Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інструментального набору для автоматизованого тестування програмного забезпечення на основі фреймворка Cucumber та нотації Gherkin

Виконала: студентка IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

(підпис) Береза Я. О.
(прізвище та ініціали)

Керівник (підпис) Готович В.А.
(прізвище та ініціали)

Нормоконтроль (підпис) Литвиненко Я.В.
(прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О.
(прізвище та ініціали)

Рецензент (підпис) Дідич І. С.
(прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)

Студентці Березі Ярині Олександрівні
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка інструментального набору для автоматизованого тестування програмного забезпечення на основі фреймворка Cucumber та нотації Gherkin

Керівник роботи Готович Володимир Анатолійович, к.т.н., доцент кафедри КН
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 7 » лютого 2023 року № 4/7-133
2. Термін подання студентом завершеної роботи 19.06.2023р.
3. Вихідні дані до роботи Перелік літературних та інтернет джерел

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз поставленої задачі. 1.1 Постановка задачі. 1.2 Моделювання архітектури програмного забезпечення. 1.3 Вибір архітекрури проєкту. 1.4 Вибір технології розробки. 1.5 Вибір допоміжних інструментів розробки. 1.6 Висновок до першого розділу. 2. Проєктування та реалізація програмного рішення. 2.1 Проєктування фреймворка Cucumber та його структура. 2.2 Реалізація інтерфейсу на основі фреймворка Cucumber та нотації Gherkin. 2.3 Процес та етапи тестування. 2.4 Тестування API. 2.5. Використання розробленого інструменту. 2.6 Висновок до другого розділу. 3. Безпека життєдіяльності, основи охорони праці. 3.1 Соціальні та психологічні фактори ризику. 3.2. Загальні вимоги безпеки з охорони праці для користувачів ПК. 3.3 Соціальне значення охорони праці.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., доцент кафедри МТ	05.06.2023	08.06.2023

7. Дата видачі завдання 23 січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	23.01.2023	*Виконано*
2.	Підбір джерел про технології розробки	24.01.2023-26.01.2023	*Виконано*
	інструментального набору для автоматизованого		
	тестування програмного забезпечення		
3.	Переклад та опрацювання джерел про технології	27.01.2023-31.02.2023	*Виконано*
	тестування програмного забезпечення.		
4.	Розробка інструментального набору для	01.02.2023-07.02.2023	*Виконано*
	автоматизованого тестування програмного		
	забезпечення на основі фреймворка Cucumber та нотації Gherkin		
5.	Оформлення розділу «1. Аналіз поставленої задачі»	08.02.2023-09.02.2023	*Виконано*
6.	Оформлення розділу «2. Проектування та реалізаціяпрограмного рішення»	10.02.2023-12.02.2023	*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	05.06.2023-06.06.2023	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони праці»	07.06.2023-08.06.2023	*Виконано*
9.	Оформлення кваліфікаційної роботи	09.06.2023-11.06.2023	*Виконано*
10.	Нормоконтроль	12.06.2023-13.06.2023	*Виконано*
11.	Перевірка на плагіат	14.06.2023	*Виконано*
12.	Попередній захист кваліфікаційної роботи	15.06.2023	*Виконано*
13.	Захист кваліфікаційної роботи	19.06.2023	

Студент _____ Береза Я.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інструментального набору для автоматизованого тестування програмного забезпечення на основі фреймворка Cucumber та нотації Gherkin // Кваліфікаційна робота освітнього рівня «Бакалавр» // Береза Ярина Олександрівна // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2023 // С. 48, рис. – 7 , табл. – 2, кресл. – , додат. – 10, бібліогр. – 28.

Ключові слова: фреймворк, автоматизоване тестування, Cucumber, нотація Gherkin, API, тест кейс.

Кваліфікаційна робота присвячена дослідженню і розробці інструментарію для автоматизованого тестування програмного забезпечення на основі фреймворку Cucumber та нотації Gherkin.

В першому розділі кваліфікаційної роботи розглянуто декілька інструментів та середовищ для створення фреймворка для автоматизованого тестування та обгрунтовано рішення щодо вибору оптимального інструментального набору засобів, відповідного до поставленого в роботі завдання.

В другому розділі кваліфікаційної роботи наведено опис архітектури та процесу проєктування фреймворка. Також описано реалізацію та практичне використання розробленого інструментального набору.

В третьому розділі кваліфікаційної роботи описано соціальні та психологічні фактори ризику для життя і здоров'я людей. Наведено загальні вимоги безпеки з охорони праці для користувачів персональних комп'ютерів та описано соціальне значення охорони праці вцілому.

ANNOTATION

Development of a toolkit for automated software testing based on the Cucumber framework and Gherkin notation // Qualification work of the educational level "Bachelor" // Bereza Yaryna Oleksandrivna // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-41 // Ternopil, 2023 // P. 48, fig. - 7 , tabl. - 2 , chair. - , annexes. – 10, references - 28.

Keywords: framework, automation testing, Cucumber, Gherkin notation, API, test case.

The qualification work is devoted to the research and development of a toolset for automated software testing based on the Cucumber framework and Gherkin notation.

The first chapter of the qualification work discusses several tools and environments for creating a framework for automated testing and justifies the decision regarding the selection of an optimal set of tools that align with the objectives set in the work.

The second chapter of the qualification work provides an overview of the architecture and design process of the framework. It also describes the implementation and practical use of the developed toolset.

The third chapter of the qualification work describes social and psychological risk factors for human life and health. General safety requirements for occupational health and safety for personal computer users are provided, and the social significance of occupational health and safety is described in general.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – набір правил і протоколів, які дозволяють різним програмам взаємодіяти одна з одною. Він визначає, які функції та операції можуть виконуватися програмою, які дані можуть передаватися та які формати даних використовуються при обміні інформацією.

BDD (Behavior-Driven Development), або розробка, орієнтована на поведінку – методологія розробки програмного забезпечення, яка ставить акцент на спільне розуміння вимог до системи між розробниками, тестувальниками і бізнес-аналітиками.

IDEA (Integrated Development Environment) – інтегроване середовище розробки, яке використовується для написання, редагування і відлагодження програмного коду.

ISO (англ. International Organization for Standardization) – міжнародна організація, яка розробляє та публікує міжнародні стандарти для різних галузей.

JVM (Java Virtual Machine) – віртуальна машина Java, яка виконує Java-програми. Вона є частиною Java Runtime Environment (JRE) і відповідає за виконання байт-коду, який генерується під час компіляції Java-програм.

QA (Quality Assurance), або гарантія якості, є процесом і практикою, спрямованими на забезпечення якості продукту або послуги. QA включає в себе набір дій, методів і стандартів, які використовуються для контролю та визначення відповідності вимогам якості.

URL (Uniform Resource Locator) є адресою, що ідентифікує розташування ресурсу в Інтернеті.

Фреймворк – структурована платформа або набір програмних компонентів, які надають основу для розробки програмного забезпечення.

ЗМІСТ

ВСТУП

Актуальність теми роботи. У сучасному світі розробка програмного забезпечення стає все складнішою і вимагає від розробників та тестувальників швидкої та ефективної роботи. Автоматизоване тестування відіграє важливу роль у забезпеченні якості програмного забезпечення та зниженні ризиків.

Одним з найпопулярніших інструментів для автоматизації тестування є фреймворк Cucumber та його нотація Gherkin. Cucumber надає можливість писати тести у зрозумілому для бізнесу форматі, що сприяє співпраці між розробниками, тестувальниками та представниками бізнесу.

Мета та завдання дослідження. Метою кваліфікаційної роботи є розробка інструментарію для автоматизованого тестування програмного забезпечення на основі фреймворку Cucumber та нотації Gherkin.

Цей інструментарій допоможе розробникам та тестувальникам швидко та ефективно створювати та виконувати автоматизовані тести, забезпечуючи високу якість продукту та знижуючи час та зусилля, витрачені на тестування.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Проаналізувати відомі засоби і методи розробки засобів для автоматизованого тестування програмного забезпечення;
2. Здійснити розробку та налаштування інструментального набору для автоматизованого тестування програмного забезпечення на основі фреймворку Cucumber та нотації Gherkin;
3. Продемонструвати практичне застосування розробленого інструментального набору.

Об’єкт дослідження. Процес розробки інструментального набору для автоматизованого тестування програмного забезпечення.

Предмет дослідження. Засоби та методи розробки інструментального набору для автоматизованого тестування програмного забезпечення на основі фреймворку Cucumber та нотації Gherkin.

Практичне значення отриманих результатів. Розроблено придатний для практичного застосування інструмент для автоматизованого тестування

програмного забезпечення. Він є важливим внеском у розробку та підтримку якісного програмного забезпечення та покликаний допомогти прискорити процес розробки та знизити ризики виникнення дефектів.

РОЗДІЛ 1. ВИБІР ТЕХНОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ

1.1 Аналіз поставленої задачі

Постановка задачі передбачає розробку комплексного інструментарію, який буде використовуватись для автоматизації тестування програмного забезпечення з використанням підходу Behavior-Driven Development (BDD).

Головна мета - це створення ефективного та гнучкого інструментального набору на основі Cucumber фреймворка, який дозволить якісно протестувати веб-сайт за допомогою тест кейсів, написаних на Gherkin синтаксі.

Отож, для досягнення цієї мети, встановлені наступні завдання. По-перше, необхідно дослідити основні принципи та можливості фреймворку Cucumber та нотації Gherkin, а також їх взаємодію з різними мовами програмування. По-друге, треба розробити інтегрований набір інструментів, який буде включати засоби для створення, виконання та аналізу тестових сценаріїв на основі Cucumber та Gherkin. Цей набір повинен забезпечувати зручну роботу зі сценаріями, можливість автоматизованого виконання тестів, а також отримання зрозумілих звітів і результатів тестування. Нарешті, треба провести експериментальне тестування розробленого інструментального набору. Виконання цих завдань допоможе вирішити проблему ефективного автоматизованого тестування програмного забезпечення з використанням фреймворку Cucumber та нотації Gherkin.

Забезпечення якості (QA) – це акт або процес підтвердження того, що вимоги організації до якості виконуються. Управління якістю продукції включає планування, реалізацію та моніторинг діяльності. Забезпечення якості є моніторинговим аспектом цієї дисципліни.

QA включає процеси та процедури, які виконуються систематично для того, щоб контролювати різні аспекти служби чи об’єкта. За допомогою аудитів та інших форм оцінювання зусилля із забезпечення якості виявляють і усувають проблеми або невідповідності, які виходять за межі встановлених стандартів

або вимог. Іншими словами: забезпечення якості гарантує високий рівень якості при розробці продуктів або послуг.

Термін «забезпечення якості» іноді використовується як взаємозамінний з «контролем якості», ще одним аспектом процесу управління. Проте контроль якості означає фактичне виконання будь-яких встановлених вимог до якості.

QA – це перевірка методів контролю якості, щоб переконатися, що вони працюють належним чином.

Основна відмінність між забезпеченням якості та контролем якості полягає в тому, що діяльність із забезпечення якості виконується під час розробки програмного забезпечення, а заходи з контролю якості виконуються після розробки програмного забезпечення [1-2].

Концепції контролю якості можна простежити принаймні до середньовіччя та появи гільдій. Ремісник міг отримати доступ до мережі зв’язків з іншими майстрами та постачальниками, приєднавшись до цехової організації. Тоді він міг отримати користь від репутації гільдії, заснованої на стандартах якості продукції, виготовленої її членами.

Міжнародна організація зі стандартизації (ISO) була заснована в 1947 році для забезпечення якості за межами національних кордонів. ISO складається з організацій стандартизації, які представляють понад 160 країн. Він підтримує ефективну систему забезпечення якості виробництва та сфери послуг. Одним із продуктів ISO є набір стандартів, тепер відомий як сімейство ISO 9000. Критерії, наведені в цих системах менеджменту, покликані допомогти організаціям відповідати законодавчим і нормативним вимогам якості продуктів і потреб споживачів.

Міжнародна організація стандартизації (ISO) опублікувала повний огляд свого стандарту ISO 9001. Цей огляд представляє низку важливих змін як для організацій, які вже мають цей сертифікат, так і для тих, хто хоче розробити та впровадити систему управління якістю [3].

1.2 Моделювання архітектури програмного забезпечення

Після ознайомлення з темою та розуміння того, що потрібно для майбутньої програмної системи, досліджуються можливі шляхи вирішення поставлених у вимогах завдань. Таким чином, виконується аналіз області прийняття рішень, явно чи неявно. Мета цієї діяльності – зрозуміти, чи можливо вирішити поставлені перед системою, що розробляється, завдання, за яких умов і обмежень це можливо, як вони вирішуються, чи є рішення, а якщо ні, то чи існує воно. Ви можете знайти спосіб його знайти або хоча б отримати приблизне рішення. Коли основні способи вирішення всіх завдань встановлені, головною проблемою є організація програмної системи таким чином, щоб вона дозволяла реалізувати всі ці рішення при задоволенні потреб програми. Шуканий спосіб організації програмного забезпечення як системи взаємодіючих компонентів називається архітектурою, а процес його створення – проєктуванням архітектури програмного забезпечення.

Архітектура програмного забезпечення – це набір внутрішніх структур програмного забезпечення, видимих з різних точок зору, що складається з компонентів, їхніх зв’язків і можливих взаємодій між компонентами, а також зовнішніх доступних властивостей цих компонентів. У даному контексті, термін "компонент" відноситься до гнучкого структурного елемента програмного забезпечення, який можна ідентифікувати шляхом визначення інтерфейсу взаємодії між цим компонентом та його оточенням. Традиційно термін «компонент» у розробці програмного забезпечення відноситься до блоку доставки, найменшої частини системи, яка може бути включена або не включена. Такий компонент також має специфічний інтерфейс і підпорядковується деяким так званим правилам компонентної моделі.

Цей термін вживається в значенні архітектури структури, а не структури. Це означає, що різні структури, що відповідають різним аспектам взаємодії компонентів, виділяються як різні аспекти архітектури та різні точки зору на неї.

Архітектура програмного забезпечення демонструє структуру, яка поєднується зі структурними елементами програмного забезпечення шляхом з’єднання всіх циклів даних. Архітектура створює характеристику якості програмного забезпечення в цілому. Архітектура виконує також роль основного комунікаційного засобу між розробниками, а також між розробниками та іншими стейкхолдерами у контексті програмного забезпечення. Вибір архітектури дає можливість реалізувати вимоги на високому рівні абстракції. Сама архітектура майже повністю розуміє такі характеристики програмного забезпечення, як надійність, портативність і ремонтопридатність. Це також істотно впливає на зручність використання та ефективність програмного забезпечення, яка, однак, сильно залежить від реалізації окремих компонентів. Традиційно набагато менший вплив архітектури на функціональність.

Отже, вибір між тією чи іншою архітектурою більшою мірою розширюється саме нефункціональними вимогами та необхідними властивостями програмного забезпечення з точки зору ремонтопридатності та переносимості. Щоб створити оптимальну архітектуру, необхідно знайти баланс між вимогами різних характеристик та уміти знаходити компромісні рішення, які забезпечать прийнятну вартість за всіма аспектами. Загалом, для підвищення ефективності рекомендується використовувати монолітні архітектури, де кількість компонентів обмежена і складає лише один компонент. Це економить як пам'ять, оскільки кожен компонент фактично має свої дані, а тут кількість компонентів мінімальна, так і робочий час, оскільки можливість оптимізувати роботу алгоритмів обробки даних також є в межах одного компонента.

Навпаки, для підвищення продуктивності системи рекомендується розбити її на багато окремих, невеликих компонентів, кожен з яких виконує свою чітко визначену функціональну частину загальної задачі. А якщо це зміни у вимогах або проектах, то традиційно вони можуть стосуватися зміни в постановці однієї, рідше двох-трьох таких підзадач, і тільки для вирішення цих підзадач компоненти змінюються.

По-третє, для підвищення надійності краще використовувати або невеликий набір простих компонентів, або дублювання функцій. Однак передбачається, що помилки в програмному забезпеченні не випадкові. Вони повторювані і повинні бути компенсовані апаратним забезпеченням, де помилки часто пов’язані з випадковими змінами властивостей середовища і можуть бути подолані простим дублюванням компонентів без зміни їх внутрішньої реалізації [12-14]. Отже, для забезпечення такого рівня надійності важливо застосовувати різноманітні підходи до вирішення конкретних завдань у різних компонентах системи. Іншим прикладом відповідних вимог до послуг є функції зручності використання та захисту. Чим сильніше система захищена, тим більше користувачі мають пройти перевірки, процедури ідентифікації тощо.

1.3 Вибір архітектури проєкту

Розробка інструментального набору для автоматизованого тестування програмного забезпечення на основі фреймворка Cucumber та нотації Gherkin є складним завданням, яке вимагає ретельного вибору архітектури проекту. Архітектура проекту визначає його структуру, взаємозв'язки компонентів, принципи взаємодії між ними та загальну організацію розробки. Вибір оптимальної архітектури є критичним для успішної реалізації проекту та досягнення його цілей.

Отож мною розроблена архітектура:

– Frontend тестування з використанням Selenium WebDriver: Використання Selenium WebDriver дозволить нам автоматизувати тестування інтерфейсу веб-сайту. Selenium WebDriver надає багато можливостей для взаємодії з елементами сторінки, заповнення форм, виконання дій користувача та перевірки результатів. Я використала мову програмування Java, а розробку та виконання тестів проводила у середовищі розробки Intellij.[6]

– BDD-підхід з використанням Cucumber та Gherkin: Для забезпечення легкості читання та зрозумілості тестів, я використала Cucumber

разом з Gherkin для написання тестових сценаріїв в структурованому мовному форматі. Використання Cucumber дозволить нам зв'язати ці сценарії з автоматизованими тестами, написаними з використанням Selenium WebDriver, та забезпечити виконання тестів на основі цих сценаріїв.[5]

– Тестування API з використанням RestAssured та Postman: Так як веб-сайт, який я тестую має API, я використала RestAssured для автоматизованого тестування API-запитів та перевірки відповідей сервера. Postman також може бути використаний для виконання та автоматизації тестів API. Ці інструменти допоможуть перевірити функціональність API та зв'язати ці тести зі звичайними UI-тестами, написаними з використанням Selenium WebDriver.[23]

Для розробки інструментального набору для тестування я обрала фреймворк із досить високою популярністю – Cucumber. Він базується на концепції Behavior-Driven Development (BDD), що дозволяє визначати вимоги щодо функціональності у форматі легкозрозумілих історій.

Цей фреймворк може працювати з різними мовами програмування, такі як Java, JavaScript і Ruby. Cucumber має безкоштовний, опен-сорсний варіант та платну версію Cucumber Pro, яка пропонує комфортну взаємодію членів команди в рамках CucumberStudio.

Однією з ключових особливостей Cucumber є його природньо-мовний синтаксис, який дозволяє створювати тестові сценарії у простому і зрозумілому для всіх учасників проекту форматі. Сценарії записуються у текстових файлах з розширенням .feature, які містять опис кроків, передумови та очікувані результати.

Фреймворк дозволяє автоматизувати виконання тестових сценаріїв шляхом зв'язування кроків зі специфікаціями до коду. Кожен крок в описі сценарію пов'язується з відповідним кодом, який виконує необхідні перевірки та дії.

Cucumber також підтримує інтеграцію з різними інструментами та середовищами розробки, такими як Selenium, Appium, Jenkins, і багатьма

іншими. Це дозволяє розширити можливості фреймворка та легко інтегрувати його в робоче середовище [4-6].

Насамперед, зазначу, що більшість існуючих сьогодні BDD-фреймворків є просто різні версії Cucumber. Крім того, це можуть бути:

– похідні від фреймворку JBehave, які написані в стилі його творця (Dan North);
– non-Gherkin спек-раннери.

Одні фреймворки дозволяють організовувати сценарії поведінки в окремі файли, тоді як інші розміщують їх у вихідному коді.

Говорячи про фреймворки автоматизації для Java і JVM-мов, варто згадати два основних конкуренти:

– Cucumber-JVM;
– JBehave.

Cucumber-JVM є офіційною версією Cucumber для мови програмування Java та інших JVM-мов, таких як Groovy, Scala, Clojure та ін. Cucumber-JVM повністю сумісний з Gherkin і дозволяє генерувати детальні звіти. Крім того, драйвер Cucumber-JVM можна кастомізувати.

Jbehave – один із перших та найпоширеніших BDD-фреймворків на сучасному ринку. Він розроблений Деном Нортом, якого іноді називають батьком BDD. Правда, у JBehave немає ключових особливостей Gherkin, таких як backgrounds, tags та doc strings. На додаток до цього, це перше рішення на чистій Java, яке існувало ще до появи Cucumber-JVM.[19]

Обидва вищеописані фреймворки використовуються досить широко. Тішить, що для них передбачені плагіни в більшості IDE, а установка можлива за допомогою Maven-пакетів. Порівняльна характеристика наведена в табл. 1.1.

Однак для мови Java є інші BDD-фреймворки:

– JGiven. Застосовує fluent API для написання сценаріїв, причому HTML звіти виводять сценарії разом із результатами. Крім того, радує лаконічний синтаксис.

– Spock і JDave. Це вже spec-фреймворки, щоправда, JDave протягом останніх років не розвивається.

– Scalatest. Призначений для Scala і також базується на spec-компонентах.

Таблиця 1.1 – Порівняльна характеристика фреймворків Cucumber та JBehave

Особливості	**Фреймворк Cucumber**	**Фреймворк JBehave**
Мова	Gherkin (англійська-подібна мова)	Business Readable DSL (англійська-подібна мова)
Формат файлів	.feature	.story
Ключові слова	Given, When, Then	Given, When, Then
Інтеграція	Підтримує різні мови програмування, такі як Java, Ruby, C#, і т.д.	Підтримує різні мови програмування,
Підтримка	Активно розвивається та має постійну підтримку оновленнями та виправленнями помилок.	Припинено активний розвиток та підтримку, але існують сторонні розширення та спільнота, яка надає підтримку.
Звітність	Забезпечує детальні звіти про виконання тестів, які є добре читабельними та структуризоваими	Забезпечує базові звіти, але потребує додаткових налаштувань та розширень для створення детальних звітів.
Використання	Широко використовується в індустрії та має значну популярність у спільноті тестувальників.	Використовується в деяких проектах та має свою спільноту користувачів.

Проаналізувавши всі вищеописані фреймворки, я зупинила свій вибір саме на Cucumber фреймворку, бо він, на мою думку, є найбільш універсальним та легким для розробки.

Щодо мови програмування, безсумнівно Java стала найкращим варіантом для розробки, адже саме Java є однією з основних мов програмування, яка використовується для створення Cucumber фреймворка і дозволяє легко інтегрувати Cucumber з іншими технологіями та інструментами. За допомогою Java-класів, аннотацій та конфігураційних файлів, можна підключити Cucumber до проекту і використовувати його для виконання тестів.

Використання Java для створення Cucumber фреймворка має декілька переваг:

– Гнучкість: Java надає багато можливостей для налаштування тестових сценаріїв і взаємодії з іншими компонентами проекту.

– Розширюваність: За допомогою Java можна розширити функціональність Cucumber шляхом розробки власних бібліотек та розширень.

– Багатомовність: Java підтримує багато мов програмування, що дозволяє розробникам писати тестові сценарії на різних мовах, залежно від їхніх вподобань та потреб проекту.

– Підтримка спільноти: Java є популярною мовою програмування з великою спільнотою розробників, що забезпечує доступ до великої кількості ресурсів, бібліотек і інструментів для розробки тестів [7-9].

Порівняльна характеристику найпопулярніших мов програмування, які підтримують Cucumber фреймворк, наведено в табл. 1.2.

Таблиця 1.2 – Порівняльна характеристика мов програмування

Мова	**Зрозумілість синтаксису**	**Легкість інтеграції**	**Розширені можливості**	**Популярність**
Java	Висока	Середня	Великі	Висока
JavaScript	Середня	Висока	Середні	Висока
Ruby	Висока	Висока	Великі	Середня
Python	Висока	Висока	Середні	Висока
C#	Середня	Середня	Середні	Висока

Отож, можна зробити висновок, що використання Java для створення Cucumber фреймворка є ефективним підходом до розробки автоматизованих тестів. Адже Java надає потужність та гнучкість для створення та управління тестовими сценаріями, а також інтеграцію з іншими компонентами проекту. Більше того, за допомогою Java можна розширити функціональність фреймворка і забезпечити стабільне та надійне виконання автоматизованих тестів.

1.4 Вибір технології розробки

Вибір правильної Java IDE є вирішальним кроком для успішної розробки програми. Розробка фреймворку для автоматизованого тестування вимагає потужного та функціонального IDE для забезпечення продуктивної роботи. IDE є основним інструментом розробника та впливає на швидкість розробки, якість коду, зручність відлагодження та багато інших аспектів процесу розробки. Правильна IDE допоможе розробникам працювати зі шляхом до класів, створювати файли, створювати аргументи командного рядка тощо.

На ринку існує велика різноманітність IDE, проте для розробки фреймворку я розгяладла для себе наступні популярні варіанти: IntelliJ IDEA, Eclipse та Visual Studio Code.

IntelliJ IDEA – це потужне інтегроване середовище розробки, створене компанєю JetBrains. Воно надає широку функціональність та багато інструментів, спрямованих на покращення продуктивності розробника:

– Підтримка різних мов програмування: IntelliJ IDEA працює з мовами програмування, такими як Java, Kotlin, Python, та іншими. Це важливий аспект зі сторони розробки фреймворку для автоматизованого тестування, так як дає можливість використовувати популярні мови програмування для створення скриптів.

– Інтеграція з фреймворками: IntelliJ IDEA забезпечує підтримку популярних фреймворків, таких як Cucumber, Selenium і JUnit. У ньому зручний

інтерфейс для налаштування та виконання тестів, а також функція автоматичного доповнення коду та рефакторинг для тестових сценаріїв.

– Зручний інтерфейс користувача: IntelliJ IDEA має інтуїтивно зрозумілий інтерфейс, тому це значно полегшує роботу з кодом. Пошук та навігація допомагають швидко знаходити та модифікувати код, що покращує продуктивність розробника.

IntelliJ IDEA має плагіни, написані користувачами. Середовище пропонує понад 900 плагінів, а також понад 50 плагінів у версії Enterprise. Більше того, користувачі можуть надсилати більше плагінів за допомогою вбудованих компонентів Swing. На рис. 1.1 зображений інтерфейс даного середовища.

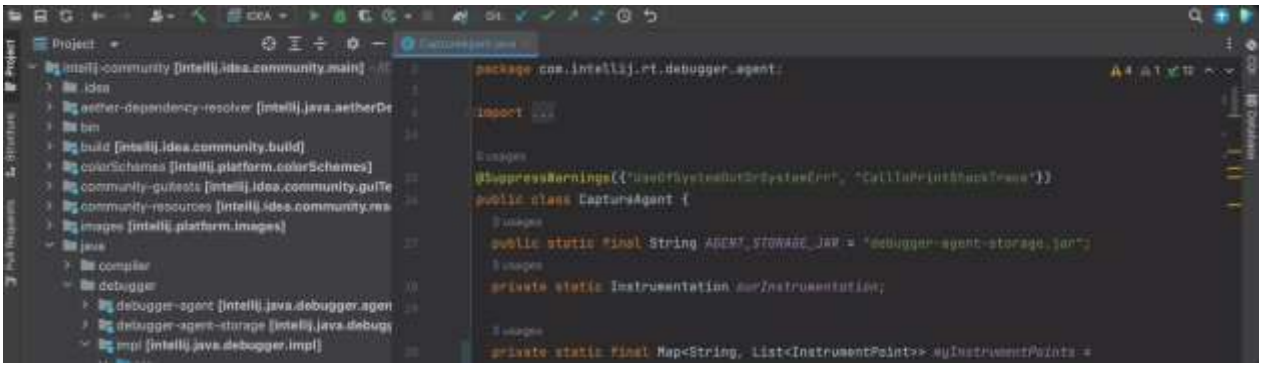


Рисунок 1.1 – Інтерфейс користувача середовища розробки IntelliJ IDEA

Порівняю IntelliJ IDEA з іншими популярними IDE, наприклад Eclipse та Visual Studio Code.

Eclipse – це інтегроване середовище розробки, яке також має підтримку розробки фреймворку для автоматизованого тестування. Велика колекція плагінів – це найбільша перевага цього IDE, адже робить його універсальним у використанні та легким для налаштувань. Можливості багатозадачності, фільтрації та налагодження Eclipse є сильними сторонами. Він був спеціально розроблений для задоволення потреб великих проектів розробки та виконує такі завдання, як аналіз і проектування, управління продуктами, впровадження, розробка вмісту, тестування та документування [11].

Проте, варто зазначити, що IntelliJ IDEA є більш функціональним та, на мою думку, має зручніші інструменти, такі як більш потужний редактор коду та вбудована підтримка фреймворків.

Visual Studio Code є легким та досить швидким інтегрованим середовищем розробки, яке також може використовуватись для розробки фреймворку для автоматизованого тестування. Однак, IntelliJ IDEA має більше розширених функцій та засобів, які полегшують розробку та тестування, такі як інтегрований відлагоджувач та підтримка різних мов програмування.

Після аналізу сильних сторін і особливостей усіх трьох IDE та порівняння їх із функціями та вимогами для розробки фреймворку для автоматизованого тестування, який буде розроблено, було вирішено вибрати IntelliJ IDEA як основне середовище розробки, використане в цьому проекті. Адже саме IntelliJ IDEA надає широкі можливості підтримки мов програмування, інтеграції з фреймворками та зручний інтерфейс користувача, що сприяє продуктивності та якості розробки тестового коду.

1.5 Вибір допоміжних інструментів розробки

Selenium WebDriver – це інструмент для автоматизації тестування веб-додатків. Він надає розширений набір функцій і можливостей для взаємодії з веб-браузерами. WebDriver дозволяє автоматизувати взаємодію з елементами сторінки, виконувати дії, такі як заповнення форм, клікання на кнопки, наведення курсора тощо [15].

WebDriver і Cucumber – це часто використовувана комбінація для автоматизованого тестування веб-сайтів. Selenium WebDriver і Cucumber працюють разом, дозволяючи розробникам автоматизувати тестові сценарії, описані з використанням Cucumber, за допомогою Selenium WebDriver. WebDriver взаємодіє з веб-браузерами, виконуючи дії, описані в тестових сценаріях, і перевіряє результати.

Використання Selenium WebDriver для тестування веб-сайту має декілька переваг. Воно дозволяє автоматизувати тестування функціональності веб-сайту, забезпечуючи швидке виконання тестів і зниження ризику помилок, пов'язаних з ручним тестуванням. Також, використання Selenium дозволяє проводити

повторне використання тестових сценаріїв і забезпечує широкі можливості взаємодії з різними веб-браузерами.

Архітектура Selenium WebDriver (рис. 1.2) складається з трьох компонентів: WebDriver, браузер-специфічнох драйверів і браузера [16]. Взаємодія між цими компонентами є двосторонньою, відправляючи HTTP-запити та отримуючи HTTP-відповіді у формі RESTful сервісів.

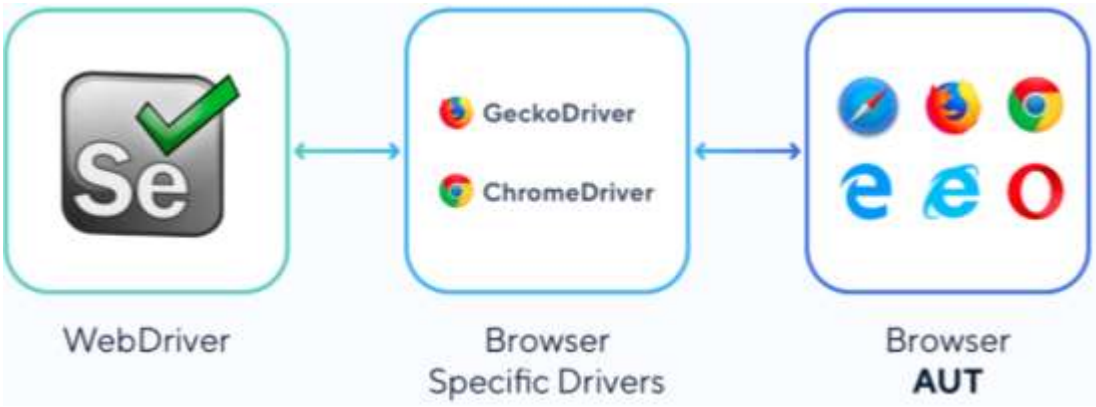


Рисунок 1.2 – Архітектура Selenium Webdriver

WebDriver містить мовоспецифічні клієнтські бібліотеки, так звані прив'язки мови, для написання автоматизованих скриптів і виконання інструкцій скрипта в браузері за допомогою драйверів. Ці прив'язки мови дозволяють Selenium підтримувати написання скриптів на багатьох різних мовах, таких як Java, C#, Python, Ruby і JavaScript. Стороння сторона може розширити Selenium, реалізувавши клієнтську бібліотеку для нової мови.

Браузер-специфічні драйвери відповідають за управління фактичним браузером, передаючи запити від WebDriver. Кожен драйвер специфічний для кожного браузера і зазвичай працює на тій самій системі, що й браузер. Драйвери можуть бути реалізовані і розширені сторонніми сторонами. GeckoDriver для Firefox і ChromeDriver для Chrome - це дві реалізації драйверів для цих браузерів.

Браузер є фактичним браузером, який отримує запити від компонента Drivers для взаємодії з додатком, що тестується (AUT). Selenium підтримує всі основні браузери, включаючи Chrome, Firefox, Internet Explorer, Opera і Safari.

WebDriver може працювати на іншій системі, ніж компоненти Drivers і Браузер. В такому випадку WebDriver спілкується віддалено з цими компонентами за допомогою Selenium Server або Remote WebDriver. Це дозволяє Selenium запускати автоматизовані тести на кількох браузерах і машинах одночасно.

Перевагами такої архітектури є те, що вона є досить простою та надійною. Вона має лише кілька компонентів, кожен з яких виконує відповідні завдання з написання та виконання автоматизованих тестів на AUT у браузері.

Архітектура є слабко зв'язаною. Три компоненти спілкуються за допомогою протоколу HTTP з RESTful веб-сервісами як формою комунікації. Ця характеристика дозволяє одному компоненту бути незалежним від іншого. Вдосконалення та зміни в одному компоненті можуть бути легко внесені без впливу на інші. Ця слабко зв'язана архітектура дозволяє Selenium бути платформо- та мовонезалежним [16].

Також Selenium надає розширені можливості для маніпуляції з елементами веб-сторінок і перевірки їх стану. Ось кілька головних методів Selenium:

- get(url): Цей метод відкриває вказану URL-адресу у браузері. Він завантажує веб-сторінку, яка відповідає вказаній URL-адресі.
- close(): Цей метод закриває поточне вікно браузера. Він закриває активне вікно браузера, з яким працює Selenium.
- quit(): Цей метод закриє всі вікна браузера і завершить сеанс драйвера.
- findElement(By locator): Цей метод знаходить перший елемент на веб-сторінці, який відповідає заданому локатору. Локатор може бути CSS селектором, XPath виразом або іншими методами пошуку.
- sendKeys(CharSequence... keysToSend): Цей метод вводить вказаний текст або послідовність клавіш в активний елемент на сторінці. Він дозволяє взаємодіяти з текстовими полями, паролями, формами тощо.

- click(): Цей метод натискає на елемент на сторінці. Він використовується для взаємодії з кнопками, посиланнями, чекбоксами та іншими елементами, які можна "клацнути".
- getText(): Цей метод повертає текстове значення елемента на сторінці. Він зазвичай використовується для отримання тексту з елементів, таких як заголовки, абзаци, мітки тощо.
- isDisplayed(): Цей метод перевіряє, чи відображається елемент на сторінці. Він повертає значення true, якщо елемент видимий, і false, якщо він прихований або відсутній.
- getAttribute(String attributeName): Цей метод повертає значення атрибуту вказаного елемента. Атрибути можуть бути, наприклад, значенням атрибуту "href" у посиланні або "value" у текстовому полі.

Загалом, використання Selenium WebDriver з Cucumber дозволяє розробникам і тестувальникам швидко і ефективно автоматизувати тестування веб-сайтів, забезпечуючи надійність і зручність у взаємодії з різними зацікавленими сторонами.

1.6 Висновок до першого розділу

Таким чином, забезпечення якості включає процеси та процедури, які систематично контролюють різні аспекти послуги або об’єкта. За допомогою аудитів та інших форм оцінювання зусилля із забезпечення якості виявляють і усувають проблеми або невідповідності, які виходять за межі встановлених стандартів або вимог. Іншими словами: забезпечення якості забезпечує високий рівень якості при розробці продуктів або послуг.

Моделювання архітектури програмного забезпечення є важливим етапом у розробці програмного продукту. Воно допомагає визначити структуру системи, забезпечити її функціональність та якість. Використання фреймворку Cucumber та нотації Gherkin у процесі моделювання архітектури дозволяє зрозуміло та чітко описати компоненти та їх взаємодію, що сприяє ефективному розробленню та тестуванню програмного забезпечення.

Використання фреймворків, звичайно, не є обов'язковим. Але це може зробити розробку набагато легшою та швидшою, а також покращити якість і стабільність коду. Тому абсолютна більшість розробників активно використовують їх у своїй роботі.

Фреймворки пропонують готові рішення типових проблем і завдань. Це дозволяє вам зосередитися на створенні більших завдань під час розробки додатків і веб-сайтів. Фреймворки також допомагають покращити командну співпрацю та спростити підтримку та розробку проекту.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО РІШЕННЯ

2.1 Проєктування фреймворка Cucumber та його структура

Я розробила Maven Cucumber фреймворк, який базується на POM (Page Object Model) з Page Factory (додаток А). Мій фреймворк є гібридним і підтримує як BDD, так і DDT. Він розроблений з використанням мови програмування Java та підтримує тестування користувацького інтерфейсу, бази даних та API.

Тепер детальніше про структуру мого фреймворку:

– Maven. Спочатку у нас є проект Maven, який допомагає нам будувати структуру проекту, керувати залежностями проекту та виконувати автоматизовані скрипти. У файлі pom ми додаємо всі необхідні залежності (Selenium, Cucumber, JUnit, JDBC, Rest Assured) для проекту (див. лістинг 2.1).

Лістинг 2.1 – Добавляння Cucumber залежності

```
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>6.9.1</version>
</dependency>
```

– Test Runner. Це початкова точка виконання нашого фреймворку. У класі runner ми вказуємо конфігурації наших тестів з використанням параметрів Cucumber (див. лістинг 2.2). У нас є окремі класи runner для smoke, regression та API тестів на основі JUnit.

Лістинг 2.2 – Runner клас

```
@RunWith(Cucumber.class)
@CucumberOptions(
        features = "src/test/resources/features/",
        glue = "steps",
        dryRun = false,
        tags = "@sprint3",
        plugin = {"pretty", "html:target/cucumber.html",
"json:target/cucumber.json",
```

```
        "rerun:target/failed.txt"}
)
public class Runner {
}
```

– Common methods (Додаток Д). Тут зберігаються всі спільні методи, які використовуються в межах всього проекту декілька разів.

– Cucumber Hooks (Додаток Л). З використанням анотацій @Before та @After ми викликаємо метод налаштування в нашому базовому класі, який ініціалізує наш драйвер і викликає метод для ініціалізації всіх елементів сторінки. Також, за допомогою хука @After, ми робимо знімки екрана після кожного неуспішного сценарію. Приклад використання хуків подано в лістингу 2.3.

Лістинг 2.3 – Hook клас

```
    @Before
    public void preCondition(){
        openBrowserAndLaunchApplication();
    }
    //here we use special class called scenario class from
cucumber
    //this class holds the complete information of your execution

    @After
    public void postCondition(Scenario scenario){
        byte[] pic;
        if(scenario.isFailed()){
            //failed screenshot will be available inside failed
folder
           pic =  takeScreenshot("failed/" + scenario.getName());
        }else {
            pic = takeScreenshot("passed/" + scenario.getName());
        }
        //to attach the screenshot in our report
        scenario.attach(pic, "image/png", scenario.getName());
        closeBrowser();
    }
}
```

– Base Class. У цьому класі ми отримуємо значення браузера та URL та за допомогою цих значень ініціалізуємо WebDriver та переходимо на конкретні URL (див. лістинг 2.4).

Лістинг 2.4 – Метод для запуску браузеру та навігації до певної url.

```
    public static void setUp(String url){
            System.setProperty("webdriver.chrome.driver",
"driver/chromedriver");
            driver = new ChromeDriver();
            driver.get(url);
            driver.manage().window().maximize();
            driver.manage().timeouts().implicitlyWait(20,
TimeUnit.SECONDS);
        }
```

– Pages (Додаток К). Розроблений фреймворк побудований за підходом POM для усунення дублювання коду та покращення повторного використання коду. У цьому класі ми зберігаємо інформацію про сторінки та їх елементи. Ми використовуємо анотацію @FindBy для локалізації елемента та PageFactory для ініціалізації всіх елементів (рис. 2.1).

Рисунок 2.1 – Локалізація елементів

– Файл Feature (Додаток Б). Тут ми пишемо наші тестові сценарії для перевірки UI, бази даних та API. Наші тести написані з використанням

синтаксису Gherkin, використовуючи Given, When, Then разом з ключовими словами Background, Scenario та Scenario Outline.

– Step Definitions (Додаток Л): В цьому класі ми фактично реалізовуємо наші автоматизовані скрипти та логіку з використанням Selenium та мови Java і виконуємо перевірки за допомогою перевірок JUnit.[21]

Ця структура Cucumber надає гнучкість та чіткість при написанні тестових сценаріїв, розробці та виконанні автоматизованих тестів програмного забезпечення. Вона сприяє спільному розумінню вимог та полегшує співпрацю між розробниками, тестувальниками та зацікавленими сторонами проекту.

2.2 Реалізація інтерфейсу на основі фреймворка Cucumber та нотації Gherkin

Cucumber – це фреймворк, який реалізує підхід BDD. Відповідно, для написання тестів Cucumber використовується нотація Gherkin, яка визначає структуру тесту та набір ключових слів:

Тест записується у файл із розширенням .feature і може містити один або декілька сценаріїв. Приклад такого файлу зображений на рис. 2.2.

Ключові слова допомагають структурувати та описати сценарії тестування у файлі .feature, забезпечуючи зрозумілість та читабельність для всіх учасників проєкту. Отож детальніше розглянемо кожне з основних ключових слів:

– Feature / Функціонал. Ключове слово "Feature" використовується для визначення функціональності або функцій, які потрібно перевірити у тестовому файлі. Історія повинна бути описана одним реченням.

– Scenario / Сценарій. Ключове слово "Scenario" використовується для визначення окремого сценарію тестування. Воно описує одну конкретну дію або перевірку в рамках функціональності.

– Given, When, Then/ Припустимо, Коли, Тоді. Ключові слова "Given", "When" і "Then" використовуються для створення кроків сценарію.

"Given" вказує початковий стан або передумови, "When" описує дію, що відбувається, а "Then" вказує очікуваний результат або перевірку.[17]

– And, But / І, Але. Ключові слова "And" і "But" використовуються для продовження попереднього кроку або для додавання додаткових кроків в сценарій. Вони допомагають зберігати структуру та читабельність сценарію.

```
Feature:  Adding employees

  Background:
    And  user is logged in with valid admin credentials
    When user clicks on PIM option
    And user clicks on Add employee button

  @smoke
  Scenario: Adding employee from add employee page
    And user enters firstname middlename and lastname
    And user clicks on save button option
    Then employee added successfully

  @smoke
  Scenario: Adding employee from add employee page via feature file
    And user enters firstname "Yarina123" middlename "MS" and lastname "Yarina456"
    And user clicks on save button option
    Then employee added successfully

  @example
  Scenario Outline: Adding employee from add employee page via feature file
    And user enters "<FirstName>" "<MiddleName>" and "<LastName>" in the application
    And user clicks on save button option
    Then employee added successfully

  Examples:
  |FirstName|MiddleName|LastName|
  |Test123456|MS       |Test9876|
  |Test228   |MS       |Test7849|
  |Test12345 |MS       |Test3476|
  |Test785   |MS       |Test7669|
```

Рисунок 2.2 – Приклад feature файлу

Як бачимо, перші три пункти належать до стандартної нотації Gherkin. У свою чергу Background та Scenario Outline додані вже у Cucumber. Розглянемо їх докладніше:

– Background/ Передісторія. Часто виходить, що ті самі Given кроки повторюються у всіх сценаріях у рамках feature-файлу. І те, що вони повторюються в кожному сценарії, говорить про те, що вони не такі принципові щодо конкретного сценарію. Тому такі Given кроки можна винести окремо та згрупувати у секції Background (див. лістинг 2.5).

Характеристики Background секції:

– може містити один або кілька етепів Given;
– виконується перед кожним сценарієм;
– тільки одна в рамках одного .feature файлу.

Лістинг 2.5 – Приклад використання Background:

```
Background:
                When user enters valid username and valid
            password
                And user clicks on login button
                Then user is successfully logged in
                When user clicks on PIM option
                And user clicks on EmployeeList option
```

Отож, основна функція ключового слова Background є те, що це дозволяє уникнути повторення кроків в кожному сценарії.

– Scenario Outline/ Сценарій-аутлайн. Ключове слово "Scenario Outline" використовується для виконання того самого сценарію кілька разів, але з різними комбінаціями параметрів (див. лістинг 2.6).

– Examples / Приклади. Ключове слово "Examples" використовується для визначення набору вхідних даних для сценарію шаблону у вигляді таблиці. Він містить значення параметрів, які використовуються в сценарії шаблону (див. лістинг 2.6).

Лістинг 2.6 – застосування Scenario Outline з Examples

```
    @errorvalidation
           Scenario Outline: Login with multiple username and
password combinations
             When user enters different "<usernamevalue>" and
"<passwordvalue>" and verify the "<error>" for all the
combinations

             Examples:
             |usernamevalue|passwordvalue|error                  |
             |Admin        |123!   |Invalid credentials      |
             |abd77        |321    |Invalid credentials      |
             |James        |128    |Password cannot be empty|
             |             |345    |Username cannot be empty|
```

Застосування цих ключових слів Cucumber сприяє полегшенню спілкування між учасниками проекту, забезпечує чітку структуру тестових сценаріїв та дозволяє легко підтримувати та розширювати автоматизоване тестування програмного забезпечення.

2.3 Процес та етапи тестування

Запуск тестів відбувається саме з Test Runner класу. Це можна зробити за допомогою інтегрованих засобів розробки, командного рядка або засобів інтеграції з системами збірки, такими як Jenkins.

Варто детальніше описати опції Cucumber, які ми використовуємо у класі Runner для запуску тестів:

– features – шлях до папки з файлами .feature. Фреймворк шукатиме файли в цій та у всіх дочірніх папках. Можна вказати кілька папок, наприклад: features = {src/test/resources/features};

– glue – пакет, у якому знаходяться класи з реалізацією кроків та «хуків».

– tags – фільтр тестів, що запускаються за тегами. Список тегів можна перерахувати через кому. Символ ~ виключає тест зі списку тестів, що запускаються, наприклад ~@fail;

– dryRun – якщо true, то відразу після запуску тесту фреймворк перевіряє, чи всі кроки тесту розроблені, якщо ні, видає попередження. При false попередження видаватиметься після досягнення нерозробленого кроку. За промовчанням false [18].

Спочатку відбувається інтерпретація файлів .feature: Cucumber інтерпретує файл .feature та знаходить описані в ньому сценарії тестування. Він аналізує кожен крок сценарію та знаходить відповідний метод-виконавець для кожного кроку.

Після Cucumber викликає методи-виконавці для кожного кроку сценарію. Ці методи містять код, який взаємодіє з програмним забезпеченням, виконує дії та перевіряє результати.

Після виконання кожного сценарію Cucumber генерує детальну звітність про результати тестування та зберігає ці звіти у папці target. Приклад звіту зображений на рис. 2.3.

	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
1st feature	10	0	0	0	0	10	1	0	1	1m 39s 263ms	Passed
Second feature	5	1	2	1	2	11	1	1	2	092ms	Failed
2	15	1	2	1	2	21	2	1	3	1m 39s 355ms	
	71.43%	4.76%	9.52%	4.76%	9.52%		66.67%	33.33%			50.00%

Рисунок 2.3 – Звіт Cucumber після виконання тестів

Звіт містить інформацію про пройдені та непройдені кроки, помилки, логи, тривалість виконання тестів тощо. Це сприяє швидкому виявленню проблем та полегшує комунікацію в команді розробників із зацікавленими сторонами [22].

2.4 Тестування API

Відсутність елементів UI-інтерфейсу з незвички може ввести QA-фахівця у ступор: немає ні кнопок, ні полів, ні зрозумілого формату звернення до сервісів. Полегшують взаємодію з API спеціальні інструменти. Найбільш популярні серед них - SoapUI та Postman.

SoapUI. Програма з відкритим вихідним кодом для тестування Soap та Rest API. Він повністю побудований на платформі Java і використовує Swing для інтерфейсу користувача (тобто SoapUI - кросплатформовий). Його функціональні можливості включають перевірку веб-служби, запуск, розробку,

моделювання та макетування, функціональне тестування, тестування навантаження та відповідності.

SoapUI може тестувати веб-сервіси SOAP та REST, JMS, AMF, а також виконувати будь-які виклики HTTP(S) та JDBC. Для написання автоматизованих скриптів використовується мова Groovy.

Postman. Як кажуть його творці, це свого роду швейцарський ніж, який дозволяє формувати та виконувати запити, документувати та моніторити сервіси в одному місці. Тестувальники мають змогу створювати тести та автоматизовано тестувати їх прямо з Postman. На рис. 2.4 зображений приклад створення POST запиту для створення нового користувача

Рисунок 2.4 – Приклад створення HTTP запиту

Валідація тест кейсів з Postman - це процес перевірки правильності виконання HTTP-запитів та отримання очікуваних відповідей на основі заздалегідь визначених критеріїв. Postman є потужним інструментом для тестування API, який надає можливість валідувати різні аспекти тестових кейсів, такі як:

– Перевірка коду статусу відповіді: Можна перевірити, чи повертає сервер очікуваний HTTP-код статусу, наприклад 200 (OK) або 404 (Не знайдено).

– Перевірка вмісту відповіді: Ви можете перевірити наявність, формат та правильність даних у відповіді. Наприклад, перевірка наявності конкретного поля або значення в JSON-об'єкті.

– Перевірка часу відповіді: Також можна валідувати час, необхідний для отримання відповіді від сервера. Це може бути корисним для виявлення швидкісних або повільних запитів.

– Перевірка заголовків: Можна перевірити наявність та значення конкретних заголовків у відповіді.

– Перевірка схеми відповіді: Використовуючи схеми JSON або XML, можна перевірити, чи відповідає структура відповіді заданим вимогам.

Postman надає багато можливостей для валідації тест кейсів, включаючи використання спеціальних бібліотек, таких як Chai або JSON Schema, для створення складних перевірок. Також можна налаштувати автоматичну валідацію тест кейсів під час запуску колекції запитів у Postman Runner або з використанням Postman API.

Загалом, валідація тест кейсів з Postman дозволяє перевіряти правильність виконання HTTP-запитів і забезпечувати якість вашого API [24].

Для автоматизації REST API я використовувала Rest Assured бібліотеку. Rest Assured – це потужна бібліотека для автоматизованого тестування API, і її досить легко інтегрувати з Cucumber фреймворком. Використання Rest Assured у Cucumber фреймворку включає наступні кроки:

– Спочатку необхідно додати залежність до бібліотеки Rest Assured у файлі pom.xml (див. лістинг 3.1).

Лістинг 3.1 – Додавання Rest Assured залежності

```
</dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>4.5.1</version>
        <scope>test</scope>
    </dependency>
```

– Потім для кожного кроку сценарію, який стосується API (Додаток В), створюю відповідний клас виконавець. У цьому класі використовуються

методи Rest Assured для взаємодії з API, надсилання запитів, отримання відповідей та перевірки результатів.

– Налаштування базового URL: У класі виконавця встановлюю базовий URL API, з яким буду взаємодіяти. Наприклад:

RestAssured.baseURI = "https://api.example.com";

– Використання Rest Assured методів: given(), when(), then(), для налаштування запиту, виконання запиту та перевірки відповіді API (див. лістинг 3.2).

Лістинг 3.2 – Приклад використання Rest Assured методів

```
        RequestSpecification preparedRequest =
given().header("Authorization", token)
                .header("Content-Type",
"application/json").queryParam("employee_id", "2033677833");
    Response response =
preparedRequest.when().get("/getOneEmployee.php");
response.then().assertThat().statusCode(201);
```

– Для інтеграції з Cucumber сценаріями використовуються ключові слова Cucumber, такі як Given, When, Then, And, для описування сценаріїв тестування API. Для реалізації логіки кожного кроку сценарію є відповідний API клас виконавець (Додаток Н).

– Запуск тестів здіюснюється за допомогою спеціального API Runner класу і Rest Assured буде виконувати взаємодію з API та перевіряти результати на відповідність очікуванням [23].

Використання RestAssured спрощує процес написання тестових сценаріїв, забезпечує зручний та ефективний спосіб виконання HTTP-запитів та перевірки відповідей сервера. Завдяки його читабельному DSL-синтаксису та вбудованим можливостям перевірки, таким як перевірка коду статусу, вмісту відповіді та валідація JSON-структур, RestAssured дозволяє зручно та швидко створювати потужні API тестові сценарії. Використання RestAssured сприяє покращенню надійності та якості API, спрощує процес автоматизації, забезпечує швидкий зворотний зв'язок та покращує загальну продуктивність розробників.

2.5 Практичне застосування розробленого інструменту

Використала я розроблений фреймворк для автоматизованого тестування HRMS (Human Resource Management System) веб-сайту, який базується на управлінні людськими ресурсами в організації (рис. 2.5).

HRMS надає засоби для автоматизації і управління багатьма аспектами діяльності відділу кадрів, включаючи управління персоналом, заробітною платою, наймом, професійним розвитком, відпустками, обліком робочого часу, адмініструванням працівників та багато іншого.

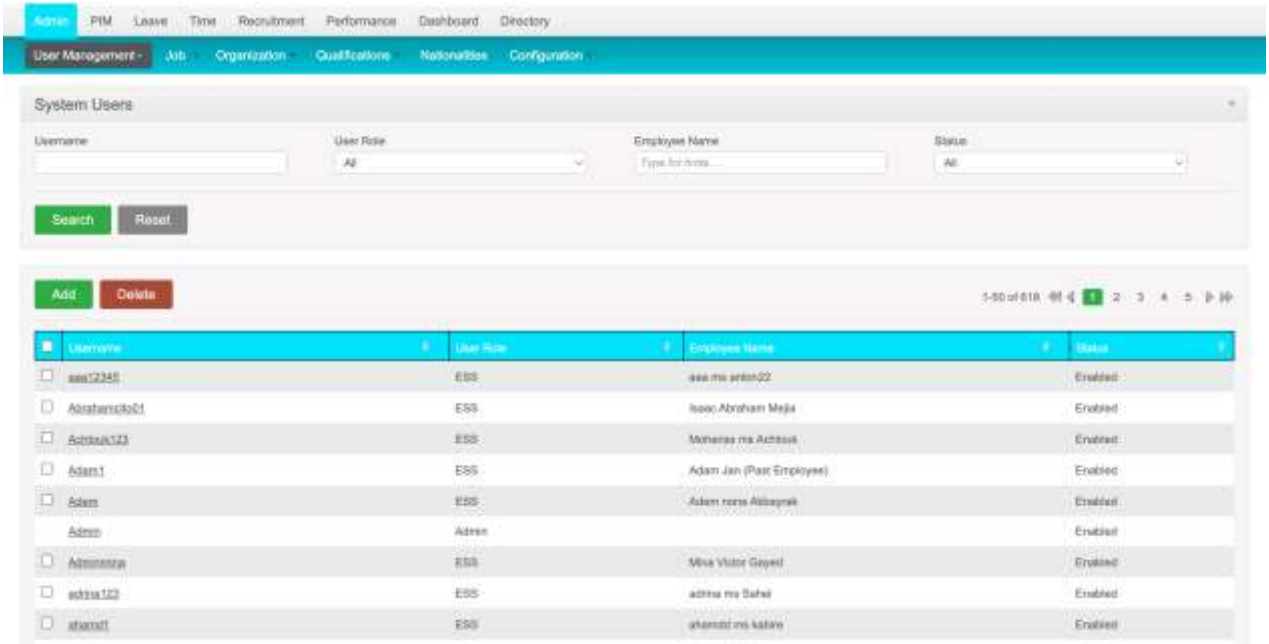


Рисунок 2.5 – Інтерфейс веб-сайту для тестування

Основні компоненти та функції HRMS веб-сайту, включають:

– Управління персоналом: Цей модуль дозволяє зберігати та керувати інформацією про персональні дані працівників, таку, контактна інформація, історія зайнятості, кваліфікація та навички.

– Заробітна плата: Цей модуль дозволяє обробляти заробітну плату, а також включати функції, такі як розрахунок заробітної плати, оподаткування, автоматичні виплати.

– Найм та підбір персоналу: Цей модуль дозволяє автоматизувати процес найму нових працівників, включаючи розміщення вакансій, прийом

заявок, відбір кандидатів, проведення співбесід та забезпечення інтеграції нових співробітників.

– Облік робочого часу та відпусток: Цей модуль дозволяє вести облік робочого часу працівників, включаючи присутність, відсутність, робочі години, відпустки та відпрацьований час.

– Продуктивність: HRMS-системи надають можливість генерувати різноманітні звіти та проводити аналітику щодо продуктивності кожного робітника [25].

За допомогою розробленого мною фреймворку ми маємо змогу протестувати основні області вищезазначеного веб-сайту. Наприклад, одним із сценаріїв може бути перевірка входу у систему із введенням правильних та неправильних логіну та паролю, таким чином ми здійснимо позитивне та негативне тестування функціональності. Також можливі сценарії можуть бути: тестування функції додавання нового співробітника за іменем чи ID, видалення вже наявного співробітника із системи, а також пошук певного співробітника серед списку.

2.6 Висновок до другого розділу

Архітектура розробленого фреймворка базується на складних структурах папок і файлів, таких як фічі, степи, хуки, допоміжні класи тощо. Ця структура дозволяє організовувати тестові сценарії в логічну і чітку ієрархію, що полегшує розуміння та підтримку тестів.

Тестування API є невід'ємною складовою частиною розробки програмного забезпечення. Було визначено, що тестування API дозволяє перевіряти правильність взаємодії з сервером, перевіряти правильність відповідей та забезпечувати стабільність та надійність API. Використання фреймворка Cucumber у поєднанні з інструментом RestAssured надає зручний та ефективний спосіб написання тестів для перевірки API. Це дозволяє забезпечити високу якість програмного забезпечення та знижує кількість помилок у роботі системи.

Використання розробленого інструменту, який базується на фреймворку Cucumber та нотації Gherkin, має багато переваг. Цей інструмент дозволяє автоматизувати тестування програмного забезпечення, полегшує процес розробки тестових сценаріїв та забезпечує зручний зворотний зв'язок для команди розробників. Використання такого інструменту сприяє покращенню якості програмного забезпечення, збільшує ефективність розробки та допомагає вчасно виявляти та виправляти помилки.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Соціальні та психологічні фактори ризику

Небезпеки, які поширені в суспільстві і загрожують життю і здоров'ю людей, називають соціально-психологічними. Носіями соціальної небезпеки є люди, які утворюють певні соціальні групи. Особливістю суспільних небезпек є те, що вони загрожують великій кількості людей. Поширення суспільної небезпеки визначається особливостями поведінки людей та окремих соціальних груп.

Соціальні небезпеки дуже численні: форми насильства, вживання речовин, що викликають психологічні та фізіологічні розлади, врівноваженість людини (алкоголь, наркотики, куріння), зрада, самогубство та інші дії, які можуть завдати шкоди здоров'ю людини.

Соціальні та психологічні небезпеки можна класифікувати за певними характеристиками.

За походженням можна виділити наступні групи ризику:

– небезпеки, пов’язані з психічним впливом на людину (шантаж, шахрайство, крадіжка);

– небезпеки, пов’язані з фізичним насильством (грабіж, бандитизм, терор, зґвалтування, захоплення заручників);

– небезпеки, пов’язані з вживанням речовин, що руйнують організм (наркоманія, алкоголізм, куріння);

– небезпеки, пов’язані з хворобами (СНІД, ЗПСШ);

– небезпека самогубства.

Залежно від масштабів подій соціальні небезпеки можна розділити на:

– місцеві;

– регіональні;

– глобальні.

Причини соціальної небезпеки.

В основному соціальні небезпеки породжені соціально-економічними процесами в суспільстві. При цьому необхідно враховувати неузгодженість причин, наслідки яких мають соціальний характер.

Недосконаліть людської природи є головною умовою виникнення соціальної небезпеки.

Профілактика та захист від соціальних небезпек.

Елементом захисту від суспільних небезпек є також залучення громадськості та підвищення інформованості населення про можливість виникнення суспільних небезпек та можливі наслідки їх виникнення.

Адаптація — динамічний процес, завдяки якому в мінливому зовнішньому середовищі підтримується сталість внутрішнього середовища в організмі [26].

Характеристики адаптації:

– розширення фізіологічних здібностей;

– підвищення фізіологічної стійкості організму до зовнішніх впливів;

–підвищення працездатності шляхом зміни порогів чутливості аналізаторів;

– підвищення стійкості фізіологічних систем;

– перехід фізіологічних систем на вищі функціональні рівні;

– розширення фізіологічних резервів;

– мобілізація енергетичних ресурсів і сил оборони.

Психічна адаптація – це процес створення оптимальної відповідності між особистістю і середовищем, які сприяють задоволенню актуальних потреб і досягненню значущих цілей, за умови, що фізичне та психічне здоров'я в нормі.

3.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Основними вимогами безпеки праці при роботі з ПК є:

– Огляд та приведення в порядок робочого місця; Переконайтеся, що на ньому немає сторонніх предметів. Усі пристрої та блоки ПК підключаються до системного блоку з’єднувальними кабелями.

– Перевірка надійності встановлення пристроїв на робочому столі.

Монітор не повинен стояти на краю столу. Поверніть монітор так, щоб вам було зручно дивитися на екран - під прямим кутом (не набік) і трохи догори ногами; при цьому екран повинен бути трохи нахилений - нижній край знаходиться ближче до користувача.

– Перевірка загального стану пристрою, справність електропроводки, з'єднувальних кабелів, вилок, розеток і заземлення захисного екрану.

– Вставте вилку в розетку та переконайтеся, що вона надійно закріплена. Забороняється вставляти і виймати вилку мокрими руками.

– Відрегулювати та зафіксувати висоту крісла та зручний кут нахилу спинки.

– При необхідності підключити необхідні пристрої (принтер, сканер) до комп’ютера. Всі кабелі, що з'єднують системний блок з іншими пристроями, повинні включатися і відключатися тільки при вимкненому комп'ютері.

– Відрегулюйте яскравість світла та контрастність монітора.

Вимоги безпеки під час роботи з ПК:

− Стабільно встановлюйте клавіатуру на столі, запобігайте її тремтінню, забезпечуючи можливість обертання та переміщення.

– Якщо конструкція клавіатури не передбачає місця для опори долонями, клавіатуру розміщують на відстані не менше 100 мм від краю столу в оптимальній зоні панелі монітора.

− Під час роботи на клавіатурі сидіть прямо і не перенапружуйтеся.

– Зменшити несприятливе навантаження на користувача при роботі з комп'ютерною мишею (вимушена поза, необхідність постійно контролювати якість дій), забезпечити велику вільну поверхню столу для переміщення комп'ютерної миші та зручний упор для ліктьового суглоба.

– При вимкненому комп'ютері періодично видаляйте пил з поверхонь пристрою спеціальними серветками.

Під час роботи з ПК заборонено:

– Самостійно розбирати та ремонтувати системний блок (корпус ноутбука), монітор, клавіатуру, комп’ютерну мишу.

– Вставляти сторонні предмети у вентиляційні отвори ПК, ноутбука або монітора.

– Не ставити металеві предмети та ємності з водою (вази, горщики, склянки) на системний блок ПК та периферійні пристрої, оскільки потрапляння води в центр пристрою може спричинити пожежу або ураження електричним струмом.

– Тривалість безперервної роботи за ПК не повинна перевищувати 2 години. Після цього потрібно зробити 15-хвилинну перерву.

– При зоровому дискомфорті або інших неприємних відчуттях необхідна невелика перерва.

– З метою зниження нервово-емоційного напруження та втоми зорового аналізатора, покращення мозкового кровообігу, подолання негативних наслідків гіподинамії та профілактики втоми комплекс вправ доцільно виконувати в кількох перервах.

Вимоги безпеки після роботи:

– зберегти інформацію;

– вимкнути ПК, монітор або ноутбук;

– вимкнути стабілізатор, якщо комп'ютер підключений до мережі через нього;

– прибрати робоче місце [27].

3.3 Соціальне значення охорони праці

Соціальне значення охорони праці полягає у сприянні підвищенню ефективності суспільного виробництва шляхом постійного поліпшення і поліпшення умов праці, підвищення їх безпеки, зниження аварійності на виробництві та професійної захворюваності.

Соціальне значення охорони праці виявляється у зростанні продуктивності праці, збереженні трудових ресурсів і збільшенні сукупного національного продукту.

Підвищення продуктивності праці відбувається за рахунок збільшення фонду робочого часу за рахунок скорочення простоїв у зміні за рахунок

усунення або зменшення мікротравматизму, а також попередження передчасної втоми шляхом раціоналізації та поліпшення умов праці та впровадження оптимального режиму роботи та відпочинку та ін. заходи, що сприяють більш ефективному використанню робочого часу.

Збереження трудових ресурсів і підвищення професійної активності робочої сили можна пояснити поліпшенням здоров'я і збільшенням середньої тривалості життя за рахунок поліпшення умов праці, що пов'язано з високою трудовою активністю і збільшенням виробничого стажу. З підвищенням кваліфікації та навичок підвищується і професійний рівень.

Збільшення сукупного національного продукту відбувається за рахунок покращення перерахованих вище показників та їх складових [28].

Плинність робочої сили завдає значних збитків компаніям, оскільки ті, хто звільняється, працюють із зниженою продуктивністю протягом певного часу.

3.4 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи розглядається питання безпеки життєдіяльності, яке полягає в тому, як соціально-психологічні фактори ризику впливають на суспільство. Загалом небезпека від цих факторів ризику досить висока. Особливістю суспільних небезпек є те, що вони загрожують великій кількості людей. Поширення суспільної небезпеки визначається особливостями поведінки людей та окремих соціальних груп.

Що стосується охорони праці, то розглядалися два питання: по-перше, дотримання правил користування ПК перед початком роботи, під час роботи з ПК, після закінчення роботи, по-друге, суспільне значення охорони праці, що виражається у зростанні продуктивність праці, збереження трудових ресурсів і збільшення сукупного національного продукту.

ВИСНОВКИ

У рамках кваліфікаційної роботи успішно розроблено та налаштовано інструментальний набір для автоматизованого тестування програмного забезпечення на основі фреймворку Cucumber та нотації Gherkin. Цей набір включає в себе необхідні компоненти, які дозволяють створювати, виконувати та аналізувати тестові сценарії, забезпечуючи зручний інтерфейс для розробників та тестерів. Більше того, використання фреймворку Cucumber та нотації Gherkin дозволяє створювати зрозумілі та легко зчитувані тестові скрипти, що сприяє залученню різних зацікавлених сторін у процес тестування.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр» проведений детальний аналіз відомих засобів та методів розробки для автоматизованого тестування програмного забезпечення. Розглянуті різні інструменти, фреймворки та підходи, спрямовані на поліпшення ефективності та надійності тестування. Цей аналіз дав змогу виокремити фреймворк Cucumber та нотацію Gherkin як потужні та гнучкі інструменти для автоматизованого тестування.

В другому розділі кваліфікаційної роботи представлені результати проєктування та здійснено розробку. Також продемонстровано практичне застосування розробленого інструментального набору шляхом створення та виконання тестових сценаріїв для реального програмного забезпечення. Це практичне застосування підтвердило ефективність та користь розробленого набору і підкреслило його значення для автоматизованого тестування програмного забезпечення.

У розділі «Безпека життєдіяльності, основи хорони праці» розглядаються соціальні та психологічні фактори ризику для суспільства в цілому та визначаються фактори з різними наслідками. Щодо дотримання правил користування ПК враховано основні правила користування ПК. Враховано суспільне значення охорони праці, що сприяє підвищенню ефективності соціальних удосконалень і поліпшенню умов праці та підвищенню їх безпеки.

Отже, в результаті дипломної роботи було проведено аналіз, розробку та налаштування інструментального набору для автоматизованого тестування програмного забезпечення. Здобуто гарний практичний досвід використання сучасних методів проектування програмного забезпечення та інструментів для розробки інструментарію для автоматизованого тестування програмного забезпечення на основі фреймворку Cucumber та нотації Gherkin. Продемонстроване практичне застосування набору підтвердило його ефективність та важливість для вдосконалення процесу тестування. Результати дипломної роботи відкривають нові перспективи в галузі автоматизованого тестування та можуть бути використані для поліпшення якості та надійності програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ

1. Ушакова І. О. Підходи до забезпечення якості програмного забезпечення / І. О. Ушакова // Сучасні інформаційні технології і системи : монографія / за заг. ред . В. С. Пономаренка. - Харків : «Стильіздат», 2021. – С. 125-140.

2. Difference Between Quality Assurance And Quality Control (QA Vs QC) [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/quality-assurance-vs-quality-control/.

3. What Is the International Organization for Standardization (ISO)? [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.investopedia.com/terms/i/international-organization-for-standardization-iso.asp.

4. Cucumber for BDD—How Effective Is It? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.stickyminds.com/article/cucumber-bdd-how-effective-it .

5. Behaviour-Driven Development [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://cucumber.io/docs/bdd/.

6. Top 5 Cucumber Best Practices For Selenium Automation [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.lambdatest.com/blog/cucumber-best-practices/.

7. Comparison of 10 Programming Languages. [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://reubenrochesingh.medium.com/comparison-of-10-programming-languages-f43b0ac337a4.

8. 5 reasons why Java is still the best programming language [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.theserverside.com/feature/5-reasons-why-Java-is-still-the-best-programming-language.

9. Why Java is the best Programming language to Learn Coding? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу:

https://medium.com/javarevisited/why-java-is-the-best-programming-language-to-learn-coding-for-beginners-cba79aed1271.

10. Голуб Б.М. C#. Концепція та синтаксис: навчальний посібник / Голуб Б.М.. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2019. – 136с.

11. IntelliJ vs Eclipse: Which is better for beginners? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.mygreatlearning.com/blog/intellij-vs-eclipse/.

12. Архітектура та проектування програмного забезпечення [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://cutt.ly/6JRclCV .

13. Архітектура програмного забезпечення [Електронний ресурс]. – 2021 – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya.

14. Software Architecture Guide [Електронний ресурс]. – 2019 – Режим доступу до ресурсу: https://martinfowler.com/architecture/.

15. Analysis and Design of Selenium WebDriver Automation Testing Framework [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S1877050915005396.

16. A Deep Dive into Selenium, Its Alternative Solution for 2022 and Beyond [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://katalon.com/resources-center/blog/selenium-alternative-solution

17. Gherkin Reference [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://cucumber.io/docs/gherkin/reference/.

18. Testing with Cucumber [Електронний ресурс]. – 2021 – Режим доступу до ресурсу: https://www.ibm.com/docs/en/rft/10.0.0.0?topic=perspective-testing-cucumber.

19. Jbehave-how-to-get-started? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://blog.knoldus.com/jbehave-how-to-get-started/.

20. Top 11 API Testing Tools For 2023: A Comprehensive Guide [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.telerik.com/blogs/top-11-api-testing-tools-comprehensive-guide .

21. Cucumber Framework Folder Structure [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.programsbuzz.com/article/cucumber-framework-folder-structure.

22. Reporting [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://cucumber.io/docs/cucumber/reporting/?lang=java.

23. API Test Automation with Rest Assured [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://blog.clairvoyantsoft.com/api-test-automation-with-rest-assured-54d6d5a470b4.

24. How To Automate API Testing With Postman [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.smashingmagazine.com/2020/09/automate-api-testing-postman/

25. What Is a HRMS? [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.netsuite.com/portal/resource/articles/human-resources/human-resources-management-system-hrms.shtml.

26. Запорожець О. І. Безпека життєдіяльності, соціальні небезпеки [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://subj.ukrlit.com/bezpeka-zhittyediyalnosti-zaporozhec-o-i/.

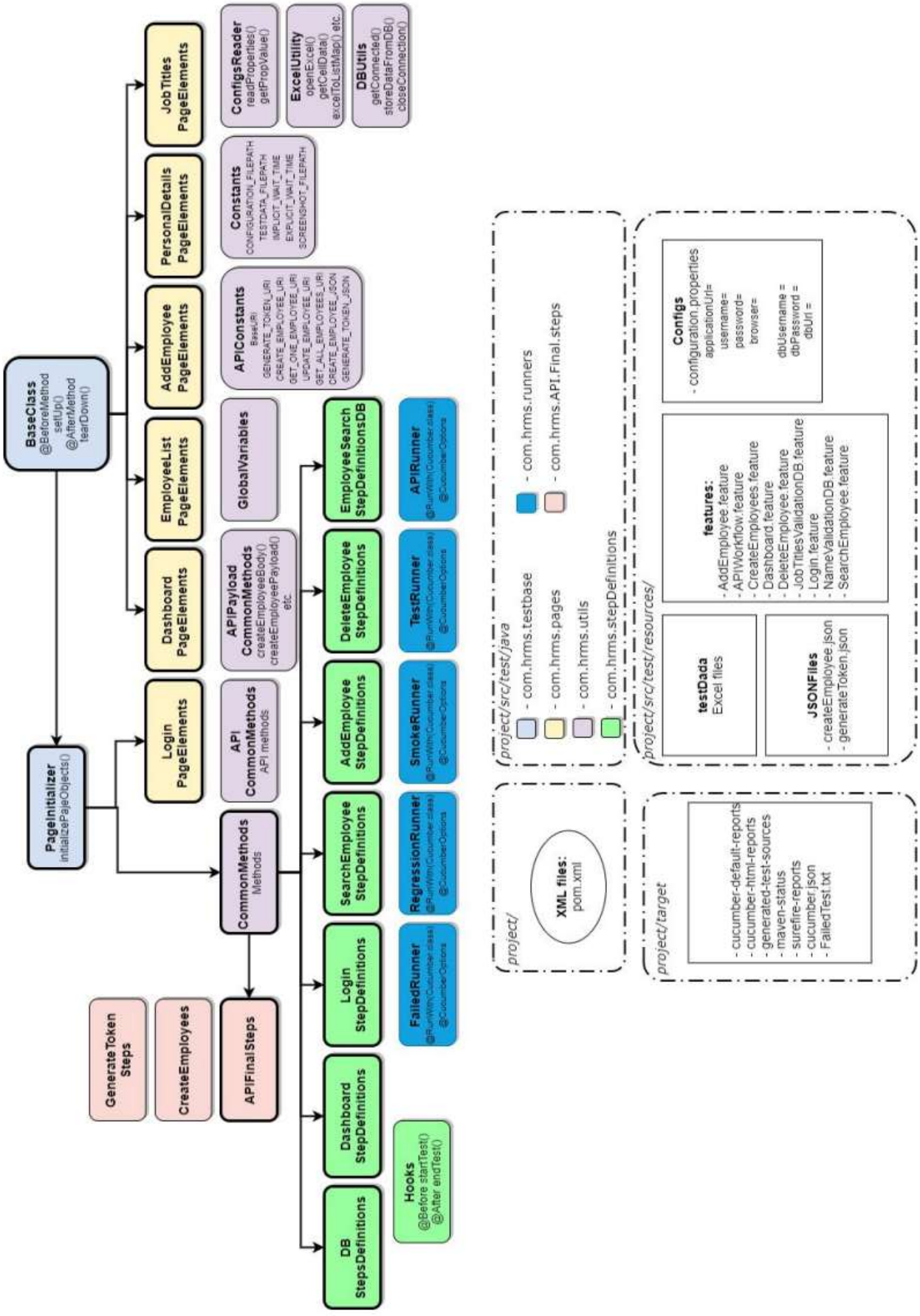
27. Інструкція з охорони праці при роботі на персональному комп’ютері [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://pro-op.com.ua/article/485-nstruktsya-z-ohoroni-prats-pri-robot-na-personalnomu-kompyuter.

28. Правові основи цивільної безпеки, працеохоронної політики та охорони праці [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://msn.khnu.km.ua/mod/page/view.php?id=110814.

ДОДАТКИ

Додаток А

Структура розробленого фреймворка



Feature файл для логін функціональності

```
Feature: Login functionality
  @sprint3 @regression @tc1101
  Scenario: Valid admin login
    #Given user is navigated to HRMS application
    When user enters valid username and valid password
    And user clicks on login button
    Then user is successfully logged in

  @regression @tc1102
  Scenario: Valid ess login
    #Given user is navigated to HRMS application
    When user enters ess username and ess password
    And user clicks on login button
    Then user is successfully logged in

  @tc1103
  Scenario: Invalid admin login
   # Given user is navigated to HRMS application
    When user enters invalid username and password
    And user clicks on login button
    Then error message displayed

 @tc1105
 Scenario Outline: Invalid login functionality
   When user enters different "<username>" and "<password>" and
verify the "<error>" for it
   Examples:
     | username | password | error |
     |admin     |cristiano |Invalid credentials|
     |ronaldo   |Hum@nhrm123|Invalid credentials|
     |          |Hum@nhrm123|Username cannot be empty|
     |admin     |           |Password cannot be empty|
```

Додаток В

Feature файл для тестування API

```
Feature: API workflow test
  Background: for generating the token before every request
    #to generate the token for all the request, we kept it here
in background
    Given a JWT is generated

  @api
  Scenario: API test case for creating the employee
    Given a request is prepared for creating an employee
    When a POST call is made to create an employee
    Then the status code for creating an employee is 201
    And the response body contains key "Message" and value
"Employee Created"
    And the employee id "Employee.employee_id" is stored as
global to be used for other request

  @api
  Scenario: Getting the created employee
    Given a request is prepared for getting a created employee
    When a GET call is made to get this employee
    Then the status code for this emp is 200
    And the employee id "employee.employee_id" should match with
global emp id
    And the retrieved data at "employee" object should match with
the data used for creating the employee

|emp_firstname|emp_lastname|emp_middle_name|emp_gender|emp_birthd
ay|emp_status|emp_job_title|
    |sara         |bou         |ms             |Female    |2011-
01-12  |confirmed |QA Engineer  |
```

Додаток Д

Клас Common Methods

```
public class CommonMethods extends PageInitializer {
    public static void closeBrowser() {
        Log.info("My test case is about to complete");
        Log.endTestCase("This is my login test again");
        driver.quit();
    }

    public static void sendText(WebElement element, String
textToSend) {
        element.clear();
        element.sendKeys(textToSend);
    }

    //to get webdriver wait
    public static WebDriverWait getWait() {
        WebDriverWait wait = new WebDriverWait(driver,
Constants.EXPLICIT_WAIT);
        return wait;
    }

    //this method will wait for the element to be clickable
    public static void waitForClickability(WebElement element) {

getWait().until(ExpectedConditions.elementToBeClickable(element));
    }

    //this method will perform click operation but before perform
click, it will wait
    //for the element to be clickable
    public static void click(WebElement element) {
        waitForClickability(element);
        element.click();
    }

    //this method will return JavascriptExecutor Object

    public static JavascriptExecutor getJSExecutor() {
        JavascriptExecutor js = (JavascriptExecutor) driver;
        return js;
    }

    //this function will perform click on element using javascript
executor
    public static void jsClick(WebElement element) {
        getJSExecutor().executeScript("arguments[0].click();",
element);
    }

    //selecting the dropdown using text
    public static void selectDropdown(WebElement element, String
```

```
text) {
        Select s = new Select(element);
        s.selectByVisibleText(text);
    }


    public static byte[] takeScreenshot(String fileName) {
        TakesScreenshot ts = (TakesScreenshot) driver;
        byte[] picBytes = ts.getScreenshotAs(OutputType.BYTES);
        File sourceFile = ts.getScreenshotAs(OutputType.FILE);

        try {
            FileUtils.copyFile(sourceFile,
        new File(Constants.SCREENSHOT_FILEPATH + fileName + " " +
                 getTimeStamp("yyyy-MM-dd-HH-mm-ss") + ".png"));

        } catch (IOException e) {
            e.printStackTrace();
        }
        return picBytes;
    }

    public static String getTimeStamp(String pattern) {
        Date date = new Date();
        SimpleDateFormat sdf = new SimpleDateFormat(pattern);
        return sdf.format(date);
    }
```

Додаток Е

Клас Constants

```
public class Constants {

    public static final String CONFIGURATION_FILEPATH =
            System.getProperty("user.dir") +
"/src/test/resources/config/config.properties";
    public static final int EXPLICIT_WAIT = 20;
    public static final int IMPLICIT_WAIT = 10;
    public static final String TESTDATA_FILEPATH =
            System.getProperty("user.dir") +
"/src/test/resources/testdata/batch14excel.xlsx";

    public static final String SCREENSHOT_FILEPATH =
System.getProperty("user.dir") + "/screenshots/";
```

Додаток Ж

Клас API Constants

```
Package utils;
import io.restassured.RestAssured;

public class APIConstants {

    public static final String BaseURI = RestAssured.baseURI
= "http://hrm.syntaxtechs.net/syntaxapi/api";
    public static final String GENERATE_TOKEN_URI =
BaseURI+"/generateToken.php";
    public static final String CREATE_EMPLOYEE_URI =
BaseURI+"/createEmployee.php";
    public static final String GET_ONE_EMPLOYEE_URI =
BaseURI+"/getOneEmployee.php";
    public static final String UPDATE_EMPLOYEE_URI =
BaseURI+"/updateEmployee.php";
    public static final String GET_ALL_EMPLOYEE_URI =
BaseURI+"/getAllEmployees.php";
    public static final String PARTIALLY_UPDATE_EMPLOYEE_URI
= BaseURI+"/updatePartialEmplyeesDetails.php";

    public static final String Header_Key_Content_Type =
"Content-Type";
    public static final String Header_Value_Content_Type =
"application/json";
    public static final String Header_Key_Authorization =
"Authorization";
```

Додаток 3

Клас API Payload Constants

```
package utils;
import org.json.JSONObject;
public class APIPayloadConstant {

    public static String createEmployeePayload(){
        String createEmployeePayload =
                "{\n" +
                  "  \"emp_firstname\": \"sara\",\n" +
                  "  \"emp_lastname\": \"bou\",\n" +
                  "  \"emp_middle_name\": \"ms\",\n" +
                  "  \"emp_gender\": \"F\",\n" +
                  "  \"emp_birthday\": \"2011-01-12\",\n" +
                  "  \"emp_status\": \"confirmed\",\n" +
                  "  \"emp_job_title\": \"QA Engineer\"\n" +
                  "}";
        return createEmployeePayload;
    }
    public static String createEmployeeJsonBody(){
        JSONObject obj = new JSONObject();
        obj.put("emp_firstname", "sara");
        obj.put("emp_lastname", "bou");
        obj.put("emp_middle_name", "ms");
        obj.put("emp_gender", "F");
        obj.put("emp_birthday", "2011-01-12");
        obj.put("emp_status", "confirmed");
        obj.put("emp_job_title","QA Engineer");
        return obj.toString();
    }
    public static String createEmployeePayloadDynamic(String
firstname, String lastname, String middlename,
String empStatus, String jobTitle){
        JSONObject obj = new JSONObject();
        obj.put("emp_firstname", firstname);
        obj.put("emp_lastname", lastname);
        obj.put("emp_middle_name", middlename);
        obj.put("emp_gender", gender);
        obj.put("emp_birthday", dob);
        obj.put("emp_status", empStatus);
        obj.put("emp_job_title",jobTitle);
        return obj.toString();
    }

    public static String adminPayload(){
        String adminPayload =
                "{\n" +
                  "  \"email\": \"batch14@test.com\",\n" +
                  "  \"password\": \"Test@123\"\n" +
                    "}";
       return adminPayload;
}
```

Додаток К

Клас Login Page

```
package pages;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import utils.CommonMethods;

public class LoginPage extends CommonMethods {

    @FindBy(xpath ="//input[@id='txtUsername']")
    public WebElement usernameTextField;

    @FindBy(id="txtPassword")
    public WebElement passwordTextField;

    @FindBy(xpath="//*[@id='btnLogin']")
    public WebElement loginButton;

    @FindBy(xpath="//*[@id='spanMessage']")
    public WebElement errorMessage;

    public LoginPage(){
       //call selenium page factory
        PageFactory.initElements(driver, this);
    }
}
```

Додаток Л

Клас Login Steps

```
public class LoginSteps extends CommonMethods {
	WebDriver driver;

	@Given("user is navigated to HRMS application")
	public void user_is_navigated_to_hrms_application() {
			openBrowserAndLaunchApplication();
	   WebDriverManager.chromedriver().setup();
	   driver=new ChromeDriver();

   driver.get("http://hrm.syntaxtechs.net/humanresources/symfony/w
   eb/index.php/auth/login");
	   driver.manage().timeouts().implicitlyWait(20,
   TimeUnit.SECONDS);
	   driver.manage().window().maximize();
	}

	@When("user enters valid username and valid password")
	public void user_enters_valid_username_and_valid_password()
   {
	   LoginPage login = new LoginPage();
	    WebElement usernameField =
   driver.findElement(By.id("txtUsername"));

   usernameField.sendKeys(ConfigReader.getPropertyValue("username"
   ));
	    sendText(login.usernameTextField,
   ConfigReader.getPropertyValue("username"));
	    WebElement passwordField =
   driver.findElement(By.id("txtPassword"));

   passwordField.sendKeys(ConfigReader.getPropertyValue("password"
   ));
	    sendText(login.passwordTextField,
   ConfigReader.getPropertyValue("password"));
	}

	@When("user clicks on login button")
	public void user_clicks_on_login_button() {
	  LoginPage login = new LoginPage();
	  WebElement loginButton =
   driver.findElement(By.id("btnLogin"));
	    click(login.loginButton);
	}

	@Then("user is successfully logged in")
	public void user_is_successfully_logged_in() {
	    WebElement welcomeMessage =
   driver.findElement(By.id("welcome"));
	    if(dashboard.welcomeMessage.isDisplayed()){
	        System.out.println("Test case is passed");
```

```
        }else{
            System.out.println("Test is failed");
        }
    }

    @When("user enters ess username and ess password")
    public void user_enters_ess_username_and_ess_password() {
        LoginPage login = new LoginPage();
        WebElement usernameField =
driver.findElement(By.id("txtUsername"));
        sendText(login.usernameTextField, "asmahuma321");
        WebElement passwordField =
driver.findElement(By.id("txtPassword"));
        sendText(login.passwordTextField, "Hum@nhrm123");
    }

    @When("user enters invalid username and password")
    public void user_enters_invalid_username_and_password() {
       LoginPage login = new LoginPage();
       WebElement usernameField =
driver.findElement(By.id("txtUsername"));
        sendText(login.usernameTextField, "admin123");
        WebElement passwordField =
driver.findElement(By.id("txtPassword"));
        sendText(login.passwordTextField, "Hum@nhrm");
    }

    @Then("error message displayed")
    public void error_message_displayed() {
        System.out.println("Error message displayed");
    }

    @When("user enters different {string} and {string} and
verify the {string} for it")
    public void
user_enters_different_and_and_verify_the_for_it(String
username, String password, String errorMessage) {
        sendText(login.usernameTextField, username);
        sendText(login.passwordTextField, password);
        click(login.loginButton);

        String errorActual =  login.errorMessage.getText();
        Assert.assertEquals(errorMessage, errorActual);
    }
}
```

Додаток Н

Клас API Steps

```
public class APISteps{

    String baseURI =  RestAssured.baseURI =
"http://hrm.syntaxtechs.net/syntaxapi/api";

    //to perform CRUD operations
    String token = "Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2NzQwODc5MTMsI
mlzcyI6ImxvY2FsaG9zdCIsImV4cCI6MTY3NDEzMTExMywidXNlcklkIjoiNDc
yNSJ9.usxFfksgB6QFw5pwTD8p14ydO3kEG0XxTZ3tzF0816g";
    static String employee_id;

    @Test
    public void bgetOneEmployee(){
        //prepare the request
        //to prepare the request, we use request specification
        RequestSpecification request =
given().header("Authorization", token)
                .header("Content-Type", "application/json")
                .queryParam("employee_id", employee_id);

        //to hit the end point/ to make the request which will
return response
        Response response =
request.when().get("/getOneEmployee.php");

       // System.out.println(response.asString());
        response.prettyPrint();
        //verifying the status code
        response.then().assertThat().statusCode(200);

        //using jsonpath method, we are extracting the value
from the response body
        String firstName =
response.jsonPath().getString("employee.emp_firstname");
        System.out.println(firstName);
        //first way of assertion
        Assert.assertEquals(firstName, "sara");
        //second way of assertion to verify the value in
response body using hamcrest matchers

response.then().assertThat().body("employee.emp_firstname",
equalTo("sara"));
    }

    @Test
    public void acreateEmployee(){
        RequestSpecification request =
given().header("Authorization", token)
                .header("Content-Type", "application/json").
                body("{\n" +
```

```
                    "  \"emp_firstname\": \"sara\",\n" +
                    "  \"emp_lastname\": \"bou\",\n" +
                    "  \"emp_middle_name\": \"ms\",\n" +
                    "  \"emp_gender\": \"F\",\n" +
                    "  \"emp_birthday\": \"2011-01-12\",\n" +
                    "  \"emp_status\": \"confirmed\",\n" +
                    "  \"emp_job_title\": \"QA Engineer\"\n" +
                     "}");
        Response response =
request.when().post("/createEmployee.php");
        response.prettyPrint();
        //verfying the statuys code which is 201
        response.then().assertThat().statusCode(201);
        //getting the employee id from the response and use it
as static one
        employee_id =
response.jsonPath().getString("Employee.employee_id");
        System.out.println(employee_id);

response.then().assertThat().body("Employee.emp_lastname",
equalTo("bou"));

response.then().assertThat().body("Employee.emp_middle_name",
equalTo("ms"));
        //verify console header
        response.then().assertThat().header("Server",
"Apache/2.4.39 (Win64) PHP/7.2.18");

    }

    @Test
    public void cupdateEmployee(){
    RequestSpecification request =
given().header("Authorization", token)
            .header("Content-Type", "application/json").
            body("{\n" +
              "  \"employee_id\": \""+ employee_id +"\",\n" +
              "  \"emp_firstname\": \"sohel\",\n" +
              "  \"emp_lastname\": \"abbasi\",\n" +
              "  \"emp_middle_name\": \"ms\",\n" +
              "  \"emp_gender\": \"M\",\n" +
              "  \"emp_birthday\": \"2015-01-12\",\n" +
              "  \"emp_status\": \"conf\",\n" +
              "  \"emp_job_title\": \"qa\"\n" +
              "}");

    Response response =
request.when().put("/updateEmployee.php");
    response.prettyPrint();
    //verification
    response.then().assertThat().statusCode(200);
    response.then().assertThat().body("Message",
equalTo("Employee record Updated"));
    }
```