МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Інститут прикладного системного аналізу
(назва факультету, інституту)

Кафедра системного проектування
(назва кафедри)

До захисту допущено

Завідувач кафедри

_____ А.І. Петренко
(підпис) (ініціали, прізвище)

“___”_____2017 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) освітньо-кваліфікаційного рівня “ **спеціаліст** ”
(назва ОКР)
з напряму підготовки (спеціальності) 7.05010103, «Системне проектування».
на тему: Створення фреймворку для функціонального тестування економічних застосувань

Студент групи ДА-52с Бабенко Володимир _____
(шифр групи) (прізвище, ім’я, по батькові) (підпис)

Керівник проекту к.т.н., доц. Кисельов Г.Д. _____
(вчені ступінь та звання, прізвище, ініціали) (підпис)

Консультанти:

з технічних вимог — к.т.н., доц. Харченко К.В. _____
(назва розділу ДП (ДР)) (вчені ступінь та звання, прізвище, ініціали) (підпис)

нормоконтроль — к.т.н., доц. Стіканов В.Ю. _____
(назва розділу ДП (ДР)) (вчені ступінь та звання, прізвище, ініціали) (підпис)

рецензент — д.т.н., проф. кафдери Обчислювальної Техніки НТУУ КПІ ім. І.Сікорського Бузовський О.В. _____
(назва розділу ДП (ДР)) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Київ – 2017

Національний технічний університет України
“Київський політехнічний інститут”

Факультет (інститут) **Інститут прикладного системного аналізу**.
(повна назва)

Кафедра **Системного проектування**.
(повна назва)

Напрям підготовки **050101, Комп’ютерні науки**.
(код, назва)

Спеціальність **7.05010103, «Системне проектування»..**
(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри
_____ А.І. Петренко .
(підпис) (ініціали, прізвище)

“___”_____ 2017 р.

ЗАВДАННЯ
на дипломний проект (роботу) освітньо-кваліфікаційного рівня
“спеціаліст” .
(назва рівня)

студенту Бабенку Володимиру Володимировичу
(прізвище, ім’я, по батькові)

1. **Тема проекту** (роботи) Створення фреймворку для функціонального тестування економічних застосувань
затверджена наказом по університету від «20» 09 2016р. № 89/1-СТ

2. **Термін здачі** студентом закінченого проекту (роботи) “15” 01 2017р.

3. **Вихідні дані до проекту** (роботи)
Нефункціональні вимоги: мова програмування C#, спеціалізовані бібліотеки та засоби автоматизації тестування, Page Object Modeling pattern.
Функціональні вимоги: програма взаємодії з інтерфейсом програмного продукту, програма виконання тестового сценарію, візуалізація виконання тестування, підготовка звітності по виконанню тестування.

4. **Перелік питань, які мають бути розроблені**
- Проаналізувати предметну область тестування ПЗ

- Проаналізувати сучасні методи та засоби автоматизованого тестування ПЗ
- Описати алгоритм розробки та структуру системи автоматизованого тестування
- Розробити систему автоматизації функціонального тестування продукту
- Визначити основні ризики у проекті та розрахувати їх вплив на проект

5. **Перелік графічного (ілюстративного) матеріалу**
   Структура прототипу системи автоматизації тестування (креслення)
   Структура модифікованої системи автоматизації тестування (креслення)
   Структура проекту та спеціалізованох бібліотеки Selenium (1-й плакат)
   Загальний алгоритм роботи при створенні системи автоматизованого тестування програмного продукту (2-й плакат)
   Схема розробки системи автоматизації тестування (3-й плакат)
   POM патерн. Діаграма класів (4-й плакат)

   .....

6. **Консультанти**
   з технічних вимог: доц.к.т.н . Харченко К.В

7. **Дата видачі завдання** " 10 " 09 2016 р.

Керівник дипломного проекту (роботи) _____ __Кисельов Г.Д.__
(підпис) (ініціали, прізвище)

Завдання прийняв до виконання _____ __Бабенко В.В.__
(підпис) (ініціали, прізвище)

ЗАТВЕРДЖУЮ
Керівник дипломного
проекту (роботи)

_____ __Кисельов Г.Д._
(підпис) (ініціали, прізвище)

“___”_____2016 р.

КАЛЕНДАРНИЙ ПЛАН-ГРАФІК
виконання дипломного проекту (роботи)
студентом Бабенком В.В.
(прізвище, ініціали)

№ з/п	Назва етапів роботи та питань, які повинні бути розроблені відповідно до завдання	Термін виконання	Позначки керівника про виконання завдань
1	Ознайомлення з технічною літературою і підготовка теоретичної частини роботи	10.09.2016–30.09.2016	
2	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	15.10.2016	
3	Проектування модулів пристрою, розробка моделей.	15.11.2016	
4	Тестування розроблених моделей модулів. Перевірка відповідності завданню.	30.12.2016	
5	Підготовка графічного матеріалу, оформлення пояснювальної записки, підготовка до захисту	05.01.2017	
6	Проходження нормоконтролю, отримання відгуку, рецензії, передача роботи в ДЕК	10.01.2017	
7	Захист дипломної роботи	15.01.2017	

Студент _____
(підпис)

АНОТАЦІЯ

до дипломного проекту (роботи) освітньо-кваліфікаційного рівня “спеціаліст” Бабенка Володимира Володимировича

на тему: «Створення фреймворку для функціонального тестування економічних застосувань»

Дипломна робота присвячена дослідженню та аналізу ручного та автоматизованого тестування програмного забезпечення, розгляду додаткових бібліотек автоматизації для веб-додатків, їх основних методів, характеристик та застосувань, основних переваг та недоліків. Було розроблено систему автоматизованого функціонального тестування веб-додатку з використанням насутпних середовищ, інструментів та засобів: Microsoft Visual Studio, C#, Selenium WebDriver, NUnit, NLog, Page Opject Modeling pattern.

Загальний об’єм роботи 139 сторінок, 17 рисунків, 3 таблиці, 15 посилань та 2 додатки

Ключові слова: автоматизоване тестування, веб-додаток, тестові сценарії, Selenium, Selenium WebDriver, Page Object Modeling.

ABSTRACT

of a specialist degree work, witch made by Babenko Volodymyr Volodymyrovych

on theme: “Creating a framework of functional testing for economic applications”

This work is devoted to research and analyse of manual and automated software testing, reviewing additional libraries of automation testing for web applications, their main methods, characteristics and practices, basic advantages and disadvantages.The system of automated functional testing of web applications was created using next enviroments, methods and tools: Microsoft Visual Studio, C #, Selenium WebDriver, NUnit, NLog, Page Opject Modeling pattern.

The total amount of work is 139 pages, 17 images, 3 tables, 15 references and 2 applications

Keywords: automated testing, web application, test scripts, Selenium, Selenium WebDriver, Page Object Modeling.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення

ТЗ – технічне завдання

ОС – операційна система

QA – quality assurance (забезпечення якості)

QC – quality control (контроль якості)

UI – user interface (інтерфейс користувача)

DDT – Data Driven Testing (Тестування з керуванням даними)

XPath – XML Path Language

CSS – Cascading Style Sheets (каскадні таблиці стилів)

ВСТУП

Тестування є одним з важливих етапів розробки ПЗ та загалом пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи етапом експлуатації. Роботи з тестування безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності ПЗ заявленим вимогам.

Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціального додаткового тестування. Саме тому виникає необхідність автоматизації даних процесів тестування. Адже завдяки автоматизації тестування відбувається економія та оптимізація ресурсів команди з розробки ПЗ.

Мета цієї роботи – дослідити та проаналізувати системні засоби, інструменти та методології розробки системи автоматизованого тестування ПЗ, розробити систему автоматизації функціонального тестування економічного веб-додатку.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- Дослідити та проаналізувати системні засоби, інструменти та методології розробки системи автоматизованого тестування ПЗ
- Розробити базовий фреймворк для автоматизованого тестування
- За допомогою створених інструментів фреймворку описати автоматизовані тести

1 ОСОБЛИВОСТІ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВИДИ, РІВНІ ТА МЕТОДИ ТЕСТУВАННЯ

1.1 Якість програмного забезпечення

Перш за все, якщо говорити про тестування та забезпечення якості продукту, потрібно дослідити питання та тему визначення якості продукту та її особливості.

Якість програмного забезпечення – набір властивостей продукту, що характеризують його здатність задовольнити встановлені або передбачувані потреби замовника.[1]

За основною структурою моделі якості програмного забезпечення можна виділити та визначити основні характеристики та атрибути:

1 **Функціональність (functionality).** Здатність програмного забезпечення за деяких умов вирішувати деякі задачі, що потрібні користувачам. Зокрема, виділяються наступні атрибути цієї характеристики:

1.1 **Функціональна придатність(suitability).** Здатність вирішувати потрібний набір задач.

1.2 **Точність (accuracy).** Здатність видавати потрібний результат.

1.3 **Здатність до взаємодії (interoperability).** Здатність взаємодіяти з іншими потрібними системами.

1.4 **Відповідність стандартам (compliance).** Відповідність стандартам, законам, актам тощо.

1.5 **Захищеність (security).** Здатність запобігати несанкціонованому доступу до даних і програм.

2 **Надійність (reliability).** Здатність програмного забезпечення витримувати визначену працездатність при деяких умовах. Зокрема, виділяються наступні атрибути цієї характеристики:

2.1 **Завершеність (maturity).** Величина, що є зворотною мірою до частоти відмови програмного забезпечення.

2.2 **Стійкість до відмов (fault tolerance).** Здатність підтримки працездатності ПЗ при порушеннях.

2.3 **Здатність до відновлення (recoverability).** Здатність відновлювати визначений рівень працездатності після відмови.

2.4 **Відповідність стандартам надійності (reliability compliance).**

3 **Зручність використання (usability).** Здатність ПЗ бути зручним та привабливим для користування. Зокрема, виділяються наступні атрибути цієї характеристики:

3.1 **Зрозумілість (understandability).** Показник зворотній до зусиль, зо затрачаються на сприйняття основ програмного забезпечення.

3.2 **Зручність освоєння (learnability).** Зворотній показник до зусиль, що потрібно витратити для навчання роботі із програмним забезпеченням.

3.3 **Зручність роботи (operability).** Зворотній показник до зусиль, що потрібно витратити для розв’язку задачі користувачеві на ПЗ.

3.4 **Привабливість (attractiveness).**

3.5 **Зручність використання (usability compliance).**

4 **Продуктивність (efficiency).** Відношення результатів роботи ПЗ до витрачених ресурсів усіх типів. Зокрема, виділяються наступні атрибути цієї характеристики:

4.1 **Часова ефективність (time behavior).** Забезпечення передачі необхідного об’єму даних за певний час.

4.2 **Ефективність використання ресурсів (resource utilization).** Здатність вирішувати потрібні задачі з використанням визначених об’ємів ресурсів визначених видів.

4.3 **Відповідність стандартам продуктивності (efficiency compliance).**

5 **Зручність супроводу (maintainability)** Зокрема, виділяються наступні атрибути цієї характеристики:

5.1 **Можливість аналізу (analyzability).** Зручність проведення аналізу помилок, дефектів та змін для них.

5.2 **Зручність внесення змін (changeability).** Зворотній показник до трудових затрат на виконання змін у ПЗ

5.3 **Стабільність (stability).** Зворотній показник до ризику виникнення дефектів при внесенні змін до ПЗ.

5.4 **Зручність перевірки (testability).** Зокрема, виділяються наступні атрибути цієї характеристики:

6 **Портативність (portability).**

6.1 **Адаптованість (adaptability).**

6.2 **Зручність встановлення (installability).**

6.3 **Здатність до співіснування (coexistence).** З іншими ПЗ у загальному оточенні.

6.4 **Відповідність стандартам портативності (portability compliance).**

Окрім технічної точки зору на якість ПЗ є також оцінка якості зі сторони звичайного користувача. Для оцінку цього аспекту потрібно ставити питання до створеного ПЗ саме зі сторони користувача. Наприклад, «чи є UI зрозумілим?», «Наскільки легко виконувати легкі або складні операції?», «Чи зрозумілі повідомлення про помилку?» тощо.[2]

1.2 Історія розвитку тестування програмного забезпечення

Перші програмні системи розроблялися в рамках програм наукових досліджень або програм для потреб міністерств оборони. Тестування таких продуктів проводилося строго формалізовано із записом всіх тестових процедур, тестових даних, отриманих результатів. Тестування виділялося в окремий процес, який починався після завершення кодування, але при цьому, як правило, виконувалося тим же персоналом.

У 1960-х багато уваги приділялося «вичерпному» тестуванню, яке повинно проводитися з використанням усіх шляхів у коді або всіх можливих вхідних даних. Було відзначено, що в цих умовах повне тестування ПЗ неможливе, тому що, по-перше, кількість можливих вхідних даних дуже велика, по-друге, існує безліч шляхів, по-третє, складно знайти проблеми в архітектурі та специфікаціях. З цих причин «вичерпне» тестування було відхилено й визнано теоретично неможливим.

На початку 1970-х тестування ПЗ розглядалося як «процес, спрямований на демонстрацію коректності продукту» або як «діяльність з підтвердження правильності роботи ПЗ». У програмній інженерії, яка в той час зароджувалася, верифікація ПЗ визначалася як «доказ правильності». Хоча концепція була теоретично перспективною, на практиці вона вимагала багато часу й не охоплювала всі аспекти тестування. Було вирішено, що доказ правильності — неефективний метод тестування ПЗ. Однак, у деяких випадках демонстрація правильної роботи використовується і в наші дні, наприклад, приймально-здавальні випробування. У другій половині 1970-х тестування представлялося як виконання програми з наміром знайти помилки, а не довести, що вона працює. Успішний тест — це тест, який виявляє раніше невідомі проблеми. Даний підхід цілком протилежний попередньому.

Зазначені два визначення являють собою «парадокс тестування», в основі якого лежать два протилежних твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре, а з іншого — виявляє помилки у ПЗ, показуючи, що продукт не працює. Друга мета тестування є більш продуктивною з точки зору поліпшення якості, оскільки не дозволяє ігнорувати недоліки ПЗ.

У 1980-х тестування розширилося таким поняттям як запобіганням дефектам. Проектування тестів — найбільш ефективний з відомих методів запобігання помилок. В цей же час почали висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування повинно включати перевірки впродовж усього циклу розроблення, при цьому це має бути керований процес. В ході тестування треба перевірити не тільки зібрану програму, але й вимоги, код, архітектуру, самі тести. «Традиційне» тестування, яке існувало до початку 1980-х, відносилося тільки до скомпілювати, готової системі (зараз це зазвичай називається системне тестування), але надалі тестувальники стали залучатися в усі аспекти життєвого циклу розроблення. Це дозволяло раніше знаходити проблеми у вимогах та архітектурі й тим самим скорочувати терміни та бюджет розроблення. У середині 1980-х з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе виконати більше тестів, ніж людина, причому зробить це більш надійно. Спочатку ці інструменти були вкрай простими й не мали можливості написання сценаріїв на скриптових мовах.

На початку 1990-х у поняття «тестування» стали включати планування, проектування, створення, підтримку й виконання тестів та тестових оточень, а це означало перехід від тестування до забезпечення якості, що охоплює весь цикл розроблення ПЗ. У цей час починають з'являтися різні програмні інструменти для підтримки процесу тестування: більш просунуті середовища

для автоматизації з можливістю створення скриптів і генерації звітів, системи управління тестами, ПЗ для проведення навантажувального тестування. У середині 1990-х з розвитком Інтернету й розробленням великої кількості веб-застосунків особливої популярності стало набувати «гнучке тестування» (за аналогією з гнучкими методологіями програмування).

У 2000-х з'явилося ще більш широке визначення тестування, коли в нього було додано поняття «оптимізація бізнес-технологій». ВТО направляє розвиток інформаційних технологій згідно з цілями бізнесу. Основний підхід полягає в оцінці та максимізації значущості всіх етапів життєвого циклу розроблення ПЗ для досягнення необхідного рівня якості, продуктивності, доступності.[3]

1.3 Введення та основні визначення

Тестування – це процес керованого експериментування з продуктом за допомогою тестів з метою виявлення в ньому помилок, тобто виявлення неточностей допущених розробниками ПЗ.

Тест – контрольна задача для перевірки коректності функціонування системи та/або її ПЗ. Основна ідея тестування – запустити ПЗ і спостерігати за його роботою та її наслідками. Якщо збій в роботі ПЗ відбувся, тоді аналізується звіт з метою виявлення місцезнаходження помилки, яка його викликала.

Вдалим тестом є той, при якому виконання програми закінчилось з помилкою і навпаки. Тестування виконує дві основні задачі[4]:

1) демонстрація якості функціонування ПЗ;

2) знаходження і усунення помилок в ПЗ.

Головною метою тестування є збільшення ймовірності того, що ПЗ, яке тестується, буде відповідати своїм специфікаціям.

Тестування - процес ітераційний. Після виявлення і виправлення кожної помилки обов'язково слідує повторення тестів, що має на меті перевірку працездатності програми. Більш того, для ідентифікації причини виявленої проблеми може бути потрібно проведення спеціального додаткового тестування. При цьому завжди потрібно пам'ятати фундаментальний висновок, зроблений професором Едсжером Дейкстрой в 1972 г: "Тестування програм може служити доказом наявності помилок, але ніколи не доведе їх відсутність!" [5].

Тестування може виконуватися вручну або автоматично. Автоматичне тестування, якщо воно виконано коректно, забезпечує більш швидке, якісне і ефективне тестування, що приводить до зменшення кількості тестів, скороченню часу тестування, підвищення стійкості системи.

Тестування пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації, та безпосередньо пов'язане з управлінням вимогами і змінами, адже метою тестування якраз є можливість переконатися у відповідності програм заявленим вимогам [4].

Як було зазначено вище, тестування підтверджує, що ПЗ працює згідно специфікації. Вважають, що **програма працює коректно**, якщо вона задовольняє наступним критеріям:

1) отримавши коректні дані, програма надає правильний результат;

2) отримавши некоректні дані програма відхиляє їх;

3) програма не зависає і не вилітає, приймаючи як коректні, так і некоректні дані;

4) програма функціонує нормально стільки часу скільки потрібно;

5) програма працює без збоїв і виконує всі необхідні функції в повному обсязі.

Також у цьому розділі викладені основні визначення, відповідно до стандартів, які використовуються як в наукових, так і в прикладних сферах стосовно ПЗ та його якості [6].

Помилка (Error) – хибне значення величини на виході системи (або підсистеми), викликане несправностями або збоями, яке, в свою чергу, може викликати відмову.

З точки зору надійності ПЗ помилку можна розглядати, як упущення або неточність, допущені проектувальниками ПЗ, програмістами, аналітиками та тестерами. Наприклад, проектувальник може неправильно зрозуміти завдання, а програміст – неправильно описати змінну та інше

Несправність, Дефект (Fault) – визнана або передбачувана причина помилки; наслідок відмови деякої системи, що обслуговувала або обслуговує в даний момент часу розглянуту систему. Дефекти також часто називають ―багами. Цей термін використовують, якщо вплив дефекту на роботу програми невеликий. Якщо ж помилка пов’язана із специфікаціями або архітектурою програми, то використовують слово ―дефект.

Збій (Malfunction) – прояв несправності, зазвичай в роботі устаткування.

Відмова (Failure) - порушення нормального функціонування системи, повна або часткова втрата працездатності системи (або підсистеми).

Під час виконання програми або роботи всієї системи, тестер, розробник або користувач можуть не отримати очікуваних результатів. В деяких

випадках така поведінка – симптом помилки. Досвідчений розробник/тестер завжди зберігає базу даних помилок, з якими він стикався.

Некоректна поведінка може також означати неправильні значення вихідних, неправильний відгук пристрою або неправильне зображення на екрані. В процесі розробки відмови та баги зазвичай виявляються тестерами, а дефекти знаходяться і виправляються самими розробками. Якщо відмова виникла у користувача, звіт про помилку прямує розробнику з метою її усунення.

Несправність у коді коду не завжди веде до відмови. Насправді неправильна частина програми може функціонувати довгий час без прояву яких-небудь недоліків. Проте, за відповідних умов несправність викликає відмову. А саме:

1) вхідні дані програми повинні використовувати частину коду, яка містить несправність;

2) наслідком несправності має стати некоректний внутрішній стан програми;

3) некоректний внутрішній стан програми повинен впливати на вихідні дані так, щоб результат помилки можна було спостерігати.

Якщо при наявності несправності у коді ПЗ відповідає умовам зазначеним вище, то воно вважається **придатним для тестування (testable)**.

Тестування (**testing**) – це:

- Процес використання ПЗ при умові аналізу або запису отримуваних результатів з метою перевірки (оцінки) деяких властивостей тестованого об'єкту. (The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component).

- Процес аналізу ПЗ з метою виявлення відмінностей між створеним станом ПЗ та зазначеним у специфікації (що свідчить про прояв помилки) при експериментальній перевірці відповідного пункту вимог. (The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs))

- Контрольоване виконання програми на множині тестових даних і аналіз результатів цього виконання для пошуку помилок.

Тест (Test) – контрольна задача для перевірки коректності функціонування ПЗ. Групу взаємозв'язаних тестів називають **комплектом тестів** (**test suite**).

Контрольний приклад (test case). Контрольний приклад в сенсі тестування – записи, що відносяться до тесту, а саме, звіти, що містять:

1. Вхідні дані тесту – інформація, яку програма отримує із зовнішнього джерела, як то пристрій, інша програма або людина.

2. Умови виконання – вимоги для проведення тесту, наприклад, певний стан бази даних або конфігурація пристрою.

3. Очікувані вихідні дані – передбачуваний результат роботи коду.

Даний перелік визначає мінімальну необхідну інформацію для контрольного прикладу відповідно до стандартів. Компанія- розробник може визначити додаткову інформацію необхідною для внесення в звіт з метою її повторного використання в майбутньому або надання докладніших даних іншим тестерам і розробникам. Наприклад, мета тесту може бути включена в запис для більшої зрозумілості результатів тесту. До визначення точного формату звіту контрольного прикладу слід залучати розробників, тестерів та/або відділ забезпечення якості ПЗ.[7]

Якість (Quality) – ступінь відповідності системи, компоненту або процесу заданим вимогам, потребам або очікуванням користувача. З метою визначення добротності системи, компоненту або процесу використовують так звані атрибути якості – характеристики, що відображають дану властивість.

Метрика (Metrics) – кількісна міра ступеня наявності атрибута системи, компоненту або процесу.

Приведемо декілька прикладів атрибутів якості з коротким описом [15]:

- властивість, що відповідає за безперервність коректного протікання процесу, відноситься до **надійності (reliability)**;

- властивість, що відповідає за постійну готовність системи, відноситься до **готовності (availability)**;

- властивість, що відповідає за відсутність катастрофічних наслідків в системному середовищі, відноситься до **безпеки (safety)**;

- властивість, що відповідає за запобігання несанкціонованому доступу до інформації, відноситься до **конфіденційності (confidentiality)**;

- властивість, що відповідає за відсутність появи в системі невідповідних змін інформації, відноситься до **цілісності (integrity)**;

- властивість, що відповідає за здатність системи піддаватися ремонту і розвитку, відноситься до **ремонтопридатність (maintainability)**.

- властивість, що відповідає за здатність – рівень зусиль, необхідних для навчання, роботи, підготовки вхідних і обробки вихідних даних ПЗ, відноситься до **зручності роботи (usability)**.

Перевірка на валідність(Validation) - процес, що дозволяє визначити, наскільки точно з позицій потенційного користувача деяка модель представляє задану суть реального миру.

Верифікація (Verification) - процес, який дозволяє визначити, що розроблене програмне забезпечення точно реалізує концептуальний опис даної системи. Або, як ще кажуть, процес перевірки відповідності системи заданими стандартами. Верифікація успішна, якщо отримані дані збігаються з очікуваними, заздалегідь визначеними як правильні. Зазначимо, що вона може бути неформальною, тобто тестер визначає успішність на основі своїх знань.

Верифікація та перевірка на валідність часто використовуються разом, але мають різні визначення. Різниця в них важлива для тестування ПЗ. Верифікація – це підтвердження того, що продукт відповідає специфікаціям, а перевірка на валідність – вимогам користувача. Може здатися, що визначення вельми схожі, але на прикладі опису проблем орбітального телескопа Хаббла видно різницю.

У квітні 1990 року до космосу було запущено орбітальний телескоп Хаббла, що використовував велике дзеркало для збільшення об'єктів, на які був націлений. Його конструкція вимагала неймовірної точності, а тестування було майже неможливим, оскільки він призначався для космосу і не міг бути встановлений на Землі. У зв'язку з цим, можна було лише перевірити на відповідність специфікаціям. Після проходження таких тестів, телескоп було запущено.

На жаль, після введення його в експлуатацію виявилось, що зображення, які надсилались їм, є розфокусованими. Дослідження показало, що було виготовлено неправильне дзеркало. Не дивлячись на те, що, можливо, це було найбільш точно розраховане дзеркало з коли-небудь створених, а допуск складав не більше 1/20 довжини хвилі видимого світла, воно виявилося дуже

плоским по краях. Відхилення від потрібної форми поверхні склало лише 2 мікрометри, але результат виявився катастрофічним — сильна сферична аберація, оптичний дефект, при якому світло, відображене від країв дзеркала, фокусується в точці, відмінній від тієї, в якій фокусується світло, відображене від центру дзеркала. Тобто тестування показало, що дзеркало пройшло верифікацію, але не пройшло перевірку на валідність. У 1993 році експедиція на космічному кораблі відремонтувала телескоп, встановивши коректуючу лінзу для відновлення фокусу. Хоча цей приклад не відноситься до ПЗ, верифікація та перевірка на валідність також успішно застосовуються і до тестування програм. Отже, сама верифікація не є достатньою. Лише повна перевірка дозволить уникнути проблем, з якими зіткнулися учені у випадку з телескопом Хаббла.

Методи забезпечення якості є техніками, що гарантують досягнення певних показників якості при їх застосуванні.

Методи контролю якості дозволяють переконатися, що певні характеристики якості ПЗ досягнуті. Самі по собі вони не можуть допомогти їх досягненню, вони лише дають змогу визначити, чи вдалося отримати в результаті те, що хотілося, чи ні, а також знайти помилки, дефекти і відхилення від вимог.

Методи контролю якості ПЗ можна класифікувати таким чином:

- Методи та техніки, пов'язані з аналізом властивостей ПЗ під час його роботи. Це, перш за все, всі види тестування, а також вимірювання кількісних показників якості, які можна визначити за наслідками роботи ПЗ, — ефективність за часом й іншими ресурсами, надійність, доступність та ін.

- Методи та техніки визначення показників якості на основі симуляції роботи ПЗ за допомогою моделей різного роду. До цього вигляду

відносяться перевірка на моделях (model checking), а також прототипіювання (макетування), що використовується для оцінки якості прийнятих рішень.

- Методи та техніки, націлені на виявлення порушень формалізованих правил побудови початкового коду ПЗ, проектних моделей та документації. До методів такого роду відноситься інспекція коду, що полягає в цілеспрямованому пошуку певних дефектів і порушень вимог в коді на основі набору шаблонів; автоматизовані методи пошуку помилок в коді, не засновані на його виконанні; методи перевірки документації на узгодженість і відповідність стандартам.

- Методи та техніки звичайного або формалізованого аналізу проектної документації і початкового коду для виявлення їх властивостей. До цієї групи відносяться численні методи аналізу архітектури ПЗ, методи формального доказу властивостей ПЗ і формального аналізу ефективності вживаних алгоритмів.

1.4 Види та рівні тестування

У процесі розробки ПЗ тестування ПЗ зазвичай відбувається на декількох рівнях інтеграції: поблочне тестування, перевірка взаємодії (інтеграційне тестування) та системне тестування.

Для зручності розуміння поділу та градації видів та методів тестування усі підходи (окрім за ступенем доступу до програмного коду, що йде вище) можна розділити на два наступні типи:

1. Функціональне тестування. Тип black-box тестування, що базується на специфікації ПЗ, що буде тестуватись. Програма тестується

запровадженням вхідних даних і результат перевіряється на відповідність необхідної призначеної функціональності. Це тестування ведеться на повній, інтегрованій системі, щоб оцінити відповідність системи з її специфікацією та вимогам. Можна виділити 5 основних кроків, що задіяні в даному типі тестування:

- Визначення функціональних можливостей, що призначені до виконання у призначеній програмі.
- Створення випробувальних даних, що базуються на специфікації ПЗ.
- Вихідні дані, на основі випробувальних даних і специфікації ПЗ.
- Написання тестових сценаріїв та виконання тестів.
- Співставлення дійсних і очікуваних результатів.

Отже до цього глобального типу можемо віднести наступні типи тестування:

- Unit testing
- Integration testing
- System testing
- Regression testing
- Acceptance testing
- Alpha testing
- Bets testing

2. Нефункціональне тестування. Включає у собі тестування нефункціональних вимог, таких як продуктивність, безпека, інтерфейс тощо.

До цього виду відносяться наступні типи тестів:

- Performance testing
- Load testing
- Stress testing
- Usability testing
- Security testing
- Portability testing

Відповідно до етапів розробки ПЗ прийнято виділяти три фази тестування: модульне, інтеграційне і системне.

- **Модульне тестування, тестування модуля, або автономне тестування (module testing, unit testing)** — контроль окремого програмного модуля, зазвичай в ізольованому середовищі (тобто ізольовано від решти всіх модулів). Під модулем розуміється логічно замкнутий фрагмент програми, який може бути викликаний через його інтерфейс. Модуль перевіряється на відповідність своїм специфікаціям і внутрішню логіку.

- **Інтеграційне тестування або тестування взаємодій (integration testing**) — контроль взаємодії між частинами системи (модулями, компонентами, підсистемами).

- **Системне тестування або комплексне тестування (system testing)** — контроль та/або випробування всього програмного забезпечення, як повної системи, в цільовому середовищі, тобто підтвердження того, що доступ до всіх компонентів системи і взаємодія з ними несуперечливі і передбачені згідно специфікацій системи.

Вочевидь, що фази не є взаємозамінними і, наприклад, проведення модульного тестування не гарантує правильності інтеграційного тестування,

бо правильність функціонування окремих компонент не гарантує правильності їх взаємодії, як між собою, так і з системою в цілому.

Як правило, перед тим, як вважатися завершеним, програмне забезпечення проходить дві стадії тестування. Перша стадія називається альфа-тестуванням.

Альфа-тестування (alpha testing) — використання майже готової версії продукту (як правило, програмного або апаратного забезпечення) штатними програмістами (розробниками і тестерами) з метою виявлення помилок в його роботі для їх подальшого усунення перед бета-тестуванням.

По закінченню тестування альфи, розробка входить в стадію бети.

Бета-тестування (beta testing) — інтенсивне використання ПЗ з метою виявлення максимальної кількості помилок в його роботі для їх подальшого усунення перед остаточним виходом (випуском) продукту на ринок, до масового споживача.

На відміну від альфа-тестування, що проводиться силами штатних розробників або тестувальників, бета-тестування припускає залучення добровольців з числа звичайних майбутніх користувачів продукту, яким розсилається згадана попередня версія продукту (так звана бета-версія). Такими добровольцями (їх називають бета- тестерами) зазвичай керує цікавість до нового продукту — цікавість, задля задоволення якої вони цілком згодні миритися з можливістю випробувати наслідки ще незнайдених (а тому і невиправлених) помилок.

Крім того, бета-тестування може використовуватися як частина стратегії просування продукту на ринок (наприклад, безкоштовна роздача бета-версій дозволяє привернути широку увагу споживачів до остаточної дорогої версії

продукту), а також для отримання попередніх відгуків про нього від широкого круга майбутніх користувачів.

Тестування, що засноване на вимогах (Requirement based testing) – тестування кожного припущення з певного документу (документи технічної підтримки, посібники та інша документація користувача).

Регресійне тестування (regression testing, від лати. regressio — рух назад) — спільна назва для всіх видів тестування програмного забезпечення, метою яких є виявлення помилок у вже протестованих ділянках початкового коду. Такі помилки — коли після внесення змін до програми перестає працювати те, що повинне було продовжувати працювати — називають регресійними помилками (regression bugs).

Зазвичай використовувані методи регресійного тестування включають повторні прогони попередніх тестів, а також перевірки, чи не потрапили регресійні помилки в чергову версію в результаті злиття коду.

З досвіду розробки ПЗ відомо, що повторна поява одних і тих самих помилок — випадок доволі поширений. Іноді це відбувається із-за слабкої техніки управління версіями або унаслідок людської помилки при роботі з системою управління версіями. Але настільки ж часто вирішення проблеми буває таким, що «недовго живе»: після наступної зміни в програмі воно перестає працювати. І нарешті, при переписуванні якої-небудь частини коду, часто спливають ті ж помилки, що були в попередній реалізації.

Тому найкращім рішенням є створення тесту на помилку, при її виявленні, з метою його використання при подальших змінах програми. Хоча регресійне тестування може бути виконане і вручну, але найчастіше це робиться за допомогою спеціалізованих програм, що дозволяють виконувати всі регресійні тести автоматично. У деяких проектах навіть використовуються

інструменти для автоматичного прогону регресійних тестів через заданий інтервал часу. Зазвичай це виконується після кожної вдалої компіляції (у невеликих проектах) або щоночі, або кожного тижня.

У термінології тестування поняття «тестування білого ящика» і «тестування чорного ящика» відносяться до того, чи має розробник тестів доступ до початкового коду тестованого ПЗ, або ж тестування виконується через призначений для користувача інтерфейс або прикладний програмний інтерфейс, наданий тестованим модулем.

При **тестуванні чорного ящика (black-box testing)**, тестер має доступ до ПЗ тільки через ті самі інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування.

Можна виділити наступні види тестування чорного ящика:

Тестування за класами еквівалентності (Equivalence class testing). Клас еквівалентності - це множина значень змінної, що, за припущенням, є еквівалентними. Контрольні приклади є еквівалентними, якщо виконується наступна сукупність вимог:

1) вони всі перевіряють один об’єкт;

2) якщо один с них „спіймає‖ дефект, то інший також;

3) якщо один з них не виявляє дефект, то інший, скоріше за все, також цього не зробить.

Якщо такий клас еквівалентності виявлено, то треба використовувати для тестування лише один з його членів.

Способи вибору таких представників також визначають певний вид тестування:

1) **Граничне тестування (Boundary testing).** Граничні значення це найбільші та найменші значення класів еквівалентності. При тестуванні граничних значень тестуються також значення менше за мале, та більше за велике.

2) **Тестування кращих представників (Best representative testing).** Тестування значень, що найбільш вірогідно призведуть до виявлення дефекту. Значення завжди можна змінити різними шляхами. Треба покрити всі можливі варіанти.

При **тестуванні білого ящика (white-box testing, також говорять — прозорого ящика)**, розробник тесту має доступ до початкового коду і може писати код, який пов'язаний з бібліотеками тестованого ПЗ. Це типово для модульного тестування (англ. unit testing), при якому тестуються тільки окремі частини системи. Воно забезпечує те, що компоненти конструкції — працездатні і стійкі, до певного ступеня.

При тестуванні білого ящика розрізняють наступні види:

Тестування логіки (Logic testing). Багато програм мають логіку типу „якщо, то , тестування логіки тестує можливі сценарії та комбінації.

Тестування станів (State-bases testing). Робота кожної програми - це переходи з одного стану в інший. Тестування станів має на меті уважну перевірку коректності роботи програми у кожному її стані.

Тестування покриття операторів та гілок (Statement and branch coverage). Сто відсоткове покриття має місце коли покриті всі рядки та всі

твердження коду. В іншому випадку береться до уваги покриття у процентному відношенні.

Тестування шляхів (Path testing). Тестування набору шляхів (під-шляхів), що проходить програма.[2]

Після дослідження цього питання можна виділити основні переваги та недоліки двох основних підходів до тестування за доступом та знанням програмного коду ПЗ.

Black-box переваги:

- Добре та ефективно підходить для великих сегментів коду;
- Доступ до коду не потрібен;
- Через явно виражені ролі розділяє точку зору користувача від розробника;
- Велика кількість помірно кваліфікованих тестувальників може протестувати додаток без знання реалізації, мови програмування або ОС.

Black-box недоліки:

- Обмежене покриття, так як фактично здійснюється тільки деяка вибрана кількість тестових сценаріїв;
- У зв’язку з тим, що тестувальник має обмежені знання про програму є не достатньо ефективним;
- Сліпе покриття, так як тестувальник не може виділити конкретний сегмент коду або область помилок;
- Тестові приклади важко спроектувати.

White-box переваги:

- Так як тестувальник знає програмний код, може досить легко визначити який тип даних може ефективніше використовуватись у тестуванні;

- Допомагає в оптимізації коду;
- Непотрібні рядки можуть бути видалені, що може показати прихований дефект;
- Завдяки знанням тестувальника про код, досягається максимальне покриття при написанні сценарію тестів.

White-box переваги:

- Ціна збільшується за рахунок проведення тестування більш кваліфікованим спеціалістом;
- Іноді неможливо продивитись увесь об’єм даних та коду, щоб знайти приховані помилки та дефекти через те, що багато шляхів лишається непротестованими
- Важко підтримувати, оскільки даний процес потребує спеціальних засобів та інструментів, наприклад, аналізатори коду та інструменти налагоджування тощо.

Існує багато прийомів тестування ПЗ. Деякі з них перевершують інші, деякі можна використовувати сукупно з іншими для кращих результатів. Нижче представлений список основних прийомів тестування:

- **Ручне тестування** – тести виконуються людьми з наперед складеними, або визначеними для кожного випробування тестовими даними.
- **Автоматизоване тестування** – тести виконуються спеціальними інструментами або самостійними процесами і можуть повторюватися багато раз. Тестові дані попередньо вводяться або генеруються.

- **Регресивне тестування** – тести, зазвичай автоматизовані, мають на меті виявлення негативного впливу змін в програмі на функції, що пройшли попередні перевірки.

- **Димове тестування** – тести, спрямовані на швидку перевірку базової функціональності, з метою виявлення, чи новий білд (версія програми чи її певного модуля) вартий тестування.

- **Дослідницьке тестування –** тести, що виконуються за відсутністю специфікацій. Тестер розроблює власну систему тестування, яка базується на накопиченому їм досвіді та оцінює ризики, створюючи сценарії тестування.

- **Мавпяче тестування -** тести, що не мають під собою певної системи, «швидка атака» програми тестером.

- **Стрес-тестування** – тести призначені для перевірки стійкості програми до надмірного навантаження при нестачі ресурсів

- **Тестування навантаження** – тести виконуються при різних рівнях навантаження з метою перевірити поведінку програми та виявити максимально дозволений рівень.

- **Тестування продуктивності** – тести виконуються для порівняння поточної продуктивності з розрахунковою.

- **Тестування інсталяції** – тести полягають у встановленні програми на різних платформах-комбінаціях та перевірки: чи всі файли було переписано, чи працює програма коректно.

- **Тестування довгим використанням –** тести виконуються довготривало, з метою виявлення такого роду помилок, що неможливо виявити при короткому використанні (наприклад, помилки при роботі з динамічним розподілом пам'яті).

1.5 Висновки

У даному розділі було досліджено, проаналізовано, описано та виділено основні характеристики та риси тестування програмного забезпечення. Як можна бачити, тестування дійсно займає значне місце в життєвому процесі розробки програмного забезпечення. Цей процес дуже ресурсозатратний, тому в наш час активно розвивається налагодження та оптимізація процесів тестування програмного забезпечення, а саме автоматизація тестування програмного забезпечення. Проаналізувавши особливості цього підходу можна виділити деякі основні порівняльні характеристики між підходом автоматизації та ручним тестуванням.

Таблиця 1 – Порівняльні характеристики автоматизованого та ручного тестування

Automation	**Manual**
Точно виконує операцію задану кількість разів	Не є точним за увесь час проведення
Дуже корисне при регресії, коли ПЗ постійно змінюється	Перше виконання дуже корисне, але при частій зміні ПЗ не завжди знаходиться регресійні дефекти
Ефективно при високій частоті запуску тестування	Ефективно при потребі проходження сценаріїв 1-2 рази
Після організації та виконання автоматичного тестування потрібна менша кількість тестувальників	Для кожного разу по проходженню тестового плану потрібна така ж кількість тестувальників

Може виконуватись з різними комбінаціями платформ та середовищ, а також на різних машинах одночасно	Для подібного виконання потрібно одночасно декілька тестувальників
Працює швидше ніж людські ресурси	Повільніше за автоматизацію
Не завжди допомагає в тестуванні UI	Дуже корисне в тестуванні UI
Дорожче	Дешевше

Отже, як можемо бачити позитивних сторін в автоматизації досить багато, що ще раз доводить актуальність даного напрямку розвитку та досліджень.

Опираючись на задачі та цілі даної роботи можна виокремити декілька основних факторів. А саме те, що розробка даного фреймворку для автоматизації тестування веб-додатку повинно значно полегшити проведення тестування таких видів та типів: black-box, функціональне, інтеграційне, регресійне, тестування інтерфейсу та при прийнятті роботи замовником. Розробка даної системи повинна суттєво скоротити витрати ресурсів на вищеописані фази та типи тестування, що повинно позитивно відобразитись на роботу та продуктивність команди загалом.

2. СИСТЕМА АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ. СИСТЕМНІ ЗАСОБИ, МЕТОДИ ТА ІНСТРУМЕНТИ

2.1 Автоматизація тестування веб-додатків

Більшість програмних продуктів, що випускаються на сьогоднішній день, є веб-орієнтовані додатки, що розраховані на роботу в інтернет-браузері. Ефективність тестування даних додатків досить відрізняється у різних компаній та організацій. У епоху високої інтерактивності і взаємодії в процесі розробки програм, коли численні організації використовують гнучкі методології розробки програмного забезпечення у різних формах, автоматизоване тестування дуже часто стає необхідністю. Під визначенням автоматизованого тестування мається на увазі використання інструментів для того, щоб багаторазово виконувати повторювані тести для додатку, що тестується. Регресійне тестування є найбільш типовим прикладом для використання даного підходу.

Автоматизоване тестування володіє багатьма перевагами, головним чином пов’язаних із високою швидкістю виконання тестів та можливістю виконувати однотипні тести раз за разом. Існує велика кількість комерційних та безкоштовних інструментів, котрі допомагають в розробці систем автоматизованого тестування ПЗ, конкретний вибір для даної роботи якого буде описаний далі.

Автоматизоване тестування забезпечує переваги, котрі можуть значно підвищити ефективність роботи відділу тестування у довгостроковій перспективі. Автоматизоване тестування дозволяє:

- Частіше проводити регресійне тестування
- Швидко надавати розробникам інформацію про стан продукту

- Отримати потенційно нескінченне число виконання тестів
- Надати підтримку гнучким та екстремальним методологіям розробки ПЗ
- Зберігати чітку документацію тестів
- Знаходити дефекти, що були пропущені на стадії ручного тестування

Досить часте питання про те, чи завжди вигідна автоматизація тестування та в яких випадках потрібно її використовувати. Не зважаючи на досить велику кількість переваг, не завжди корисно проводити автоматизацію тестів. Інколи ручне тестування може бути більш відповідним. Наприклад, коли графічний інтерфейс додатку досить сильно зміниться найближчого часу, автоматизовані тести доведеться переробляти. До того ж, іноді просто не вистачає часу на автоматизацію. У короткостроковій перспективі ручне тестування може буди більш ефективне. Якщо додаток повинен бути випущений в досить вузькі строки, у вас відсутнє автоматизоване тестування продукту, але протестувати в заданий період необхідно, то ручне тестування буде кращим рішенням.[8]

2.2 Спеціалізовані бібліотеки взаємодії з веб-інтерфейсом

Selenium – це комплект інструментів, кожен з яких передбачає свій власний підхід до автоматизації тестування. Більшість інженерів-тестувальників, котрі працюють з цим продуктом, фокусуються на одному-двох інструментах з цього набору, котрі краще за інших відповідають їх потребам. Проте вивчення всіх інструментів дозволяє краще розуміти всі доступні можливості для вирішення проблем, котрі виникають при автоматизації тестування. В сукупності набор інструментів Selenium надає

багатий вибір можливостей, зібраних разом для тестування усіх видів веб-додатків. Selenium надає декілька варіантів для ідентифікації елементів інтерфейсу, порівняння отриманого та очікуваного результатів поведінки веб додатку. Однією з ключових можливостей даного засобу є можливість запуску одних і тих самих тестів у різноманітних браузерах.

2.2.1 Набір інструментів Selenium

В рамках даного проекту розробляється серія інструментів для автоматизації тестування з відкритим вихідним кодом:

- Selenium WebDriver,
- Selenium RC,
- Selenium Server,
- Selenium Grid,
- Selenium IDE.

Selenium WebDriver. Це програмна бібліотека для управління браузерами. Досить часто використовують скорочену назву WebDriver.

WebDriver підтримує такі браузери і операційні системи:

- Google Chrome
- Microsoft Internet Explorer
- Microsoft Edge
- Mozilla Firefox
- Opera
- HtmlUnit
- PhantomJS
- Android (with Selendroid or appium)
- iOS (with ios-driver or appium)

Насправді це ціле сімейство драйверів для різноманітних браузерів, а також набір клієнтських бібліотек на різних мовах програмування, що дозволяють працювати з цими драйверами.

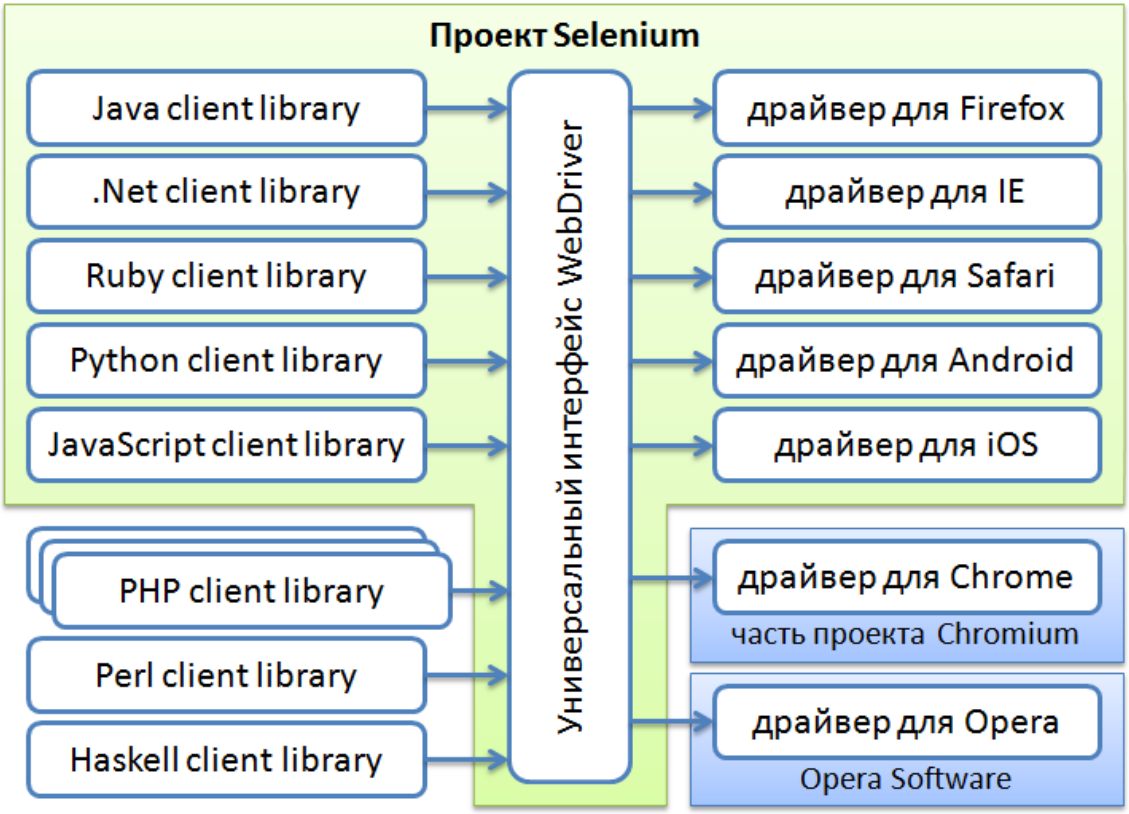

Рисунок 1 – структура інструмента Selenium WebDriver[9]

В рамках проекту Selenium розробляються драйвери для браузерів Firefox, Internet Explorer і Safari, а також драйвери для мобільних браузерів Android і iOS. Драйвер для браузера Google Chrome розробляється в рамках проекту Chromium, а драйвер для браузера Opera (включаючи мобільні версії) розробляється компанією Opera Software. Тому вони формально не є частиною проекту Selenium, поширюються і підтримуються незалежно. Але логічно можна вважати їх частиною сімейства продуктів Selenium.

Аналогічна ситуація також із клієнтськими бібліотеками - в рамках проекту Selenium розробляються бібліотеки для мов Java, .Net (C #), Python,

Ruby, JavaScript. Всі інші реалізації не мають відношення до проекту Selenium, хоча, можливо, в майбутньому, якісь з них можуть влитися в цей проект.

Selenium RC. Це попередня версія бібліотеки для керування браузерами. Абревіатура RC в назві цього продукту розшифровується як Remote Control, тобто це засіб для «віддаленого» управління браузером.

Ця версія з функціональної точки зору значно поступається WebDriver. Зараз вона знаходиться в законсервованому стані, не розвивається і навіть відомі баги не справляються. А всім, хто стикається з обмеженнями Selenium RC, пропонується переходити на використання WebDriver.

Іноді Selenium RC називається також Selenium 1.0, тоді як WebDriver називається Selenium 2.0. Хоча насправді дистрибутив версії 2.0 включає в себе одночасно обидві реалізації - і Selenium RC, і WebDriver. А ось коли вийде версія 3.0 - в ній залишиться тільки WebDriver.

З технічної точки зору WebDriver не є результатом еволюційного розвитку Selenium RC, вони побудовані на абсолютно різних принципах і у них практично немає загального коду. Об'єднує їх лише той факт, що обидві реалізації були зроблені в рамках проекту Selenium. Ну, або якщо бути зовсім точним, WebDriver спочатку був самостійним проектом, але в 2008 році відбулося злиття і зараз WebDriver є основний вектор розвитку проекту Selenium.

Selenium Server. Це сервер, який дозволяє керувати браузером з віддаленої машини, по мережі. Спочатку на тій машині, де повинен працювати браузер, встановлюється і запускається сервер. Потім на іншій машині (технічно можна і на тій же самій, звичайно) запускається програма, яка, використовуючи спеціальний драйвер RemoteWebDriver, з'єднується з сервером і відправляє йому команди. Він в свою чергу запускає браузер і виконує в ньому ці команди, використовуючи драйвер, що відповідає цьому браузеру:

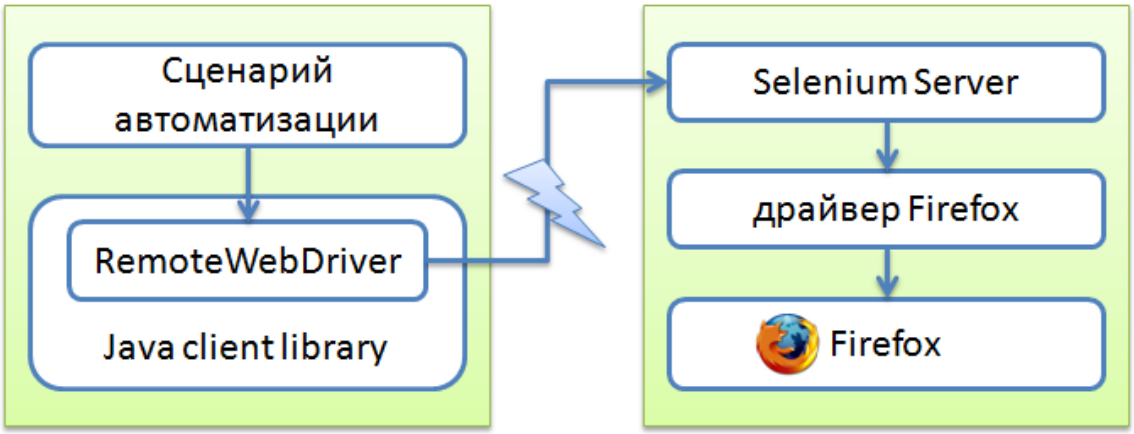

Рисунок 2 – Схема роботи Selenium Server[9]

Selenium Server підтримує одночасно два набори команд - для нової версії (WebDriver) і для старої версії (Selenium RC).

Selenium Grid. Це кластер, що складається з декількох Selenium-серверів. Він призначений для організації розподіленої мережі, що дозволяє паралельно запускати багато браузерів на великій кількості машин.

Selenium Grid має топологію «зірка», тобто в його складі є виділений сервер, який носить назву «хаб» або «комутатор», а решта сервера називаються «Ноди» або «вузли». Мережа може бути гетерогенною, тобто комутатор і вузли можуть працювати під управлінням різних операційних систем, на них можуть бути встановлені різні браузери. Одне із завдань Selenium Grid полягає в тому, щоб «підбирати» відповідний вузол, коли під час старту браузера вказуються вимоги до нього - тип браузера, версія, операційна система, архітектура процесора і ряд інших атрибутів.

Раніше Selenium Grid був самостійним продуктом. Зараз фізично продукт один - Selenium Server, але у нього є кілька режимів запуску: він може працювати як самостійний сервер, як комутатор кластера, або як вузол кластера, це визначається параметрами запуску.

Selenium IDE. Це плагін до браузера Firefox, який може записувати дії користувача, відтворювати їх, а також генерувати код для WebDriver або

Selenium RC, в якому виконуються ті ж самі дії. Загалом, це «Selenium-рекордер».

Тестувальники, які не вміють (або не хочуть) програмувати, використовують Selenium IDE як самостійний продукт, без перетворення записаних сценаріїв в програмний код. Це, звичайно, не дозволяє розробляти досить складні тестові набори, але деяким вистачає і простих лінійних сценаріїв.[9]

2.2.2 Висновок

Selenium надзвичайно гнучкий інструмент. Існує безліч способів додати функціональність як безпосередньо в самі тести, так і у фреймворк Selenium для індивідуального підстроювання ваших автоматичних тестів. Це можливо є найсильнішою стороною Selenium в порівнянні з іншими інструментами автоматизації. Ну і звичайно, так як Selenium є продуктом з відкритим кодом, ви завжди можете завантажити і модифікувати вихідний код.

Проаналізувавши потреби проекту, необхідні засоби та потрібні задачі для реалізації сміливо можна сказати, що вибір даного інструменту, зокрема WebDriver, ідеально підходить для розробки автоматизованого тестування для веб-додатка. Виділимо основні сильні сторони:

- Безкоштовна ліцензія
- Можливість роботи з різними браузерами
- Можливість роботи з різними мовами програмування
- Можливість роботи з різними операційними системами
- Можливість роботи з мобільними платформами
- Open source код бібліотеки (додаткова гнучкість інструмента)
- Постійна оновлення та актуалізація інструментів

- Хороша система підтримки користувачів
- Велика аудиторія користувачів (швидке знаходження відповіді на запитання)

2.3 Технічні методи розробки системи автоматизованого тестування веб-додатку

2.3.1 Різновиди тестів

Які частини вашого застосування слід тестувати? Це залежить від особливостей вашого проекту: користувальницьких очікувань, термінів, виділених на проект, пріоритетів, розставлених вашим проект-менеджером і тд. Але все ж, саме тестувальник має вирішувати, що і як тестувати.

Для категоризації видів тестів, придатних для веб-додатків, можна сформулювати кілька термінів. Дані терміни не є загальновизнаними, однак описувані тут методології широко використовуються при тестуванні веб-орієнтованих додатків.

Тестування статичного контенту. Найпростіший вид тестування - "контентне". Це простий тест, який перевіряє існування статичного, тобто. незмінюваного елемента користувальницького інтерфейсу. наприклад:

- Чи правильно озаглавлені сторінки? Цю перевірку можна використовувати для того, щоб упевнитися в тому, що тест виявив шукану сторінку після переходу за посиланням.
- Чи містить головна сторінка потрібну картинку зверху?
- Чи розташована контактна інформація, опис політики конфіденційності та інформація про торгову марку в футері сторінки?
- Чи стоїть на початку кожної сторінки заголовок з тегом <h1> Чи містить цей заголовок вірний текст?

Тестувати контент не завжди потрібно. Наприклад, якщо вміст вашої сторінки навряд чи буде змінюватися з часом, то ефективніше перевіряти його вручну. Якщо ж ваш додаток, наприклад, передбачає переміщення файлів, то контентне тестування може бути виправдане.

Тестування посилань. Биті посилання є частим джерелом помилок на веб сайтах. Тестування має на увазі перехід по кожному посиланні і перевірку того, що видана очікувана сторінка. Якщо статичні посилання не часто змінюються, то досить буде ручного тестування. Однак якщо ваші веб-дизайнери часто переробляють посилання, або ж файли періодично переміщаються, то тестування посилань слід автоматизувати.

Тестування функціональності. Ці тести спрямовані на перевірку певного функціоналу вашого додатку шляхом передачі йому користувацького введення і отримання певного результату. Дуже часто такий тест може складатися з переходів по декількох сторінках, введення даних в різні форми і поля, операцій підтвердження та скасування дій і перевірки результатів. Користувацьке введення може здійснюватися за допомогою текстових полів, випадаючих списків, чек-боксів та інших форм введення, які підтримує браузер.

Тестування функціональності - найчастіше найскладніший вид тестування, який доведеться автоматизувати, але він же і найважливіший. Зазвичай такими тестами виконується перевірка процесів реєстрації на сайті, входу в систему, операцій над обліковим записом користувача, зміни налаштувань, складних операцій з отримання даних, тощо. У більшості випадків функціональний тест копіює сценарії використання, що визначають функціонал і будову вашого застосування.

Тестування динамічних елементів. Найчастіше веб-елемент володіє унікальним ідентифікатором, який дає можливість знайти цей елемент на сторінці. Зазвичай такі ідентифікатори реалізовані за допомогою HTML атрибутів 'id' або 'name'. Ці значення можуть бути як статичними –тобто

незмінними строковими константами, так і динамічно генерованими значеннями, що відрізняються на різних сторінках сайту. Наприклад, якийсь веб-сервер може відображати ім'я документа як "doc3861" на одній сторінці, і як "doc6148" на інший, залежно від того, який документ запитує користувач. Це означає, що тестовому скрипту, котрий перевіряє, чи був документ створений правильно, може бути недоступний постійний відповідний документу ідентифікатор. Найчастіше сторінка з будь-якої видачою, що залежить від дій користувача, містить динамічні елементи з мінливими ідентифікаторами. Втім, конкретна реалізація, безумовно, залежить від роботи веб-додатка.

Тестування AJAX. AJAX - це технологія, що підтримує динамічна зміна елементів призначеного для користувача інтерфейсу без перезавантаження сторінки. Це, наприклад, може бути анімацією, RSS-стрічкою або оновленням даних в режимі реального часу. Існує незліченна кількість способів використання AJAX для оновлення веб-елемента на сторінці. Найпростіше це можна описати таким чином: в AJAX-орієнтованих додатках дані, що передаються сервером, потім відображаються на сторінці без її перезавантаження, оновлюється тільки її частина або ж безпосередньо змінений елемент.[10]

2.3.2 Перевірка результатів

Коли слід використовувати команду assert, а коли краще використовувати verify? Вирішувати самому, залежно від бажаної поведінки в разі, якщо результат перевірки виявиться негативним: чи щоб тест був зупинений або щоб продовжив роботу, записавши повідомлення про помилку.

Які ж плюси і мінуси? Якщо використовувати assert, то при виявленні помилки тест зупиниться, і наступні перевірки запускатися не будуть. У

деяких випадках, а іноді і в більшості, це те, що потрібно. Якщо тест не проходить, ви тут же дізнаєтеся, що результат тесту негативний. У більшості тестових фреймворків, таких як TestNG і JUnit, є розширення для найбільш поширених середовищ розробки, які умовно позначають такі тести, як провалені. Переваги: ви відразу ж бачите результат перевірки. Недоліки: коли тест зупиняється, в самому ньому залишаються невиконані перевірки, результат яких ви дізнатися не зможете.

Команда verify, навпаки, не завершує виконання тесту. Якщо ваш тест складається тільки з таких перевірок, то тест гарантовано (якщо допустити, що не відбудеться несподіваних помилок) буде виконаний, незалежно від того, чи будуть виявлені дефекти чи ні. Недоліки: необхідно витрачати більше зусиль на перевірку результатів тестування. TestNG і JUnit вже не надаватимуть додаткову інформацію. Доведеться відстежувати результати виконання тестів в консолі або в логах. Причому робити це кожен раз, коли запускається тест. Якщо ж є сотні тестів, і у кожного з них свій лог, це буде віднімати дуже багато часу. Тому через можливість миттєвої видачі результату тестування, команда assert використовується частіше команда verify.

2.3.3 Методи пошуку елементів

Існує кілька способів вибору об'єкт на сторінці. Але які ж плюси і мінуси у кожного з них? Можна виділити основні способи знаходження об'єкта:

- атрибут ID
- атрибут name
- вираження XPath
- по тексту посилання
- DOM-модель

З точки зору продуктивності, використання пошуку по атрибутах ID або name найбільш ефективне. До того ж, це дозволяє зробити тести більш чіткими (за умови так само читаємих в коді сторінки значень цих атрибутів).

Обробка XPath виразу вимагає більше часу, так як браузеру необхідно запускати його XPath обробник. Особливо повільно XPath працює в браузері Internet Explorer. Знаходження елементів по тексту посилання часто дуже швидке і зручне, однак, може бути застосовано тільки до посилань. До того ж, якщо самі посилання регулярно змінюються, пошук по тегу <a> буде значно зручніше.

І все ж, іноді вихідний код не містить атрибути ID або name і тоді не залишається іншого вибору, окрім як використовувати пошук по XPath. (Пошук по DOM-моделі в основному підтримується для забезпечення сумісності зі старими тестами, і зазвичай вже не використовуються, так як XPath може робити все те ж саме і навіть більше.)

Однак у XPath є і перевага перед пошуком по ID або name. За допомогою XPath (і DOM) можна знайти елемент відносно розташування інших елементів на цій сторінці. Наприклад, вам необхідно перевірити, що дане посилання знаходиться в другому параграфі тексту, всередині секції . Це можна задати за допомогою XPath. При використанні локаторів ID і name, можна перевірити лише факт наявності елемента на сторінці взагалі, без вказівки певного місця розташування. Так що якщо вам потрібно перевірити, що логотип компанії відображається нагорі сайту, в секції <header>, то слід використовувати локатор XPath.

Пошук AJAX елементів. Найкраще з виявленням і перевіркою AJAX елементів на сторінці справляється Selenium 2.0 WebDriver API. Він був спеціально спроектований з урахуванням необхідності тестування AJAX елементів, і саме в цій області у Selenium 1 існує ряд обмежень. У Selenim 2.0

існує метод waitFor (), який очікує завантаження елемента сторінки. У цей метод передається параметр - об'єкт класу "By", за допомогою якого в WebDriver реалізовані локатори.[11]

2.3.4 "Обгортки" викликів Selenium

У будь-якій мові програмування потрібно намагатися уникати дублювання коду, використовуючи його внутрішні службові функції. Одним із способів зробити це - "обернути" часто використовуються виклики в функції або власні методи. Наприклад, у великій кількості тестів потрібно натиснути на елемент сторінки і почекати поки вона оновиться. Замість того, щоб постійно дублювати цей код, потрібно написати свій власний метод, який буде виконувати обидві ці функції.

```
/**
 * Нажимает на элемент и ждет пока страница загрузится
 *
 * параметры: Локатор элемента и время ожидания
 *
 param elementLocator
 * param waitPeriod
 */
public void clickAndWait(String elementLocator, String waitPeriod) {
        selenium.click(elementLocator);
        selenium.waitForPageToLoad(waitPeriod);
}
```

Рисунок 3 – Приклад реалізації простого методу Selenium WebDriver

Інший поширений приклад використання обгорток для методів Selenium - перевірка наявність елемента на сторінці перед вчиненням над ним дій. Це називається "безпечною операцією". наприклад:

```
/**
 * Selenium-WebDriver -- Кликает на элемент, если он доступен на странице
 *
 * param elementLocator
 */
public void safeClick(String elementLocator) {
        WebElement webElement = getDriver().findElement(By.XXXX(elementLocator));
        if(webElement != null) {
                selenium.click(elementLocator);
        } else {
                // Используем TestNG API для логирования
                Reporter.log("Element: " +elementLocator+ ", is not available on page -
 "
                                + getDriver().getUrl());
        }
}
```

Рисунок 4 – Приклад реалізації обгортки «безпечної операції» Selenium

У цьому прикладі, замість 'XXXX' слід використовувати один з безлічі методів для пошуку елементів сторінки.

Використовувати "безпечний метод" слід на розсуд тестувальника. Якщо необхідно, щоб виконання тесту продовжилося навіть при відсутності елементу на сторінці, то можна використовувати "безпечний метод", спільно з логуванням повідомлення про неіснуючу елементі. Звичайно ж така реалізація має на увазі використання перевірки 'verify', замість перериваючої тест 'assert'. Але якщо елемент необхідний для здійснення подальших дій (наприклад, кнопка "увійти в систему" на головній сторінці), то дану техніку краще не застосовувати.

Карта призначеного для користувача інтерфейсу - практика написання коду, при якій всі локатори вашого тестового набору розташовуються в одному місці (у так званому локальному репозиторії), що спрощує їх подальшу модифікацію у разі, якщо ідентифікатори або шлях до елементів тестової програми були змінені. Скрипт використовує карту інтерфейсу для знаходження тестових елементів. Простіше кажучи, карта інтерфейсу це

репозиторій, в якому містяться об'єкти тестового коду, пов'язані з елементами інтерфейсу тестової програми.

Що ж корисного в карті інтерфейсу? Її головне завдання - полегшити обслуговування скрипта. Наприклад коли необхідно відредагувати локатор, то набагато простіше знайти об'єкт в єдиному центральному сховищі, ніж шукати його по всьому коду. До того ж, це дозволяє редагувати ідентифікатор в одному місці, а не змінювати кілька значень по всьому скрипту, або, того гірше, в коді відразу декількох скриптів.

Підсумовуючи, можна сказати, що використання карти інтерфейсу надає два значних переваги:

- замість того, щоб розкидати по всьому коду об'єкти користувацького інтерфейсу, використовується централізоване сховище. Це спрощує подальшу підтримку скриптів;
- дозволяє надати генерованим (зашифрованим) HTML-ідентифікаторам більш легкі для читання імена, що дозволяє поліпшити сприйняття коду.

Реалізувати карту призначеного для користувача інтерфейсу можна різними способами: зробити окремий клас або привласнити локаторам окремі глобальні рядкові змінні. В якості альтернативи можна використовувати текстовий файл, що містить пари ключ-значення.[11]

2.3.5 Тестування керованими даними

Тестуванням керованим даними (DDT, Data Driven Testing) називають процес, коли однаковий тест виконується безліч разів використовуючи змінювані тестові дані. Подібні набори даних, як правило, зберігаються в

зовнішніх файлах (наприклад .csv або текстовий файл) або завантажуються з баз даних. Техніка тестування DDT широко використовується, коли необхідно перевірити, як додаток обробляє різні вводяться значення. У такому випадку можна легко створювати нові тести, без зміни коду тестового скрипта, просто додаючи нові варіанти даних у файл.

```
# Строковый массив значений из файла
source = open("input_file.txt", "r")
values = source.readlines()
source.close()
# Цикл для вызова функции Поиска, для каждого значения из массива
for search in values:
    sel.open("/")
    sel.type("q", search)
    sel.click("btnG")
    sel.waitForPageToLoad("30000")
    self.failUnless(sel.is_text_present("Results * for " + search))
```

Рисунок 5 – Приклад реалізації DDT підходу

У наведеному вище прикладі спочатку відкривається файл. У цьому файлі містяться різні рядкові змінні, що містять пошуковий запит. Потім всі змінні зберігаються в окремому масиві. Потім в циклі для кожного значення з масиву виконується пошук і отриманий результат перевіряється.

2.3.6 Page Object паттерн

Page Object - це шаблон проектування, який широко використовується в автоматизованому тестуванні і дозволяє розділяти логіку виконання тестів від їх реалізації. Page Object як би моделює сторінки тестової програми в якості об'єктів в коді. В результаті його використання у вас вийдуть окремі класи, що

відповідають за роботу з HTML кожної конкретної веб-сторінки. Такий підхід значно зменшує обсяг повторюваного коду, тому що одні й ті ж об'єкти сторінок можна використовувати в різних тестах. Основна перевага Page Object полягає в тому, що в разі зміни призначеного для користувача інтерфейсу, можна виконати виправлення тільки в одному місці, а не виправляти кожен тест, в якому цей інтерфейс використовується.

Клас у Page Object не обов'язково повинен являти собою всю сторінку. Він може бути частиною сторінки, яка часто використовується на сайті (або навіть на одній сторінці). Це може бути, наприклад, меню. Основний принцип полягає в тому, що є тільки один клас у проекті, який знає про структуру HTML конкретної сторінки або її частини.

Поділ логіки і реалізації. Існує велика різниця між логікою тестування (що перевірити) і його реалізацією (як перевірити). Приклад тестового сценарію: «Користувач вводить невірний логін або пароль, натискає кнопку входу, отримує повідомлення про помилку». Цей сценарій описує логіку тесту, в той час як реалізація містить в собі такі дії як пошук полів введення на сторінці, їх заповнення, перевірку отриманої помилки і тд. І якщо, наприклад, змінитися спосіб виведення повідомлення про помилку, то це ніяк не вплине на сценарій тесту, все також потрібно буде ввести невірні дані, натиснути кнопку входу і перевірити помилку. Але це безпосередньо торкнеться реалізацію тесту - необхідно буде змінити метод, котрий отримує і обробляє повідомлення про помилку. При поділі логіки тесту від його реалізації автоматизовані тести стають більш гнучкими і їх, як правило, легше підтримувати.

Page Object в Selenium. Паттерн Page Object в Selenium реалізований за допомогою бібліотеки Page Factory і класу сторінки. Page Object являє собою окремий клас, що містить локатори елементів, методи для роботи з ними і конструктор, що приймає як параметр об'єкт WebDriver. Методи класу Page

Object можуть повертати об'єкти інших Page Object класів. За допомогою цього можна відтворити копію переходів і поведінки веб-додатки. Наприклад, метод успішної реєстрації в класі RegistrationPage повинен повертати екземпляр HomePage, тому що після реєстрації на сайті користувача перенаправляє на домашню сторінку. Одним із наслідків такого підходу є те, що необхідно моделювати як успішні, так і неуспішні методи. Або, наприклад, в разі якщо натискання на елемент може відкривати різні сторінки в залежності від умов, то також необхідно створювати різні методи для кожного необхідного випадку

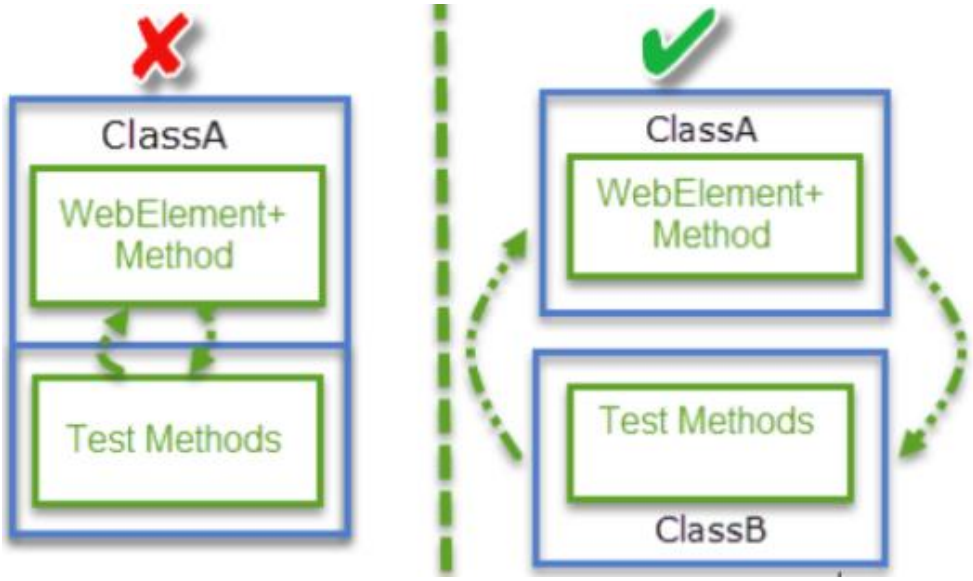

Рисунок 6 – Схематичний приклад Page Object підходу[9]

Page Object - це популярний в автоматизованому тестуванні шаблон проектування, який спрощує підтримку написаних тестів і зменшує кількість дубльованого коду. Це об'єктно-орієнтована клас, який виступає інтерфейсом веб-сторінки тестової програми. Методи даного класу використовуються в тесті при взаємодії з елементами призначеного для користувача інтерфейсу додатку. Великою перевагою є те, що при зміні дизайну користувальницького

інтерфейсу, потрібно виправляти самі тести, а тільки лише код всередині класу "Page Object". Відповідно, всі зміни, які необхідно здійснити для реалізації підтримки нового інтерфейсу будуть зосереджені в одному місці.[12]

Шаблон проектування "Page Object" має наступні переваги:

- Чіткий поділ між безпосередньо кодом тестів і залежить від сторінки коду, на кшталт локаторів (або їх застосування, якщо використовується карта призначеного для користувача інтерфейсу, розмітка);
- Наявність єдиного сховища для всіх служб і операцій, представлених на сторінці, а не безлічі розкиданих по всьому тестовому набору.

В обох випадках при змінах в інтерфейсі потрібно модифікувати код тільки в одному місці.

2.3.7 Перевірка результатів з використанням бази даних

Іншим поширеним видом тестування є порівняння даних для користувача інтерфейсу з даними, які зберігаються в базі даних тестової програми. Мови програмування дозволяють здійснювати запити до бази даних, завдяки яким можливо перевірити, що дані, які відображаються в вашому додатку, правильні.

Припустимо, що вам потрібно знайти в базі даних існуючий e-mail, і використовувати його на сайті через призначений для користувача інтерфейс. Приклад установки з'єднання з базою даних і відправкою запиту, виглядає наступним чином:

```
// Загружаем драйвер Microsoft SQL Server JDBC
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

// Сохраняем ссылку для соединения
String url = "jdbc:sqlserver://192.168.1.180:1433;DatabaseName=TEST_DB";

// Устанавливаем соединение с базой данных
public static Connection con =
DriverManager.getConnection(url, "username", "password");

// Создаем объект, который будет использоваться
// для DDL и DML SQL запросов.
public static Statement stmt = con.createStatement();

// Отсылаем с помощью метода Statement.executeQuery
// запросы SQL SELECT, которые вернут запрашиваемую
// информацию в объект класса ResultSet

ResultSet result =  stmt.executeQuery
("select top 1 email_address from user_register_table");

// Извлекаем значение "email_address"
String emailaddress = result.getString("email_address");

// Используем полученный emailAddress
// для входа в приложение
selenium.type("userID", emailaddress);
selenium.type("password", secretPassword);
selenium.click("loginButton");
selenium.waitForPageToLoad(timeOut);
Assert.assertTrue(selenium.isTextPresent("Добро пожаловать" +emailaddress), "Не удалось
 войти, используя " +emailaddress)
```

Рисунок 7 – Приклад реалізації перевірки з бази даних

2.4 Опис системи автоматизації функціонального тестування економічних застосувань.

Вище вже були описані основні програмні прийоми та засоби, що застосовувались у роботі, тому в цьому розділі буде наведено загальний опис тестованої програми, структура та метод розробки самого фреймворку для автоматизованого тестування. Також буде показана різниця між двома різними підходами до організації процесу розробки автоматизованого тестування ПЗ.

2.4.1 Відомості про продукт тестування

Продукт для розробки автоматизованого тестування - Автоматизована Банківська Система «БАРС» (Розробка компанії UNITY-BARS).

Продукт являє собою Автоматизовану банківську систему з комплексним або індивідуальним підходом до клієнта. Автоматизує широкий спектр бізнес-процесів та функцій банку. Забезпечення роботи декількох МФО в централізованій БД

Можливості та переваги використання АБС БАРС:

- Зберігає й обробляє великі об'єми даних (15 млн. клієнтів, 25 млн. рахунків, 10 тис. користувачів; щоденно обробляється більше ніж 600 тис. документів)
- Автоматизація повного спектру функціоналу банку всіх напрямків: фронт та бек офіс; РБ, МСБ та КБ, казначейство, бухгалтерія
- Інтерфейс користувача на базі веб-технологій, забезпечує роботу через повільні канали зв’язку. Відсутня необхідність оновлень клієнтського робочого місця
- Відповідність сучасним вимогам інформаційної безпеки. Побудова архітектури на базі за 3-рівневою моделлю
- Логічна організація функціоналу по принципу модульності, надає можливість використання лише необхідних функцій
- Підтримка існуючих вимог регулятивних органів та реалізація нових у межах супроводження

Продукт для демонстрації другого підходу для побудови системи автоматизованого тестування - Корпоративний WEB-banking

- Витримує навантаження в час пік 2-3 тисячі одночасно працюючих користувачів
- Система надає можливість одному користувачу постійний доступ до рахунків декількох клієнтів
- (юридичні особи, та корпорації)
- Корпоративний клієнт банк має більше ніж 20 000 користувачів

- Багаторівнева автентифікація
- Містить 200 000 рахунків та 300 000 -400 000 документів в день
- Імпорт документів різних форматів: 1С, текстовий формат, dbf-формат, на замовлення

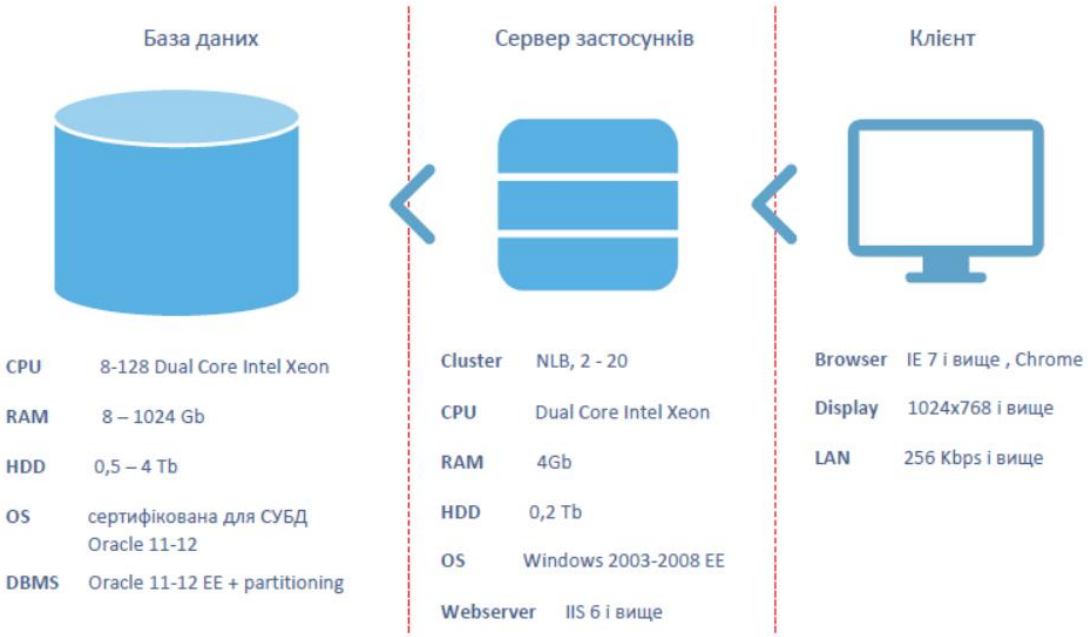

Рисунок 8 Архітектура тестованих продуктів[13]

2.4.2 Основні інструменти, методи та засоби реалізації

Ціль розробки фреймворку автоматизації, покриття видів тестування: перш за все, дана система автоматизації повинна покривати собою потреби регресійного тестування продукту, що включає в собі black-box тестування, функціональне тестування, частину тестування UI, частину acceptance тестування та виконувати функції презентації функціоналу (окремий набір тестів, як окрема частина acceptance)

Також в рамках даної роботи була розроблена додаткова система автоматизації тестування з однаковими технічними засобами, проте без використання чіткого планування та методологій проектування для виділення недоліків та переваг різних підходів при різних вимогах.

Основні інструменти та засоби реалізації:

- Написання фреймворку для тестування та власне тестів:
  - Мова програмування: C#
  - Середовище: Microsoft Visual Studio 15
  - Спеціалізовані бібліотеки та інструменти: Selenium Webriver, NUnit, NLog
  - Засоби для формування та перевірки запитів до БД: SQL, Toad for Oracle
- Ведення документації та звітності по тестуванню (в автоматичному режимі):
  - Test Rail, Jira
- Патерни та методики проектування проекту:
  - Page Object Model (POM)
  - BDD (behavior-driven development)

2.4.3 Особливості розробки системи автоматизації

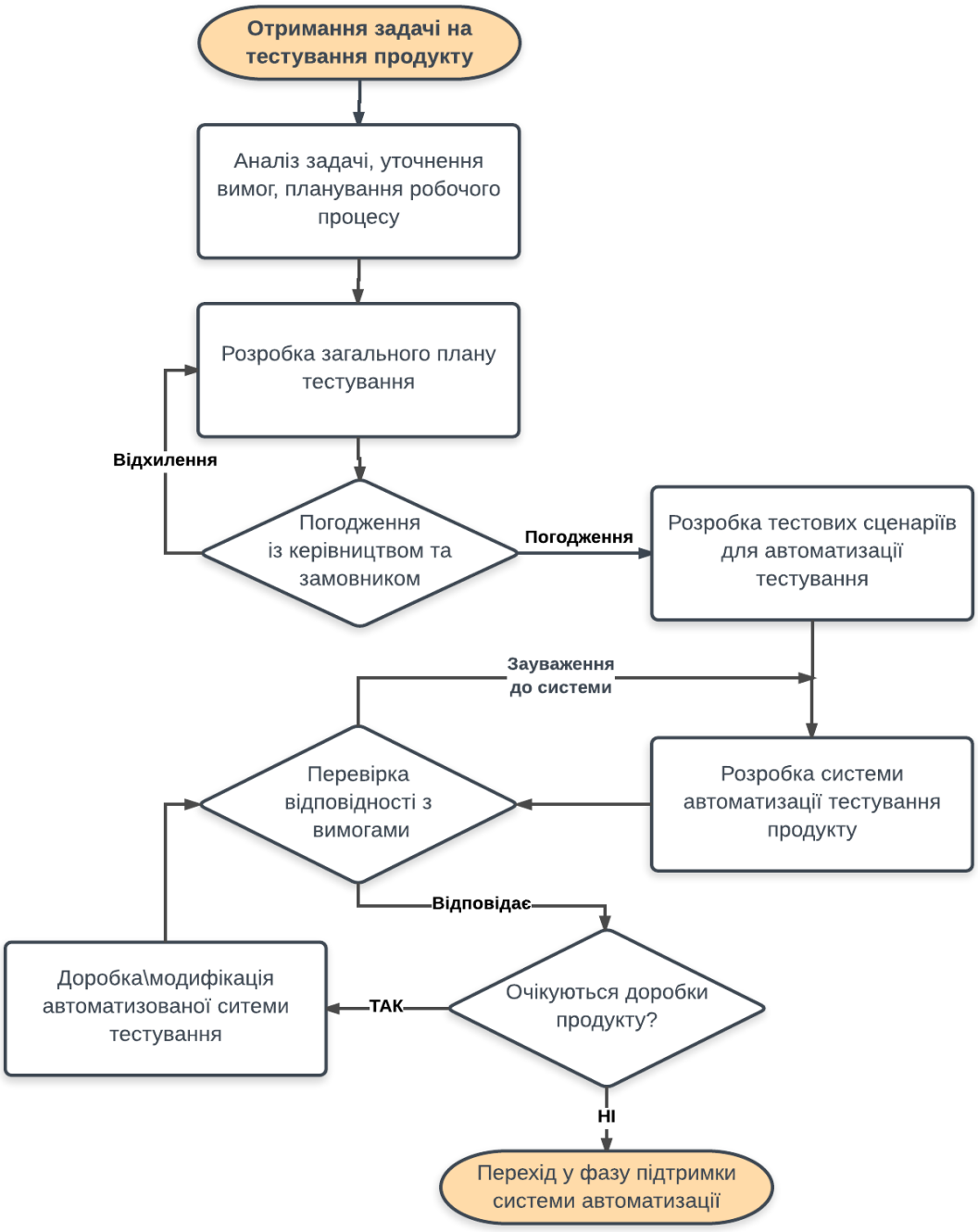

Рисунок 9 – Загальний алгоритм роботи над розробкою автоматизованої системи тестування продукту

Тут наведена загальна схема роботи тестувальника\відділу тестування при виконанні задачі щодо створення автоматизованої системи тестування для програмного забезпечення. Перш за все проводиться мітинг для аналізу потреб та особливостей задачі (у цьому випадку – автоматизувати тестування), команда обговорює та планує свої подальші дії, оцінює і розраховує свої можливості, час тощо. Після чого розробляється загальний план тестування або вивчається існуючий, якщо до цього часу цей функціонал вже проходив тестування або планувалось це саме проходження. Далі розроблений план погоджують з керівництвом та замовниками, уточнюють потреби замовника та видають остаточний варіант плану та підписують його (затверджують). Після чого виділяються та документуються конкретні сценарії для автоматизації, розставляються порядок та пріоритет автоматизації. Після вирішення документаційних питань, розпочинається безпосередньо етап розробки системи автоматизації тестування, в якому налаштовуються середовище, бібліотеки та інші додаткові інструменти, підключаються драйвери. Наступний етап – це написання базового фреймворку для роботи з бібліотеками автоматизованого тестування, веб-сторінками та їх елементами. Після описання базових методів (очікування, натискання елементів, пошук елементів тощо) приступається до розробки в залежності від методології та плану розробки автоматизації, про що буде йти мова нижче.

У кінці цього етапу на виході ми отримуємо описані тестові сценарії, з котрими вже можуть напряму працювати як ручні тестувальники команди, так і спеціалісти на стороні замовника. Проводиться верифікація результатів та перевірка на відповідність вимогам тест-плану та вимогам тестових сценаріїв. У випадку, якщо проект ще не завершився або заплановані зміни чи доробки, можна продивитись специфікацію та підготувати систему автоматизації до даних доробок або очікувати конкретних завдань по них, якщо ж такого в планах немає, то розробка системи автоматизації переходить у фазу підтримки.

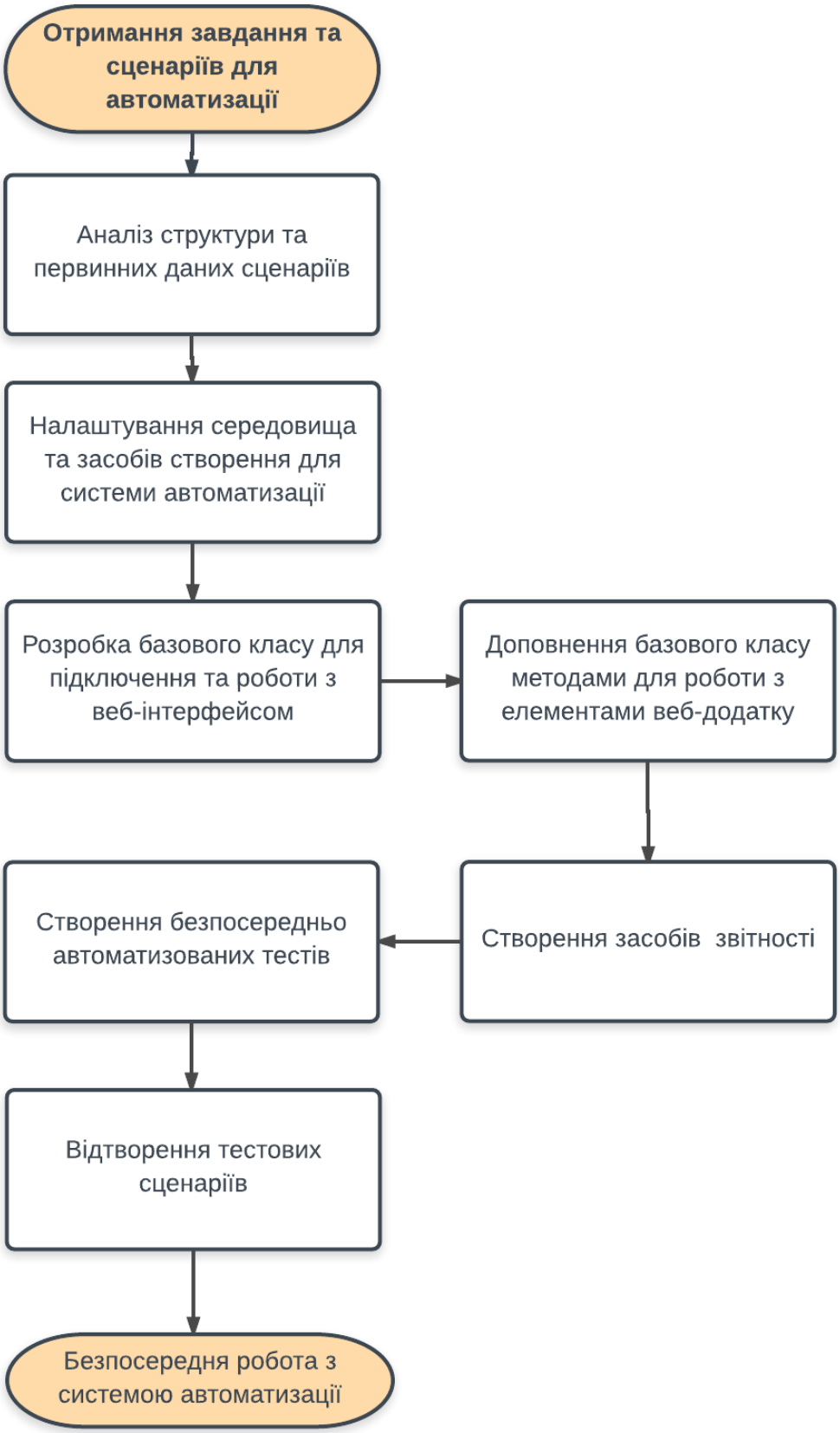

Рисунок 10 – Алгоритм розробки прототипу системи автоматизації

Далі більш детально опишу власне процес розробки прототипу системи автоматизованого тестування, відбувалося воно без застосування методологій та особливих застосувань, з першого погляду це може здаватись нераціональним, але в кінці опишу переваги такого підходу та коли його зручно застосовувати. Перш за все, при початку роботи над розробкою системи автоматизації проводиться аналіз структури тестових сценаріїв, що буди розроблені під автоматизацію тестування, акцентуються основні потреби та виділяються потрібні засоби і технології. Після чого власне ці засоби та різні бібліотеки налаштовуються разом з використовуваним середовищем. Після всіх налаштувань можна переходити безпосередньо до розробки самої систем та перше, що потрібно виконати – це створити базовий клас та описати базову роботу з підключенням драйверів з урахуванням особливостей тестування продукту (операційна система, браузери тощо), після чого розробляється та описується основний фреймворк з інструментами для взаємодії з веб-додатком та його різноманітними елементами, загалом це відбувається шляхом доповнення основного класу, також описуються методи роботи з очікуванням, робота з датами, опис основних маніпуляцій з веб-елементами, наприклад, пошук елемента, зчитування значень основних атрибутів та іншої інформації елемента, натискання, перетягування елементів, обробка вікон з повідомленнями, помилками та попередженнями і тд. Після розробки основної маси методів для опису тестів, створюється та описується клас для формування звітності, зокрема робота з використованою баг-трекінговою системою.

У результаті маємо цілком працюючий фреймворк для подальшої роботи з описом тестових сценаріїв автоматизації, що власне потім і робиться, використовуючи вже існуючу базу засобів описану спеціально під наш тестований продукт. Після опису тестових сценаріїв маємо повноцінну систему автоматизації тестування тестованого проекту. Виконуються приймальні роботи та задача переводиться у фазу підтримки цієї системи автоматизації.

Рисунок 11 – Алгоритм розробки системи автоматизації із використання спеціальних методологій та практик

Тут вже приведений більш розгорнутий алгоритм підходу до розробки системи автоматизованого тестування. Система будується на базі прототипу, що був розроблений раніше, тому деякі процеси на початку та в кінці будуть співпадати, проте у середині процесу досить відчутні зміни, що впливають на саму структури системи. Перша основна відмінність це створення загального репозиторію локаторів веб-елементів, що набагато полегшує подальшу роботу з усіма цими елементами та суттєво спрощує подальшу підтримку системи, так як для використання елементу потрібно просто звернутись до поля класу-репозиторію і при потребі зміни локатора потрібно змінити його тільки в репозиторії і методи, що його використовують, автоматично будуть бачити новий локатор. Проте це займає досить великий обсяг часу, тому що веб-елементів у проекті може бути дуже багато, але коли система автоматизації розробляється на довгу перспективу то цей етап просто необхідний для нормальної розробки системи та подальшої роботи з нею. У даному проекті для більшості локаторів елементів використовується XPath

XPath, або XML Path Language — це мова виразів, для визначення частини XML документа, або для обчислення величин (наприклад, рядкових, числових або булевих) на основі вмісту XML документа. XPath був розроблений World Wide Web Consortium (W3C).[15]

Приклад даного опису у робочому проекті:

- private const string ButtonFilterOfFunctionByCode = "//span[@data-field='RESOURCE_CODE']/span/span[@title='']";
- public const string ButtonSaveCreateARM = "//button[@id='saveCreateAdm-btn']";
- private const string ButtonMoveToTheNextPage = "//div[@id='ADMGrid']/div[3]/a/span[text()='Перейдіть на наступну сторінку']";

Далі доповнюється основний клас фреймворку інструментами для основних дій та маніпуляцій, при чому бажано умовно та візуально виділити блоками по принципу застосування. Далі наведений опис частини основних інструментів даної системи.

ToolsForTesting methods:

SELECTORS

- public IWebElement FindElementByXPath(string xPath) – Пошук елемента за xPath, повертає знайдений елемент
- public void FindElementByXPathAndClick(string xPath) – Пошук елемента та клік по ньому
- public void FindElementByXPathAndSendKeys(string xPath, string key) – Пошук елемента та запис тексту в нього
- public string FindElementByXPathAndGetAttribute(string xPath, string attribute) – Пошук елемента та отримання значення обраного атрибуту
- public string FindElementByXPathAndGetInnerText(string xPath) – Пошук елемента та витягування його внутрішнього тексту
- public void FindElementByXPathAndClearField(string xPath) – Пошук елемента та очищення поля вводу
- public IWebElement FindAllElements(string xPath) – Пошук усіх елементів з однаковим xPath, повертає видимий елемент
- public void FindAllElementsAndClickOnDisplayed(string xPath) – Пошук усіх елементів та клік по видимому
- public string FindAllElementsAndGetAttributeFromDisplayed(string xPath, string attribute) - Створює колекцію елементів, вибирає з неї видимий елемент та повертає його атрибут

- public bool ElementIsDisplayed(string xPath) – Перевірка видимості елемента
- public bool ElementIsDisplayedForCheckLoading(string xPath) – Перевірка видимості елемента для методу CheckLoading (різне логування)
- public bool ElementIsPresent(string xPath) – Перевірка наявності елемента
- public bool ElementIsSelected(string xPath) – Перевірка чи відмічений елемент (checkbox, radio batton, etc.)

GETS

- public string GetAttributeByXPath(string xPath, string attribute) – Отримання значення атрибута
- public string GetElementTextByXPath(string xPath) – Отримання тексту елемента
- public string GetDataFromDB(string sqlQuery, string columnNameOfSearchingRecords) – Підключення до БД та витягування даних через sql-запит

WAITS

- public void WaitForPageToLoad() – Очікування доки не завантажиться статус сторінки
- public void WaitForElementToBeClicable(string xPath) – Очікування клікабельності елемента
- public void WaitForElementIsVisible(string xPath) – Очікування видимості елемента
- public void WaitForElementToBeStalenessOf(string xPath) – Очікування видалення елементу з DOM для методу CheckLoading

- public void WaitForElementToBeStalenessOf_Element(IWebElement element) – Очікування видалення елементу з DOM
- public void WaitForTextToBePresentInElement(string xPath, string text) – Очікування присутності тексту в елементі
- public void WaitForFrameToBeAvailable(string xPath) – Очікування переходу до фрейма

ACTIONS

- public void ActionsMoveToElementAndClick_String(string xPath) - Перехід до елемента та клік по ньому, аргумент xPath
- public void ActionsMoveToElementAndClick_IWebElement(IWebElement desiredElement) - Перехід до елемента та клік по ньому, аргумент елемент
- public void ActionsMoveToElementAndDoubleClick(string xPath) - Перехід до елемента та подвійний клік по ньому
- public void ActiveElement_SendKeysOrText(string keysOrText) - Уведення клавіш або тексту, переважно для клавіш типу enter, tab
- public void ActivElement_Click() - Клік по активному елементу

OTHER

- public void SwitchToNewWindow(string baseWindow) - Перехід до нового вікна
- public void RefreshPage() – Оновлення сторінки
- public void SwitchToTheDefaultFrame() – Перехід до фрейма за замовчуванням

Після даних етапів маємо грамотний базовий інструментарій для подальшої роботи по створенню системи автоматизації. Далі відбувається опис методів для роботи з окремими частинами (веб-сторінками) проекту за принципом Page Object Modeling. Для даного проекту окремо можна виділити сторінку логування в систему, адже з неї починається майже кожен тест, та головну сторінку з головним меню, адже за структурою вона завжди активна, а основні дії відбуваються у окремому фреймі. Далі наведено приклад для двох класів-сторінок проекту:

1. Конструктор АРМів (ConstructorOfARMs)

1.1. Загальний функціонал сторінки(NavigationsInBlockOfARMs) – Логін, відкрити АРМ, обрати функцію, виконання передумов(завантаження вікна функцій), перечитати дані, навігація по сторінкам.

1.2. Фільтри в блоці АРМів (FiltersInBlockOfARMs) – Логін, фільтрування та сортування таблиці.

1.3. Створити АРМ (CreateARM) – Логін, створення АРМу та перевірка створення запису в БД (за полем ARMName)

1.4. Редагувати АРМ (EditARM) – Логін, вибір АРМу, його редагування та перевірка змін через БД (за полем ARMName)

1.5. Навігація у блоках ресурсів (NavigationsInBlockOfResourses) – загальна навігація

1.6. Фільтри в блоці ресурсів (FiltersInBlockOfResources) - – Логін, фільтрування та сортування таблиці

1.7. Додати функцію (ProvideFunctions) – додає вибраному арму права на вибрану функцію, після чого перевіряє зміни у БД

1.8. Вилучити функцію (RemoveFunctions) – забирає у вибраного арму права на вибрану функцію, після чого перевіряє зміни у БД

1. Адміністрування ролей (RolesAdministration)

1.1. (NavigationInBlockOfRoles) – базовий функціонал форми

1.2. (FiltersInBlockOfRoles) – фільтри блоку ролей

1.3. (CreateRole) – створення ролі

1.4. (EditRole) – редагування ролі

1.5. (BlockRole) – блокування ролі

1.6. (UnblockRole) – розблокування ролі

1.7. (NavigationInBlockOfARMs) – базовий функціонал форми

1.8. (FiltersInBlockOfARMs) – фільтри вибору АРМів

1.9. (ProvideARM) – доступ до АРМу

1.10. (RemoveARM) – закрити доступ до АРМу

У подальшому використання такої структури зумовлює роботу з окремими сторінками зверненням до конкретних методів для потрібних сторінок.

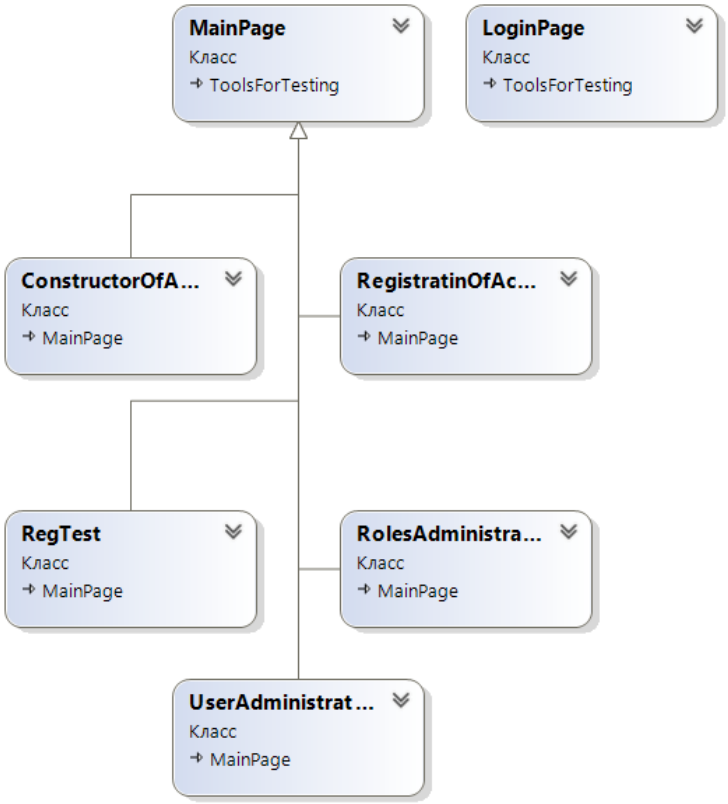

Рисунок 12 – Діаграма класів, по POM

Після цього іде налаштування системи логування роботи тестів. В даному проекті логи записуються у звичайному текстовому файлі, завдяки їм значно полегшується подальша робота зі звітністю та робота з системою загалом, адже тестувальник може вивчати логи після роботи системи та одразу виділяти та аналізувати проблеми, що виникли тощо. У даній роботі для логування виділено три рівні:

- INFO – для виводу загальної інформації, опису процесу проходження сценаріїв і тд.
- DEBUG – для логування системної інформації, котра не відноситься до загальної, можуть виводитися системні умови та не критичні попередження системи
- ERROR – виведення критичних помилок системи та некоректних результатів роботи сценарію

Загальна структура повідомлення логу:

Системна дата (YYYY-MM-DD HH:MM:SS.SSSS) | Рівень інформативності логу | Використовуваний клас команди | Повідомлення логу.

Наступний етап – створення і налаштування системи звітності. У даному проекті використовується систему баг-трекінгу Test Rail. Система автоматизації після проходження тесту заходить в окремому вікні у систему баг-трекінгу, знаходить про назві сценарію та тесту, записує результат проходження та виставляє відповідний статус проходження тесту.

Після виконаної роботи з наявними структурою та інструментарієм описуються власне тестові сценарії. Завдяки обраній структурі розробки системи автоматизації на цьому етапі опис тестів проходить вкрай легко, адже вже є прописані все елементи та локатори, методи для роботи з ними та описані основні робочі функції для роботи на кожній сторінці. Тож лишається тільки викликати потрібні методи та задати вхідні дані та очікуваний результат

Далі проводиться етап відтворення тестових сценаріїв та виправлення недоліків, що могли виникнути у ході розробки. Ось приклади кінцевого результату роботи тестів:

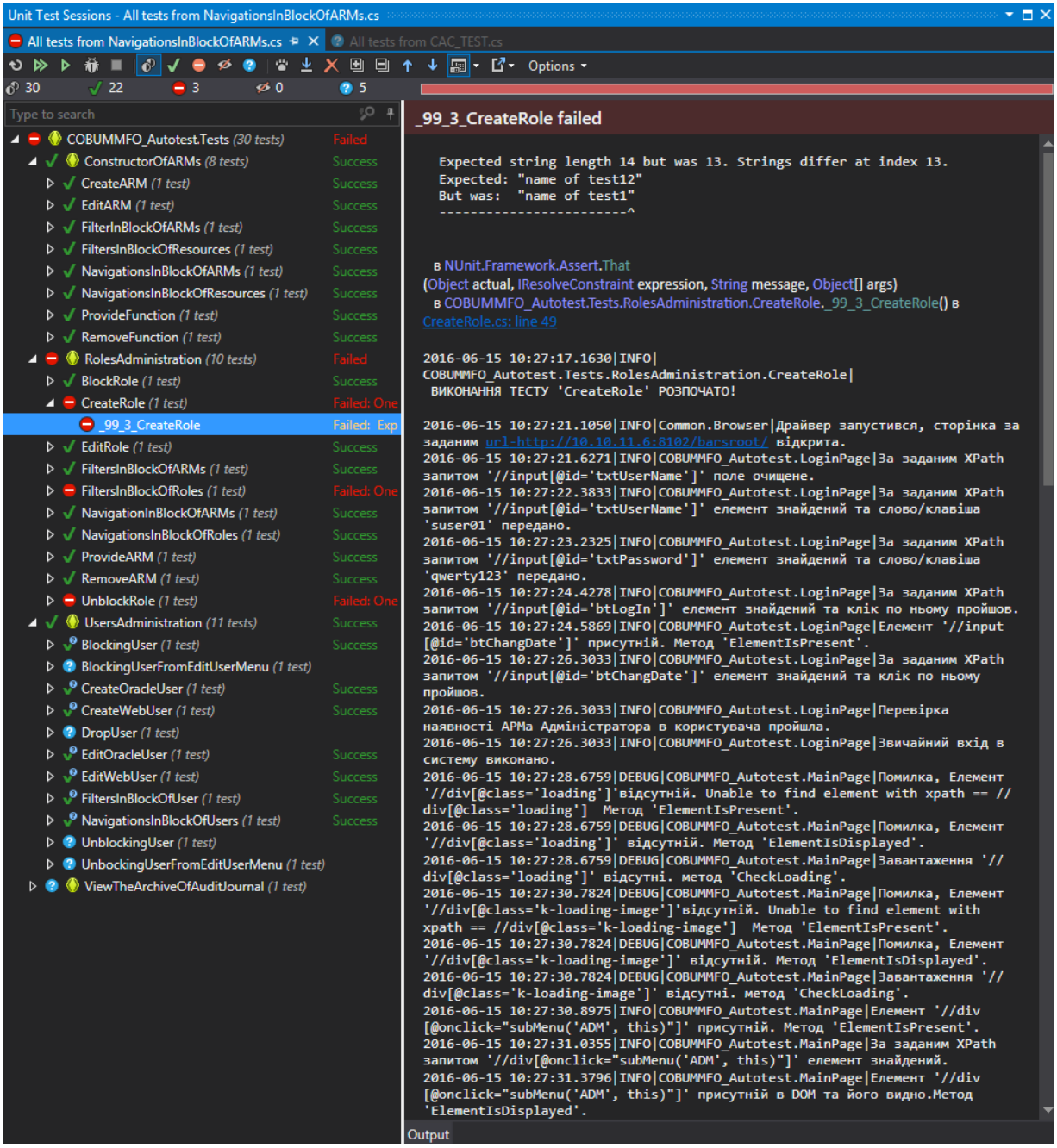

Рисунок 13 – Приклад виконання роботи системи тестування (Failure результат)

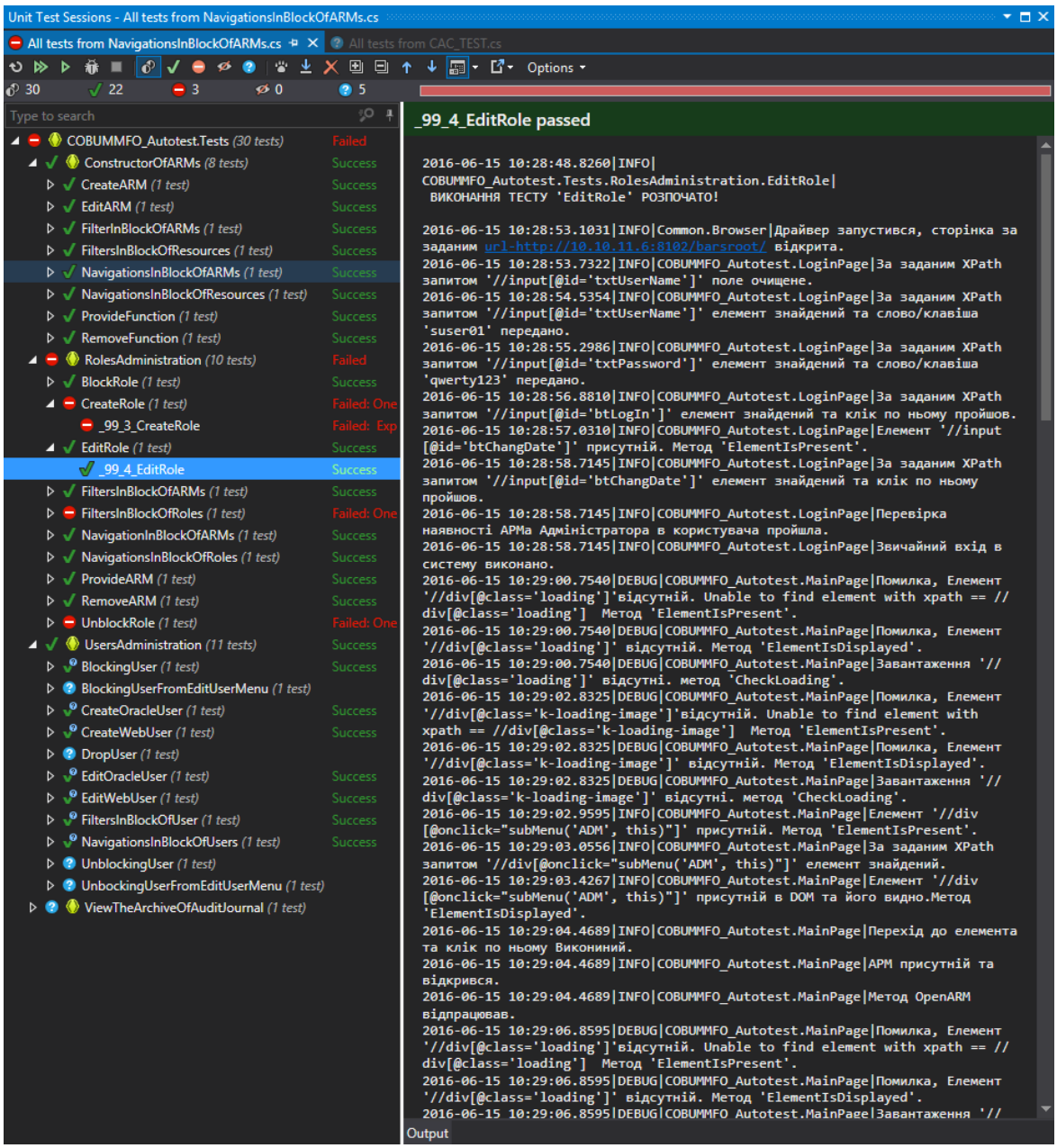

Рисунок 14 – Приклад виконання роботи системи тестування (Passed результат)

2.5 Висновки

У ході вищеописаних дій було розроблено прототип системи автоматизованого тестування та згодом саму систему на базі розробок прототипу. Приклад її структури наведено нижче. Проаналізувавши алгоритм

дій при розробці та кінцеву структуру можна виділити основні переваги та недоліки обох систем:

Прототип системи автоматизації тестування. Переваги:

- Потребує менше часу на виконання
- Проста структура системи
- Потребує менш кваліфікованих кадрів
- Зручна для невеликих тимчасових задач з дуже обмеженим часом виконання
- Зручна для навчання не надто досвідчених співробітників

Прототип системи автоматизації тестування. Недоліки:

- Не підходить для великого комерційного проекту
- Багато повторів коду
- Складна підтримка
- Багато не реалізованого функціоналу

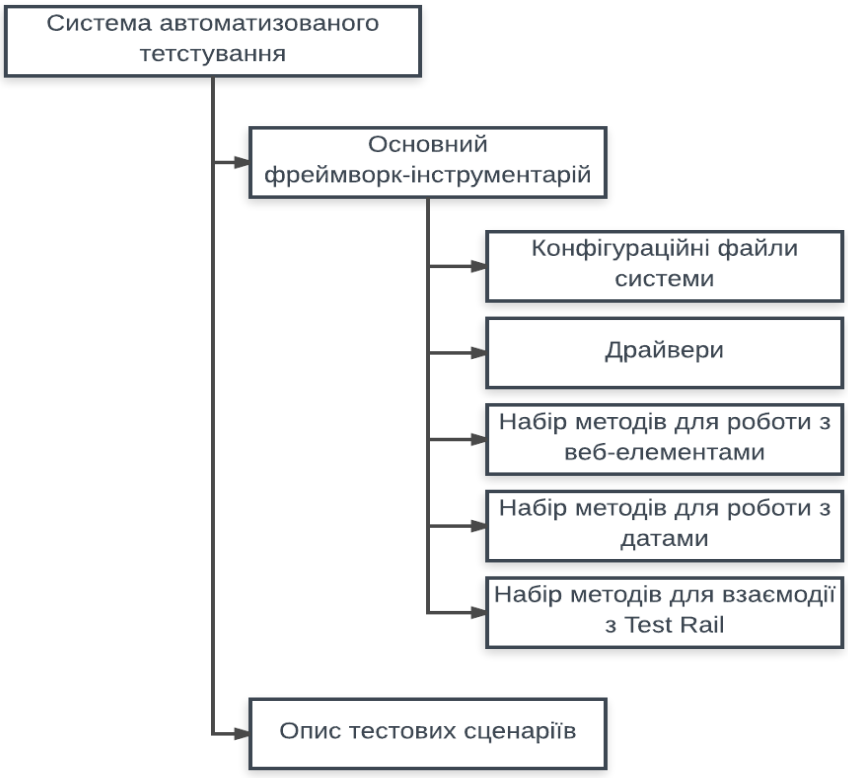

Рисунок 15 - Структура прототипу системи автоматизації тестування

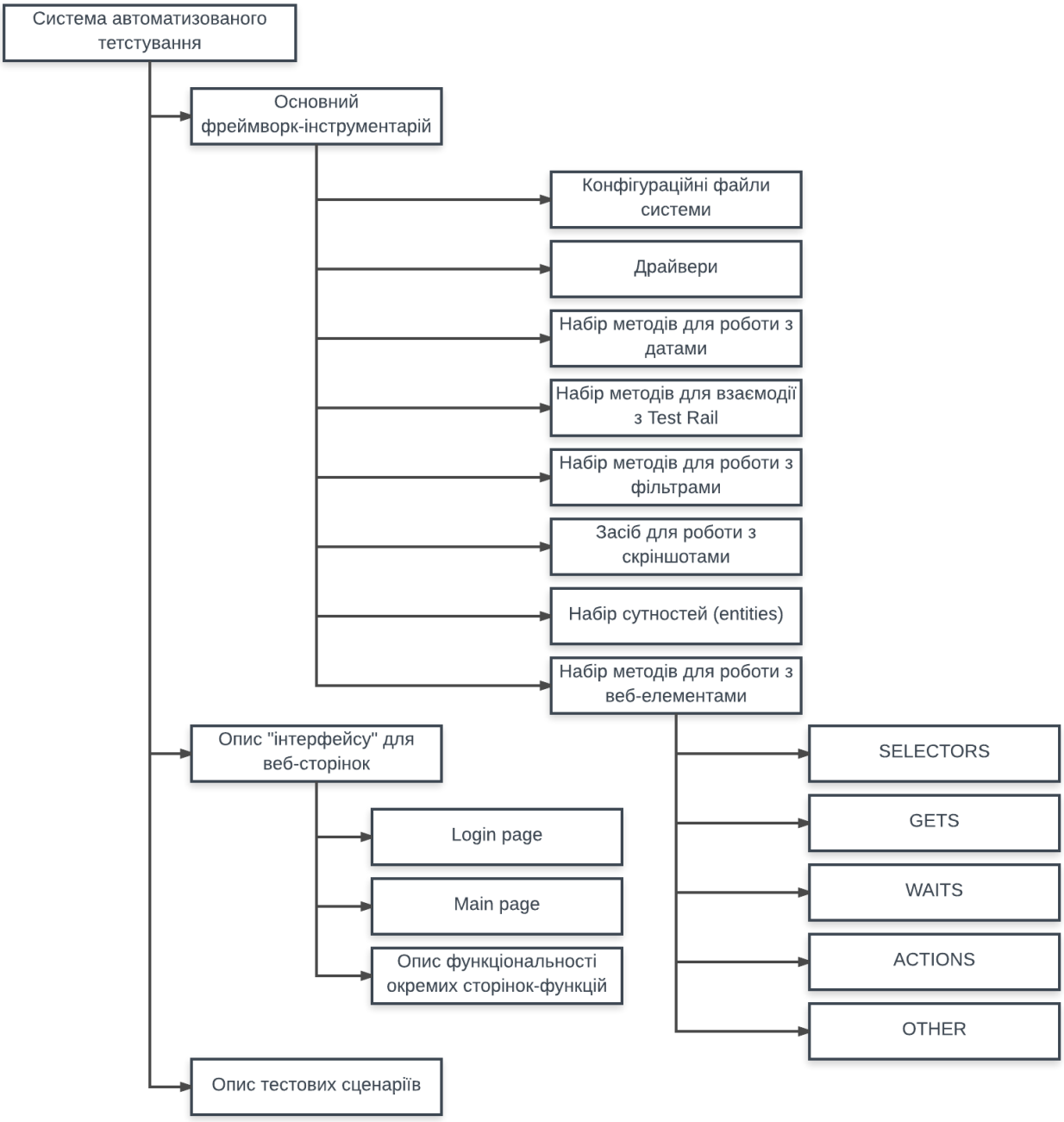

Рисунок 16 – Структура кінцевої системи автоматизації

Кінцева система автоматизації. Переваги:

- Система підходить для великих та складних проектів
- Досить широкий функціонал, що підходить під різноманітні задачі
- Зручна підтримка системи
- Покриває велику кількість видів тестування

Кінцева система автоматизації. Недоліки:

- Потребує більше часу на реалізацію
- Потребує більш кваліфікованих розробників
- Дорожча реалізація

Але не зважаючи на все, для достатньо серйозного комерційного проекту потрібно використовувати повний підхід для більшого покриття вимог та потреб замовника. У роботі були використано багато різних прийомів, технік та інструментів, котрі були описані вище. Так як така система автоматизованого тестування використовувалась у процесі розробки програмного забезпечення в комерційному проекті (додаток Б), то можна стверджувати, що дана робота цілком актуальна та корисна для реального процесу розробки програмного забезпечення.

3. КЕРУВАННЯ ТЕРМІНАМИ ВИКОНАННЯ ДИПЛОМНОЇ РОБОТИ. КЕРУВАННЯ РИЗИКАМИ

Тема: «Створення фреймворку для функціонального тестування економічних застосувань»

Об’єкт дослідження – засоби автоматизованого тестування ПЗ.

Предмет дослідження – використання засобів автоматизованого тестування ПЗ для створення системи автоматизованого тестування.

Задачі:

1. Дослідження інструментів та засобів для автоматизованого тестування веб-додатків
2. Розробка базового фреймворку для автоматизації тестування
3. Опис в системі автоматизованого тестування тестових сценаріїв

3.1 Керування термінами виконання дипломної роботи

Діаграма Ганта:

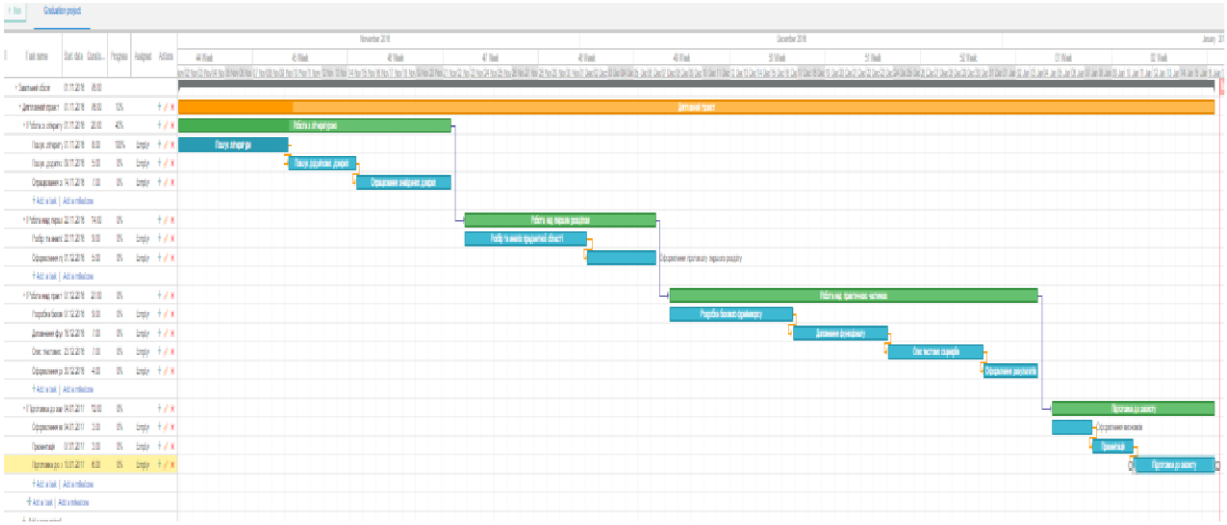

Рисунок 17 – Діаграма Ганта по процесу роботи над дипломним проектом

План роботи:

1. Робота з літературою
   1.1. Пошук літератури
   1.2. Пошук додаткових джерел
   1.3. Опрацювання знайдених джерел
2. Робота над першим розділом
   2.1. Розбір та аналіз предметної області
   2.2. Оформлення протоколу першого розділу
3. Робота над практичною частиною
   3.1. Розробка базового фреймворку
   3.2. Доповнення функціоналу
   3.3. Опис тестових сценаріїв
   3.4. Оформлення результатів
4. Підготовка до захисту
   4.1. Оформлення висновків
   4.2. Презентація
   4.3. Підготовка до захисту

Оцінка тривалості робіт (оптимістична (**О**)– песимістична (**Р**) – реалістична (**М**)).

1. Робота з літературою (20 – 30 – 22)
2. Робота над першим розділом (14 – 20 – 16)
3. Робота над практичною частиною (27 – 35 – 30)
4. Підготовка до захисту (12 – 15 – 13)

Таблиця 2 – Оцінка параметрів дипломної роботи

№	P	M	O	Дисп.	Ср.ст.от.	Te
1	30	22	20	2,56	1,6	23
2	20	16	14	1	1	16,(3)
3	35	30	27	1,7	1,33	30,(3)
4	15	13	12	0,25	0,5	13,1(6)
∑	100	81	73	20,25	4,5	82,8(3)

Z = 0,66; F(z) = 0,74

Відповідно, із вірогідністю 74% проект буде завершений за 85 днів

3.2 Керування ризиками

Таблиця 3 – Таблиця ризиків проекту

Ризик	Подія	Вірогідність	Вплив
Вихід з ладу робочого комп’ютера	Збереження даних в хмарних сховищах	Низька	Сильний
Зміна вимог до дипломної роботи	Розраховувати час із запасом	Висока	Середнє
Зміна ліцензії фреймворків, що використовуються у роботі	Переробка проекту з використанням засобів із відкритими ліцензіями	Низька	Сильний
Зайнятість наукового керівника	Підготовка чітко сформульованих питань на кожну з консультацій, підтримання спілкування в мережі	Середня	Сильний

Відсутність в мережі достатньої кількості літератури по темі дослідження	Заздалегідь дізнатися назви наукових журналів по даній тематиці, дізнатися наявність літератури в бібліотеці	Низька	Середній
Завантаження додатковою діяльністю (робота)	Виконання більшості задач заздалегідь до кінцевих строків	Середня	Середній
Відсутність достатніх знань та навичок у роботі з інструментами для виконання завдання	Виділення більшої кількості часу на вивчення та роботи над завданням, виконання більш чіткого планування	Середня	Слабкий

3.3 Висновки

Було спроектовано та сплановано процес підготовки дипломного проекту, розрахована вірогідність закінчення роботи в запланований термін дорівнює 74%, що є досить непоганим результатом для початкового планування, проте потрібно внести відповідні корективи в планування часу. Також було описано таблицю ризиків та описано засоби запобігання та протидії цим ризикам, відповідно вони і були застосовані.

ВИСНОВКИ

Дана дипломна робота присвячена розробці системи автоматизованого тестування програмного забезпечення, зокрема веб-додатку.

У першому розділі було досліджено та проаналізовано предметну область тестування програмного забезпечення. Розглянуто рівні, види та техніки тестування програмного забезпечення. Було виділено типи тестування, котрі повинні покриватись розробляємою системою автоматизації та виділено основні переваги цього підходу до тестування . Як вже було сказано, автоматизація тестування займає досить важливе місце у життєвому циклі розробки програмного продукту, так як після впровадження воно економить значні людські та часові ресурси відділу тестування.

У другому розділі вже було детально розглянуто технічні аспекти дипломної роботи. У основному розглядалась спеціалізований інструмент Selenium WebDriver, так як цей продукт є основним інструментом для побудови автоматизованого тестування веб-додатків. Було виділено основні переваги цього інструменту, котрі підтверджують правильність вибору цього інструменту для даного виду задач автоматизації. Розглянуто основні засоби та прийоми використання даної бібліотеки, котрі були задіяні у розробці проекту, зокрема підключення до бази даних та перевірка інформації через неї, обгортки простих функцій та методів у більш складні за структурою, але загальні за ціллю їх використання, використання XPath для локаторів веб-елементів, використанням методів очікувань тощо. Окрема увага була приділена шаблону проектування Page Object Modeling, так як використання її в структурі системи автоматизації тестування приносить досить багато переваг, що було показано у порівнянні з додатковою системою автоматизації. Цей шаблон проектування активно використовується у комерційних проектах, завдяки зручній структурованості проекту, що робить його більш легким для розуміння, а отже і для сумісної роботи над системою декількох людей та

значно полегшує роботу при підтримці системи автоматизованого тестування на проекті.

Дану систему було розроблено на реальних проектах, що підтверджують довідки в додатках, що підтверджує працездатність даної системи (акт впровадження наведений у додатку Б). Під ці ж продукти були розроблені та описані тестові сценарії під автоматизацію, котрі покривають більшу частину потреб регресійного, функціонального та UI тестувань. У кінці роботи було графічно зображено основну структуру систем та описано їх головні переваги та недоліки, котрі показали, що попри порівняну складність та більші затрати часу щодо деяких реалізацій та підходів, переваги, що отримуються завдяки цим діям, безперечно варті цього.

Перспективою для подальших досліджень є впровадження роботи з системою безперервної інтеграції та розробка засобів автоматичного сповіщення відділу тестування та розробки.Також перспективою є дослідження нижчого рівня взаємодії з елементами, що дозволить будувати більш гнучку систему автоматизації, а також розширення системи для автоматичного тестування сервісів та арі.

ПЕРЕЛІК ПОСИЛАНЬ

1. ISO/IEC 9126. 2001. Software engineering – Software product quality – Part 1: Quality model. Part 2: External metrics. Part 3: Internal metric. Part 4: Quality in use metrics [Text] — Geneva, Switzerland: International Organization for Standardization.
2. Текст лекцій до курсу «Технології розробки і тестування програм» Дідковська М. В. – Режим доступу: http://mmsa.kpi.ua/disciplines/sac/14c - Дата доступу: 15.11.2016.
3. Myers G.J. The Art Of Software Testing [Text] / G.J. Myers — New York: John Wiley & Sons, Inc., 2004. — 254 p.
4. Липаев В.В. Тестирование программ [Текст] / В.В. Липаев. — М.: Радио и связь, 1986. — 296 с.
5. Дейкстра Дисциплина программирования [Текст] / Э. Дейкстра ; пер. с англ. И. Х. Зусман ; ред. Э. З. Любимский. — М. : Мир, 1978. — 275 с.
6. ДСТУ 2844-94. Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення [Текст]. — Введ. 1.08.1995. — К.: Держстандарт України, 1995. — 57 с.
7. ДСТУ 2850-94. Програмні засоби ЕОМ. Показники і методи оцінювання якості [Текст]. — К.: Держстандарт України, 1994. – 20 с.
8. Інформаційно-навчальний портал автоматизації тестування – Режим доступу: http://autoqa.org/. – Дата доступу: 20.11.2016.
9. Офіційний сайт Selenium Browser Automation. – Режим доступу: http://www.selenium.org/. – Дата доступу: 20.11.2016.
10. Документація Selenium WebDriver. – Режим доступу: http://selenium2.ru/docs.html. – Дата доступу: 21.11.2016.

11. Selenium Documentation. – Режим доступу: http://www.seleniumhq.org/docs. – Дата доступу: 24.11.2016.
12. Інформаційно-навчальний портал по тестуванню та автоматизації тестування ПЗ. – Режим доступу: http://toolsqa.com/. Дата доступу: 30.11.2016.
13. Офіційний сайт компанії УНІТІ-БАРС(UNITY-BARS). – Режим доступу: http://unity-bars.com. – Дата доступу: 10.12.2016.
14. Документація Microsoft MSDN. – Режим доступу: https://msdn.microsoft.com/ru-ru/library. – Дата доступу: 10.12.2016.
15. World Wide Web Consortium (W3C) official site. – Режим доступу: https://www.w3.org/. – Дата доступу: 12.12.2016

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

Browser.cs

```
using System;
using OpenQA.Selenium;
using NLog;

namespace Common
{
    public class Browser : ToolsForTesting
    {
        public Browser(IWebDriver driver) : base(driver)
        {
        }
        private static Logger logger = LogManager.GetCurrentClassLogger();

        public static IWebDriver OpenPageDefault(IWebDriver driver, string url)
        {
            try
            {
                driver.Navigate().GoToUrl(url);
                driver.Manage().Window.Maximize();
                driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(2));
                logger.Info("Драйвер запустився, сторінка за заданим url-" + url + "
відкрита.");
                return driver;
            }
            catch(Exception e)
            {
                logger.Error("Помилка, Сторінка за заданим url НЕ відкрита." +
e.Message);
                return driver;
            }
        }
    }
}
```

Date.cs

```
using System;


namespace Common
{
    public class Date
    {
        public string Today()
        {
            return DateTime.Now.ToString("dd.MM.yyyy");
        }

        public string Tomorrow()
        {
            return DateTime.Now.AddDays(1).ToString("dd.MM.yyyy");
        }

        public string Yesterday()
        {
            return DateTime.Now.AddDays(-1).ToString("dd.MM.yyyy");
        }

        public string StartOfWeek()
        {
```

```
            DateTime date = DateTime.Now;

            while (date.DayOfWeek.ToString() != "Monday")
            {
                date = date.AddDays(-1);
            }

            return date.ToString("dd.MM.yyyy");
        }

        public string StartOfPrevWeek()
        {
            DateTime date = DateTime.Now;

            while (date.DayOfWeek.ToString() != "Monday")
            {
                date = date.AddDays(-1);
            }

            date = date.AddDays(-7);

            return date.ToString("dd.MM.yyyy");

        }

        public string StartOfMonth()
        {
            DateTime date = DateTime.Now;

            date = new DateTime(date.Year,date.Month,1);

            return date.ToString("dd.MM.yyyy");
        }

        public string StartOfPrevMonth()
        {
            DateTime date = DateTime.Now;

            date = new DateTime(date.Year, date.Month-1, 1);

            return date.ToString("dd.MM.yyyy");

        }

        public string EndOfPrevMonth()
        {
            DateTime date = DateTime.Now;

            date = new DateTime(date.Year, date.Month, 1);

            date = date.AddDays(-1);

            return date.ToString("dd.MM.yyyy");
        }

        public string StartOfQuater(string format = "")
        {
            DateTime date = DateTime.Now;
            int month = date.Month;
            int year = date.Year;

            switch (month)
```

```
            {
                case 1:
                    break;
                case 4:
                    break;
                case 7:
                    break;
                case 10:
                    break;
                default:
                    month--;
                    break;
            }

            date = new DateTime(year, month, 1);

            if (format != "")
            {
                return date.ToString("yyyy-MM-dd");
            }
            else
            {
                return date.ToString("dd.MM.yyyy");
            }
        }

        public string StartOfPrevQuater()
        {
            DateTime date = DateTime.Now;
            int month = date.Month;
            int year = date.Year;

            switch (month)
            {
                case 1:
                    break;
                case 4:
                    break;
                case 7:
                    break;
                case 10:
                    break;
                default:
                    month--;
                    break;
            }

            date = new DateTime(year, month, 1);

            date = date.AddMonths(-3);

            return date.ToString("dd.MM.yyyy");
        }

        public string EndOfPrevQuater()
        {
            DateTime date = DateTime.Now;
            int month = date.Month;
            int year = date.Year;

            switch (month)
            {
```

```
                case 1:
                    break;
                case 4:
                    break;
                case 7:
                    break;
                case 10:
                    break;
                default:
                    month--;
                    break;
            }

            date = new DateTime(year, month, 1);

            date = date.AddDays(-1);

            return date.ToString("dd.MM.yyyy");
        }
    }
}
```

Filter.cs

```
namespace Common
{
   public class Filter
    {
        public static readonly string ValueOfFilterEqualsForInt = "//li[text()='рівне']";
        public static readonly string ValueOfFilterNotEqualsForInt = "//li[text()='не
рівними']";
        public static readonly string ValueOfFilterEqualsForString =
"//li[text()='рівні']";
        public static readonly string ValueOfFilterNotEqualsForString = "//li[text()='не
рівні']";
        public static readonly string ValueOfFilterBegin = "//li[text()='починаються
на']";
        public static readonly string ValueOfFilterIn = "//li[text()='містять']";
        public static readonly string ValueOfFilterNotIn = "//li[text()='не містять']";
        public static readonly string ValueOfFilterEnd = "//li[text()='закінчуються
на']";                                                    //Values of filter
        public static readonly string ValueOfFilterGreaterOrEqual = "//li[text()='більше
або рівними']";
        public static readonly string ValueOfFilterLessThanOrEqual = "//li[text()='менше
або рівними']";
        public static readonly string ValueOfFilterGreater = "//li[text()='більше']";
        public static readonly string ValueOfFilterLess = "//li[text()='менше']";

        /// <summary>
        /// Вибирає значення фільтру типу містить, не містить, більше, менше...
        /// </summary>
        /// <param name="enterValueOfFilter">Значення фільтра</param>
        /// <returns></returns>
        public static string SelectTypeFilter(string enterValueOfFilter)
        {
            const string equalsForInt = "рівне";
            const string notEqualsForInt = "не рівними";
            const string equalsForString = "рівні";
            const string notEqualsForString = "не рівні";
            const string begin = "починаються на";
            const string end = "закінчуються на";
            const string _In = "містять";
            const string notIn = "не містять";
```

```
            const string greater = "більше";
            const string less = "менше";
            const string greaterOrEqual = "більше або рівними";
            const string lessThanOrEqual = "менше або рівними";

// Метод лише задає тип фільтра, він нічого не вибирає і не шукає
            string valueOfFilter;

            switch (enterValueOfFilter)
            {
                case equalsForInt:
                    valueOfFilter = ValueOfFilterEqualsForInt;
                    break;

                case notEqualsForInt:
                    valueOfFilter = ValueOfFilterNotEqualsForInt;
                    break;
                case equalsForString:
                    valueOfFilter = ValueOfFilterEqualsForString;
                    break;

                case notEqualsForString:
                    valueOfFilter = ValueOfFilterNotEqualsForString;
                    break;

                case begin:
                    valueOfFilter = ValueOfFilterBegin;
                    break;

                case end:
                    valueOfFilter = ValueOfFilterEnd;
                    break;

                case _In:
                    valueOfFilter = ValueOfFilterIn;
                    break;

                case notIn:
                    valueOfFilter = ValueOfFilterNotIn;
                    break;

                case greater:
                    valueOfFilter = ValueOfFilterGreater;
                    break;

                case less:
                    valueOfFilter = ValueOfFilterLess;
                    break;

                case greaterOrEqual:
                    valueOfFilter = ValueOfFilterGreaterOrEqual;
                    break;

                case lessThanOrEqual:
                    valueOfFilter = ValueOfFilterLessThanOrEqual;
                    break;

                default:
                    valueOfFilter = ValueOfFilterIn;
                    break;
            }
            return valueOfFilter;
```

```
        }
    }
}
```

Screenshot.cs

```
using OpenQA.Selenium;
using System.Drawing.Imaging;

namespace Common
{
    public class Screenshot1
    {
        public void TakeScreenshot(IWebDriver driver, string saveLocation)
        {
            ITakesScreenshot screenshotDriver = driver as ITakesScreenshot;
            Screenshot screenshot = screenshotDriver.GetScreenshot();
            screenshot.SaveAsFile(saveLocation, ImageFormat.Png);
        }
    }
}
```

TestRail.cs

```
using Common.Properties;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Interactions;
using OpenQA.Selenium.Support.UI;
using System;
using System.Collections.ObjectModel;

namespace Common
{
    public class TestRail
    {
        public IWebDriver Driver { get; set; }
        public string TestResult { get; set; }
        public string TestName { get; set; }
        public string TestComment { get; set; }
        public string TestRunName { get; set; }

        private string _username;
        private string _password;
        private string _url;
        private string _assignee;
        private string _addTestRunUrl;
        private string _testRunUrl;

        private void SetParams()
        {
            Driver = new ChromeDriver();
            Driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(20));
            _username = Config.Default.TRail_User;
            _password = Config.Default.TRail_Pass;
            _url = Config.Default.TRail_Url;
            TestRunName = Config.Default.TestRun_Name;
            _assignee = Config.Default.Assignee;
            _addTestRunUrl = _url + "/index.php?/runs/add/23";
            _testRunUrl = _url + "/index.php?/runs/overview/6";

            TestComment = "";
        }

        public TestRail(string run = "norun")
```

```
        {
            SetParams();
            Login();
            GetTestRunPage();

            if (run != "norun")
            {
                CreateTestRun();
            }
        }

        private void Login()
        {
            Driver.Url = _url;
            //Authorization
            IWebElement loginElement = Driver.FindElement(By.Id("name"));
            loginElement.SendKeys(_username);
            IWebElement passwordElement = Driver.FindElement(By.Id("_password"));
            passwordElement.SendKeys(_password);
            passwordElement.SendKeys(Keys.Enter);
        }

        public void GetTestRunPage()
        {
            Driver.Url = _testRunUrl;
            IWebElement testRunLink = Driver.FindElement(By.LinkText(TestRunName));
            testRunLink.Click();
        }

        private void CreateTestRun()
        {
            Driver.Url = _addTestRunUrl;

            IWebElement runName = Driver.FindElement(By.Id("name"));
            runName.Clear();
            runName.SendKeys(TestRunName);

            //Select _assignee
            SelectElement assigneeElement = new
SelectElement(Driver.FindElement(By.Id("assignedto_id")));
            assigneeElement.SelectByText(_assignee);

            //Submit form
            Driver.FindElement(By.TagName("button")).Submit();
        }

        public void SaveTestResult()
        {
            //Get current window name
            string baseWindow = Driver.CurrentWindowHandle;

            //Open test case in new window
            Actions action = new Actions(Driver);
            IWebElement caseLink = Driver.FindElement(By.LinkText(TestName));

            action.KeyDown(Keys.Shift).MoveToElement(caseLink).Click().Build().Perform();
            action.KeyUp(Keys.Shift).Perform();

            ReadOnlyCollection<string> handles = Driver.WindowHandles;

            foreach (string handle in handles)
            {
```

```
                if (handle != baseWindow)
                {
                    Driver.SwitchTo().Window(handle);
                }
            }

            Driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(20));


            //Add test result
            IWebElement addTestResultButton = Driver.FindElement(By.Id("addResult"));
            addTestResultButton.Click();

            //Add status
            SelectElement addResultStatus = new
SelectElement(Driver.FindElement(By.Id("addResultStatus")));
            addResultStatus.SelectByText(TestResult);

            //Add description
            IWebElement addResultComment = Driver.FindElement(By.Id("addResultComment"));
            addResultComment.SendKeys(TestComment);

            //Click add button
            IWebElement addResultSubmit = Driver.FindElement(By.Id("addResultSubmit"));
            addResultSubmit.Click();


            Driver.Close();

            Driver.SwitchTo().Window(baseWindow);

        }

        //private string Win1251ToUTF8(string source)
        //{

        //    Encoding utf8 = Encoding.GetEncoding("utf-8");
        //    Encoding win1251 = Encoding.GetEncoding("windows-1251");

        //    byte[] utf8Bytes = win1251.GetBytes(source);
        //    byte[] win1251Bytes = Encoding.Convert(win1251, utf8, utf8Bytes);
        //    source = win1251.GetString(win1251Bytes);
        //    return source;

        //}

        public void Quit()
        {
            Driver.Quit();
        }
    }
}
```

ToolsFprTesting.cs

```
using System;
using System.Collections.Generic;
using OpenQA.Selenium;
using OpenQA.Selenium.Support.UI;
using System.Collections.ObjectModel;
using OpenQA.Selenium.Interactions;
using System.Data.OracleClient;
using System.Data;
```

```
using System.Linq;
using NLog;

namespace Common
{
    public abstract class ToolsForTesting
    {
        public IWebDriver Driver;

        public WebDriverWait Wait;
        public Logger Log;


        protected ToolsForTesting()
        {
            Log = LogManager.GetCurrentClassLogger();
        }

        protected ToolsForTesting(IWebDriver driver)
        {
            Driver = driver;
            Log = LogManager.GetCurrentClassLogger();
        }

        protected ToolsForTesting(IWebDriver driver, WebDriverWait wait)
        {
            Driver = driver;
            Wait = wait;
        }

        protected static string Grid;              // Для метода ChooseFoundString

        // SELECTORS BEGAN

        /// <summary>
        /// Simple search by XPath
        /// </summary>
        /// <param name="xPath">XPath query</param>
        /// <returns>found element</returns>
        public IWebElement FindElementByXPath(string xPath)
        {
            try
            {
                Driver.FindElement(By.XPath(xPath));
                Log.Info("За заданим XPath запитом '" + xPath + "' елемент знайдений.");
                return Driver.FindElement(By.XPath(xPath));
            }
            catch (Exception e)
            {
                Log.Error("Помилка, За заданим XPath запитом '" + xPath + "' елемент НЕ
знайдений. " + e.Message);
                return Driver.FindElement(By.XPath(xPath));
            }
        }


        /// <summary>
        /// Search element by XPath and click on it
        /// </summary>
        /// <param name="xPath">desired element</param>
        public void FindElementByXPathAndClick(string xPath)
        {
```

```
            try
            {
                Driver.FindElement(By.XPath(xPath)).Click();
                Log.Info("За заданим XPath запитом '" + xPath + "' елемент знайдений та клік по ньому пройшов.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, За заданим XPath запитом '" + xPath + "' елемент НЕ знайдений або клік по ньому НЕ пройшов. " + e.Message);
            }
        }


        /// <summary>
        /// Search element by XPath and send 'key' on it
        /// </summary>
        /// <param name="xPath">desired element</param>
        /// <param name="key">key or word which will be sent</param>
        public void FindElementByXPathAndSendKeys(string xPath, string key)
        {
            try
            {
                Driver.FindElement(By.XPath(xPath)).SendKeys(key);
                Log.Info("За заданим XPath запитом '" + xPath + "' елемент знайдений та слово/клавіша '"+ key +"' передано.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, За заданим XPath запитом '" + xPath + "' елемент НЕ знайдений та слово/клавіша не передано. " + e.Message);
            }
        }


        /// <summary>
        /// Search element by XPath and send 'key' on it
        /// </summary>
        /// <param name="xPath">desired element</param>
        /// <param name="attribute">Attribute of tag which values will be received</param>
        /// <returns>Значення атрибута вибраного елемента</returns>
        public string FindElementByXPathAndGetAttribute(string xPath, string attribute)
        {
            try
            {
                Driver.FindElement(By.XPath(xPath)).GetAttribute(attribute);
                var textOfAttribute = Driver.FindElement(By.XPath(xPath)).GetAttribute(attribute);
                Log.Info("За заданим XPath запитом '" + xPath + "' елемент знайдений та атрибут '" + attribute + "' витягнуто. Значення атрибута: " + textOfAttribute);
                return textOfAttribute;
            }
            catch (Exception e)
            {
                Log.Error("Помилка, За заданим XPath запитом '" + xPath + "' елемент НЕ знайдений або атрибут НЕ витягнуто. Стек: " + e.Message);
                return "1";
            }
        }
```

```
        /// <summary>
        /// Search element by XPath and get inner text
        /// </summary>
        /// <param name="xPath">desired element</param>
        /// <returns>Внутрішній текст елемента</returns>
        public string FindElementByXPathAndGetInnerText(string xPath)
        {
            try
            {
                var textOfAttribute = Driver.FindElement(By.XPath(xPath)).Text;
                Log.Info("За заданим XPath запитом '" + xPath + "' елемент знайдений та
внутрішній текст: '" + textOfAttribute + "' витягнуто.");
                return textOfAttribute;
            }
            catch (Exception e)
            {
                Log.Error("Помилка, За заданим XPath запитом '" + xPath + "' елемент НЕ
знайдений або внутрішній текст НЕ витягнуто. Стек: " + e.Message);
                return "1";
            }
        }


        /// <summary>
        /// Search element by XPath and get inner text
        /// </summary>
        /// <param name="xPath">desired element</param>
        /// <returns>Внутрішній текст елемента</returns>
        public void FindElementByXPathAndClearField(string xPath)
        {
            try
            {
                Driver.FindElement(By.XPath(xPath)).Clear();
                Log.Info("За заданим XPath запитом '" + xPath + "' поле очищене.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, За заданим XPath запитом '" + xPath + "' поле НЕ
очищене. Стек: " + e.Message);
            }
        }


        /// <summary>
        /// Створює колекцію елементів, вибирає з неї видимий елемент та повертає його
        /// </summary>
        /// <param name="xPath">XPath селектор елемента</param>
        /// <returns>Видимий елемент</returns>
        public IWebElement FindAllElements(string xPath)
        {
            IReadOnlyCollection<IWebElement> elementList =
Driver.FindElements(By.XPath(xPath));
            foreach (IWebElement element in elementList.Where(element =>
element.Displayed))                    //This record equals >    foreach (IWebElement
element in elementList){
            {
//                                  if(element.Displayed)){...} }
                Log.Info("Видимий елемент знайдений та передався. Метод
'FindAllElements'.");
                return element;
            }
```

```
            Log.Error("Видимих елементів за запитом '" + xPath + "' НЕ знайдено. Метод
'FindAllElements'!");
            return null;
        }


        /// <summary>
        /// Створює колекцію елементів, вибирає з неї видимий елемент та клікає по ньому
        /// </summary>
        /// <param name="xPath">XPath селектор елемента</param>
        /// <returns>Видимий елемент</returns>
        public void FindAllElementsAndClickOnDisplayed(string xPath)
        {
            try
            {
                IReadOnlyCollection<IWebElement> elementList =
Driver.FindElements(By.XPath(xPath));
                foreach (IWebElement element in elementList.Where(element =>
element.Displayed))             //This record equals >    foreach (IWebElement element in
elementList){
                {
//                                  if(element.Displayed)){...} }
                    element.Click();
                    Log.Info("Видимий елемент знайдений та клік по ньому пройшов. Метод
'FindAllElementsAndClickOnDisplayed'.");
                }
                //Log.Error("Видимих елементів за запитом '" + xPath + "' НЕ знайдено.
Метод 'FindAllElementsAndClickOnDisplayed'!");
            }
            catch (Exception e)
            {
                Log.Error("Метод 'FindAllElementsAndClickOnDisplayed' НЕ відпрацював!
Стек: " + e.Message);
            }
        }


        /// <summary>
        /// Створює колекцію елементів, вибирає з неї видимий елемент та повертає його
атрибут
        /// </summary>
        /// <param name="xPath">XPath селектор елемента</param>
        /// <param name="attribute">Атрибут який повертається</param>
        /// <returns>Видимий елемент</returns>
        public string FindAllElementsAndGetAttributeFromDisplayed(string xPath, string
attribute)
        {
            try
            {
                IReadOnlyCollection<IWebElement> elementList =
Driver.FindElements(By.XPath(xPath));
                foreach (var a in elementList.Where(element =>
element.Displayed).Select(element => element.GetAttribute(attribute)))
//This records equals > foreach (IWebElement element in elementList.Where(element =>
element.Displayed))
                {
//                                  { string a = element.GetAttribute(attribute); }
                    Log.Info("Видимий елемент знайдений та атрибут переданий. Метод
'FindAllElementsAndGetAttributeFromDisplayed'.");
                    return a;
                }
```

```
                Log.Error("Видимих елементів за запитом '" + xPath + "' НЕ знайдено.
Метод 'FindAllElementsAndGetAttributeFromDisplayed'!");
                return null;
            }
            catch (Exception e)
            {
                Log.Error("Метод 'FindAllElementsAndGetAttributeFromDisplayed' НЕ
відпрацював! Стек: " + e.Message);
                return null;
            }
        }


        /// <summary>
        /// Checking the presence of element ...........
        /// </summary>
        /// <param name="xPath">XPath query</param>
        /// <returns>false or true</returns>
        public bool ElementIsDisplayed(string xPath)
        {
            if (ElementIsPresent(xPath))
            {
                try
                {
                    FindElementByXPath(xPath).Displayed.Equals(true);
                    Log.Info("Елемент '" + xPath + "' присутній в DOM та його видно.Метод
'ElementIsDisplayed'.");
                    return true;
                }
                catch
                {
                    Log.Info("Помилка. Елемент '" + xPath + "' присутній в DOM та його НЕ
видно.Метод 'ElementIsDisplayed'.");
                    return false;
                }
            }
            else
            {
                Log.Error("Помилка, Елемент '" + xPath + "' відсутній. Метод
'ElementIsDisplayed'.");
                return false;
            }
        }


        /// <summary>
        /// Checking the presence of element, only for 'CheckLoading' method
        /// </summary>
        /// <param name="xPath">XPath query</param>
        /// <returns>false or true</returns>
        public bool ElementIsDisplayedForCheckLoading(string xPath)
        {
            if (ElementIsPresent(xPath))
            {
                try
                {
                    FindElementByXPath(xPath).Displayed.Equals(true);
                    Log.Info("Елемент '" + xPath + "' присутній в DOM та його видно.Метод
'ElementIsDisplayed'.");
                    return true;
                }
                catch
```

```
                {
                    Log.Info("Помилка. Елемент '" + xPath + "' присутній в DOM та його НЕ
видно.Метод 'ElementIsDisplayed'.");
                    return false;
                }
            }
            else
            {
                Log.Debug("Помилка, Елемент '" + xPath + "' відсутній. Метод
'ElementIsDisplayed'.");
                return false;
            }
        }


        /// <summary>
        /// Checking the presence of element
        /// </summary>
        /// <param name="xPath">XPath query</param>
        /// <returns>false or true</returns>
        public bool ElementIsPresent(string xPath)
        {
            try
            {
                Driver.FindElement(By.XPath(xPath));
                Log.Info("Елемент '" + xPath + "' присутній. Метод 'ElementIsPresent'.");
                return true;
            }
            catch (Exception e)
            {
                Log.Debug("Помилка, Елемент '" + xPath + "'відсутній. " + e.Message + "
Метод 'ElementIsPresent'.");
                return false;
            }
        }


        /// <summary>
        /// Перевірка чи відмічений елемент (checkbox, radio batton, etc.)
        /// </summary>
        /// <param name="xPath">XPath селектор елемента</param>
        /// <returns></returns>
        public bool ElementIsSelected(string xPath)
        {
            try
            {
                if (FindElementByXPath(xPath).Selected)
                {
                    Log.Info("Елемент '" + xPath + "' відмічений (вибраний). Метод
'ElementIsSelected'.");
                    return true;
                }
                else
                {
                    Log.Error("Елемент '" + xPath + "' НЕ відмічений (НЕ вибраний)! Метод
'ElementIsSelected'.");
                    return false;
                }
            }
            catch (Exception e)
            {
                Log.Error("Метод 'ElementIsSelected' НЕ відпрацював: " + e);
```

```
                return false;
            }
        }

        //////public void SelectElementByText(string id, string text)
        //////{
        //////    WaitForPageToLoad();
        //////    SelectElement element = new
SelectElement(Driver.FindElement(By.Id(id)));
        //////    element.SelectByText(text);
        //////    WaitForPageToLoad();
        //////}

        //////public void SelectElementByOptionNumber(string id, int number)
        //////{
        //////    WaitForPageToLoad();
        //////    SelectElement element = new
SelectElement(Driver.FindElement(By.Id(id)));
        //////    element.SelectByIndex(number);
        //////    WaitForPageToLoad();
        //////}

        //////public void SelectFromDirectoryById(string id, int elementNumber, int
pageNumber = 1)
        //////{
        //////    ClickElementById(id);
        //////    WaitForPageToLoad();
        //////    SwitchToFrameByXPath("//*[@id='iframeContent']/iframe");
        //////    if (pageNumber != 1)
        //////    {
        //////        ClickElementByTextLink(pageNumber.ToString());
        //////    }
        //////
ClickElementByXPath("//*[@id='ctl00_MainContentPlaceHolder_References_gv']/tbody/tr[" +
elementNumber.ToString() + "]");
        //////    ClickElementById("ctl00_MainContentPlaceHolder_btSelect");
        //////    SwitchToTheDefaultFrame();
        //////}

        //////public void SetElementAttributeById(string id, string attribute, string
value)
        //////{
        //////    WaitForPageToLoad();
        //////    IWebElement element = Driver.FindElement(By.Id(id));
        //////    IWrapsDriver wrappedElement = element as IWrapsDriver;
        //////    if (wrappedElement == null)
        //////        throw new ArgumentException("element", "Element must wrap a web
driver");

        //////    //IWebDriver driver = wrappedElement.WrappedDriver;
        //////    IJavaScriptExecutor javascript = Driver as IJavaScriptExecutor;
        //////    if (javascript == null)
        //////        throw new ArgumentException("element", "Element must wrap a web
driver that supports javascript execution");

        //////    javascript.ExecuteScript("arguments[0].setAttribute(arguments[1],
arguments[2])", element, attribute, value);
        //////    WaitForPageToLoad();
        //////}

        // SELECTORS END
        // GETS BEGAN
```

```
        /// <summary>
        /// Find element by XPath and get attribute from it
        /// </summary>
        /// <param name="xPath">XPath query</param>
        /// <param name="attribute">attribute of element</param>
        /// <returns>return attribute of element or 1</returns>
        public string GetAttributeByXPath(string xPath, string attribute)
        {
            try
            {
                WaitForPageToLoad();
                var a = FindElementByXPath(xPath).GetAttribute(attribute);
                Log.Info("Атрибут витягнутий з елемента.");
                return a;
            }
            catch(Exception e)
            {
                Log.Error("Помилка, Не вдалося витягнути атрибут з елемента. " +
e.Message);
                return "1";
            }


        }

        /// <summary>
        /// Found web element by XPath and get text from it
        /// </summary>
        /// <param name="xPath">Xpath query</param>
        /// <returns>return text of element or 1</returns>
        public string GetElementTextByXPath(string xPath)
        {
            try
            {
                WaitForPageToLoad();
                var text = FindElementByXPath(xPath).Text;
                Log.Info("Текст витягнутий з елемента.");
                return text;
            }
            catch(Exception e)
            {
                Log.Error("Помилка, Не вдалося витягнути текст з елемента. " +
e.Message);
                return "1";
            }

        }

        /// <summary>
        /// Підключення до БД та витягування даних через sql-запит
        /// </summary>
        /// <param name="sqlQuery">Запит sql, БЕЗ ';' В КІНЦІ</param>
        /// <param name="columnNameOfSearchingRecords">Стовпчик в в якому шукаємо
запис</param>
        /// <returns>Повертає значення вибраного стовпчика</returns>
        public string GetDataFromDB(string sqlQuery, string columnNameOfSearchingRecords)
        {
            var dataFromDb = "";
            var oracleDbConnection = "Data Source=(DESCRIPTION="
                                                    + "(ADDRESS_LIST="
                                                        + "(ADDRESS="
```

```
                                                                + "(PROTOCOL=TCP)"
                                                                +
"(HOST=10.10.11.110)"
                                                                + "(PORT=1521)"
                                                                + ")"
                                                        + ")"
                                                        + "(CONNECT_DATA="
                                                                +
"(SERVER=DEDICATED)"
                                                                +
"(SERVICE_NAME=OBMMFOC)"
                                                        + ")"
                                                + ");"
            + "User Id=bars;Password=barsbars;";

            OracleConnection connnection = new OracleConnection(oracleDbConnection);

            try
            {
                connnection.Open();
                Log.Info(" З'єднання з БД успішно створено.");
            }
            catch (OracleException ex)
            {
                if (connnection.State != ConnectionState.Open)
                {
                    Log.Info(" Помилка. З'єднання з БД НЕ створено!");
                }
                Log.Error(ex.Message);
            }

            OracleCommand cmd = new OracleCommand(sqlQuery, connnection);
            {
                //using (OracleDataReader dataReader =
cmd.ExecuteReader(CommandBehavior.CloseConnection))
                using (OracleDataReader dataReader = cmd.ExecuteReader())
                {
                    while (dataReader.Read())
                    {
                        Log.Info("-> Значення стовпчика '" + columnNameOfSearchingRecords
+ "': {0}.", dataReader[columnNameOfSearchingRecords]);
                        dataFromDb = dataReader[columnNameOfSearchingRecords].ToString();
                    }
                }
            }
            connnection.Close();
            connnection.Dispose();
            //Console.ReadLine();
            Log.Info(" Дані з БД витягнуто.");
            return dataFromDb;
        }

        //////public string GetUrl()
        //////{
        //////    WaitForPageToLoad();
        //////    return Driver.Url;
        //////}

        //////public string GetCurrentWindowName()
        //////{
        //////    string baseWindow = Driver.CurrentWindowHandle;
        //////    return baseWindow;
```

```
        //////}

        // GETS END
        // WAITS BEGAN

        /// <summary>
        /// Wait, until status of page to be loaded
        /// </summary>
        public void WaitForPageToLoad()
        {
            WebDriverWait wait = new WebDriverWait(Driver, TimeSpan.FromSeconds(30));

            IJavaScriptExecutor javascript = Driver as IJavaScriptExecutor;
            if (javascript == null)
                throw new ArgumentException("driver", "Driver must support javascript
execution");

            wait.Until(d =>
            {
                try
                {
                    var readyState = javascript.ExecuteScript(
                    "if (document.readyState) return document.readyState;").ToString();
                    return readyState.ToLower() == "complete";
                }
                catch (InvalidOperationException e)
                {
                    //Window is no longer available
                    return e.Message.ToLower().Contains("unable to get browser");
                }
                catch (WebDriverException e)
                {
                    //Browser is no longer available
                    return e.Message.ToLower().Contains("unable to connect");
                }
                catch (Exception)
                {
                    return false;
                }
            });
        }


        /// <summary>
        /// Wait for element to be clicable
        /// </summary>
        /// <param name="xPath">XPath query</param>
        public void WaitForElementToBeClicable(string xPath)
        {
            try
            {
                var wait = new WebDriverWait(Driver, TimeSpan.FromSeconds(50));
                wait.Until(ExpectedConditions.ElementToBeClickable(By.XPath(xPath)));
                Log.Info("Очікування поки елемент '" + xPath + "' не стане клікабельним
Виконано.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Очікування поки елемент '" + xPath + "' не стане
клікабельним НЕ виконано. " + e.Message);
            }
        }
```

```
        /// <summary>
        /// Wait for element to be visible
        /// </summary>
        /// <param name="xPath">XPath query</param>
        public void WaitForElementIsVisible(string xPath)
        {
            try
            {
                var wait = new WebDriverWait(Driver, TimeSpan.FromSeconds(50));
                wait.Until(ExpectedConditions.ElementIsVisible(By.XPath(xPath)));
                Log.Info("Очікування поки елемент '" + xPath + "' не стане видимим
Виконано.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Очікування поки елемент '" + xPath + "' не стане
видимим НЕ виконано. " + e.Message);
            }
        }


        /// <summary>
        /// Wait, until element to be missing in DOM
        /// </summary>
        /// <param name="xPath">XPath query</param>
        public void WaitForElementToBeStalenessOf(string xPath)
        {
            try
            {
                IWebElement element = Driver.FindElement(By.XPath(xPath));
                var wait = new WebDriverWait(Driver, TimeSpan.FromSeconds(100));
                wait.Until(ExpectedConditions.StalenessOf(element));
                Log.Info("Очікування поки елемент '" + xPath + "' не зникне з DOM
Виконано.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Очікування поки елемент '" + xPath + "' не зникне з
DOM НЕ виконано. " + e.Message);
            }
        }


        /// <summary>
        /// Wait, until element to be missing in DOM
        /// </summary>
        /// <param name="element">Already found web element</param>
        public void WaitForElementToBeStalenessOf_Element(IWebElement element)
        {
            try
            {
                var wait = new WebDriverWait(Driver, TimeSpan.FromSeconds(100));
                wait.Until(ExpectedConditions.StalenessOf(element));
                Log.Info("Очікування поки елемент '" + element + "' не зникне з DOM
Виконано.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Очікування поки елемент '" + element + "' не зникне з
DOM НЕ виконано. " + e.Message);
```

```
            }
        }


        /// <summary>
        /// Wait, until text to be present in element
        /// </summary>
        /// <param name="xPath">XPath query</param>
        /// <param name="text">text in element</param>
        public void WaitForTextToBePresentInElement(string xPath, string text)
        {
            try
            {
                IWebElement element = Driver.FindElement(By.XPath(xPath));
                var wait = new WebDriverWait(Driver, TimeSpan.FromSeconds(60));
                wait.Until(ExpectedConditions.TextToBePresentInElement(element, text));
                Log.Info("Очікування для елемента '" + xPath + "' з текстом '" + text +
"' Виконано.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Очікування для елемента '" + xPath + "' з текстом '"
+ text + "' НЕ виконано. " + e.Message);
            }
        }


        /// <summary>
        /// Wait for Frame to be available and switch to it
        /// </summary>
        /// <param name="xPath">XPath query</param>
        public void WaitForFrameToBeAvailable(string xPath)
        {
            try
            {
                var wait = new WebDriverWait(Driver, TimeSpan.FromSeconds(20));

wait.Until(ExpectedConditions.FrameToBeAvailableAndSwitchToIt(By.XPath(xPath)));
                Log.Info("Перехід до Фрейма '" + xPath + "' Виконаний.");
            }
            catch(Exception e)
            {
                Log.Error("Помилка, Перехід до Фрейма НЕ виконаний. " + e.Message);
            }
        }

        // WAITS END
        // ACTIONS BEGAN

        /// <summary>
        /// Move to element using class Actions
        /// </summary>
        /// <param name="xPath">XPath query</param>
        public void ActionsMoveToElement(string xPath)
        {
            try
            {
                IWebElement element = Driver.FindElement(By.XPath(xPath));
                var actions = new Actions(Driver);
                actions.MoveToElement(element).Build().Perform();
                Log.Info("Перехід до елемента Виконний.");
            }
```

```
            catch (Exception e)
            {
                Log.Error("Помилка, Перехід до елемента НЕ викониний. " + e.Message);
            }
        }


        /// <summary>
        /// Move to element and Click on it using class Actions. Takes String parameter
        /// </summary>
        /// <param name="xPath">XPath query</param>
        public void ActionsMoveToElementAndClick_String(string xPath)
        {
            try
            {
                IWebElement element = Driver.FindElement(By.XPath(xPath));
                var actions = new Actions(Driver);
                actions.MoveToElement(element).Click().Build().Perform();
                Log.Info("Перехід до елемента та клік по ньому Викониний.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Перехід до елемента та клік по ньому НЕ викониний. "
+ e.Message);
            }
        }

        /// <summary>
        /// Move to element and Click on it using class Actions. Takes IWebElement
parameter
        /// </summary>
        /// <param name="desiredElement">XPath query</param>
        public void ActionsMoveToElementAndClick_IWebElement(IWebElement desiredElement)
        {
            try
            {
                var actions = new Actions(Driver);
                actions.MoveToElement(desiredElement).Click().Build().Perform();
                Log.Info("Перехід до елемента та клік по ньому Викониний.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Перехід до елемента та клік по ньому НЕ викониний. "
+ e.Message);
            }
        }


        /// <summary>
        /// Move to element and DoubleClick on it using class Actions
        /// </summary>
        /// <param name="xPath"></param>
        public void ActionsMoveToElementAndDoubleClick(string xPath)
        {
            try
            {
                IWebElement element = Driver.FindElement(By.XPath(xPath));
                var actions = new Actions(Driver);
                actions.MoveToElement(element).DoubleClick().Build().Perform();
                Log.Info("Перехід до елемента та подвійний клік по ньому Викониний.");
            }
            catch (Exception e)
```

```
            {
                Log.Error("Помилка, Перехід до елемента та подвійний клік по ньому НЕ
викониний. " + e.Message);
            }
        }


        /// <summary>
        /// Send keys or text to active element
        /// </summary>
        /// <param name="keysOrText">some text or element</param>
        public void ActiveElement_SendKeysOrText(string keysOrText)
        {
            try
            {
                Driver.SwitchTo().ActiveElement().SendKeys(keysOrText);
                Log.Info("Уведення клавіш або тексту пройшов успішно.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Ввод клавіш або тексту недоступний для елемента. " +
e.Message);
            }
        }

        /// <summary>
        /// Click on active element
        /// </summary>
        public void ActivElement_Click()
        {
            try
            {
                Driver.SwitchTo().ActiveElement().Click();
                Log.Info("Клік по активному елементу пройшов успішно.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Неможливо клікнути по елементу, елемент не доступний.
" + e.Message);
            }
        }

        // ACTIONS END
        // OTHER BEGAN

        public string FindBaseWindow()
        {
            string baseWindow = Driver.CurrentWindowHandle;
            return baseWindow;
        }


        public void SwitchToNewWindow(string baseWindow)
        {
            try
            {
                ReadOnlyCollection<string> handles = Driver.WindowHandles;

                foreach (var handle in handles.Where(handle => handle != baseWindow))
                {
                    Driver.SwitchTo().Window(handle);
                }
```

```
                Log.Info("Перехід до нового вікна Відбувся.");
            }
            catch(Exception e)
            {
                Log.Error("Помилка, Перехід до нового вікна НЕ відбувся. " + e.Message);
            }
        }


        public void SwitchBackToWindow(string baseWindow)
        {
            try
            {
                Driver.Close();
                Driver.SwitchTo().Window(baseWindow);
                Log.Info("Перехід назад до вікна пройшов успішно.");
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Перехід назад до вікна НЕ відбувся. " + e.Message);
            }

        }


        public void RefreshPage()
        {
            try
            {
                Driver.Navigate().Refresh();
                WaitForPageToLoad();
                Log.Info("Сторінка оновилась.");
            }
            catch(Exception e)
            {
                Log.Error("Помилка, Оновлення сторінки не відбулося. " + e.Message);
            }
        }


        /// <summary>
        /// Switch to the default frame
        /// </summary>
        public void SwitchToTheDefaultFrame()
        {
            try
            {
                WaitForPageToLoad();
                Driver.SwitchTo().DefaultContent();
                WaitForPageToLoad();
                Log.Info("Перехід до фрейма за замовчуванням Відбувся.");
            }
            catch(Exception e)
            {
                Log.Error("Помилка, Перехід до фрейма за замовчуванням НЕ відбувся. " +
e.Message);
            }
        }


        /// <summary>
        /// Вибрати відсортований запис в ресурсах користувача
```

```
        /// </summary>
        /// <param name="codeOfSelectedResources">Код вибраного ресурсу</param>
        /// <returns></returns>
        public bool ChooseFoundString(string codeOfSelectedResources)
        {
            try
            {
                FindAllElementsAndClickOnDisplayed("//div[@id='" + Grid +
"']/div[2]/table/tbody/tr/td[text()='" + codeOfSelectedResources + "']");
                var compare =
FindAllElementsAndGetAttributeFromDisplayed(xPath:"//div[@id='" + Grid +
"']/div[2]/table/tbody/tr/td[text()='"
                                                                        +
codeOfSelectedResources + "']/..", attribute: "class");
                //FindElementByXPathAndClick("//div[@id='" + Grid +
"']/div[2]/table/tbody/tr/td[text()='" + codeOfSelectedResources + "']");
                //var compare = FindElementByXPathAndGetAttribute(xPath: "//div[@id='" +
Grid + "']/div[2]/table/tbody/tr/td[text()='" + codeOfSelectedResources + "']/..",
attribute: "class");
                Log.Info(compare);
                Log.Info("Grid is: " + Grid);
                if (compare.Contains("k-state-selected"))
                {
                    Log.Info("Відсортований запис вибраний. Метод 'ChooseFoundString'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка, Відсортований запис НЕ вибраний. Метод
'ChooseFoundString'!");
                    return false;
                }
            }
            catch
            {
                Log.Error("Помилка, Метод відсортувати запис НЕ відпрацював. Метод
'ChooseFoundString'!");
                return false;
            }
        }



        // OTHER END
    }
}
```

MainPage.cs

```
using Common;
using OpenQA.Selenium;
using OpenQA.Selenium.Support.UI;
using NLog;

namespace COBUMMFO_Autotest
{
    public class MainPage : ToolsForTesting
    {
        public const string ButtonShowProfile = "//a[@id='btnProfile']";
        public const string LoginUsUser =
            "//div[@class='user-data']/table/tbody/tr[1]/td/span[@class='bf_bold']";
        public const string ButtonExit = "//button[text()='вихід']";
        //public const string PopupBoxUserProfile = "//div[@id='userProfile']";
```

```
        public const string TopFrame = "//iframe[@id='abs-top-frame']";
        public const string MainFraim = "//iframe[@id='mainFrame']";

        public const string ImageOfLoadingNew = "//div[@class='loading']";
        public const string ImageOfLoadingK = "//div[@class='k-loading-image']";
        //public const string ImageOfLoadingOld = "//img[contains(@src, 'process.gif')]";


        public MainPage(IWebDriver driver) : base(driver)
        {
            Log = LogManager.GetCurrentClassLogger();
        }

        public MainPage(IWebDriver driver, WebDriverWait wait) : base(driver, wait)
        {
        }

        /// <summary>
        /// Open profile menu and get text of user login
        /// </summary>
        /// <returns>text value of user login from prifile menu</returns>
        public string GetCurrentLogin()
        {
            FindElementByXPathAndClick(ButtonShowProfile);
            WaitForPageToLoad();
            return FindElementByXPathAndGetInnerText(LoginUsUser);
        }

        /// <summary>
        /// Compare the current user login with records in profile menu
        /// </summary>
        /// <param name="userLogin">Login of current user</param>
        public bool CheckLoginUsUser(string userLogin)
        {
            try
            {
                if (GetCurrentLogin().Equals(userLogin))
                {
                    Log.Info("Вхід в систему користувачем " + userLogin + " виконаний
ПРАВИЛЬНО.");
                }
                else
                {
                    Log.Info("Вхід в систему користувачем " + userLogin + " виконаний НЕ
правильно. Поточний користувач - " + GetCurrentLogin() + ".");
                }
                Log.Info("Перевірка правильності входу виконана.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Перевірка правильності входу НЕ виконана.");
                return false;
            }
        }

        /// <summary>
        /// Switch to the default frame and exit from account by the profile menu
        /// </summary>
        public bool ExitFromAccount()
        {
            try
```

```
            {
                SwitchToTheDefaultFrame();
                var compare = FindElementByXPathAndGetAttribute(xPath: ButtonShowProfile,
attribute: "class");
                if (compare.EndsWith("active"))
                {
                    FindElementByXPathAndClick(ButtonExit);
                }
                else
                {
                    FindElementByXPathAndClick(ButtonShowProfile);
                    FindElementByXPathAndClick(ButtonExit);
                }
                Log.Info("Вихід з акаунта здійснено.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Вихід з акаунта НЕ здійснено.");
                return false;
            }
        }

        /// <summary>
        /// Find and open ARM by codeapp
        /// </summary>
        /// <param name="codeapp">code of ARM from table "applist"</param>
        public bool OpenARM(string codeapp)
        {
            try
            {
                CheckLoading();
                var armXPath = "//div[@onclick=\"subMenu('" + codeapp + "', this)\"]";
                if (ElementIsDisplayed(armXPath))
                {
                    ActionsMoveToElementAndClick_String(armXPath);
                    Log.Info("АРМ присутній та відкрився.");
                }
                else
                {
                    Log.Info("АРМ відсутній в користувача або введено некоректне значення
codeapp");
                }
                Log.Info("Метод OpenARM відпрацював.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Метод OpenARM НЕ відпрацював.");
                return false;
            }
        }

        /// <summary>
        /// Find and choose function from ARM by codeoper
        /// </summary>
        /// <param name="parentArm">Батьківський АРМ в якому розташована функція</param>
        /// <param name="codeoper">code of operations from table "operlist"</param>
        public bool ChooseFunction(string parentArm, string codeoper)
        {
            try
            {
```

```
                CheckLoading();
                var function = "//div[@id='" + parentArm + "']/div[@data-codeoper='" +
codeoper + "']";
                if (ElementIsDisplayed(function))
                {
                    FindElementByXPathAndClick(function);
                    Log.Info("Функція присутня та клік по ній здійснено.");
                }
                else
                {
                    Log.Info("Помилка, функція відсутня в користувача або введено
некоректне значення codeoper");
                }
                Log.Info("Метод ChooseFunction відпрацював.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Метод ChooseFunction НЕ відпрацював.");
                return false;
            }
        }

        ///// <summary>
        ///// Checking the different logos of loading and wait for their disappearance
        ///// </summary>
        //public void CheckLoading()
        //{
        //    Log.Info("Method 'CheckLoading' Started.");
        //    if (ElementIsDisplayed(imageOfLoadingNew))
        //    {
        //        WaitForElementToBeStalenessOf(imageOfLoadingNew);
        //    }
        //    else
        //    {
        //        Log.Info("Зображення завантаження - imageOfLoadingNew не
з'являється.");
        //    }
        //    if (ElementIsDisplayed(imageOfLoading_k))
        //    {
        //        WaitForElementToBeStalenessOf(imageOfLoading_k);
        //    }
        //    else
        //    {
        //        Log.Info("Зображення завантаження - imageOfLoading_k не з'являється.");
        //    }
        //    Log.Info("Method 'CheckLoading' End.");
        //}



        /// <summary>
        /// Checking the different logos of loading and wait for their disappearance
        /// </summary>
        public void CheckLoading()
        {
            string[] imagesOfLoading = { ImageOfLoadingNew, ImageOfLoadingK };

            foreach (var xPath in imagesOfLoading)
            {
                if (ElementIsDisplayedForCheckLoading(xPath))
                {
```

```
                    try
                    {
                        WaitForElementToBeStalenessOf(xPath);
                        Log.Debug("Очікування елемента '" + xPath + "' виконано. Метод
'CheckLoading'.");

                    }
                    catch
                    {
                        Log.Debug("Помилка. Очікування елемента '" + xPath + "' НЕ
виконано.  Метод 'CheckLoading'.");

                    }
                }
                else
                {
                    Log.Debug("Завантаження '" + xPath + "' відсутні. метод
'CheckLoading'.");
                }
            }
        }
    }
}
```

LoginPage.cs

```
using Common;
using OpenQA.Selenium;
using NLog;

namespace COBUMMFO_Autotest
{
    public class LoginPage : ToolsForTesting
    {
        private const string InputUserLogin = "//input[@id='txtUserName']";
        private const string InputUserPassword = "//input[@id='txtPassword']"; // Regular
entry
        private const string ButtonLogin = "//input[@id='btLogIn']";


        private const string LinkForgotPassword = "//a[@id='ForgotPassword']";
        private const string LinkBackToAutorizationPage = "//input[@id='GotoLogin']";
        private const string InputUserLoginForChange = "//input[@id='userName']";
        private const string InputUserPasswordOld = "//input[@id='passwordOld']"; //
Entry with change password
        private const string InputUserPasswordNew = "//input[@id='password']";
        private const string InputUserPasswordDuplicate =
"//input[@id='passwordConfirmation']";
        private const string ButtonConfirmChanges = "//input[@id='btChangePsw']";

        private const string ButtonContinue = "//input[@id='btChangDate']"; // When user
have permission to change Bank date

        private const string NewMessage = "//div[text()='Нові повідомлення']"; // When
user have a new message


        public LoginPage(IWebDriver driver) : base(driver)
        {
            Log = LogManager.GetCurrentClassLogger();
        }

        /// <summary>
```

```
        /// Regular simple login
        /// </summary>
        /// <param name="userLogin"></param>
        /// <param name="password"></param>
        public bool LoginRegular(string userLogin, string password)
        {
            try
            {
                //WaitForPageToLoad();
                FindElementByXPathAndClearField(InputUserLogin);
                FindElementByXPathAndSendKeys(xPath: InputUserLogin, key: userLogin);
                FindElementByXPathAndSendKeys(xPath: InputUserPassword, key: password);
                FindElementByXPathAndClick(ButtonLogin);
                IfUserHaveAdminArm(userLogin);
                //WaitForPageToLoad();
                Log.Info("Звичайний вхід в систему виконано.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Звичайний вхід в систему НЕ виконано.");
                return false;
            }
        }

        /// <summary>
        /// Login With Change Password
        /// </summary>
        /// <param name="userLogin"></param>
        /// <param name="userPassOld"></param>
        /// <param name="userPassNew"></param>
        public bool LoginWithChangePassword(string userLogin, string userPassOld, string
userPassNew)
        {
            try
            {
                //WaitForPageToLoad();
                FindElementByXPathAndSendKeys(xPath: InputUserLoginForChange, key:
userLogin);
                FindElementByXPathAndSendKeys(xPath: InputUserPasswordOld, key:
userPassOld);
                FindElementByXPathAndSendKeys(xPath: InputUserPasswordNew, key:
userPassNew);
                FindElementByXPathAndSendKeys(xPath: InputUserPasswordDuplicate, key:
userPassNew);
                FindElementByXPathAndClick(ButtonConfirmChanges);
                //IfUserHaveAdminArm(userLogin);
                //WaitForPageToLoad();
                Log.Info("Вхід в систему з заміною пароля виконано.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Вхід в систему з заміною пароля НЕ виконано.");
                return false;
            }
        }

        /// <summary>
        /// Check the continue button and click on it if found
        /// </summary>
        /// <param name="userLogin">only for write text</param>
```

```
        public bool IfUserHaveAdminArm(string userLogin)
        {
            try
            {
                if (ElementIsPresent(ButtonContinue))
                {
                    FindElementByXPathAndClick(ButtonContinue);
                }
                else
                {
                    Log.Info("В користувача " + userLogin + " немає АРМа
Адміністратора.");
                }
                Log.Info("Перевірка наявності АРМа Адміністратора в користувача
пройшла.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Перевірка наявності АРМа Адміністратора в користувача
НЕ пройшла.");
                return false;
            }
        }

        /// <summary>
        /// Перевірка на наявність вікна з повідомленням про натходження користувачу
нових листів
        /// </summary>
        /// <returns></returns>
        public bool CheckNewMessage()
        {
            try
            {
                if (ElementIsPresent(NewMessage))
                {
                    WaitForElementToBeClicable(NewMessage);
                    FindElementByXPathAndClick(NewMessage);
                    Log.Info("У користувача є нові повідомлення.");
                    return true;
                }
                else
                {
                    Log.Info("У користувача немає нових повідомленнь.");
                    return true;
                }
            }
            catch
            {
                Log.Error("Помилка, Перевірка наявності нових повідомлень у користувача
НЕ пройшла.");
                return false;
            }
        }

    }
}
```

ConstructorOfARMs.cs

```
using System;
using System.Threading;
using Common;
```

```
using OpenQA.Selenium;
using NLog;

namespace COBUMMFO_Autotest.Functions
{
    public class ConstructorOfARMs: MainPage
    {
        private const string ButtonFilterOfARMsById =
"//div[@id='ADMGrid']/div[1]/div/table/thead/tr[2]/th[2]/span/span/span[2]";
        private const string ButtonFilterOfARMsByCode =
"//div[@id='ADMGrid']/div[1]/div/table/thead/tr[2]/th[3]/span/span/span[2]";
//Buttons of filters of ARMs
        private const string ButtonFilterOfARMsByName =
"//div[@id='ADMGrid']/div[1]/div/table/thead/tr[2]/th[4]/span/span/span[2]";
        private const string ButtonFilterOfARMsByType =
"//div[@id='ADMGrid']/div[1]/div/table/thead/tr[2]/th[5]/span/span/span[2]";
        public const string InputFilterOfARMsById =
"//div[@id='ADMGrid']/div[1]/div/table/thead/tr[2]/th[2]/span/span/span[1]/span/input[@ti
tle='']";   //<-- так треба
        public const string InputFilterOfARMsByCode = "//input[@data-text-
field='ARM_CODE']";
        public const string InputFilterOfARMsByName = "//input[@data-text-
field='ARM_NAME']";                                                    //Input fields of
filters of ARMs
        public const string InputFilterOfARMsByType = "//input[@data-text-
field='APPLICATION_TYPE']";

        private const string ButtonFilterOfFunctionByCode = "//span[@data-
field='RESOURCE_CODE']/span/span[@title='']";
        private const string ButtonFilterOfFunctionByName = "//span[@data-
field='RESOURCE_NAME']/span/span[@title='']";              //Buttons of filters of
Resources (Functions)
        public const string InputFilterOfFunctionByCode = "//input[@data-text-
field='RESOURCE_CODE']";
        public const string InputFilterOfFunctionByName = "//input[@data-text-
field='RESOURCE_NAME']";                           //Input fields of filters of Resources
(Functions)

        //private const string ButtonChooseQuantityOfRecordsOnARMsPage =
"//div[@id='ADMGrid']/div[3]/span/span/span/span/span[text()='select']";
//Тимчасово не використовується
        private const string ButtonMoveToTheNextPage =
"//div[@id='ADMGrid']/div[3]/a/span[text()='Перейдіть на наступну сторінку']";
        private const string ButtonMoveToThePreviousPage =
"//div[@id='ADMGrid']/div[3]/a/span[text()='Перейти на попередню сторінку']";
        private const string ButtonMoveToTheLastPage =
"//div[@id='ADMGrid']/div[3]/a/span[text()='До останньої сторінки']";
        private const string ButtonMoveToTheFirstPage =
"//div[@id='ADMGrid']/div[3]/a/span[text()='Повернутися на першу сторінку']";
//Navigation on the pages of list of ARMs
        private const string ButtonMorePagesRight7 =
"//div[@id='ADMGrid']/div[3]/ul/li[7]/a[@title='Більше сторінок']";
        private const string ButtonMorePagesRight8 =
"//div[@id='ADMGrid']/div[3]/ul/li[8]/a[@title='Більше сторінок']";
        private const string ButtonMorePagesLeft =
"//div[@id='ADMGrid']/div[3]/ul/li[2]/a[@title='Більше сторінок']";
        private const string ActivePageInARMsList =
"//div[@id='ADMGrid']/div[3]/ul/li/span[@class='k-state-selected']";

        public const string TabFunctionsWeb = "//span[text()='Функції меню (Веб)']";
        public const string TabHandBooks = "//span[text()='Довідники']";          //Tabs
```

```
        public const string ButtonRefreshARMsList =
"//div[@id='ADMGrid']/div[3]/a/span[text()='Оновити']";      //Refresh all ARMs

        public const string ButtonAddARM = "//button[@id='AddADM']";
        public const string ButtonEditARM = "//button[@id='EditADM']";
//ARMs Config ToolBar
        public const string ButtonConfirmInInfoAlertbox =
"//div[@id='barsUiAlertDialog']/../div/button[contains(text(), 'Ok')]";

        public const string AccessOfResourcesAdd =
"//ul[@id='ACCESS_MODE_ID_listbox']/li[text()='Доступний']";
        public const string AccessOfResourcesRemove =
"//ul[@id='ACCESS_MODE_ID_listbox']/li[text()='Не доступний']";

        public const string ButtonSaveCreateARM = "//button[@id='saveCreateAdm-btn']";
        public const string ButtonCancelCreateARM = "//button[@id='cencelCreateAdm-
btn']";
        public const string InputCodeOfARM_CreateARM = "//input[@id='admCode']";
        public const string InputNameOfARM_CreateARM = "//input[@id='admName']";
        public const string ButtonChooseTypeOfARM_CreateARM = "//div[@id='container-
Create']/form/div[3]/span/span/span/span[text()='select']";

        public const string ButtonSaveEditARM = "//button[@id='saveEditAdm-btn']";
        public const string ButtonCancelEditARM = "//button[@id='cencelEditAdm-btn']";
        public const string InputNameOfRole_EditARM = "//input[@id='admNameEdit']";
        public const string ButtonChooseTypeOfARM_EditARM = "//div[@id='container-
Edit']/form/div[3]/span/span/span/span[text()='select']";

        public const string ButtonChangeStatusOfResources = "//span[text()='Змінити
на...']";


        public ConstructorOfARMs(IWebDriver driver) : base(driver)
        {
            Log = LogManager.GetCurrentClassLogger();
        }

        /// <summary>
        /// Приготування до роботи після кліку по функції - чекати завантаження і т.д.
        /// </summary>
        /// <returns></returns>
        public bool Precondition()
        {
            try
            {
                WaitForFrameToBeAvailable(MainFraim);
                CheckLoading();
                Log.Info("Приготування до роботи з функцією виконані. Метод
'Precondition'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Приготування до роботи з функцією НЕ виконані. Метод
'Precondition'.");
                return false;
            }
        }


        //FOR CREATE OR EDIT ARMs BEGUN
```

```
        /// <summary>
        /// Створення нового АРМа
        /// </summary>
        /// <param name="codeARM">Код АРМа</param>
        /// <param name="nameARM">Назва АРМа</param>
        /// <returns></returns>
        public bool CreateARM(string codeARM, string nameARM, string typeOfARM)
        {
            try
            {
                FindElementByXPathAndClick(ButtonAddARM);
                WaitForElementToBeClicable(ButtonSaveCreateARM);
                FindElementByXPathAndClick(InputCodeOfARM_CreateARM);
                FindElementByXPathAndSendKeys(xPath: InputCodeOfARM_CreateARM, key:
codeARM);
                FindElementByXPathAndSendKeys(xPath: InputNameOfARM_CreateARM, key:
nameARM);
                FindElementByXPathAndClick(ButtonChooseTypeOfARM_CreateARM);
                Thread.Sleep(TimeSpan.FromSeconds(1));
                ActionsMoveToElementAndClick_IWebElement(desiredElement:
FindAllElements(xPath: "//ul[@id='admAppType_listbox']/li[text()='" + typeOfARM + "']"));
                FindElementByXPathAndClick(ButtonSaveCreateARM);
                WaitForElementToBeClicable(ButtonConfirmInInfoAlertbox);
                FindElementByXPathAndClick(ButtonConfirmInInfoAlertbox);
                Log.Info("Створення нового АРМа виконано. Метод 'CreateARM'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка. Створення нового АРМа НЕ виконано. Метод
'CreateARM'.");
                return false;
            }
        }

        /// <summary>
        /// Редагування існуючого АРМа
        /// </summary>
        /// <param name="nameARM">Назва АРМа</param>
        /// <returns></returns>
        public bool EditARM(string nameARM, string typeOfARM)
        {
            try
            {
                FindElementByXPathAndClick(ButtonEditARM);
                WaitForElementToBeClicable(InputNameOfRole_EditARM);
                FindElementByXPathAndClick(InputNameOfRole_EditARM);
                FindElementByXPathAndClearField(InputNameOfRole_EditARM);
                FindElementByXPathAndSendKeys(xPath: InputNameOfRole_EditARM, key:
nameARM);
                WaitForElementToBeClicable(ButtonChooseTypeOfARM_EditARM);
                FindElementByXPathAndClick(ButtonChooseTypeOfARM_EditARM);
                Thread.Sleep(TimeSpan.FromSeconds(1));
                ActionsMoveToElementAndClick_IWebElement(desiredElement:
FindAllElements(xPath: "//ul[@id='admAppTypeEdit_listbox']/li[text()='" + typeOfARM +
"']"));
                WaitForElementToBeClicable(ButtonSaveEditARM);
                FindElementByXPathAndClick(ButtonSaveEditARM);
                WaitForElementToBeClicable(ButtonConfirmInInfoAlertbox);
                FindElementByXPathAndClick(ButtonConfirmInInfoAlertbox);
                Log.Info("Редагування АРМа виконано. Метод 'EditARM'.");
```

```
                return true;
            }
            catch
            {
                Log.Error("Помилка. Редагування АРМа НЕ виконано. Метод 'EditARM'.");
                return false;
            }
        }


        //FOR CREATE OR EDIT ARMs END
        //FOR ADD OR REMOVE RESOURCES TO ARMs BEGUN


        /// <summary>
        /// Розкриває АРМ в списку АРМів
        /// </summary>
        /// <param name="armId">Id АРМа</param>
        /// <returns></returns>
        public bool OpenResourcesOfARM(string armId)
        {
            try
            {
                FindElementByXPathAndClick("//td[text()='" + armId + "']/../td/a");
// Припускається, що при виконанні тесту буде 'розкрито' лише один АРМ
                WaitForElementToBeClicable(TabFunctionsWeb);
// бо XPath вкладки типу 'Функції меню (Веб)' не унікальний...
                Log.Info("Розкриття АРМа '" + armId + "' виконано. Метод
'OpenResourcesOfARM'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка. Розкриття АРМа '" + armId + "' НЕ виконано. Метод
'OpenResourcesOfARM'.");
                return false;
            }
        }

        /// <summary>
        /// Перевірка чи активна вкладка, та її відкриття в разі  якщо - неактивна
        /// </summary>
        /// <param name="tabXPath">XPath-запит береться з класу
'ConstructorOfARMs'</param>
        /// <returns></returns>
        public bool MakeActiveTab(string tabXPath)
        {
            try
            {
                var compare = FindElementByXPathAndGetAttribute(xPath: tabXPath + "/..",
attribute: "class");
                if (compare.Contains("k-state-active"))
                {
                    Log.Info("Вкладка зараз активна. Метод 'MakeActiveTab'.");
                    return true;
                }
                else
                {
                    FindElementByXPathAndClick(tabXPath);
                    var compareAgain = FindElementByXPathAndGetAttribute(xPath: tabXPath
+ "/..", attribute: "class");
                    if (compareAgain.Contains("k-state-active"))
```

```
                    {
                        if (tabXPath.Equals(TabFunctionsWeb))
                        {
                            Grid = "AdmResourceFUNCTION_WEB";
                        }
                        if (tabXPath.Equals(TabHandBooks))
                        {
                            Grid = "AdmResourceDIRECTORIES";
                        }
                        Log.Info("Grid is : " + Grid);
                        Log.Info("Вкладка стала активна після кліку по ній. Метод
'MakeActiveTab'.");
                        return true;
                    }
                    else
                    {
                        Log.Error("Помилка! Вкладка НЕ активна. Метод 'MakeActiveTab'.");
                        Log.Error("Значення атрибуту 'class' шуканої вкладки: " +
compareAgain);
                        return false;
                    }
                }
            }
            catch
            {
                Log.Error("Помилка! Метод 'MakeActiveTab' НЕ відпрацював.");
                return false;
            }
        }

        /// <summary>
        /// Дозволити доступ до ресурса (функції)
        /// </summary>
        /// <param name="codeOfResources">Код ресурсу (функції)</param>
        /// <returns></returns>
        public bool ChangeAccessAdd(string codeOfResources)
        {
            try
            {
                Thread.Sleep(TimeSpan.FromSeconds(2));
                FindElementByXPathAndClick("//td[text()='" + codeOfResources +
"']/../td[text()='Не доступний']");
                WaitForElementToBeClicable(ButtonChangeStatusOfResources);
                FindElementByXPathAndClick(ButtonChangeStatusOfResources);
                Thread.Sleep(TimeSpan.FromSeconds(2));
                ActionsMoveToElementAndClick_String(AccessOfResourcesAdd);
                WaitForElementToBeClicable(ButtonConfirmInInfoAlertbox);
                FindElementByXPathAndClick(ButtonConfirmInInfoAlertbox);
                if (ElementIsDisplayed("//td[text()='" + codeOfResources +
"']/../td[last()][text()='0']"))
                {
                    Log.Info("Доступ до ресурса дозволено. Метод 'ChangeAccessAdd'.");
                    return true;
                }
                Log.Error("Помилка! Доступ до ресурса НЕ змінено. Метод
'ChangeAccessAdd'.");
                return false;
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод 'ChangeAccessAdd' НЕ відпрацював. Стек: " +
e.Message);
```

```
                return false;
            }
        }

        /// <summary>
        /// Заборонити доступ до ресурса (функції)
        /// </summary>
        /// <param name="codeOfResources">Код ресурсу (функції)</param>
        /// <returns></returns>
        public bool ChangeAccessRemove(string codeOfResources)
        {
            try
            {
                Thread.Sleep(TimeSpan.FromSeconds(2));
                FindElementByXPathAndClick("//td[text()='" + codeOfResources +
"']/../td[text()='Доступний']");
                WaitForElementToBeClicable(ButtonChangeStatusOfResources);
                FindElementByXPathAndClick(ButtonChangeStatusOfResources);
                Thread.Sleep(TimeSpan.FromSeconds(2));
                ActionsMoveToElementAndClick_String(AccessOfResourcesRemove);
                WaitForElementToBeClicable(ButtonConfirmInInfoAlertbox);
                FindElementByXPathAndClick(ButtonConfirmInInfoAlertbox);
                if (ElementIsDisplayed("//td[text()='" + codeOfResources +
"']/../td[last()][text()='0']"))
                {
                    Log.Info("Доступ до ресурса заборонено. Метод 'ChangeAccessRemove'.
Метод 'ChangeAccessRemove'.");
                    return true;
                }
                Log.Error("Помилка! Доступ до ресурса НЕ змінено. Метод
'ChangeAccessRemove'. Метод 'ChangeAccessRemove'.");
                return false;
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод 'ChangeAccessRemove' НЕ відпрацював. Стек: " +
e.Message);
                return false;
            }
        }


        //FOR ADD OR REMOVE RESOURCES TO ARMs END
        //SELECT ARM BEGAN


        /// <summary>
        /// Пошук АРМа в списку за Id
        /// </summary>
        /// <param name="armId">Id ролі</param>
        /// <param name="enterValueOfFilter">Значення фільтра</param>
        public bool SelectARMById(string armId, string enterValueOfFilter)
        {
            try
            {
                FindElementByXPathAndClick(ButtonFilterOfARMsById);
                Thread.Sleep(TimeSpan.FromSeconds(2));
                ActionsMoveToElementAndClick_IWebElement(desiredElement: FindAllElements(
                        xPath: Filter.SelectTypeFilter(enterValueOfFilter:
enterValueOfFilter)));        // <-- (тільки для цього рядка) Метод
'ActionsMoveToElementAndClick_IWebElement'
```

```
                ActivElement_Click();
// приймає вихідне IWebElement значення метода 'FindAllElements',
                FindElementByXPathAndSendKeys(xPath: InputFilterOfARMsById, key: armId);
// який в свою чергу приймає вихідне стрінгове значення метода 'Filter.SelectTypeFilter'
                ActiveElement_SendKeysOrText(Keys.Enter);
                CheckLoading();
                Grid = "ADMGrid";
                Log.Info("Пошук АРМа в списку за Id відбувся. Метод 'SelectARMById'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка, Пошук АРМа в списку за Id НЕ відбувся. Метод
'SelectARMById'.");
                return false;
            }
        }

        /// <summary>
        /// Пошук АРМа в списку за кодом
        /// </summary>
        /// <param name="armCode">Код ролі</param>
        /// <param name="enterValueOfFilter">Значення фільтра</param>
        public bool SelectARMByCode(string armCode, string enterValueOfFilter)
        {
            try
            {
                FindElementByXPathAndClick(ButtonFilterOfARMsByCode);
                Thread.Sleep(TimeSpan.FromSeconds(1));
                ActionsMoveToElementAndClick_IWebElement(desiredElement:
FindAllElements(xPath: Filter.SelectTypeFilter(enterValueOfFilter: enterValueOfFilter)));
                ActivElement_Click();
                FindElementByXPathAndSendKeys(xPath: InputFilterOfARMsByCode, key:
armCode);
                ActiveElement_SendKeysOrText(Keys.Enter);
                CheckLoading();
                Grid = "ADMGrid";
                Log.Info("Пошук АРМа в списку за кодом відбувся. Метод
'SelectARMByCode'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка! Пошук АРМа в списку за кодом НЕ відбувся! Метод
'SelectARMByCode'.");
                return false;
            }

        }

        /// <summary>
        /// Пошук АРМа в списку за назвою
        /// </summary>
        /// <param name="armName">Назва ролі</param>
        /// <param name="enterValueOfFilter">Значення фільтра</param>
        public bool SelectARMByName(string armName, string enterValueOfFilter)
        {
            try
            {
                FindElementByXPathAndClick(ButtonFilterOfARMsByName);
                Thread.Sleep(TimeSpan.FromSeconds(1));
```

```
                ActionsMoveToElementAndClick_IWebElement(desiredElement:
FindAllElements(xPath: Filter.SelectTypeFilter(enterValueOfFilter: enterValueOfFilter)));
                ActivElement_Click();
                FindElementByXPathAndSendKeys(xPath: InputFilterOfARMsByName, key:
armName);
                ActiveElement_SendKeysOrText(Keys.Enter);
                CheckLoading();
                Grid = "ADMGrid";
                Log.Info("Пошук АРМа в списку за назвою відбувся. Метод
'SelectARMByName'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка! Пошук АРМа в списку за назвою НЕ відбувся! Метод
'SelectARMByName'.");
                return false;
            }
        }

        /// <summary>
        /// Пошук АРМа в списку за типом
        /// </summary>
        /// <param name="armType">Стан ролі</param>
        /// <param name="enterValueOfFilter">Значення фільтра</param>
        public bool SelectARMByType(string armType, string enterValueOfFilter)
        {
            try
            {
                FindElementByXPathAndClick(ButtonFilterOfARMsByType);
                Thread.Sleep(TimeSpan.FromSeconds(1));
                ActionsMoveToElementAndClick_IWebElement(desiredElement:
FindAllElements(xPath: Filter.SelectTypeFilter(enterValueOfFilter: enterValueOfFilter)));
                ActivElement_Click();
                FindElementByXPathAndSendKeys(xPath: InputFilterOfARMsByType, key:
armType);
                ActiveElement_SendKeysOrText(Keys.Enter);
                CheckLoading();
                Grid = "ADMGrid";
                Log.Info("Пошук АРМа в списку за типом відбувся. Метод
'SelectARMByType'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка! Пошук АРМа в списку за типом НЕ відбувся! Метод
'SelectARMByType'.");
                return false;
            }

        }


        //SELECT ARM END
        //SELECT RESOURCES (FUNCTIONS) BEGAN


        /// <summary>
        /// Пошук Функції (ресурсу) в списку за назвою
        /// </summary>
        /// <param name="enterValueOfFilter">Вид фільтра</param>
        /// <param name="nameOfFunction">Назва Функції</param>
```

```
        /// <param name="idOfARM">Id АРМа, в якому переглядаємо Функцію</param>
        /// <returns></returns>
        public bool SelectFunctionByName(string nameOfFunction, string
enterValueOfFilter, string idOfARM)
        {
            try
            {
                WaitForElementToBeClicable(ButtonFilterOfFunctionByName);
                FindElementByXPathAndClick(ButtonFilterOfFunctionByName);
                Thread.Sleep(TimeSpan.FromSeconds(1));
                ActionsMoveToElementAndClick_IWebElement(desiredElement:
FindAllElements(xPath: Filter.SelectTypeFilter(enterValueOfFilter: enterValueOfFilter)));
                ActivElement_Click();
                FindElementByXPathAndSendKeys(xPath: InputFilterOfFunctionByName, key:
nameOfFunction);
                ActiveElement_SendKeysOrText(Keys.Tab);
                CheckLoading();
                if (Grid.EndsWith(idOfARM))
                {
                    Log.Info("Grid вже і так актуальний, значення: " + Grid + ". Метод
'SelectFunctionByName'.");
                }
                else
                {
                    Grid = Grid + idOfARM;
                    Log.Info("Grid не був актуальний, але зараз Ок, значення: " + Grid +
". Метод 'SelectFunctionByName'.");
                }
                Log.Info("Пошук Функції в списку за назвою, відбувся. Метод
'SelectARMByName'.");
                return true;
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Пошук Функції в списку за назвою, НЕ відбувся. Метод
'SelectFunctionByName'. Стек: " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Пошук Функції (ресурсу) в списку за кодом
        /// </summary>
        /// <param name="enterValueOfFilter">Вид фільтра</param>
        /// <param name="codeOfFunction">Код Функції</param>
        /// <param name="idOfARM">Id АРМа, в якому переглядаємо Функцію</param>
        /// <returns></returns>
        public bool SelectFunctionByCode(string codeOfFunction, string
enterValueOfFilter, string idOfARM)
        {
            try
            {
                WaitForElementToBeClicable(ButtonFilterOfFunctionByCode);
                FindElementByXPathAndClick(ButtonFilterOfFunctionByCode);
                Thread.Sleep(TimeSpan.FromSeconds(1));
                ActionsMoveToElementAndClick_IWebElement(desiredElement:
FindAllElements(xPath: Filter.SelectTypeFilter(enterValueOfFilter: enterValueOfFilter)));
                ActivElement_Click();
                FindElementByXPathAndSendKeys(xPath: InputFilterOfFunctionByCode, key:
codeOfFunction);
                ActiveElement_SendKeysOrText(Keys.Tab);
                CheckLoading();
```

```
                if (Grid.EndsWith(idOfARM))
                {
                    Log.Info("Grid вже і так актуальний, значення: " + Grid + ". Метод
'SelectFunctionByCode'.");
                }
                else
                {
                    Grid = Grid + idOfARM;
                    Log.Info("Grid не був актуальний, але зараз Ок, значення: " + Grid +
". Метод 'SelectFunctionByCode'.");
                }
                Log.Info("Пошук Функції в списку за кодом, відбувся. Метод
'SelectFunctionByCode'.");
                return true;
            }
            catch (Exception e)
            {
                Log.Error("Помилка, Пошук Функції в списку за кодом, НЕ відбувся. Метод
'SelectFunctionByCode'. Стек: " + e.Message);
                return false;
            }
        }


        //SELECT RESOURCES (FUNCTIONS) END
        //NAVIGATIONS ON THE PAGES OF ARMs BEGAN


        ///// <summary>
        ///// Вибирає задану кількість рядків для відображення на сторінці АРМів
        ///// </summary>
        ///// <param name="setQuantityOfRecords">Кількість рядків на сторінці</param>
        ///// <returns></returns>
        //public bool SetQuantityOfRecordsOnARMsPage(string setQuantityOfRecords)
        //{
//Вибір кількості рядків для відображення на сторінці АРМів,
        //    try
//покищо відсутній але напевно буде.
        //    {
        //        var xPathOfQuantity = "//html[@id='ng-
app']/body/div/div/div/ul/li[text()='" + setQuantityOfRecords + "']";
        //        FindElementByXPathAndClick(ButtonChooseQuantityOfRecordsOnARMsPage);
        //        Thread.Sleep(TimeSpan.FromSeconds(1));
        //        ActionsMoveToElementAndClick_String(xPath: xPathOfQuantity);
        //        Log.Info("Вибір кількості рядків для відображення на сторінці відбувся.
Метод 'SetQuantityOfRecordsOnARMsPage'.");
        //        return true;
        //    }
        //    catch
        //    {
        //        Log.Error("Помилка! Вибір кількості рядків для відображення на сторінці
НЕ відбувся! Метод 'SetQuantityOfRecordsOnARMsPage'.");
        //        return false;
        //    }
        //}

        /// <summary>
        /// Клік по кнопці Більше сторінок справа в списку ролей
        /// </summary>
        public bool ClickMorePageRightInARMsList()
        {
            try
```

```
            {
                if (ElementIsPresent(ButtonMorePagesRight8))
                {
                    var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                    FindElementByXPathAndClick(ButtonMorePagesRight8);
                    CheckLoading();
                    var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                    if (valueOfActivePageLast.Equals(valueOfActivePageFirst + 5))
                    {
                        Log.Info("Метод Клік по кнопці Більше сторінок справа спрацював.
Метод 'ClickMorePageRightInARMsList'.");
                        return true;
                    }
                    else
                    {
                        Log.Error("Помилка! Клік по кнопці Більше сторінок справа НЕ
спрацював! Метод 'ClickMorePageRightInARMsList'.");
                        return false;
                    }
                }
                if (ElementIsPresent(ButtonMorePagesRight7))
                {
                    var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                    Log.Info("VALUE OF valueOfActivePageFirst IS: " +
valueOfActivePageFirst);
                    FindElementByXPathAndClick(ButtonMorePagesRight7);
                    CheckLoading();
                    var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                    Log.Info("VALUE OF valueOfActivePageLast IS: " +
valueOfActivePageLast + valueOfActivePageLast + 1);
                    if (valueOfActivePageLast.Equals(valueOfActivePageFirst + 5))
                    {
                        Log.Info("Метод Клік по кнопці Більше сторінок справа спрацював.
Метод 'ClickMorePageRightInARMsList'.");
                        return true;
                    }
                    else
                    {
                        Log.Error("Помилка! Клік по кнопці Більше сторінок справа НЕ
спрацював! Метод 'ClickMorePageRightInARMsList'.");
                        return false;
                    }
                }
                else
                {
                    Log.Error("Помилка! Кнопки Більше сторінок справа НЕ знайдено! Метод
'ClickMorePageRightInARMsList'.");
                    return false;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Клік по кнопці Більше сторінок справа НЕ відбувся!
Метод 'ClickMorePageRightInARMsList'. Стек: " + e.Message);
                return false;
            }
        }
```

```
        /// <summary>
        /// Клік по кнопці Більше сторінок зліва в списку ролей
        /// </summary>
        public bool ClickMorePageLeftInARMsList()
        {
            try
            {
                if (ElementIsPresent(ButtonMorePagesLeft))
                {
                    var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                    FindElementByXPathAndClick(ButtonMorePagesLeft);
                    CheckLoading();
                    var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                    if (valueOfActivePageLast.Equals(valueOfActivePageFirst - 1))
                    {
                        Log.Info("Метод Клік по кнопці Більше сторінок зліва спрацював.
Метод 'ClickMorePageLeftInARMsList'.");
                        return true;
                    }
                    else
                    {
                        Log.Error("Помилка! Клік по кнопці Більше сторінок зліва НЕ
спрацював! Метод 'ClickMorePageLeftInARMsList'.");
                        return false;
                    }
                }
                else
                {
                    Log.Info("Ти знаходися на початку відліку сторінок (немає куди
рухатись вліво)! Метод 'ClickMorePageLeftInARMsList'.");
                    return true;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод Клік по кнопці Більше сторінок зліва НЕ
спрацював! Метод 'ClickMorePageLeftInARMsList'. Стек: " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Переходить на вказану сторінку в списку АРМів
        /// </summary>
        /// <param name="numberOfPage">Номер сторінки</param>
        public bool SwitchToThePageInARMsList(string numberOfPage)
        {
            try
            {
                WaitForElementIsVisible(ActivePageInARMsList);
                var buttonMoveToTheDefinedPage =
"//div[@id='ADMGrid']/div[3]/ul/li/a[text()='" + numberOfPage + "']";
                if (ElementIsDisplayed(buttonMoveToTheDefinedPage))
                {
                    FindElementByXPathAndClick(buttonMoveToTheDefinedPage);
                    CheckLoading();
                }
                else
                {
```

```
                    Log.Info("Номер сторінки відсутній або його не видно! Метод
'SwitchToThePageInARMsList'.");
                }
                Log.Info("Метод Перейти на вказану сторінку в списку АРМів спрацював.
Метод 'SwitchToThePageInARMsList'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка! Метод 'SwitchToThePageInARMsList' не спрацював.");
                return false;
            }
        }

        /// <summary>
        /// Переходить на наступну сторінку в списку АРМів
        /// </summary>
        public bool ClickMoveToTheNextPageInARMsList()
        {
            try
            {
                WaitForElementIsVisible(ActivePageInARMsList);
                var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                FindElementByXPathAndClick(ButtonMoveToTheNextPage);
                CheckLoading();
                var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                if (valueOfActivePageLast.Equals(valueOfActivePageFirst + 1))
                {
                    Log.Info("Перехід на наступну сторінку в списку АРМів відбувся. Метод
'ClickMoveToTheNextPageInARMsList'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на наступну сторінку в списку АРМів НЕ
відбувся. Метод 'ClickMoveToTheNextPageInARMsList'.");
                    return false;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод 'ClickMoveToTheNextPageInARMsList' не
відпрацював. Стек: " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Переходить на попередню сторінку в списку АРМів
        /// </summary>
        public bool ClickMoveToThePreviousPageInARMsList()
        {
            try
            {
                WaitForElementIsVisible(ActivePageInARMsList);
                var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                FindElementByXPathAndClick(ButtonMoveToThePreviousPage);
                CheckLoading();
```

```
                var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                if (valueOfActivePageLast.Equals(valueOfActivePageFirst - 1) ||
valueOfActivePageFirst.Equals(1))
                {
                    Log.Info("Перехід на попередню сторінку в списку АРМів відбувся.
Метод 'ClickMoveToThePreviousPageInARMsList'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на попередню сторінку в списку АРМів НЕ
відбувся. Метод 'ClickMoveToThePreviousPageInARMsList'.");
                    return false;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод 'ClickMoveToThePreviousPageInARMsList' не
відпрацював. Стек: " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Переходить на останню сторінку в списку АРМів
        /// </summary>
        public bool ClickMoveToTheLastPageInARMsList()
        {
            try
            {
                WaitForElementToBeClicable(ButtonMoveToTheLastPage);
                FindElementByXPathAndClick(ButtonMoveToTheLastPage);
                CheckLoading();
                if (ElementIsDisplayed(ButtonMorePagesRight7) == false &&
ElementIsDisplayed(ButtonMorePagesRight8) == false)
                {
                    Log.Info("Перехід на останню сторінку в списку АРМів відбувся. Метод
'ClickMoveToTheLastPageInARMsList'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на останню сторінку в списку АРМів НЕ
відбувся. Метод 'ClickMoveToTheLastPageInARMsList'.");
                    return false;
                }
            }
            catch
            {
                Log.Error("Помилка! Метод 'ClickMoveToTheLastPageInARMsList' не
відпрацював.");
                return false;
            }
        }

        /// <summary>
        /// Переходить на першу сторінку в списку АРМів
        /// </summary>
        public bool ClickMoveToTheFirstPageInARMsList()
        {
            try
```

```
            {
                WaitForElementToBeClicable(ButtonMoveToTheFirstPage);
                FindElementByXPathAndClick(ButtonMoveToTheFirstPage);
                CheckLoading();
                const string numberOfFirstPage = "1";
                var numberOfActivePage =
FindElementByXPathAndGetInnerText(ActivePageInARMsList);
                if (numberOfFirstPage.Equals(numberOfActivePage))
                {
                    Log.Info("Перехід на першу сторінку в списку АРМів відбувся. Метод
'ClickMoveToTheFirstPageInARMsList'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на першу сторінку в списку АРМів НЕ
відбувся. Метод 'ClickMoveToTheFirstPageInARMsList'.");
                    return false;
                }
            }
            catch
            {
                Log.Error("Помилка! Метод 'ClickMoveToTheFirstPageInARMsList' не
відпрацював.");
                return false;
            }
        }

        /// <summary>
        /// Оновлення в блоці АРМів
        /// </summary>
        /// <returns></returns>
        public bool RefreshARMs()
        {
            try
            {
                FindElementByXPathAndClick(ButtonRefreshARMsList);
                CheckLoading();
                Log.Info("Клік по кнопці оновлення в блоці АРМів виконано. Метод
'RefreshARMs'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка. Клік по кнопці оновлення в блоці АРМів НЕ виконано.
Метод 'RefreshARMs'.");
                return false;
            }
        }

        //NAVIGATIONS ON THE PAGES OF ARMs END
        //NAVIGATIONS ON THE PAGES OF RESOURCES (FUNCTIONS) BEGAN

        /// <summary>
        /// Вибирає задану кількість рядків для відображення на сторінці ресурсів
(Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
        /// <param name="setQuantityOfRecords">Кількість рядків на сторінці</param>
        /// <returns></returns>
        public bool SetQuantityOfRecordsOnFunctionsPage(string idOfARM, string
setQuantityOfRecords)
```

```
        {
            try
            {
                var xPathOfQuantity = "//html[@id='ng-
app']/body/div/div/ul/li[text()='" + setQuantityOfRecords + "']";
                FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/span/span/span/span/span[text()='select']");
                Thread.Sleep(TimeSpan.FromSeconds(1));
                ActionsMoveToElementAndClick_String(xPath: xPathOfQuantity);
                CheckLoading();
                Log.Info("Вибір кількості рядків для відображення на сторінці (ресурси)
відбувся. Метод 'SetQuantityOfRecordsOnFunctionsPage'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка! Вибір кількості рядків для відображення на сторінці
(ресурси) НЕ відбувся! Метод 'SetQuantityOfRecordsOnFunctionsPage'.");
                return false;
            }
        }

        /// <summary>
        /// Клік по кнопці Більше сторінок справа в списку ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
        public bool ClickMorePageRightInFunctionsList(string idOfARM)
        {
            try
            {
                if (ElementIsPresent("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM +
"']/div[3]/ul/li[8]/a[@title='Більше сторінок']"))   //
                {
                    var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
//
                        idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                    FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li[8]/a[@title='Більше сторінок']");
                    CheckLoading();
                    var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
//
                        idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                    if (valueOfActivePageLast.Equals(valueOfActivePageFirst + 5))
                    {
                        Log.Info("Метод Клік по кнопці Більше сторінок справа (ресурси)
спрацював. Метод 'ClickMorePageRightInFunctionsList'.");
                        return true;
                    }
                    else
                    {
                        Log.Error("Помилка! Клік по кнопці Більше сторінок справа
(ресурси) НЕ спрацював! Метод 'ClickMorePageRightInFunctionsList'.");
                        return false;
                    }
                }
                if (ElementIsPresent("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM +
"']/div[3]/ul/li[7]/a[@title='Більше сторінок']"))
                {
                    var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
```

```
                    Log.Info("VALUE OF valueOfActivePageFirst IS: " +
valueOfActivePageFirst);
                    FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li[7]/a[@title='Більше сторінок']");
                    CheckLoading();
                    var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText(ActivePageInARMsList));
                    Log.Info("VALUE OF valueOfActivePageLast IS: " +
valueOfActivePageLast + valueOfActivePageLast + 1);
                    if (valueOfActivePageLast.Equals(valueOfActivePageFirst + 5))
                    {
                        Log.Info("Метод Клік по кнопці Більше сторінок справа (ресурси)
спрацював. Метод 'ClickMorePageRightInFunctionsList'.");
                        return true;
                    }
                    else
                    {
                        Log.Error("Помилка! Клік по кнопці Більше сторінок справа
(ресурси) НЕ спрацював! Метод 'ClickMorePageRightInFunctionsList'.");
                        return false;
                    }
                }
                else
                {
                    Log.Error("Помилка! Кнопки Більше сторінок справа (ресурси) НЕ
знайдено! Метод 'ClickMorePageRightInFunctionsList'.");
                    return false;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Клік по кнопці Більше сторінок справа (ресурси) НЕ
відбувся! Метод 'ClickMorePageRightInFunctionsList'. Стек : " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Клік по кнопці Більше сторінок зліва в списку ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
        public bool ClickMorePageLeftInFunctionsList(string idOfARM)
        {
            try
            {
                if (ElementIsPresent("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM +
"']/div[3]/ul/li[2]/a[@title='Більше сторінок']"))
                {
                    var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                    FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li[2]/a[@title='Більше сторінок']");
                    CheckLoading();
                    var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                    if (valueOfActivePageLast.Equals(valueOfActivePageFirst - 1))
                    {
                        Log.Info("Метод Клік по кнопці Більше сторінок зліва (ресурси)
спрацював. Метод 'ClickMorePageLeftInFunctionsList'.");
                        return true;
```

```
                    }
                    else
                    {
                        Log.Error("Помилка! Клік по кнопці Більше сторінок зліва
(ресурси) НЕ спрацював! Метод 'ClickMorePageLeftInFunctionsList'.");
                        return false;
                    }
                }
                else
                {
                    Log.Info("Ти знаходися на початку відліку сторінок (ресурси) (немає
куди рухатись вліво)! Метод 'ClickMorePageLeftInFunctionsList'.");
                    return true;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод 'ClickMorePageLeftInFunctionsList' НЕ
спрацював! Стек : " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Переходить на вказану сторінку в списку ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
        /// <param name="numberOfPage">Номер сторінки</param>
        public bool SwitchToThePageInFunctionsList(string idOfARM, string numberOfPage)
        {
            try
            {
                WaitForElementToBeClicable("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/ul/li/span[@class='k-state-selected']");
                //Thread.Sleep(TimeSpan.FromSeconds(2));
                var buttonMoveToTheDefinedPage = "//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li/a[text()='" + numberOfPage + "']";
                if (ElementIsDisplayed(buttonMoveToTheDefinedPage))
                {
                    FindElementByXPathAndClick(buttonMoveToTheDefinedPage);
                    CheckLoading();
                }
                else
                {
                    Log.Info("Номер сторінки відсутній або його не видно! (ресурси).
Метод 'SwitchToThePageInFunctionsList'.");
                }
                Log.Info("Метод Перейти на вказану сторінку в списку ресурсів спрацював.
Метод 'SwitchToThePageInFunctionsList'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка! Метод 'SwitchToThePageInARMsList' не спрацював.");
                return false;
            }
        }

        /// <summary>
        /// Переходить на наступну сторінку в списку ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
```

```
        public bool ClickMoveToTheNextPageInFunctionsList(string idOfARM)
        {
            try
            {
                WaitForElementToBeClicable("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/ul/li/span[@class='k-state-selected']");
                //Thread.Sleep(TimeSpan.FromSeconds(2));
                var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/a/span[text()='Перейдіть на наступну сторінку']");
                CheckLoading();
                var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                if (valueOfActivePageLast.Equals(valueOfActivePageFirst + 1))
                {
                    Log.Info("Перехід на наступну сторінку в списку ресурсів відбувся.
Метод 'ClickMoveToTheNextPageInFunctionsList'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на наступну сторінку в списку ресурсів НЕ
відбувся. Метод 'ClickMoveToTheNextPageInFunctionsList'.");
                    return false;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод 'ClickMoveToTheNextPageInFunctionsList' не
відпрацював. Стек : " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Переходить на попередню сторінку в списку ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
        public bool ClickMoveToThePreviousPageInFunctionsList(string idOfARM)
        {
            try
            {
                WaitForElementToBeClicable("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/ul/li/span[@class='k-state-selected']");
                //Thread.Sleep(TimeSpan.FromSeconds(2));
                var valueOfActivePageFirst =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/a/span[text()='Перейти на попередню сторінку']");
                CheckLoading();
                var valueOfActivePageLast =
int.Parse(FindElementByXPathAndGetInnerText("//div[@id='AdmResourceFUNCTION_WEB" +
idOfARM + "']/div[3]/ul/li/span[@class='k-state-selected']"));
                if (valueOfActivePageLast.Equals(valueOfActivePageFirst - 1) ||
valueOfActivePageFirst.Equals(1))
                {
                    Log.Info("Перехід на попередню сторінку в списку ресурсів відбувся.
Метод 'ClickMoveToThePreviousPageInFunctionsList'.");
```

```
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на попередню сторінку в списку ресурсів
НЕ відбувся. Метод 'ClickMoveToThePreviousPageInFunctionsList'.");
                    return false;
                }
            }
            catch (Exception e)
            {
                Log.Error("Помилка! Метод 'ClickMoveToThePreviousPageInFunctionsList' не
відпрацював. Стек : " + e.Message);
                return false;
            }
        }

        /// <summary>
        /// Переходить на останню сторінку в списку ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
        public bool ClickMoveToTheLastPageInFunctionsList(string idOfARM)
        {
            try
            {
                WaitForElementToBeClicable("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/a/span[text()='До останньої сторінки']");
                //Thread.Sleep(TimeSpan.FromSeconds(2));
                FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/a/span[text()='До останньої сторінки']");
                CheckLoading();
                if (ElementIsDisplayed("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM +
"']/div[3]/ul/li[7]/a[@title='Більше сторінок']") == false
                    && ElementIsDisplayed("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/ul/li[8]/a[@title='Більше сторінок']") == false)
                {
                    Log.Info("Перехід на останню сторінку в списку ресурсів відбувся.
Метод 'ClickMoveToTheLastPageInFunctionsList'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на останню сторінку в списку ресурсів НЕ
відбувся. Метод 'ClickMoveToTheLastPageInFunctionsList'.");
                    return false;
                }
            }
            catch
            {
                Log.Error("Помилка! Метод 'ClickMoveToTheLastPageInFunctionsList' не
відпрацював.");
                return false;
            }
        }

        /// <summary>
        /// Переходить на першу сторінку в списку ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id АРМа в якому знаходимось</param>
        public bool ClickMoveToTheFirstPageInFunctionsList(string idOfARM)
        {
            try
```

```
            {
                WaitForElementToBeClicable("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/a/span[text()='Повернутися на першу сторінку']");
                //Thread.Sleep(TimeSpan.FromSeconds(2));
                FindElementByXPathAndClick("//div[@id='AdmResourceFUNCTION_WEB" + idOfARM
+ "']/div[3]/a/span[text()='Повернутися на першу сторінку']");
                CheckLoading();
                const string numberOfFirstPage = "1";
                var numberOfActivePage =
FindElementByXPathAndGetInnerText("//div[@id='RoleResourceARM_WEB" + idOfARM +
"']/div[3]/ul/li/span[@class='k-state-selected']");
                if (numberOfFirstPage.Equals(numberOfActivePage))
                {
                    Log.Info("Перехід на першу сторінку в списку ресурсів відбувся. Метод
'ClickMoveToTheFirstPageInFunctionsList'.");
                    return true;
                }
                else
                {
                    Log.Error("Помилка! Перехід на першу сторінку в списку ресурсів НЕ
відбувся. Метод 'ClickMoveToTheFirstPageInFunctionsList'.");
                    return false;
                }
            }
            catch
            {
                Log.Error("Помилка! Метод 'ClickMoveToTheFirstPageInFunctionsList' не
відпрацював.");
                return false;
            }
        }

        /// <summary>
        /// Оновлення в блоці ресурсів (Функцій)
        /// </summary>
        /// <param name="idOfARM">Id вкладки на якій клікаємо по кнопці refresh</param>
        /// <returns></returns>
        public bool RefreshResources(string idOfARM)
        {
            try
            {
                FindElementByXPathAndClick("//div[@id='RoleResourceARM_WEB" + idOfARM +
"']/div[3]/a/span[text()='Оновити']");
                CheckLoading();
                Log.Info("Клік по кнопці оновлення в блоці ресурсів (АРМів) виконано.
Метод 'RefreshResources'.");
                return true;
            }
            catch
            {
                Log.Error("Помилка. Клік по кнопці оновлення в блоці ресурсів (АРМів) НЕ
виконано. Id вибраної вкладки: '" + idOfARM + "'. Метод 'RefreshResources'.");
                return false;
            }
        }

        //NAVIGATIONS ON THE PAGES OF RESOURCES (FUNCTIONS) END
    }
}
```

CreateARM.cs

```
using System;
using System.Threading;
using Common;
using Common.Entities;
using NLog;
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.IE;

namespace COBUMMFO_Autotest.Tests.ConstructorOfARMs
{
    class CreateARM
    {                                          //ТУТ, ПО СУТІ, ТРЕБА ЗМІНЮВАТИ ЛИШЕ
"armForCreate.armCode, armForCreate.armName ТА armForCreate.armType" ТОБТО ПАРАМЕТРИ
НОВОГО АРМу
        // Test Data Begin
        private const string Url = "http://10.10.11.6:8102/barsroot/";
        private string SQLQuery = "select NAME FROM APPLIST WHERE lower(CODEAPP)='" +
armForCreate.ARMCode + "'";
        private const string ColumnFromDb = "NAME";

        Users user_absadm = new Users(userLogin: "suser01", userPassword: "qwerty234");
        ARMs arm1 = new ARMs(armName: "ADM", function_1: "799");
        static ARMs armForCreate = new ARMs(armId: "", armCode: "tes7", armName: "test
arm name 7", armType: "Web Browser");

        private static Logger Log = LogManager.GetCurrentClassLogger();

        // Test Data End

        private IWebDriver _driver;

        [SetUp]
        public void BeforeClass()
        {
            Log.Info("\r\n ВИКОНАННЯ ТЕСТУ 'CreateARM' РОЗПОЧАТО! \r\n");
            _driver = Browser.OpenPageDefault(driver: new InternetExplorerDriver(), url:
Url);
        }

        [Test]
        public void _25_3_CreateARM()
        {
            var loginPage = new LoginPage(_driver);
            var mainPage = new MainPage(_driver);
            var constructorOfArMs = new Functions.ConstructorOfARMs(_driver);

            Assert.IsTrue(loginPage.LoginRegular(userLogin: user_absadm.UserLogin,
password: user_absadm.UserPassword));
            Assert.IsTrue(mainPage.OpenARM(codeapp: arm1.ArmName));
            Assert.IsTrue(mainPage.ChooseFunction(parentArm: arm1.ArmName, codeoper:
arm1.Function_1));
            Assert.IsTrue(constructorOfArMs.Precondition());
            Assert.IsTrue(constructorOfArMs.CreateARM(codeARM: armForCreate.ARMCode,
nameARM: armForCreate.ARMName, typeOfARM: armForCreate.ARMType));
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Assert.AreEqual(expected: armForCreate.ARMName, actual:
constructorOfArMs.GetDataFromDB(sqlQuery: SQLQuery, columnNameOfSearchingRecords:
ColumnFromDb));
        }
```

```
        [TearDown]
        public void Exit()
        {
            var s = new Screenshot1();
            s.TakeScreenshot(driver: _driver, saveLocation: @"D:\Screenshot of test
result\ConstructorOfARMs\CreateARM.png");
            _driver.Quit();
            Log.Info("\r\n ВИКОНАННЯ ТЕСТУ 'CreateARM' ЗАКІНЧЕНО! \r\n");
        }

    }
}
```

EditARM.cs

```
using System;
using System.Threading;
using Common;
using Common.Entities;
using NLog;
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.IE;

namespace COBUMMFO_Autotest.Tests.ConstructorOfARMs
{
    class EditARM
    {                                   //ТУТ, ПО СУТІ, ТРЕБА ЗМІНЮВАТИ ЛИШЕ
"armNew.armName ТА armNew.armType" ТОБТО НОВІ ПАРАМЕТРИ ДЛЯ ІСНУЮЧОГО АРМу
        // Test Data Begin
        private const string Url = "http://10.10.11.6:8102/barsroot/";
        private string SQLQuery = "select NAME FROM APPLIST WHERE lower(CODEAPP)='" +
armForEdit.ARMCode + "'";
        private const string ColumnFromDb = "NAME";

        Users user_absadm = new Users(userLogin: "suser01", userPassword: "qwerty234");
        ARMs arm1 = new ARMs(armName: "ADM", function_1: "799");
        static ARMs armForEdit = new ARMs(armId: "294", armCode: "tes6", armName: "test
arm name 6", armType: "Web Browser");
        ARMs armNew = new ARMs(armId: "", armCode: "", armName: "test arm name 66",
armType: "Centura Desktop");
        FilterForIntegerValues integerValue = new FilterForIntegerValues(equals: "рівне",
notEquals: "не рівними", greater: "більше", less: "менше", greaterOrEqual: "більше або
рівними", lessThanOrEqual: "менше або рівними");
        FilterForStringValues stringValue = new FilterForStringValues(eqauls: "рівні",
notEquals: "не рівні", _in: "містять", notIn: "не містять", begin: "починаються на", end:
"закінчуються на");

        private static Logger Log = LogManager.GetCurrentClassLogger();

        // Test Data End

        private IWebDriver _driver;

        [SetUp]
        public void BeforeClass()
        {
            Log.Info("\r\n ВИКОНАННЯ ТЕСТУ 'EditARM' РОЗПОЧАТО! \r\n");
            _driver = Browser.OpenPageDefault(driver: new InternetExplorerDriver(), url:
Url);
        }
```

```
        [Test]
        public void _25_4_EditARM()
        {
            var loginPage = new LoginPage(_driver);
            var mainPage = new MainPage(_driver);
            var constructorOfArMs = new Functions.ConstructorOfARMs(_driver);

            Assert.IsTrue(loginPage.LoginRegular(userLogin: user_absadm.UserLogin,
password: user_absadm.UserPassword));
            Assert.IsTrue(mainPage.OpenARM(codeapp: arm1.ArmName));
            Assert.IsTrue(mainPage.ChooseFunction(parentArm: arm1.ArmName, codeoper:
arm1.Function_1));
            Assert.IsTrue(constructorOfArMs.Precondition());
            Assert.IsTrue(constructorOfArMs.SelectARMById(armId: armForEdit.ARMId,
enterValueOfFilter: integerValue._Equals));
            Assert.IsTrue(constructorOfArMs.ChooseFoundString(codeOfSelectedResources:
armForEdit.ARMId));
            Assert.IsTrue(constructorOfArMs.EditARM(nameARM: armNew.ARMName, typeOfARM:
armNew.ARMType));
            Thread.Sleep(TimeSpan.FromSeconds(2));
            Assert.AreEqual(expected: armNew.ARMName, actual:
constructorOfArMs.GetDataFromDB(sqlQuery: SQLQuery, columnNameOfSearchingRecords:
ColumnFromDb));
        }

        [TearDown]
        public void Exit()
        {
            var s = new Screenshot1();
            s.TakeScreenshot(driver: _driver, saveLocation: @"D:\Screenshot of test
result\ConstructorOfARMs\EditARM.png");
            _driver.Quit();
            Log.Info("\r\n ВИКОНАННЯ ТЕСТУ 'EditARM' ЗАКІНЧЕНО! \r\n");
        }

    }
}
```

ProvideFunctions.cs

```
using Common;
using Common.Entities;
using NLog;
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.IE;

namespace COBUMMFO_Autotest.Tests.ConstructorOfARMs
{
    class ProvideFunction
    {               //ТУТ ТРЕБА ЗВІРИТИ ЧИ НАДАНА ФУНКЦІЯ АРМу, ЯКЩО ВЖЕ НАДАНА ТО АБО
ВИБРАТИ ІНШУ ФУНКЦІЮ АБО ЗАБРАТИ ЇЇ ПЕРЕД ПОЧАТКОМ ТЕСТУ
        // Test Data Begin
        private const string Url = "http://10.10.11.6:8102/barsroot/";
        private string SQLQuery = "select ACCESS_MODE_ID from ADM_RESOURCE_ACTIVITY where
GRANTEE_ID=" + armForEdit.ARMId + " and ID =(SELECT MAX(ID) FROM ADM_RESOURCE_ACTIVITY)";
        private const string ColumnFromDb = "ACCESS_MODE_ID";
        private const string ExpectedStatusOfFunction = "1";

        Users user_absadm = new Users(userLogin: "suser01", userPassword: "qwerty234");
        ARMs arm1 = new ARMs(armName: "ADM", function_1: "799");
```

```
        static ARMs armForEdit = new ARMs(armId: "293", armCode: "TES7", armName: "test
arm name 7", armType: "Web Browser");
        FilterForIntegerValues integerValue = new FilterForIntegerValues(equals: "рівне",
notEquals: "не рівними", greater: "більше", less: "менше", greaterOrEqual: "більше або
рівними", lessThanOrEqual: "менше або рівними");
        FilterForStringValues stringValue = new FilterForStringValues(eqauls: "рівні",
notEquals: "не рівні", _in: "містять", notIn: "не містять", begin: "починаються на", end:
"закінчуються на");
        Resources function = new Resources(resourceCode: "10000561", resourceName:
"Надіслати повідомленя");

        // Test Data End

        private static Logger Log = LogManager.GetCurrentClassLogger();
        private IWebDriver _driver;

        [SetUp]
        public void BeforeClass()
        {
            Log.Info("\r\n ВИКОНАННЯ ТЕСТУ 'ProvideFunction' РОЗПОЧАТО! \r\n");
            _driver = Browser.OpenPageDefault(new InternetExplorerDriver(), Url);
        }
        [Test]
        public void _25_7_ProvideFunction()
        {
            var loginPage = new LoginPage(_driver);
            var mainPage = new MainPage(_driver);
            var constructorOfArMs = new Functions.ConstructorOfARMs(_driver);
            Assert.IsTrue(loginPage.LoginRegular(userLogin: user_absadm.UserLogin,
password: user_absadm.UserPassword));
            Assert.IsTrue(mainPage.OpenARM(codeapp: arm1.ArmName));
            Assert.IsTrue(mainPage.ChooseFunction(parentArm: arm1.ArmName, codeoper:
arm1.Function_1));
            Assert.IsTrue(constructorOfArMs.Precondition());
            Assert.IsTrue(constructorOfArMs.SelectARMById(armId: armForEdit.ARMId,
enterValueOfFilter: integerValue._Equals));
            Assert.IsTrue(constructorOfArMs.ChooseFoundString(codeOfSelectedResources:
armForEdit.ARMId));
            Assert.IsTrue(constructorOfArMs.OpenResourcesOfARM(armId: armForEdit.ARMId));
            Assert.IsTrue(constructorOfArMs.MakeActiveTab(tabXPath:
Functions.ConstructorOfARMs.TabFunctionsWeb));
            Assert.IsTrue(constructorOfArMs.RefreshResources(idOfARM: armForEdit.ARMId));
            Assert.IsTrue(constructorOfArMs.SelectFunctionByCode(codeOfFunction:
function.ResourceCode, enterValueOfFilter: stringValue._In, idOfARM: armForEdit.ARMId));
            Assert.IsTrue(constructorOfArMs.ChooseFoundString(codeOfSelectedResources:
function.ResourceCode));
            Assert.IsTrue(constructorOfArMs.ChangeAccessAdd(codeOfResources:
function.ResourceCode));                                    //Принципова різниця з тестом
'RemoveFunction' саме тут
            Assert.AreEqual(expected: ExpectedStatusOfFunction, actual:
constructorOfArMs.GetDataFromDB(sqlQuery: SQLQuery, columnNameOfSearchingRecords:
ColumnFromDb));
        }
        [TearDown]
        public void Exit()
        {
            var s = new Screenshot1();
            s.TakeScreenshot(_driver, @"D:\Screenshot of test
result\ConstructorOfARMs\ProvideFunction.png");
            _driver.Quit();
            Log.Info("\r\n ВИКОНАННЯ ТЕСТУ 'ProvideFunction' ЗАКІНЧЕНО! \r\n");
        }
```

ДОДАТОК Б. АКТ ВПРОВАДЖЕННЯ