRun;
import com.yakubt.testCoreBackend.persistence.TestRunDao;

@RestController()
@RequestMapping("user")
public class TestSuiteController {

    @Autowired
    private TestSuiteDao testSuiteDao;


    @Autowired
    private TestRunDao testRunDao;

    @Autowired
    private                                  GetCommonErrorsFromTestRunService
getCommonErrorsFromTestRunService;

    @GetMapping("testRun/{testRunID}/testSuites")
    @Transactional
    public                                  ResponseEntity<List<TestSuiteDTO>>
getProjectsByUser(@PathVariable Long testRunID,

Authentication authentication) {
        TestRun testRun = testRunDao.read(testRunID).orElse(null);
        if (testRun == null || !testRun.getTestProject().getUser()
                .getUserName().equals(authentication.getName())) {
            return
ResponseEntity.badRequest().body(Collections.emptyList());
        }
        return ResponseEntity.ok(testRun.getTestSuites().
                stream().map(testSuite                ->                new
TestSuiteDTO(testSuite.getId(),
                        testSuite.getName())
                ).toList());
    }

    @GetMapping("/testSuite/{testSuiteID}/common_errors")
    @Transactional
    public       ResponseEntity<List<Map.Entry<String,       List<String>>>>
getCommonErrors(
            @PathVariable Long testSuiteID,
            Authentication authentication) {
        TestSuite testSuite = testSuiteDao.read(testSuiteID).orElse(null);
```

```
        if            (testSuite            ==            null            ||
!testSuite.getTestRun().getTestProject().getUser()
                .getUserName().equals(authentication.getName())) {
            return
ResponseEntity.badRequest().body(Collections.emptyList());
        }
        return ResponseEntity.ok(testSuite.getCommonErrors());

    }
}

package com.yakubt.testCoreBackend.presentation;

import java.util.Collections;
import java.util.List;
import java.util.Map;

import com.yakubt.testCoreBackend.models.TestProject;
import com.yakubt.testCoreBackend.models.TestSuite;
import jakarta.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import com.yakubt.testCoreBackend.business.CreateProjectService;
import
com.yakubt.testCoreBackend.persistence.repoInterfaces.UserRepository;
import com.yakubt.testCoreBackend.persistence.TestProjectDao;
import com.yakubt.testCoreBackend.models.DTO.TestProjectDTO;

@RestController()
@RequestMapping("user")
public class UserProjectsController {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private TestProjectDao testProjectDao;

    @Autowired
    private CreateProjectService createProjectService;

    @GetMapping("projects")
    @Transactional
    public                              ResponseEntity<List<TestProjectDTO>>
getProjectsByUser(Authentication authentication) {
        List<TestProjectDTO>                    dtoList                    =
userRepository.findByUserName(authentication.getName()).
                get()
                .getProjects().
                stream()
                .map(project -> new TestProjectDTO(project.getId(),
                        project.getName()
                ))
                .toList();

        return ResponseEntity.ok(dtoList);

    }
```

```
    @PostMapping("project")
    public ResponseEntity<String> createNewProject(@RequestParam String
projectName,
                                                   Authentication
authentication) {
        createProjectService.createNewProject(projectName,
authentication.getName());
        return ResponseEntity.ok("OK");
    }

    @GetMapping("project/{projectID}/common_errors")
    @Transactional
    public      ResponseEntity<List<Map.Entry<String,      List<String>>>>
getCommonErrors(
            @PathVariable Long projectID,
            Authentication authentication) {
        TestProject                      testProject                      =
testProjectDao.read(projectID).orElse(null);
        if (testProject == null || !testProject.getUser()
                .getUserName().equals(authentication.getName())) {
            return
ResponseEntity.badRequest().body(Collections.emptyList());
        }
        return ResponseEntity.ok(testProject.getCommonErrors());

    }
}

package com.yakubt.testCoreBackend.models;

import jakarta.persistence.MappedSuperclass;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

@MappedSuperclass
public abstract class TestComposite implements ITestComponent {


    public abstract List<? extends ITestComponent> getChildren();

    @Override
    public List<Map.Entry<String, List<String>>> getCommonErrors() {
        List<List<Map.Entry<String, List<String>>>> listOfCommonErrors =

getChildren().stream().map(ITestComponent::getCommonErrors).toList();
        List<String> errorsList = listOfCommonErrors.stream().flatMap(list -
> list.stream()
                .map(Map.Entry::getKey)).distinct().toList();
        List<Map.Entry<String, List<String>>> result = new ArrayList<>();
        for (String error : errorsList) {
            result.add(Map.entry(error, listOfCommonErrors
                    .stream()
                    .flatMap(list     ->     list.stream().filter(el     ->
el.getKey().equals(error))
                            .flatMap(el                                    ->
el.getValue().stream())).distinct().toList()));
        }
        return result;
    }
}
```

```
package com.yakubt.TestCoreLib.ui.driver;

import org.openqa.selenium.WebDriver;

import com.yakubt.TestCoreLib.ConfigOptions;
import com.yakubt.TestCoreLib.Config;
import
com.yakubt.TestCoreLib.ui.driver.driverobtainable.AndroidImplementation;
import com.yakubt.TestCoreLib.ui.driver.driverobtainable.DriverObtainable;
import com.yakubt.TestCoreLib.ui.driver.driverobtainable.WebImplementation;
import com.yakubt.TestCoreLib.ui.enums.Platform;

public class DriverHelper implements DriverObtainable {

    private DriverObtainable driverObtainable;

    private static DriverHelper instance;

    private static String previousPlatform;


    private DriverHelper() {
        if   (Config.getInstance().getValue(ConfigOptions.platformName)   ==
null || Config.getInstance().getValue(ConfigOptions.platformName).
                equals(Platform.WEB.getValue())) {
            driverObtainable = new WebImplementation();
        }                                   else                                if
(Config.getInstance().getValue(ConfigOptions.platformName).equals(Platform.
ANDROID.getValue())) {
            driverObtainable = new AndroidImplementation();
        }
    }


    public static DriverHelper getInstance() {
        if            (instance            ==            null            ||
!previousPlatform.equals(Config.getInstance().getValue(ConfigOptions.platfo
rmName))) {
            instance = new DriverHelper();
            previousPlatform                                                    =
Config.getInstance().getValue(ConfigOptions.platformName);
            if (previousPlatform == null) {
                previousPlatform = Platform.WEB.getValue();
            }
        }
        return instance;
    }

    public WebDriver getDriver() {
        return driverObtainable.getDriver();
    }
}

package com.yakubt.TestCoreLib.ui.driver;

import com.yakubt.TestCoreLib.ui.driver.capabilities.CapabilitiesFactory;
import io.github.bonigarcia.wdm.WebDriverManager;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.edge.EdgeOptions;
```

```
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.safari.SafariDriver;
import org.openqa.selenium.safari.SafariOptions;

public class BrowserFactory {

    public static WebDriver getDriver(String browserName) {
        WebDriver driver = null;

        switch (browserName.toLowerCase()) {
            case "firefox":
                WebDriverManager.firefoxdriver().setup();
                driver      =      new      FirefoxDriver((FirefoxOptions)
CapabilitiesFactory.getOptions("firefox"));
                break;

            case "edge":
                WebDriverManager.edgedriver().setup();
                driver        =        new        EdgeDriver((EdgeOptions)
CapabilitiesFactory.getOptions("edge"));
                break;
            case "safari":
                WebDriverManager.safaridriver().setup();
                driver       =       new       SafariDriver((SafariOptions)
CapabilitiesFactory.getOptions("safari"));
                break;

            default:
                WebDriverManager.chromedriver().setup();
                driver       =       new       ChromeDriver((ChromeOptions)
CapabilitiesFactory.getOptions("chrome"));
                break;
        }

        return driver;
    }
}

package com.yakubt.TestCoreLib.ui.driver.driverobtainable;

import java.net.MalformedURLException;
import java.net.URL;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.RemoteWebDriver;

import com.yakubt.TestCoreLib.Config;
import com.yakubt.TestCoreLib.ui.driver.BrowserFactory;
import com.yakubt.TestCoreLib.ui.driver.capabilities.CapabilitiesFactory;
import com.yakubt.TestCoreLib.ConfigOptions;

public  class  WebImplementation  extends  DriverObtainableCommon  implements
DriverObtainable {

    private static ThreadLocal<WebDriver> drivers = new ThreadLocal<>();

    @Override
    protected WebDriver createNewDriver() {
        WebDriver webDriver = null;
        if   (Config.getInstance().getValue(ConfigOptions.seleniumGrid)   ==
null                                                                    ||
```

```
Config.getInstance().getValue(ConfigOptions.seleniumGrid).equals("false"))
{
            webDriver                                                    =
BrowserFactory.getDriver(Config.getInstance().getValue(ConfigOptions.browse
r));
        } else {
            String host = "localhost";
            String port = "4444";
            if (Config.getInstance().getValue(ConfigOptions.host) != null) {
                host = Config.getInstance().getValue(ConfigOptions.host);
            }
            if (System.getProperty(ConfigOptions.port) != null) {
                port = Config.getInstance().getValue(ConfigOptions.port);
            }
            try {
                webDriver         =         new         RemoteWebDriver(new
URL(String.format("http://%s:%s/wd/hub",           host,           port)),
CapabilitiesFactory.getOptions(Config.getInstance().getValue(ConfigOptions.
browser)));
            } catch (MalformedURLException e) {
                throw new RuntimeException(e);
            }

        }
        webDriver.manage().window().maximize();
        return webDriver;
    }
}

package com.yakubt.TestCoreLib.ui;

import com.google.common.reflect.ClassPath;
import com.yakubt.TestCoreLib.Config;
import com.yakubt.TestCoreLib.ConfigOptions;
import com.yakubt.TestCoreLib.ui.annotations.PlatformType;
import com.yakubt.TestCoreLib.ui.enums.Platform;

import java.io.IOException;
import java.lang.reflect.InvocationTargetException;

public class PageInitializer {

    public static <T extends AbstractPage> T initPage(Class<T> _class) {
        try {
            for (ClassPath.ClassInfo classInfo :
                    ClassPath.from(ClassLoader.getSystemClassLoader()).
                            getAllClasses()) {
                try {
                    Class<?> clazz = classInfo.load();
                    if          (_class.isAssignableFrom(clazz)          &&
!clazz.equals(_class)) {
                        if (clazz.isAnnotationPresent(PlatformType.class)) {
                            PlatformType           platformType           =
clazz.getAnnotation(PlatformType.class);
                            String configPlatform = Config.getInstance().
                                    getValue(ConfigOptions.platformName);
                            if (configPlatform == null) {
                                configPlatform = Platform.WEB.getValue();
                            }

                            if
(platformType.value().getValue().equals(configPlatform)) {
                                return (T) clazz.getDeclaredConstructor().
```

```
                                        newInstance();
                                }
                            }
                        }
                    } catch (java.lang.NoClassDefFoundError e) {

                    }
                }
                throw new RuntimeException("No classes");
            } catch (IOException e) {
                throw new RuntimeException(e);
            } catch (InvocationTargetException e) {
                throw new RuntimeException(e);
            } catch (InstantiationException e) {
                throw new RuntimeException(e);
            } catch (IllegalAccessException e) {
                throw new RuntimeException(e);
            } catch (NoSuchMethodException e) {
                throw new RuntimeException(e);
            }
        }
    }
```

```
package com.yakubt.TestCoreCLI.presentation;

import com.yakubt.TestCoreCLI.Config;
import com.yakubt.TestCoreCLI.ResponseWrapper;
import com.yakubt.TestCoreCLI.bussiness.GetCommonErrorsService;

public class CommonErrorsPresentationService implements IOperationStrategy {

    private String transformLevel(String baseLevel) {
        if (baseLevel.equals("test_run")) {
            return "projects/testruns";
        }
        if (baseLevel.equals("test_suite")) {
            return "testSuite";
        }


        return baseLevel;
    }

    @Override
    public int perform(String[] args) {
        String login = Config.loadConfig().getProperty("login", "");
        String password = Config.loadConfig().getProperty("password", "");
        int elementID;
        String level;
        if (args.length > 3) {
            login = args[1];
            password = args[2];
            level = args[3];
            elementID = Integer.parseInt(args[4]);
        } else {
            level = args[1];
            elementID = Integer.parseInt(args[2]);
        }
        ResponseWrapper responseWrapper =
                GetCommonErrorsService.
                        getCommonErrors(login,     password,     elementID,
transformLevel(level));
        System.out.println(responseWrapper.body);
```

```
        if (responseWrapper.code != 200) {
            return 1;
        }
        return 0;
    }
}

package com.yakubt.testcoregui;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.ResourceBundle;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.yakubt.testcoregui.bussiness.TestCoreCLICommunicationService;
import com.yakubt.testcoregui.models.Project;
import com.yakubt.testcoregui.models.TestArtifact;

public abstract class TestArtifactController implements Initializable {
    @FXML
    private ListView<TestArtifact> listView = new ListView<>();

    @FXML
    protected Label usernameLabel;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {

usernameLabel.setText(TestCoreCLICommunicationService.getUser().getMessage(
));
        List<TestArtifact> testArtifactList = fetchArtifacts();
        try {
            listView.getItems().addAll(testArtifactList);

            listView.setOnMouseClicked(event -> {
                if (event.getClickCount() == 2) {
                    TestArtifact                 selected                 =
listView.getSelectionModel().getSelectedItem();
                    if (selected != null) {

ProgramState.getArtifactsIDs().add(selected.getId());
                        openWindow();
                    }
                }
            });
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    //Шаблоний метод - Design pattern
    protected abstract String getCommandMessage();

    private List<TestArtifact> fetchArtifacts() {
        String output = getCommandMessage();

        ObjectMapper mapper = new ObjectMapper();
        List<Map<String, Object>> parsed = null;
        try {
            parsed = mapper.readValue(output, new TypeReference<>() {
            });
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }

        List<TestArtifact> testArtifactList = new ArrayList<>();
        for (Map<String, Object> item : parsed) {
            int id = (Integer) item.get("id");
            String name = (String) item.get("name");
            testArtifactList.add(new Project(id, name));
        }

        return testArtifactList;
    }

    public void openWindow() {
        FXMLLoader fxmlLoader = null;
        if (ProgramState.getArtifactsIDs().isEmpty()) {
            fxmlLoader = new FXMLLoader(TestCoreApplication.class.
                    getResource("projects_page.fxml"));
        }
        if (ProgramState.getArtifactsIDs().size() == 1) {
            fxmlLoader = new FXMLLoader(TestCoreApplication.class.
                    getResource("project_testruns.fxml"));
        }
        if (ProgramState.getArtifactsIDs().size() == 2) {
            fxmlLoader = new FXMLLoader(TestCoreApplication.class.
                    getResource("testrun_testsuites.fxml"));
        }

        if (ProgramState.getArtifactsIDs().size() == 3) {
            fxmlLoader = new FXMLLoader(TestCoreApplication.class.
                    getResource("test_suite_test_cases.fxml"));
        }

        Stage stage = new Stage();
        Scene scene = null;
        try {
            scene = (Scene) fxmlLoader.load();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

scene.getStylesheets().add(getClass().getResource("style.css").toExternalFo
rm());
        stage.setResizable(false);
        Map<Integer, String> map = Map.of(0, "Test Projects", 1, "Test
Project info",
                2, "TestRun info", 3, "TestSuite info");
        stage.setTitle(map.get(ProgramState.getArtifactsIDs().size()));
        stage.setScene(scene);
        stage.show();
        ((Stage) usernameLabel.getScene().getWindow()).close();
```

```
    }

    @FXML
    public void onClickAuthButton() {
        FXMLLoader fxmlLoader = new
FXMLLoader(TestCoreApplication.class.getResource("auth.fxml"));
        Stage stage = new Stage();
        Scene scene = null;
        try {
            scene = (Scene) fxmlLoader.load();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

scene.getStylesheets().add(getClass().getResource("style.css").toExternalFo
rm());
        stage.setResizable(false);
        stage.setTitle("Authorization");
        stage.setScene(scene);
        stage.show();
        ((Stage) usernameLabel.getScene().getWindow()).close();
    }

    public void getCommonErrors(String json, Accordion errorAccordion) {
        ObjectMapper mapper = new ObjectMapper();
        TypeReference<List<Map<String, List<String>>>> typeRef = new
TypeReference<>() {
        };
        List<Map<String, List<String>>> errorList = null;
        try {
            errorList = mapper.readValue(json, typeRef);
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
        for (Map<String, List<String>> errorMap : errorList) {
            for (Map.Entry<String, List<String>> entry :
errorMap.entrySet()) {
                String errorMessage = entry.getKey();
                List<String> testNames = entry.getValue();
                ScrollPane scrollPane = new ScrollPane();
                scrollPane.setFitToWidth(true);
                ListView<String> testList = new ListView<>();
                testList.getItems().addAll(testNames);

                TitledPane pane = new TitledPane();
                StringBuilder textForPane = new StringBuilder();
                for (int i = 0; i < Math.min(errorMessage.length(), 300); i
+= 50) {
                    textForPane.append(errorMessage, i,
Math.min(errorMessage.length(), i + 50)).append("\n");
                }
                pane.setText(textForPane.toString());
                scrollPane.setMinHeight(testNames.size() * 45);
                testList.setMinHeight(testNames.size() * 45);
                scrollPane.setContent(testList);
                pane.setContent(scrollPane);

                pane.setExpanded(false);

                errorAccordion.getPanes().add(pane);
            }
        }
```

```
    }

    public void goBackButton() {
        ProgramState.getArtifactsIDs().pop();
        openWindow();
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.Scene?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.effect.DropShadow?>

<?import java.lang.String?>
<Scene                                    xmlns="http://javafx.com/javafx/11"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.yakubt.testcoregui.AuthController">
   <root>
      <Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity"   prefHeight="600.0"   prefWidth="800.0"   style="-fx-
background-color: CFE9E6;">
         <children>
            <Label layoutX="163.0" layoutY="30.0" text="Welcome to TestCore"
textFill="#000000" wrapText="true">
               <font>
                  <Font name="System Bold" size="48.0" />
               </font>
            </Label>
            <Button  fx:id="loginButton"  layoutX="253.0"  layoutY="153.0"
mnemonicParsing="false" prefHeight="147.0" prefWidth="294.0"
                  style="-fx-background-color: #6ED6CA; -fx-border-color:
#337870; -fx-border-width: 5; -fx-cursor: hand;"
                  text="Login" onAction="#onLoginButtonClick">
               <font>
                  <Font size="48.0" />
               </font>
               <effect>
                  <DropShadow radius="5" color="rgba(0,0,0,0.2)"/>
               </effect>
               <styleClass>
                  <String fx:value="hover-button" />
               </styleClass>
            </Button>
            <Button  fx:id="registerButton"  layoutX="253.0"  layoutY="337.0"
mnemonicParsing="false" prefHeight="147.0" prefWidth="294.0"
                  style="-fx-background-color: #6ED6CA; -fx-border-color:
#337870; -fx-border-width: 5; -fx-cursor: hand;"
                  text="Register" onAction="#onRegisterButtonClick">
               <font>
                  <Font size="48.0" />
               </font>
               <effect>
                  <DropShadow radius="5" color="rgba(0,0,0,0.2)"/>
               </effect>
               <styleClass>
                  <String fx:value="hover-button" />
               </styleClass>
            </Button>
         </children>
```

```
      </Pane>
   </root>
</Scene>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.String?>
<?import javafx.scene.Scene?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.shape.Rectangle?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<Scene                                    xmlns="http://javafx.com/javafx/11"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.yakubt.testcoregui.LoginController">
   <root>
      <Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity"   prefHeight="600.0"   prefWidth="800.0"   style="-fx-
background-color: #8D91BF;">
         <children>
            <Label layoutX="237.0" layoutY="22.0" text="Log in to TestCore"
textAlignment="CENTER" textFill="#cc1f21">
               <font>
                  <Font name="System Bold" size="40.0" />
               </font>
            </Label>
            <Rectangle   arcHeight="30.0"   arcWidth="30.0"   fill="#d9d9d9"
height="451.0"     layoutX="190.0"     layoutY="89.0"     stroke="#d9d9d9"
strokeType="INSIDE" width="439.0" />
            <TextField   fx:id="userName"   layoutX="248.0"   layoutY="170.0"
prefHeight="59.0" prefWidth="323.0">
               <font>
                  <Font size="18.0" />
               </font>
            </TextField>
            <Text   layoutX="291.0"   layoutY="156.0"   strokeType="OUTSIDE"
strokeWidth="0.0"          text="Username"          textAlignment="CENTER"
wrappingWidth="236.27655029296875">
               <font>
                  <Font size="36.0" />
               </font>
            </Text>
            <Text   layoutX="290.0"   layoutY="298.0"   strokeType="OUTSIDE"
strokeWidth="0.0"          text="Password"          textAlignment="CENTER"
wrappingWidth="236.27655029296875">
               <font>
                  <Font size="36.0" />
               </font>
            </Text>
            <Button   fx:id="loginButton"   layoutX="306.0"   layoutY="411.0"
mnemonicParsing="false"   prefHeight="48.0"   prefWidth="205.0"   style="-fx-
background-color: #A4BF7D;"
                    text="Log in" onAction="#submit">
               <font>
                  <Font size="26.0" />
               </font>
               <styleClass>
                  <String fx:value="hover-button" />
```

```
               </styleClass>
            </Button>
            <PasswordField fx:id="password" layoutX="247.0" layoutY="315.0"
prefHeight="58.0" prefWidth="323.0">
               <font>
                  <Font size="18.0" />
               </font>
            </PasswordField>
            <Button   fx:id="goBackButton"   layoutX="16.0"   layoutY="28.0"
mnemonicParsing="false"  prefHeight="46.0"  prefWidth="104.0"  style="-fx-
background-color: #A4BF7D;" text="Go back"
                    textFill="#262424" onAction="#onGoBackButtonClick">
               <font>
                  <Font size="17.0" />
               </font>
               <styleClass>
                  <String fx:value="hover-button" />
               </styleClass>
            </Button>
         </children>
      </Pane>
   </root>
</Scene>

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.String?>
<?import javafx.scene.Scene?>
<?import javafx.scene.control.Accordion?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>

<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.layout.VBox?>
<Scene                                    xmlns="http://javafx.com/javafx/11"
xmlns:fx="http://javafx.com/fxml/1"
       fx:controller="com.yakubt.testcoregui.ProjectTestRunsController">
    <root>
        <Pane   maxHeight="-Infinity"   maxWidth="-Infinity"   minHeight="-
Infinity" minWidth="-Infinity" prefHeight="600.0"
              prefWidth="800.0" style="-fx-background-color: #8D91BF;">
            <children>
                <Label    layoutX="258.0"    layoutY="33.0"    text="User:"
textFill="#dc4b4b">
                    <font>
                        <Font size="40.0"/>
                    </font>
                </Label>
                <Label fx:id="usernameLabel" layoutX="347.0" layoutY="33.0"
text="User:" textFill="#dc4b4b">
                    <font>
                        <Font size="40.0"/>
                    </font>
                </Label>
                <Label layoutX="134.0" layoutY="109.0" text="Test runs">
                    <font>
                        <Font size="35.0"/>
                    </font>
                </Label>
```

```
                <Button  fx:id="authButton"  layoutX="612.0"  layoutY="22.0"
mnemonicParsing="false"
                        onAction="#onClickAuthButton"     prefHeight="46.0"
prefWidth="174.0"
                        style="-fx-background-color: #A4BF7D;" text="Change
user/Logout">
                    <font>
                        <Font size="15.0"/>
                    </font>
                    <styleClass>
                        <String fx:value="hover-button"/>
                    </styleClass>
                </Button>
                <ListView  fx:id="listView"  layoutX="53.0"  layoutY="170.0"
prefHeight="358.0" prefWidth="320.0"/>
                <Label layoutX="296.0" layoutY="349.0">
                    <font>
                        <Font size="47.0"/>
                    </font>
                </Label>
                <Button  fx:id="goBackButton"  layoutX="8.0"  layoutY="22.0"
mnemonicParsing="false"
                        onAction="#goBackButton"          prefHeight="46.0"
prefWidth="174.0"
                        style="-fx-background-color:               #A4BF7D;"
styleClass="hover-button"
                        text="Go back">
                    <font>
                        <Font size="15.0"/>
                    </font>
                </Button>
                <Label layoutX="478.0" layoutY="109.0" text="Common errors">
                    <font>
                        <Font size="35.0"/>
                    </font>
                </Label>
                <ScrollPane       fitToWidth="true"       fitToHeight="true"
layoutX="433.0" layoutY="160.0" prefHeight="376.0"
                            prefWidth="358.0">
                    <VBox>
                        <Accordion                    fx:id="errorAccordion"
VBox.vgrow="ALWAYS"/>
                    </VBox>
                </ScrollPane>
            </children>
        </Pane>
    </root>
</Scene>

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.Scene?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>

<?import java.lang.String?>
<Scene                                xmlns="http://javafx.com/javafx/11"
xmlns:fx="http://javafx.com/fxml/1"
       fx:controller="com.yakubt.testcoregui.UserProjectsController">
   <root>
```

```
      <Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity"   prefHeight="600.0"   prefWidth="800.0"   style="-fx-
background-color: #8D91BF;">
         <children>
            <Label     layoutX="199.0"     layoutY="33.0"     text="Welcome"
textFill="#dc4b4b">
               <font>
                  <Font size="40.0" />
               </font>
            </Label>
            <Label   fx:id="usernameLabel"   layoutX="407.0"   layoutY="33.0"
text="Welcome" textFill="#dc4b4b">
               <font>
                  <Font size="40.0" />
               </font>
            </Label>
            <Label layoutX="311.0" layoutY="92.0" text="Projects">
               <font>
                  <Font size="47.0" />
               </font>
            </Label>
            <Button   fx:id="authButton"   layoutX="612.0"   layoutY="22.0"
mnemonicParsing="false"   prefHeight="46.0"   prefWidth="174.0"   style="-fx-
background-color: #A4BF7D;"
                    text="Change user/Logout" onAction="#onClickAuthButton">
               <font>
                  <Font size="15.0" />
               </font>
               <styleClass>
                  <String fx:value="hover-button" />
               </styleClass>
            </Button>
            <ListView   fx:id="listView"   layoutX="53.0"   layoutY="170.0"
prefHeight="358.0" prefWidth="680.0" />
            <Label layoutX="296.0" layoutY="349.0">
               <font>
                  <Font size="47.0" />
               </font>
            </Label>
         </children>
      </Pane>
   </root>
</Scene>

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.String?>
<?import javafx.scene.Scene?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.shape.Rectangle?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<Scene                                    xmlns="http://javafx.com/javafx/11"
xmlns:fx="http://javafx.com/fxml/1"
       fx:controller="com.yakubt.testcoregui.RegisterController">
    <root>
        <Pane   maxHeight="-Infinity"   maxWidth="-Infinity"   minHeight="-
Infinity" minWidth="-Infinity" prefHeight="600.0"
```

```
            prefWidth="800.0" style="-fx-background-color: #8D91BF;">
          <children>
              <Label  layoutX="237.0"  layoutY="22.0"  text="Register  to
TestCore" textAlignment="CENTER"
                     textFill="#994704">
                  <font>
                      <Font name="System Bold" size="40.0"/>
                  </font>
              </Label>
              <Rectangle  arcHeight="30.0"  arcWidth="30.0"  fill="#d9d9d9"
height="451.0" layoutX="190.0" layoutY="89.0"
                         stroke="#d9d9d9"             strokeType="INSIDE"
width="439.0"/>
              <TextField fx:id="userName" layoutX="248.0" layoutY="156.0"
prefHeight="59.0" prefWidth="323.0">
                  <font>
                      <Font size="18.0"/>
                  </font>
              </TextField>
              <Text  layoutX="291.0"  layoutY="144.0"  strokeType="OUTSIDE"
strokeWidth="0.0" text="Username"
                    textAlignment="CENTER"
wrappingWidth="236.27655029296875">
                  <font>
                      <Font size="24.0"/>
                  </font>
              </Text>
              <Text  layoutX="291.0"  layoutY="252.0"  strokeType="OUTSIDE"
strokeWidth="0.0" text="Password"
                    textAlignment="CENTER"
wrappingWidth="236.27655029296875">
                  <font>
                      <Font size="24.0"/>
                  </font>
              </Text>
              <Button        fx:id="registerButton"        layoutX="306.0"
layoutY="440.0" mnemonicParsing="false" prefHeight="48.0"
                       prefWidth="205.0"      style="-fx-background-color:
#A4BF7D;" text="Register" onAction="#submit">
                  <font>
                      <Font size="26.0"/>
                  </font>
                  <styleClass>
                      <String fx:value="hover-button"/>
                  </styleClass>
              </Button>
              <PasswordField       fx:id="password"       layoutX="248.0"
layoutY="258.0" prefHeight="58.0" prefWidth="323.0">
                  <font>
                      <Font size="18.0"/>
                  </font>
              </PasswordField>
              <Button  fx:id="goBackButton"  layoutX="16.0"  layoutY="28.0"
mnemonicParsing="false" prefHeight="46.0"
                       prefWidth="104.0"      style="-fx-background-color:
#A4BF7D;" text="Go back" textFill="#262424"
                       onAction="#onGoBackButtonClick">
                  <font>
                      <Font size="17.0"/>
                  </font>
                  <styleClass>
                      <String fx:value="hover-button"/>
                  </styleClass>
```

```
                </Button>
                <PasswordField    fx:id="confirmPassword"    layoutX="247.0"
layoutY="356.0" prefHeight="58.0" prefWidth="323.0">
                    <font>
                        <Font size="18.0"/>
                    </font>
                </PasswordField>
                <Text  layoutX="290.0"  layoutY="349.0"  strokeType="OUTSIDE"
strokeWidth="0.0" text="Confirm password"
                      textAlignment="CENTER"
wrappingWidth="236.27655029296875">
                    <font>
                        <Font size="24.0"/>
                    </font>
                </Text>
            </children>
        </Pane>
    </root>
</Scene>

.hover-button {
    -fx-effect: dropshadow(three-pass-box, rgba(0,0,0,0.2), 5, 0, 0, 0);
    -fx-transition: all 0.3s ease;
}

.hover-button:hover {
    -fx-scale-x: 1.25;
    -fx-scale-y: 1.25;
    -fx-effect: dropshadow(three-pass-box, rgba(0,0,0,0.3), 10, 0, 0, 0);
}

.list-cell {
    -fx-font-size: 20px;
}

.combo-box .list-cell {
    -fx-font-size: 10px;
}
.combo-box-popup .list-view .list-cell {
    -fx-font-size: 10px;
}

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.String?>
<?import javafx.scene.Scene?>
<?import javafx.scene.control.Accordion?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>

<Scene                                    xmlns="http://javafx.com/javafx/11"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.yakubt.testcoregui.TestSuiteController">
    <root>
        <Pane   maxHeight="-Infinity"   maxWidth="-Infinity"   minHeight="-
Infinity" minWidth="-Infinity" prefHeight="600.0" prefWidth="800.0" style="-
fx-background-color: #8D91BF;">
```

```
            <children>
                <Label    layoutX="258.0"    layoutY="33.0"    text="User:"
textFill="#dc4b4b">
                    <font>
                        <Font size="40.0" />
                    </font>
                </Label>
                <Label fx:id="usernameLabel" layoutX="347.0" layoutY="33.0"
text="User:" textFill="#dc4b4b">
                    <font>
                        <Font size="40.0" />
                    </font>
                </Label>
                <Label layoutX="134.0" layoutY="109.0" text="Test results">
                    <font>
                        <Font size="35.0" />
                    </font>
                </Label>
                <Button  fx:id="authButton"  layoutX="612.0"  layoutY="22.0"
mnemonicParsing="false"   onAction="#onClickAuthButton"   prefHeight="46.0"
prefWidth="174.0"   style="-fx-background-color:   #A4BF7D;"   text="Change
user/Logout">
                    <font>
                        <Font size="15.0" />
                    </font>
                    <styleClass>
                        <String fx:value="hover-button" />
                    </styleClass>
                </Button>
                <TableView       fx:id="testCaseTable"       layoutX="13.0"
layoutY="160.0" prefHeight="377.0" prefWidth="403.0">
                    <columns>
                        <TableColumn                         fx:id="idColumn"
prefWidth="36.79998779296875" text="ID" />
                        <TableColumn                       fx:id="nameColumn"
prefWidth="132.79995727539062" text="Name" />
                        <TableColumn                     fx:id="statusColumn"
prefWidth="101.5999755859375" text="Status" />
                        <TableColumn                   fx:id="recheckColumn"
prefWidth="144.79998779296875" text="Recheck Status" />
                    </columns>
                </TableView>

                <Label layoutX="296.0" layoutY="349.0">
                    <font>
                        <Font size="47.0" />
                    </font>
                </Label>
                <Button  fx:id="goBackButton"  layoutX="8.0"  layoutY="22.0"
mnemonicParsing="false"     onAction="#goBackButton"     prefHeight="46.0"
prefWidth="174.0" style="-fx-background-color: #A4BF7D;" styleClass="hover-
button" text="Go back">
                    <font>
                        <Font size="15.0" />
                    </font>
                </Button>
                <Label layoutX="478.0" layoutY="109.0" text="Common errors">
                    <font>
                        <Font size="35.0" />
                    </font>
                </Label>
                <ScrollPane       fitToHeight="true"       fitToWidth="true"
layoutX="433.0" layoutY="160.0" prefHeight="376.0" prefWidth="358.0">
```

```
                    <VBox>
                        <Accordion                    fx:id="errorAccordion"
VBox.vgrow="ALWAYS" />
                    </VBox>
                </ScrollPane>
            </children>
        </Pane>
    </root>
</Scene>

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.String?>
<?import javafx.scene.Scene?>
<?import javafx.scene.control.Accordion?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ListView?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>

<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.layout.VBox?>
<Scene                                  xmlns="http://javafx.com/javafx/11"
xmlns:fx="http://javafx.com/fxml/1"
       fx:controller="com.yakubt.testcoregui.TestRunTestSuiteController">
    <root>
        <Pane   maxHeight="-Infinity"   maxWidth="-Infinity"   minHeight="-
Infinity" minWidth="-Infinity" prefHeight="600.0"
              prefWidth="800.0" style="-fx-background-color: #8D91BF;">
            <children>
                <Label    layoutX="258.0"    layoutY="33.0"    text="User:"
textFill="#dc4b4b">
                    <font>
                        <Font size="40.0"/>
                    </font>
                </Label>
                <Label fx:id="usernameLabel" layoutX="347.0" layoutY="33.0"
text="User:" textFill="#dc4b4b">
                    <font>
                        <Font size="40.0"/>
                    </font>
                </Label>
                <Label layoutX="134.0" layoutY="109.0" text="Test suites">
                    <font>
                        <Font size="35.0"/>
                    </font>
                </Label>
                <Button  fx:id="authButton"  layoutX="612.0"  layoutY="22.0"
mnemonicParsing="false"
                        onAction="#onClickAuthButton"     prefHeight="46.0"
prefWidth="174.0"
                        style="-fx-background-color: #A4BF7D;" text="Change
user/Logout">
                    <font>
                        <Font size="15.0"/>
                    </font>
                    <styleClass>
                        <String fx:value="hover-button"/>
                    </styleClass>
                </Button>
                <ListView  fx:id="listView"  layoutX="53.0"  layoutY="170.0"
prefHeight="358.0" prefWidth="320.0"/>
```

```
                <Label layoutX="296.0" layoutY="349.0">
                    <font>
                        <Font size="47.0"/>
                    </font>
                </Label>
                <Button fx:id="goBackButton" layoutX="8.0" layoutY="22.0"
mnemonicParsing="false"
                        onAction="#goBackButton" prefHeight="46.0"
prefWidth="174.0"
                        style="-fx-background-color: #A4BF7D;"
styleClass="hover-button"
                        text="Go back">
                    <font>
                        <Font size="15.0"/>
                    </font>
                </Button>
                <Label layoutX="478.0" layoutY="109.0" text="Common
errors">
                    <font>
                        <Font size="35.0"/>
                    </font>
                </Label>
                <ScrollPane fitToWidth="true" fitToHeight="true"
layoutX="433.0" layoutY="160.0" prefHeight="376.0"
                            prefWidth="358.0">
                    <VBox>
                        <Accordion fx:id="errorAccordion"
VBox.vgrow="ALWAYS"/>
                    </VBox>
                </ScrollPane>
            </children>
        </Pane>
    </root>
</Scene
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ФРЕЙМВОРК ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Виконав: Якубишин Анатолій Сергійович

Керівник: доц. кафедри ПЗКС, к.т.н., доц. Заболотня Тетяна Миколаївна

Київ – 2025

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити фреймворк для написання якісних автоматизованих тестів для перевірки коректності роботи ПЗ та їх подальшого оновлення

Завдання:

1. Проаналізувати ринок фреймворків для автоматизованого тестування програмного забезпечення
2. Порівняти конкурентів, виявити їхні переваги та недоліки
3. Сформувати вимоги до ПЗ
4. Обґрунтувати вибір технологій для розроблення фреймворку
5. Реалізувати фреймворк для автоматизованого тестування програмного забезпечення
6. Протестувати розроблене ПЗ
7. Визначити можливості подальшого вдосконалення

https://github.com/AnatoliiYak/testCoreInstallation

АКТУАЛЬНІСТЬ

1. Зростаюча складність програмних продуктів
2. Часті випуски оновлень
3. Гнучкі фреймворки легко інтегруються в сучасні процеси розробки (CI/CD)
4. Зростаючі вимоги щодо якості ПЗ
5. Попит на ринку

https://github.com/AnatoliiYak/testCoreInstallation

ІСНУЮЧІ РІШЕННЯ

Найбільш вагомі спільні недоліки конкурентів

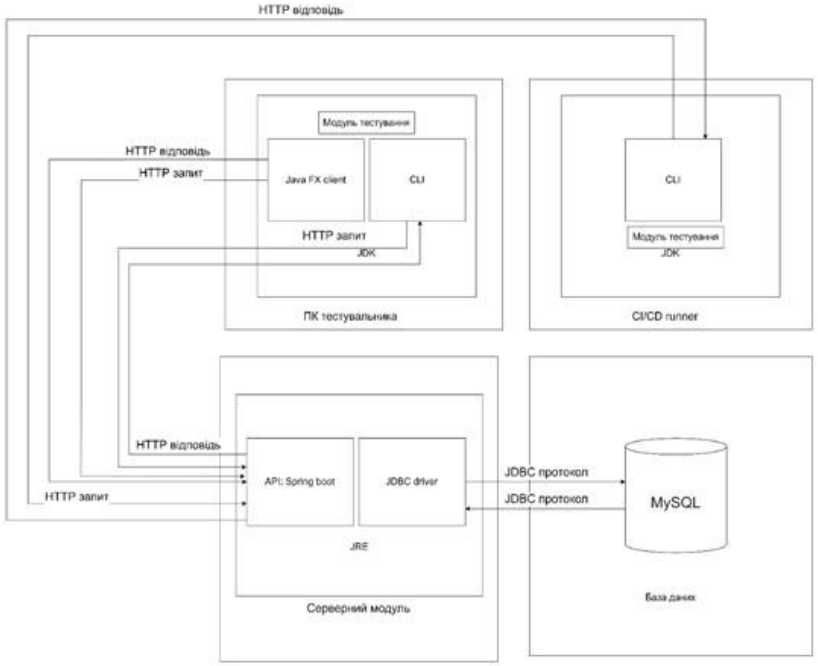
1. Відсутність підтримки мобільного тестування:
    - Postman, Cypress, Playwright, Selenide не підтримують повноцінне мобільне тестування
2. Одна сфера тестування
3. Відсутність єдиного централізованого середовища
    - вимагають зовнішніх засобів для звітності, обробки результатів або мають складну інтеграцію

https://github.com/AnatoliiYak/testCoreInstallation

ОБРАНІ ЗАСОБИ РЕАЛІЗАЦІЇ

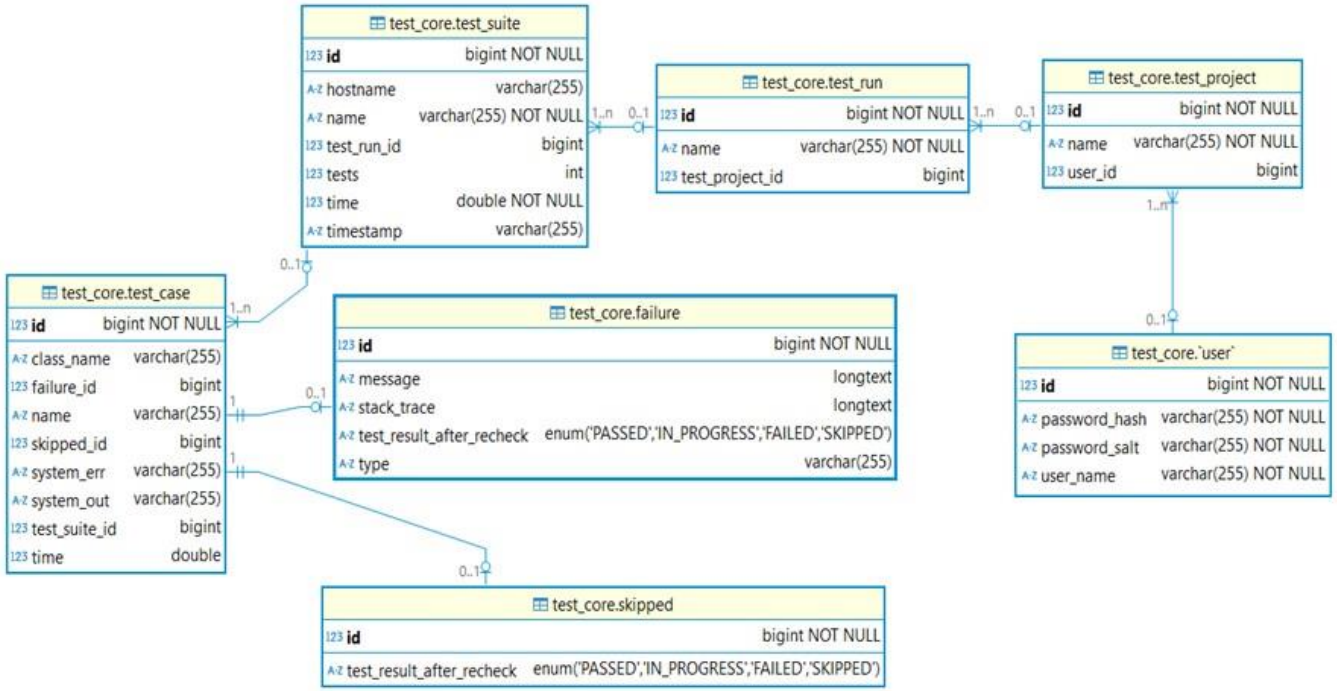
https://github.com/AnatoliiYak/testCoreInstallation

АРХІТЕКТУРА РОЗРОБЛЕНОГО ФРЕЙМВОРКУ



https://github.com/AnatoliiYak/testCoreInstallation

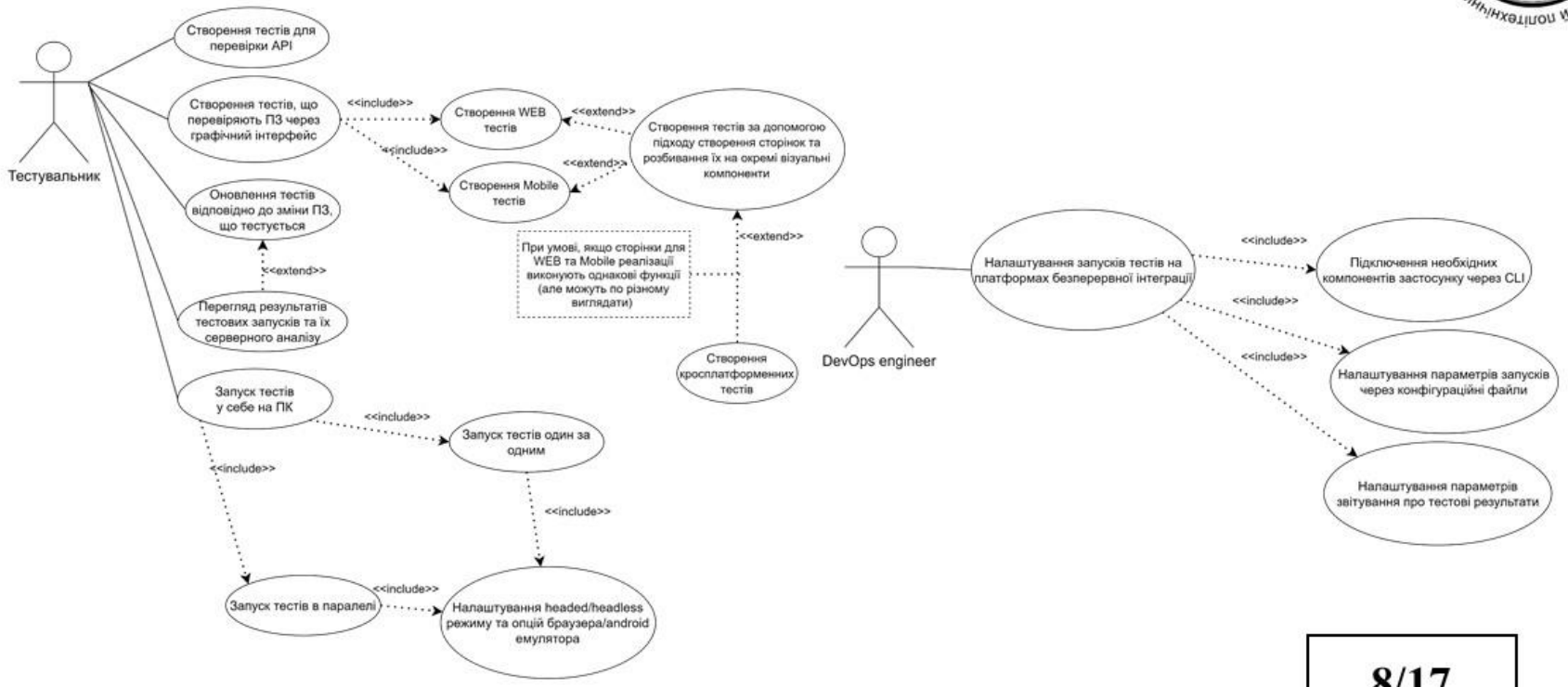
СХЕМА БАЗИ ДАНИХ



https://github.com/AnatoliiYak/testCoreInstallation

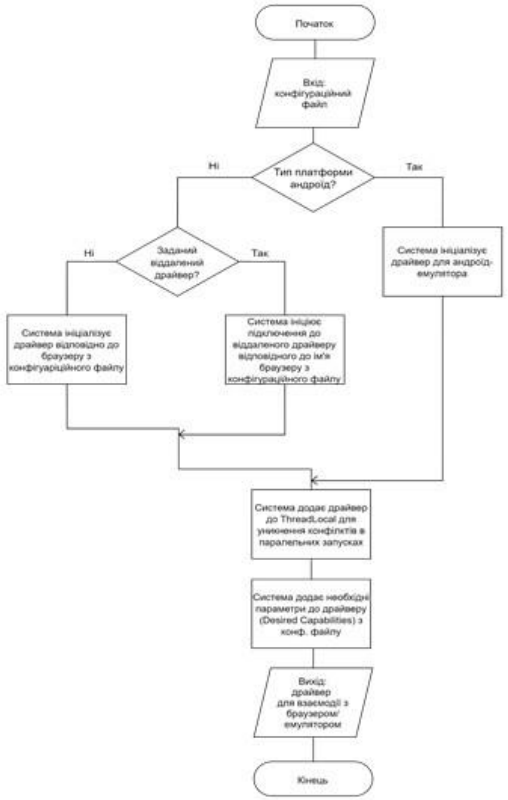
ДІАГРАМА ПРЕЦЕДЕНТІВ

Фреймворк для автоматизованого тестування
програмного забезпечення



https://github.com/AnatoliiYak/testCoreInstallation

АЛГОРИТМ ІНІЦІАЛІЗАЦІЇ ДРАЙВЕРУ ДЛЯ ВЗАЄМОДІЇ З БРАУЗЕРОМ/ЕМУЛЯТОРОМ



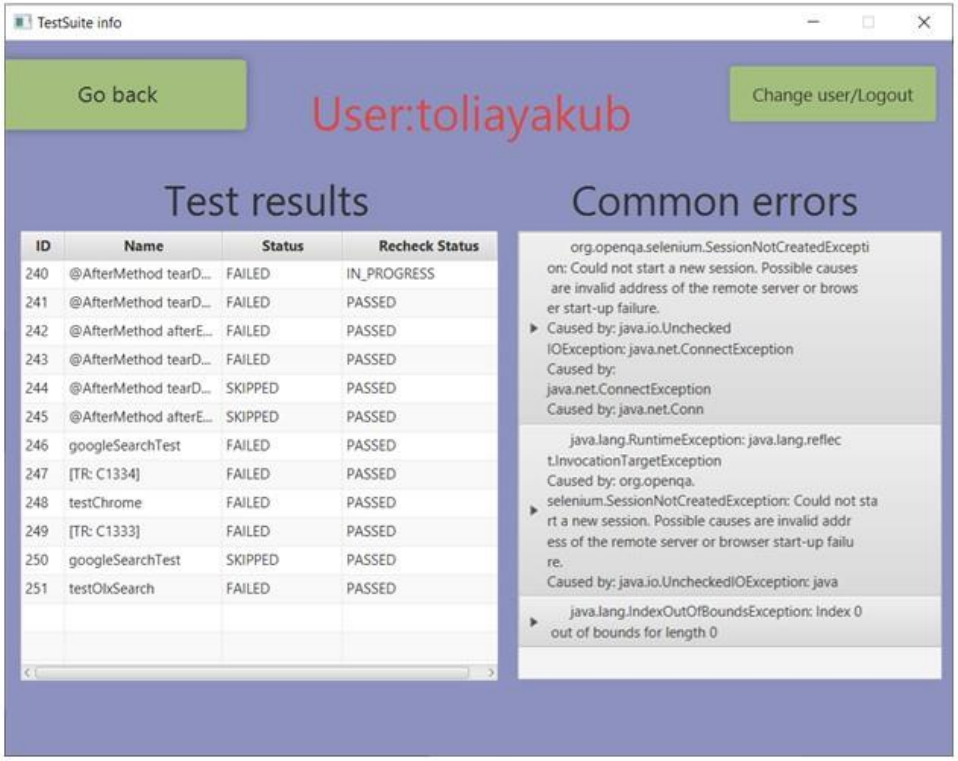
https://github.com/AnatoliiYak/testCoreInstallation

ІНСТРУМЕНТ КОМАДНОГО РЯДКА РОЗРОБЛЕНОГО ФРЕЙМВОРКУ


```
PS C:\Users\Anatoly> testcorecli help
List of commands:
login <username> <password>
register <username> <password>
For further commands <username> <password> are optional
download_java_template <project_name>
send_result - sends results for the current project, run in the project root directory
get_projects
get_test_runs <project_id>
get_test_suites <test_run_id>
get_test_cases <test_run_id>
get_test_case <test_case_id>
get_common_errors <level> <artifact_id> - levels: project, test_run, test_suite
set_status <id> <status>  - status: passed, skipped, failed, in_progress
get_user
```


https://github.com/AnatoliiYak/testCoreInstallation

ГРАФІЧНИЙ ІНТЕРФЕЙС РОЗРОБЛЕНОГО ФРЕЙМВОРКУ



ID	Name	Status	Recheck Status
240	@AfterMethod tearD...	FAILED	IN_PROGRESS
241	@AfterMethod tearD...	FAILED	PASSED
242	@AfterMethod afterE...	FAILED	PASSED
243	@AfterMethod tearD...	FAILED	PASSED
244	@AfterMethod tearD...	SKIPPED	PASSED
245	@AfterMethod afterE...	SKIPPED	PASSED
246	googleSearchTest	FAILED	PASSED
247	[TR: C1334]	FAILED	PASSED
248	testChrome	FAILED	PASSED
249	[TR: C1333]	FAILED	PASSED
250	googleSearchTest	SKIPPED	PASSED
251	testOlxSearch	FAILED	PASSED

https://github.com/AnatoliiYak/testCoreInstallation

ОСНОВНА ЧАСТИНА ФРЕЙМВОРКУ

Maven™

Page Object Model

TestNG

https://github.com/AnatoliiYak/testCoreInstallation

РЕЗУЛЬТАТИ ТЕСТУВАННЯ

```
Results:

Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
```

Автоматизоване тестування
серверної частини фреймворку

```
[INFO] Results:
[INFO]
[INFO] Tests run: 22, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

Автоматизоване тестування
основної частини фреймворку

```
Results:

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  9.176 s
Finished at: 2025-05-14T19:29:54+03:00
------------------------------------------------------------------------
NG] The requested profile "with-http-repo" could not be activated because it does not exist.
```

CLI

Ручне димне наскрізне тестування

https://github.com/AnatoliiYak/testCoreInstallation

ШЛЯХИ ПОДАЛЬШОГО РОЗВИТКУ ФРЕЙМВОРКУ

https://github.com/AnatoliiYak/testCoreInstallation

ВИСНОВКИ

У процесі виконання дипломного проєкту було проаналізовано предметну область, проаналізовано існуючи рішення та обґрунтовано доцільність розробки даного ПЗ.

На етапі проєктування фреймворку було обрано засоби реалізації. Було обґрунтовано доцільність використання мови Java та БД MySQL. Також вибрано платформу для запуску тестів TestNG та Maven як систему збірки. Розглянуто й інші альтернативні засоби реалізації.

ПЗ було створено на основі комплексної архітектури в основі якої лежить архітектура клієнт-сервер.

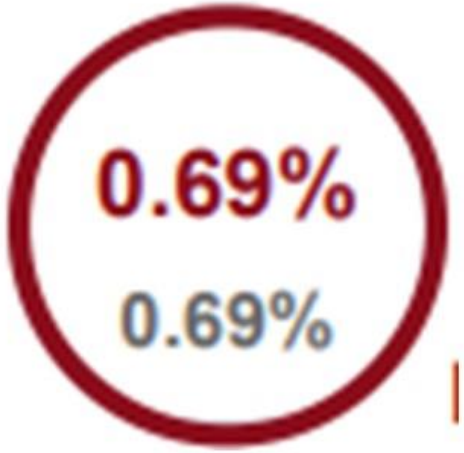
ПЗ було протестовано ручними перевірками та за допомогою автоматизованих тестів.

ПЗ відповідає висунутим до нього загальним вимогам та вимогам до безпеки.

https://github.com/AnatoliiYak/testCoreInstallation

УНІКАЛЬНІСТЬ

Коефіцієнт подібності



ДЯКУЮ ЗА УВАГУ!

17/17

https://github.com/AnatoliiYak/testCoreInstallation

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

ФРЕЙМВОРК ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Програма та методика тестування

ДП.045490-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анатолій ЯКУБИШИН

2024

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Фреймворк для автоматизованого тестування програмного забезпечення, що складається з графічного інтерфейсу у вигляді настільного додатку, інструменту командного рядка, основної частини фремворку та серверної частини.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів вікон графічної частини;
2) функціональна працездатність операцій інструменту командного рдяка;
3) функціональна працездатність REST API серверної частини;
4) забезпечення належного рівня безпеки даних;
5) зручність роботи з фреймворком;
6) відповідність розроблення вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується за допомогою ручного димного наскрізного тестування та автоматизованого юніт-тестування та частково автоматизованого інтеграційного тестування. Тестування виконується методом тестування білого ящика. Перевіряється код на відповідність принципів «чистого програмування» так і безпосередньо програмний продукт на відповідність вимогам.

Використовуються наступні методи:

1) функціональне тестування;
2) інтеграційне тестування;
3) тестування інтерфейсу;

4) димне тестування.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами TestNG, JUnit та Mockito.

Працездатність фреймворку перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) автоматизації тестування модулів коду;
5) тестування зручності використання;
6) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2025 р.

ФРЕЙМВОРК ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Керівництво користувача

ДП.045490-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Анатолій ЯКУБИШИН

2025

ЗМІСТ

1. Опис структури фреймворку

Фремворк для автоматизації тестування програмного забезпечення складається із серверної частини, до якої можна робити запити через REST API, командний рядок та графічний інтерфейс, що є настільним додатком. Також фреймворк складається з основної частини, з якою користувач взаємодіє у шаблоні тестового проєкту, що завантажується через запит до REST API або через виклик відповідної команди інструменту командного рядка. Мовою інтерфейсу командного рядка та графічного інтерфейсу є англійська.

Вміст вікон графічного інтерфейсу командного рядка є динамічним та залежить від даних у БД про користувача. Початкове вікно при запуску графічної частини залежить від того чи користувач був авторизований раніше та під якими обліковими даними. Є вікна, що мають однаковий вигляд для всіх користувачів.

До вікон, що виглядають однаково не залежно від користувача належать:

– «Вікно авторизації»;

– «Вікно реєстрації»;

– «Вікно логіну».

До вікон, що мають різний вигляд в залежності від даних про користувача у БД належать:

– «Вікно проєктів»;

– «Вікно інформації про проєкт»;

– «Вікно інформації про тестовий запуск»;

– «Вікно інформації про тестовий набір».

2. Процедура авторизації користувача

Для виконання авторизації можна скористатися графічним інтерфейсом або інструментом командного рядка.

Для виконання операцій з командним рядком рекомендовано попередньо використовувати команду testcorecli help – це дозволить отримати довідку користувача по всім доступним командам. Для входу в систему виконується команда testcorecli login <ім’я> <пароль>.

При успішному вході у систему у консолі виводиться відповідне повідомлення. При не успішному вході окрім відповідного повідомлення також повертається статус код рівний 1, що особливо зручно при використанні у CI/CD системах. На рис. 1 наведено приклад успішної та неуспішної авторизації.

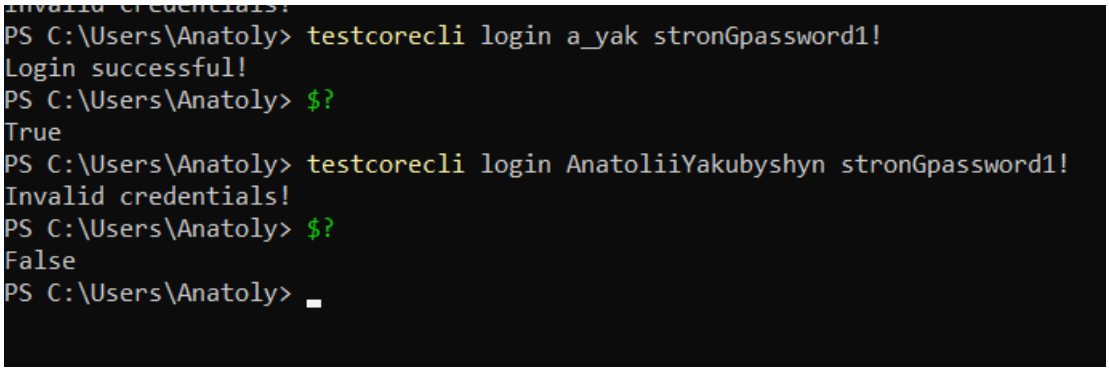


Рис. 1. Приклад використання командного рядка для успішної та неуспішної авторизації

Останній успішний вхід у системі запам’ятовується, тому вказування логіну та паролю для виконання інших операцій, як перші два аргументи є необов’язковим. Також якщо вхід був виконаний за допомогою інструменту командного рядка, то в графічному клієнті підтягнеться останній успішний вхід і навпаки.

Якщо користувач мав хоча б один успішний вхід, то він перейде одразу на сторінку проєктів цього користувача. Для того, щоб змінити

користувача необхідно натиснути «change user/logout» кнопку, відкриється сторінка авторизації/реєстрації, далі після натискання кнопки з написом «Login» відкриється вікно авторизації з якого можна повернутися назад до попередньої сторінки натиснувши кнопку з написом «Go back». На рис. 2 наведено зображення вікна входу у систему.

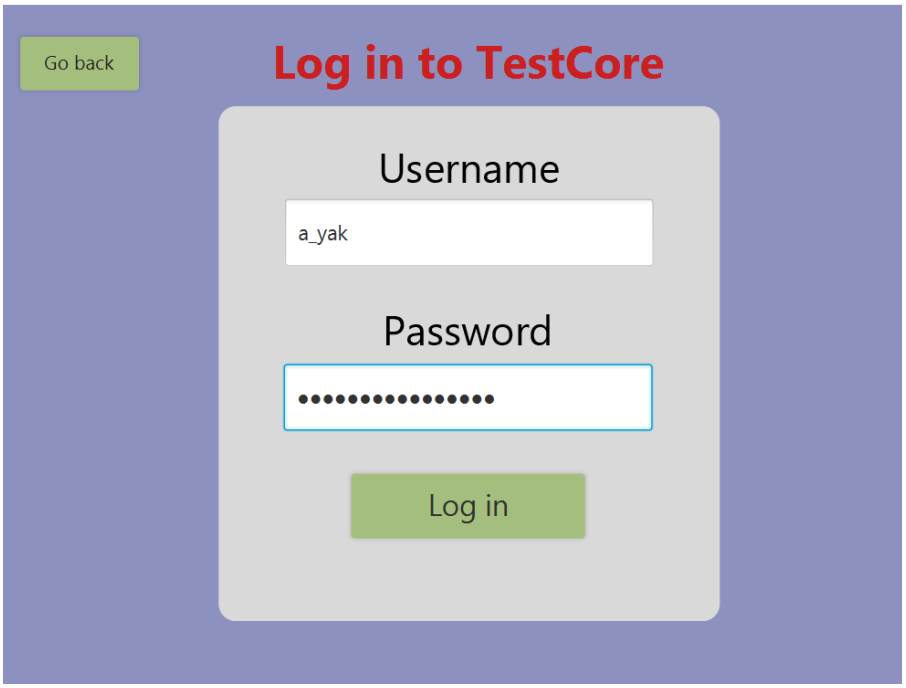


Рис 2. Вікно входу у систему

3. Процедура реєстрації користувача

Для реєстрації також можна скористатися або командним рядком або кросплатформним настільним додатком. Система має перевірку на силу паролю. При чому ця перевірка іде на рівні серверної частини, що унеможливлює створити користувача з слабким паролем через REST API. Так само як і в процесі авторизації при неуспішному виконанні команди повертається ненульовий код. Цей підхід застосовується для всіх операцій, надалі цей момент не згадуватиметься. Для виконання реєстрації за допомогою командного рядка використовується команда testcorecli register

<ім’я> <пароль>. На рис. 3 наведено приклад неуспішної реєстрації при використанні слабкого паролю.

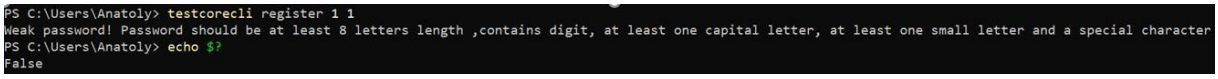


Рис. 3. Приклад неуспішної спроби реєстрації за допомогою інструменту командного рядка через слабкий пароль

Також реєстрація може завершитися невдало, якщо користувач вже існує у системі. При чому даний тип помилки має більший пріоритет, тобто при використанні існуючого імені користувача і слабкого паролю користувач отримає повідомлення в консолі, що дане ім’я вже використане. На рис. 4 наведено приклад реєстрації з використанням існуючого ім’я користувача.

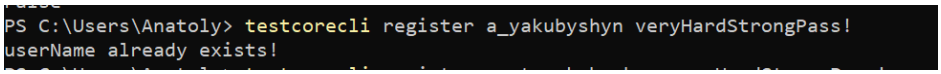


Рис. 4. Приклад неуспішної спроби реєстрації за допомогою інструменту командного рядка через використання існуючого імені користувача у системі

Якщо всі умови успішної реєстрації виконано – в консолі буде виведено про це повідомлення. Приклад наведено на рис. 5.

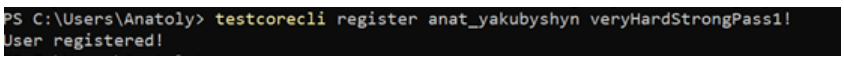


Рис. 5. Приклад успішної реєстрації користувача у системі

Щоб додатково перевірити чи користувач зареєструвався можна перевірити ім’я поточного користувача за допомогою команди get_user.

Також зареєструватися можна і через графічний інтерфейс. Якщо користувач заходить у систему перший раз у нього з’явиться вікно в якому необхідно натиснути на кнопку «Register», далі відкриється вікно з формою

реєстрації в якій необхідно вказати ім’я користувача, пароль та підтвердити пароль. На рис. 6 наведено вікно з формою реєстрації користувача.

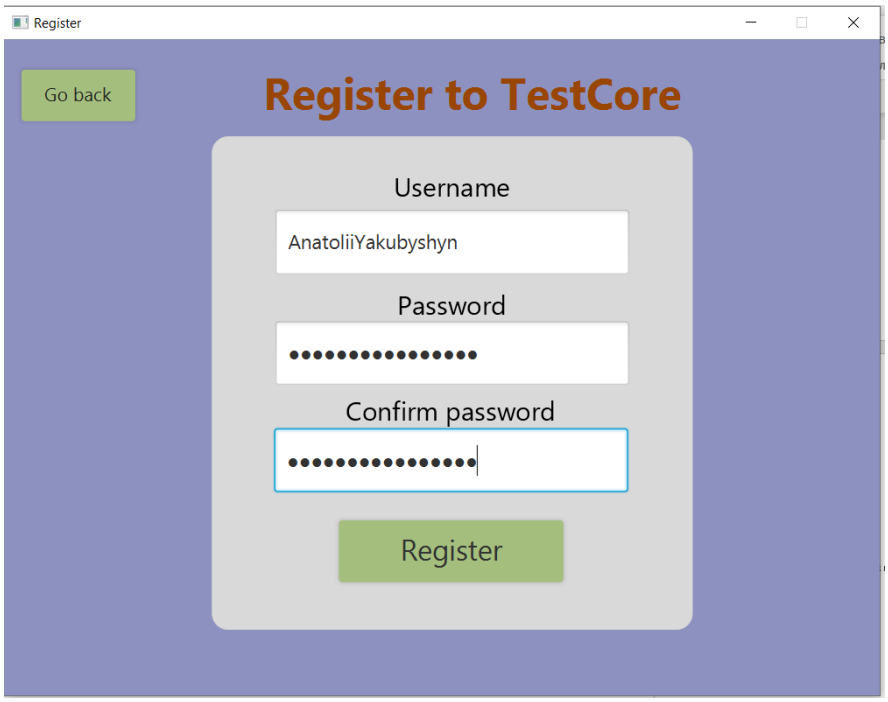


Рис. 6. Форма реєстрації користувача

Під час введення паролей у відповідні поля, поля не будуть відображати свій вміст, приховуючи символи.

4. Процедура перегляду проєктів користувача

Для перегляду проєктів користувача можна скористатися інструментом командного рядка або графічним інтерфейсом.

Для отримання списку назв проєктів користувача з їхніми ідентифікаторами у JSON форматі протрібно виконати команду: testcorecli get_projects, додатково можна виконати testcorecli <ім’я користувача> <пароль> get_projects для отримання інформації про проєкти користувача відмінного від поточного. На рис. 7 наведено процес виконання цієї команди.

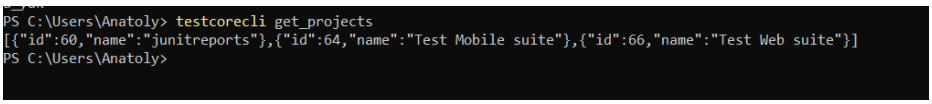


Рис. 7. Результат виконання команди для отримання списку проєктів користувача

Для того, щоб отримати інформацію про користувача за допомогою графічного інтерфейсу потрібно просто запустити настільний додаток у випадку, якщо був виконаний вхід у систему хоча б один раз, якщо це перший вхід у систему потрібно спочатку зареєструватися/авторизуватися, після цього відкриється вікно проєктів, що на самому початку не буде містити жодного. На рис. 8 наведено вікно проєктів користувача.

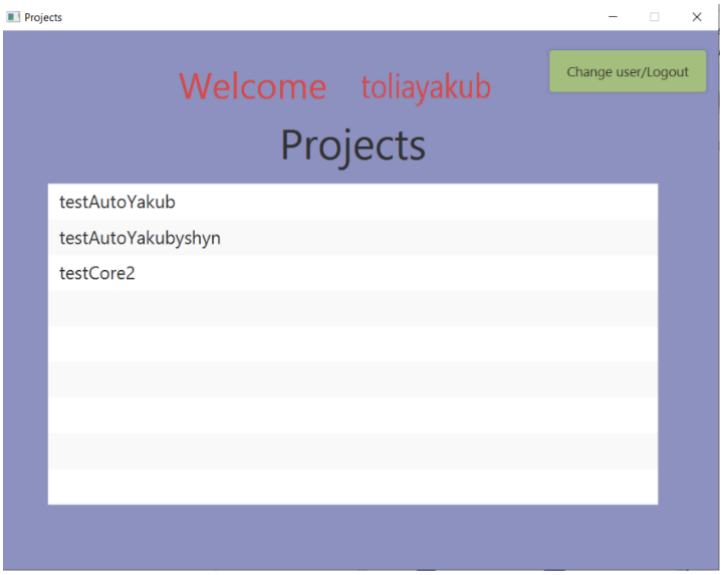


Рис. 8. Вікно проєктів користувача

5. Процедура перегляду тестових запусків та тестових наборів

Користувач може мати кілька проєктів, проєкт може містити декілька тестових запусків, тестові запуски містять тестові набори, а тестові набори містять результати тестів. Переглядати інформацію про вищезгадані сутності можна за допомогою інструменту командного рядка або через графічний інтерфейс.

Для отримання інформації по тестовим запускам необхідно виконати команду testcorecli get_test_runs <id проєкту>. Для того щоб отримати інформацію про тестові набори потрібно виконати команду testcorecli get_test_suites <id тестового запуску>. Результатом виведення команд будуть списки у форматі JSON виведені у консоль.

Щоб переглянути цю інформацію за допомогою графічного інтерфейсу потрібно вибрати необхідний тестовий проєкт або тестовий запуск зі списку. Щоб повернутися на попередній рівень потрібно натиснути кнопку з написом «Go back». На рис. 9 наведено вікно тестових запусків користувача.

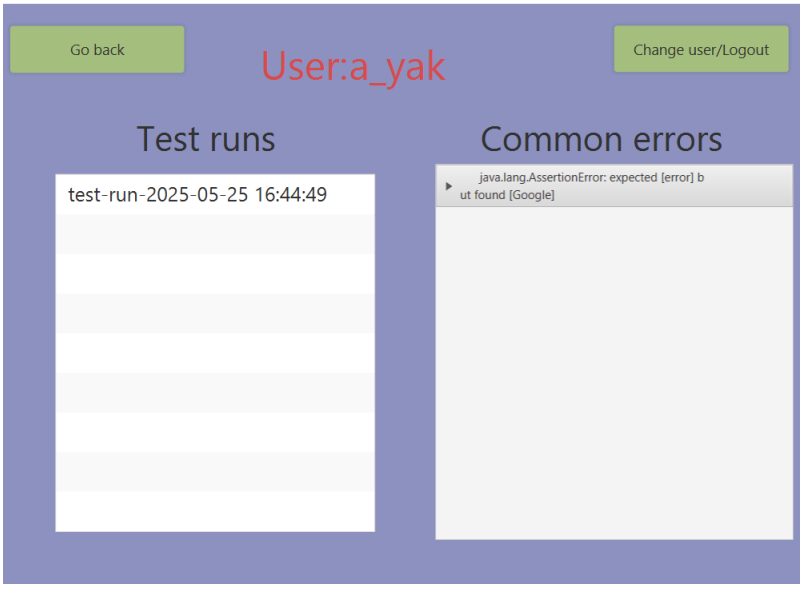


Рис. 9. Вікно тестових запусків користувача

6. Процедура перегляду та зміни результатів проходження тестів

Результат проходження тестів можна переглянути у вікні тестового набору, де результати подані у вигляді таблиці. Також це можна зробити за допомогою інструменту командного рядка.

Для того, щоб подивитися результат окремо по одному тесту потрібно скористатися командою testcorecli get_test_case <id тест-кейсу>, якщо потрібно отримати результати всіх тестів по тестовому набору потрібно

скористатися командою testcorecli get_test_cases <id тестового набору>. Результати виводяться у консолі у форматі JSON.

На рис. 10 наведено вікно результатів проходження тестів з відовідною таблицею.

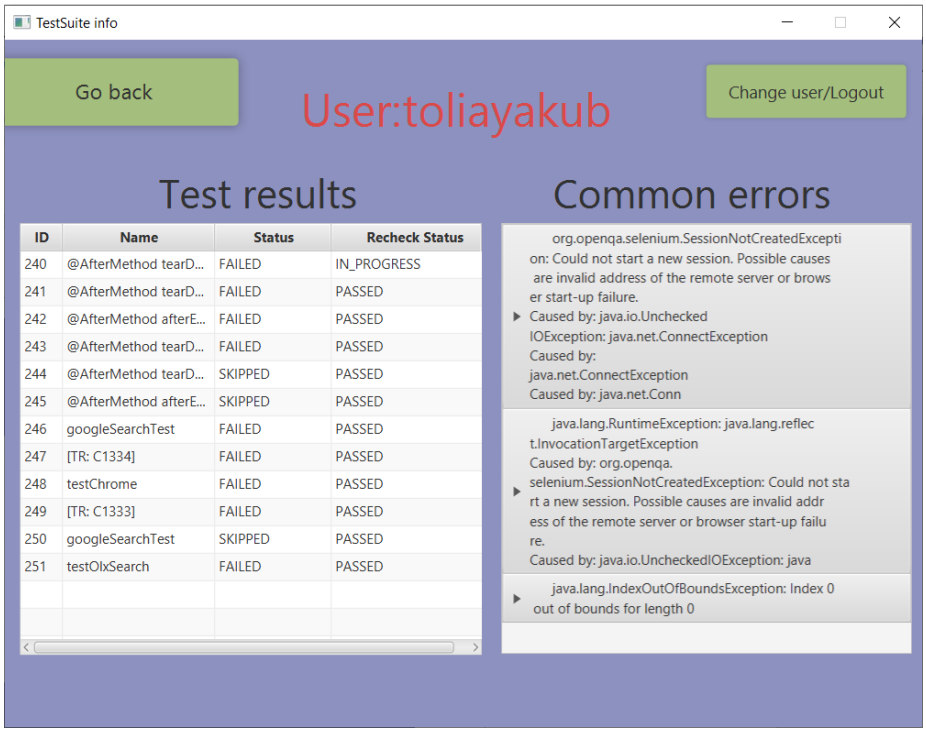


Рис. 10. Вікно результатів проходження тестів

Для того, щоб змінити статус проходження тесту, потрібно скористатися командою testcorecli set_status <id тесту> або натиснути на відовідну клітинку у колонці «Recheck status». Статус проходження тесту зміниться в клітинці колонки «Recheck status» в обох варіантах при чому збережеться перший результат проходження тесту в колонці «Status».

7. Процедура перегляду результатів групування тестів за спільними помилками

Для того, щоб переглянути результати групування тестів за спільними помилками потрібно виконати команду testcorecli get_common_errors

<рівень перегляду>. Також результат групування можна побачити у вікні проєкту, тестового запуску, тестового набору. Приклад результату групування можна побачити на рис. 9 – 10 під написом «Common errors». Рівень project – це групування на рівні проєкту, test_run – групування на рівні тестового запуску, test_suite – групування на рівні тестового набору. Рівні перегляду: project, test_run, test_suite. На рис. 11 наведено результат групування тестів за допомогою інструменту командного рядка.

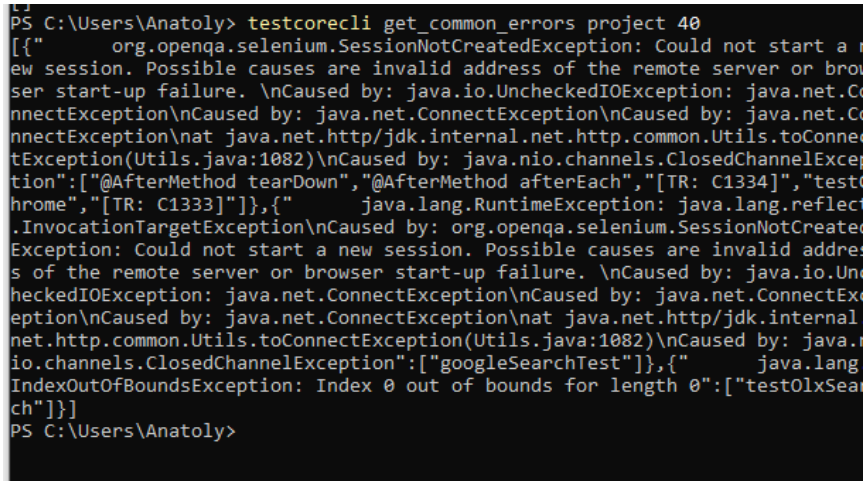


Рис. 11. Результат виконання групування на рівні проєкту за допомогою інструменту командного рядка

8. Процедура перегляду результатів групування тестів за спільними помилками

Для ініціалізації шаблону проєкту для автоматизації тестування програмного забезпечення використовується команда testcorecli download_java_template <назва проєкту>. В результаті виконання команди завантажується архів у zip форматі шаблону проєкту та автоматично розпаковується у теку з назвою проєкту, вказаного під час виклику команди. Ця функціональність доступна лише в інструменті командного рядка, що відображена на рис 12.

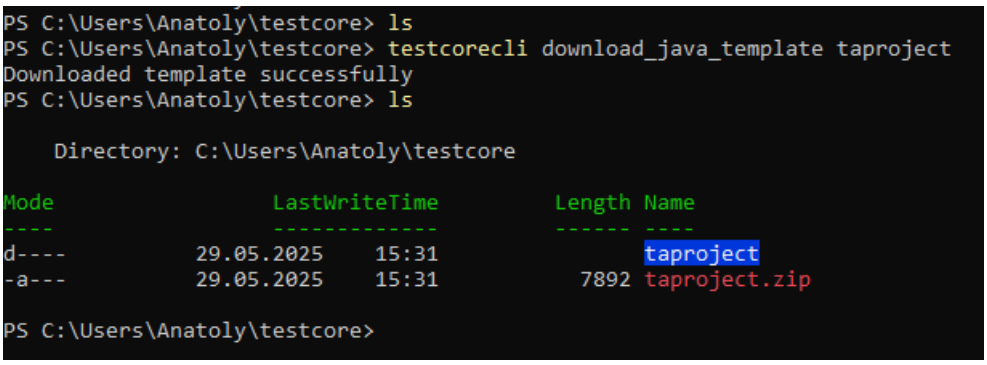


Рис. 12. Приклад виконання команди для завантаження шаблону проєкту для автоматизації тестування програмного забезпечення

9. Процедура створення тестів

Першим кроком для створення тестів потрібно відкрити шаблон проєкту в IDE для розробки мовою Java. Для запуску тестів необхідно створити конфігураційний файл з розширенням properties. Файлів конфігурацій може бути декілька і вони підключаються за допомогою статичного методу setConfigPath класу Config. На рис. 13 наведено приклад конфігураційного файлу для запуску тестів на найновіші версії chrome driver з автоматичним підвантаженням необхідних компонент.

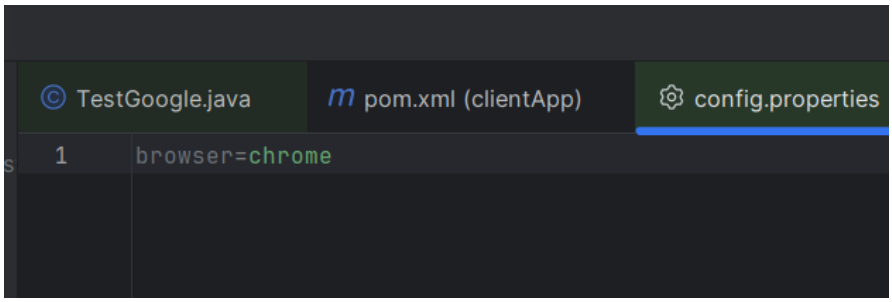


Рис. 13. Приклад вмісту конфігураційного фалу для запуску тестів на найновішій версії chrome driver

Також фреймворк надає можливість запуску тестів за допомогою Selenium grid та Selenoid. За замовчуванням вони запускаються на localhost на порту 4444. На рис. 14 наведено приклад такої конфігурації.

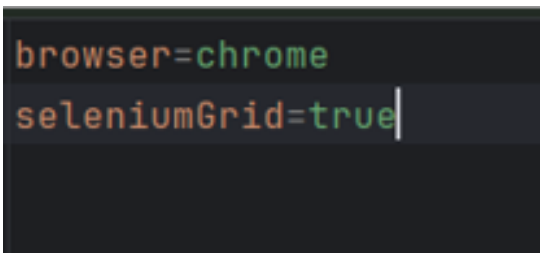


Рис. 14. Приклад конфігураційного файлу для запуску на selenium grid

Також шаблон підтримує написання Android тестів для цього параметри емулятора також потрібно задати в конфігураційному файлі. Також потрібно задати url для Appium. На рис. 15 наведено приклад такої конфігурації.

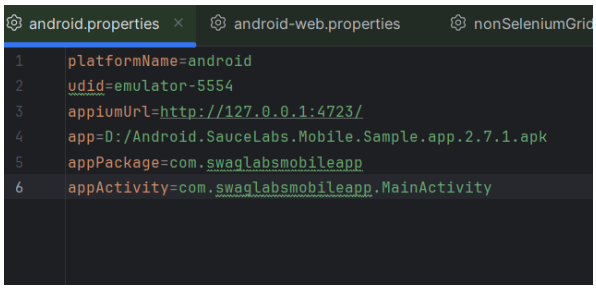


Рис. 15. Приклад конфігурації для запуску тестів на Android емуляторі

Також можливо задати окремо конфігурацію для запуску тестів з мобільного браузера з емулятора, наведено на рис. 16.

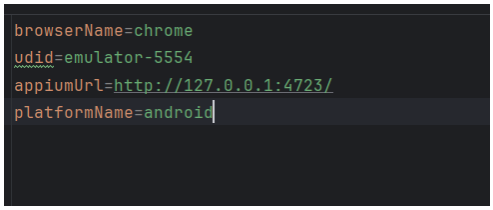


Рис. 16. Приклад конфігурації для запуску тестів на браузері Android емулятора

Необхідні конфігурації драйверу браузеру можна задати через формат: caps.<необхідне поле конфігурації>. Найчастішим випадком такого використання є headless режим, що наведено на рис. 17.

```
browser=chrome
seleniumGrid=false
caps.headless=true
```

Рис. 17. Приклад конфігурації headless режиму

Для виконання операцій в браузері без використання шаблону POM можна використовувати DriverHelper клас. Приклад такого використання наведено на рис. 18.

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

import com.yakubt.TestCoreLib.Config;
import com.yakubt.TestCoreLib.ui.driver.DriverHelper;

public class TestGoogle {

    @Test
    public void test() throws InterruptedException {
        Config.setConfigPath("src/test/resources/config.properties");
        DriverHelper.getInstance().getDriver().get("https://www.google.com/");
        Assert.assertTrue(DriverHelper.
                getInstance().getDriver().getTitle().toLowerCase().contains("google"));
    }
}
```

Рис. 18. Приклад простого тесту перевірки заголовку сторінки google

Для створення репрезентації UI компоненти в коді потрібно унаслідуватися від класу UIComposite.

Для створення дочірних елементів використовуються методи: createChildNode, createListOfChildNodes. Першим параметром яких є локатор, а другим тип дочірнього елементу. На рис. 19 наведено приклад створеної компоненти.

```
3 usages
public class SearchComponent extends UIComposite {

    1 usage
    private ExtendedWebElement searchInput = createChildNode(
            By.xpath( xpathExpression: "//textarea[contains(@name,'q')]"), ExtendedWebElement.class);

    no usages
    public SearchComponent(By locator, WebDriver driver, IComposable parentNode) { super(locator, driver, parentNode); }

    1 usage
    public GoogleResultPage makeRequest(String text) {
        searchInput.getWebElement().sendKeys( ...keysToSend: text + Keys.ENTER);
        return new GoogleResultPage();
    }

}
```

Рис. 19. Приклад створеної компоненти

Для взаємодії з об’єктом класу ExtendedWebElement потрібно викликати метод getElement() і далі використовувати стандартні методи Selenium web елементу. Тобто ExtendedWebElement є обгорткою. Такий підхід був необхідний для підтримки вкладеності об’єктів інтерфейсу та їх перевикористання між різними сторінками.

Для забезпечення кросплатформеності використовується анотація @PlatformType, через яку можна задати тип page object класу: mobile чи web. Для використання об’єкту класу page object необхідно викликати статичний метод initPage(<тип класу>). Це метод автоматично підтягне необхідну реалізацію page object відповідно до налаштувань в конфігураційному файлі. На рис. 20 наведено приклад використання анотації PlatformType.

Рис. 20. Приклад використання анотації PlatformType

На рис. 21 наведено приклад використання методу initPage.

Рис. 21. Приклад використання методу initPage

Для написання API тестів чи виконання API викликів використовується клас RequestSender. На рис. 22 наведено приклад тесту API.

Рис. 22. Приклад API тесту

10. Процедура запуску тестів

Для запуску тестів можна скористатися TestNG плагіном та запускати тести з IDEA відповідно.

Для процесів CI/CD використовуються команди системи збірки Maven. В шаблоні встановлений необхідний плагін для запуску TestNG test_ng файлів. Нижче наведено приклад запуску кількох test_ng файлів одночасно. Звичайно можна і не використовувати test_ng файли, а запускати всі тести відразу без групування. На рис. 23 наведено приклад запуску з таким групуванням.

Рис. 23. Приклад запуску тестів за допомогою Maven

11. Процедура надсилання результатів тестових запусків на сервер

Для того, щоб відправити результати тестів на сервер використовується команда CLI testcorecli send_result. Інструмент

командного рядка автоматично відправить результати запуски тестів з теки target викростовуючи ім’я проєтку для розробки на Java в якості проєкту тестових запусків. Команда має бути виконана з кореневої директорії проєкту. На рис. 24 наведено приклад використання команди send_result.

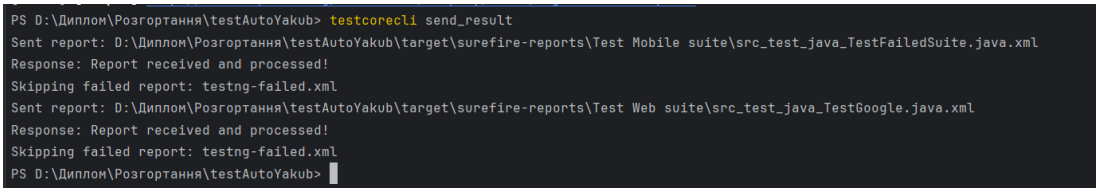


Рис. 24. Приклад використання команди send_result