МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2022р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"МАГІСТР"**

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Тестовий автоматизований фреймворк з використанням технологій Rest Assured Jenkins мовою програмування Java"

Виконавець: студент групи УС-212М Мороз Богдан Петрович

Керівник: к.т.н., доцент Колісник Олена Василівна

Нормоконтролер: Ігор РАЙЧЕВ

Київ – 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Кібербезпеки, комп'ютерної та програмної інженерії*

Кафедра *Комп'ютерних інформаційних* технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«____»_____2022р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Мороза Богдана Петровича
(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Тестовий автоматизований фреймворк з використанням технологій Rest Assured Jenkins мовою програмування Java» затверджена наказом ректора від 28 вересня 2022 р. за № 1774/ст.
2. **Термін виконання роботи:** з 26.09.2022 по 21.11.2022
3. **Вихідні дані до роботи:** основи теорії тестування, ключові підходи та методології в автоматизації тестування, розробка автоматизованої системи тестування.
4. **Зміст пояснювальної записки**: вступ, аналітичний огляд і постановка завдання, розгляд особливостей тестування, дослідження технологій та засобів, розробка програмного продукту, оцінка якості технології, висновки.
5. **Перелік обов'язкового ілюстративного матеріалу:** методи чорного, білого та сірого ящика; схеми процесу тестування; рівні тестування;.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу, створення плану дипломної роботи та побудова плану-графіку виконання робіт.	26.09.2022 – 28.09.2022	
2.	Огляд та аналіз наукової літератури по темі дипломної роботи та написання Розділу 1.	29.09.2022 – 09.10.2022	
3.	Написання Розділу 2 дипломної роботи.	10.10.2022 –20.10.2022	
4.	Написання Розділу 3 дипломної роботи. Завершення створення пояснювальної записки дипломної роботи.	21.10.2022 – 31.10.2022	
5.	Оформлення та друк пояснювальної записки.	01.11.2022 – 07.11.2022	
6.	Створення презентації, доповіді та підготовка до захисту дипломної роботи.	08.11.2022 – 15.11.2022	
7.	Підготовка матеріалів дипломної роботи для передачі секретарю ДЕК (папка, конверт, диск із файлом диплому, рецензія, відгук).	16.11.2022 – 18.11.2022	

7. **Дата видачі завдання:** «26» вересня 2022 р.

Керівник дипломної роботи Олена КОЛІСНИК

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання Богдан МОРОЗ

(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Тестовий автоматизований фреймворк з використанням технологій Rest Assured Jenkins мовою програмування Java» складається із вступу, трьох розділів, загальних висновків, списку використаних джерел містить 103 сторінки, та 18 рисунків. Список бібліографічних посилань складається з 11 найменувань.

Ключові слова: QA, FUNCIONAL TESTING, MON-FUNCIONAL TESTING, JAVA , REST ASSURED, JENKINS, SELENIUM, TESTING COVERAGE, TESTING METHODOLOGIES

Актуальність. На даний час жоден проект не може обійти етап тестування, який спрямований на виявлення та усунення якомога більшої кількості помилок. В результаті цього зростає кінцева якість продукту та рівень задоволеності кінцевого користувача. Проте, незважаючи на те скільки часу та якими методиками ми б не тестували проект, все протестувати не можливо.

Метою дипломної роботи є дослідження процесу тестування, видів, способів створення і застосування тест-кейсів. В результаті, на основі отриманих знань необхідно розробити проект тестового автоматизованого фрйемворку.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

- виявити місце, яке займає тестування в процесі розробки ПС;
- розглянути найвідоміші види дефектів і причини їх виникнення, а такождослідити системи пошуку та відслідковування помилок;
- ознайомитися з техніками створення тестів та їх застосуванням;
- обрати платформу для навантажувального тестування веб-сервісу;
- оволодіти процесом автоматизації функціонального тестування;
- розробити проект авто-тестів для веб-сервісу.

Об’єктом дослідження є процес тестування ІТ проектів, автоматизація

рутинного тестування та тестові фреймворки.

Предметом дослідження є автоматизація роботи мануального тестувальника за допомогою найкращих практик та ряду програмних систем.

Методи дослідження включають у себе:

- методи тестування програмного забезпечення;
- методи верифікації рівня якості програмного забезпечення;
- методи створення автоматизованих тест-сценаріїв.

Теоретичною основою дипломної роботи стали вітчизняні та зарубіжні дослідження щодо забезпечення якості програмного забезпечення та публікації на сайтах, присвячені питанням тестування програмних систем.

Теоретична і практична значимість роботи полягає в тому, що на основі отриманих знань:

1) по зібраному матеріалу можна сформувати методичний посібник для студентів і включити його в програму навчання в якості додаткового курсу до дисципліни "Тестування ПЗ інформаційних систем";

2) розроблено проект автоматизованого створення тестів для тестування веб-сервісу, який успішно введено в експлуатацію.

На захист виносяться наступні положення:

1) процес ручного тестування ПЗ;

2) автоматизація тестування;

3) проект автоматизованого фрйемворку для веб-сервісу.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ – Програмне забезпечення

ТЗ – Технічне завдання

QA – Контроль якості

QAE – Тестувальник

FT – Функціональне тестування

NFT – Не функціональне тестування

BB – Чорний ящик

SDLC – Життєвий цикл програмного забезпечення

WB – Білий ящик

GB – Сірий Ящик

TDD – Керована тестами розробка

BDD – Керована поведінкою розробка

EQC – Equivalence classes

RQ – Вимога

MT – Модульне тестування

ST – Системне тестування

SQL – Декларативна мова програмування для взаємодії користувача з базами даних

PT – Тестування проуктивності

ВСТУП

В дипломній роботі проведено дослідження особливостей та практик написання фреймворків автоматизованого тестування, а також запропоновано проект безкоштовної спрощеної програмної системи базового тестування веб додатку крамниці для продажу товарів.

Зважаючи на постійно зростаючий рівень навантаження на мануальних тестувальників та людський фактор при тестуванні доцільною є розробка автоматизованого тестового фреймворку , який буде легко адаптувати під потреб замовника та завдяки ній буде можливо зменшити кількість рутинного тестування серед мануальних тестувальників. Це дозволить підвищити ефективність, якість тестування та зменшить його час.

Використання сучасних програмних технологій дозволить сформувати оптимальний розклад навантаження, враховуючи специфіку проектного завдання. Реалізація даного проекту – це ще один крок до полегшення роботи мануальних тестувальників та поліпшення кінцевої якості проекту.

Теоретичні дослідження роботи зосереджені на аналізі методів та способів тестування та розробки автоматизованих тестових фреймворків, які були використані при розробці програмних аналогів. Мета теоретичного дослідження полягає у розробленні алгоритму розробки автоматизованого фреймворку, що дозволить покращити процес тестування на проекті.

Експериментальні дослідження полягають у перевірці працездатності та ефективності розробленого автоматизованого фреймворку та доцільності його використання при різних специфіка проекту , шляхом тестування системи та аналізу результатів автоматизованого тестування.

Наукова новизна полягає у створенні універсального продукту, який буде легко адаптуватися під будь-який проект з обраного кола специфіки, звести до мінімуму задіяність мануальних тестувальників у рутинному тестуванні та зменшити час необхідний для проходження циклу тестування.

Практична значимість полягає у розробленні автоматизованого тестувального фреймворку з врахуванням специфіки обраного кола проектів.Використання розробленого фреймворку дозволить оптимальним чином розподілити задачі між тестувальниками під час проходження циклу тестування та запобігти впливу людського фактору.

В основі розробки лежить каркас, в якому є різні шари, та кожен з них включає в себе декілька модулів, а самі модулі складаються з тестових сценаріїв. Для реалізації модулів необхідно побудувати підсистему подальшого редагування вже згенерованих сценаріїв, причому потрібні можливості редагувати власне тести, дублювати, добавляти, видаляти та створювати нові тестові сценарії.

В результаті проведеної роботи буде отримано систему, яку можна буде повноцінно використовувати в якості автоматизованого тестового фреймворку для повсякденного тестування критичних функцій веб-додатку з обраного кола фукціональностей.

РОЗДІЛ 1.
ТЕОРІЯ ТЕСТУВАННЯ. МЕТОДОЛОГІЇ ТА ПІДХОДИ У ТЕСТУВАННІ

1.1. Тестування

Тестування — це процес оцінювання системи або її компонентів з метою визначення того, задовольняє вона встановленим вимогам чи ні. Простими словами, тестування — це виконання системи з метою виявлення будь-яких прогалин, помилок або відсутніх вимог, що суперечать фактичним вимогам.

Відповідно до стандарту ANSI/IEEE 1059, тестування можна визначити як — процес аналізу елемента програмного забезпечення для виявлення відмінностей між існуючими та необхідними умовами (тобто дефекти/помилки/баги) і для оцінки функцій елемента програмного забезпечення [1].

Це залежить від процесу та пов’язаних із ним зацікавлених сторін проекту(ів). У IT-індустрії великі компанії мають команду, яка відповідає за оцінку розробленого програмного забезпечення в контексті заданих вимог. Крім того, розробники також проводять тестування, яке називається модульним тестуванням. У більшості випадків до тестування системи в межах своїх повноважень залучені такі спеціалісти:

- Тестер програмного забезпечення;
- Розробник програмного забезпечення;
- Керівник проекту/менеджер;
- Кінцевий користувач.

Різні компанії мають різні позначення для людей, які тестують програмне забезпечення на основі свого досвіду та знань, наприклад, тестувальник програмного забезпечення, інженер із забезпечення якості програмного забезпечення, аналітик із забезпечення якості тощо.

Неможливо протестувати програмне забезпечення в будь-який час протягом його циклу.

Різні компанії мають різні позначення для людей, які тестують програмне забезпечення на основі свого досвіду та знань, наприклад, тестувальник програмного забезпечення, інженер із забезпечення якості програмного забезпечення, аналітик із забезпечення якості тощо.

Неможливо протестувати програмне забезпечення в будь-який час протягом його циклу.

Ранній початок тестування зменшує витрати та час на переробку та створення безпомилкового програмного забезпечення, яке доставляється клієнту. Однак у життєвому циклі розробки програмного забезпечення (SDLC) тестування можна розпочати з фази збору вимог і продовжувати до розгортання програмного забезпечення.

Це також залежить від моделі розробки, яка використовується. Наприклад, у моделі Waterfall формальне тестування проводиться на етапі тестування; але в інкрементній моделі тестування виконується в кінці кожного збільшення/ітерації, а в кінці тестується вся програма.

Тестування проводиться в різних формах на кожному етапі SDLC :

- На етапі збору вимог аналіз і перевірка вимог також розглядаються як тестування;
- Перегляд дизайну на етапі проектування з наміром покращити дизайн також вважається тестуванням;
- Тестування, яке виконує розробник після завершення коду, також класифікується як тестування.

Важко визначити, коли припинити тестування, оскільки тестування — це нескінченний процес, і ніхто не може стверджувати, що програмне забезпечення перевірено на 100%. Щоб зупинити процес тестування, необхідно врахувати наступні аспекти :

- Терміни тестування;
- Завершення виконання тесту;
- Завершення функціонального та кодового покриття до певної точки;
- Рівень помилок падає нижче певного рівня, і жодних високопріоритетних помилок не виявлено;
- Управлінське рішення.

Багато організацій у всьому світі розробляють і впроваджують різні стандарти для покращення якості свого програмного забезпечення. У цій главі коротко описано деякі широко використовувані стандарти, пов’язані із забезпеченням якості та тестуванням.

ISO/IEC 9126

Цей стандарт розглядає наступні аспекти для визначення якості програмного додатку:

- Якісна модель;
- Зовнішні показники;
- Внутрішні показники;
- Показники якості у використанні.

Цей стандарт представляє певний набір атрибутів якості для будь-якого програмного забезпечення, наприклад :

- Функціональність;
- Надійність;
- Юзабіліті;
- Ефективність;
- Ремонтопридатність;

- Портативність.

Вищезазначені атрибути якості далі поділяються на підфактори, які ви можете вивчити під час детального вивчення стандарту.

ISO/IEC 9241-11

Частина 11 цього стандарту стосується того, якою мірою продукт може бути використаний певними користувачами для досягнення визначених цілей з ефективністю, ефективністю та задоволенням у визначеному контексті використання.

Цей стандарт пропонує структуру, яка описує компоненти зручності використання та взаємозв'язок між ними. У цьому стандарті зручність використання розглядається з точки зору продуктивності та задоволеності користувача. Відповідно до ISO 9241-11, зручність використання залежить від контексту використання, і рівень зручності буде змінюватися зі зміною контексту [2].

ISO/IEC 25000:2005

ISO/IEC 25000:2005 широко відомий як стандарт, який містить вказівки щодо вимог до якості програмного забезпечення та оцінювання (SQuaRE). Цей стандарт допомагає в організації та покращенні процесу, пов'язаного з вимогами до якості програмного забезпечення та їх оцінками. Насправді ISO-25000 замінює два старі стандарти ISO, тобто ISO-9126 та ISO-14598.

SQuaRE поділено на підчастини, такі як :

- ISO 2500n − Відділ управління якістю;
- ISO 2501n − Відділ моделі якості;
- ISO 2502n − Відділ вимірювання якості;
- ISO 2503n − Відділ вимог до якості;
- ISO 2504n − Відділ оцінки якості.

Основним вмістом SQuaRE є :

- Терміни та визначення;
- Еталонні моделі;

- Загальний посібник;
- Індивідуальні напрямні поділу;
- Стандарт, пов’язаний із розробкою вимог (тобто процес специфікації, планування, вимірювання та оцінки).

ISO/IEC 12119

Цей стандарт стосується пакетів програмного забезпечення, що поставляються клієнту. Він не фокусується на виробничому процесі клієнта. Основний зміст пов’язаний із такими пунктами :

- Набір вимог до пакетів програм;
- Інструкції щодо перевірки наданого програмного пакета на відповідність заданим вимогам.

1.2. Тестування вручну

Ручне тестування включає тестування програмного забезпечення вручну, тобто без використання будь-яких автоматизованих інструментів чи сценаріїв. У цьому типі тестер бере на себе роль кінцевого користувача та перевіряє програмне забезпечення, щоб виявити будь-яку несподівану поведінку чи помилку. Існують різні етапи ручного тестування, такі як модульне тестування, інтеграційне тестування, системне тестування та тестування прийнятності користувача.

Тестувальники використовують плани тестування, тестові випадки або тестові сценарії для перевірки програмного забезпечення, щоб забезпечити повноту тестування. Ручне тестування також включає пошукове тестування, оскільки тестувальники досліджують програмне забезпечення, щоб виявити в ньому помилки.

1.3. Етапи ручного тестування

1.3.1. Модульне тестування

Модульне тестування передбачає перевірку окремих компонентів або одиниць вихідного коду. Блок можна назвати найменшою тестованою частиною будь-якого програмного забезпечення. Він зосереджений на тестуванні функціональності окремих компонентів у програмі. Його часто використовують розробники для виявлення помилок на ранніх стадіях циклу розробки.

Випадок модульного тестування був би таким же фундаментальним, як натискання кнопки на веб-сторінці та перевірка того, чи виконує вона потрібну операцію. Наприклад, переконайтеся, що кнопка спільного доступу на веб-сторінці дає змогу поділитися правильним посиланням на сторінку.

1.3.2. Інтеграційне тестування

Інтеграційне тестування є наступним кроком після модульного тестування. Кілька блоків інтегровані для тестування як єдиного цілого. Наприклад, тестування серії веб-сторінок у певному порядку для перевірки сумісності.

Цей підхід допомагає QA оцінити, як кілька компонентів програми працюють разом, щоб забезпечити бажаний результат. Виконання інтеграційного тестування паралельно з розробкою дозволяє розробникам швидше виявляти та знаходити помилки.

1.3.3. Тестування системи

Як випливає з назви, тестування системи передбачає тестування всіх інтегрованих модулів програмного забезпечення в цілому. Це допомагає QA перевірити, чи відповідає система бажаним вимогам. Він включає численні тести, такі як перевірка виходу на основі певних вхідних даних,тестування користувацького досвіду тощо.

Команди виконують кілька типів тестування системи, як-от регресійне тестування, стрес-тестування, функціональне тестування тощо, залежно від їхнього доступу до часу та ресурсів.

1.3.4. Тестування інтерфейсу користувача

Тестування інтерфейсу користувача, також відоме як тестування графічного інтерфейсу користувача, перевіряє та перевіряє різні аспекти будь-якого програмного забезпечення, з яким користувач взаємодіє під час його використання. Зазвичай це означає тестування візуальних елементів, щоб переконатися, що вони функціонують відповідно до вимог щодо функціональності та продуктивності. Тестування інтерфейсу користувача охоплює широкий спектр візуальних індикаторів і графічних піктограм – панелі інструментів, шрифти, меню, текстові поля, перемикачі, прапорці, кольори тощо. Це гарантує, що функції інтерфейсу користувача без помилок і працюють саме так, як вони повинні.

Разом з тестуванням елементів інтерфейсу користувача тестування має враховувати різні браузери, версії браузерів і пристрої. Люди отримують доступ до Інтернету з широкого діапазону комбінацій браузер-пристрій-ОС, а це означає, що користувальницький інтерфейс має ідеально відтворюватися та функціонувати з кожної з них. Іншими словами, міжбраузерне тестування має бути невід'ємною частиною будь-якої стратегії тестування інтерфейсу користувача.

Замість того, щоб завантажувати кожну версію браузера та купувати кожен пристрій, який використовує ваша цільова аудиторія, подумайте про використання хмарної інфраструктури тестування, такої як та, яку надає BrowserStack. Хмара реальних пристроїв BrowserStack пропонує понад 3000 реальних пристроїв і браузерів для ручного та автоматизованого тестування. Це означає, що користувачі можуть тестувати на кількох реальних пристроях і браузерах, зареєструвавшись, увійшовши в систему та вибравши необхідні комбінації. Перевірте, як ваш веб-сайт або додаток виглядає та працює в різних

браузерах, пристроях і операційних системах кількома клацаннями миші на вашій робочій станції [3].

1.3.5. Приймальні випробування

Основна мета приймального тестування — перевірити, чи придатна система в цілому для використання в реальному світі.

Приймальні випробування проводяться як внутрішні, так і зовнішні. Внутрішнє приймальне тестування (також відоме як альфа-тестування) виконується членами організації. Зовнішнє тестування (також відоме як бета-тестування) виконується обмеженою кількістю фактичних кінцевих користувачів. Цей підхід допомагає командам оцінити, наскільки продукт відповідає стандартам користувача. Він також визначає помилки на останньому етапі перед випуском продукту.

Серед інших форм приймального тестування особливої згадки заслуговує тестування доступності. Тестування доступності гарантує, що кожною функцією веб-сайту чи програми легко користуватися людям із вадами зору, слуху, дальтонізмом або будь-якими іншими фізичними проблемами. Вони можуть відчувати певну форму інвалідності, що означає, що їм потрібні певні допоміжні технології для роботи з певною технологією.

Управлінські функції мають певну важливу діяльність, яку треба робити лише спираючись, що вони відомі всім учасникам процесів . Якщо це не так, то потрібно звернути увагу та виправити отримане непорозуміння.

До функцій керування проектом відносяться:

- Створення плану;
- Робота з контролюванням процесів;
- Рахування вірогідності та аналіз;
- Отримання та винесення рішень;

- Обробка моніторингу;
- Оцінка результатів;
- Створення звітів;
- Експертність;
- Тестування результатів;
- Адміністрування процесів та людей.

1.4. Тестування чорного ящика

Black Box Testing — це метод тестування програмного забезпечення, за якого функціональні можливості програмного забезпечення перевіряються без знання внутрішньої структури коду, деталей реалізації та внутрішніх шляхів. Тестування Black Box в основному зосереджується на введенні та виведенні програмних додатків і повністю базується на вимогах і специфікаціях програмного забезпечення. Він також відомий як поведінкове тестування [4].

Рис. 1.1. Графічне зображення Black Box

Вищезазначена чорна скринька може бути будь-якою програмною системою, яку ви хочете перевірити. Наприклад, операційна система Windows, веб-сайт Google, база даних Oracle або навіть ваша власна програма. Використовуючи «Тестування чорного ящика» ви можете протестувати ці програми, просто зосереджуючись на входах і виходах, не знаючи реалізації їх внутрішнього коду.

Нижче наведено загальні кроки для проведення будь-якого типу тестування чорної скриньки.

Спочатку перевіряються вимоги та характеристики системи:

- Тестер вибирає дійсні вхідні дані (позитивний тестовий сценарій), щоб перевірити, чи SUT обробляє їх правильно. Крім того, деякі недійсні вхідні дані (негативний тестовий сценарій) вибираються для перевірки того, що SUT здатний їх виявити;
- Тестер визначає очікувані результати для всіх цих входів;
- Тестер програмного забезпечення створює тестові випадки з вибраних вхідних даних;
- Тестові випадки виконано;
- Тестер програмного забезпечення порівнює фактичні результати з очікуваними;
- Дефекти, якщо такі є, усуваються та перевіряються повторно.

Види тестування чорної скриньки

Існує багато типів тестування чорної скриньки, але нижче наведено найвідоміші з них:

- Функціональне тестування – цей тип тестування чорної скриньки пов’язаний із функціональними вимогами системи; це роблять тестувальники програмного забезпечення;
- Нефункціональне тестування – цей тип тестування чорної скриньки пов’язаний не з тестуванням конкретної функціональності, а з нефункціональними вимогами, такими як продуктивність, масштабованість, зручність використання;
- Регресійне тестування – регресійне тестування виконується після виправлень коду, оновлень або будь-якого іншого обслуговування системи, щоб перевірити, чи новий код не вплинув на існуючий.

Інструменти, що використовуються для тестування чорного ящика:

Інструменти, які використовуються для тестування чорної скриньки, значною мірою залежать від типу тестування чорної скриньки, який ви виконуєте.

Для функціональних/регресійних тестів можна використовувати – QTP, Selenium

Для нефункціональних тестів ви можете використовувати – LoadRunner, Jmeter

Методи тестування чорної скриньки

Нижче наведено відомі стратегії тестування серед багатьох, що використовуються в тестуванні чорної скриньки:

- Тестування класу еквівалентності: використовується для мінімізації кількості можливих тестових випадків до оптимального рівня, зберігаючи прийнятне тестове покриття;
- Перевірка граничних значень: Тестування граничних значень зосереджено на значеннях на межах. Ця методика визначає, чи є певний діапазон значень прийнятним системою чи ні. Це дуже корисно для зменшення кількості тестів. Він найбільш підходить для систем, де вхідний сигнал знаходиться в межах певних діапазонів;
- Тестування таблиці рішень: таблиця рішень поміщає причини та їхні наслідки в матрицю. У кожному стовпчику є унікальна комбінація.

Життєвий цикл тестування та розробки програмного забезпечення (SDLC)

Тестування «чорної скриньки» має власний життєвий цикл під назвою «Життєвий цикл тестування програмного забезпечення» (STLC), і він стосується кожного етапу життєвого циклу розробки програмного забезпечення розробки програмного забезпечення.

- Вимога – це початковий етап SDLC, і на цьому етапі збирається вимога. У цьому етапі також беруть участь тестувальники ПЗ;
- Планування та аналіз тестування – визначаються типи тестування, що застосовуються до проекту. Створюється План тестування, який визначає можливі ризики проекту та їх зменшення;
- Проектування – на цьому етапі тестові приклади/сценарії створюються

на основі документів вимог до програмного забезпечення;

- Виконання тесту – на цьому етапі виконуються підготовлені тестові випадки. Помилки, якщо такі є, виправлені та перевірені повторно.

1.5. Тестування білого ящика

Тестування білого ящика — це техніка тестування, за якої перевіряються внутрішня структура, дизайн і кодування програмного забезпечення, щоб перевірити потік введення-виведення та покращити дизайн, зручність використання та безпеку. У тестуванні білого ящика код видимий тестувальникам, тому його також називають тестуванням чистого ящика, тестування відкритого ящика, тестування прозорого ящика, тестування на основі коду та тестування скляного ящика.

Це одна з двох частин підходу Box Testing до тестування програмного забезпечення. Його аналог, тестування Blackbox, передбачає тестування з точки зору зовнішнього або кінцевого користувача. З іншого боку, тестування білої скриньки в розробці програмного забезпечення базується на внутрішній роботі програми та обертається навколо внутрішнього тестування.

Термін «WhiteBox» використовувався через концепцію прозорої коробки. Прозора коробка або назва WhiteBox символізує можливість бачити крізь зовнішню оболонку (або «коробку») програмного забезпечення його внутрішню роботу. Подібним чином «чорний ящик» у «Тестуванні чорного ящика» символізує відсутність можливості побачити внутрішню роботу програмного забезпечення, тому можна перевірити лише роботу кінцевого користувача.

Тестування білого ящика передбачає перевірку програмного коду для наступного:

- Внутрішні отвори безпеки;
- Порушені або погано структуровані шляхи в процесах кодування;
- Потік конкретних вхідних даних через код;
- Очікуваний результат;

- Функціональність умовних циклів;
- Тестування кожного оператора, об'єкта та функції на індивідуальній основі.

Тестування може проводитися на рівні системи, інтеграції та модуля розробки програмного забезпечення. Однією з основних цілей тестування whitebox є перевірка робочого процесу програми. Він передбачає перевірку серії попередньо визначених вхідних даних на очікувані або бажані результати, щоб, коли певний вхідний сигнал не призвів до очікуваного результату, ви зіткнулися з помилкою.

Ми розділили тестування на два основні кроки, щоб дати вам спрощене пояснення тестування білого ящика. Ось що роблять тестувальники під час тестування програми за допомогою техніки тестування білого ящика:

КРОК 1) РОЗУМІЙТЕ ВИХІДНИЙ КОД

Перше, що зазвичай робить тестувальник, це вивчає та розуміє вихідний код програми. Оскільки тестування білого ящика передбачає тестування внутрішньої роботи програми, тестувальник повинен добре знати мови програмування, які використовуються в програмах, які вони тестують. Крім того, особа, яка проводить тестування, повинна добре знати методи безпечного кодування. Безпека часто є однією з основних цілей тестування програмного забезпечення. Тестер має вміти виявляти проблеми з безпекою та запобігати атакам хакерів і наївних користувачів, які можуть свідомо чи несвідомо впровадити шкідливий код у програму.

КРОК 2) СТВОРИТИ ТЕСТОВІ КЕЙСИ ТА ВИКОНАТИ

Другий основний крок до тестування «білого ящика» включає перевірку вихідного коду програми на відповідність потоку та структури. Одним із способів є написання додаткового коду для перевірки вихідного коду програми. Тестер розробить невеликі тести для кожного процесу або серії процесів у програмі. Цей метод вимагає, щоб тестувальник мав глибоке знання коду, і часто це робить розробник. Інші методи включають ручне тестування, тестування проб і помилок і використання інструментів тестування.

Методи тестування білого ящика

Основною технікою тестування білої скриньки є аналіз покриття коду. Аналіз покриття коду усуває прогалини в наборі тестів. Він визначає області програми, які не перевіряються набором тестів. Після виявлення прогалин ви створюєте тестові випадки для перевірки неперевірених частин коду, тим самим підвищуючи якість програмного продукту

Існують автоматизовані інструменти для аналізу покриття коду. Нижче наведено кілька методів аналізу покриття, які можуть використовувати тестери:

Покриття операторів: Ця техніка вимагає перевірки кожного можливого оператора в коді принаймні один раз під час процесу тестування розробки програмного забезпечення.

Покриття розгалужень – ця техніка перевіряє всі можливі шляхи (if-else та інші умовні цикли) програмного додатку.

Крім перерахованого вище, існують численні типи покриття, такі як покриття умов, покриття кількох умов, покриття шляхів, покриття функцій тощо. Кожна техніка має свої переваги та намагається перевірити (покрити) усі частини програмного коду. Використовуючи покриття Statement і Branch, ви зазвичай досягаєте 80-90% покриття коду, чого достатньо.

Нижче наведено важливі методи тестування WhiteBox:

- Покриття заяви;
- Покриття рішень;
- Покриття філій;
- Покриття умов;
- Покриття кількох умов;
- Покриття кінцевого автомата;
- Покриття шляху;
- Тестування контрольного потоку;
- Тестування потоку даних.

Типи тестування білого ящика

Тестування білого ящика охоплює кілька типів тестування, які використовуються для оцінки зручності використання програми, блоку коду чи конкретного програмного пакета. Нижче перераховано :

- Модульне тестування: часто це перший тип тестування програми. Модульне тестування виконується для кожного блоку або блоку коду в міру його розробки. Модульне тестування по суті виконується програмістом. Як розробник програмного забезпечення, ви розробляєте кілька рядків коду, одну функцію або об’єкт і перевіряєте його, щоб переконатися, що він працює, перш ніж продовжити. Модульне тестування допомагає виявити більшість помилок на ранніх етапах життєвого циклу розробки програмного забезпечення. Помилки, виявлені на цьому етапі, дешевші та їх легко виправити;
- Перевірка витоку пам’яті: витік пам’яті є основною причиною повільної роботи програм. Спеціаліст із забезпечення якості, який має досвід виявлення витоків пам’яті, необхідний у випадках, коли у вас повільно працює програмне забезпечення.

Окрім вищезазначеного, кілька типів тестування є частиною як чорного, так і білого ящиків. Вони наведені нижче:

- Тестування проникнення білого ящика: під час цього тестування тестер/розробник має повну інформацію про вихідний код програми, детальну інформацію про мережу, задіяні IP-адреси та всю інформацію про сервер, на якому працює програма. Мета полягає в тому, щоб атакувати код з кількох точок зору, щоб виявити загрози безпеці;
- Тестування на мутації білого ящика: Тестування на мутації часто використовується для виявлення найкращих методів кодування для розширення програмного рішення.

Інструменти тестування White Box

Нижче наведено список найпопулярніших інструментів для тестування:

- EclEmma;
- NUnit;
- PyUnit;
- HTMLUnit;
- CppUnit.

Переваги тестування White Box

- Оптимізація коду шляхом пошуку прихованих помилок;
- Тести білого ящика можна легко автоматизувати;
- Тестування є більш ретельним, оскільки зазвичай охоплюються всі шляхи коду;
- Тестування може початися на ранній стадії SDLC, навіть якщо GUI недоступний.

Недоліки тестування WhiteBox

- Тестування білого ящика може бути досить складним і дорогим;
- Розробники, які зазвичай виконують тестові випадки білого ящика, ненавидять це. Тестування білого ящика розробниками не є детальним і може призвести до помилок виробництва;
- Тестування білого ящика вимагає професійних ресурсів із детальним розумінням програмування та реалізації;
- Тестування білої скриньки займає багато часу, більшим програмним програмам потрібен час для повного тестування.

1.6. Автоматизоване Тестування

Автоматичне тестування — це техніка тестування програмного забезпечення, яка виконується за допомогою спеціальних програмних засобів автоматизованого тестування

для виконання набору тестових випадків. Навпаки, ручне тестування виконується людиною, яка сидить перед комп'ютером і ретельно виконує тестові кроки [5].

Програмне забезпечення для автоматизованого тестування також може вводити тестові дані в тестовану систему, порівнювати очікувані та фактичні результати та створювати детальні звіти про тестування. Автоматизація тестування програмного забезпечення вимагає значних вкладень грошей і ресурсів.

Послідовні цикли розробки вимагатимуть багаторазового виконання того самого набору тестів. Використовуючи інструмент автоматизації тестування, можна записати цей набір тестів і відтворити його за потреби. Після автоматизації набору тестів втручання людини не потрібне. Це покращило ROI автоматизації тестування. Метою автоматизації є зменшення кількості тестів, які потрібно запускати вручну, а не повна ліквідація ручного тестування [6].

Автоматизація тестування — найкращий спосіб підвищити ефективність, охоплення тестом і швидкість виконання тестування програмного забезпечення. Автоматизоване тестування програмного забезпечення є важливим з таких причин:

- Ручне тестування всіх робочих процесів, усіх полів, усіх негативних сценаріїв потребує часу та грошей;
- Важко перевірити наявність багатомовних сайтів вручну;
- Автоматизація тестування в тестуванні програмного забезпечення не вимагає втручання людини. Ви можете запустити автоматичний тест без нагляду (на ніч);
- Автоматизація тестування збільшує швидкість виконання тесту;
- Автоматизація допомагає збільшити охоплення тестуванням;
- Ручне тестування може стати нудним і, отже, схильним до помилок.

Тестові випадки, які потрібно автоматизувати, можна вибрати за наступним критерієм для підвищення рентабельності інвестицій автоматизації:

- Високий ризик – Критичні тестові випадки для бізнесу;
- Тестові випадки, які багаторазово виконуються;

- Тестові випадки, які дуже стомлюють або важко виконувати вручну;
- Тестові випадки, які забирають багато часу;

Наступна категорія тестів не підходить для автоматизації:

- Тестові випадки, створені заново і не виконані вручну принаймні один раз;
- Тестові випадки, вимоги до яких часто змінюються;
- Тестові випадки, які виконуються на нерегулярній основі.

У процесі автоматизації виконуються наступні кроки:

Крок 1) Вибір тестового інструменту;

Крок 2) Визначте область автоматизації;

Крок 3) Планування, проектування та розробка;

Крок 4) Виконання тесту;

Крок 5) Технічне обслуговування.

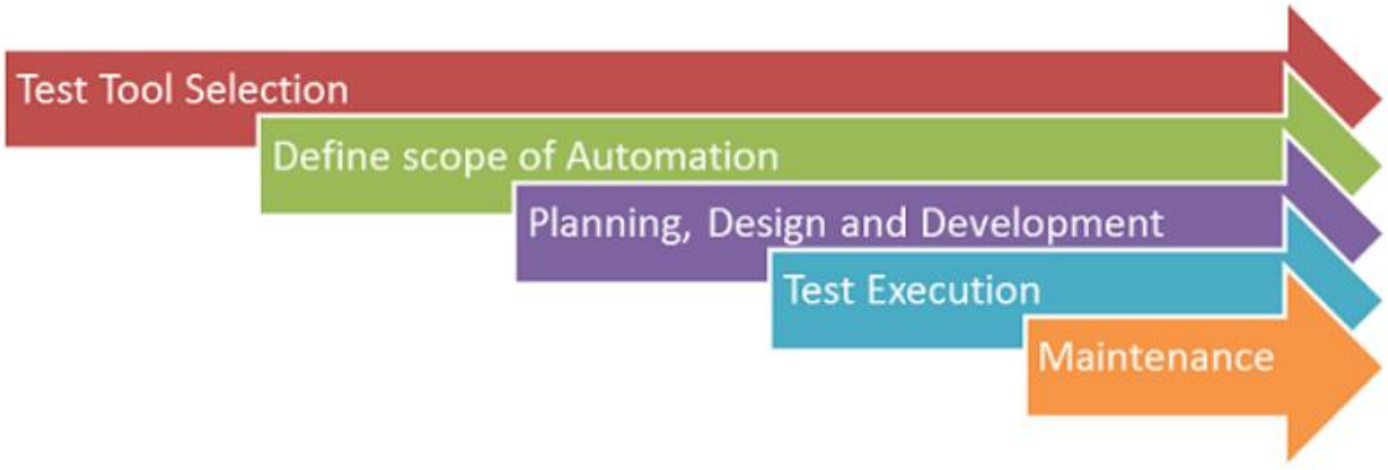

Рис 1.2. Процес автоматизованого тестування

Вибір інструменту тестування значною мірою залежить від технології, на якій побудовано тестову програму. Наприклад, QTP не підтримує Informatica. Тому QTP не можна використовувати для тестування програм Informatica. Було б гарною ідеєю провести перевірку концепції інструменту на AUT.

Сфера автоматизації – це область вашої тестованої програми, яка буде автоматизована. Наступні пункти допомагають визначити обсяг:

- Функції, важливі для бізнесу;
- Сценарії з великою кількістю даних;
- Загальні функції в програмах;
- Технічна можливість;
- Ступінь повторного використання бізнес-компонентів;
- Складність тестових випадків;
- Можливість використання тих самих тестів для кросбраузерного тестування.

Планування, дизайн і розробка

Під час цього етапу ви створюєте стратегію та план автоматизації, який містить такі деталі:

- Вибрані засоби автоматизації;
- Конструкція каркаса та його особливості;
- Внутрішні та поза межі предметів автоматизації;
- Підготовка автоматизованого стенду;
- Розклад і графік створення сценаріїв і виконання;
- Результати автоматизованого тестування.

Виконання тесту

Під час цієї фази виконуються сценарії автоматизації. Сценарії потребують вхідних тестових даних, перш ніж вони будуть налаштовані на запуск. Після виконання вони надають детальні звіти про випробування.

Виконання можна виконати безпосередньо за допомогою інструменту автоматизації або через інструмент керування тестуванням, який викличе інструмент автоматизації.

Приклад: Центр якості — це інструмент керування тестами, який, у свою чергу, викликає QTP для виконання сценаріїв автоматизації. Сценарії можуть виконуватися на одній машині або групі машин. Щоб заощадити час, виконання можна виконати вночі.

Test Automation Maintenance Approach — це фаза автоматизованого тестування, яка виконується для того, щоб перевірити, чи добре працюють нові функції, додані до програмного забезпечення. Технічне обслуговування в тестуванні автоматизації виконується, коли додаються нові сценарії автоматизації, і їх потрібно переглядати та підтримувати, щоб підвищити ефективність сценаріїв автоматизації з кожним наступним циклом випуску.

Структура для автоматизації

Фреймворк – це набір інструкцій з автоматизації, які допомагають у:

- Підтримання послідовності тестування;
- Покращує структурування тестів;
- Мінімальне використання коду;
- Менше обслуговування коду;
- Покращення можливості повторного використання;
- До коду можна залучати нетехнічних тестувальників;
- Термін навчання користування інструментом можна скоротити;
- Залучає дані, де це доречно.

Існує чотири типи фреймворків, які використовуються для автоматизованого тестування програмного забезпечення:

1. Структура автоматизації на основі даних;
2. Основа автоматизації, керована ключовими словами;
3. Модульна структура автоматизації;
4. Гібридна платформа автоматизації.

Щоб отримати максимальну рентабельність інвестицій автоматизації, дотримуйтеся наступного:

- Обсяг автоматизації необхідно детально визначити перед початком проекту. Це правильно визначає очікування від автоматизації;
- Виберіть правильний інструмент автоматизації. Інструмент слід вибирати не на основі його популярності, а на основі відповідності вимогам автоматизації;
- Стандарти сценаріїв. Під час написання сценаріїв для автоматизації необхідно дотримуватися стандартів;
- Вимірюйте показники.

Успіх автоматизації не можна визначити шляхом порівняння ручних зусиль із зусиллями автоматизації, але також зафіксувавши наступні показники:

- Відсоток виявлених дефектів;
- Час, необхідний для тестування автоматизації для кожного циклу випуску;
- Для випуску потрібно мінімальний час;
- Індекс задоволеності клієнтів;
- Підвищення продуктивності.

Дотримання наведених вище вказівок може значно допомогти зробити вашу автоматизацію успішною.

Переваги автоматизованого тестування

Рис.1.3. Вартість автоматизованого тестування

Нижче наведено переваги автоматизації тестування:

- На 70% швидше ніж ручне тестування;
- Більш широке тестове охоплення функцій програми;
- Надійні результати;
- Забезпечте послідовність;
- Економія часу та коштів;
- Покращує точність;
- Під час виконання не потрібне втручання людини;
- Підвищує ефективність;
- Краща швидкість виконання тестів;
- Тестові сценарії для повторного використання;
- Тестуйте часто і ретельно;
- Більшого циклу виконання можна досягти за допомогою автоматизації;
- Ранній час виходу на ринок.

Види автоматизованого тестування:

- Випробування диму;
- Модульне тестування;
- Інтеграційне тестування;
- Функціональне тестування;
- Тестування ключових слів;
- Регресійне тестування;
- Тестування на основі даних;
- Тестування чорного ящика.

Вибір правильного інструменту може бути складним завданням. Наступний критерій допоможе вам вибрати найкращий інструмент для ваших вимог:

- Екологічна підтримка;
- Простота використання;
- Тестування бази даних;
- Ідентифікація об'єкта;

- Тестування зображення;
- Тестування відновлення після помилок;
- Відображення об'єктів;
- Використана мова сценаріїв;
- Підтримка різних типів тестів – у тому числі функціональних, керування тестами, мобільних тощо;
- Підтримка кількох тестових фреймворків;
- Легко налагоджувати сценарії програмного забезпечення автоматизації;
- Здатність розпізнавати предмети в будь-якому середовищі;
- Великі звіти про випробування та результати;
- Мінімізуйте витрати на навчання обраних інструментів.

Вибір інструменту є однією з найбільших проблем, які необхідно вирішити перед тим, як переходити до автоматизації. По-перше, визначте вимоги, вивчіть різні інструменти та їх можливості, встановіть очікування від інструменту та перейдіть до перевірки концепції.

Висновки до розділу 1

Для упровадження на будь-якому проекті системи тестування потрібно зробити базову модель автоматизованого тестового фреймворку з відкритим для вдосконалень кодом.

Для втілення в життя цієї ідеї покращення правильним буде вибір керуватися даними правилами:

- Пріоритизувати потрібно покращення кінцевої якості продукту;
- В основі має лежати ступінь покриття тестами частин продукту;
- Автоматизований тестовий фреймворк має містити універсальну основу та легко модернізуватися;
- Повинні бути як і сценарії автоматизованого тестування так і ручного.

Серед найважливіших питань з підходу до побудови тестового фреймворку і тестування проекту в цілому можна виділити наступні:

- Розгляд вимог до продукту та створення загальної стратегії тестування ще на початкових етапах SDLC;
- Співпраця всієї команди розробки та замовника з тестувальниками;
- Розгляд шляхів оптимізації та покращення створених тест плану, стратегії, та сценаріїв.

Саме це допоможе покращити кінцеву якість продукту ще на початкових етапах його розробки та зменшити витрати на реалізацію та вірно обрати шлях яким рухатися з подальшими покращеннями для тестування та розробки автоматизованого тестового фреймворку.

РОЗДІЛ 2.
ЗАСОБИ ТА ТЕХНОЛОГІЇ СТВОРЕННЯ АВТОМАТИЗОВАНИХ ПРОДУКТІВ ТЕСТУВАННЯ

Підходів та засобів до тестування продуктів дуже багато, тому перш ніж зробити вибір, потрібно мати уявлення про найбільш використовувані. При цьому потрібно розуміти переваги та недоліки.

2.1. Введення в засоби автоматизованого тестування

Важливо розуміти, кожен засіб тестування пропонує різноманітні застосунки та підходи, які надають потрібний рівень якості продукту. Правильно обрані підходи дозволять оптимально витратити час та зусилля витрачені на тестування та безпосередню автоматизацію, покращити взаємодію між тестувальниками, розробниками та замовником і покращити досвід кінцевих користувачів та їх загальне враження від продукту.

2.2. Мови програмування для створення тестових автоматизованих фреймворків

Нині існує досить велике різноманіття мов програмування, які дозволяють покрити тестами продукт. Найбільш поширеними та зручними у використанні є Java, C#, Python, JavaScript, Go.

2.2.1. Мова програмування Java

Мова програмування Java була розроблена Sun Microsystems на початку 1990-х років. Незважаючи на те, що Java в основному використовується для Інтернет-програм, вона є простою, ефективною мовою загального призначення. Java спочатку була розроблена для вбудованих мережевих програм, що працюють на кількох платформах. Це портативна, об'єктно-орієнтована, інтерпретована мова.

Java надзвичайно портативна. Той самий Java-додаток працюватиме однаково на будь-якому комп'ютері, незалежно від апаратного забезпечення чи операційної системи, якщо він має інтерпретатор Java. Окрім портативності, ще однією з ключових переваг Java є її набір функцій безпеки, які захищають ПК, на якому запущено програму Java, не лише від проблем, викликаних помилковим кодом, але й від шкідливих програм (таких як віруси). Ви можете безпечно запускати аплет Java, завантажений з Інтернету, оскільки функції безпеки Java запобігають доступу цих типів аплетів до жорсткого диска комп'ютера або мережевих з'єднань. Аплет – це, як правило, невелика програма Java, вбудована в сторінку HTML.

Java можна вважати як скомпільованою, так і інтерпретованою мовою, оскільки її вихідний код спочатку компілюється в двійковий байт-код. Цей байт-код працює на віртуальній машині Java (JVM), яка зазвичай є програмним інтерпретатором. Використання скомпільованого байт-коду дозволяє інтерпретатору (віртуальній машині) бути малим і ефективним (і майже таким же швидким, як центральний процесор, що виконує власний скомпільований код). Крім того, цей байт-код надає Java її переносимість: вона працюватиме на будь-якій JVM, яка правильно реалізована, незалежно від комп'ютерної апаратної чи програмної конфігурації. Більшість веб-браузерів (таких як Microsoft Internet Explorer або Netscape Communicator) містять JVM для запуску Java-аплетів [7].

Порівняно з C++ (іншою об'єктно-орієнтованою мовою), код Java працює трохи повільніше (через JVM), але він більш портативний і має набагато кращі функції безпеки. Віртуальна машина забезпечує ізоляцію між ненадійною програмою Java і ПК, на якому

запущено програмне забезпечення. Синтаксис Java подібний до C++, але мови досить різні. Наприклад, Java не дозволяє програмістам реалізувати перевантаження операторів, тоді як C++ дозволяє. Крім того, Java є динамічною мовою, де ви можете безпечно змінювати програму під час її роботи, тоді як C++ це не дозволяє. Це особливо важливо для мережевих програм, які не можуть дозволити собі будь-які простої. Крім того, усі базові типи даних Java є попередньо визначеними та не залежать від платформи, тоді як деякі типи даних можуть змінюватися залежно від платформи, що використовується в C або C++ (наприклад, тип int).

Програми Java структуровані краще, ніж еквіваленти C++. Усі функції (або методи Java) і виконувані оператори в Java повинні знаходитися в межах класу, тоді як C++ дозволяє визначенням функцій і рядкам коду існувати поза класами (як у програмах у стилі C). Глобальні дані та методи не можуть перебувати поза класом у Java, тоді як C++ це дозволяє. Ці обмеження, хоч часом і обтяжливі, допомагають підтримувати цілісність і безпеку програм Java і змушують їх бути повністю об'єктно-орієнтованими.

Іншою ключовою особливістю Java є те, що це відкритий стандарт із загальнодоступним вихідним кодом. Sun Microsystems контролює мову Java та пов'язані з нею продукти, але ліберальна ліцензійна політика Sun сприяла тому, що Інтернет-спільнота прийняла Java як стандарт. Ви можете безкоштовно завантажити всі інструменти, необхідні для розробки та запуску Java-аплетів і додатків, із веб-сайту Sun Java.

```
public class AverageProgram     // start of class definition
{
 public static void main(String[] args)
                                 // start of method definition
 {
   int npoints, counter, acc, average;     // declare variables

   System.out.println("Enter the number of points to average: ")
   npoints = ConsoleIn.readInt();  // read npoints
   counter = 0;                    // initialize variables
   acc = 0;
   while (counter <npoints)
    {                              // start of while loop
     System.out.println("Enter value: ");
     acc = acc + ConsoleIn.readInt();    // add in current valu
     counter = counter + 1;              // increment counter
    }                                    // end of while loop
   average = acc / npoints;              // calculate average
   System.out.println("Average value = " + average);
                                         // display result
 }                           // end of method definition
}                            // end of class definition
```

Рис. 2.1. Приклад програми мовою Java яка усереднює числа

У цьому прикладі клас AverageProgram (який є програмою) містить лише один метод (функцію), main(). Зверніть увагу, що більша частина синтаксису така сама, як C або C++, включаючи розділювачі коментарів: у Java можна використовувати роздільники стилю C (/* */) або C++ (//). Навіть оператор while() працює так само, як і в C/C++. Виведення на екран виконується за допомогою System.out.println(), де println() є викликаним методом стандартного об'єкта Java System.out. У Java також є об'єкт System.in для читання з клавіатури, але він має бути оброблений, щоб бути корисним. У цьому прикладі передбачається, що ConsoleIn є попередньо визначеним класом (який використовує System.in), який містить метод Readlnt() для читання цілочисельного значення.

Платформа Java — це набір програм, які допомагають програмістам ефективно розробляти та запускати програми програмування на Java. Він включає механізм виконання, компілятор і набір бібліотек. Це набір комп'ютерного програмного забезпечення та специфікацій. Джеймс Гослінг розробив платформу Java в Sun Microsystems, а пізніше її придбала корпорація Oracle.

Java — це багатоплатформна, об'єктно-орієнтована та мережево-орієнтована мова. Це одна з найбільш використовуваних мов програмування. Java також використовується як обчислювальна платформа.

Вона вважається однією зі швидких, безпечних і надійних мов програмування, якій надає перевагу більшість організацій для створення своїх проектів.

Для чого використовується Java?

Ось кілька важливих програм Java:

- Він використовується для розробки додатків Android;
- Допомагає створювати корпоративне програмне забезпечення;
- Широкий вибір мобільних java-додатків;
- Наукові обчислювальні програми;
- Використовуйте для Big Data Analytics;
- Програмування апаратних пристроїв на Java;

- Використовується для серверних технологій, таких як Apache, JBoss, GlassFish тощо.

Історія мови програмування Java

Ось важливі віхи з історії мови Java:

- Мова Java спочатку називалася OAK;
- Спочатку він був розроблений для роботи з портативними пристроями та приставками. Oak зазнав серйозної невдачі;
- У 1995 році Sun змінила назву на «Java» і змінила мову, щоб скористатися перевагами розвитку бізнесу www (World Wide Web), що розвивається;
- Пізніше, у 2009 році, корпорація Oracle придбала Sun Microsystems і стала власником трьох ключових програмних активів Sun: Java, MySQL і Solaris.

Ось деякі важливі функції Java:

- Це одна з простих у використанні мов програмування для вивчення;
- Напишіть код один раз і запустіть його майже на будь-якій обчислювальній платформі;
- Java не залежить від платформи. Деякі програми, розроблені на одній машині, можуть бути виконані на іншій машині;
- Він призначений для створення об'єктно-орієнтованих програм;
- Це багатопотокова мова з автоматичним керуванням пам'яттю;
- Він створений для розподіленого середовища Інтернет;
- Сприяє розподіленим обчисленням, оскільки він орієнтований на мережу.

Програміст на Java пише програму мовою, зрозумілою людині, яка називається вихідним кодом. Тому центральний процесор або чіпи ніколи не розуміють вихідний код, написаний будь-якою мовою програмування.

Ці комп’ютери або чіпи розуміють лише одну річ, яка називається машинною мовою або кодом. Ці машинні коди виконуються на рівні ЦП. Тому для інших моделей ЦП будуть різні машинні коди.

Однак вам слід потурбуватися про машинний код, оскільки програмування пов’язане з вихідним кодом. Машина розуміє цей вихідний код і перекладає його в зрозумілий машині код, який є виконуваним кодом.

Усі ці функції відбуваються в наступних 3 компонентах платформи Java

Набір для розробки Java (JDK)

JDK — це середовище розробки програмного забезпечення, яке використовується для створення аплетів і програм Java. Повною формою JDK є Java Development Kit. Розробники Java можуть використовувати його в Windows, macOS, Solaris і Linux. JDK допомагає їм кодувати та запускати програми Java. На одному комп’ютері можна інсталювати більше однієї версії JDK.

Ось основні причини використання JDK:

- JDK містить інструменти, необхідні для написання програм на Java та JRE для їх виконання;
- Він включає компілятор, засіб запуску додатків Java, Appletviewer тощо;
- Компілятор перетворює код, написаний на Java, у байт-код;
- Засіб запуску програм Java відкриває JRE, завантажує необхідний клас і виконує його головний метод.

Віртуальна машина Java (JVM)

Віртуальна машина Java (JVM) — це механізм, який забезпечує середовище виконання для керування кодом Java або програмами. Він перетворює байт-код Java на машинну мову. JVM є частиною Java Run Environment (JRE). В інших мовах програмування компілятор створює машинний код для конкретної системи. Однак компілятор Java створює код для віртуальної машини, відомої як віртуальна машина Java.

Ось важливі причини використання JVM:

- JVM забезпечує незалежний від платформи спосіб виконання вихідного коду Java;
- Він має численні бібліотеки, інструменти та фреймворки;
- Запустивши програму Java, ви можете працювати на будь-якій платформі та заощадити багато часу;
- JVM поставляється з компілятором JIT (Just-in-Time), який перетворює вихідний код Java на машинну мову низького рівня. Таким чином, він працює швидше, ніж звичайний додаток.

Java Runtime Environment (JRE)

JRE — це частина програмного забезпечення, призначена для запуску іншого програмного забезпечення. Він містить бібліотеки класів, клас завантажувача та JVM. Простіше кажучи, якщо ви хочете запустити програму на Java, вам потрібна JRE. Якщо ви не програміст, вам не потрібно встановлювати JDK, а лише JRE для запуску програм на Java.

Ось основні причини використання JRE:

- JRE містить бібліотеки класів, JVM та інші допоміжні файли. Він не містить жодних інструментів для розробки Java, таких як налагоджувач, компілятор тощо;
- Він використовує такі важливі класи пакетів, як бібліотеки math, swing, util, lang, awt і runtime;
- Якщо вам потрібно запустити Java-аплети, у вашій системі має бути встановлено JRE.

За допомогою віртуальної машини Java цю проблему можна вирішити. Але як це працює на різних процесорах і ОС. Розберемося в цьому процесі крок за кроком.

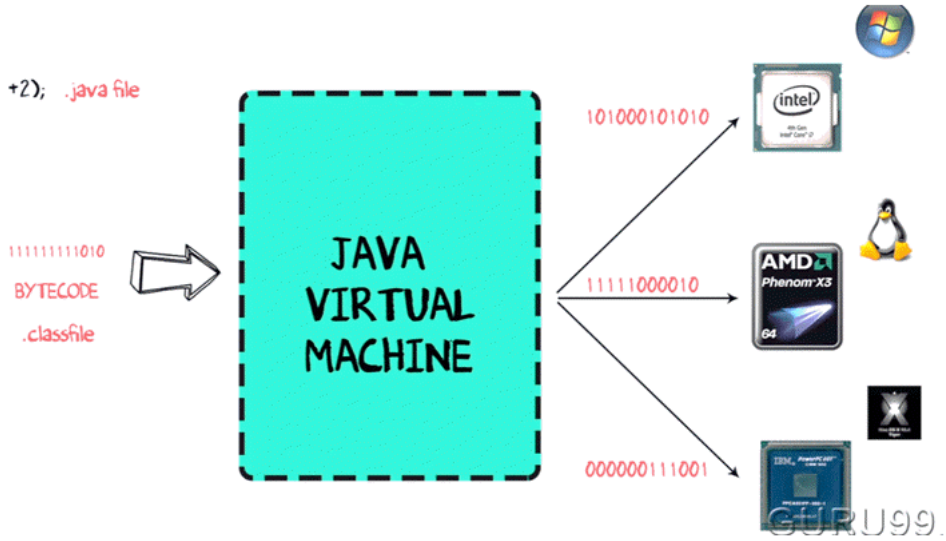

Рис.2.2. Графічне зображення JVM

Крок 1) Код для відображення додавання двох чисел – System.out.println(1+2) і збережений як файл .java.

Крок 2) За допомогою компілятора Java код перетворюється на проміжний код, який називається байт-кодом. Результатом є файл .class.

Крок 3) Цей код не розуміє жодна платформа, а лише віртуальна платформа під назвою віртуальна машина Java.

Крок 4) Ця віртуальна машина знаходиться в оперативній пам'яті вашої операційної системи. Коли віртуальна машина отримує цей байт-код, вона визначає платформу, на якій працює, і перетворює байт-код у рідний машинний код.

Під час роботи на комп'ютері чи перегляду веб-сторінок, коли ви бачите одну з цих піктограм, будьте впевнені, що віртуальна машина Java завантажується у вашу оперативну пам'ять. Але те, що робить Java прибутковою, так це те, що після компіляції код може працювати не лише на всіх платформах ПК, але й на мобільних телефонах чи інших електронних гаджетах, які підтримують Java.

Версії Java

Version	Release date	End of Free Public Updates[3][8][9][10]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	September 2003	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
Java SE 5	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018 December 2026 for Azul[11]
Java SE 7	July 2011	July 2019	July 2022
Java SE 8 (LTS)	March 2014	**March 2022 for Oracle (commercial)** December 2030 for Oracle (non-commercial) December 2030 for Azul May 2026 for IBM Semeru[12] At least May 2026 for Eclipse Adoptium At least May 2026 for Amazon Corretto	December 2030[13]
Java SE 9	September 2017	March 2018 for OpenJDK	—
Java SE 10	March 2018	September 2018 for OpenJDK	—
Java SE 11 (LTS)	September 2018	September 2026 for Azul 024 for IBM Semeru[12] At least October 2024 for Eclipse Adoptium At least September 2027 for Amazon Corretto At least October 2024 for Microsoft[14][15]	September 2026 September 2026 for Azul[11]
Java SE 12	March 2019	September 2019 for OpenJDK	—
Java SE 13	September 2019	March 2020 for OpenJDK	—
Java SE 14	March 2020	September 2020 for OpenJDK	—
Java SE 15	September 2020	March 2021 for OpenJDK March 2023 for Azul[11]	—
Java SE 16	March 2021	September 2021 for OpenJDK	—
Java SE 17 (LTS)	September 2021	September 2029 for Azul October 2027 for IBM Semeru[12] At least September 2027 for Microsoft[14] At least September 2027 for Eclipse Adoptium	September 2029 or later September 2029 for Azul
Java SE 18	March 2022	September 2022 for OpenJDK and Adoptium	—
Java SE 19	September 2022	March 2023 for OpenJDK	—
Java SE 20	March 2023	September 2023 for OpenJDK	—
Java SE 21 (LTS)	September 2023	September 2028	September 2031[13]

Older version, yet still maintained

Рис2.3. Версії мови програмування Java

2.2.2. Мова програмування C#

C# — це сучасна, об’єктно-орієнтована мова програмування загального призначення, яка вимовляється як «C Sharp». Він був розроблений Microsoft під керівництвом Андерса Хейлсберга та його команди в рамках ініціативи .NET і був схвалений Європейською асоціацією виробників комп’ютерів (ECMA) і Міжнародною організацією стандартів (ISO). C# є однією з мов для спільної мовної інфраструктури. Синтаксично C# дуже схожий на Java і простий для користувачів, які мають знання C, C++ або Java.

.NET — це програмна основа, розроблена та розроблена корпорацією Майкрософт. Першою версією фреймворку .Net була 1.0, яка вийшла в 2002 році. Простіше кажучи, це віртуальна машина для компіляції та виконання програм, написаних різними мовами, такими як C#, VB.Net тощо.

Він використовується для розробки додатків на основі форм, веб-додатків і веб-служб. На платформі .Net доступні різні мови програмування, найпоширенішими з яких є VB.Net і C#. Він використовується для створення програм для Windows, телефонів, Інтернету тощо. Він надає багато функцій, а також підтримує галузеві стандарти.

.NET Framework підтримує понад 60 мов програмування, у яких 11 мов програмування розроблено та розроблено Microsoft. Інші мови, що не належать Microsoft, підтримуються .NET Framework, але не розроблені та не розроблені Microsoft.

Розрізняють три важливі фази розвитку технології .NET:

- Технологія OLE;
- Технологія COM;
- Технологія .NET.

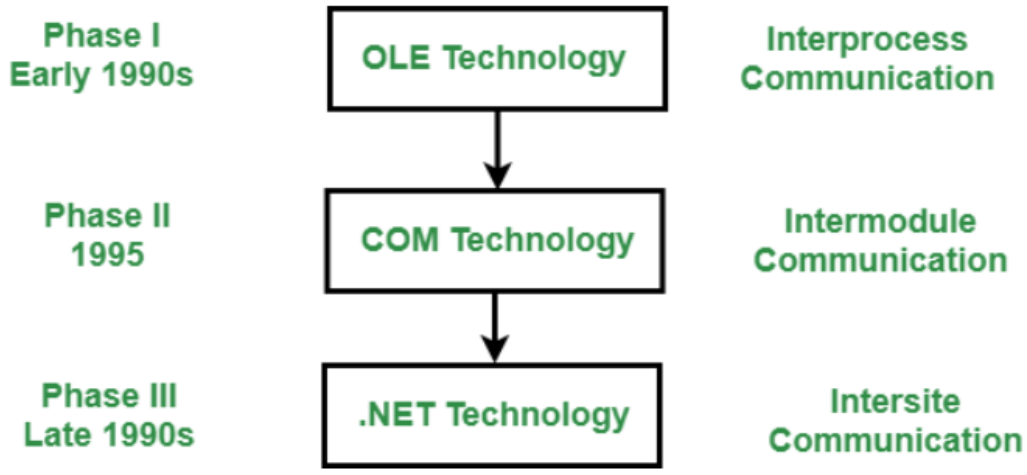

Рис. 2.4. Історія розвитку .Net

Технологія OLE: OLE (зв’язування та вбудовування об’єктів) — це одна з технологій компонентного документа Microsoft. По суті, його основна мета полягає в тому, щоб зв’язати елементи з різних програм один в одному.

Технологія COM: технологія сімейства операційної системи Microsoft Windows Microsoft COM (Common Object Model) забезпечує взаємодію між різними програмними компонентами. COM здебільшого використовується розробниками для різних цілей, таких як створення багаторазових компонентів програмного забезпечення, зв’язування компонентів разом для створення програм, а також використання переваг служб Windows. Об’єкти COM можна створювати за допомогою широкого діапазону мов програмування.

Технологія .NET: технологія .NET колекції або набору технологій для розробки вікон і веб-додатків. Технологія .Net розроблена корпорацією Майкрософт і була запущена в лютому 2002 року, за основним визначенням, новою стратегією Microsoft в Інтернеті. Спочатку він називався NGWS (веб-сервіси наступного покоління). Вважається, що це одна з потужних, популярних і дуже корисних Інтернет-технологій, доступних сьогодні.

Основні компоненти .NET Framework

Common Language Runtime (CLR): CLR є основним компонентом віртуальної машини .NET Framework. Це середовище виконання в .NET Framework, яке запускає коди та допомагає полегшити процес розробки, надаючи різноманітні послуги, такі як дистанційне керування, керування потоками, безпека типів, керування пам’яттю, надійність тощо. По суті, це відповідає за керування виконанням програм .NET незалежно від мови програмування .NET. Це також допомагає керувати кодом, оскільки код, націлений на середовище виконання, називається керованим кодом, а код, який не націлений на середовище виконання, називається некерованим кодом [8].

Framework Class Library (FCL): це набір повторно використовуваних об’єктно-орієнтованих бібліотек класів і методів тощо, які можна інтегрувати з CLR. Також називаються Асамблеями. Це так само, як файли заголовків у C/C++ і пакунки в java.

Інсталяція .NET Framework в основному полягає в установці CLR і FCL у систему. Нижче наведено огляд .NET Framework

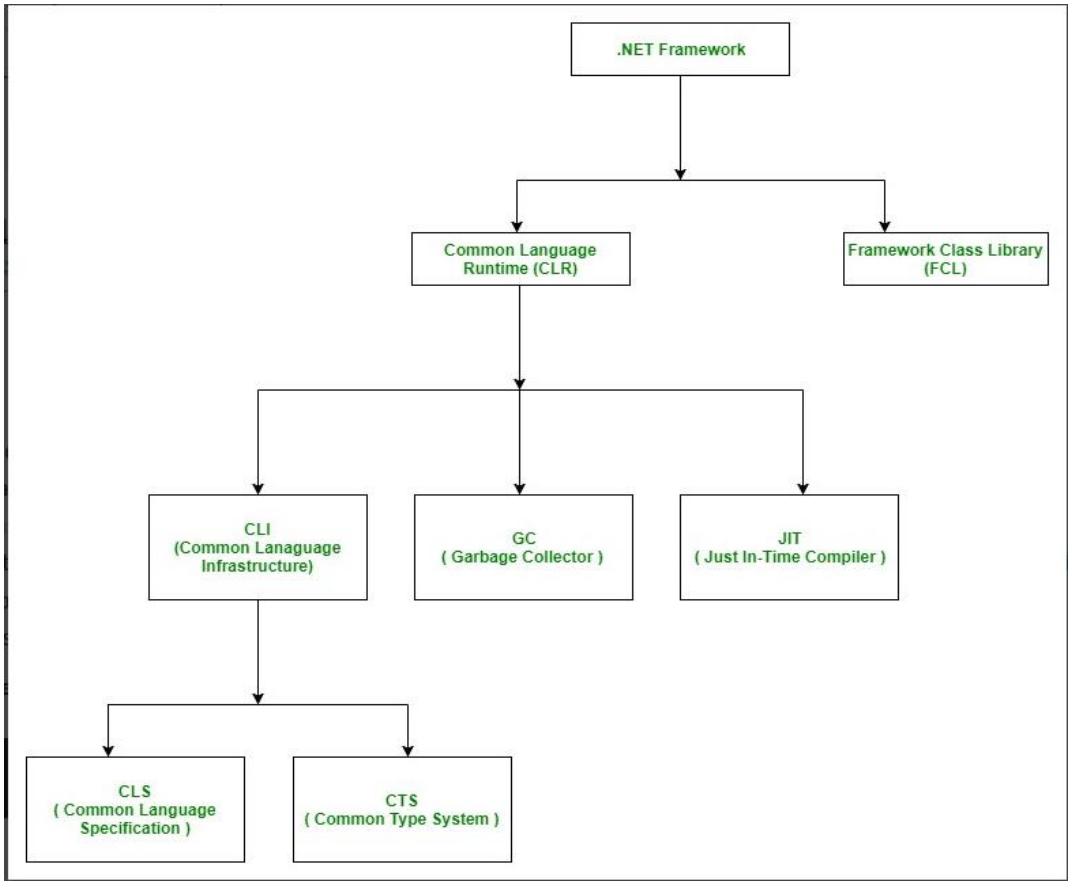

Рис. 2.5. Архітектура .Net framework

Поєднання архітектури операційної системи та архітектури ЦП відоме як платформа. Залежність від платформи означає, що код мови програмування працюватиме лише в певній операційній системі. Програма .NET залежить від платформи через платформу .NET, яка може працювати лише в операційній системі Windows. Програма .Net є незалежною від платформи також через структуру Mono. Використовуючи структуру Mono, програма .Net може працювати в будь-якій операційній системі, включаючи Windows. Mono framework — це стороннє програмне забезпечення, розроблене компанією Novell, яка зараз є частиною компанії Micro Focus. Це платний фреймворк.

Важливі моменти:

- Visual Studio — це інструмент розробки, який використовується для проектування та розробки програм .NET. Щоб використовувати Visual Studio, користувач повинен спочатку інсталювати платформу .NET у системі;
- У старішій версії ОС Windows, як-от XP SP1, SP2 або SP3, платформа .NET була інтегрована з інсталяційним носієм;
- Windows 8, 8.1 або 10 не надають попередньо встановленої версії .NET Framework 3.5 або новішої. Тому версію, вищу за 3.5, потрібно інсталювати з інсталяційного носія Windows або з Інтернету за запитом. Оновлення Windows надасть рекомендації щодо встановлення .NET framework.

2.3. Застосування для автоматизації мови програмування Java

2.3.1. Selenium

Selenium — це безкоштовна (з відкритим вихідним кодом) система автоматизованого тестування, яка використовується для перевірки веб-додатків у різних браузерах і платформах. Для створення тестових сценаріїв Selenium можна використовувати кілька мов програмування, наприклад Java, C#, Python тощо. Тестування, виконане за допомогою інструменту тестування Selenium, зазвичай називають Selenium Testing [9].

Програмне забезпечення Selenium — це не просто окремий інструмент, а набір програмного забезпечення, кожна частина якого задовольняє різні потреби організації в тестуванні Selenium QA. Ось список інструментів:

- Інтегроване середовище розробки Selenium (IDE);
- Selenium Remote Control (RC);
- WebDriver;
- Selenium grid.

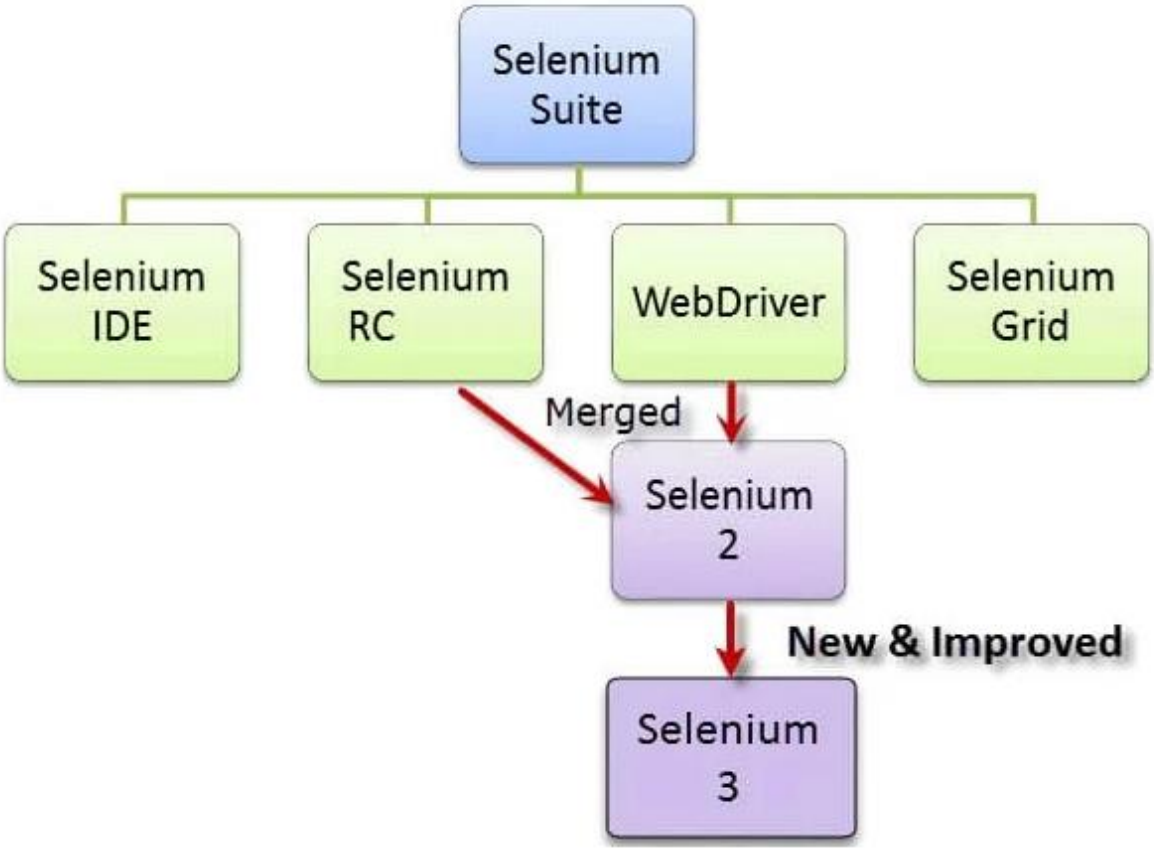

Рис.2.6. Сім’я Selenium

WebDriver виявився кращим за Selenium IDE та Selenium RC у багатьох аспектах. Він реалізує більш сучасний і стабільний підхід до автоматизації дій браузера. WebDriver, на відміну від Selenium RC, не покладається на JavaScript для автоматизованого тестування Selenium. Він керує браузером шляхом прямого зв’язку з ним.

Підтримувані мови такі ж, як і в Selenium RC:

- Java;
- C#;
- PHP;
- Python;
- Perl.

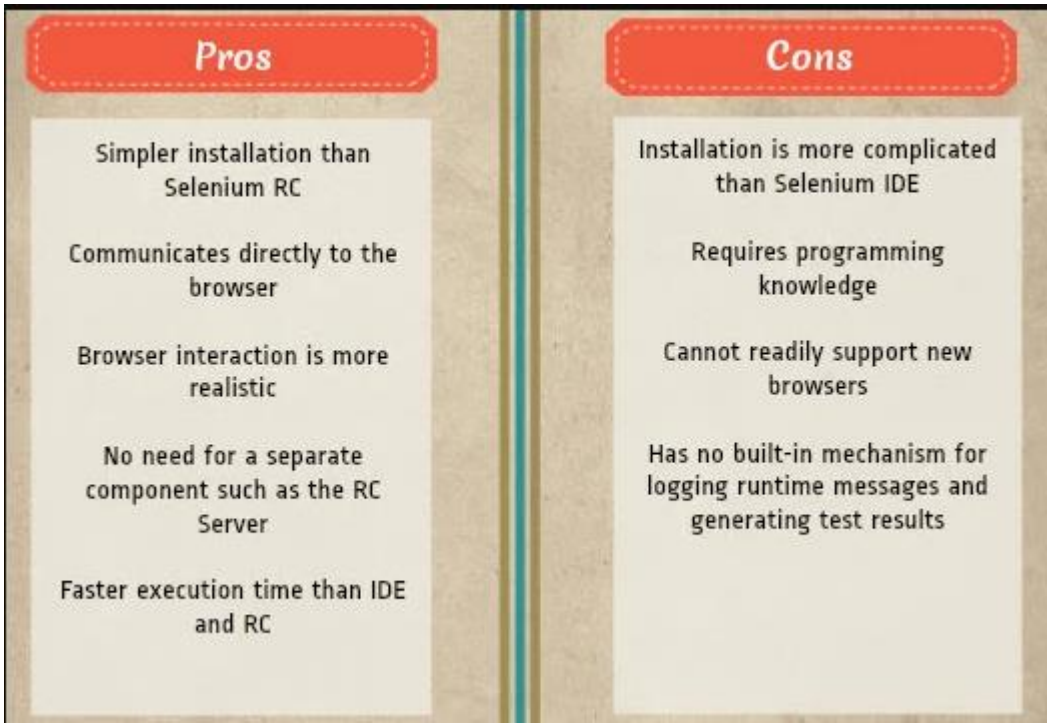

Рис.2.7. Плюси та мінуси сім'ї Selenium

2.3.2. Cucumber

Cucumber — це інструмент тестування, який підтримує розвиток, керований поведінкою (BDD). Він пропонує спосіб написання тестів, який може зрозуміти кожен, незалежно від його технічних знань. У BDD користувачі (бізнес-аналітики, власники продуктів) спочатку пишуть сценарії або приймальні тести, які описують поведінку системи з точки зору клієнта, для перегляду та підписання власниками продукту, перш ніж розробники напишуть свої коди. Framework Cucumber використовує мову програмування Ruby. [10]

Вважайте, що вам призначено створити модуль переказу коштів у програмі Net Banking.Є кілька способів перевірити його в рамках Cucumber Testing :

- Переказ коштів повинен здійснюватися, якщо на початковому рахунку є достатній баланс;
- Переказ коштів має бути здійснено, якщо деталі кондиціонера призначення

правильні;

- Переказ коштів має відбутися, якщо пароль транзакції / код RSA / автентифікація безпеки для транзакції, введені користувачем, правильні;
- Переказ коштів має здійснюватися, навіть якщо це вихідний день;
- Переказ коштів має відбутися в дату в майбутньому, встановлену власником рахунку.

Сценарій тестування стає більш детальним і складним, оскільки ми розглядаємо додаткові функції, такі як сума переказу X для інтервалу Y днів/місяців, зупинка переказу за розкладом, коли загальна сума досягає Z тощо

Загальна тнденція розробників полягає в тому, щоб розробляти функції та писати тестовий код пізніше. Як видно з наведеного вище випадку, розробка тестового прикладу для цього випадку є складною, і розробник відкладе тестування до випуску, після чого він проведе швидке, але неефективне тестування.

Щоб подолати цю проблему, був розроблений Cucumber BDD (Behavior Driven Development). Це полегшує весь процес тестування для розробника

У Cucumber BDD все, що ви пишете, має проходити за кроками «Дано-коли-тоді». Давайте розглянемо той же приклад вище в BDD

Given that a fund transfer module in net banking application has been developed
And I am accessing it with proper authentication
WhenI shall transfer with enough balance in my source account
Or I shall transfer on a Bank Holiday
Or I shall transfer on a future date
And destination a/c details are correct
And transaction password/RSA code/security authentication for the transaction is correct
And press or click send button
Then amount must be transferred
And the event will be logged in log file

Чи не легко писати, читати і розуміти? Він охоплює всі можливі тестові випадки для модуля переказу коштів і може бути легко модифікований для розміщення більшої кількості. Крім того, це більше схоже на написання документації для модуля переказу коштів.

Переваги Cucumber Software:

- Корисно залучати бізнес-стейкхолдерів, які не можуть легко читати код;
- Інструмент Cucumber Testing зосереджений на досвіді кінцевого користувача;
- Стиль написання тестів дозволяє легше використовувати код у тестах;
- Швидке та просте налаштування та виконання;
- Інструмент тестування огірка є ефективним інструментом для тестування.

2.4. Rest Assured

Rest Assured дозволяє тестувати REST API за допомогою бібліотек Java і добре інтегрується з Maven. Він має дуже ефективні методи зіставлення, тому стверджувати очікувані результати також досить просто. Rest Assured має методи для отримання даних майже з кожної частини запиту та відповіді, незалежно від того, наскільки складними є структури JSON [11].

Для спільноти тестувальників API Automation Testing все ще є новим і нішевим. Складність JSON залишає тестування API недослідженим. Але це не робить його менш важливим у процесі тестування. Будьте певні. Фреймворк Rest Assured.io зробив це дуже простим, використовуючи базові основи Java, що робить його дуже бажаним для вивчення.

Уявіть, що ви відкриваєте карту Google і шукаєте місце, куди хочете піти, і одразу бачите найближчі ресторани, ви бачите варіанти маршруту; від деяких провідних туристичних постачальників і побачите стільки варіантів у вас під рукою. Ми всі знаємо, що вони не є продуктами Google, тоді як Google вдається це показати. Вони використовують відкритий API цих постачальників. Тепер, якщо вас попросять

протестувати такий тип налаштувань, навіть до того, як користувальницький інтерфейс буде створено або знаходиться в стадії розробки, тестування API стає надзвичайно важливим, а їх повторне тестування з різними комбінаціями даних робить це дуже придатним випадком для автоматизації.

Раніше ми використовували динамічні мови, такі як groovy, ruby, щоб досягти цього, і це було складно. Тому тестування API не вивчалося функціональним тестуванням.

Але використання Rest Assured, автоматизоване тестування API, надсилання простих https-запитів із зручними налаштуваннями є простими, якщо ви маєте базові знання Java. Це необхідно для розуміння тестування API та інтеграційного тестування, але після цього автоматизація, будьте впевнені, дає дуже високу впевненість у серверній частині, тоді як зовнішнє тестування може зосереджуватися лише на інтерфейсі користувача та операціях на стороні клієнта. Будьте впевнені, це відкрите програмне забезпечення з великою кількістю додаткових методів і бібліотек, що робить його чудовим вибором для автоматизації API.

Синтаксис Rest Assured.io є найкрасивішою частиною, оскільки він дуже схожий на BDD і зрозумілий.

```
Given().
    param("x", "y").
    header("z", "w").
when().
Method().
Then().
    statusCode(XXX).
    body("x, ”y", equalTo("z"));
```

Given() Ключове слово Given дозволяє встановити фон, тут ви передаєте заголовки запитів, параметри запиту та шляху, тіло, файли cookie. Це необов’язково, якщо ці елементи не потрібні в запиті

When() Ключове слово «when» позначає передумову вашого сценарію. Наприклад, «коли» ви щось отримуєте/опублікуєте/кладете, зробіть щось інше.

Method() Замініть це будь-якою з операцій CRUD (get/post/put/delete)

Then() Ваші умови assert і matcher знаходяться тут

Rest Assured — це група бібліотек Java, які дозволяють автоматизувати тестування Rest API

Rest Assured базується на Java, і знання ядра Java достатньо, щоб її вивчити

Це допомагає отримати значення запиту та відповіді зі складних структур JSON

Запит API можна налаштувати за допомогою різноманітних заголовків, запитів, параметрів шляху та будь-якого сеансу чи файлів cookie, які потрібно встановити.

Це допомагає встановлювати твердження та умови.

Хоча Rest Assured дуже корисний, коли відповідь має тип JSON, його методи можуть не працювати бездоганно, якщо ідентифікатор типу вмісту HTML або звичайний текст.

Тут перелічено корисні поради щодо використання REST-assured, однієї з найпоширеніших Java-бібліотек для автоматизації тестування REST-API.

Всі життєві приклади, вони зібрані з моєї практики проведення code-review у більш ніж 50 проектах з автотестами.

Виносьте end-point'и в окреме місце

Здавалося б, це очевидно. Але ні, досить часто доводиться бачити код із захардшкіреними end-point'ами у запиті.

Найкраще виносити end-point'и у статичні константи фінального класу. При цьому варто уникати антипаттерну «константний інтерфейс» - це погана практика.

Не забувайте, що REST-assured дозволяє виносити параметри шляху, наприклад:

```
public final class EndPoints {
   public static final String users = "/users/{id}";
...
given().pathParams("id", someId).get(EndPoints.users)...
```

Також, якщо в багатьох запитах ви використовуєте один і той же базовий шлях, то буде гарною практикою винести його в готельну константу і передавати в basePath, наприклад:

```
// маємо наступну url програми http://host:port/appname/rest/someEndpoints
private static final basePath = "/appname/rest/";
..
// можемо задати базовий шлях на глобальному рівні,
// він застосовуватиметься всім запитам:
RestAssured.basePath = basePath;
// або на рівні одного запиту:
given().basePath(basePath)...
// або на рівні специфікації, але про це далі
```

Те ж саме стосується хоста і порту тестованого додатка.

ContentType/Accept

Ці заголовки використовуються практично у всіх HTTP-запитах. Автори REST-assured, розуміючи це, уможливили їх встановлення через виклик спеціальних методів:

```
// погана практика написання:
```

given().header("content-type", "application/json").header("accept", "application/json")...;

// хороша практика написання:

given().contentType(ContentType.JSON).accept(ContentType.JSON)...;

Хорошою практикою буде встановити ці заголовки в специфікації або на глобальному рівні. Це підвищить читабельність вашого коду.

StatusCode тощо.

REST-assured надає зручний синтаксис для перевірки кожної складової HTTP-відповіді, проте на практиці продовжуєш зустрічати подібний код:

// погана практика написання:

Response response = given()...when().get(someEndpoint);

Assert.assertEquals(200, response.then().extract().statusCode());

// хороша практика написання:

given()...when().get(someEndpoint).then().statusCode(200);

Використовуйте специфікації

Дублювання коду – це погано. Використовуйте специфікації для зменшення дублювання. У REST-assured можна створювати специфікації як запиту, так відповіді. У специфікацію запиту виносимо все, що можна продублювати в запитах.

RequestSpecification requestSpec = новий RequestSpecBuilder()

.setBaseUri("http://localhost")

.setPort(8080)

.setAccept(ContentType.JSON)

.setContentType(ContentType.ANY)

...

```
    .log(LogDetail.ALL)
    .build();
// можна задати одну специфікацію для всіх запитів:
RestAssured.requestSpecification = requestSpec;
// або окремого:
given().spec(requestSpec)...when().get(someEndpoint);
```

У специфікацію відповіді виносимо всі перевірки, які дублюються від запиту на запит.

```
ResponseSpecification responseSpec = новий ResponseSpecBuilder()
    .expectStatusCode(200)
    .expectBody(containsString("success"))
    .build();
// можна поставити одну специфікацію всім відповідей:
RestAssured.responseSpecification = responseSpec;
// або окремого:
given()...when().get(someEndpoint).then().spec(responseSpec)...;
```

Можна створювати кілька специфікацій для різних типів запитів/відповідей та використовувати у потрібній ситуації.

Не пишіть свої милиці для перетворення об'єктів

Не варто перетворювати свої POJO на JSON за допомогою Jackson ObjectMapper'a, а потім отриманий рядок передавати в тіло запиту. REST-assured чудово справляється з цим завданням. Для цього використовується той самий Jackson або Gson, залежно від того, що знаходиться в classpath. Для перетворення на XML використовується JAXB. Вихідний формат визначається автоматично за значенням Content-Type.

```
given().contentType(ContentType.JSON).body(somePojo)
    .when().post(EndPoints.add)
    .then()
    .statusCode(201);
// те саме працює і у зворотний бік:
SomePojo pojo = given().
    .when().get(EndPoints.get)
    .then().extract().body().as(SomePojo.class);
```

Крім того, REST-assured чудово справляється з перетворенням HashMap в JSON і назад.

Використовуйте всю потужність Groovy

Сама бібліотека REST-assured написана на Groovy і дозволяє застосовувати різні методи з Groovy до отриманої JSON/XML відповіді. Наприклад:

```
// методи find, findAll застосовуються до колекції для пошуку першого та всіх входжень, метод collect для створення нової колекції зі знайдених результатів.
// Змінна it створюється неявно і вказує на поточний елемент колекції
Map<String, ?> map = get(EndPoints.anyendpoint).path("rootelement.find { it.title =~ 'anythingRegExp'}");
```

Використання методів Groovy дозволяє сильно скоротити кількість коду, написаного вами для пошуку необхідного значення з відповіді.

2.5. Платформа Jenkins

Jenkins це сервер безперервної інтеграції з відкритим вихідним кодом, написаний на Java для організації ланцюжка дій для досягнення процесу безперервної інтеграції в

автоматизований спосіб. Дженкінс підтримує повний життєвий цикл розробки програмного забезпечення від створення, тестування, документування програмного забезпечення, розгортання та інших етапів життєвого циклу розробки програмного забезпечення.

Jenkins — широко використовувана програма в усьому світі, яка має близько 300 тисяч установок і зростає з кожним днем. Використовуючи Jenkins, компанії-розробники програмного забезпечення можуть прискорити процес розробки програмного забезпечення, оскільки Jenkins може швидко автоматизувати створення та тестування.

Це серверна програма, для якої потрібен веб-сервер, наприклад Apache Tomcat. Причина, по якій програмне забезпечення Jenkins стало таким популярним, полягає в його моніторингу повторюваних завдань, які виникають під час розробки проекту. Наприклад, якщо ваша команда розробляє проект, Дженкінс буде постійно тестувати збірки вашого проекту та показувати вам помилки на ранніх етапах розробки.

Що таке безперервна інтеграція?

Безперервна інтеграція – це процес багаторазової інтеграції змін коду від кількох розробників в один проект. Програмне забезпечення тестується одразу після фіксації коду. З кожним комітом коду код створюється та тестується. Якщо тест пройдено, збірка перевіряється для розгортання. Якщо розгортання успішне, код надсилається до виробництва.

Ця фіксація, збірка, тестування та розгортання є безперервним процесом, і тому назва безперервної інтеграції/розгортання.

Jenkins — це серверна програма, яка потребує веб-сервера, як-от Apache Tomcat, для роботи на різних платформах, таких як Windows, Linux, macOS, Unix тощо. Щоб використовувати Jenkins, вам потрібно створити конвеєри, які є серією кроків, які виконує сервер Jenkins. буде. Конвеєр безперервної інтеграції Jenkins — це потужний інструмент, який складається з набору інструментів, призначених для розміщення, моніторингу, компіляції та тестування коду або змін коду, наприклад:

- Сервер безперервної інтеграції (Jenkins, Bamboo, CruiseControl, TeamCity та інші);
- Інструмент керування джерелами (наприклад, CVS, SVN, GIT, Mercurial, Perforce, ClearCase та інші);
- Інструмент збірки (Make, ANT, Maven, Ivy, Gradle та інші);
- Платформа автоматизованого тестування (Selenium, Appium, TestComplete, UFT та інші).

Я впевнений, що всі ви знаєте про старий телефон Nokia. Раніше Nokia реалізувала процедуру під назвою нічна збірка. Після кількох комітів від різних розробників протягом дня програмне забезпечення збиралося щовечора. Оскільки програмне забезпечення створювалося лише раз на день, ізолювати, ідентифікувати та виправляти помилки у великій кодовій базі – це дуже важко.

Пізніше вони прийняли підхід безперервної інтеграції. Програмне забезпечення було створено та протестовано, щойно розробник зафіксував код. Якщо буде виявлено будь-яку помилку, відповідний розробник може швидко виправити її.

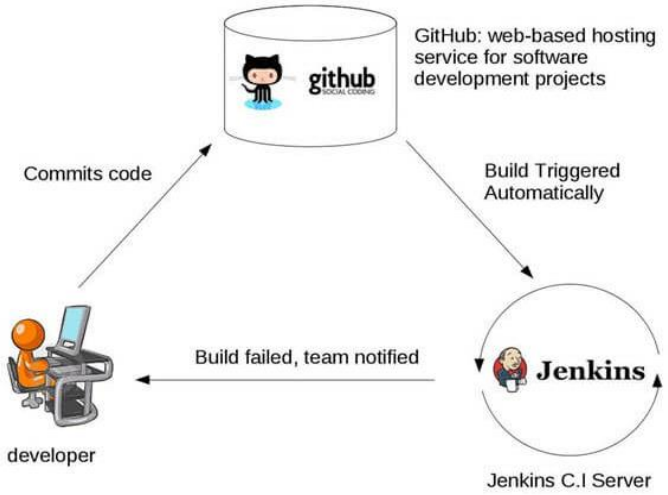

Рис 2.8. Візуалізація Jenkins

Переваги використання Jenkins:

- Дженкінсом керує спільнота, яка є дуже відкритою. Щомісяця вони проводять публічні зустрічі та беруть участь у розвитку проекту Jenkins від громадськості;
- Наразі закрито близько 280 квитків, і проект видає стабільний випуск кожні три місяці;
- З розвитком технологій зростає і Дженкінс. Наразі Jenkins опублікував близько 320 плагінів у своїй базі даних плагінів. З плагінами Jenkins стає ще потужнішим і багатшим на функції;
- Інструмент Jenkins також підтримує хмарну архітектуру, тому ви можете розгортати Jenkins на хмарних платформах;
- Причина, чому Дженкінс став популярним, полягає в тому, що він був створений розробником для розробників.

Недоліки використання Jenkins

Хоча Дженкінс є дуже потужним інструментом, він має свої недоліки.

- Його інтерфейс застарів і незручний у порівнянні з поточними тенденціями інтерфейсу користувача;
- Хоча Jenkins люблять багато розробників, підтримувати його не так просто, оскільки Jenkins працює на сервері та вимагає певних навичок адміністратора сервера, щоб контролювати його діяльність;
- Однією з причин, чому багато людей не впроваджують Jenkins, є складність встановлення та налаштування Jenkins;
- Безперервна інтеграція регулярно переривається через незначні зміни налаштувань. Постійну інтеграцію буде призупинено, тому розробник потребує певної уваги.

Висновки до розділу 2

У наш час кожен продукт та замовник прагне скоротити рівень витрат до мінімуму, а рівень якості кінцевого продукту підняти до максимуму. Однак не всі знають як це зробити за допомогою введення правильних підходів та методів тестування і автоматизації в частковому випадку. Тож, ідеальним варіантом в цьому випадку буде створити правильну стратегію тестування та використати правильні застосунки та спеціалістів.

Тестовий автоматизований фреймворк — це набір інструкцій або правил, які використовуються для створення та розробки тестових випадків. Фреймворк складається з комбінації практик та інструментів, розроблених, щоб допомогти фахівцям із забезпечення якості тестувати ефективніше.

Ці вказівки можуть включати стандарти кодування, методи обробки тестових даних, сховища об’єктів, процеси для зберігання результатів тестів або інформацію про те, як отримати доступ до зовнішніх ресурсів.

Хоча це не є обов’язковими правилами, і тестувальники все одно можуть писати сценарії або записувати тести, не дотримуючись їх, використання організованої структури зазвичай надає додаткові переваги, які в іншому випадку вони б втратили.Тестовий автоматизований фреймворк є зручним застосунком для полегшення рутинної роботи тестувальників. Важливим є саме правильне створення його структури та написання потрібних тест сценаріїв. Якщо все правильно зробити, то результати тестів та рівень «здоров’я» продукту зможе перевірити будь-яка особа, яка розуміє специфіку проекту завдяки зрозумілій, «людській» системі звітності.Це буде корисним не тільки для тестувальників чи керівництва, а і для девелоперів, які зможуть перевіряти якість своїх змін завдяки швидкому прогону юніт-тестів після кожного оновлення коду проекту.Розглянуті до цього атоматизовані фреймворки мають ряд певних недолік покращення яких буде розглянуто в наступному, 3 розділі дипломної роботи:

- Висока ціна побудови тестового фреймворку;

- Важкість підтримки без затрат часу кваліфікованими кадрами;
- Складність у освоєнні;
- Чистота та зрозумілість фреймворку;
- Незрозумілість найменувань;
- Складність вибору потрібних іструментів;
- Важкість актуалізації тестових сценаріїв.

РОЗДІЛ 3.
СТВОРЕННЯ ТЕСТОВОГО ФРЕЙМВОРКУ

Широке впровадження та залучення у життя проекту теставого автоматизованого фреймворку є корисним та актуальним у будь-який час. В час збільшення архітектурних розмірів та нарощування функціоналу додатку без їх застосування обійтися неможливо. Адже важко уявити яку кількість ручних тестувальників потрібно залучити для тестування онлайн застосунку вікіпедії або онлайн-гіпермаркету та скільки помилок вони можуть допустити через втому та рутинну роботу.

В загальному кажучи , перехід від щоденного ручного тестування до атоматизації рутини – цілком логічний етап для досягнення високого рівня кінцевої якості продукту. Однак важливо щоб фреймворк був якомога більш універсальним, захищеним, легко модифікуватися та не залежати від платформи чи браузера. В тому і полягає сенс розробки автоматизованого фреймворку, щоб розробити свою власну систему яка буде легко працювати з вашим проектом на усіх платформах і допоможе покращити його кінцеву якість.

3.1. Побудова загальної архітектура фреймворку

В основі нашого створюваного фреймворку буде лежати звичайний додаток мовою програмування Java, який ми будемо розширювати потрібними додатками та налаштуваннями

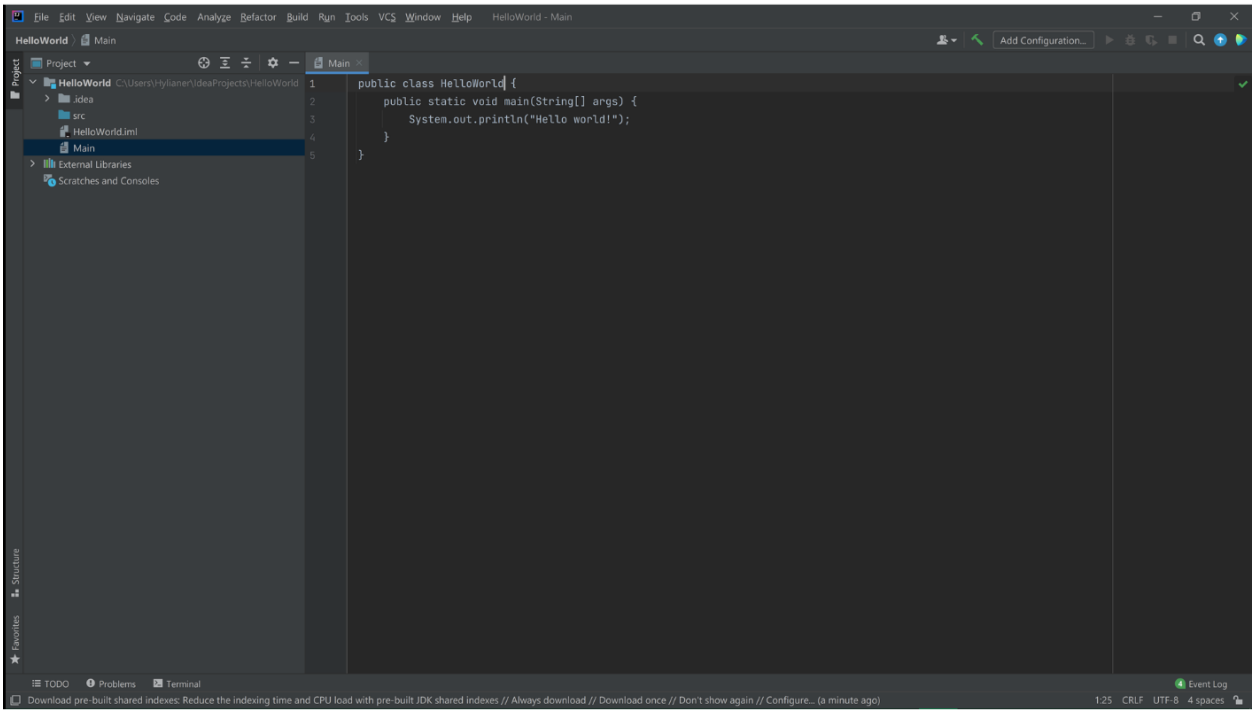

Рис.3.1. Приклад «порожнього» проекту в середовищі IntelliJ IDEA

Однак при створенні ми будемо використовувати більш розширену конфігурацію і створимо проект, який може модифікуватися за допомогою maven ресурсів.

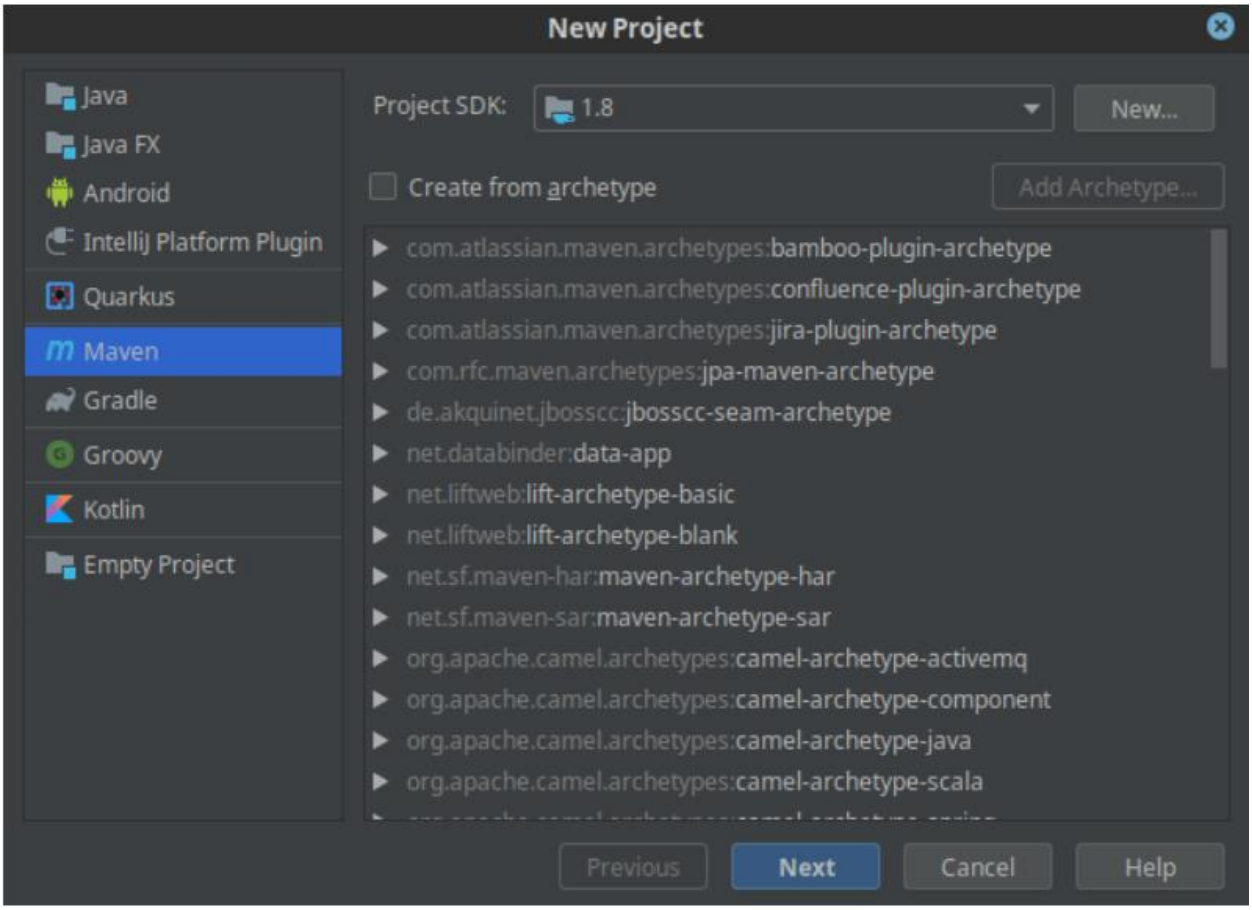

Рис.3.2. Приклад конфігурації Maven

При створенні такого проекту ми маємо можливість додавати до нього різноманітні налаштування, які підтримуються в даному середовищі розробки та даною мовою програмування просто додавши їх в pom. файл. Приклад цього файлу

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>

</project>
```

Рис.3.3. Приклад найпростішого pom. файлу

Додавши туди все потрібне та закінчивши налаштування проекту ми можемо перейти до написання безпосередніх класів нашого фреймворку.

3.1.1. Створення тестових сценарії для UI частини

Розглянемо один з чи не найважливіших класів Driver, який дозволяє нам створювати сутності браузерів для тестування інтерфейсу користувача. Його реалізацію приведено нижче

```
public class Driver {
    private static final Logger LOG = LoggerFactory.getLogger(Driver.class);

    private static final String WORKING_DIR = System.getProperty("user.dir");

    private static final String PATH_TO_DOWNLOADS = WORKING_DIR +
File.separator + "downloads";

    private static String browserType =

Optional.ofNullable(TestContext.getInstance().getBrowserType()).orElse(Browse
rType.CHROME).toLowerCase();

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

    private static List<WebDriver> storedDrivers = new ArrayList<>();

    public static WebDriver getDriver() {
        return DRIVER.get();
    }

    public static void removeDriver() {
        if (DRIVER.get() != null) {
            storedDrivers.remove(DRIVER.get());
            DRIVER.get().quit();
            DRIVER.remove();
        }
    }

    public static void addDriver() {
        String platformRunType =
Optional.ofNullable(System.getProperty("platform")).orElse("local").toLowerCa
se();
        WebDriver createdDriver = platformRunType.equals("remote") ?
getSelenoidDriver() : getLocalDriver();
        storedDrivers.add(createdDriver);
        DRIVER.set(createdDriver);
    }
```

```
    public static String getBrowserType() {
        return browserType;
    }

    static {
        Runtime.getRuntime().addShutdownHook(new Thread(() ->
storedDrivers.stream().forEach(WebDriver::quit)));
    }

    //
////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    // Driver creation
    //
////////////////////////////////////////////////////////////////////////////
//////////////////////////////

    public Driver() {
    }

    private static WebDriver getLocalDriver() {
        WebDriver builtDriver;
        switch (browserType) {
            case BrowserType.CHROME:
                builtDriver = buildChrome();
                break;
            case BrowserType.IE:
                builtDriver = buildIE();
                break;
            case BrowserType.SAFARI:
                builtDriver = buildSafari();
                break;
            default:
                browserType = BrowserType.CHROME;
                builtDriver = buildChrome();
        }
        builtDriver.manage().window().maximize();
        return builtDriver;
    }

    private static WebDriver getSelenoidDriver() {
        Stream.of(BrowserType.CHROME, BrowserType.FIREFOX).filter(bt ->
bt.equals(browserType)).findFirst().orElseThrow(
            () -> new IllegalArgumentException("Error! unsupported Selenoid
browser type: " + browserType));

        return buildSelenoid();
    }

    //
////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    // Chrome
    //
////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    private static WebDriver buildChrome() {
        LOG.info("Initializing Chrome driver...");
        ChromeOptions options = getChromeCustomOptions();
        WebDriverManager.chromedriver().setup();
        return new ThomsonReutersChromeDriver(options);
    }

    private static ChromeOptions getChromeCustomOptions() {
```

```
        ChromeOptions options = new ChromeOptions();
        Map<String, Object> prefs = new HashMap<>();

        //turn off message "Let's save your password for this site"
        prefs.put("credentials_enable_service", false);
        prefs.put("profile.password_manager_enabled", false);

prefs.put("profile.content_settings.exceptions.automatic_downloads.*.setting"
, 1);

        //
        prefs.put("download.default_directory", PATH_TO_DOWNLOADS);
        options.setExperimentalOption("prefs", prefs);

        //turn off message "Chrome is being controlled by automated test
software"
        options.setExperimentalOption("excludeSwitches",
Collections.singletonList("enable-automation"));
        options.setExperimentalOption("useAutomationExtension", false);
        //
        options.addArguments("--no-sandbox");
        options.addArguments("--disable-logging");
        options.addArguments("--start-maximized");
        options.addArguments("--disable-extensions");
        options.addArguments("--disable-web-security");
        options.addArguments("--disable-notifications");
        options.addArguments("--no-default-browser-check");

        options.setCapability(CapabilityType.HAS_NATIVE_EVENTS, true);
        options.setCapability(CapabilityType.SUPPORTS_JAVASCRIPT, true);
        return options;
    }

    //
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    // InternetExplorer
    //
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    private static WebDriver buildIE() {
        Capabilities capabilities = getIEDesiredCapabilities();
        WebDriverManager.iedriver().arch32().setup();
        return new ThomsonReutersInternetExplorerDriver(capabilities);
    }

    private static DesiredCapabilities getIEDesiredCapabilities() {
        DesiredCapabilities capabilities =
DesiredCapabilities.internetExplorer();
        capabilities.setCapability("requireWindowFocus", false);
        capabilities.setCapability(InternetExplorerDriver.NATIVE_EVENTS,
true);

capabilities.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNO
RING_SECURITY_DOMAINS, true);

capabilities.setCapability(InternetExplorerDriver.IE_ENSURE_CLEAN_SESSION,
true);

capabilities.setCapability(InternetExplorerDriver.ENABLE_PERSISTENT_HOVERING,
true);

capabilities.setCapability(InternetExplorerDriver.UNEXPECTED_ALERT_BEHAVIOR,
"accept");
```

```
capabilities.setCapability(InternetExplorerDriver.IGNORE_ZOOM_SETTING, true);

capabilities.setCapability(CapabilityType.ForSeleniumServer.ENSURING_CLEAN_SESSION, true);
        capabilities.setCapability("unhandledPromptBehavior", "dismiss");
        capabilities.setCapability("platformName", "windows");
        capabilities.setCapability("ignoreProtectedModeSettings", true);
        capabilities.setCapability("disable-popup-blocking", true);
        capabilities.setJavascriptEnabled(true);
        return capabilities;
    }

    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////
    // Safari
    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////
    private static WebDriver buildSafari() {
        LOG.info("Initializing Safari driver...");
        SafariOptions options = new SafariOptions();
        options.setCapability("browserName", "safari");
        return new ThomsonReutersSafariDriver(options);
    }
   }
```

Використовуючи цей клас ми можемо створювати сутності декількох найпоширеніших браузерів для нашого тестування графічного інтерфейсу користувача. Тепер розглянемо клас, який буде базовим у нашому фреймворку і від нього будуть унаслідуватися усі інші.

Базовий клас

```
       public class BasePage {

    private static final Logger LOG =
LoggerFactory.getLogger(BasePage.class);

    public static final String DOWNLOAD_PATH = getProperty("user.dir") +
separator + "downloads" + separator;

    public static String savedId;

    protected static final String GRID_STATUS = ".dgrid-status";

    protected static final String GLOBE_ICON_IN_LIST = ".icon-globe";

    protected static final String SUCCESS_MESSAGE = ".messageContent";

    protected static final String LOADING_INDICATOR = ".loadingSpinner";

    protected static final String DIALOG_UNDERLAY = ".dijitDialogUnderlay";

    private static final int DEFAULT_POLLING_IN_MILLS = 500;

    private static final int AFTER_SCENARIO_CLEANUP_WAIT = 2000;
```

```
    private static final int AFTER_SCENARIO_CLEANUP_ATTEMPTS = 60;

    private static final int AFTER_SCENARIO_CLEANUP_DB_CHECK_ATTEMPTS = 2;

    private static final String SHOWN = "shown";

    private static final String HIDDEN = "hidden";

    private static final String LOGO = ".logo";

    private static final String TEST = "test ";

    private static final String ASSIGNED_USER = "bdd testuser";

    private static final String PROXY_HOST_PROPERTY = "https.proxyHost";

    private static final String PROXY_PORT_PROPERTY = "https.proxyPort";

    private static final String ERROR_STATUS_MESSAGE =
            "Incorrect status parameter '%s', correct are: 'shown' or
'hidden'";

    private static final String VALIDATION_MESSAGE_ERROR =
"div.wrapper.error";

    private static final String CUSTOM_FIELD = "TESTCF" +
UUID.randomUUID().toString();

    private static final String VALIDATION_MESSAGE =
"div.wrapper.error,div.wrapper.success";

    private static final String STATUS_MESSAGE_LOCATOR = ".statusMessage
.messageContent";

    private static final By STATUS_MESSAGE_CSS =
By.cssSelector(".statusMessage .messageContent");

    private static final By MAIN_MENU_ITEMS =
            new By.ByXPath("//coral-tab[@level='3']//span");

    private static final String RESET_LOADER = ".CaseManager
.loadingIndicatorView:not(.hidden)";

    private static final String DIALOG_MESSAGE_RIGHT = ".dijitTooltipRight
.dijitTooltipContents";

    private static final String DIALOG_MESSAGE_BELOW =
".dijitTooltipDialogPopup .dijitTooltipContents";

    private static final String RUN_TIME_FILES_PATH = getProperty("user.dir")
+ separator + "testFiles" + separator;

    private static final String HOVER_OVER_ELEMENT_JS = "var evObj =
document.createEvent('MouseEvents');"
            + "evObj.initMouseEvent(\"mouseover\",true, false, window, 0, 0,
0, 0, 0, false, false, false, false, 0, null);"
            + "arguments[0].dispatchEvent(evObj);";

    private static final String CANCEL_HOVER_OVER_ELEMENT_JS = "var evObj =
document.createEvent('MouseEvents');"
            + "evObj.initMouseEvent(\"mouseout\",true, false, window, 0, 0,
0, 0, 0, false, false, false, false, 0, null);"
            + "arguments[0].dispatchEvent(evObj);";
```

```
    private static Map<String, String> testContext = new HashMap<>();

    private static Map<Object, Object> testSession = new HashMap<>();

    private static Map<String, String> groupIdReferenceMap = new HashMap<>();

    public Waiters waiters = new Waiters(Driver.getDriver());

    protected Actions action = new Actions(Driver.getDriver());

    private SetUpRest setupRest = new SetUpRest();

    private String host = getProperty("host");

    @FindBy(xpath = "//link[@rel='shortcut icon']")
    private WebElement favicon;

    @FindBy(css = "div.world-check-one-logo")
    private WebElement navbarLogo;

    @FindBy(css = ".statusMessage .messageContent")
    private WebElement statusMessage;

    @FindBy(css = ".wrapper span.close")
    private WebElement closePersistenceMessageIcon;

    @FindBy(xpath = "//coral-tab[@active='true']//span")
    private WebElement activePageText;

    @FindBy(css = ".windowApp")
    private WebElement frame;

    @FindBy(css = ".dijitAlignLeft .content.parent .name")
    private WebElement parentGroup;

    @FindBy(css = "a.whatsNew")
    private WebElement whatsNewLink;

    @FindBy(css = ".home-button")
    private WebElement homeIcon;

    public BasePage() {
        PageFactory.initElements(new ProjectFieldDecorator(new
DefaultElementLocatorFactory(Driver.getDriver())), this);
    }

    public String getPageTitle() {
        return Driver.getDriver().getTitle();
    }

    public Actions getAction() {
        return action;
    }

    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
    // Element
    //
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
    public WebElement findElement(By by) {
        return Driver.getDriver().findElement(by);
```

```
    }


    public List<WebElement> findElements(By by) {
        return Driver.getDriver().findElements(by);
    }

    //
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    // Element status
    //
////////////////////////////////////////////////////////////////////////////////
//////////////////////////////
    public boolean isElementPresent(By locator) {
        try {
            return findElement(locator) != null;
        } catch (NoSuchElementException e) {
            return false;
        }
    }

    public boolean isElementPresent(WebElement element) {
        try {
            return element.getLocation() != null;
        } catch (NullPointerException | NoSuchElementException |
StaleElementReferenceException e) {
            return false;
        }
    }

    public boolean isElementDisplay(WebElement webElement) {
        try {
            return webElement.isDisplayed();
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }

    public boolean isElementClickable(WebElement webElement) {
        try {
            return webElement.isDisplayed() && webElement.isEnabled();
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }

    public boolean isElementClickable(By locator) {
        try {
            return isElementClickable(findElement(locator));
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }

    public boolean isElementDisplay(By locator) {
        try {
            return findElement(locator).isDisplayed();
        } catch (NoSuchElementException e) {
            return false;
        }
    }

    public boolean isNestedElementDisplayed(WebElement parent, By by) {
```

```
        try {
            if (isElementPresent(parent.findElement(by))) {
                scrollIntoView(parent.findElement(by));
            }
            return isElementDisplay(parent.findElement(by));
        } catch (NoSuchElementException | StaleElementReferenceException e) {
            return false;
        }
    }

    public void verifyDisplayStatusOfElement(String displayStatus, By
locator) {
        switch (displayStatus) {
            case SHOWN:
                waiters.waitForElementToBeDisplay(locator);
                break;
            case HIDDEN:
                waiters.waitForElementToDisappear(locator);
                break;
            default:
                throw new
IllegalArgumentException(String.format(ERROR_STATUS_MESSAGE, displayStatus));
        }
    }

    public void verifyDisplayStatusOfElement(String displayStatus, WebElement
webElement) {
        switch (displayStatus) {
            case SHOWN:
                waiters.waitForElementToBeDisplay(webElement);
                break;
            case HIDDEN:
                waiters.waitForElementToDisappear(webElement);
                break;
            default:
                throw new
IllegalArgumentException(String.format(ERROR_STATUS_MESSAGE, displayStatus));
        }
    }

    //
////////////////////////////////////////////////////////////////////////////////
////////////////////
    // Selenium actions functionality
    //
////////////////////////////////////////////////////////////////////////////////
////////////////////
    public void enterText(WebElement element, CharSequence... dataToSend) {
        waiters.waitForElementToBeDisplay(element);
        waiters.waitForElementToBeClickable(element);
        element.clear();
        element.click();
        element.sendKeys(dataToSend);
    }

    // Workaround for stale element fix
    public void enterText(By by, CharSequence... dataToSend) {

waiters.waitForElementToBeDisplay(Driver.getDriver().findElement(by));

waiters.waitForElementToBeClickable(Driver.getDriver().findElement(by));
        try {
            Driver.getDriver().findElement(by).clear();
        } catch (InvalidElementStateException e) {
```

```
            LOG.error("Element isn't user editable");
        }
        Driver.getDriver().findElement(by).click();
        Driver.getDriver().findElement(by).sendKeys(dataToSend);
    }

    public void inputText(WebElement element, CharSequence... dataToSend) {
        waiters.waitForElementToBeDisplay(element);
        element.clear();
        element.sendKeys(dataToSend);
    }

    public void inputTextNoClean(WebElement element, CharSequence...
dataToSend) {
        waiters.waitForElementToBeDisplay(element);
        element.sendKeys(dataToSend);
    }

    public void clickOnElement(WebElement element) {
        try {
            waiters.waitForPresenceOfElement(element);
            waiters.waitForElementToBeClickable(element).click();
        } catch (WebDriverException e) {
            throw new IllegalStateException("WebDriver exception encountered:
" + e.getMessage(), e);
        }
    }

    public void clickOnElementWithDelay(By locator) {
        try {
            waiters.waitForElementToBeClickable(locator).click();
        } catch (WebDriverException e) {
            throw new IllegalStateException(
                    "Element found by locator '" + locator + "' not found on
the page: " + e.getMessage(), e);
        }
    }

    public void clickOnElement(By locator) {
        try {
            waiters.waitForElementToBeClickable(locator).click();
        } catch (WebDriverException e) {
            throw new IllegalStateException(
                    "Element found by locator '" + locator + "' not found on
the page: " + e.getMessage(), e);
        }
    }

    public void moveToElementUsingJS(WebElement element) {
        ((JavascriptExecutor)
Driver.getDriver()).executeScript(HOVER_OVER_ELEMENT_JS, element);
    }

    public void cancelMoveToElementUsingJS(WebElement element) {
        ((JavascriptExecutor)
Driver.getDriver()).executeScript(CANCEL_HOVER_OVER_ELEMENT_JS, element);
    }

    public void actionMoveToElement(WebElement element) {
        getAction().moveToElement(element).perform();
    }

    public void actionClickOnWebElement(WebElement element) {
        waiters.waitForElementToBeClickable(element);
```

```
        getAction().moveToElement(element).click(element).build().perform();
    }

    public void minimizeBrowserWindow() {
        Driver.getDriver().manage().window().setSize(new Dimension(300,
500));
    }

    public void maximizeBrowserWindow(){
        Driver.getDriver().manage().window().maximize();
    }

    public void switchToDefaultContent() {
        Driver.getDriver().switchTo().defaultContent();
    }

    public void switchToFrame() {
        waiters.waitForElementToBeDisplay(frame);
        Driver.getDriver().switchTo().frame(frame);
    }

    public void switchToFrameIfExists() {
        if (isElementDisplay(frame)) {
Driver.getDriver().switchTo().frame(frame);}
    }

    public void refreshPage() {
        LOG.info("Refreshing page...");
        boolean isCurrentFrameFspApp =

waiters.waitForPresenceOfElement(By.tagName(BODY.getValue())).getAttribute(CL
ASS.getAttributeValue())
                        .contains(FSP_APP);
        Driver.getDriver().navigate().refresh();
        if (isCurrentFrameFspApp) {
            switchToFrame();
        }
    }

    //
////////////////////////////////////////////////////////////////////////////
////////////////////
    // Selenium Asserts functionality
    //
////////////////////////////////////////////////////////////////////////////
////////////////////

    /**
     * Transforms List of WebElements to List of Strings
     *
     * @param elements list of WebElements
     * @return transformed List of Strings
     */
    public List<String> transformElementToText(List<WebElement> elements) {
        return
elements.stream().map(WebElement::getText).collect(Collectors.toList());
    }

    /**
     * Verify equality of transformed WebElement List with List of Strings
     *
     * @param stringList:     list of Strings
     * @param webElementList: list of WebElements
     * @return true if lists content equals
```

```
     */
    public boolean isWebElementListEqualsToStringList(List<String>
stringList, List<WebElement> webElementList) {
        return
stringList.equals(transformElementToText(waiters.waitForAllElementsToBeDispla
y(webElementList)));
    }

    /**
     * @param webElementList: list of WebElements
     * @param elements:       list of Strings
     * @return true if WebElements list doesn't contain elements from list of
Strings
     */
    public boolean isWebElementListNotContainsElements(List<WebElement>
webElementList, List<String> elements) {
        return webElementList.stream()
                .noneMatch(item -> elements.contains(item.getText()));
    }

    /**
     * Asserts order of list of WebElements
     *
     * @param expected list
     * @param elements list of WebElements
     */
    public void assertListOrder(List<String> expected, List<WebElement>
elements) {
        List<String> elementsAsText = transformElementToText(elements);
        assertThat(elementsAsText, contains(expected.toArray()));
    }

    /**
     * Asserts list of WebElements in any order
     *
     * @param expected list
     * @param elements list of WebElements
     */
    public void assertListAnyOrder(List<String> expected, List<WebElement>
elements) {
        List<String> elementsAsText = transformElementToText(elements);
        assertThat(elementsAsText, containsInAnyOrder(expected.toArray()));
    }

    /**
     * Asserts expected and actual text in web page
     *
     * @param element      String element
     * @param expectedText String text
     */
    public void assertTextOnWebElement(WebElement element, String
expectedText) {
        String actualText =
waiters.waitForElementToBeDisplay(element).getText();
        assertThat(actualText, containsString(expectedText));
    }

    /**
     * Asserts WebElement is not displayed
     *
     * @param cssSelector String used inside css selector
     */
    public void assertWebElementNotPresent(String cssSelector) {
        assertThat("Error: Element is present by cssSelector " + cssSelector,
```

```
                isElementPresent(By.cssSelector(cssSelector)), is(false));
    }

    //
////////////////////////////////////////////////////////////////////////////////
//////////////////
    // Extra base selenium functionality
    //
////////////////////////////////////////////////////////////////////////////////
//////////////////
    public List<String> getListedMenuItems() {
        switchToDefaultContent();
        List<WebElement> elements =
waiters.waitForAllElementsToBeDisplay(MAIN_MENU_ITEMS);
        return elements.stream()
                .map(WebElement::getText)
                .collect(Collectors.toList());
    }

    public boolean isStatusMessageContains(String expectedMessage) {
        String statusMessage = getValidationMessage().toLowerCase();
        waitForInvisibilityOfValidationMessage();
        if (!StringUtils.containsIgnoreCase(statusMessage, expectedMessage))
{
            LOG.error(String.format("Actual: '%s'. Expected: '%s'.",
statusMessage, expectedMessage));
            return false;
        }
        return true;
    }

    public void assertTextOnNavActivePage(String text) {
        switchToDefaultContent();
        assertTextOnWebElement(activePageText, text);
        switchToFrame();
    }

    public String getValidationMessage() {
        try {
            return
waiters.waitForElementToBeDisplay(statusMessage).getText();
        } catch (TimeoutException e) {
            LOG.error("Notification Message for Action was not displayed.
Getting text from hidden element.", e);
            try {
                return getTextByJavascript(statusMessage);
            } catch (NoSuchElementException e2) {
                throw new RuntimeException("Notification Status Message is
not present", e2);
            }
        }
    }

    // Due to slowness of application sometimes script do not catch
notification message appeared.
    protected String getTextByJavascript(final WebElement element) {
        String script = "var element = arguments[0];" + "return
element.textContent;";
        return (String) ((JavascriptExecutor)
Driver.getDriver()).executeScript(script, element);
    }

    public String getValidationMessageColor() {
        try {
```

```
waiters.getNewWait(5).until(ExpectedConditions.visibilityOfElementLocated(STA
TUS_MESSAGE_CSS));
        } catch (TimeoutException e) {
            return "no banner";
        }
        return
findElement(By.cssSelector(VALIDATION_MESSAGE_ERROR)).getCssValue(BACKGROUND_
COLOR.getValue());
    }

    public void waitForValidationMessageDisappear() {

waiters.waitForElementToDisappear(By.cssSelector(VALIDATION_MESSAGE));
    }

    public void waitForInvisibilityOfValidationMessage() {
        waiters.waitForElementToDisappear(STATUS_MESSAGE_LOCATOR);
    }

    public void waitForVisibilityOfValidationMessage() {
        waiters.waitForElementToBeDisplay(STATUS_MESSAGE_LOCATOR);
    }

    public String getAlertDialogMessageBelow() {
        return
waiters.waitForElementToBeDisplay(By.cssSelector(DIALOG_MESSAGE_BELOW)).getTe
xt();
    }

    public String getAlertDialogMessageRight() {
        return
waiters.waitForNonBlankElementText(By.cssSelector(DIALOG_MESSAGE_RIGHT));
    }
       }
```

Насліюючись від цього класу ми можемо створювати наші класи, які є уособленням сторінок нашого проекту. Приклад такого класу приведено нижче

```
       public class Login  extends BasePage {

    private static final Logger LOG = LoggerFactory.getLogger(Login.class);

    private static final SubscriptionPage subscriptionPage =
PageFactory.initElements(Driver.getDriver(), SubscriptionPage.class);

    public static TestContext testContext = TestContext.getInstance();

    public static String user = testContext.getUser();

    public static String password = testContext.getPassword();

    private String SIGN_OUT_LOCATOR = "//*[contains(text(), '%s')]";

    @FindBy(css = "input[type='text']")
    private WebElement userNameSelector;

    @FindBy(css = "input[type='password']")
```

```
    private WebElement passwordSelector;

    @FindBy(css = ".button_img")
    private WebElement signInButton;

    @FindBy(css = ".button_75")
    private WebElement continueSignInCss;

    @FindBy(css = ".body_message a")
    private WebElement signBackInLink;

    @FindBy(xpath ="//*[contains(@data-bind, 'SkipMfaRegistration')]")
    private WebElement skipRegistration;

    @FindBy(css = "[type='Submit']")
    private WebElement nextButton;

    @FindBy(xpath ="//input[@name='IDToken2' and @type='password']")
    private WebElement choosePassword;

    @FindBy(xpath ="//input[@name='IDToken3' and @type='password']")
    private WebElement confirmPassword;

    @FindBy(xpath ="//div[text()='Next']")
    private WebElement setPasswordNextButton;

    public void setPassword(String password) {
        enterText(choosePassword, password);
        enterText(confirmPassword, password);
        clickOnElement(setPasswordNextButton);
    }

    public void login() {
        login(user, password);
    }

    public void login(String user, String password) {
            LOG.info("Navigate to: " + Url.url());
            Driver.getDriver().navigate().to(Url.url());
            LOG.info("Logging as   user...");
            enterText(userNameSelector, user);
            enterText(passwordSelector, password);
            signInButton.submit();
            continueSignInIfLoggedInElsewhere();
    }

    public void eikonLogin() {
        By usernameInput = By.cssSelector("input[type='text']");
        By passwordInput = By.cssSelector("input[type='password']");
        By signInButton = By.xpath("//div[text()='Sign In']");
        enterText(usernameInput, user);
        enterText(passwordInput, password);
        clickOnElement(signInButton);

        if (Driver.getDriver().getTitle().equalsIgnoreCase("You are signed in
to another device")) {
            clickOnElement(signInButton);
        }
    }

    public void loginSpecifyingRegKey(String user, String password, String
userRegKeyLabel) {
        try {
            login(user, password);
```

```
        } catch (TimeoutException e) {
            LOG.info("No login form, as we are logged in before.");
        }
        subscriptionPage.selectUserRegKeyIfRequired(userRegKeyLabel);
    }

    public void loginBookmarkSpecifyingRegKeyLabel(String user, String
password, String userRegKeyLabel) {
        LOG.info("Navigate to: " +
TestContext.getInstance().getProperty("BOOKMARK_LOGIN_URL"));

Driver.getDriver().navigate().to(TestContext.getInstance().getProperty("BOOKM
ARK_LOGIN_URL"));
        LOG.info("Bookmarked Logging as   user...");

        enterText(userNameSelector, user);
        enterText(passwordSelector, password);
        signInButton.submit();
        if (isElementDisplay(signInWhenUserAlreadySignedIn)) {
            LOG.info("SignIn click.");
            clickOnElement(signInWhenUserAlreadySignedIn);
        }
        subscriptionPage.selectUserRegKeyIfRequired(userRegKeyLabel);
    }

    public void loginSsoSpecifyingRegKeyLabel(String user, String password,
String userRegKeyLabel) {
        LOG.info("Navigate to: " +
TestContext.getInstance().getProperty("SSO_LOGIN_URL"));

Driver.getDriver().navigate().to(TestContext.getInstance().getProperty("SSO_L
OGIN_URL"));
        LOG.info("Logging via SSO...");

        enterText(ssoLogin, user);
        clickOnElement(nextButton);

        enterText(ssoPassword, password);
        clickOnElement(nextButton);

        if (isElementDisplay(skipRegistration)) {
            LOG.info("Skip registration click.");
            clickOnElement(skipRegistration);
        }
        clickOnElement(nextButton);
        subscriptionPage.selectUserRegKeyIfRequired(userRegKeyLabel);
    }

    public boolean is LoginPageDisplayed() {
        try {
            waiters.waitForElementToBeDisplay(userNameSelector);
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    public boolean verifySuccessfulSignOutMessage() {

waiters.waitForElementToBeDisplay(By.xpath(String.format(SIGN_OUT_LOCATOR,
"You have")));
        if
(waiters.isElementDisplayed(By.xpath(String.format(SIGN_OUT_LOCATOR,
                "You have successfully signed off your single sign-on
```

```
session.")))  ||

waiters.isElementDisplayed(By.xpath(String.format(SIGN_OUT_LOCATOR,
                        "You have been successfully signed out."))))
        {
            LOG.info("Logout message is expected. Url: " +
Driver.getDriver().getCurrentUrl());
            return true;
        }
        else {
           return false;
        }
    }

    public String getLogoutUrl() {
        return Driver.getDriver().getCurrentUrl();
    }

    public void clickSignBackInLink() {
        clickOnElement(signBackInLink);
    }
```

Після цього такі сторінки ми зможемо використовувати у проксі-класах визначення кроків, які потім підв’яжемо до юзер сценаріїв мовою Gherkin. Приклад такого класу

```
       public class  UsersOperations_sd {

    static final Logger LOG = LoggerFactory.getLogger(
UsersOperations_sd.class);

    private  RestClient  RestClient = new  RestClient();

    private AdminClientPage adminClientPage =
PageFactory.initElements(Driver.getDriver(), AdminClientPage.class);

    private AdminPage adminPage =
PageFactory.initElements(Driver.getDriver(), AdminPage.class);

    private AdminTRPage adminTRPage =
PageFactory.initElements(Driver.getDriver(), AdminTRPage.class);

    private AdminUserPage adminUserPage =
PageFactory.initElements(Driver.getDriver(), AdminUserPage.class);

    private Login  login Page = PageFactory.initElements(Driver.getDriver(),
Login .class);

    private BasePage basePage = new BasePage();

    Scenario scenario;

    public final static String  _CLIENT_DETAILS = " _CLIENT_DETAILS";

    private final static String  _USER_ID = " _USER_ID";

    private String getShortRandomValue() {
        return UUID.randomUUID().toString().replaceAll("-", "").substring(0,
8);
```

```
    }

    @Before
    public void beforeScenario(Scenario scenario) {
        this.scenario = scenario;
    }

    @When("I open {string} client")
    public void iOpenClientByName(String clientName) {
        ClientDto clientDto = getClientDto(clientName);
        if (clientDto == null) {
            throw new RuntimeException(String.format("Unable to open Client
'%s'", clientName));
        }
        adminTRPage.openClientViaDirectUrl(clientDto.getClientName());

adminPage.getAdminSideBarItemAfterWaitForIsDisplayed(clientDto.getClientName(
));
        scenario.write("Client Name:" + clientDto.getClientName());
    }

    @Given("I logged in to WCO as static   user")
    public void iEnsureIHaveLoggedInAs StaticUser() {
        login
Page.loginSpecifyingRegKeyLabel(TestContext.getInstance().getUser(),

TestContext.getInstance().getPassword(),

TestContext.getInstance().getUserRegKeyLabel());
        login Page.switchToFrame();
    }

    @Given("I ensure that client \"([^\"]*)\" is generated and stored")
    public void iEnsureClientIsGeneratedAndStored(String clientName,
Map<String, String> specificDetails) {
        ClientDto clientDto = getClientDto(clientName);
        if (clientDto == null) {
            String trustId = specificDetails.get("Trust ID");
            adminClientPage.clickCreateClient();

iFillInClientSubscriptionInformationWithSpecificDetailsAndStoreIt(clientName,
trustId, specificDetails);
            adminClientPage.clickOnAdminSettingsButton("Next >>");
            adminClientPage.fillInObligotaryClientSubscriptionDetails();
            adminClientPage.clickOnAdminSettingsButton("Next >>");
            adminClientPage.fillInObligotaryResolutionSettings();
            adminClientPage.clickOnAdminSettingsButton("Next >>");
            adminClientPage.clickOnAdminSettingsButton("Create client >>");
            if (!adminClientPage.isStatusMessageContains("Client created
successfully")) {
                storeClientDto(clientName, null);
                throw new RuntimeException("Unable to create Client via UI");
            }

TestContext.getInstance().addClientToClientListForFutureDeletionInAfterHook(g
etClientDto(clientName));
            LOG.info("Client is created: " +
getClientDto(clientName).getClientName());
        }
        else {
            LOG.info("Client is already created: " +
getClientDto(clientName).getClientName());
        }
    }
```

```
    @Then("I fill in client Details information with Trust ID \"([^\"]*)\"
and specific details")
    public void
iFillInClientSubscriptionInformationWithSpecificDetails(String trustId,

Map<String, String> specificDetails) {
        iFillInClientSubscriptionInformationWithSpecificDetailsAndStoreIt(
_CLIENT_DETAILS, trustId, specificDetails);
    }

    private void
iFillInClientSubscriptionInformationWithSpecificDetailsAndStoreIt(String
clientName, String trustId,

Map<String, String> specificDetails) {
        String defaultValue = "999999999";
        String shortRandomValue = getShortRandomValue();
        String emailAddress = specificDetails
            .getOrDefault("Email", String.format("bddtestuser_%s@tr.com",
shortRandomValue))
            .replace("replaceRandomNumber", shortRandomValue)
            .replace("mockUserEmail",
Optional.ofNullable(getStoredEmail("mockUserEmail")).orElse(""));

        ClientDto clientDto =
            ClientDto.NewClientBuilder()
                    .trustId(trustId)
                    .clientName(specificDetails.getOrDefault("Client name",

"zFrontEndClient-replaceRandomNumber")

.replace("replaceRandomNumber", shortRandomValue))
                    .salesforceId(specificDetails.getOrDefault("Salesforce
ID", defaultValue))
                    .numberOfUsers(specificDetails.getOrDefault("Number of
users", defaultValue))
                    .numberOfSearches(
                        specificDetails.getOrDefault("Number of searches",
defaultValue))
                    .numberOfWcOgsSearches(specificDetails.getOrDefault(
                        "Number of World Check and Watchlist OGS Searches",
defaultValue))
                    .numberOfMcOgsSearches(specificDetails
                                                .getOrDefault("Number of
Media Check OGS searches",
                                                        defaultValue))
                    .numberOfVcOgsSearches(specificDetails
                                                .getOrDefault("Number of
Vessel Check searches",
                                                        defaultValue))
                   .numberOfTotalCases(specificDetails.getOrDefault("Number
of Total Cases", defaultValue))
                    .complianceLeaderName(
                        specificDetails.getOrDefault("Compliance leader
name", "Nucky"))
                    .complianceLeaderEmail(
                        specificDetails.getOrDefault("Compliance leader
email", emailAddress))
                    .clientAdminFirstName(
                        specificDetails.getOrDefault("Client admin first
name", defaultValue))
                    .clientAdminLastName(
                        specificDetails.getOrDefault("Client admin last
```

```
name", defaultValue))
                    .clientAdminEmail(
                        specificDetails.getOrDefault("Client admin email",
emailAddress))
                    .clientAdminUserName(
                        specificDetails.getOrDefault("Client admin user
name", emailAddress))
                    .build();
        adminClientPage.fillInClientDetailsViaDto(clientDto);
        storeClientDto(clientName, clientDto);
    }

    @When("I validate that User Status for \"([^\"]*)\" email is
\"([^\"]*)\"")
    public void iValidateUserStatus(String storedEmailKey, String userStatus)
{
        String email = getStoredEmail(storedEmailKey);
        String  RequestId = AccelusDatabaseHelper.getInstance().get
RequestId(email);
        assertThat(" _REQUEST_ID should be not null.",  RequestId,
notNullValue());
        Awaitility.await().with()
                .pollInterval(Duration.ONE_SECOND)
                .atMost(new Duration(

Integer.parseInt(TestContext.getInstance().getProperty("PENDING_USER_TIMEOUT"
)),
                        TimeUnit.MINUTES)
                )
                .conditionEvaluationListener(condition ->
                        LOG.info("Wait User Status in DB. Expected: " +
userStatus))
                .until(() ->
AccelusDatabaseHelper.getInstance().getUserStatus( RequestId),
equalTo(userStatus));
    }

    @When("I clear User Data for \"([^\"]*)\" email")
    public void iClearUserData(String storedEmailKey) {
        String email = getStoredEmail(storedEmailKey);
        AccelusDatabaseHelper.getInstance().clearUserData(email);
    }

    @Then("I verify that User Name field is \"([^\"]*)\"$")
    public void iVerifyUserNameField(String userName) {
        assertThat("UserName is wrong.", adminUserPage.getUsername(),
is(userName));
    }

    @Then("I validate that   user role is \"([^\"]*)\" as \"([^\"]*)\"$")
    public void iValidateThat UserRoleIs(String expected UserRole, String
additionalExplanation) {
        String  UserId = get UserId();
         RestClient.validateUserRole( UserId, expected UserRole,
additionalExplanation);
    }

    @And("I generate test \"([^\"]*)\" email as \"([^\"]*)\"$")
    public void iGenerateTestEmail(String storedEmailKey, String
emailTemplate) {
        String emailAddress = emailTemplate.replace("replaceRandomNumber",
getShortRandomValue());
        storeEmail(storedEmailKey, emailAddress);
    }
```

```
    @And("I successfully created a user with firstname \"([^\"]*)\" lastname
\"([^\"]*)\" and \"([^\"]*)\" email and roles$")
    public void iCreatedAUserWithFirstnameLastnameAndEmailAndRoles(String
firstName, String lastName,
                                                                   String
emailKey, List<String> roles) {
        String userName = getStoredEmail(emailKey);
        stabilizedTrAdminOperationsPage
            .createUser(firstName, lastName, getStoredEmail(emailKey), roles,
true);
        //store reg key
        storeEmail(firstName + " " + lastName,
FspDB.getRegKeyForEmailAndLastName(userName, lastName));
    }

    @And("I successfully created a user with firstname \"([^\"]*)\" lastname
\"([^\"]*)\" and \"([^\"]*)\" not stored email and roles$")
    public void
iCreatedAUserWithFirstnameLastnameAndNotStoredEmailAndRoles(String firstName,
String lastName,

String email, List<String> roles) {
        stabilizedTrAdminOperationsPage
                .createUser(firstName, lastName, email, roles, true);
    }

    @And("I successfully created a user with firstname \"([^\"]*)\" lastname
\"([^\"]*)\" and \"([^\"]*)\" email and without roles$")
    public void iCreatedAUserWithFirstnameLastnameAndEmailWithoutRoles(String
firstName, String lastName,
                                                                   String
emailKey) {
        String userName = getStoredEmail(emailKey);
        stabilizedTrAdminOperationsPage
            .createUser(firstName, lastName, userName,
Collections.emptyList(), true);
        //store reg key
        storeEmail(firstName + " " + lastName,
FspDB.getRegKeyForEmailAndLastName(userName, lastName));
        scenario.write("User name:" + userName);
    }

    @And("I create a user with firstname \"([^\"]*)\" lastname \"([^\"]*)\"
and static user email and without roles$")
    public void iCreateUserWithFirstnameLastnameAndEmailWithoutRoles(String
firstName, String lastName) {
        String emailKey =
TestContext.getInstance().getProperty("STATIC_USERNAME");
        stabilizedTrAdminOperationsPage
                .fillNewUserFields(firstName, lastName, emailKey,
Collections.emptyList(), true);
    }

    @When("I create a user with firstname \"([^\"]*)\" lastname \"([^\"]*)\"
email \"([^\"]*)\" and without roles")
    public void
i_create_a_user_with_firstname_lastname_email_and_without_roles(String
firstName, String lastName, String email) {
        stabilizedTrAdminOperationsPage
                .fillNewUserFields(firstName, lastName, email,
Collections.emptyList(), true);
    }
```

```
    @When("I create a user with firstname \"([^\"]*)\" lastname \"([^\"]*)\"
email \"([^\"]*)\" and roles")
    public void
i_create_a_user_with_firstname_lastname_email_and_roles(String firstName,
String lastName, String email, List<String> roles) {
        stabilizedTrAdminOperationsPage
                .fillNewUserFields(firstName, lastName, email, roles, true);
    }

    @Then("I validate that a user is not created with firstname \"([^\"]*)\"
lastname \"([^\"]*)\" and \"([^\"]*)\" email and roles$")
    public void
iDidntCreatedAUserWithFirstnameLastnameAndEmailAndRoles(String firstName,
String lastName,

String emailKey, List<String> roles) {
        stabilizedTrAdminOperationsPage
                .createUserFailed(firstName, lastName,
getStoredEmail(emailKey), roles, true);
    }

    @And("I successfully added roles for user \"([^\"]*)\"$")
    public void iAddedRolesForUser(String userName, List<String> roles) {

stabilizedTrAdminOperationsPage.addRolesForUserAndBackToUsers(userName,
roles);
    }

    @And("I successfully added role for user \"([^\"]*)\"$")
    public void iAddedRoleForUser(String userName, List<String> roles) {
        stabilizedTrAdminOperationsPage.addRolesForUser(userName, roles);
    }

    @And("I validate that roles are not added for user \"([^\"]*)\"$")
    public void rolesForUserAreNotAdded(String userName, List<String> roles)
{
        stabilizedTrAdminOperationsPage.addRolesForUserFailed(userName,
roles);
    }

    @And("I successfully unassigned roles for user \"([^\"]*)\"$")
    public void iRemovedRolesForUser(String userName, List<String> roles) {

stabilizedTrAdminOperationsPage.removeRolesForUserAndBackToUsers(userName,
roles);
    }

    @And("I successfully unassigned role for user \"([^\"]*)\"$")
    public void iRemovedRoleFromUser(String userName, List<String> roles) {
        stabilizedTrAdminOperationsPage.removeRolesForUser(userName, roles);
    }

    public ClientDto getClientDto(String clientName) {
        return basePage.getObjectFromSession(clientName, ClientDto.class);
    }

    public void storeClientDto(String clientName, ClientDto clientDto) {
        basePage.storeObjectToSession(clientName, clientDto);
    }

    public String getStoredEmail(String storedEmailKey) {
        return basePage.getObjectFromSession(storedEmailKey, String.class);
    }
```

```
    public void storeEmail(String emailKey, String emailAddress) {
        basePage.storeObjectToSession(emailKey, emailAddress);
    }

    @Then("^I successfully delete a user \"([^\"]*)\"$")
    public void iSuccessfullyDeleteAUser(String userName) {
        stabilizedTrAdminOperationsPage.deleteUser(userName);
    }
```

Медоти цього класу, які прив'язані до певних певних кодових фраз(степів або кроків) ми можемо використовувати у юзер сценаріях. Приклад такого сценарію

```
       @logout_url_change
Feature: Logout URL change [WC1-28057]
    As a WC1 User
    I want to be able to log out from WC1
    So that I do not stay permanently logged in

    @ _extra_tracked_by_petros @logout_url_change1
    Scenario: logout as   TR Admin
        Given I logout from app
        And I logged in to WCO as static user
        When I refresh the page
        When I am logging out
        Then I see success sign out message
        And I verify logout page URL
        #When I click on Sign back in link
        #Then I see login page

    @ _extra_tracked_by_petros @logout_url_change2
    Scenario: logout as OnePass user
        Given I have logged in to WCO as "ALL_ROLE_USER"
        When I am logging out
        Then I see success sign out message
        And I verify logout page URL
        #When I click on Sign back in link
        #Then I see login page
```

Тепер коли UI частина у нашому фреймворку реалізована, ми можемо перейти до створення частини, яка буде відповідати за API тестування.

3.1.2. API чатисна

Перш за все, нам потрібно зібрати всі наші ендпїнти в одному місці для зручності модифікуання та додавння нових. Приклад такого місця показано нижче

```
        enum AccelusEndpoints {

    GET_USER("/users/user/%s"),
    GET_USER_SUBSCRIPTION("/user/subscription"),
    GET_ACCELUS_USER_DETAILS("/user-details/%s"),
    GET_CLIENTS("/client"),
    GET_CLIENT("/client/%s"),
    GET_FT_BY_PRODUCT_CODE("/toggleablefeature/product/%s"),

GET_FT_STATE_BY_PRODUCT_CODE("/toggleablefeature/product/%s/state"),
    GET_FT_BY_ID_BY_REGION("/toggleablefeature/%s/region/%s"),

GET_FT_BY_PRODUCT_CODE_AND_REGION("/toggleablefeature/product/%s/
region/%s/enabled"),

GET_FT_BY_USER_REG_KEY("/toggleablefeature/persona/%s/product/%s/regio
n/%s"),
    GET_VALIDATE_AAA_IDENTIFIER("/users/validate/%s"),
    GET_LOGIN_TOKEN("users/loginToken/%s"),
    GET_CLIENT_SUBSCRIPTION("/client-subscription/product/%s/region/%s"),

GET_SUBSCRIPTION("/subscription/product/%s/region/%s?search=%s&page=1
&size=1"),
    GET_USERS("/user"),
    GET_USER_SUBSCRIPTION_REG_KEY_LOOKUP("/user-subscription"),
    GET_TOKEN("/token/%s"),
    GET_LICENCE_USER("/licence/user/%s"),
    GET_LICENCE_CLIENT("/licence/client/%s/pools"),
    GET_USER_CURRENT_PREF("/user/current/preference"),
    GET_PRODUCTS("/product"),
```

```
    GET_PRODUCT("/product/%s"),
    GET_REGIONS("/region"),
    GET_REGION("/region/%s"),
    GET_PRODUCT_REGIONS("/region/product/%s"),
    GET_PRODUCTS_CONFIGURATIONS("/product-configuration"),

GET_PRODUCTS_CONFIGURATION_BY_REGION_AND_PRODUCT_CODE("
/product-configuration/product/%s/region/%s"),
    GET_LICENSES_BY_REGKEY("licence/user/%s"),
    GET_BASE_API_URL_BY_REG_KEY("/product-configuration/%s"),
    GET_LOCATION_BY_CLIENTCODE("client/%s/locations"),
    GET_CLIENT_ROLE("client/%s/role/%s"),
    POST_AUTHENTICATE("/authenticate"),
    POST_AUTHENTICATE_TRANSFER_TOKEN("/authenticate/transfer-token"),
    POST_CREATE_TRANSFER_TOKEN("/transfer-token"),
    POST_USER_REGISTRATION("/user-registration"),
    POST_BULK_USER_REGISTRATION("/user-registration/bulk"),
    POST_SUBSCRIPTION_STATUS("/subscription/user-status"),
    POST_SUBSCRIPTION_UPDATE("/subscription/update"),
    PUT_SUBSCRIPTION_USER_DETAILS("/subscription/userdetails"),
    POST_CREATE_CLIENT("/client"),
    POST_REGISTER_CLIENT("/client/%s/register"),
    POST_UPDATE_CLIENT("/client/update"),
    POST_AAA_SUMMARY_CLIENTS("/aaa/summary/clients"),
    POST_RETRY_USER_CREATION("/users/user/%s/sync-aaa"),
    POST_CREATE_FT("/toggleablefeature"),
    POST_CREATE_ACCELUS_ADMIN_SUBSCRIPTION("/accelus-
admin/subscription/add"),
    PUT_UPDATE_AAA_IDENTIFIER("/users/user/%s"),
    POST_FT_BY_ID_BY_REGION("/toggleablefeature/%s/region/%s"),
```

```
    POST_LICENCE_USER("/licence/user"),
    POST_USER_LOOKUP("/users/user/lookup"),
    POST_SEARCH_LOCATION_BY_TERM("/location/search"),
    DELETE_FT("/toggleablefeature/%s"),
    DELETE_TOKEN("/seamless/%s"),
    DELETE_SUBSCRIPTION("/subscription/delete"),
    DELETE_CLIENT("/client/%s"),
    DELETE_ACCELUS_ADMIN_SUBSCRIPTION("/accelus-
admin/subscription/delete"),
    GET_INFOBAR_USER_OFFERS("/infobar/offers/%s"),
    PUT_INFOBAR_ACCEPT_USER_OFFER("/infobar/offer/accept"),
    PUT_INFOBAR_REJECT_USER_OFFER("/infobar/offer/reject"),
    V1("/v1"),
    V2("/v2"),
    V3("/v3"),
    V4("/v4"),
    V5("/v5");

    private final String endpoint;

    AccelusEndpoints(String endpoint) {
        this.endpoint = endpoint;
    }

    public String getEndpoint() {
        return endpoint;
    }
}
```

Тепер перейдемо до класу де у нас будуть викликаися ці сцеенарії, а потім і до класу де буде сам API тест

```
       public class RestClient {

    private static final Header AUTORIZATION_HEADER_SYSTEM = new
Header("Authorization", "SYSTEM");

    private  Header userNameHeader;
    private  Header authorizationHeader;

    public AccelusRestClient(String userRegKey) {
        this.userNameHeader = new Header(USER_NAME_HEADER_KEY, userRegKey);
        this.authorizationHeader = new Header("Authorization", userRegKey);
        RestAssured.baseURI =
TEST_CONTEXT.getProperty("ACCELUS_SECURITY_REST_BASE_URI");
    }

    public void setHeaders(String headersRegKey){
        this.userNameHeader = new Header(USER_NAME_HEADER_KEY,
headersRegKey);
        this.authorizationHeader = new Header("Authorization",
headersRegKey);
    }

    public Response getUserDetails(String regKey) {
        RestAssured.basePath = V4.getEndpoint();
        String url = String.format(GET_USER.getEndpoint(), regKey);
        Headers headers = new Headers(AUTORIZATION_HEADER_SYSTEM,
JSON_HEADER);

        return doGetRestCall(url, headers);
    }

    public Response getAccelusUserDetails(String regKey) {
        RestAssured.basePath = V2.getEndpoint();
        String url = String.format(GET_ACCELUS_USER_DETAILS.getEndpoint(),
regKey);
        Headers headers = new Headers(AUTORIZATION_HEADER_SYSTEM,
JSON_HEADER);

        return doGetRestCall(url, headers);
    }

    public Response getAllUsersWithPagination(Long page, Long pageSize) {
        String accelusSecurityRestHost =
TEST_CONTEXT.getProperty("ACCELUS_SECURITY_REST_BASE_URI");
        String template = "%s/v2/user?page=%d&size=%d";
        String url = String.format(template, accelusSecurityRestHost, page,
pageSize);
        Headers headers = new Headers(AUTORIZATION_HEADER_SYSTEM,
JSON_HEADER);

        return doGetRestCall(url, headers);
    }

    public Response retrieveUserSubscriptionWithPagination(Long page, Long
pageSize) {
        String accelusSecurityRestHost =
TEST_CONTEXT.getProperty("ACCELUS_SECURITY_REST_BASE_URI");
        String template = "%s/v2/user-subscription";
        String url = String.format(template, accelusSecurityRestHost, page,
pageSize);
        Headers headers = new Headers(AUTORIZATION_HEADER_SYSTEM,
JSON_HEADER);
        String body = String.format("{\"paginationDetails\": {"
                                    + "\"currentPage\": %d,"
```

```
                                    + "\"itemsPerPage\": %d"
                                    + "}"
                                    + "}", page, pageSize);
        return doPostRestCall(body, url, headers);
    }
```

І сам клас з тестом

```
    public class LocationSearchAT extends BaseAT {

    HttpStatus successStatusCode = OK;
    private String rdpUsername = TestContext.getInstance().getUser();
    private String rdpPassword = TestContext.getInstance().getPassword();
    private String rdpRegKey = TEST_CONTEXT.getProperty("RDP_REGKEY");
    private String searchBy;
    private String filter;
    private List<String> relatedLocations;

    @After
    public void tearDown() {
        accelusRestClient.setHeaders(getTRAdminUser());
    }

    @Before
    public void prepare() throws EncoderException {
        LOG.info("############### Login to wco.");
        AccelusUtils.accelusPingLogin(rdpUsername, rdpPassword, rdpRegKey);
        accelusRestClient.setHeaders(rdpRegKey);
    }


    @Test
    public void getLocationBySearchTerm() {
        searchBy = "location_id";
        filter = "A-88888888";
        relatedLocations = Collections.emptyList();
        String searchName = "Internal TR Fixed-Interest Trading Application";

        GetLocationsByTerm getLocationBody = new GetLocationsByTerm(searchBy,
filter, relatedLocations);
        searchBy = getLocationBody.getSearchBy();
        LOG.info("############### Search location by  " + searchBy);
        Response locationsByLocationId =
accelusRestClient.getLocationsByTerm(getLocationBody);
        accelusRestClient.validateResponseLocations(locationsByLocationId,
filter, searchBy, successStatusCode.value());

        getLocationBody.setSearchBy("location_name");
        getLocationBody.setFilter(searchName);

        searchBy = getLocationBody.getSearchBy();
        filter = getLocationBody.getFilter();
        LOG.info("############### Search location by " + searchBy);
        Response locationsByName =
accelusRestClient.getLocationsByTerm(getLocationBody);
        accelusRestClient.validateResponseLocations(locationsByName, filter,
searchBy, successStatusCode.value());

        getLocationBody.setSearchBy("all_fields");
        searchBy = getLocationBody.getSearchBy();
        LOG.info("############### Search location by " +
getLocationBody.getSearchBy());
        Response locationsByAllFields =
accelusRestClient.getLocationsByTerm(getLocationBody);
```

```
        assertThat(locationsByAllFields.getStatusCode(),
is(INTERNAL_SERVER_ERROR.value()));
    }

    @Test
    public void getLocationBySearchTermWithRelatedLocation() {
        searchBy = "location_id";
        filter = "A-00166143";
        String searchName = "Reliance Industries";
        relatedLocations = Arrays.asList("A-00105574", "A-00571044");

        GetLocationsByTerm getLocationBody = new GetLocationsByTerm(searchBy,
filter, relatedLocations);
        LOG.info("############### Search location by  " + searchBy);
        Response locationsByLocationId =
accelusRestClient.getLocationsByTerm(getLocationBody);

accelusRestClient.validateResponseLocationWithRelatedLocation(locationsByLoca
tionId, filter, searchBy,
                relatedLocations, successStatusCode.value());

        getLocationBody.setSearchBy("location_name");
        getLocationBody.setFilter(searchName);

        searchBy = getLocationBody.getSearchBy();
        filter = getLocationBody.getFilter();
        LOG.info("############### Search location by " +
getLocationBody.getSearchBy());
        Response locationsByName =
accelusRestClient.getLocationsByTerm(getLocationBody);

accelusRestClient.validateResponseLocationWithRelatedLocation(locationsByName
, filter, searchBy,
                relatedLocations, successStatusCode.value());

        getLocationBody.setSearchBy("all_fields");
        searchBy = getLocationBody.getSearchBy();
        LOG.info("############### Search location by " +
getLocationBody.getSearchBy());
        Response locationsByAllFields =
accelusRestClient.getLocationsByTerm(getLocationBody);

accelusRestClient.validateResponseLocationWithRelatedLocation(locationsByAllF
ields, filter, searchBy,
                relatedLocations, successStatusCode.value());
    }
}
```

Хочу відмітити, що ми створили аналогічний Base клас як і для UI частини, однак вони між схожі, тож його лістинг було опущено.

Не потрібно забувати, що у нас тіла реквесту і респонзу є досить специфічними і для кращого опрацювання їх варто було б створити класи що відповідають цим тілам. У фреймворку нині є два види реалізації даної властивості

Через анотації loombok

```
        @Getter
@AllArgsConstructor
@NoArgsConstructor
@Setter
@Builder
@ToString
@EqualsAndHashCode

public class RdpLicense {
    private String productId;
    private String productName;
}
```

Та стандартний метод реалізації JAVA класу

```
        @JsonIgnoreProperties(
         ignoreUnknown = true
)
public class UserRegistrationResponse {
    private final Status status;
    private final FirstTimeLoginToken firstTimeLoginToken;
    private final String registrationKey;
    private final boolean success;
    private final String failureReason;
    private final String failureData;

    private UserRegistrationResponse() {
        this.status = null;
        this.firstTimeLoginToken = null;
        this.registrationKey = null;
        this.success = false;
        this.failureReason = null;
        this.failureData = null;
    }

    public UserRegistrationResponse(Status status, String registrationKey,
FirstTimeLoginToken firstTimeLoginToken, boolean success, String
failureReason, String failureData) {
        this.status = status;
        this.firstTimeLoginToken = firstTimeLoginToken;
        this.registrationKey = registrationKey;
        this.success = success;
        this.failureReason = failureReason;
        this.failureData = failureData;
    }

    public UserRegistrationResponse(UserRegistrationResponse.Builder builder)
{
        this.status = builder.getStatus();
        this.firstTimeLoginToken = builder.getFirstTimeLoginToken();
        this.registrationKey = builder.getRegistrationKey();
        this.success = builder.isSuccess();
        this.failureReason = builder.getFailureReason();
        this.failureData = builder.getFailureData();
    }

    public String getRegistrationKey() {
        return this.registrationKey;
    }

    public Status getStatus() {
```

```
        return this.status;
    }

    public FirstTimeLoginToken getFirstTimeLoginToken() {
        return this.firstTimeLoginToken;
    }

    public boolean isSuccess() {
        return this.success;
    }

    public String getFailureReason() {
        return this.failureReason;
    }

    public String getFailureData() {
        return this.failureData;
    }

    public boolean equals(Object o) {
        if (this == o) {
            return true;
        } else if (o != null && this.getClass() == o.getClass()) {
            UserRegistrationResponse response = (UserRegistrationResponse)o;
            return this.success == response.isSuccess() && this.status ==
response.getStatus() && Objects.equals(this.firstTimeLoginToken,
response.getFirstTimeLoginToken()) && Objects.equals(this.registrationKey,
response.getRegistrationKey()) && Objects.equals(this.failureData,
response.getFailureData()) && Objects.equals(this.failureReason,
response.getFailureReason());
        } else {
            return false;
        }
    }

    public int hashCode() {
        return Objects.hash(new Object[]{this.status,
this.firstTimeLoginToken, this.registrationKey, this.success,
this.failureReason, this.failureData});
    }

    public String toString() {
        return ToStringBuilder.reflectionToString(this,
ToStringStyle.SHORT_PREFIX_STYLE);
    }

    public static final class Builder {
        private Status status;
        private FirstTimeLoginToken firstTimeLoginToken;
        private String registrationKey;
        private boolean success;
        private String failureReason;
        private String failureData;

        public Builder() {
        }

        public Builder(UserRegistrationResponse userRegistrationResponse) {
            this.status = userRegistrationResponse.getStatus();
            this.firstTimeLoginToken =
userRegistrationResponse.getFirstTimeLoginToken();
            this.registrationKey =
userRegistrationResponse.getRegistrationKey();
            this.success = userRegistrationResponse.isSuccess();
```

```
            this.failureReason = userRegistrationResponse.getFailureReason();
            this.failureData = userRegistrationResponse.getFailureData();
        }

        public UserRegistrationResponse build() {
            return new UserRegistrationResponse(this);
        }

        public String getRegistrationKey() {
            return this.registrationKey;
        }

        public Status getStatus() {
            return this.status;
        }

        public FirstTimeLoginToken getFirstTimeLoginToken() {
            return this.firstTimeLoginToken;
        }

        public boolean isSuccess() {
            return this.success;
        }

        public String getFailureReason() {
            return this.failureReason;
        }

        public String getFailureData() {
            return this.failureData;
        }

        public UserRegistrationResponse.Builder withStatus(Status status) {
            this.status = status;
            return this;
        }

        public UserRegistrationResponse.Builder
withFirstTimeLoginToken(FirstTimeLoginToken firstTimeLoginToken) {
            this.firstTimeLoginToken = firstTimeLoginToken;
            return this;
        }

        public UserRegistrationResponse.Builder withRegistrationKey(String
registrationKey) {
            this.registrationKey = registrationKey;
            return this;
        }

        public UserRegistrationResponse.Builder withSuccess(boolean success)
{
            this.success = success;
            return this;
        }

        public UserRegistrationResponse.Builder withFailureReason(String
failureReason) {
            this.failureReason = failureReason;
            return this;
        }

        public UserRegistrationResponse.Builder withFailureData(String
failureData) {
            this.failureData = failureData;
```

```
                return this;
            }
        }
}
```

Отже наші тести були успішно імплементовано і ми можемо їх спробувати запустити локально для перевірки їх працездатності. Ось як це виглядає в разі успішного виконання без помилок

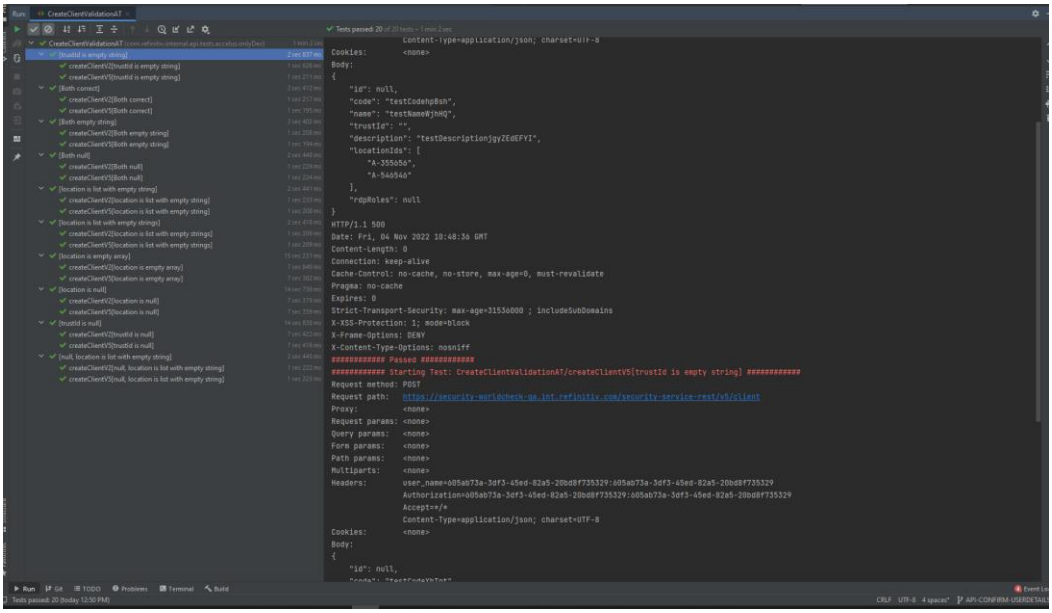

Рис.3.5. Успішне виконання

І в разі невдачі

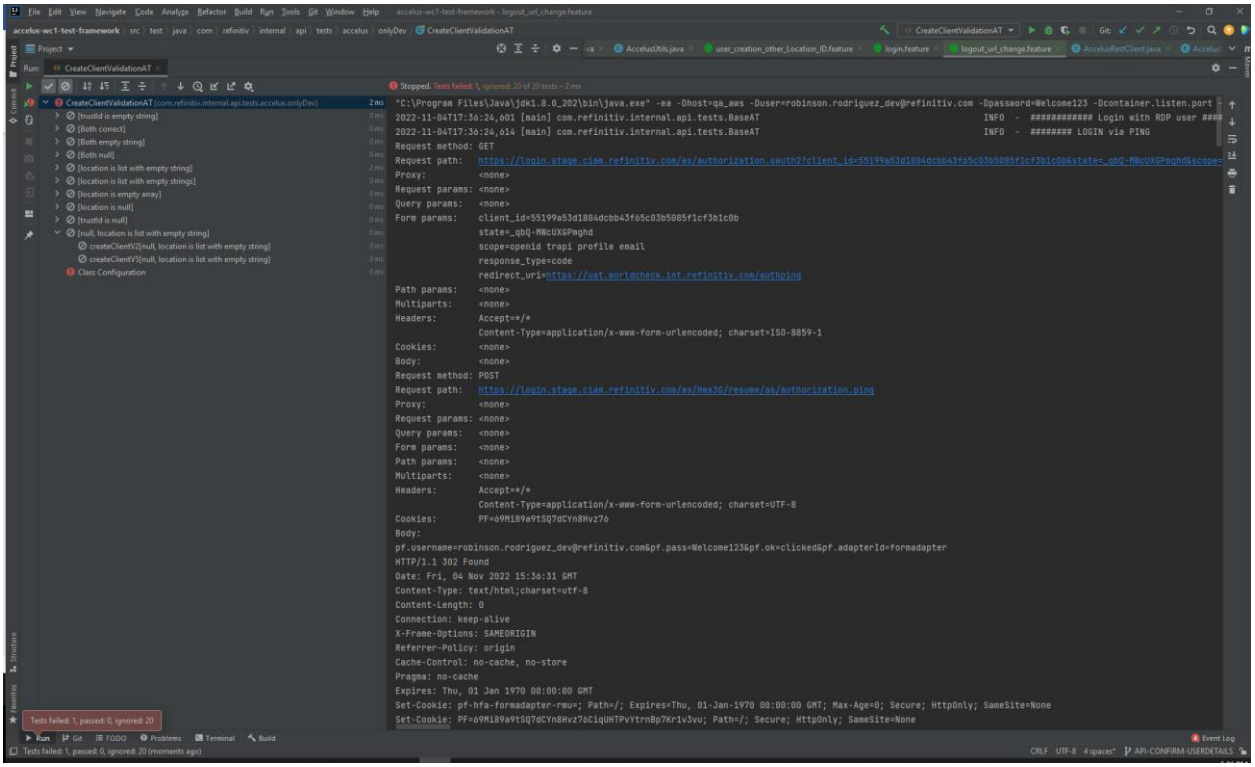

Рис.3.6. Невдале виконання

Однак ця картина не є досить привабливою, і кожен раз виконувати тести в ручну не є досить легко. Саме тому ми переходимо до фінальної частини нашиго проекту налаштування Jenkins job.

Сам дженкінс налаштувати не важко і є досить багато гайдів, як це зробити. Вкажемо тільки основні кроки.

Встановити Дженкінс на локальній машині та перейти на порт 8080(за замовчуванням) за допомогою команд та браузера

```
sudo wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | apt-ke
add -
sudo echo deb http://pkg.jenkins-ci.org/debian binary/ >
/etc/apt/sources.list.d/jenkins.list
```

Рис.3.7. Команди для інсталювання Jenkins

В результаті має вийти таке

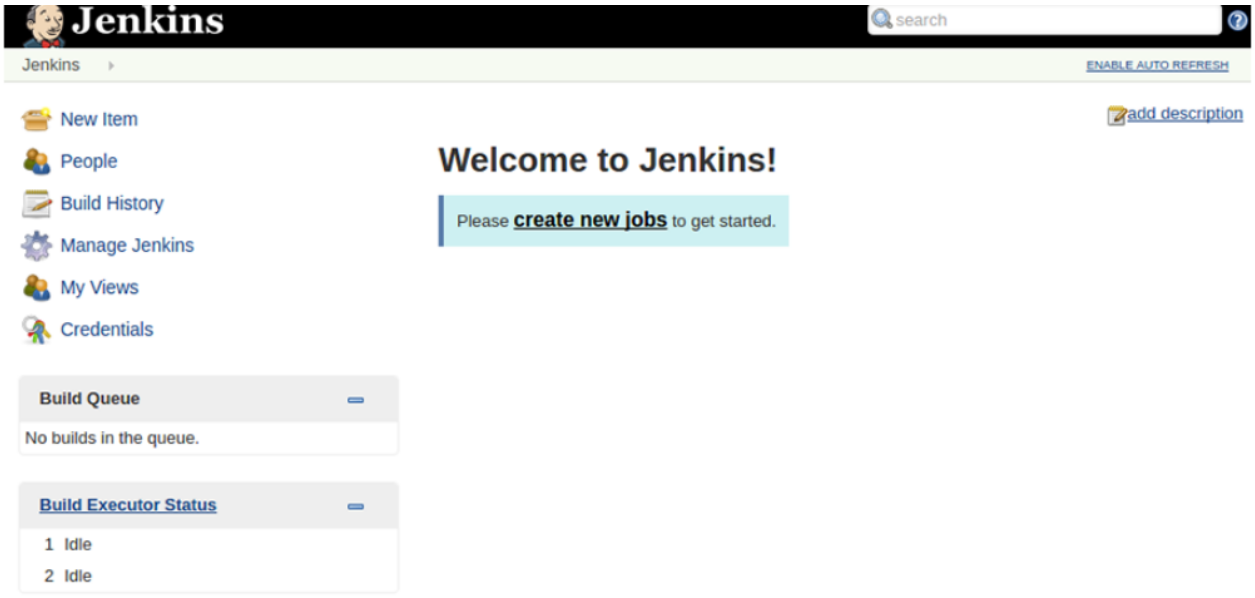

Рис.3.8. Головне вікно додатку

Після цього потрібно додати свій проект на сервіс GitHub та дати Jenkins до нього доступ через SSH ключ.

Потім налаштувати джобу та насолоджуватися автоматизацією. Ось приклади роботи Jenkins job та графічне зображення рельтатів виконання тестів.

Висновки до розділу 3

Створений нами автоматизований тестовий фреймворк призначений для виконання настпуних завдань:

- Додавання нових класів сутностей та ендпоїнтів при розширенні функціоналу продукту;
- Автоматизування виконання тестів з формування чибельної для людини звітності;
- Інформування про результати виконання тестів;
- Зберезення історії виконання тестів, їх успіхи та місця конкретних невдач, а також історію змін конфігаріції автоматизації;
- Відстеження працездатності конкретних модулів додатку;
- Забезпечення доступу до тестових сценаріїв та їх редагування чи оптимізації для обмеженого кола осіб;
- Забезпечення збереження історії змін самих тестів та будь-якої частини тестового автоматизованого фреймворку, включаючи інформацію хто вносив зміни і коли.

Власники проекту що будуть використовувати даний продукт для покращення кінцевої якості проекту значно зекономлять свій час та кошти і максимально швидко будуть отримувати рівень здоров’я проекту після внесення змін.Всі тестові сценераї можна виконувати без прив’язки до локації чи платформи, що значно спрощує життя тестувальникам. Продукт містить тестові сценарії по всіх важливих функціях проекту що полегшує оцінку загальнго рівня оцінки здоров’я проекту то знаходження конкретного місця відмови у разі несправності.Це забезпечує зручність, оскільки при автоматизованному тестувані видно усі передумови та кроки, які пройшли перед помилкою.

Створена система працює без відмов та має досить гарні результати робити.

Висновки

Для упровадження на будь-якому проекті системи тестування потрібно зробити базову модель автоматизованого тестового фреймворку з відкритим для вдосконалень кодом.

Для втілення в життя цієї ідеї покращення правильним буде вибір керуватися даними правилами:

- Пріоритизувати потрібно покращення кінцевої якості продукту;
- В основі має лежати ступінь покриття тестами частин продукту;
- Автоматизований тестовий фреймворк має містити універсальну основу та легко модернізуватися;
- Повинні бути як і сценарії автоматизованого тестування так і ручного.

Серед найважливіших питань з підходу до побудови тестового фреймворку і тестування проекту в цілому можна виділити наступні:

- Розгляд вимог до продукту та створення загальної стратегії тестування ще на початкових етапах SDLC;
- Співпраця всієї команди розробки та замовника з тестувальниками;
- Розгляд шляхів оптимізації та покращення створених тест плану, стратегії, та сценаріїв.

Саме це допоможе покращити кінцеву якість продукту ще на початкових етапах його розробки та зменшити витрати на реалізацію та вірно обрати шлях яким рухатися з подальшими покращеннями для тестування та розробки автоматизованого тестового фреймворку.

У наш час кожен продукт та замовник прагне скоротити рівень витрат до мінімуму, а рівень якості кінцевого продукту підняти до максимуму. Однак не всі знають як це зробити за допомогою введення правильних підходів та методів тестування і автоматизації в частковому випадку. Тож, ідеальним

варіантом в цьому випадку буде створити правильну стратегію тестування та використати правильні застосунки та спеціалістів.

Тестовий автоматизований фреймворк — це набір інструкцій або правил, які використовуються для створення та розробки тестових випадків. Фреймворк складається з комбінації практик та інструментів, розроблених, щоб допомогти фахівцям із забезпечення якості тестувати ефективніше.

Ці вказівки можуть включати стандарти кодування, методи обробки тестових даних, сховища об'єктів, процеси для зберігання результатів тестів або інформацію про те, як отримати доступ до зовнішніх ресурсів.

Хоча це не є обов'язковими правилами, і тестувальники все одно можуть писати сценарії або записувати тести, не дотримуючись їх, використання організованої структури зазвичай надає додаткові переваги. Тестовий автоматизований фреймворк є зручним застосунком для полегшення рутинної роботи тестувальників. Важливим є саме правильне створення його структури та написання потрібних тест сценаріїв. Якщо все правильно зробити, то результати тестів та рівень «здоров'я» продукту зможе перевірити будь-яка особа, яка розуміє специфіку проекту завдяки зрозумілій, системі звітності.

Це буде корисним не тільки для тестувальників чи керівництва, а і для девелоперів, які зможуть перевіряти якість своїх змін завдяки швидкому прогону юніт-тестів після кожного оновлення коду проекту. Розглянуті до цього атоматизовані фреймворки мають ряд певних недолік покращення яких буде розглянуто в наступному, 3 розділі дипломної роботи:

- Висока ціна побудови тестового фреймворку;
- Важкість підтримки без затрат часу кваліфікованими кадрами;
- Складність у освоєнні;
- Чистота та зрозумілість фреймворку;
- Незрозумілість найменувань;
- Складність вибору потрібних іструментів;

- Важкість актуалізації тестових сценаріїв.

Створений нами автоматизований тестовий фреймворк призначений для виконання наступних завдань:

- Додавання нових класів сутностей та ендпоїнтів при розширенні функціоналу продукту;
- Автоматизування виконання тестів з формування чибельної для людини звітності;
- Інформування про результати виконання тестів;
- Зберезення історії виконання тестів, їх успіхи та місця конкретних невдач, а також історію змін конфігаріції автоматизації;
- Відстеження працездатності конкретних модулів додатку;
- Забезпечення доступу до тестових сценаріїв та їх редагування чи оптимізації для обмеженого кола осіб;
- Забезпечення збереження історії змін самих тестів та будь-якої частини тестового автоматизованого фреймворку, включаючи інформацію хто вносив зміни і коли.

Власники проекту що будуть використовувати даний продукт для покращення кінцевої якості проекту значно зекономлять свій час та кошти і максимально швидко будуть отримувати рівень здоров’я проекту після внесення змін.Всі тестові сценераї можна виконувати без прив’язки до локації чи платформи, що значно спрощує життя тестувальникам. Продукт містить тестові сценарії по всіх важливих функціях проекту що полегшує оцінку загальнго рівня оцінки здоров’я проекту то знаходження конкретного місця відмови у разі несправності.Це забезпечує зручність, оскільки при автоматизованному тестувані видно усі передумови та кроки, які пройшли перед помилкою.

Створена система працює без відмов та має досить гарні результати роботи.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Бібліотека MSDN. Джерело інформації для розробників, які використовують засоби, продукти, технології та служби корпорації Майкрософт. [Електронний ресурс]. – Режим доступу: http://msdn.microsoft.com
2. Вікіпедія. Вільна енциклопедія. [Електронний ресурс]. –Режим доступу:http://ru.wikipedia.org/wiki
3. SQA Days. [Електронний ресурс]. – Режим доступу: http://sqadays.com/
4. Software testing training and software testing services. [Електронний ресурс]. –Режим доступу: http://www.rbcs-us.com/
5. Code Project. Товариство розробки програмного забезпечення. [Електронний ресурс]. – Режим доступу: http://codeproject.com
6. SeleniumHQ Browser Automation. [Електронний ресурс]. – Режим доступу:http://docs.seleniumhq.org/.
7. Святослав Кулик. Тестування програмного забезпечення. Базовий курс -- 2 видання, Мінськ , 2015 – 296 с.
8. Farrell, C. Harrison N. Under the Hood of .NET Memory Management. Simple Talk Publishing. 2011. 213 с. ISBN 978-1-906434-74-8
9. The Art of Software Testing / Glenford J. Myers, Revised and Updated by Tom Badgett, Todd M.Thomas, Corey Sandler. - 2nd ed. - Hoboken, New Jersey.: John Wiley & Sons, Inc., 2004 - 234 p
10. Блек Р. Ключові процеси тестування / Р. Блек., Лорі, 2006. 544 с
11. Osherove Roy. The Art of Unit Testing: With Examples in .Net. 1st edition. Greenwich, CT, USA : Manning Publications Co., 2009. ISBN: 1933988274, 9781933988276.