АНОТАЦІЯ

У магістерській кваліфікаційній роботі було запропоновано універсальну архітектуру програмного забезпечення для тестування Web-додатків. Запропонована архітектура є універсальною для додатків та, в подальшому, може слугувати основною при розробці програмного забезпечення для тестування Web-додатків. На основі запропонованої архітектури було розроблено програмне забезпечення для тестування веб додатку на мові програмування C# та з використанням бібліотеки OpenQA.Selenium.

ANOTATION

In the master's qualification work, a universal software architecture for testing Web-applications was proposed. The proposed architecture is universal for applications and, in the future, can serve as a basis for the development of software for testing Web-applications. Based on the proposed architecture, software was developed for testing web applications using C# programming language and the OpenQA.Selenium library.

ВСТУП

Цю магістерську роботу присвячено створенню універсальної архітектури щодо програмного забезпечення для тестування Web-додатків, які дозволяють розробникам знаходити помилки та швидко і вчасно їх виправляти.

Існує безліч способів написання такого програмного забезпечення. Велика кількість мов програмування мають спеціальні бібліотеки для проведення автоматизованого тестування додатків будь-якого призначення.

Створення будь-якого типу додатку є складною задачею, під час якого розробники можуть допускають багато помилок. Крім того, помилки можуть з’являтися під час внесення змін в програмне забезпечення в процесі його супроводження. Тестування дозволяє знаходити та виправляти ці помилки протягом всього життєвого циклу даного додатку.

Існують різноманітні методики тестування, які дозволяють перевіряти інтерфейс користувача та код програми для створеного програмного забезпечення. В ході тестування виконується велика кількість одноманітної (рутинної) роботи. Для того щоб ефективно використовувати людські ресурси, існує необхідність в створенні програмного забезпечення, яке б займалось такою роботою.

При розробці програмного забезпечення для тестування Web-додатків основною частиною є розробка архітектури. Даний етап є одним із найважливіших, так як правильно розроблена архітектура заощаджує час при створенні додатків та внесення модифікацій в існуючий код. Для розробки архітектури під конкретні вимови використовуються базові принципи та деякі базові компоненти архітектури, які пройшли випробування часом. Але для розробки програмного забезпечення для тестування веб-додатків не існує універсальних принципів побудови архітектури, чи вони знаходяться в закритому

доступі. Тому **актуальною задачею** є розробка універсальної архітектури додатків, що виконують автоматизоване тестування.

Метою дослідження є метод розробки програмного забезпечення для автоматизації тестування Web-додатків у процесі їх створення та експлуатації.

Задачами дослідження є:

— проаналізувати існуючі види та методи тестування та засоби для тестування Web-додатків;

— запропонувати універсальну архітектуру для додатків;

— розробити програмне забезпечення з використання запропонованої архітектури;

— протестувати розроблене програмне забезпечення.

Об’єктом дослідження є процес автоматизації тестування Web-додатків.

Предметом дослідження є методи програмного тестування Web-додатків.

Науковою новизною отриманих результатів магістерської роботи є:

— представлення універсальної архітектури для розробки сучасного програмного забезпечення для автоматизованого тестування Web-додатків, яка дає змогу спростити створення нових тестових методів, та пришвидшити внесення змін та корегування існуючих тестів;

— запропонована архітектура є універсальною для будь-якої мови програмування.

Практичним значенням роботи є те, що:

— представлено універсальну архітектуру для розробки сучасного програмного забезпечення для автоматизованого тестування Web-додатків, яка дає змогу спростити створення нових тестових методів, та пришвидшити внесення змін до існуючих тестів та корегування;

— запропонована архітектура є універсальною для будь-якої мови програмування.

1 АНАЛІЗ ІСНУЮЧИХ ПРИНЦИПІВ ПОДУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ДОДАТКІВ

1.1 Класифікація методів тестування

Тестування програмного забезпечення охоплює багато видів діяльності, подібних до ряду процесів розробки програмного забезпечення. Це включає постановку тестового завдання, розробку та написання тестів, тестування розроблених тестів, проведення підсумкових тестів та вивчення результатів тестування. Архітектура тестів відіграє важливу роль.

Тестування необхідно задокументовувати для подальшого використання. Стандарт документації описаний в міжнародному документі IEEE 829-2008 Standard for Software Test Documentation.

Існує декілька підстав, по яких прийнято робити класифікацію видів тестування [1]:

1) за об’єктом тестування;
   - а) функціональне тестування;
   - б) тестування навантаження;
   - в) тестування легкості та зручності використання;
   - г) тестування інтерфейсу користувача;
   - д) тестування перекладу (локалізації);
   - е) тестування надійності та безпеки.
2) по знаннях про систему, що тестується;
   - а) тестування за методологією «чорної скриньки»;
   - б) тестування за методологією «білої скриньки».
3) за рівнем автоматизації;
   - а) мануальне (ручне) тестування;
   - б) автоматизоване тестування (засобами програмування).
4) за мірою ізольованості;
   - а) юніт тестування, тестування окремих модулів;

б) інтеграційне тестування;

в) системне тестування.

5) за технологією тестування;

а) висхідне тестування;

б) низхідне тестування.

1.1.1 Висхідне і низхідне тестування

При висхідному програма збирається і тестується знизу вгору. Тільки модулі найнижчого рівня (модулі "терміналу": модулі, які не викликають інші модулі) перевіряються незалежно та індивідуально. Після перевірки цих модулів вони повинні бути такими ж надійними, як виклик вбудованих мовних функцій або операторів присвоєння. Потім тестується вже перевірений модуль прямого виклику.

Ці модулі вищого рівня не тестуються автономно, але тестуються разом із модулями нижчого рівня, які протестували. Цей процес повторюють, поки він не досягне вершини. Даний процес охоплює як тест модуля, так і тест поєднання програм.

Для висхідного тесту потрібен драйвер для кожного модуля. Тести повинні подаватися відповідно до комбінації тестованих модулів. Одним з можливих рішень є написати невелику програму для кожного модуля. Тестові дані відображаються як "вбудовані" безпосередньо в цю змінну програми та структуру даних, неодноразово викликаючи тестований модуль, кожен виклик передає нові тестові дані.

Альтернативою даного рішення є використання програми для тестування окремих модулів. Дана програма є інструментом тестування, яка дає змогу розробникам програмного забезпечення описувати тести на будь-якій мові програмування. Даний підхід виключає необхідність написання драйверів.

В даному підході немає проблем, пов’язаних з відсутністю можливості або відносної затратні процеси створення усіх тестових кейсів, що характерні для

низхідного тестування. Драйвер безпосередньо застосовується до модуля, що тестується, і жоден проміжний модуль не розглядається як тестовий інструмент.

Іншою проблемою при висхідному тестуванні є неможливість об’єднання тестів з архітектурою програми, так як тестування розпочинається безпосередньо після розробки та проектування модулів нижчого рівня.

Також, у висхідному тестів з кількома версіями заглушок немає труднощів з показом даних тесту, тому немає труднощів у неповному тестуванні одного модуля при тестуванні іншого модуля.

Висхідний тест - хороший спосіб швидко знайти помилки. Якщо під час тестування одного модуля виявляється помилка, очевидно, що помилка міститься саме в цьому модулі. Не потрібно аналізувати код всієї системи, щоб знайти джерело. Якщо під час єдиного процесу тестування двох попередньо протестованих модулів виникає помилка, це інтерфейс.

Ще однією перевагою проведення тестів методом висхідної тестування є те, що програмісти можуть зосередитись на дуже вузьких областях (одиночний модуль, передача даних між парами модулів тощо). Це робить тест більш ретельним і з більшою ймовірністю виявляє помилки.

Найбільшим недоліком тестування висхідної лінії зв'язку є те, що вам потрібно написати оболонку, що є спеціальним кодом, який запускає тестований модуль. Якщо ви викликаєте інший модуль по черзі, вам потрібно написати "заглушку". Заглушка - це імітація функції, що повертає заздалегідь продумані та розраховані дані. Але даний компонент не несе корисного навантаження.

Зрозуміло, що написання оболонок і заглушок уповільнює роботу, а для кінцевого продукту вони абсолютно даремні. Але написані одного разу, ці елементи можуть використовуватися повторно при кожній зміні програми. Великий набір оболонок і заглушок - це дуже ефективний інструмент тестування.

В протилежність висхідному тестуванню, стратегія цілісного тестування припускає, що до повної інтеграції системи її окремі модулі не проходять особливо ретельного тестування.

Дана стратегія передбачає написання мінімального необхідного набору тестів, які разом покриють увесь функціонал програми. Керівники проектів вибирають даний метод тестування для економії людських ресурсів.

Отже, підсумовуючи переваги та недоліки висхідного методу, можна винести такі основні положення:

— необхідність написання драйверів;
— відносна невеликий розмір тестів;
— мінімальний набір тестів;
— відсутні проблеми з джерелами даних;
— фокусування розробників на окремих, невеликих модулях;
— необхідність написання оболонки;
— неможливість поєднання тесту з архітектурою.

Протилежністю висхідному тестуванню є низхідне тестування. Основною різницею цих двох підходів являється напрям вибору модулів, що тестуються. Так, при обох тестування модулі тестуються не цілком, а частинами, але при низхідному тестуванню спочатку тестуються модулі найвищого рівня, тобто батьківські модулі, а далі, покроково, модулі нижчих рівнів (top-down). Обидві дані способи ще називають інкрементальними видами тестування, саме через поступовість на покроковість вибору методів для перевірки на коректність роботи.

При висхідному тестуванні, одними з основних компонентів тесту є оболонки, що витрачають велику частину часу на їх розробку. При низхідному тестуванні відпадає необхідність в написанні таких оболонок, а заглушки, в ході пониження рівні тестованих модулів, замінюються на реальні модулі.

Думки фахівців про те, яка з двох інкрементальних стратегій тестування ефективніша, сильно розходяться. На практиці питання вибору стратегії тестування зазвичай вирішується просто: кожен модуль по можливості тестується відразу після його написання, в результаті послідовність тестування одних частин програми може виявитися висхідною, а інших - низхідною.

Але насправді, хоч дані підходи є досить різними, але мають низки спільних рис. При низхідному тестуванні програма тестується зверху вниз, що дає певну перевагу в швидкості написані тестів, а також модулі тестуються пов’язуючись між собою. Ізольованим в такому тестуванні залишається лише головний модуль. Після цього до нього по черзі приєднуються класи наступних рівнів, що безпосередньо викликаються основним модулем, і далі виконується загальне тестування даної комбінації.

Даний процес є циклічним на повторюється доки не будуть протестовані нижні рівні програми. При цьому тести будуть максимально схожими до реальних дій, так як на даних рівнях заглушки та оболонки відкидаються, та заміняються на реальні компоненти.

Але при такій концепції тестування виникає очевидна проблема. Як проводити тестування, якщо головний модуль вже написаний, а модулі нижчих рівнів відсутні? Та як подавати тестові дані? Для відповіді на перше питання приходять вже знайомі заглушки, які імітують присутність модуля. При написанні тестів даного типу часто можна зустріти вираз «просто створіть заглушку», що може ввести в оману того, хто буде надалі користуватись даними тестами, через те що дані заглушки в основному зводяться до вже звичного оператора return, який повертає максимально пустий об’єкт, з яким модуль, що викликав дану заглушку зможе правильно функціонувати. Підсумовуючи – заглушка, це такий елемент, який частіше всього повертає заздалегідь створені для основного модуля дані. Але дана концепція має проблему, так як в деяких випадках основний модуль чекає на змінені вихідні параметри, так як змінив вхідні. Інколи, така заглушка має бути досить витонченою та потребує уважності при створенні, адже неправильність роботи такого компоненти може призвести до невірних результатів тесту.

Цікаве і друге питання: в якій формі готуються тестові дані як вони передаються програмі? Якби головний модуль містив усі потрібні операції введення і виведення, відповідь була б проста: тести пишуться у вигляді звичайних для користувачів зовнішніх даних і передаються програмі через

виділені їй облаштування введення. Так, проте, трапляється рідко. Добре спроектованій програмі фізичні операції введення - виведення виконуються на нижніх рівнях структури, оскільки фізичне введення - виведення – абстракція досить низького рівня.

На жаль, метод тестування низхідним принципом має деякі недоліки. Основна проблема полягає в тому, що модулі рідко ретельно перевіряються відразу після підключення. Насправді для детального та коректного тестування певного модуля можуть знадобитися дуже складні вилки. Програмісти, як правило, вирішують створити просту заглушку і перевірити лише деякі умови модуля, не витрачаючи багато часу на програмування усіх можливих випадків.

Другий тонкий недолік методу низхідної тестування полягає полягає в тому, що він може призвести до переконання, що програму можна запрограмувати та протестувати до вищого рівня, перш ніж вся програма буде повністю розроблена. На перший погляд, ця ідея виглядає економічно, але в цілому все навпаки.

Поширеним стилем проектування структури програми є те, що розробник раптом знаходить кращий спосіб, тому, спроектувавши нижчий рівень, необхідно повернутись назад і корегувати код верхнього рівня.

Значна кількість програмного тестування контролює центральний блок, який є симулятором (навіть до управління), паралельно з управлінням периферійними пристроями. Місія симулятора - імітувати поведінку кожного пристрою для підтримки поведінки центрального блоку. Загалом, заглушки дають прості результати, такі як константи та повідомлення про те, що вони мають відношення до вашої роботи.

Тому для розв’язання даної проблеми, програмне забезпечення будуються не строго в низхідній послідовності, а змішаній. Основу даної змішаності складає необхідність в побудові модулів найнижчого рівня, а саме модулів введення – виведення, модулів для роботи з фізичною базою даних, та будь-яких інших джерел інформації. Коли дані модулі розроблені, усі інші модулі будуть готові до тестування, так як немає необхідності в створенні заглушок для інформації. А

отже, усі подальші тести будуть створюватись в формі, які розрахована на кінцевого користувача.

Даний метод, як і висхідний, має свої недоліки та переваги. Основною перевагою над висхідним модулем являється можливість будь-якогої форми тестування, а саме:

— повне тестування;

— часткове тестування;

— тестування будь-яких комбінацій модулів.

Також, очевидною перевагою є форма кінцевих тестів, вони являються зручними та скомпонованими під кінцевого користувача. Іншою перевагою являється можливість проведення тестування, коли є сумніви щодо готовності усіх модулів, який приймають в ньому участь або якщо в проекті програми існують серйозні помилки.

Щодо економії часу, низхідне тестування має переваги, так як немає необхідності в написанні драйверів. Достатньо просто написати необхідні заглушки в тих областях модулів, які не є готовими.

Але дана перевага може виступати і ролі недоліка, так як готовий модуль зазвичай не тестується досконально після його підключення, так як може потребувати вкрай витончених заглушок. Програміст часто нехтує необхідністю повного продумування даних компонентів, і проводить тестування в самих легких та очевидних аспектів модуля.

Другим недоліком даного виду тестування являється ілюзія готовності модуля, і початок його тестування в даному вигляді. При завершенні розробки модулів нижніх рівнів не береться до уваги необхідність у внесенні модифікацій в тести, що були написані з використанням заглушок. Дані тести не правильно виконуються тестування, що призводить до появи помилок в роботі програми.

Нормальний стиль проектування структури програми припускає після закінчення проектування нижніх рівнів повернутися назад і підправити верхній рівень, винісши всього деякі удосконалення або виправляючи помилки, або іноді

навіть викинути проект і почати все спочатку, тому що розробник несподівано побачив кращий підхід.

Отже, підсумовуючи переваги та недоліки низхідного методи, можна винести такі основні положення:

— варіативність вибору форми тестування;
— зрозуміла кінцева форма тестів;
— відсутність необхідності написання драйверів;
— можливість написання заглушок;
— недосконале тестування модулів;
— ілюзія готовності та коректної роботи модуля;
— можливість незакінченого тестування.

Коли ж модуль просто повертає управління або видає деяке повідомлення без передачі певних осмислених результатів, головний модуль працює невірно не внаслідок помилок в модулі, що виконує операцію, а через неправильну роботу модуля-заглушки. Крім того, результат може виявитися незадовільним, якщо дані, отримані заглушкою, не вірними, або призводять до непередбачуваних результатів.

Перевагою ранньої відладки центрального блоку при низхідному тестуванні є те, що програміст швидко дістає можливість перевірити блоки в умовах, які необхідному ступені наближені до реальних. Дійсно, центральний блок, забезпечений хоч би і простими функціональними можливостями, можна розглядати як реальне середовище, в яке «занурюються» відлагоджувані блоки, що додаються до центральної частини.

Варіанти програми для тестування різняться в залежності від мови програмування. Ці варіанти реалізують одні і ті ж функції і при певних тестових даних повинні видавати тотожні результати. Коли результати роботи однакових програм на різних мовах програмування є різними, то існує помилка в роботі тестового модуля.

На практиці створюється 2 варіанти програми, рідко можна зустріти 3 та більше варіантів, оскільки це є невиправдано з точки зору затрат. [2]

1.1.2 Методи «біла скринька» та «чорна скринька»

При тестуванні методом «білої скриньки» розробник тесту має доступ до початкового тестованого коду і може компонувати з ним код тестів. Така ситуація типова для модульного тестування, при якому тестуються тільки окремі частини системи. При тестуванні методом «білої скриньки» може використовуватися знання про внутрішній устрій ПО, що перевіряється, у тому числі і для організації перевірки обробки допустимих, граничних і некоректних даних. Крім того, цей вид тестування дозволяє оцінити рівень покриття коду тестами.

Термін «біла скринька» означає, що при розробці тестових випадків тестувальники використовують будь-які доступні відомості про внутрішню структуру або коду. Технології, вживані під час тестування методом «білої скриньки», зазвичай називають технологіями статичного тестування.

Цей метод не ставить мети виявлення синтаксичних помилок, оскільки дефекти такого роду зазвичай виявляє компілятор [3]. Методи «білої скриньки» спрямовані на локалізацію помилок, які складніше виявити, знайти і зафіксувати. З їх допомогою можна виявити логічні помилки і перевірити міру покриття тестами

Тести, пов'язані з використанням стратегії «білої скриньки», використовують логіку процедур, що керує. Вони надають ряд послуг, у тому числі:

— дають гарантію того, що усі незалежні шляхи в модулі перевірені принаймні один раз;

— перевіряють усі логічні рішення на предмет того, істини вони або неправдиві;

— виконують усі цикли усередині операційних меж і з використанням граничних значень;

— досліджують структури внутрішніх даних з ціллю перевірки їх достовірності.

Тестування за допомогою «білої скриньки», як правило, включає стратегію модульного тестування, при якому тестування ведеться на модульному або функціональному рівні і роботи по тестуванню спрямовані на дослідження внутрішнього устрою модуля. Цей тип тестування називають також модульним тестуванням, тестуванням прозорої скриньки (clear box) або прозорим (translucent) тестуванням, оскільки співробітники, що проводять тестування, мають доступ до програмного коду і можуть бачити роботу програми зсередини.

Цей підхід до тестування відомий також як структурний підхід.

На цьому рівні тестування перевіряється логіка, що керує, яка проявляється на модульному рівні. Тестові драйвери використовуються для того, щоб усі шляхи в цьому модулі були перевірені хоч би один раз, усі логічні рішення розглянуті у всіляких умовах, цикли були виконані з використанням верхніх і нижніх меж і проконтрольовані структури внутрішніх даних. Розглянемо методи тестування на основі стратегії «білої скриньки».

1.1.2.1 Введення невірних значень

При введенні невірних значень тестувальник примушує коди повернення показувати помилки і дивиться на реакцію коду. Це хороший спосіб моделювання певних подій, наприклад переповнювання диска, нестачі пам'яті і так далі. Популярним методом є заміна alloc() функцією, яка повертає значення NULL з метою з'ясування, скільки збоїв буде в результаті. Такий підхід ще називають тестуванням помилкових вхідних даних.

При такому тестуванні перевіряється обробка як вірних, так і невірних вхідних даних. Тестувальники можуть вибрати значення, які перевіряють діапазон вхідних/вихідних параметрів, а також значення, що виходять закордон діапазону.

1.1.2.2 Модульне тестування

При створенні коду кожного модуля програмного продукту проводиться модульне тестування для перевірки того, що код працює вірно і коректно реалізує архітектуру.

При модульному тестуванні новий код перевіряється на відповідність детальному опису архітектури; обстежуються шляхи в коді, встановлюється, що екрани, спадаючі меню і повідомлення належним чином відформатували; перевіряються діапазон і тип даних, що вводяться, а також те, що кожен блок коду, коли треба, генерує виключення і повертає помилки (error returns).

Тестування кожного модуля програмного продукту проводиться для того, щоб перевірити коректність алгоритмів і логіки і те, що програмний модуль задовольняє вимогам, що пред'являються, і забезпечує необхідну функціональність. За підсумками модульного тестування фіксуються помилки, що відносяться до логіки програми, перевантаження і виходу з діапазону, часу роботи і витоку пам'яті.

1.1.2.3 Тестування обробки помилок

При використанні цього методу признається, що нереально на практиці перевірити кожну можливу умову виникнення помилки. З цієї причини програма обробки помилок може згладити наслідки при виникненні несподіваних помилок. Тестувальник зобов'язаний переконатися в тому, що додаток належним чином видає повідомлення про помилку.

Так, додаток, який не повідомляє про системну помилку, що виникла із-за проміжного програмного забезпечення представляє невелику цінність, як для кінцевого користувача, так і для тестувальника.

1.1.2.4 Витік пам'яті

При тестуванні витоку пам'яті додаток досліджується з метою виявлення ситуацій, при яких додаток не звільняє виділену пам'ять, внаслідок чого знижується продуктивність або виникає тупикова ситуація. Ця технологія застосовується як для тестування версії додатка, так і для тестування готового програмного продукту. Можливе застосування інструментів тестування.

Вони можуть стежити за використанням пам'яті додатка впродовж декількох годин або навіть днів, щоб перевірити, чи буде рости об'єм

використовуваної пам'яті. З їх допомогою можна також виявити ті оператори програми, які не звільняють виділену пам'ять.

1.1.2.5 Комплексне тестування

Метою комплексного тестування є перевірка того, що кожен модуль програмного продукту правильно узгоджується з іншими модулями продукту. При комплексному тестуванні може використовуватися технологія обробки зверху вниз і від низу до верху, при якій кожен модуль, що є листом в дереві системи, інтегрується з наступним модулем нижчого або більш вищого рівня, поки не буде створено дерево програмного продукту.

Ця технологія тестування спрямована на перевірку не лише тих параметрів, які передаються між двома компонентами, але і на перевірку глобальних параметрів і, у разі об'єктно-орієнтованого застосування, усіх класів верхнього рівня.

1.1.2.6 Тестування ланцюжків

Тестування ланцюжків має на увазі перевірку групи модулів, що становлять функцію програмного продукту. Ці дії відомі ще як модульне тестування, з його допомогою забезпечується адекватне тестування компонентів системи. Це тестування виявляє, чи досить надійно працюють модулі для того, щоб утворити єдиний модуль, і чи видає модуль програмного продукту точні результати.

1.1.2.7 Дослідження покриття

При виборі інструменту для дослідження покриття важливо, щоб група тестування проаналізувала тип покриття, необхідний для додатка. Дослідження покриття можна провести за допомогою різних технологій. Метод покриття операторів часто називають C1, що також означає покриття вузлів. Ці виміри показують, чи був перевірений кожен виконуваний оператор. Цей метод тестування зазвичай використовує програму протоколювання (profiler) продуктивності.

1.1.2.8 Покриття рішень

Метод покриття рішень спрямований на визначення (у процентному співвідношенні) усіх можливих результатів рішень, які були перевірені за допомогою комплекту тестових процедур. Метод покриття рішень іноді відносять до покриття гілок і називають C2.

Він вимагає: щоб кожна точка входу і виходу в програмі була досягнута хоч би одного разу, щоб усі можливі умови для рішень в програмі були перевірені не менше одного разу і щоб кожне рішення в програмі хоч би одного разу було протестовано при використанні усіх можливих результатів.

1.1.2.9 Покриття умов

Покриття умов схоже на покриття рішень. Воно спрямоване на перевірку точності істинних або неправдивих результатів кожного логічного виразу. Цей метод включає тести, які перевіряють вирази незалежно один від одного. Результати цих перевірок аналогічні тим, що отримують при застосуванні методу покриття рішень, за винятком того, що метод покриття рішень чутливіший до логіки програми, що управляє.

При тестуванні методом «чорної скриньки» тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування. Як правило, тестування методом «чорної скриньки» ведеться з використанням специфікацій або інших документів, що описують вимоги до системи. У цьому виді тестування намагаються забезпечити покриття вимог і вхідних даних.

Тестування на основі стратегії «чорної скриньки» можливо лише за наявності встановлених відкритих інтерфейсів, таких як інтерфейс користувача або програмний інтерфейс додатка (API). Якщо тестування на основі стратегії «білої скриньки» досліджує внутрішню роботу програми, то методи тестування «чорної скриньки» порівнюють поведінку додатка з відповідними вимогами.

Крім того, ці методи зазвичай спрямовані на виявлення трьох основних видів помилок: функціональності, підтримуваній програмним продуктом; вироблюваних обчислень; допустимого діапазону або зони дії значень даних, які можуть бути оброблені програмним продуктом. На цьому рівні тестувальники не досліджують внутрішню роботу компонентів програмного продукту, проте вони перевіряються неявно. Група тестування вивчає вхідні і вихідні дані програмного продукту.

У цьому ракурсі тестування за допомогою методів «чорної скриньки» розглядається як синонім тестування на рівні системи, хоча методи «чорної скриньки» можуть також застосовуватися під час модульного або компонентного тестування.

При тестуванні методами «чорної скриньки» важлива участь користувачів, оскільки саме вони краще всього знають, яких результатів слід чекати від бізнес-функцій. Ключем до успішного завершення системного тестування є коректність даних. Тому на фазі створення даних для тестування украй важливо, щоб кінцеві користувачі надали якомога більше вхідних даних.

Тестування за допомогою методів «чорної скриньки» спрямоване на отримання безлічі вхідних даних, які якнайповніше перевіряють усі функціональні вимоги системи. Це не альтернатива тестуванню по методу «білої скриньки». Цей тип тестування націлений на пошук помилок, що відносяться до цілого ряду категорій, серед них:

— невірна або пропущена функціональність;

— помилки інтерфейсу;

— проблеми зручності використання;

— методи тестування на основі автоматизованих інструментів;

— помилки в структурах даних або помилки доступу до зовнішніх баз даних;

— проблеми зниження продуктивності і інші помилки продуктивності;

— помилки завантаження;

— помилки розрахованого на багато користувачів доступу;

— помилки ініціалізації і завершення;

— проблеми збереження резервних копій і здатності до відновлення роботи;

— проблеми безпеки;

— методи тестування на основі стратегії «чорної скриньки».

Розглянемо методи тестування для пошуку рішення цих помилок.

1.1.2.10 Еквівалентне розбиття

Вичерпне тестування вхідних даних, як правило, нездійсненно. Тому слід проводити тестування з використанням підмножини вхідних даних.

При тестуванні помилок, пов'язаних з виходом за межі області допустимих значень, застосовують три основні типи еквівалентних класів : значення усередині межі діапазону, за межею діапазону і на межі. Виправдовує себе практика створення тестових процедур, які перевіряють граничні випадки плюс/мінус один щоб уникнути пропуску помилок «на одиницю більше» або «на одиницю менше».

Окрім розробки тестових процедур, що використовують сильно структуровані класи еквівалентності, група тестування повинна провести дослідницьке тестування. Тестові процедури, при виконанні яких видаються очікувані результати, називаються правильними тестами. Тестові процедури, проведення яких повинне привести до помилки, носять назву неправильних тестів.

1.1.2.11 Аналіз граничних значень

Аналіз граничних значень можна застосувати як на структурному, так і нефункціональному рівні тестування. Межі визначають дані трьох типів : правильні, неправильні і такі, що лежать на межі. Тестування меж використовує значення, що лежать усередині або на межі (наприклад, крайні точки), і максимальні/мінімальні значення (наприклад, довжини полів). При такому дослідженні завжди повинні враховуватися значення на одиницю більше і менше заграничний.

При тестуванні за межами межі використовується репрезентативний зразок даних, що виходять за кордон, тобто невірні значення.

1.1.2.12 Діаграми причинно-наслідкових зв'язків

Складання діаграм причинно-наслідкових зв'язків - це метод, що дає чітке уявлення про логічні умови і відповідні дії.

Метод припускає чотири етапи. Перший етап полягає в складанні переліку причин (умов введення) і наслідків (дій) для модуля і в привласненні ідентифікатора кожному модулю. На другому етапі розробляється діаграма причинно-наслідкових зв'язків. На третьому етапі діаграма перетвориться в таблицю рішень. Четвертий етап включає встановлення причин і наслідків процесі читання специфікації функцій. Кожній причині і наслідку привласнюється власний ідентифікатор. Причини перераховуються в стовпчику з лівого боку аркуша паперу, а слідства - з правою. Потім причини і наслідки з’єднуються лініями так, щоб були відбиті наявні між ними відповідності. На діаграмі проставляються булеві вирази, які об'єднують дві або більше за причини, пов'язані із слідством.

1.1.3 За об’єктом тестування

Функціональне тестування проводиться для перевірки виконання системою функціональних вимог.

Популярним методом тестування програмного забезпечення є метод тестування навантаження. В ході тестування система проводиться збір інформацію про швидкість оброки інформації в екстремальних умовах, тобто під великим навантаженням. Даний вид тестування дає змогу визначити показники роботи системи, такі як:

— вимоги до ресурсів системи;

— масштабованість системи;

— надійність та стійкість системи.

З точки зору замовника програмного забезпечення, тестування навантаження є одним із способів перевірки роботи системи в максимально незручних умовах.

Виділяють основні показники продуктивності системи, що характеризують швидкість оброки інформації:

— час відгуку (час виконання операції);

— число операцій, що виконуються за одиницю часу.

Після проведення такого тестування проводять аналіз системи та програмного забезпечення, локалізують вузькі місця та створють план оптимізації системи. Загальноприйнятим являється побудова графіка «крива деградація». Даний графік схематично демонструє залежність продуктивності системи від робочого навантаження [4].

Стресове (stress) тестування проводиться в умовах недостатніх системних ресурсів і дозволяє оцінити рівень надійності роботи системи під навантаженням.

Основною ціллю тестування зручності використовування додатку є оцінка прийнятності призначеного для користувача інтерфейсу додатка (кількість часу, що потрібно будь-якому випадковому користувачеві для досягнення мети, а також отриманий результат, простота в доступі до необхідної інформації, інтерпретація відповідей системи і так далі)

Щоб охопити усі аспекти зручності використання, разом з фахівцями із забезпечення якості в цьому виді тестування беруть участь фахівці з різних сфер, а також самі користувачі. Деякими з такий сфер можуть бути психологія та маркетинг, так як саме фахівці даної спеціальності можуть дати оцінку зручності використання та доцільності розробленого функціоналу.

Тестування графічного інтерфейсу користувача (User Interface Testing) припускає перевірку відповідності ЗА вимогами до графічного інтерфейсу користувача. Розрізняють наступні види тестування графічного інтерфейсу користувача :

— на відповідність стандартам графічних інтерфейсів;

— з різними дозволами екрану;

— тестування локалізації додатку, а саме довжини елементів, розмір елементів в якому надписи знаходяться тощо;

— графічного інтерфейсу користувача на різних системних пристроях та з використанням різних допоміжних додатків, наприклад браузер. Для смартфонів такими засобами є емулятори системи.

Іншим підвидом тестування є перевірки безпеки системи. В ході даного тестування проводиться оцінка стану безпеки системи, та вираховують коефіцієнт уразливості системи. Так як і при інших видах тестуваннях, в систему засобами програмного забезпечення подаються максимально граничні значення для перевірки реакції захисних механізмів, реалізованих в системи, на спробу порушити цілісність системи та обходу захисту. Підсумовуючи, в ході такого тестування перевіряють такі спекти системи:

— тестування компонентів системи, що контролюють доступ до неї – допомагає виявити дефекти, в результаті яких користувачі несанкціонований доступ до об'єктів і функцій додатка;

— тестування авторизації та аунтетифікації користувачів – виявляє дефекти, пов'язані з авторизацією та аунтетифікацією окремих користувачів і груп користувачів і з перевіркою їх достовірності;

— тестування систем, що контролюють правильність введених даних - має на меті виявлення помилок в методах перевірки даних, що поступають в систему ззовні;

— тестування механізмів криптографії системи – використовується для виявлення дефектів, пов'язаних з шифруванням і де-шифруванням даних, використанням цифрових підписів і перевіркою цілісності даних;

— тестування правильності обробки помилок – включає перевірку таких аспектів, як виведення на екран фрагментів коду при помилці, вплив помилок на роботу усього застосування, аналіз помилок в коді їх обробки;

— тестування на переповнювання буфера - виявляє вихід за межі буферів при обробці даних;

— тестування конфігурації сервера – дає змоги виявити помилки при конфігуруванні програмного забезпечення, та виключає змогу злому та підробки файлів конфігурації для підміни даних та подальшій неправильній роботі системи.

Завдання архітектора системи полягає в тому, щоб зробити витрати на систематизацію захисту вище, ніж ціна отримуваної в результаті інформації.

В процесі тестування локалізації системи (переклад системи на інші мови) перевіряються різні ділянки графічної частини програми, пов'язані з регіональними особливостями (перевірка роботи різних мовних версій, одиниць вимірів, форматів дат, нумерації днів тижня, порядку сортування і так далі)

Тестування сумісності – перевірка характеристики системи, що вказують на можливість комбінування системи з різними програмними та апаратними рішеннями.

1.1.4 Автоматизоване і мануальне тестування

Тестування, при якому основну роботу по перевірці на правильність роботі перевіяють програмні засоби – називають автоматизованим тестування. Для такого тестування людина приймає участь тільки на стадії розробки тестів. Основною перевагою даного методу тестування являється можливість багаторазового запуску тестів для перевірки на правильність роботи програми. Розглянемо основні принципи автоматизації тестування.

1.1.4.1 Мінімізація дій

Більшість з людей часів пише одиничному тесту, яке перевіряє одну функцію. Але часто люди забувають, що одиничний тест повинен містити єдине твердження. Вони додають велику кількість різних тверджень для єдиної функції, кожного разу передаючи різні комбінації початкових даних до функції під тестом. В цьому випадку, коли існують будь-які несправності твердження, це не управляє іншою частиною тверджень і так результати не забезпечують міру деталізації в усіх можливих помилках, які існують.

З цієї ж причини, утримуйте число тверджень, як можна менше, якщо можливо - в складових і системних випробуваннях також. Також, чим довший тест, тим гірше сприймається. Головна причина, чому люди прагнуть написати довгі тести є, тому що вони хочуть покрити повний випадок використання. Також вони думають, поки ми знаходимося в цьому кроці, ми можемо затверджувати декілька інші речі, які тестувальники зазвичай повинні з мануальним тестуванням.

1.1.4.2 Незалежність

Ні один з тестів не має бути залежне від іншого. Це дуже не важко для дотримання але є одним з найзагальніших помилок, що роблять автоматизатори.

Коли випадок тесту має декілька кроків, часто люди пишуть кожен крок як окреме випробування а потім роблять тестового реалізатора один на одному. Проблема з таким підходом тестувальник не може управляти будь-яким одним випробуванням в ізоляції, або він не може легко вибрати категорію тесту. Також, вам треба відстежувати зміни між ними. Правильне рішення - гарантувати, що кожне випробування незалежне і затверджує один крок. Декілька разів ви можете уникати попередніх кроків в технологічному процесі, встановлюючи залежні дані в основі бази даних. Коли це не можливо, ви можете зробити їх частиною установки тесту.

1.1.4.3 Детермінованість

Наявність невизначеності у тестах підриває будь-яке зусилля автоматизації тестування, тому що ніхто не довіряє результатам. Якщо тест є недетермінованим – 20 це терпить невдачу через неправдивість результатів тестування. Це виникає тоді, коли те, що тест має перевірити на коректну роботу, не працює через розлад в системі, що знаходиться під впливом тестування.

Основним і найбільш популярним методом тестування являється організація тестування на рівні коду, а саме – написання тестів для окремих модулів.

Проте прибічники Test Driven Development (TDD) показали, що при правильній організації коду з використанням паттернів Model View Controller,

створення тестових методів, що імітують дії користувача, не є проблемою. Цей підхід дозволяє протестувати максимальний набір функціоналу серверної частини програмного забезпечення, залишаючи не покритою тестами лише частину, що відноситься до безпосереднього відображення даних.

Другий спосіб автоматизації тестування полягає в імітації дій користувача з використанням спеціальних інструментальних засобів (GUI - тестування). Цей вид тестування відноситься до тестування методом «чорної скриньки».

Існують чотири покоління інструментів і техніки, призначених для організації GUI- тестування.

1.1.4.4 Перше покоління. Утиліти запису і відтворення

Це компоненти, що записують дії під час проведення мануального тестування (макроси). Записані скриптовані макроси можна запустити в будь-який момент часу, імітуючи реального користувача. За допомогою даних утиліт можна наладити автоматичне тестування функціоналу у вигляді комбінації серверної та клієнтської частини. Дані засоби допомагають уникнути одноманітних, рутинних дій та прискорити процес тестування. Головним недоліком даного інструменту являється те, що будь-яка мінімальна зміна положення елементів програми чи зміна порядку виконання певних дій програми приводить до необхідності повторного запису ручних тестів.

1.1.4.5 Друге покоління. Сценарії

Сценарій це форма автоматизації тестування з використанням спеціалізованих скриптованих мов. Мова повинна підтримувати емуляцію дій користувача і отримання результатів дій. Розробкою тестів займаються програмісти, працюючі окремо від тестувальники, безпосередньо запускаючи тести. Зміни в протестованому ПО вимагають внесення виправлень і у відповідних криптах.

1.1.4.6 Третє покоління. Тестування вхідними даними

Це методологія автоматизації тестування, ґрунтована на використанні в скриптах параметрів виконання тестів. Параметри, задаючи логіку роботи тестів (наприклад, вхідні значення і вихідні результати), знаходяться в деякому зовнішньому джерелі даних, наприклад – базі даних. Даний підхід дає можливість зовнішнього налаштування тестів та дає змогу швидко створювати різні набори вхідних параметрів.

1.1.4.7 Четверте покоління. Тестування ключовими словаи

При даному тестуванні створюється спеціалізований набір ключових слів, що описують деякі системні події (наприклад, видалення продукту). З кожним ключовим словом пов'язані необхідні ключові параметри (ідентифікатор продукту, його назва) і очікувані результати. Для кожного ключового слова має бути заданий опис. Даний підхід дає можливість інтуїтивного написання тестів розробниками та дає можливість будь-кому зрозуміти сценарій тесту.

Автоматизовані тести, як правило, на тестування регресії, тобто спрямовані на багаторазовому тестуванні конкретної ділянки програми (модуля), що дозволяє виявити помилки при внесенні змін до існуючих компонентів.

Головною проблем автоматизованого тестування являється необхідність в постійному оновленні тестів, так як модулі, на які націлені створені тести, також оновлюються. Процес оновлення тестів може займати стільки ж часу, а іноді й більше, ніж оновлення самого модуля програмного забезпечення. Проте подібні інвестиції у більшості випадків виправдані, оскільки мануальне тестування вимагає значно більше ресурсів.

Мануальне тестування - це процес пошуку дефектів в роботі програми, коли тестувальник перевіряє працездатність усіх компонентів програми, начебто він був звичайним відвідувачем ресурсу. Для точності перевірки використовуються заздалегідь заготовлені плани тестування, в якому відмічені найбільш важливі аспекти роботи програми. Дані плани також потребують постійного оновлення в відповідності до програмного забезпечення.

Мануальне тестування – це ключовий етап розробки програмного забезпечення. Тестер може недотримуватися строго плану тестування, а відхилятися від нього для повнішого тестування, наближеного до використання програми звичайним користувачем.

Великі проекти дотримуються строгої методології тестування в цілях системного, своєчасного виявлення дефектів програми та їх усунення. Під системністю тестування розуміють такі етапи:

— вибір методології тестування, придбання необхідного устаткування (комп'ютери, програмне забезпечення), прийняття людей на посаду тестерів;

— складання тестів з описом виконання і очікуваним результатом;

— передача сценаріїв мануального тестування тестерам, які послідовно виконують дії, зазначені в даних сценаріях, та записують отримані результати;

— передача результатів тестів розробникам в детальній доповіді з описом усіх знайдених дефектів в роботі програми для їх подальшого обговорення та усунення.

Для тестування можуть бути використані статичний і динамічний підходи. Динамічні підхід включає запуск програмного забезпечення. Статистичне тестування включає перевірку синтаксис і інші особливості коду програми.

Мануальне тестування може застосовуватись як для невеликого програмного забезпечення, так і для складних систем. Але основною сферою застосування являється тестування клієнтської сторони, так як при розробці складних серверних програмних систем можливості мануального тестування сильно обмежені, оскільки при внесенні змін до коду вимагається організувати повторне виконання тестів. Проте мануальне тестування все ж має свої переваги, так як людина здібна знаходити самі витонченні проблеми коду та може проаналізувати будь-яку нестандартну поведінку [5].

1.2 Платформа .NET та мова програмування C#

.NET Framework – програмна платформа, створена Microsoft, яка слугує інструментом для написання додатків різного типу. Основні можливості фреймворку:

— створення десктоп додатків, з допомогою Windows Forms та WPF;

— створення консольних додатків;

— створення бібліотек класів;

— створення WEB – додатків та служб.

Розглянемо основні риси .NET.

1.2.1 Підтримка декількох мов.

Підтримка декількох мов програмування, що дає можливість вибрати найбільш зручну індивідуально та під поставлену задачу. Таку можливість досягнути завдяки загальномовному середовищу виконання CLR (Common Language Runtime), що являється основним компонентом в екосистемі .NET. Даний компонент займається виконанням коду, а саме контролює виділення та очищення пам’яті, компіляцію з проміжного коду в машинний код за допомогою Just-In-Time компілятора. Такий код називають керованим кодом.

1.2.2 Кросплатформеність.

.NET є гнучкою програмною платформою, так як дає можливість написання програм під різні операційні системи. Так, додатки написані з використання .NET Core можуть бути швидко встановлені як на Windows, так на Linux-подібних операційних системах, що дає можливість абстрагуватись під особливостей реалізації цих систем. А завдяки Xaramin можна створювати додатки для сучасних смартфонів на базі Android чи IOS.

1.2.3 Потужна бібліотека класів.

.NET має широкий набір бібліотеки класів, які можуть бути використані на всіх сучасних мовах програмування. Також, велику роль для будь-якої мови

програмування грає обсяг відкритої інформації та кількість людей, що працюють з даною платформою. І тут .NET займає одну з лідируючих позицій, так як обсяг громади є дуже великим. [24]

1.2.4 Різноманітність технологій.

.NET дає можливість створювати додатки будь-яких типів, починаючи від найпростіших Windows-додатків, закінчуючи хмаровими аплікачіями та додатків для сучасних смартфонів. [6]

1.3 Бібліотека OpenQA.Selenium та NUnit

Selenium - відкритий інструмент, який використовується для автоматизації тестування, що здійснюється у браузерах. Тестові методи можуть бути написані у будь-якій з цих мов програмування: Java, Python, C#, Php, Рубін, Perl та .Net.

Тести можуть здійснюватися у будь-яксq з цих ОС: Windows, Mac або Linux.

Тестування може здійснюватися з використанням будь-яких браузерів: Mozilla Firefox, Internet Explorer, Google Chrome , Safari або Opera. Це може бути бути інтегровано з інструментами створення тестів, як наприклад Testng & Junit для управління тестових випадків і виробництва повідомлень. Також може бути інтегровано з Maven, Jenkins та Docker, щоб досягти Continious Integration [25].

Selenium використовується тільки щоб перевірити браузерні застосування. Ми не можемо перевірити desktop додатки або будь-яке інше програмне забезпечення з його допомогою. Не можливо виконувати випробування на зображеннях. Нам треба об'єднати Selenium з Sikuli для тестування зображень, розміщених у Web – додатках.[7]

1.3.1 NUnit

Nunit - бібліотека, яка слугує засобом створення коду, що проектується для написання і запуску тестів в Microsoft .NET. Nunit, подібно до Junit, - аспект Test Driven Development (TDD), який є частиною великого програмного забезпечення, що проектує парадигму, відому як Екстремальне Програмування. Nunit була

створена групою розробників, таких як Джеймі Кенсдейл, Гарі Фелдмен, Чарлі Пул та ін.

Вона має графічний інтерфейс користувача (GUI), подібний до того, що використовувався в Junit. Тести можуть виконуватись безперервно, що робить отримання результатів тесту одразу після виконання. Багаторазові тести можуть виконуватись одночасно. Ніяких суб'єктивних людських суджень або інтерпретацій випробувальних результатів не потрібно. Простота структури робить можливим легко виправляти помилки в роботі додатку, оскільки вони знайдені. Поточна версія Nunit пишеться в C#, об'єктно-орієнтована мова програмування, яка комбінує владу C з простотою Visual Basic. Nunit - один з сім'ї пов'язаних структур випробування, відомих як xunit.[8]

1.4 Аналіз ринку методів побудови програмного забезпечення для автоматизації тестування додатків

Проектування архітектури програмного забезпечення – процес індивідуальний, та виконується на стадії початку розробки. Архітектура базується на основних критеріях тестів від замовника, або від особливостей архітектури тестованого додатку. Загальних положень архітектури ПЗ для автоматизованих тестів немає в відкритому доступі, що створює труднощі при початку їх розробки та втілення в процес тестування додатків. Проаналізувавши наявні публікації на дану тему, було зроблено висновки, які основні тези має включати в себе додатку для автоматизації тестування [26]. Для зручності накладення автоматизованих тестів, на вже наявні тест кейси, структура тест скриптів має бути аналогічна структурі тестового випадку — попередній стан, кроки тесту та стан завершення.

Перерахуємо основні функції скрипта.

1.4.1 Попередній стан

Це початковий етап тесту, в якому проходить ініціалізація тесту. Його основними функціями є:

— ініціалізація додатка (наприклад, відкриття головної сторінки, вхід під тестовим користувачем, перехід в необхідну частину додатка і підведення системи до стану придатного для тестування);

— ініціалізація тестових даних.

1.4.2 Кроки тесту

Цей компонент структури скрипта відповідає за проведення тесту. Його основними функціями є:

— безпосереднє проведення тесту;

— занесення даних про результат тесту, з обов'язковим збереженням причин провалу і кроків, по яких проходив тест.

1.4.3 Стан завершення

Цей етап структури тестів є завершальним. Його основними функціями є:

— видалення, створених в процесі виконання скрипта, непотрібних тестових даних;

— коректне завершення роботи додатку.

Рекомендується також створити загальну бібліотеку по обробці помилок і виняткових ситуацій. Наприклад:

— PreConditionException;

— TestCaseException.

Отже, відповідно до поставленої задачі для розробки універсальної архітектури та створення програмного забезпечення найбільш вдалим типом тестування буде тестування методом «чорної скриньки», так як тести будуть написані без будь-яких знань щодо того, яким чином написаний код для функціоналу додатку, що потрібно протестувати та його алгоритмів. Програмна платформа .NET Framework та мова тестування C# дають увесь необхідний набір бібліотек та функціоналу для створення програмного забезпеченого згідно з технічним завданням. [26]

2 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Загальні відомості про архітектуру програмного забезпечення

Основою написання правильного коду та створення якісного програмного забезпечення, що слугуватиме контейнером з тестовими кейсами, є створення архітектури програми. Архітектура програмного забезпечення це спосіб визначення основних модулів додатку на стадії його розробки, та абстрагуватись від конкретної реалізації поставленої задачі.

На стадії розробки архітектури для програмного забезпечення необхідно чітко визначитись з основними компонентами системи та способи їх взаємодії між собою, а саме шляхи передачі інформації та інші. Усі дослідження записують у формальному вигляді та притримуються цих напрацювань в ході створення програмного забезпечення. Правильно збудована архітектура являється основою продуманого додатку та, в подальшому, заощаджує час на написання окремих модулів.[9]

Згідно матеріалів з досліджень Perry та Wolf архітектурою являється певний набір елементів, що мають такі характеристики:

— властивості форми, що описують її основний напрям і концепцію;

— обмеження, які накладаються на форму;

— пояснення та обґрунтування будови цих форм.

Обґрунтування є основою архітектури, так як пояснює мотиви створення конкретних форм, що, в подальшому, полегшує розуміння деяких специфічних аспектів компонентів, що імплементуються. Незнання вимог, що накладені на форму, може призвести до необачних змін, що порушують розроблену концепцію додатку.

Архітектурою також називають документацію основних принципів, що мають виконуватись при розробці програмного забезпечення. Вона дозволяю чітко зафіксувати, прийняті на етапі розробки архітектури, рішення, та дозволяє

повторно використовувати компоненти дизайну і шаблони проектування повторно в буд-якому іншому проекті.

Основними характеристики архітектури є:

— ізольованість компонентів архітектури;

— складність;

— схема зв’язків компонентів.

2.2 Розробка архітектури для програмного забезпечення

Основними компонентами будь-якого програмного забезпечення для автоматизації тестування являються:

— моделі даних (Рівень моделей даних);

— набір послідовних дій тестування (Рівень бізнес логіки);

— опис логіки дій тестування (Рівень логіки тестів).

Дані компоненти є основними складовими запропонованої архітектури для створення програмного забезпечення, що використовується для тестування веб додатків. Кожен з компонентів має свій рівень ізоляції, що випливає в його залежність тільки від більш високих рівнів. Такий формат залежності дає змогу вносити зміни в будь-який рівень логіки, не змінюючи інші рівні, так як комунікація між ними відбувається неявно, завдяки абстракціям. Абстракція це такий тип компоненту, який описує його роботу без конкретної реалізації, що дає змогу абстрагувати компоненти різних рівнів. Взаємозв’язок компонентів запропонованої архітектури наведено на рисунку 2.1

Використання триярусної архітектури є гарантом можливості розширення та швидкого внесення змін в уже існуюче програмне забезпечення, що є головними критеріями в ході оцінки архітектури додатку. Основним рівнем в даній архітектурі є рівень бізнес логіки, що включає в себе усю логіку виконуваних тестів. Рівень бізнес логіки є найбільшим компонентом даної архітектури, який включає в себе усі необхідні компоненти для виконання тестів.

Для того що виконати основні принципи побудови архітектури, необхідно розбити цей рівень на низько рівневі компоненти. Це дасть можливість легкого

внесення змін в існуючі тести, створення нової логіки для тестів та розширення вже існуючих.[10]

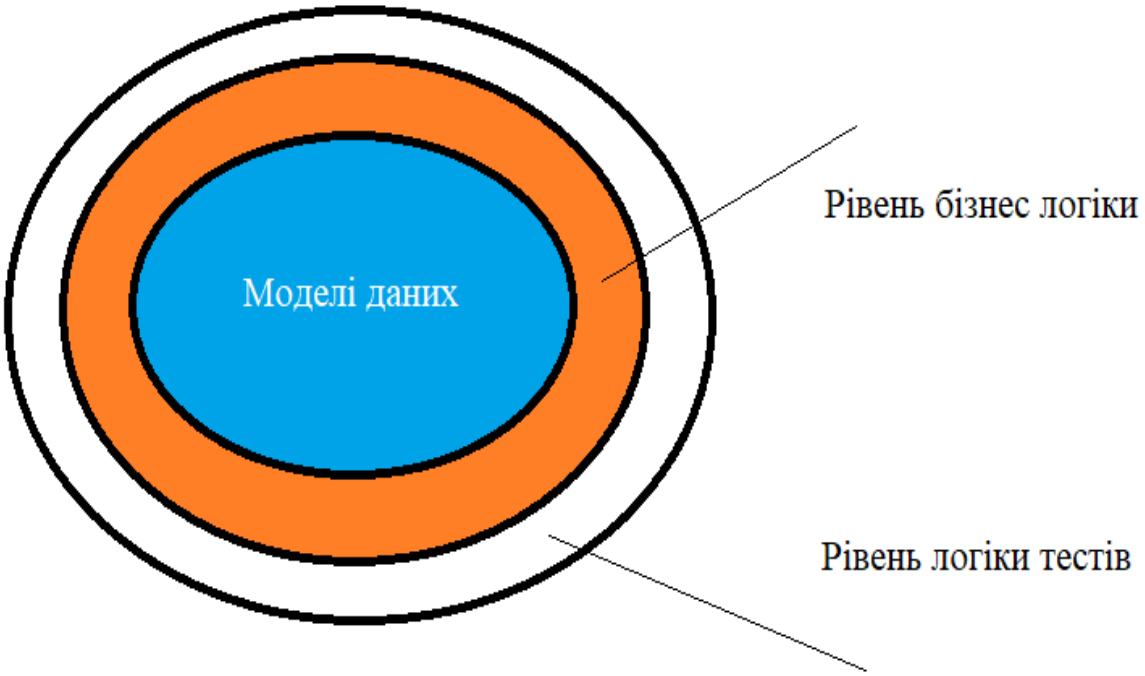


Рисунок 2.1 — Взаємозв’язок компонентів архітектури

2.3 Рівень моделі даних

Основою програмного забезпечення, що було створено в процесі виконання дипломної роботи, є моделі даних. Моделі даних – невід’ємна частина будь-якого програмного забезпечення. Моделі визначають поля, які вона може зберігати, та функціонал, що вона може виконувати, в мові програмування представлений методами та властивостями. Моделі даних є найнижчим рівнем в представленій архітектурі програмного забезпечення. Даний рівень не залежить від будь-якого іншого рівня та являється кореневим рівнем. [11]

2.4 Рівень бізнес логіки

Цей рівень є найбільшим, так як містить основну логіку виконання тестів. Тут відбувається початкове конфігурування тестів, запити до баз даних,

використання зовнішніх ресурсів для обробки та зберігання деяких даних, та сам код, необхідний для виконання тесту.

Основними компонентами для автоматизації тестування веб додатків є:

— веб сторінки (конвертація веб сторінки в елементи коду);

— код виконання тестів (перехід між сторінками та будь-які дії).

Веб сторінки – це компоненти, які зберігаються в собі об’єкти, які напряму зв’язані з веб сторінкою. Прикладом такого об’єкта є будь яка кнопка на сторінці чи текстове поле. Дані об’єкти мають на етапі зв’язування з браузером зберігають в собі всі дані про елемент та надають усі необхідні методи використання цього елементу, такі як введення даних чи натиснення, які потім транслюються в браузер. Основою для таких сторінок в мові програмування C# та бібліотеці OpenQA.Selenium є клас BasePage, що наслідується від HtmlElements.Elements.HtmlPage. Цей клас містить в собі базові поля, для зберіганння основних даних для всіх Web – сторінок, та метод Refresh, визвавши який буде перезавантажено сторінку. Клас HtmlPage наслідується від HtmlElements.WebDriverWrapper, який містить в собі деякий функціонал для роботи з елементами веб сторінок [12]. Основними його властивостями є: protected IPageObjectFactory PageObjectFactory – основна властивість для роботи згідно Page Object Model (POM). POM – це файл-декларації усіх необхідних елементів Web-сторінки. Завдяки цій властивості, кожен раз, при створенні об’єкту класу «сторінки», усі властивості, що задекларовами в цьому класі, та помічені атрибутом «FindBy» будуть проініціалізованими елементами з сторінки (якщо шлях, що вказаний в параметрах FindByAttribute знайдено), та усіма можливи даними, наприклад Id, Text, Attributes та ін. Атрибути FindsBy приймає в себе 2 параметра – How, один з способів пошуку елементу, та string, шлях до елемента згідно з вибраним способом. Основними способами, що використовувались при створенні ПЗ – How.xPath та How.CssSelector, public object ExecuteScript(string script, params object[] args) – метод, який створений для програміста, щоб виконати деякий javascript на сторінці, для отримання деяких даних. Виконання деяких скріптів на сторінці є важливою складовою тестування

Web-додатків, так як сучасні Web-додатки містять в собі велику кількість різноманітних javascript-ів, виконання яких може бути навіть непомітно для кінцевого користувача, public IWebElement FindElement(By by) – метод створений для пошуку елемента на Web-сторінці. Це більш громісткий спосіб в порівняні з використання POM моделі, так як передбачає собою постійний виклик цього методу в коді. Його єдиним параметром є об’єкт, схожий до параметрів атрибута FindsBy.

Компонент, що містить в собі основний код для виконання тестів безпосередньо керує браузером та використовує компонент сторінок, що випливає в його пряму залежність від них. Даний компонент забезпечує цілісність тесту, має гарантувати його виконання та усі необхідні перевірки. Прикладом таких перевірок може бути перевірка тексту будь-якого поля, перевірку наявності елемента на сторінці тощо.

В створеному додатку основою для виконання тестів, а також безспосередньо реалізацією компонента для виконання логіки тестів є так звані «хелпери». Класом батьком для таких «хелперів» є BaseTestHelper, що реалізовую інтерфейс IDisposable та наслідується від класу AssertionHelper.

AssertionHelper – клас, що знаходиться в бібліотеці класів NUnit. Він містить в собі засоби порівняння декількох змінних, що є невід’ємною частиною тестування [13]. Основним його методом є Exptect() та його перевантаження. Повний список перевантажень цього методу представлено на рисунку 2.2 Розглянем найвикористовуваніший з них.

2.4.1 Expect

Першим його параметром є один з об’єтів, що потрібно порівняти між собою, представлений класом object. Другий його параметр – expression, об'єкт класу IResolverConstaint. Цей клас представляє собою одну з логік, за яким слід виконувати перевірку. Для створення об’єкту цього класу існує клас, що знаходиться в просторі імен NUnit.Framework – Is. Наприклад, для перевірки на рівність двох об’єктів можна скористатись методом Is.EqualTo(object o), що

вертає об’єкт класу NUnit.Framework.Constraints.EqualConstraints. Даний об’єкт дає змогу продовжити виконання коду, якщо валідація пройшла усупішно, та закінчує код на непроходженні валідації.

```
public AssertionHelper();

public void Expect(object actual, IResolveConstraint expression);
public void Expect(object actual, IResolveConstraint expression, string message);
public void Expect(object actual, IResolveConstraint expression, string message, params object[] args);
public void Expect(bool condition, string message, params object[] args);
public void Expect(bool condition, string message);
public void Expect(bool condition);
public void Expect<T>(ActualValueDelegate<T> del, IResolveConstraint expr);
public void Expect<T>(ActualValueDelegate<T> del, IResolveConstraint expr, string message);
public void Expect<T>(ActualValueDelegate<T> del, IResolveConstraint expr, string message, params object[] args);
public void Expect<T>(ref T actual, IResolveConstraint expression);
public void Expect<T>(ref T actual, IResolveConstraint expression, string message);
public void Expect<T>(ref T actual, IResolveConstraint expression, string message, params object[] args);
public void Expect(TestDelegate code, IResolveConstraint constraint);
public ListMapper Map(ICollection original);
```

Рисунок 2.2 — Всі методи класу AssertionHelper

Класи, що наслідуються від BaseTestsHelper, мають весь необхідний набір методів, для забезпечення коректного виконання тестів. [14]

Так як рівень бізнес логіки залежить від моделей даних, а, рівень бізнес логіки, що поділяється на сторінки та код виконання тестів, побудуємо фінальну схему взаємозв’язку компонентів запропонованої архітектури (рис. 2.3).

2.5 Рівень логіки тестів

Заключним рівнем архітектури є класи, що містять в собі безпосередньо логіку проведення тестів. Дана логіка полягає в послідовному виконанню операцій, а саме – виклику ключових методів програмного забезпечення, які містяться на рівні бізнес логіки. Отже, даний рівень повністю залежить в рівня бізнес логіки, так як усі необхідні дії для проведення тестів зберігаються саме там. Даний рівень представляє собою набір об’єктів, що створюються по різним

критеріям, наприклад сторінки, що тестується, чи області тестування. Дані об’єкти містять в собі набір тестів для повного покриття тестованої області.[15]

Кожен з таких класів в представленому додатку наслідується від базового класу: BaseTest<T>, where T: BaseTestsHelper

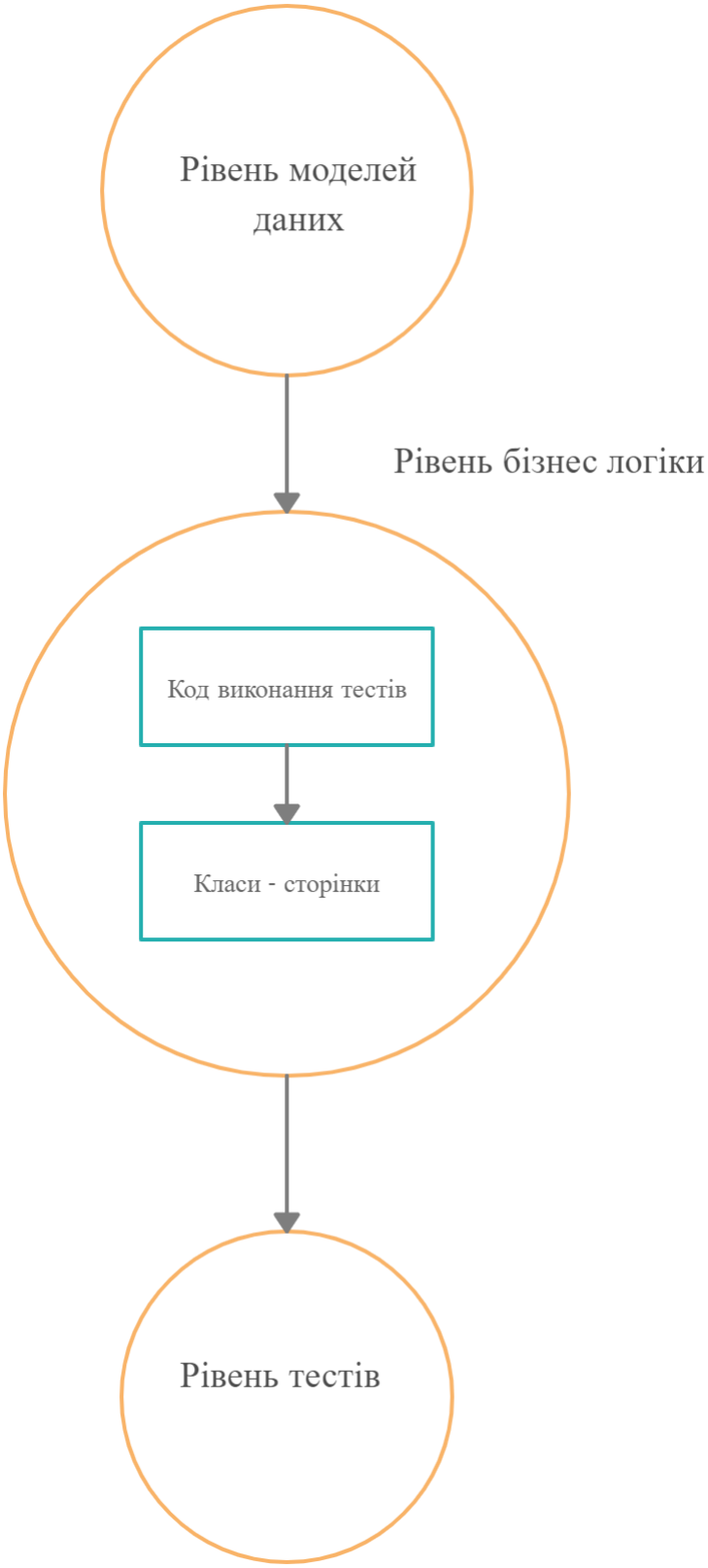


Рисунок 2.3 — Схема взаємозв’язків компонентів архітектури

Цей клас містить в собі початкову конфігурацію для всіх тестів в загалом, та окремо. Конкретну реалізацію такої конфігурації буде розглянуто далі. Клас BaseTest містить в собі об’єкт класу T, де T – BaseTestsHelper. А отже, усі тести мають доступ до своїх «хелперів», а саме до їх методів та полів. Таким чином, було реалізовано пряму залежність рівня логіки тестів з рівним бізнес логіки, так я конкретний об’єкт, що містить в собі набір тестів може виконувати тільки ті дії, що містяться в конкретному об’єкті бізнес логіки.

Отже, було проаналізовано поставлено задачу та запропоновано універсальну архітектуру, яка задовольняє усім вимогам. Запропонована архітектура може слугувати основою при створенні програмного забезпечення для тестування веб додатків. Дана архітектура має ряд переваг, а саме наявність простого механізму додавання тесті та модифікації вже існуючих тестів, що є головними критеріями при розробці програмного забезпечення для тестування як для тестування веб додатків, так для будь – яких інших потреб.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для прикладу буде описано тест для реєстрації нового користувача. Для виконання цього тесту необхідно виконати такі кроки:

— зайти на головну сторінку https://qa.constructsecure.com/csap3;

— натиснути кнопку «Enroll»;

— натиснути кнопку «More»;

— вибрати будь-яку компанію;

— заповнити всі необхідні поля для реєстрації;

— завершити реєстрації;

— вийти з профіля новоствореного користувача;

— зайти в систему під адміністратором;

— перевірити, чи новостворений користувач з’явився в таблиці користувачів;

— видалити тесового користувача.

3.1 Розширення існуючих компонентів

Перед початком створення програмного забезпечення для тестування веб додатку, необхідно створити усі необхідні елементи для коректного проведення тестів.[16] Такими елементами можуть бути розширення існуючих об’єктів в бібліотеці OpenQA.Selenium, що представленні в додатку як об’єкти Elements, та деякі об’єкти, що забезпечують набір специфічних для програмного забезпечення дій – утиліти (Utils)

3.1.1 Кастомні елементи (Elements)

Кастомними елементами є класи, що унаслідуються від класу HtmlElement та розширяють його можливості. Ініціалізуються вони так само як і базовий клас, але містять додаткові поля та функції, що розширяють його можливості.

Наприклад, стандартний клас HtmlElement.DropDown, що представляє собою клас, для роботи з випадаючим списком на веб-сторінці, не задовольняє

усіх потреб для тестованого Web-додатку, так як його випадаючі списки мають нетривіальну структуру. Цей клас містить додаткові HtmlElement-и, які ініціалізуються разом з основним, і виступають в ролі кнопок випадаючого списку, наприклад стрілочка, при натисненні якої мають відобразитись елементи, доступні для вибору, але ця стрілка написана не з допомогою стандартних засобів html.

Іншим прикладом кастомного елемента є клас Menu, що містить в собі набір елементів, які доступні при перегляді будь-якого користувача системи. Це зроблено для того, що розмежувати основні елементи сторінки користувача з його головним меню.[17]

3.1.2 Утиліти (Utils)

Утиліти є важливим елементом бізнес логіки. В основному це класи – що містять в собі логіку обробки деяких специфічних сервісів, бібліотек та засобів, наприклад Google Online Document та робота з JSON форматом даних. Деяка конфігурація проекту та деякі тестові дані, наприклад дані для авторизації в веб-додатку зберігаються в онлайн документі Google Excel. Для роботи з ним було написано утиліту, що представляється класом GoogleSheet.

3.1.3 Файли конфігурації та файли з даними для проведення тестування.

Файли конфігурації – це файли, що містять в собі інформацію, що використовується для початкового конфігурування тестів та всіх службових засобів. Прикладом такого файлу є файл конфігурації App.config. Цей файл містить в собі службову інформацію, для конфігурування віддалених тестових машин, а також необхідну службову інформацію для проведення тестування, наприклад, URL веб-додатку, дані для авторизації під адміністратором та ін.

3.2 Попередня ініціалізація тестів

Як було сказано раніше, уся початкова ініціалізація здійснюється в класі BaseTest<T>. Для ініціалізації тестів, використовуються методи позначені такими атрибутами,

3.2.1 TestFixtureSetUpAttiribute

Метод, позначений цим атрибутом, слугує початковою ініціалізацією для всіх тестів, що будуть виконані за час життя екземпляру програми.[18] Загалом, це стосується тестів, що виконуються в залежності від кількості тестових даних, поданих як вхідні параметри методу, та тестів, що виконуються на віддаленій машині, наприклад Jenkins. Реалізацією такого методу є метод public void InitTests. Впродовж його виконання буде створено та проініціалізовано «хелпер», зконфігуровано об’єкт, що служить для логування ходу виконання тесту та його результатів, створено окрему папку в директорії проекту, яка є контейнером знімків екрану, що будуть зроблені в ході виконання тесту. Код створеного методу подано на рисунку 3.1

```
[TestFixtureSetUp]
public void InitTests()
{
    Helper = new T();
    log4net.Config.XmlConfigurator.Configure(new FileInfo(Path.GetFullPath("App.config")));
    // BaseTestsHelper.TestStatus = TestStatus.Failed;
    Directory.CreateDirectory(Helper.Reports.screenShotDirrectory);
    XLSTestInfo = new XLSTestInformation(Helper);
}
```

Рисунок 3.1 — Лістинг методу InitTests

3.2.2 SetUpAttiribute

Метод, помічений цим атрибутом, слугує конфігурування кожного тестового методу окремо. В ході виконання даного методу буде очищено куки

браузера, для чистоти проведення тестів, обнуління поля StepCounter, який слугує лічильником по виконаним діям в ході проведення тесту, та логування службової інформації, такої як:

— повна назва тесту;

— аргументи, що подані в тестовий метод.

Код створеного методу подано на рисунку 3.2

3.2.3 TearDownAttribute

Метод, позначений цим атрибутом викликається по завершенню одиничних тестів. Він логує інформацію про результат виконання тесту та робить знімок екрану, якщо результат тесту Failed. Даний метод має назву CheckResultsOfTest. Код створеного методу подано на рисунку 3.3

```
[SetUp]
public void Crear()
{
    Helper.StepCounter = 1;
    Helper.WebDriver.Manage().Cookies.DeleteAllCookies();
    BaseTestsHelper.TestStatus = TestStatus.Passed;
    Helper.LOG($"****************************************");
    Helper.LOG($"STARTING NEW TEST: { BaseTestsHelper.TestNameOnly}");
    if (BaseTestsHelper.TestNameWithArguments.Contains('('))
    {
        var tmp = System.Text.RegularExpressions.Regex.Match(BaseTests
        Helper.LOG($"        ARGUMENTS: {tmp.Value.Remove(tmp.Value.La
    }
    Helper.LOG($"****************************************");

}
```

Рисунок 3.2 — Лістинг методу Crear

```
[TearDown]
public void CheckResultOfTests()
{
    Boolean testFailed =
        TestContext.CurrentContext.Result.Status !=
        TestStatus.Passed ||
        BaseTestsHelper.TestStatus !=
        TestStatus.Passed;
    Helper.LOG($"****************************************");
    string status = testFailed ? "FAILED" : "PASSED";
    Helper.LOG($"TEST CASE {status}: {BaseTestsHelper.TestNameOnly}");
    Helper.LOG($"****************************************" + Environment.NewLine);
    if (testFailed && BaseTestsHelper.ScreenshotIterator == 0)
        Helper.takeScreenShot();
}
```

Рисунок 3.3 — Лістинг методу CheckResultsOfTest

3.2.4 TestFixtureTearDownAttiribute

Метод, позначений цим атрибутом викликається після виконання всіх тестів. Він представлений в програмному забезпеченні під назвою CloseBrowser. Він викликає метод Dispose «хелпера», який передбачає звільнення всіх ресурсів, що займав виконуючий процес, закриття браузера, та зберігання результатів тесту. Код створеного методу подано на рисунку 3.4

```
[TestFixtureTearDown]
public void CloseBrowser()
{
    Helper.Dispose();
    Helper.SaveReportBase();
}
```

Рисунок 3.4 — Лістинг методу CloseBrowser

3.3 Створення моделі даних

Для зберігання усієї інформації про користувача та робити з нею було створено клас Sub, що наслідується від класу AccountInfo.

Клас AccountInfo містить основну інформацію про користувача.

Код цього класу подано на рисунку 3.5

```
public class AccountInfo
{
    public string MainOwnerId { get; set; }
    public string CompanyName { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public string UserName { get; set; }
    public bool Fallen { get; set; }
    public UsersType UserType { get; set; }
    public string FederalEIN { get; set; }
    public string Phone { get; set; }
    public string ContactName { get; set; }
}
```

Рисунок 3.5 — Лістинг класу AccountInfo

Поля цього класу буде проініціалізовані випадково згенерованими даними при створенні екземпляру класу Sub. [19]

Конструктор класу Sub подано на рисунку 3.6

В ході виконання конструктора поля CompanyName, Email, FederalEIN буде проініціалізовано випадково згенерованими значеннями. Для цього був створений метод розширення для «хелпера» GenerateRandom.

```
public Sub ()
{
    CompanyName =
        this.GenerateRandom(useNumericCharacters: false,
        presetText: Properties.Settings.Default.DefaultPresetText);
    Email = this.GenerateRandom() + "@constructsecure.com";
    FederalEIN = this.GenerateRandom(false, false, true, false, 9);
    UserName = Email;
    Password = Properties.Settings.Default.OwnersDefaultPassword;
}
```

Рисунок 3.6 — Конструктору класу Sub

Він приймає такі параметри:

— this T helper – сам «хелпер»;

— useLowerCaseChacters – чи використовувати літери нижнього реєстру, за замовчуванням – true;

— useNumericCaseChacters – чи використовувати цифри, за замовчуванням – true;

— useSpecialCaseChacters чи використовувати спеціальні символи, за замовчуванням – false;

— useUpperCaseChacters чи використовувати літери верхнього реєстру, за замовчуванням – true.

Лістинг цього методу наведений на рисунку 3.7

Роглянемо деякі з властивостей класу Sub (Див. рис. 3.8):

— IsUsCountry – чи реєструвати користувача з використанням країни США. Від цього залежить, яку інформацію потрібно вносити при реестрації користувача, та від цього залежить подальші його можливості. По замовчуванню – true;

— Id – унікальний ідентифікатор користувача;

— OwnerName та List<OwnerName> – яку компанію представляє або їх список відповідно.

```
public static string GenerateRandom<T>(this T helper,
    bool useUpperCaseCharacters = true,
    bool useLowerCaseCharacters = true,
    bool useNumericCharacters = true,
    bool useSpecialCharacters = false,
    int size = -1, string presetText = null)
{
    if (size == -1)
        size = RandomGenerator.GetNumber(3, 10);
    var text = RandomGenerator.GetText(
        size,
        useUpperCaseCharacters,
        useLowerCaseCharacters,
        useNumericCharacters,
        useSpecialCharacters
        );
    if (!string.IsNullOrEmpty(presetText))
        text = presetText + text;
    return text;
}
```

Рисунок 3.7 — Лістинг методу GenerateRandom

— IsSafetyComplete та IsFinancialComplete – стан користувача в системі. Після реєстрації вони false;

— IsSafeSupported та IsFinancialSupported – чи підтримує компанія користувача данні профілі;

— IsExpired – стан користувача, чи не закінчився проплачений термін дії.

```
public class Sub: AccountInfo
{
    public Sub ()...
    public bool IsUScountry { get; set; } = true;
    public bool IsUScountrySupported { get; internal set; } = false;
    private string _hrefProfile = string.Empty;
    public string Id { get; set; }
    public OwnerName ownerName { get; set; }
    public List<OwnerName> ownersList;
    public bool FinancialProfileSupported { get; set; }
    public bool TestFinanceComplete { get; set; } = false;
    public bool TestSafetyComplete { get; set; } = false;
    public virtual string GetOwnerName()...
    public static string GetOwnerClassName(string owner)...
    public bool IsSafetyComplete { get; set; }
    public bool IsFinancialComplete { get; set; }
    public bool IsIncorrectData { get; set; }
    public bool IsExpired { get; set; }
    public bool IsPaymentSupported { get; set; } = true;
    public float TotalScore { get; set; }
    public TypeOfSubsRow TypeOfSubsRow { get; set; }
    public bool IsIncompleteData...
    public bool IsSafetyIncomplete => !IsSafetyComplete;
    public bool IsFinancialIncomplete => !IsFinancialComplete;
    public string HrefProfile...
    public bool IsActive { get; set; }
```

Рисунок 3.8 — лістинг класу Sub

3.4 Створення бізнес логіки

Отже, перейдем до безпосереднього опису функціоналу, що був створений для задоволення потреб тесту. Було створено декілька методів в «хелпері»:

— CreateNewSub(Sub out currentSub) – основний метод тесту, що виконує перший 5 пунктів;

— VerifySubIsAvailibleOnVC(Sub currentSub) – логін в систему під адміністратором;

— RemoveSub(Sub currentSub) - видалення новоствореного користувача.

Розберем детально кожен з них.

Public EnrollTestsHelper CreateNewSub(out Sub currentSub)

Це метод оболонка, який вміщає в собі 2 приватних метода.

3.4.1 ClickRandomSubToEnroll

Метод, який виконує основні дії, такі як завантаження головною сторінки, натиснення кнопок «Enroll» та «More» та генерація випадкового значення для визначення компанії, яку буде представляти користувач. Повний лістинг методу приведений на рисунку 3.9

3.4.2 GoToLoginPage

Метод, що створений як метод розширення для всіх «хелперів», знаходиться в CS.AutoTests.Extensions.HelperExtensions. [20] Він очищає куки-файли перед завантаженням головної сторінки, виконую перехід на головну сторінку, в залежності від серверної машини, на якій виконується тестування, та закриває випадаюче вікно, що з'являється при першому відвідуванні сайту (або з порожніми куками), та попереджає про використання куків (рис. 3.10).

```
public EnrollTestsHelper ClickRandomSubToEnroll(out Sub sub)
{
    this.GoToLoginPage();
    ClickEnrollMoreOwners();
    int index = RandomGenerator.GetNumber
        (0, LoginPage.EnrollPopOver.OwnerLinks.Count);
    sub = AllOwnersList.GetSubObjectByName
        (LoginPage.EnrollPopOver.OwnerLinks[index].Text.Clone().ToString());
    LoginPage.ClickOwnerLink(sub);
    Log.Info($"EnrollIn: Owner [{sub.GetOwnerName()}] ");
    return this;
}
```

Рисунок 3.9 — Лістинг методу ClickRandomSubToEnroll

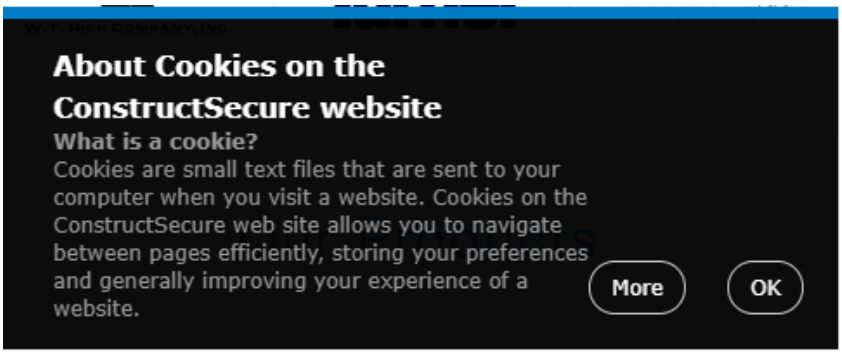


Рисунок 3.10 — Випадаюче вікно Cookie

3.4.3 ClickEnrollMoreOwners

Метод виконує такі дії – натиснення кнопок «Enroll» та «More» та зберігає назви усіх можливих компаній в полі public List<string> OwnersToEnroll. Кнопки, що потрібно натиснути, були задекларовані в класі public class LoginPage : BasePage. Об’єкт цього класу знаходиться в базовому «хелпері», та його поля оновлюються при кожному зверненні до цього поля:

```
public LoginPage LoginPage => PageFactory.Create<LoginPage>(WebDriver);
```

Декларація необхідних кнопок:

```
[FindsBy(How = How.XPath, Using = «//button[@id='Enroll']»)]
public HtmlLink EnrollButton { get; private set; }
[FindsBy(How = How.XPath, Using = «.//button[@id='BtnMoreOwners']»)]
public HtmlLink BtnMoreOwners { get; private set; }
```

Пошук цих кнопок відбувається за допомогою XPath. XPath – це мова запитів до XML, WEB документів для отримання даних. Оператор «//button», означає, що пошук елементу має проводитись для всіх тегів button веб-сторінки, далі в дужках – пошук елемента по його атрибутом, а саме елемент, id якого рівний BtnMoreOwners. При зверненні до поля LoginPage ці кнопки будуть знайдені на сторінці та проініціалізовані необхідними даними. В даному методі також використовуються методи розширення для класу HtmlElement.

3.4.4 WaitFor

Приймає в себе основним параметром функцію, що вертає bool значення, виконую цю функцію кожну секунду доти, доки повернене значення не буде true але не більше 10 секунд за замовчуванням;

3.4.5 TryClick

Метод розширення, що повертає true, якщо елемент був успішно натиснутий, та false, якщо ні. Перевагою цього методу поруч з стандартним методом Click() є те, що стандартний метод, при провальній спробі натиснення на кнопку створить Exception та зруйнує хід виконання програми, а метод

розширення використовує конструкцію try..catch..finally та запобігання таких випадків.

Та базові методи цього класу:

— WaitForPresent() – пауза у виконанні програми доки елемент не з’явиться в коді сторінки, не більше ніж 10 секунд за замовчування;

— WaitForVisible() – пауза у виконанні програми доки елемент не з’явиться на сторінці, не більше ніж 10 секунд за замовчування.

Далі створюється потрібний тип користувача, та базова ініціалізація класу.

internal LoginPage ClickOwnerLink(Sub sub)

Метод, що виконує пошук потрібної кнопки з назвою компанії на сайті, що відповідає новоствореному об’єкту користувача. Декларація цих кнопок:

[FindsBy(How = How.XPath, Using = «.//div[@class='cs-owners-row']/div»)]

public IList<HtmlElement> OwnerLinks { get; private set; }

Заключним етапом є логування назви компанії, що було випадково вибрано.

3.4.6 FillSubDataToEnroll

Метод, що широко використовується в створеному ПЗ (рис. 3.12). Його задачою є заповнення усіх необхідний полів для створення нового користувача. Після вибору компанії, представником якої є новостворений користувач, відкривається перша сторінка, в ході якої потрібно заповнити основну інформацію про користувача.

3.4.7 FillFields

Метод, що займається безпосередньо заповненням всіх полів (рис. 3.11, рис. 3.14). Для заповнення полів необхідно виконати такі дії:

— задекларувати елементи веб-сторінки, які мають бути заповнені тестовими даними;

— внести дані до цих полів.

Усі елементи були задекларовані в класі namespace CS.AutoTests.Pages.Profile.TabContent. Наприклад, там виглядає поле декларації для CompanyName:

[FindsBy(How = How.XPath, Using = «.//input[@id = 'CompanyName']»)]

public HtmlInput Name { get; private set; }

```
public EnrollTestsHelper FillFields(Sub currentSub = null)
{
    this.STEP
        ($"FillFields: for new sub [{currentSub.GetOwnerName()}]");
    var tab =
        EnrollPage.WaitFor(p => p.TabContent, TimeSpan.FromSeconds(20));
    FillFieldsWithoutTrades(currentSub);
    var countSelected = RandomGenerator.GetNumber(2, 5);
    var listOfIndexes = new List<int>();
    scrollIntoView(tab.SelectedTrades);
    for (var i = 0; i < countSelected; i++)...
    tab.SelectedTrades.Click();
    Log.Info
        ($"Finished filling: CompanyName = [{ EnrollPage.TabContent.Name.Value }]");
    return this;
}
```

Рисунок 3.11 — Лістинг методу FillFields

Для декларації поля для введення Federal EIN було створено окремий клас FederalEIN, код якого представлено на рисунку 3.14

```
public EnrollTestsHelper FillSubDataToEnroll
    (Sub currentSub, bool CloseDuplicateWindow = false,
    bool NeedToPay = true)
{
    FillFields(currentSub);
    EnrollSubmit();
    if (CloseDuplicateWindow)
    {
        VerifyMessageBoxText
            (EnrollTests.
            DuplicateFederalIDMessageExpectedText);
        BasePage.MessageBox.ButtonOk.Click();
        Thread.Sleep(2000);
        EnrollSubmit();
        WaitUntilJQueryNonActive();
        // close windo
    }
    if (NeedToPay)
    {
        EnrollAccept(currentSub);
        FillPaymentsFields(currentSub);
        TryCreatePassword(currentSub);
    }
    return this;
}
```

Рисунок 3.12 — Лістинг методу FillSubDataToEnroll

Основним методом для введення даних в необхідний елемент веб-сторінки є методи InputRandom та Input:

— public static T InputRandom<T> (this T helper, HtmlInput element);

— public Boolean Input(HtmlInput input, string text).

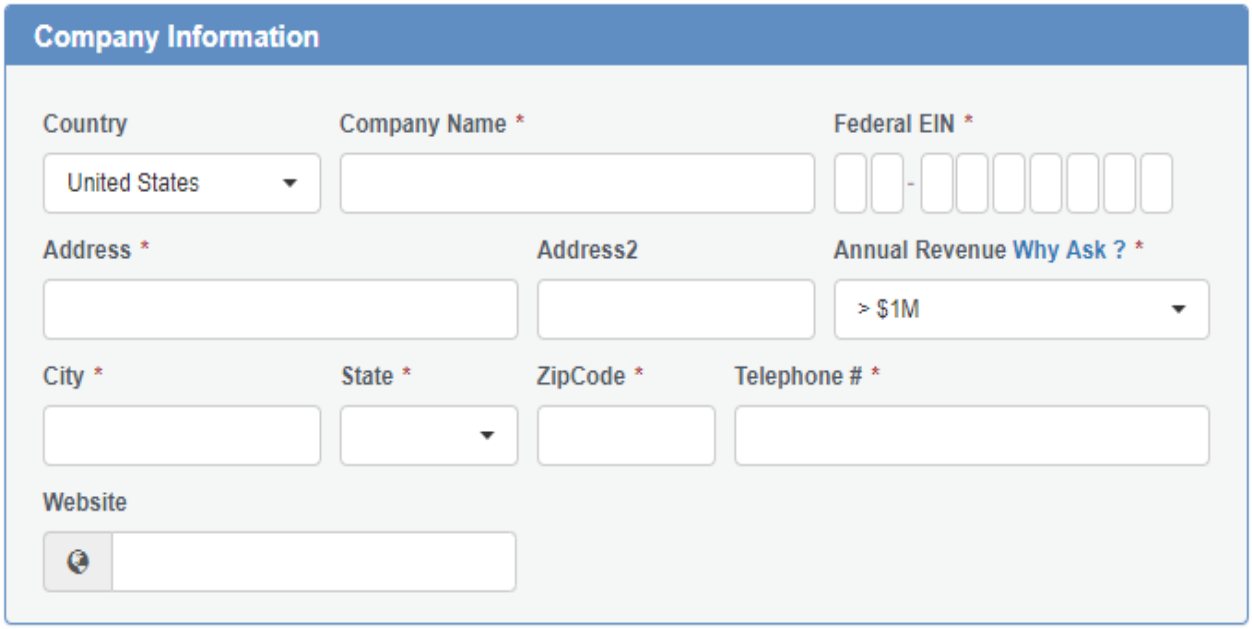


Рисунок 3.13 — Вікно для заповнення інформації

```
public class FederalEIN : HtmlElement
{
    public FederalEIN(IWebElement webElement) : base(webElement)
    {
    }
    [FindsBy(How = How.XPath,
        Using = "../../div[contains(@class,'cs-required')]")]
    public HtmlInput FederalEINLabelRequired { get; private set; }
    [FindsBy(How = How.XPath,
        Using = "../input[contains(@class, 'form-control cs-fed-ein-input')]")]
    public IList<HtmlInput> FederalEINFields { get; private set; }
}
```

Рисунок 3.14 — Лістинг класу FederalEIN

3.4.8 Видалення користувачів з системи

Видалення створеного користувача є також важливим етапом виконання тестів. Так як кількість запусків тестових методів з різними вхідними параметрами є дуже великою, після кожного виконаного тесту потрібно очищати всі створені тестові дані, крім тих випадків, коли тест дай збій. Для видалення користувача з системи був створений метод.

public EnrollTestsHelper RemoveSub(Sub currentSub, bool logingNeeded = true)

Основними його етапами є вхід в систему під створеним користувачем та видалення його з системи. Для того щоб виконати вхід в систему, необхідно прочитати з файлів конфігурації дані користувача адміністратора.

Для останньої дії був створений приватний метод ExecuteRemoveSub (рис. 3.15)

```
private void ExecuteRemoveSub()
{
    try { SafetyPage.SafetyProfileContent.DeleteProfileButton.WaitForPresent(); }
    catch { SafetyPage.SafetyProfileContent.Click(); }
    finally { SafetyPage.SafetyProfileContent.DeleteProfileButton.WaitForPresent().Click(); }
    BasePage.MessageBox.WaitForPresent().ButtonOk.WaitForPresent().Click();
    VerifyMessageBoxText("Your account was deleted.");
    BasePage.MessageBox.ButtonOk.Click();
}
```

Рисунок 3.15 — Лістинг методу ExecuteRemoveSub

Для виконання цього методу було створено декларація елементів, що з’являються після входу в систему під користувачем. Після натиснення необхідних кнопок з’являється повідомлення про видалення користувача, що і є перевіркою правильності виконання операції.

Для порівняння тексту повідомлення, що з’являється, з потрібним (в даному випадку «Your account was deleted.»), було створено окремий метод для верифікації (рис. 3.16).

public void VerifyMessageBoxText(string expectedText)

```
public void VerifyMessageBoxText(string expectedText)
{
    BasePage
        .MessageBox.WaitForPresent(TimeSpan.FromSeconds(30))
        .TryWaitUntil(m => m.MessageBoxContent.Text == expectedText);
    Verify
        (LoginPage.MessageBox.MessageBoxContent.Text,
        expectedText,
        $"Looking for the text on message box: '{expectedText}'");
}
```

Рисунок 3.16 — Лістинг методу VerifyMessageBoxText

В його основі лежить метод Verify, що створений для виконання порівняння очікуваних даних з актуальними, та очікування появи «pop-up» вікна. Про метод Verify буде розказано детальніше в наступному розділі.

3.5 Створення засобів порівняння очікуваних та актуальних даних

Як вже було сказано раніше, основним методом для порівняння та отримання результатів тесту є метод Expect. Але крім безпосереднього порівняння потрібно ще логувати результат та застосовувати деяку логіку обробки даних. [21]

Тому було створено декілька методів «оболонок» для цього методу. Одним і основним таким методом є метод Vefiry (Див. рис. 3.17).

```
public bool Verify(object actual,
    object expected,
    string verifyMessage,
    bool softVerify = false
    )
{
    bool result = false;
    try
    {
        Expect(actual, Is.EqualTo(expected));
        Log.Info("*** PASSED VERIFICATION FOR: " + verifyMessage);
        result = true;
    }
    catch
    {
        string errorMsg = "*** FAILED VERIFICATION FOR: "
            + verifyMessage;
        Log.Info($"      ACTUAL: {actual}");
        Log.Info($"    EXPECTED: {expected}");
        if (softVerify)...
        else
            throw new Exception(errorMsg);
    }
    return result;
}
```

Рисунок 3.17 — Лістинг методу Verify

Основною проблемою Expect є те, що при незпівпаданні вхідних і вихідних даних метод видає помилку в виді Exception та програма закінчує своє виконання. Для цього, виклик цього методу був поміщений в блок try..catch..finally.

Для тестів, в який потрібно зробити перевірку на наявність користувача в системі було створено окремий метод, який інкапсулює в собі використання методу Verify.

public EnrollTestsHelper VerifySubIsAvailibleOnVC(Sub currentSub)

При передачі даному методу об’єкту користувача, він передає в метод Verify bool змінну, яка отримується шляхом виклику методу

GetContractorIdByCompanyName(Sub sub)

Цей метод шукає в таблиці користувачів юзера, що має такий же Id, що і новостворений користувач. Якщо такого знайдено, метод повертає true. А отже, в метод Expect буде передано два аргументи – true та true, а отже верифікацію пройдено.

3.6 Створення тестового методу

Тестовий метод являє собою звичайний метод, помічений атрибутом TestAttribute з бібліотеки класів NUnit. [22] Для тестів, функціонал яких включає в себе реєстрацію нового користувача, було створено окремий клас:

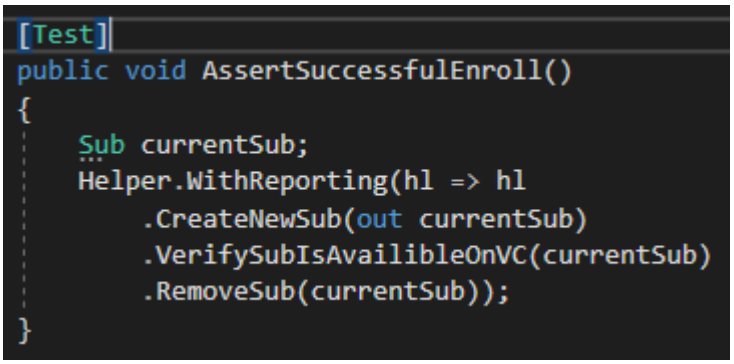
public class EnrollTests : BaseTests<EnrollTestsHelper>

Для задоволення умов тесту було створено тест-метод (рис. 3.18).

Метод WithReporting «хелпера» передбачає собою додаткову логіку для кожного методу в ході виконання тесту.

Отже, було проаналізовано поставлено задачу та продумано необхідно архітектуру, яка задовольняє усім вимогам. Відповідно до поставленої задачі було розроблено програмне забезпечення на основі запропонованої архітектури програмного забезпечення, шо містить в собі автоматизовані методи для проведення випробувань створеного Web-додатку. В третьому розділі було описано процес створення одного з таких методів. Вхідними даними для створення тестів є набір дії, що він має виконувати, та необхідні дані для його

проведення. Лістинг створеного програмного забезпечення наведено в ДОДАТКУ Б та ДОДАТКУ В.



```
[Test]
public void AssertSuccessfulEnroll()
{
    Sub currentSub;
    Helper.WithReporting(hl => hl
        .CreateNewSub(out currentSub)
        .VerifySubIsAvailibleOnVC(currentSub)
        .RemoveSub(currentSub));
}
```

Рисунок 3.18 — Лістинг методу Verify

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Виконання тестового методу

Відповідно до пунктів, описаних у третьому розділі дипломної роботи, буде продемонстровано роботу тесту.

4.1.1 Завантаження головної сторінки https://qa.constructsecure.com/csap3 (рис. 4.1)

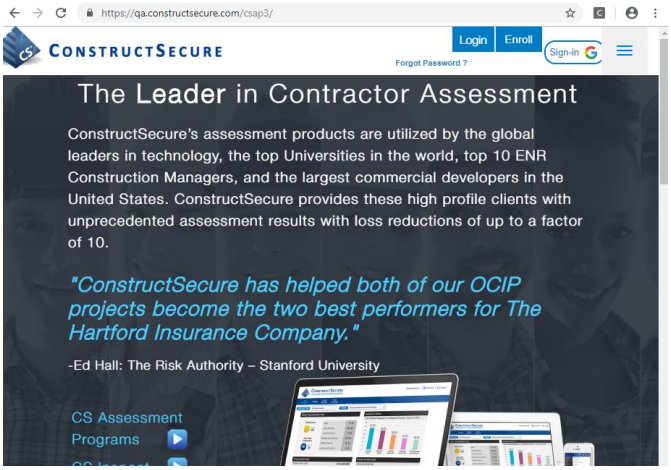


Рисунок 4.1 — Завантаження головної сторінки

4.1.2 Натиснути кнопку «Enroll» та кнопку «More» (рис. 4.2)

Рисунок 4.2 — Результат натиснення кнопок «Enroll» та «More»

4.1.3 Вибрати будь-яку компанію (рис. 4.3)

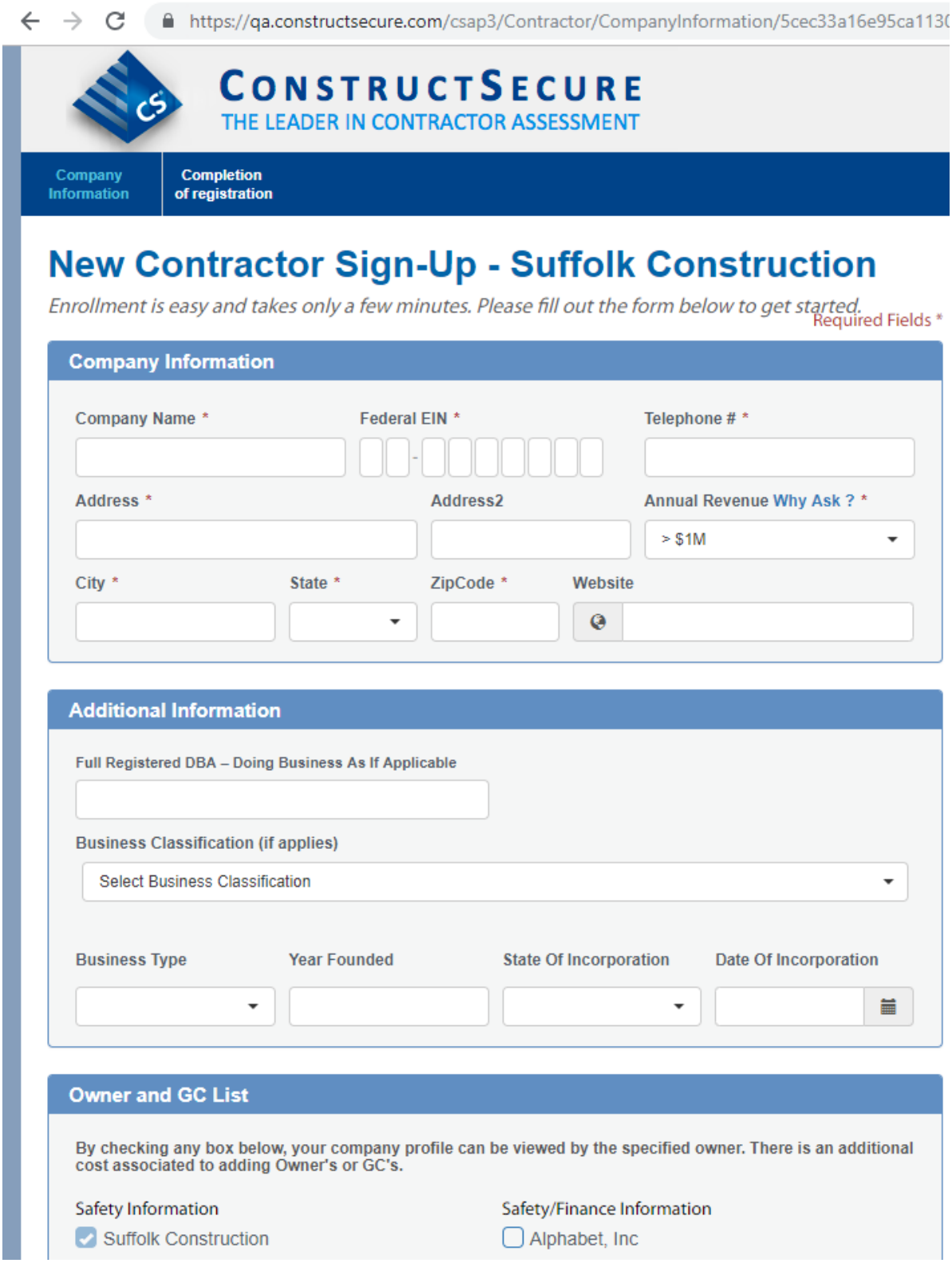


Рисунок 4.3 — Результат вибору компанії

4.1.4 Заповнити всі необхідні поля для реєстрації та перехід до оплати користувача (рис. 4.4, рис. 4.5)

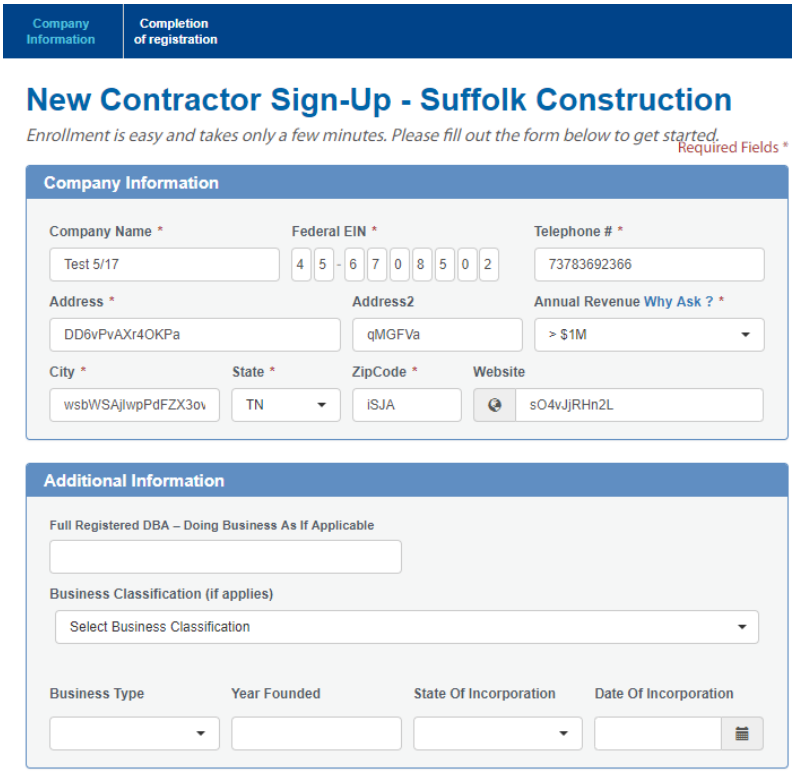


Рисунок 4.4 — Заповнені поля

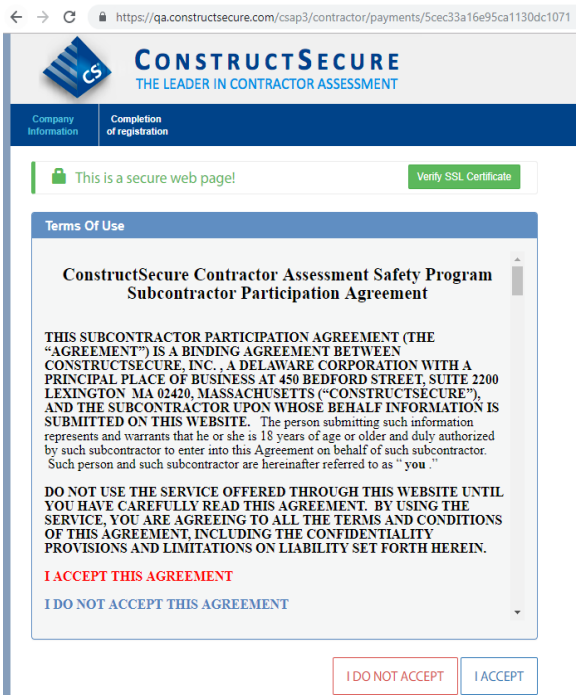


Рисунок 4.5 — Перехід на наступну сторінку

4.1.5 Завершення реєстрації (рис. 4.6, рис. 4.7, рис. 4.8)

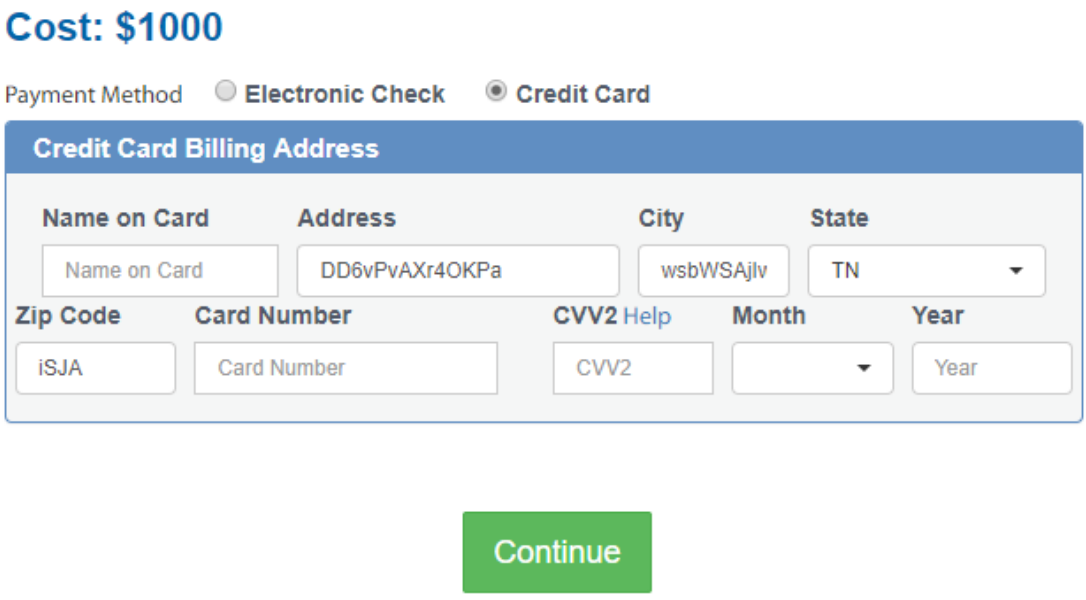


Рисунок 4.6 — Заповнення полів для оплати користувача

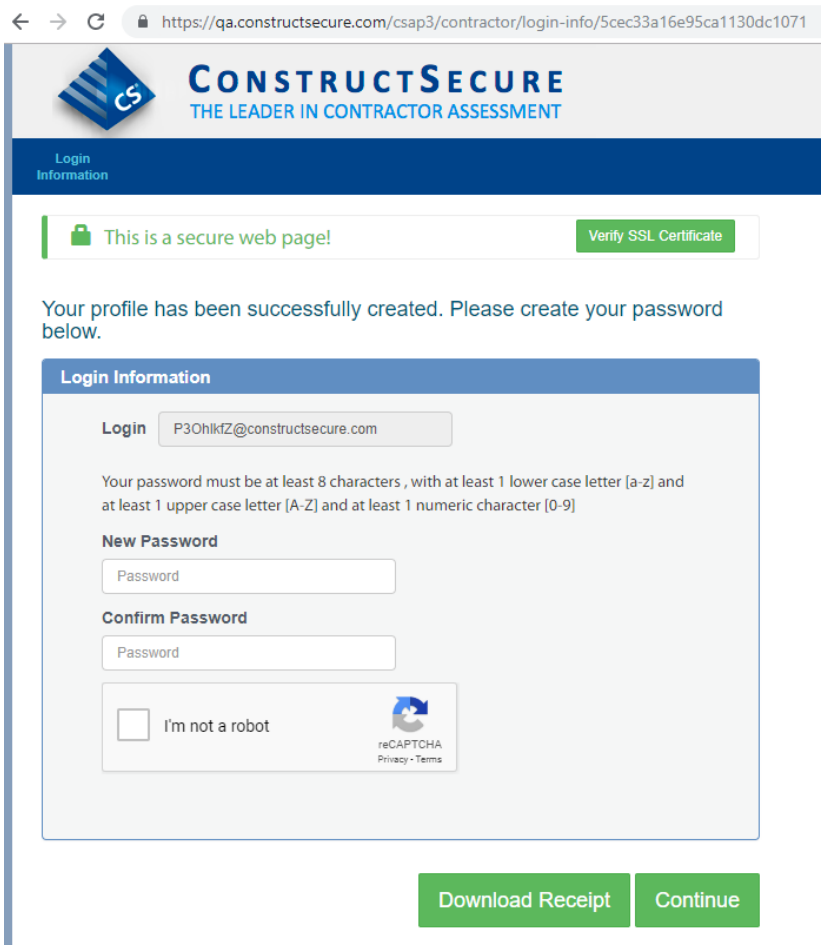


Рисунок 4.7 — Створення паролю

Рисунок 4.8 — Профіль створеного користувача

4.1.6 Вийти з профіля новоствореного користувача

4.1.7 Зайти в систему під адміністратором

4.1.8 Перевірити, чи новостворений користувач з’явився в таблиці користувачів (рис. 4.9)

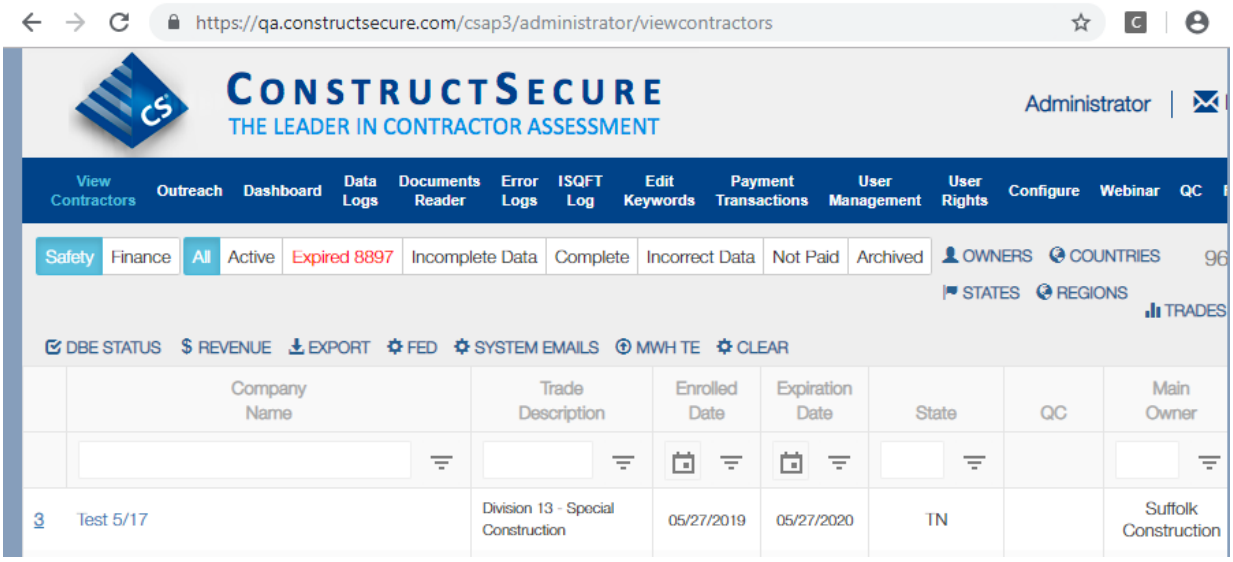


Рисунок 4.9 — Верифікація наявності новоствореного користувача

4.1.9 Видалити новоствореного користувача (рис. 4.10)

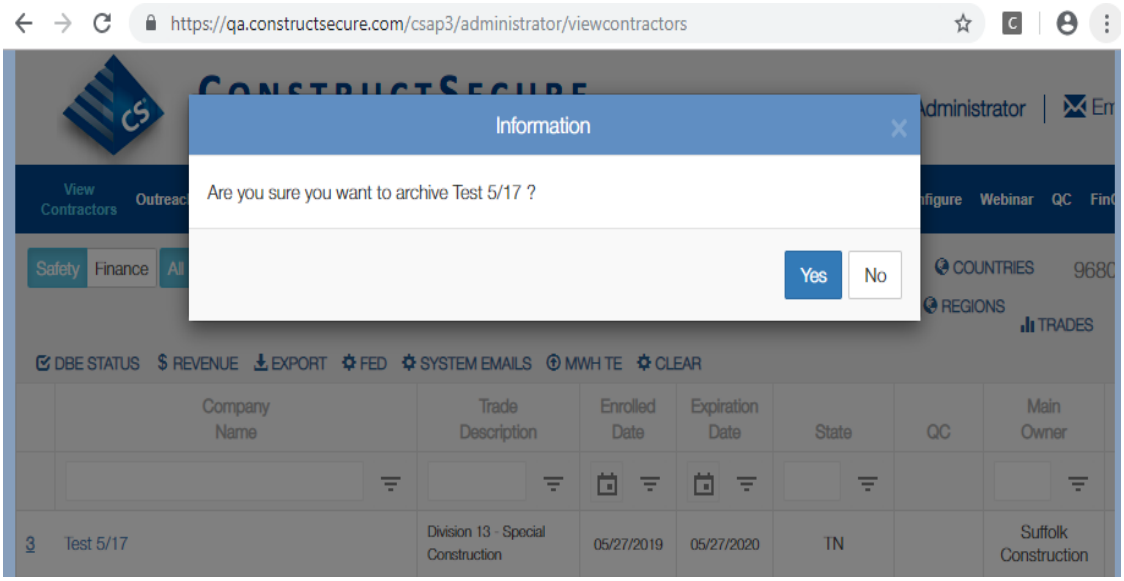


Рисунок 4.10 — Видалення новоствореного користувача

4.2 Перевірка результатів

Отже, в ході виконання автоматизованого тесту було зареєстровано нового користувача. За поставленою задачею для верифікації тесту виконавчому процесу

необхідно було перевірити наявність створеного користувача в таблиці користувачів. Дана таблиця доступна тільки адміністратору систему. Після авторизації під користувачем адміністратору було виконано верифікацію користувача, через надпис у правому верхньому кутку веб сторінки.

В даному розділі було продемонстровано роботу тесту, який був створений та описаний в третьому розділі. Створений тестовий метод повністю задовольняє вхідні дані та значно скорочує час виконання такого тесту в мануальному режимі, так як для його проведення необхідно заповнити велику кількість полів з даними.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Крупельницький Леонід Віталійович (к.т.н., доц. кафедри ОТ ВНТУ), Азаров Олексій Дмитрович (д.т.н., проф. кафедри ОТ ВНТУ).

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Проведено оцінювання комерційного потенціалу за критеріями, наведеними в таблиці 5.1.1.

Таблиця 5.1 — Критерії оцінювання комерційного потенціалу розробки бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					

2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-тер.	0	1	2	3	4
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна Конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					

8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне значне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Кінець таблиці 5.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні Незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої	Необхідно отримання великої кількості дозвільних документів на виробництв	Процедура отримання дозвільних документів для виробництва та реалізації	Необхідно тільки повідомлення відповідним органам про виробництво та	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

	кількості дозвільних документів на виробництво та реалізацію продукту	о та реалізацію продукту, що вимагає значних коштів та часу	продукту вимагає незначних коштів та часу	реалізацію продукту	

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Азаров О.Д	2. Крупельницький Л.В.
	Бали, виставлені експертами:	
1	3	2
2	4	3
3	2	3
4	3	3
5	3	2
6	4	3
7	3	3
8	3	4
9	3	3
10	4	4
11	4	4
12	3	3
Сума балів	$СБ_1 = 39$	$СБ_2 = 37$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 38$	

Отже, з отриманих даних таблиці 5.1.2 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько-технологічної роботи.

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_o = \frac{M}{T_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$T_р$ - кількість робочих днів у місяці, $T_р$ = 22 дні;

t - число днів роботи розробника, t = 45 днів.

Розрахунки заробітних плат для керівника і програміста, їх окрему за загальну суми наведені в таблиці 5.3

Таблиця 5.3 — Розрахунки основної заробітної плати

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	25000	1136	4	4544
Інженер-програміст	25000	1136	20	22720
Всього:				27264

Розрахуємо додаткову заробітну плату:

$$Здод = 0{,}1 \cdot 27264 = 2726{,}4 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 22% від суми їхньої основної та додаткової заробітної плати:

$$Нзп = (Зо + Зр) \cdot \frac{\beta}{100}, \qquad (5.2)$$

$$Нзп = (27264 + 2726{,}4) \cdot \frac{22}{100} = 6597{,}9 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \qquad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.4 — Розрахунок амортизаційних відрахувань

Найменування	Ціна, грн.	Норма амортизації, %	Термін використання, м.	Сума амортизації
ПК + прилади керування	9000	20	2	300
Прилади маніпуляції ПК	1000	20	2	33
Всього	333			

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$К = \sum_1^n Н_і \cdot Ц_і \cdot К_і, \qquad (5.4)$$

де n – кількість комплектуючих;

$Н_і$ - кількість комплектуючих і-го виду;

$Ц_і$– покупна ціна комплектуючих і-го виду, грн;

$К_і$ – коефіцієнт транспортних витрат (приймемо $К_і$ = 1,1).

Таблиця 5.5 — Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	165	1	165
Пачка паперу	уп.	130	1	130
Ручка	шт.	15	1	15
Всього з урахуванням транспортних витрат				310

Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п ; \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,0 грн/кВт);

П – установлена потужність комп'ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп'ютера (Ф=210 год.);

$К_п$ – коефіцієнт використання потужності ($К_п$< 1, $К_п$ = 0,7).

$$Ве = 1,0 \cdot 0,6 \cdot 210 \cdot 0,7 = 88,2 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1.5 * (27264 + 2726,4) = 44985,6 \text{ (грн).}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

$$В = 27264+2726,4+44985,6+88,2+320+333= 75717,2 \text{ (грн.)}$$

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{75717,2}{1} = 75717,2$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{75717,2}{0,9} = 84130,2 \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \Delta N)_i \quad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення програмного продукту зменшаться на 1000 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 1000 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 15 користувачів, протягом другого року – на 12 користувачів, протягом третього року – 10 користувачів. Реалізація програмного продукту до впровадження результатів наукової розробки складала 20 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 20000 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 20 \cdot 1000 + 1000 \cdot 15 = 35000 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 20 \cdot 1000 + 1000 \cdot (15 + 12) = 47000 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 20 \cdot 1000 + 1000 \cdot (15 + 12 + 10) = 57000 \text{ грн.}$$

5.4 Розрахунок ефективності вкладеній інвестицій та період їх окупності.

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується за формулою:

$$Е_{абс} = (ПП - PV), \quad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, який представлено на рисунку 5.1

Рисунок 5.1 — Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки; 42350 62557

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{20000}{(1+0,1)^0} + \frac{35000}{(1+0,1)^2} + \frac{47000}{(1+0,1)^3} + \frac{57000}{(1+0,1)^4} = 208360,7 \text{ (грн.)}$$

Тоді розрахуємо $Е_{абс}$:

$$Е_{абс} = 208360,7 - 84130,2 = 124230,5 \text{ грн.}$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$E_в = \sqrt[T]{1 + \frac{E_{абс}}{PV}} - 1 \qquad (5.12)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$E_в = \sqrt[3]{1 + \frac{208360,7}{124230,5}} - 1 = 0,38 \text{ або } 38\ \%$$

Далі, розраховану величина $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0,2 + 0,1 = 0,3$$

Оскільки $E_в$ = 38% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $Т_{ок}$ розраховується за формулою:

$$Т_{ок} = \frac{1}{0,38} = 2,63 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

ВИСНОВКИ

У даній магістерській кваліфікаційній роботі було представлено універсальну архітектуру програмного забезпечення для тестування Web-додатків. Було проаналізовано принципи побудови програмного забезпечення для тестування Web-додатків, запропоновано універсальну архітектуру для додатків, розроблено програмне забезпечення з використання запропонованої архітектури та перевірено його працездатність. Представлена універсальна архітектура програмного забезпечення може слугувати основою при створенні додатків, для тестування Web-аплікацій та значно заощаджує час написання тестових методів, а також внесення змін в існуючі додатки. Також, розроблене програмне забезпечення суттєво скорочує часові ресурси виконання тестів мануальних тестувальників. Досягнення даної мети стало можливим після вирішення таких задач, поставлених на початку магістерської кваліфікаційної роботі. Основою даної роботи є розробка універсальної архітектури. Розробка програмного забезпечення була почата з створення проекту в середовищі Visual Studio. Було вдало обрано мову програмування – C#, так як він багатий на відкриті бібліотеки класів, що дають широкий спектр можливостей в сфері тестування. Для імітації роботи користувача в веб-додатку було використано бібліотеку OpenQA.Selenium, що представляє програмісту функціонал для створення тестового вікна браузера та виконання ряду вказаних дій. Основною для створення тестових методів було обрано стандартну бібліотеку nUnit. Було створено ряд моделей даних, що забезпечують зберігання необхідних даних в ході виконання тестів. В ході розробки ПЗ написано близько десяти «хелперів», що інкапсулюють в собі функціонал для проведення тестових методів, а також розроблено додаткові класи-утиліти, що використовувались для обробки нетривіальних даних та роботу з сторонніми ресурсами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сем Канер. Тестирование програмного обеспечения / Сем Канер, Джек Фолк – DIA SOFT, 2010 – 552 с.

2. Восходящее и нисходящее тестирование [Електронний ресурс] – Режим доступу: https://life-prog.ru/2_85719_voshodyashchee-i-nishodyashchee-testirovanie.html

3. White/Black/Grey Box-тестирование [Електронний ресурс] – Режим доступу: https://qalight.com.ua/baza-znaniy/white-black-grey-box-testirovanie/

4. Функціональне тестування [Електронний ресурс] – Режим доступу: https://qaevolution.ru/testirovanie-po/vidy-testirovaniya-po/funkcionalnoe-testirovanie/

5. Дж. Рихтер. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Дж. Рихтер , 2005. – 1516 с.

6. OpenQA.Selenium [Електронний ресурс] – Режим доступу: https://www.seleniumhq.org/

7. NUnit [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/NUnit

8. Peter Eeles. Архітектура програмного забезпечення [Електронний ресурс] – Режим доступу: https://www.ibm.com/developerworks/ru/library/eeles/index.html

9. Комп’ютерні мережі / О. Д. Азаров, С. М. Захарченко, О. В. Кадук, М. М. Орлова, В. П. Тарасенко // Навчальний посібник. – Вінниця: ВНТУ, 2013./МОНУ (Лист №1/11 – 8260 від 15.05 2013 р.) – 500 с.

10. Комп’ютерні мережі / О. Д. Азаров, С. М. Захарченко, О. В. Кадук та ін. // Підручник. – Вінниця: ВНТУ, 2020 – 378с.

11. Інтегроване середовище розробки Microsoft Visual Studio [Електронний ресурс]: режим доступу до матеріалу: http://uk.wikipedia.org/wiki/Microsoft_Visual_Studio

12. Степанченко И.В. Методы тестирования программного обеспечения: учебное пособие / И.В. Степанченко. – Волгоград : ВолгГТУ, 2006. – 74с.

13. Роберт Мартін. Чиста архітектура. Мистецтво розробки програмного забезпечення [Електронний ресурс] – Режим доступу: https://mybookshelf.com.ua/robert-martin-chista-arhitektura-mistetstvo-rozrobki-programnogo-zabezpechennya/p3526

14. Кормен Т. Алгоритмы: построение и анализ, 2-е издание.: Пер. с англ. / Т. Кормен – М.: Издательский дом «Вильямс», 2005. – 1296 с.

15. Симен Марк. Внедрение зависимостей в .NET Core/ М. Симен, 2011. – 317 с.

16. Филип Япиксе, Эндрю Троелсен. Язык программирования C# 6.0 и платформа .NET 4.6 / Ф. Япиксе, Э. Троелсен, 2015 – 1440 с.

17. Филип Япиксе. Pro C# 7: With .NET and .NET Core / Ф. Япиксе, 2017 – 1328 с.

18. «Банда четырёх»: Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / «Gang Of Four», 1994 – 395с.

19. Кати Сьерра, Робсон Элизабет. Паттерны проектирования / К. Сьерра, Р. Элизабет, 1994 – 657с.

20. Эрик Фримен. Паттерны проектирования / Э. Фримен, 2011 – 645с.

21. Сергей Тепляков. Паттерны проектирования на платформе .NET / С. Тепляков, 2015 – 320с.

22. Дороти Грейам, Марк Фьюстер. Software Test Automation: Effective Use of Test Execution Tools / Д. Грейам, М. Фьюстер, 1999 – 574с.

23. О. Д. Азаров, С. М. Захарченко. Сучасні методи підготовки та сертифікації спеціалістів в галузі мережних технологій / О. Азаров, С. Захарченко // Шляхи та проблеми входження освіти України в світовий освітянський простір: Збірник доповідей міжнародної науково-технічної конференції, м. Вінниця, 8-9 червня 1999р. В 2-х томах. Том 2.- Вінниця: “Універсам-Вінниця”, 1999. - С. 276-278.

24. Джирард Месарош. XUnit Test Patterns: Refactoring Test Code / Д. Месарош, 2007 – 944с.

25. Иван Винниченко. Автоматизация процессов тестирования / И.Винниченко, 2005 – 214.

26. Роберт Мартин. Создание, анализ и рефакторинг / Р.Мартин, 2008 – 464с.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖЕНО
Завідувач кафедри КІ
_____проф., д.т.н., Т. Б. Мартинюк
«___»_____2020 р.

Технічне завдання
На магістерську кваліфікаційну роботу:
«Універсальна архітектура побудови додатків для автоматизованого тестування»
08-23.МКР.017.00.000 ТЗ

Керівник роботи:
_____ д.т.н., проф. Азаров О.Д.
«___»_____2020 р.
Виконав: студент гр. 1КІ-19М
Шевченко Андрій Ігорович
«___»_____2020 р.

2020

1 Назва і галузь застосування

Магістерська кваліфікаційна робота на тему: «Універсальна архітектура побудови додатків для автоматизованого тестування».

2 Підстава для розробки

Магістерська кваліфікаційна робота виконується на підставі завдання керівника роботи проф., д.т.н., Азаров О.Д.

3 Мета і призначення розробки

Метою дослідження є метод розробки програмного забезпечення для автоматизації тестування Web-додатків у процесі їх створення та експлуатації.

Задачами дослідження є:

— проаналізувати існуючі види та методи тестування та засоби для тестування Web-додатків;

— запропонувати універсальну архітектуру для додатків;

— розробити програмне забезпечення з використання запропонованої архітектури;

— протестувати розроблене програмне забезпечення.

Об’єктом дослідження є процес автоматизації тестування Web-додатків.

Предметом дослідження є методи програмного тестування Web-додатків.

Наукова новизна отриманих результатів магістерської роботи полягає у тому, що:

— представлено універсальну архітектуру для розробки сучасного програмного забезпечення для автоматизованого тестування Web-додатків, яка дає змогу спростити створення нових тестових методів, та пришвидшити внесення змін до існуючих тестів та корегування;

— запропонована архітектура є універсальною для будь-якої мови програмування.

Практичне значення полягає в тому, що

— представлено універсальну архітектуру для розробки сучасного програмного забезпечення для автоматизованого тестування Web-додатків, яка дає змогу спростити створення нових тестових методів, та пришвидшити внесення змін до існуючих тестів та корегування;

— запропонована архітектура є універсальною для будь-якої мови програмування.

4 Джерела розробки

— Дж. Рихтер CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Дж. Рихтер , 2005. – 1516 с.

— Peter Eeles Архітектура програмного забезпечення [Електронний ресурс] – Режим доступу: https://www.ibm.com/developerworks/ru/library/eeles/index.html

— Gennadii_M Тестирование. Фундаметальная теория. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/279535/

5 Вимоги до архітектури

5.1 Вимоги до характеристик архітектури:

— універсальність для усіх додатків для тестування програмного забезпечення;

— легкість розробки додатку;

— можливість легкого внесення змін;

— створення нових тестів без внесення змін до існуючих;

5.2 Вимоги до техніки безпеки та охорони праці

Під час розробки архітектури програми та її використання необхідно дотримуватися вимог, які вказані в нормативних документах щодо забезпечення охорони праці користувачів: “Правила охорони праці під час експлуатації електронно-обчислювальних машин”, “Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних

машин ДСанПіН 3.3.2.007-98”, затверджені Міністерством охорони здоров‘я від 10.12. 1998 року.

6 Стадії та етапи розробки

6.1 Робота виконується в шість етапів, наведених в таблиці 6.1

Таблиця 6.1 — Етапи виконання роботи

Етап	Зміст	Початок	Кінець	Результат
1	Інформаційний пошук та огляд літературних джерел.			Розділ 1
2	Розробка архітектури			Розділ 2
3	Розробка програмного засобу			Розділ 3
4	Опис результатів роботи ПЗ			Розділ 4
5	Економічна частина			Розділ 5
6	Підготовка матеріалів пояснювальної записки.			Пояснювальна записка.

7 Порядок контролю та прийому

7.1 До приймання дипломної роботи надається:

— пояснювальна записка з відповідними узгодженнями;

— демонстраційні матеріали;

— відгук керівника роботи;

— стороння рецензія.

7.2 Для проведення захисту магістерської дипломної роботи утворюється державна екзаменаційна комісія

Технічне завдання до виконання прийняв _____ Шевченко Андрій Ігорович

ДОДАТОК Б

Лістинг створеного програмного забезпечення

EnrollTest.cs

```
namespace CS.AutoTests.Tests
{
   [TestFixture]
   [Categories(Categories.Enroll)]
   public class EnrollTests : BaseTests<EnrollTestsHelper>
   {
      public const string DuplicateFederalIDMessageExpectedText = «The Federal ID entered matches an existing profile in CS-Safety.\r\nYou can continue to create the profile by clicking \»Continue\».\r\nOr click \»Cancel\» and retrieve your company login by contacting us at support@constructsecure.com or press \»Forgot Login\» on our home page www.constructsecure.com.»;

      private readonly int _enrollTestsCount = Settings.Default.EnrollTestsCount;

      private readonly Sub _DefaultOwnerName = new SubAlphabet_Inc();

      [Test]
      [Categories(Categories.SmokeTest)]
      [Categories(Categories.ServerDev)]
      public void AssertSuccessfulEnrollTitle()
      {
         Sub sub = new SubHarvard_University();
         Helper.WithReporting(hl => hl
            .EnrollIn(sub)
            .CheckPageIsEnroll());
```

```
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.NonStable)]
[Categories(Categories.SmokeTest)]
[Categories(Categories.ErrorsToVerify)]
public void AssertValidationOfContractorType()
{
   SubHarvard_University sub = new SubHarvard_University();
   Helper.WithReporting(hl => hl
      .EnrollIn(sub)
      .FillFields(sub)
      .ClearContractorType()
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(sub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfCompanyName()
{
   Sub currentSub;
   Helper.WithReporting(hl => hl
      .ClickRandomSubToEnroll(out currentSub)
      .FillFields(currentSub)
      .Execute(h => h.EnrollPage.TabContent.Name.Clear())
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(currentSub));
```

```
        }

        [Test]
        [Categories(Categories.NonStable)]
        [Categories(Categories.ServerDev)]
        [Categories(Categories.SmokeTest)]
        public void AssertValidationOfFederalEin1()
        {
            SubHarvard_University sub = new SubHarvard_University();
            Helper.WithReporting(hl => hl
                .EnrollIn(sub)
                .FillFields(sub)
                .Execute(h                                                    =>
h.EnrollPage.FederalEIN.FederalEINFields.ToList().ForEach(c => c.Clear()))
                .EnrollSubmit()
                .CheckUnsuccessfulSubmit(sub));
        }
        [Test]
        [Categories(Categories.ServerDev)]
        [Categories(Categories.SmokeTest)]
        [Categories(Categories.ErrorsToVerify)]
        public void AssertValidationOfTelephone()
        {
            SubHarvard_University sub = new SubHarvard_University();
            Helper.WithReporting(hl => hl
                .EnrollIn(sub)
                .FillFields(sub)
                .Execute(h => h.EnrollPage.TabContent.ContactsPhone.Clear())
                .EnrollSubmit()
                .CheckUnsuccessfulSubmit(sub));
```

```
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfAddress()
{
   Sub currentSub;
   Helper.WithReporting(hl => hl
      .ClickRandomSubToEnroll(out currentSub)
      .FillFields(currentSub)
      .Execute(h => h.EnrollPage.TabContent.ContactsAddress.Clear())
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfCity()
{
   SubHarvard_University sub = new SubHarvard_University();
   Helper.WithReporting(hl => hl
      .EnrollIn(sub)
      .FillFields(sub)
      .Execute(h => h.EnrollPage.TabContent.ContactsCity.Clear())
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(sub));
}
```

```
[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfYearlyRevenue()
{
   SubHarvard_University sub = new SubHarvard_University();
   Helper.WithReporting(hl => hl
      .EnrollIn(sub)
      .FillFields(sub)
      .Execute(h => h.EnrollPage.TabContent.YearlyRevenue.Clear())
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(sub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfState()
{
   SubHarvard_University sub = new SubHarvard_University();
   Helper.WithReporting(hl => hl
      .EnrollIn(sub)
      .FillFields(sub)
      .Execute(h => h.EnrollPage.TabContent.ContactsStateId.Clear())
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(sub));
}

[Test]
[Categories(Categories.ServerDev)]
```

```
[Categories(Categories.SmokeTest)]
public void AssertValidationOfZipCode()
{
   Sub currentSub;
   Helper.WithReporting(hl => hl
      .ClickRandomSubToEnroll(out currentSub)
      .FillFields(currentSub)
      .Execute(h => h.EnrollPage.TabContent.ContactsZipCode.Clear())
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfTradeDescription()
{
   SubWestfield_Construction sub = new SubWestfield_Construction();
   Helper.WithReporting(hl => hl
      .EnrollIn(sub)
      .FillFieldsWithoutTrades()
      .EnrollSubmit()
      .CheckUnsuccessfulSubmit(sub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfContacts0FirstName()
{
```

```
    Sub currentSub;
    Helper.WithReporting(hl => hl
        .ClickRandomSubToEnroll(out currentSub)
        .FillFields(currentSub)
        .Execute(h => h.EnrollPage.TabContent.Contacts0FirstName.Clear())
        .EnrollSubmit()
        .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.NonStable)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfContacts0LastName()
{
    Sub currentSub;
    Helper.WithReporting(hl => hl
        .ClickRandomSubToEnroll(out currentSub)
        .FillFields(currentSub)
        .Execute(h => h.EnrollPage.TabContent.Contacts0LastName.Clear())
        .EnrollSubmit()
        .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfContacts0Email()
{
    Sub currentSub;
```

```
    Helper.WithReporting(hl => hl
        .ClickRandomSubToEnroll(out currentSub)
        .FillFields(currentSub)
        .Execute(h => h.EnrollPage.TabContent.Contacts0Email.Clear())
        .EnrollSubmit()
        .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfContacts1FirstName()
{
    Sub currentSub;
    Helper.WithReporting(hl => hl
        .ClickRandomSubToEnroll(out currentSub)
        .FillFields(currentSub)
        .Execute(h => h.EnrollPage.TabContent.Contacts1FirstName.Clear())
        .EnrollSubmit()
        .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.ServerDev)]
[Categories(Categories.NonStable)]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfContacts1LastName()
{
    Sub currentSub;
    Helper.WithReporting(hl => hl
        .ClickRandomSubToEnroll(out currentSub)
```

```
        .FillFields(currentSub)
        .Execute(h => h.EnrollPage.TabContent.Contacts1LastName.Clear())
        .EnrollSubmit()
        .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.SmokeTest)]
public void AssertValidationOfContacts1Email()
{
    Sub currentSub;
    Helper.WithReporting(hl => hl
        .ClickRandomSubToEnroll(out currentSub)
        .FillFields(currentSub)
        .Execute(h => h.EnrollPage.TabContent.Contacts1Email.Clear())
        .EnrollSubmit()
        .CheckUnsuccessfulSubmit(currentSub));
}

[Test]
[Categories(Categories.SmokeTest)]
[Categories(Categories.ErrorsToVerify)]
public void AssertValidationOfCreditCardNumber()
{
    SubWestfield_Construction sub = new SubWestfield_Construction();
    Helper.WithReporting(hl => hl
        .EnrollIn(sub)
        .FillFields(sub)
        .EnrollSubmit()
        .EnrollAccept()
```

```
            .FillFieldsOfCreditCard(random:true, needToSubmit:false)
            .Execute(h => h.EnrollPage.CreditCardForm.CardNumber.Clear())
            .EnrollSubmit()
            .CheckUnsuccessfulSubmit(sub));
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        [Categories(Categories.ErrorsToVerify)]
        public void AssertValidationOfCreditCardCvv2()
        {
            SubHarvard_University sub = new SubHarvard_University();
            Helper.WithReporting(hl => hl
                .EnrollIn(sub)
                .FillFields(sub)
                .Execute(h                                                    =>
h.EnrollPage.TabContent.YearlyRevenue.SelectByIndex(2))
                .Execute(h                                                    =>
h.EnrollPage.TabContent.SelectedTrades.SelectByIndex(1))
                .EnrollSubmit()
                .EnrollAccept()
                .FillFieldsOfCreditCard(random: true, needToSubmit: false)
                .Execute(h => h.EnrollPage.CreditCardForm.CVV2.Clear())
                .EnrollSubmit()
                .CheckUnsuccessfulSubmit(sub));
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        [Categories(Categories.ErrorsToVerify)]
```

```
        public void AssertValidationOfCreditCardExpirationDateMonth()
        {
            SubHarvard_University sub = new SubHarvard_University();
            Helper.WithReporting(hl => hl
                .EnrollIn(sub)
                .FillFields(sub)
                .Execute(h                                                    =>
h.EnrollPage.TabContent.YearlyRevenue.SelectByIndex(2))
                .Execute(h                                                    =>
h.EnrollPage.TabContent.SelectedTrades.SelectByIndex(1))
                .EnrollSubmit()
                .EnrollAccept()
                .FillFieldsOfCreditCard(random: true, needToSubmit: false)
                .Execute(h                                                    =>
h.EnrollPage.CreditCardForm.ExpirationDateMonth.SelectByIndex(0))
                .EnrollSubmit()
                .CheckUnsuccessfulSubmit(sub));
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        [Categories(Categories.ErrorsToVerify)]
        public void AssertValidationOfCreditCardExpirationDateYear()
        {
            SubHarvard_University sub = new SubHarvard_University();
            Helper.WithReporting(hl => hl
                .EnrollIn(sub)
                .FillFields(sub)
                .Execute(h                                                    =>
h.EnrollPage.TabContent.YearlyRevenue.SelectByIndex(2))
```

```
                .Execute(h => h.EnrollPage.TabContent.SelectedTrades.SelectByIndex(1))
                .EnrollSubmit()
                .EnrollAccept()
                .FillFieldsOfCreditCard(random: true, needToSubmit: false)
                .Execute(h => h.EnrollPage.CreditCardForm.ExpirationDateYear.Clear())
                .EnrollSubmit()
                .CheckUnsuccessfulSubmit(sub));
            }

            [Test]
            [Categories(Categories.SmokeTest)]
            [Categories(Categories.ErrorsToVerify)]
            public void AssertValidationOfCreditCardNameOnCard()
            {
               SubHarvard_University sub = new SubHarvard_University();
               Helper.WithReporting(hl => hl
                  .EnrollIn(sub)
                  .FillFields(sub)
                  .Execute(h => h.EnrollPage.TabContent.YearlyRevenue.SelectByIndex(2))
                  .Execute(h => h.EnrollPage.TabContent.SelectedTrades.SelectByIndex(1))
                  .EnrollSubmit()
                  .EnrollAccept()
                  .FillFieldsOfCreditCard(random: true, needToSubmit: false)
                  .Execute(h => h.EnrollPage.CreditCardForm.NameOnCard.Clear())
                  .EnrollSubmit()
                  .CheckUnsuccessfulSubmit(sub));
```

```
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        public void VerifyDuplicateFederalEINinfoMessage()
        {
            Sub subStantec = new SubAllan_Myers() { FederalEIN =
Helper.GetContractorFederalEIN()};
            Helper.WithReporting(h1 => h1
                .EnrollIn(subStantec)
                .FillFields(subStantec)
                .EnrollSubmit()
                .VerifyMessageBoxText(DuplicateFederalIDMessageExpectedText));
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        public void VerifyNoDuplicateFederalEINinfoMessageForNOTPaidSub()
        {
            Sub sub = new SubMWH_Constructors_SPGC() { FederalEIN =
Helper.GetNotPaidContractorFederalEIN() };
            Helper.WithReporting(h1 => h1
                .CreateNewSub(sub)
                .VerifySubIsAvailibleOnVC(sub)
                .RemoveSub(sub));
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        public void AssertSuccessfulEnroll()
```

```
        {
            Sub currentSub;
            Helper.WithReporting(hl => hl
                .CreateNewSub(out currentSub)
                .VerifySubIsAvailibleOnVC(currentSub)
                .RemoveSub(currentSub));
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        public                                                    void
AssertSuccessfulEnrollDeleteSubWithSpecialCharactersInCompanyName()
        {
            Sub currentSub = new SubIntel_Supplier() { CompanyName =
this.GenerateRandom(true, true, true, true, -1, «S'p'e'c'i'a'l'C'h'r_» )};
            Helper.WithReporting(hl => hl
                .CreateNewSub(currentSub)
                .VerifySubIsAvailibleOnVC(currentSub)
                .RemoveSub(currentSub));
        }

        [Test]
        [Categories(Categories.SmokeTest)]
        public void AssertSuccessfulDeleteFromArhive()
        {
            Sub currentSub;
            Helper.WithReporting(hl => hl
                .CreateNewSub(out currentSub)
                .VerifySubIsAvailibleOnVC(currentSub)
                .RemoveSubFromVC(currentSub)
```

```
                .VerifySubIsArchived(currentSub)
                .RemoveSubFromARCHIVE(currentSub)
                .LoginAsAdministrator()
                .VerifySubIsArchived(currentSub,
expectedAvailableOnArchive:false));
            }

            [Test]
            [Categories(Categories.EnrollNewSubUnderEachOwner)]
            [TestCaseSource(typeof(AllOwnersList),
«AllOwnersList_DataSource_Independent»)]
            public void EnrollNewSubUnderEachOwner(Sub currentSub)
            {
               Helper.WithReporting(h => h
                  .CreateNewSub(currentSub)
                  .VerifySubIsAvailibleOnVC(currentSub)
                  .RemoveSubFromVC(currentSub)
                  .VerifySubIsArchived(currentSub)
                  .RemoveSubFromARCHIVE(currentSub)
                  .LoginAs(currentSub, skipLoginVerify: true)
                  .CheckUnSuccessfullLogin());
            }
            [Test]
            [Categories(Categories.SmokeTest)]
            public void DeleteNonPaidSub()
            {
               Sub currentPaidSub = null, currentNotPaidSub;
               Helper.WithReporting(h =>
               {
                  try
```

```
                {
                    h
                        .CreateNewSub(out currentPaidSub)
                        .ClickRandomSubToEnroll(out currentNotPaidSub,
PaymentNeeded: true);
                    currentNotPaidSub.FederalEIN = currentPaidSub.FederalEIN;
                    h
                        .FillSubDataToEnroll(currentNotPaidSub,
CloseDuplicateWindow: true, NeedToPay: false)
                        .VerifySubIsAvailibleOnVC(currentNotPaidSub, true,
TypeOfSubsRow.NotPaid, NeedToRefreshNotPaid: true)
                        .RemoveSubFromVC(currentNotPaidSub,
TypeOfSubsRow.NotPaid, false)
                        .VerifySubIsAvailibleOnVC(currentNotPaidSub, false,
TypeOfSubsRow.NotPaid, false)
                        .VerifySubIsAvailibleOnVC(currentPaidSub, NeedToLogin:
false);
                }
                catch (Exception e)
                {
                    h.RemoveSub(currentPaidSub);
                    throw e;
                }
            });
        }
    }
}
```

ДОДАТОК В

Лістинг створеного програмного забезпечення (продовження)

EnrollTestHelper.cs

```
using CS.AutoTests.Components;
using CS.AutoTests.Elements;
using CS.AutoTests.Extensions;
using CS.AutoTests.Framework;
using CS.AutoTests.Models;
using CS.AutoTests.Models.Subs;
using CS.AutoTests.Pages.Enroll;
using CS.AutoTests.Pages.Profile;
using CS.AutoTests.Pages.ViewContractors;
using CS.AutoTests.Tests;
using HtmlElements.Elements;
using HtmlElements.Extensions;
using OpenQA.Selenium;
using OpenQA.Selenium.Interactions;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace CS.AutoTests.TestHelpers
{
   public class EnrollTestsHelper : LoginTestsHelper
   {
      public List<string> OwnersToEnroll;
      public EnrollPage EnrollPage => PageFactory.Create<EnrollPage>(WebDriver);
      public FinancialPage FinancialPage =>
PageFactory.Create<FinancialPage>(WebDriver);
      public SafetyPage SafetyPage => PageFactory.Create<SafetyPage>(WebDriver);
      public ViewContractorsPage ViewContractorsPage =>
PageFactory.Create<ViewContractorsPage>(WebDriver);
      public EnrollTestsHelper ClickRandomSubToEnroll(out Sub sub)
      {
         this.GoToLoginPage();
         ClickEnrollMoreOwners();
         int index = RandomGenerator.GetNumber(0,
LoginPage.EnrollPopOver.OwnerLinks.Count);
```

```
            sub =
AllOwnersList.GetSubObjectByName(LoginPage.EnrollPopOver.OwnerLinks[index].T
ext.Clone().ToString());
            LoginPage.ClickOwnerLink(sub);
            Log.Info($»EnrollIn: Owner [{sub.GetOwnerName()}] «);
            return this;
        }
        public EnrollTestsHelper ClickRandomSubToEnroll(out Sub sub, bool
PaymentNeeded)
        {
            this.GoToLoginPage();
            ClickEnrollMoreOwners();
            do
            {
                int index = RandomGenerator.GetNumber(0,
LoginPage.EnrollPopOver.OwnerLinks.Count);
                sub =
AllOwnersList.GetSubObjectByName(LoginPage.EnrollPopOver.OwnerLinks[index].T
ext.Clone().ToString());
            } while (sub.IsPaymentSupported != PaymentNeeded);
            this.STEP($»Choosen a random owner: {sub.GetOwnerName()},
IsPaymentSupported={sub.IsPaymentSupported} (expected payment:
{PaymentNeeded})»);
            LoginPage.ClickOwnerLink(sub);
            Log.Info($»ClickRandomSubToEnroll: Owner [{sub.GetOwnerName()}] «);
            return this;
        }
        public EnrollTestsHelper EnrollClickByRandomSub()
        {
            Sub sub;
            ClickRandomSubToEnroll(out sub);
            return this;
        }

        public EnrollTestsHelper GetContractorFederalEIN(out string FederalEIN, string
ContractorId = null, ServerType st = ServerType.QA, TypeOfSubsRow rowType =
TypeOfSubsRow.All)
        {
            string tmp = ««;
            this.LoginAsAdministrator();
            switch (rowType)
            {
                case TypeOfSubsRow.NotPaid:
                    ViewContractorsPage.Navigation.IsNotPaid.WaitForVisible().TryClick();
```

```
                WaitUntilJQueryNonActive();
                break;
            }
            if (String.IsNullOrEmpty(ContractorId))
                ContractorId =
ViewContractorsPage.ContractorsGrid.Rows[RandomGenerator.GetNumber(0,
ViewContractorsPage.ContractorsGrid.Rows.Count - 1)].ContractorId;
            var url = Components.Configuration.getServerUrl(st) +
«/contractor/companyinformation/» + ContractorId;
            this.STEP($»Go to URL: '{url}'«);
            WebDriver.Navigate().GoToUrl(url);
            EnrollPage.FederalEIN.FederalEINFields.ToList().ForEach(c => tmp +=
c.Value);
            FederalEIN = tmp;
            if (String.IsNullOrEmpty(FederalEIN))
                Log.Warn($»Federal EIN is empty: '{FederalEIN}'«);
            else
                Log.Info($»Got Federal EIN: '{FederalEIN}'«);
            return this;
        }

        public string GetNotPaidContractorFederalEIN()
        {
            string FederalEIN = null;
            do
            {
                GetContractorFederalEIN(out FederalEIN, rowType:
TypeOfSubsRow.NotPaid);
            } while (String.IsNullOrEmpty(FederalEIN));
            return FederalEIN;
        }


        public string GetContractorFederalEIN()
        {
            string FederalEIN = null;
            do
            {
                GetContractorFederalEIN(out FederalEIN);
            } while (String.IsNullOrEmpty(FederalEIN));
            return FederalEIN;
        }
        public EnrollTestsHelper FillFederalEINField(Sub currentSub)
        {
```

```
            if (currentSub.IsUScountry)
            {
                List<string> tmpList = new List<string>();
                for (int i = 0; i < currentSub.FederalEIN.Length; i++)
                    tmpList.Add(currentSub.FederalEIN[i].ToString());
                int numElement = 0;
                tmpList.ForEach(value =>
                {

EnrollPage.FederalEIN.FederalEINFields.ToList()[numElement].SendKeys(value);
                    numElement++;
                });
                this.STEP($»Typing into field Federal EIN value = [ {String.Join(« «,
tmpList)} ]»);
            }
            return this;
        }
        public EnrollTestsHelper CreateNewSub(out Sub currentSub)
        {
            this.STEP(«Enroll new sub»);
            ClickRandomSubToEnroll(out currentSub);
            FillSubDataToEnroll(currentSub);
            return this;
        }
        /// <summary>
        /// Creates new sub
        /// </summary>
        public EnrollTestsHelper CreateNewSub(Sub sub)
        {
            this.STEP($»EnrollNewSub: Enrolling new sub [{ sub.GetOwnerName() }]»);
            EnrollIn(sub);
            FillSubDataToEnroll(sub);
            return this;
        }
        public EnrollTestsHelper FillSubDataToEnroll(Sub currentSub, bool
CloseDuplicateWindow = false, bool NeedToPay = true)
        {
            FillFields(currentSub);
            EnrollSubmit();
            if (CloseDuplicateWindow)
            {

VerifyMessageBoxText(EnrollTests.DuplicateFederalIDMessageExpectedText);
                BasePage.MessageBox.ButtonOk.Click();
```

```
        Thread.Sleep(2000);
        EnrollSubmit();
        WaitUntilJQueryNonActive();
        // close windo
    }
    if (NeedToPay)
    {
        EnrollAccept(currentSub);
        FillPaymentsFields(currentSub);
        TryCreatePassword(currentSub);
    }
    return this;
}

public EnrollTestsHelper EnrollIn(Sub currentSub)
{
    this.STEP($»EnrollIn: Click to Enroll [{ currentSub.GetOwnerName() }]»);
    this.GoToLoginPage()
        .ClickEnrollMoreOwners()
        .WaitUntilJQueryNonActive();
    LoginPage.ClickOwnerLink(currentSub);
    return this;
}
public EnrollTestsHelper ClearContractorType()
{
    var tab = EnrollPage.WaitFor(p => p.TabContent);
    tab.ClearContractorType();
    return this;
}
public EnrollTestsHelper FillFields(Sub currentSub = null)
{
    this.STEP($»FillFields: for new sub [{currentSub.GetOwnerName()}]»);
    var tab = EnrollPage.WaitFor(p => p.TabContent, TimeSpan.FromSeconds(20));

    FillFieldsWithoutTrades(currentSub);

    var countSelected = RandomGenerator.GetNumber(2, 5);
    var listOfIndexes = new List<int>();
    scrollIntoView(tab.SelectedTrades);
    for (var i = 0; i < countSelected; i++)
    {
        var index = RandomGenerator.GetNumber(0, tab.SelectedTrades.Count);

        Log.Info(«Choosed trade number: « + index);
```

```
            if (listOfIndexes.Contains(index))
            {
               Log.Info(«Trade was selected already, try again: « + index);
               i--;
               continue;
            }
            try
            {
               tab.SelectedTrades.SelectByIndex(index);
               listOfIndexes.Add(index);
            }
            catch (Exception ex)
            {
               Log.Info($»Got Exception during tab.SelectedTrades.SelectByIndex: [{
ex.Message }]»);
               Log.Info($»tab.SelectedTradesXP.IsPresent() = [{
tab.SelectedTradesXP.IsPresent() }]»);
               Log.Info($»tab.SelectedTradesXP.IsHidden() = [{
tab.SelectedTradesXP.IsHidden() }]»);
               Log.Info($»tab.SelectedTradesXP.Disabled = [{
tab.SelectedTradesXP.Disabled }]»);
               Log.Info($»tab.SelectedTradesXP.Displayed = [{
tab.SelectedTradesXP.Displayed }]»);
               Log.Info($»tab.SelectedTrades.IsPresent() = [{
tab.SelectedTrades.IsPresent() }]»);
               Log.Info($»tab.SelectedTrades.IsHidden() = [{
tab.SelectedTrades.IsHidden() }]»);
               Log.Info($»tab.SelectedTrades.Disabled = [{ tab.SelectedTrades.Disabled
}]»);
               Log.Info($»tab.SelectedTrades.Displayed = [{
tab.SelectedTrades.Displayed }]»);

               if (tab.SelectedTrades.OptionFirst.IsHidden())
               {
                  (new
Actions(WebDriver)).MoveToElement(tab.SelectedTrades).Click().Perform();
               }
               tab.SelectedTrades.OptionFirst.WaitForVisible().Click();
               tab.SelectedTrades.Click();
               break;
            }
            this.STEP($»SelectedTrades: index = [{ index }]»);//
         }
         tab.SelectedTrades.Click();
```

```
        Log.Info($»Finished filling: CompanyName = [{
EnrollPage.TabContent.Name.Value }]»);
        return this;
    }

    public EnrollTestsHelper FillFieldsWithoutTrades(Sub currentSub = null)
    {
        if (currentSub == null)
            currentSub = new Sub();
        Log.Info(«* Start Fill Tab Content»);
        var tab = EnrollPage.WaitFor(p => p.TabContent, TimeSpan.FromSeconds(20));
        this.STEP($»ButtonNext.TryClick(): [{ tab.ButtonNext.TryClick() }]»);
        this.STEP($»ChooseContractorType(): [{ tab.ChooseContractorType() }]»);
        if (!currentSub.IsUScountry)
        {
            EnrollPage.TabContent.Country.SelectRandom(minItem: 2);
            EnrollPage.TabContent.ContactsStateId.SelectRandom();
        }
        else
        {

EnrollPage.TabContent.ContactsStateId.SelectRandom(Enum.GetNames(typeof(StateO
fIncorporation)).Length, 1);
        }
        if (currentSub.ownersList != null)
        {

            EnrollPage.TabContent.OwnerAndGcList.BtnMore.Click();
            //foreach (var owner in currentSub.ownersList)
            foreach (var ownerName in currentSub.ownersList)
            {
                try
                {
                    var t = AllOwnersList.GetSubObjectByName(ownerName.ToString());
                    EnrollPage.TabContent.OwnerAndGcList.OwnersList.FirstOrDefault(
                     o => o.Text == t.GetOwnerName()
                        ).TryClick();
                }
                catch
                {

                }
            };
        }
```

```
Input(EnrollPage.TabContent.Name.WaitForPresent(TimeSpan.FromSeconds(30)),
currentSub.CompanyName);
            FillFederalEINField(currentSub);
            Input(EnrollPage.TabContent.Contacts0Email, currentSub.Email);
            this.InputRandom(EnrollPage.TabContent.ContactsPhone, false, false, true,
false);
            this.InputRandom(EnrollPage.TabContent.ContactsAddress, true, true, true,
false);
            this.InputRandom(EnrollPage.TabContent.ContactsAddress2, true, true, true,
false);
            EnrollPage.TabContent.YearlyRevenue.SelectByIndex(3);
            this.InputRandom(EnrollPage.TabContent.ContactsCity, true, true, true, false);
            //this.SelectDropDownRandom(h => h.EnrollPage.TabContent.ContactsStateId,
Enum.GetNames(typeof(StateOfIncorporation)).Length);
            this.InputRandom(EnrollPage.TabContent.ContactsZipCode, true, true, true,
false);
            this.InputRandom(EnrollPage.TabContent.Website, true, true, true, false);
            this.InputRandom(EnrollPage.TabContent.Contacts0FirstName, true, true, true,
false);
            this.InputRandom(EnrollPage.TabContent.Contacts0LastName, true, true, true,
false);
            this.InputRandom(EnrollPage.TabContent.Contacts0Phone, true, true, true,
false);
            this.InputRandom(EnrollPage.TabContent.Contacts1FirstName, true, true, true,
false);
            this.InputRandom(EnrollPage.TabContent.Contacts1LastName, true, true, true,
false);
            this.InputRandomEmail(h => h.EnrollPage.TabContent.Contacts1Email, 15);
            this.InputRandom(EnrollPage.TabContent.Contacts1Phone, true, true, true,
false);
            Log.Info($»Finished filling: CompanyName = [{
EnrollPage.TabContent.Name.Value }]»);
            return this;
        }

        public List<OwnerName> GetAllOwners()
        {
            return Enum.GetValues(typeof(OwnerName)).Cast<OwnerName>().ToList();
        }
        public EnrollTestsHelper FillPaymentsFields(Sub sub = null)
        {
            this.STEP(«FillPaymentsFields: Fill payments fields»);
            if (sub == null || sub.IsPaymentSupported)
```

```
            FillFieldsOfCreditCard(needToSubmit: false);
        else
            Log.Warn(«Payment is not supported for this owner:
sub.IsPaymentSupported=« + sub.IsPaymentSupported);
        return this;
    }
    public EnrollTestsHelper FillFieldsOfCreditCard(bool random = false, bool
needToSubmit = false)
    {
        EnrollPage.WaitUntil(e => e.ButtonContinue.Displayed);
        if (!EnrollPage.CreditCardForm.Displayed)
            return this;
        this.STEP($»Select payment method 'Credit Card' =
{PaymentMethods.CreditCard.ToString()}»);

EnrollPage.PaymentMethodContainer.SetPaymentMethod(PaymentMethods.CreditCard
);
        // this.Execute(p =>
p.EnrollPage.SelectedPaymentMethod.SelectByIndex((int)PaymentMethods.CreditCard
));
        this.STEP(«Filling: CreditCardForm «);
        this.InputRandom(EnrollPage.CreditCardForm.NameOnCard, true, true, true,
false)
            .InputRandom(EnrollPage.CreditCardForm.CVV2, false, false, true, false)
            .InputRandom(EnrollPage.CreditCardForm.ExpirationDateYear, true, true,
true, false)
            .Execute(p =>

p.EnrollPage.CreditCardForm.ExpirationDateMonth.SelectByIndex(RandomGenerator.
GetNumber(1, 12)));
        if (random)
        {
            this.InputRandom(EnrollPage.CreditCardForm.CardNumber, false, false, true,
false);
            // return this;
        }
        else
        {
            Input(EnrollPage.CreditCardForm.CardNumber, «4111-1111-1111-1111»);
            EnrollSubmit();
            EnrollPage.WaitFor(e =>
e.TestCardPass).WaitForVisible().SendKeys(«cS^41»);
            EnrollPage.MessageBox.ButtonOk.Click();
            EnrollPage.MessageBox.WaitUntilSoft(m => m.IsPresent());
```

```
            }
            if (needToSubmit)
                EnrollSubmit();
            return this;
        }

        public EnrollTestsHelper EnrollSubmit()
        {
            this.STEP(«Click to Submit button»);
            try
            {
                EnrollPage.SubmitContinue();
            }
            catch (ElementNotVisibleException)
            {
                EnrollPage.UiDialog.WaitUntil(d => d.Button.IsPresent(),
TimeSpan.FromSeconds(180));
                WaitUntilJQueryNonActive();
                EnrollPage.UiDialog.Button.WaitForVisible().Click();
                Log.Info(«Submit - on UiDialog»);
                Thread.Sleep(100);
                WaitUntilJQueryNonActive();
            }
            WaitUntilJQueryNonActive();
            if (EnrollPage.UiDialog.IsPresent() &&
EnrollPage.UiDialog.Content.TextContent.Contains(«error 500»))
            {
                Log.Error(«Server Error 500»);
                throw new Exception(«Server error 500»);
            }
            return this;
        }

        private EnrollTestsHelper VerifyInjuryTab_CONTROLS_GROUP3(TestManage
testManage = TestManage.Positive)
        {

            string negativeTestMessageText = «The EMR value cannot be greater than 2. If
you are entering a percentage value for example 108%, then please enter 1.08 in the
box»;
            string textToEnter = testManage == TestManage.Positive ? «1.1» : «22»;
            string baselineText = testManage == TestManage.Positive ? textToEnter :
negativeTestMessageText;
```

```
            List<HtmlCheckBox> hcbList = new List<HtmlCheckBox> {
                SafetyPage.InjuryContent.LostWorkDayCases,
                SafetyPage.InjuryContent.RestrictedWorkDayCases
            };

            hcbList.ForEach(h =>
            {
                h.WaitUntilSoft(c => c.ClickOrNull() != null);
                h.WaitForPresent().EnterText(textToEnter + Keys.Tab);
                string text = h.Value;
                Log.Info($»Entered text: '{textToEnter}'. Text from field: '{text}'«);
                Verify(text, baselineText, $»Entered text verification to: [{h.Text}] field»);
                h.Clear();
            });
            return this;
        }

        private EnrollTestsHelper
VerifyInjuryTab_ControlsWithMessageValidation(TestManage testManage =
TestManage.Positive, string negativeTestMessageText = ««, string textToEnter = ««,
string baselineText = ««, List<CustomHtmlInput> hcbList = null)
        {
            Log.Info($»Injury Tab: Controls input verification»);
            hcbList.ForEach(h =>
            {
                h.WaitUntilSoft(c => c.ClickOrNull() != null);
                h.WaitForPresent().EnterText(textToEnter + Keys.Tab);
                string text = testManage == TestManage.Positive ? h.Value :
EnrollPage.MessageBox.WaitForPresentSoft(TimeSpan.FromSeconds(2)).getTextAndC
lose();
                Log.Info($»Entered text: '{textToEnter}'. Text to Verify: '{text}'«);
                Verify(text, baselineText, $»Entered text verification to: [{h.Text}] field»);
                if (testManage == TestManage.Negative)
                    h.Clear();
            });
            return this;
        }

        public EnrollTestsHelper VerifyInjuryTab(TestManage testManage =
TestManage.Positive)
        {
            SafetyPage.SafetyProfileContent.Click();
            BasePage.Menu.Injury.WaitUntil(m => m.TryClick());
            // Set all years to «Yes»
```

```
            SafetyPage.InjuryContent.WaitForPresent().WorkThisYear.SetYears();

            SafetyPage.InjuryContent.WorkersCompensationData.EMR.ToList().ForEach(e
=> e.WaitUntil(c => !c.Disabled));

            string negativeTestMessageText = «Only whole numbers are allowed for these
fields»;
            string textToEnter = testManage == TestManage.Positive ? «22» : «1.1»;
            string baselineText = testManage == TestManage.Positive ? textToEnter :
negativeTestMessageText;

            List<CustomHtmlInput> hcbList = new List<CustomHtmlInput>();
            hcbList.AddRange(SafetyPage.InjuryContent.SafetyData.RecordableCases);
            hcbList.AddRange(SafetyPage.InjuryContent.SafetyData.DARTCases);
            //EnrollPage.InjuryContent.LostWorkDayCases,
            hcbList.AddRange(SafetyPage.InjuryContent.SafetyData.TotalDaysAway);
            //EnrollPage.InjuryContent.RestrictedWorkDayCases,
            hcbList.AddRange(SafetyPage.InjuryContent.SafetyData.TotalDaysOfJob);
            hcbList.AddRange(SafetyPage.InjuryContent.SafetyData.Fatalities);
            hcbList.AddRange(SafetyPage.InjuryContent.SafetyData.HoursWorked);

            VerifyInjuryTab_ControlsWithMessageValidation(testManage,
negativeTestMessageText, textToEnter, baselineText, hcbList);

            negativeTestMessageText = «The EMR value cannot be greater than 2. If you
are entering a percentage value for example 108%, then please enter 1.08 in the box»;
            textToEnter = testManage == TestManage.Positive ? «1.1» : «22»;
            baselineText = testManage == TestManage.Positive ? textToEnter :
negativeTestMessageText;

            hcbList = new List<CustomHtmlInput>();

hcbList.AddRange(SafetyPage.InjuryContent.WorkersCompensationData.EMR);
            VerifyInjuryTab_ControlsWithMessageValidation(testManage,
negativeTestMessageText, textToEnter, baselineText, hcbList);

            // VerifyInjuryTab_CONTROLS_GROUP3(testManage);
            return this;
        }

        public EnrollTestsHelper EnrollAccept(Sub sub = null)
        {
            this.STEP(«Click to Accept agreement button»);
            WaitUntilJQueryNonActive();
```

```
        if (LoginPage/*.waitForCaptcha()*/.CaptchaForm.IsPresent()) //
        {
           Log.Warn(«Captcha verificaion found. Further User Enrollment skipped.»);
        }
        else
        {
           bool ButtonAcceptClicked = false;
           EnrollPage.WaitUntilSoft(p => { ButtonAcceptClicked = p.TryAccept();
return ButtonAcceptClicked; }, TimeSpan.FromSeconds(5), $»Waiting for
'{EnrollPage.GetType().Name}.Accept' button to disappear»);
           int i = 0;
           while (!ButtonAcceptClicked && i < 4)
           {
              if (sub == null || sub.IsPaymentSupported)
              {
                 if
(!EnrollPage.CreditCardForm.WaitForPresentSoft(TimeSpan.FromSeconds(5)).Display
ed)
                 {
                    EnrollPage.WaitUntil(p => p.TryAccept(),
                       $»Waiting for '{EnrollPage.GetType().Name}.Accept' button to
disappear»);
                 }
                 else
                 {
                    ButtonAcceptClicked = true;
                    Log.Info(«'Accept' button clicked and Credit Card Form is
displayed»);
                 }
              }
              else
              {
                 if
(!EnrollPage.CreatePassword[0].WaitForPresentSoft(TimeSpan.FromSeconds(5)).Displ
ayed)
                 {
                    EnrollPage.WaitUntil(p => p.TryAccept(),
                       $»Waiting for '{EnrollPage.GetType().Name}.Accept' button to
disappear»);
                 }
                 else
                 {
                    ButtonAcceptClicked = true;
                    Log.Info(«'Accept' button clicked»);
```

```
                }
            }
            Thread.Sleep(1000);
            i++;
        }
        Verify(ButtonAcceptClicked, «Button accept is clicked»);
    }
    return this;
}

public EnrollTestsHelper CheckPageIsEnroll()
{
    if (!Verify(EnrollPage.Title.ToString().Contains(«ConstructSecure Inc.»), «Page
Title has 'ConstructSecure Inc.'«, true))
        Verify(EnrollPage.Title.ToString().Contains(«Nueva Contratista Sign-Up»),
«Page Title has 'Nueva Contratista Sign-Up'«);
    return this;
}

public EnrollTestsHelper TryCreatePassword(Sub currentSub)
{
    this.STEP($»Entering password={currentSub.Password}»);
    EnrollPage.CreatePassword[0].SendKeys(currentSub.Password);
    this.WaitUntil(c => EnrollPage.DoneMessage.Count != 0);
    if (!EnrollPage.DoneMessage[0].WaitForPresentSoft().IsPresent())
    {
        currentSub.Password = currentSub.Password + «0»;
        Log.Info($»password was changed to password={currentSub.Password}»);
        foreach (var tmp in EnrollPage.CreatePassword)
            tmp.EnterText(currentSub.Password);
    }
    else
        EnrollPage.CreatePassword[1].EnterText(currentSub.Password);
    this.WaitUntil(c => EnrollPage.DoneMessage.Count > 1);
    EnrollPage.ButtonContinue.Click();
    try
    {
        SafetyPage.SafetyProfileContent.WaitForPresent();
    }
    catch
    {
        this.LoginAsAdministrator()
            .GoToQuickSearchPage()
```

```
                .SetPasswordByCompanyName(UsersType.Contractor,
currentSub.CompanyName, currentSub.Password)
                .LoginAs(currentSub.Email, currentSub.Password);
        }
        return this;
    }

    public EnrollTestsHelper CheckContentOfUiDialog(List<string> content)
    {

        var text =
LoginPage.MessageBox.WaitForPresent().MessageBoxContent.WaitForPresent().Text;

        string messageText = «<Unable to match proper text in list>«;
        if (content.Contains(text))
        {
            messageText = content.Find(x => x == text);
        }
        Verify(messageText, text, $»Validate text on Message Box: [{text}]»);
        return this;
    }

    public EnrollTestsHelper CheckUnsuccessfulSubmit(Sub currentSub)
    {
        WaitUntilJQueryNonActive();
        if (!EnrollPage.TryWaitUntil(c => c.MessageBox.ButtonOk.Displayed))
        {
            EnrollPage.WrappedDriver.Navigate().Back();
            throw new Exception($»Validation of unsuccessful submit for Owner: {
currentSub.GetOwnerName()} FAILED. Message didn't appear.»);
        }
        CheckContentOfUiDialog(new List<string>
            {
                «error 500»,
                «Please fill in all required data shaded in red»,
                «Please fill in all required fields.»
                // «Por favor rellene todos los campos requeridos»
            });

        Log.Info(«Ok»);
        return this;
    }

    public EnrollTestsHelper ClickEnrollMoreOwners()
```

```
        {
            WaitUntilJQueryNonActive();
            LoginPage.EnrollButton.WaitUntil(b =>
b.WaitForPresent().WaitForVisible().TryClick());
            LoginPage.WaitFor(l => l.EnrollPopOver.BtnMoreOwners).TryWaitUntil(b =>
b.WaitForVisible().TryClick());
            WaitUntilJQueryNonActive();
            OwnersToEnroll = LoginPage.EnrollPopOver.OwnerLinks.Select(s =>
s.Text).ToList();
            return this;
        }
        public string GetContractorIdByCompanyName(string companyName)
        {
            try
            {
                ViewContractorsPage.ContractorsGrid.WaitUntilSoft(r => r.Rows.Count >
0);
                return ViewContractorsPage.ContractorsGrid.Rows.FirstOrDefault(f =>
f.WaitForVisible().CompanyName.WaitForPresent().Text ==
companyName).ContractorId; //
            }
            catch { return null; }
        }

        public EnrollTestsHelper VerifySubIsAvailibleOnVC(Sub currentSub, bool
expectedAvailableOnVC = true, TypeOfSubsRow type = TypeOfSubsRow.All, bool
NeedToLogin = true, bool NeedToRefreshNotPaid = false)
        {
            this.STEP($»Verify sub [{currentSub.CompanyName}]  is available on VC
({type})»);
            if (NeedToLogin)
                this.LoginAsAdministrator();
            ClickOnNavigation(type);
            if (type == TypeOfSubsRow.NotPaid && NeedToRefreshNotPaid)
            {
                Thread.Sleep(2000);

ViewContractorsPage.UpdateNotPaidVC.WaitForPresent().WaitForVisible().Click();
                WaitUntilJQueryNonActive();
                Thread.Sleep(2000);
            }
            string verifyMessageText = expectedAvailableOnVC == true ? «available» :
«not available»;
```

```
Verify(!string.IsNullOrEmpty(GetContractorIdByCompanyName(currentSub.Company
Name)), expectedAvailableOnVC, $»Contractor is {verifyMessageText} on VC»);
            return this;
        }
        public EnrollTestsHelper VerifySubIsArchived(Sub currentSub, bool
expectedAvailableOnArchive = true)
        {
            this.STEP(«Verify sub is available on archive page»);
            ViewContractorsPage.Navigation.Archived.ClickAgain();
            var row = WaitForViewContractors()
                .ContractorsGrid.Rows.FirstOrDefault(r =>
r.CompanyName.Text.Equals(currentSub.CompanyName));
            string verifyMessageText = expectedAvailableOnArchive == true ? «available»
: «not available»;
            Verify(row != null, expectedAvailableOnArchive, $»Contractor
{currentSub.CompanyName} is {verifyMessageText} on Archive View»);
            return this;
        }
        public EnrollTestsHelper RemoveSubFromVC(Sub currentSub, TypeOfSubsRow
type = TypeOfSubsRow.All, bool NeedToLogin = false)
        {
            this.STEP($»Removing sub from VC ({type}), Company Name: [{
currentSub.CompanyName }]»);
            if (NeedToLogin)
                this.LoginAsAdministrator();
            ClickOnNavigation(type);
            var row = WaitForViewContractors()
                .ContractorsGrid.Rows.FirstOrDefault(r =>
r.CompanyName.Text.Equals(currentSub.CompanyName));
            row.DeleteButton.Click();
            ViewContractorsPage.MessageBox.ButtonYes.WaitForVisible().Click();
            ViewContractorsPage.WaitUntil(p => p.ContractorsGrid.Displayed);
            ViewContractorsPage.MessageBox.WaitUntil(mb => mb.Backdrop.IsHidden());
            return this;
        }
        public EnrollTestsHelper RestoreSubFromARCHIVE(Sub currentSub)
        {
            this.STEP($»Restoring sub from ARHIVE, Company Name: [{
currentSub.CompanyName }]»);
            var row = WaitForViewContractors()
                .ContractorsGrid.Rows.FirstOrDefault(r =>
r.CompanyName.Text.Equals(currentSub.CompanyName));
            ViewContractorsPage.Navigation.Archived.ClickAgain();
```

```
            row.RestoreButton.Click();
            ViewContractorsPage.MessageBox.ButtonYes.WaitForVisible().Click();
            ViewContractorsPage.WaitUntil(p => p.ContractorsGrid.Displayed);
            ViewContractorsPage.MessageBox.WaitUntil(mb => mb.Backdrop.IsHidden());
            return this;
        }
        private void ExecuteRemoveSub()
        {
            try { SafetyPage.SafetyProfileContent.DeleteProfileButton.WaitForPresent(); }
            catch { SafetyPage.SafetyProfileContent.Click(); }
            finally {
SafetyPage.SafetyProfileContent.DeleteProfileButton.WaitForPresent().Click(); }
            BasePage.MessageBox.WaitForPresent().ButtonOk.WaitForPresent().Click();
            VerifyMessageBoxText(«Your account was deleted.»);
            BasePage.MessageBox.ButtonOk.Click();
        }
        public EnrollTestsHelper RemoveSub(Sub currentSub, bool logingNeeded = true)
        {
            this.STEP($»Removing sub, Company Name: [{ currentSub.CompanyName
}]»);
            if (logingNeeded)
                this.LoginAs(currentSub);
            ExecuteRemoveSub();
            return this;
        }
        public EnrollTestsHelper RemoveSubFromARCHIVE(Sub currentSub)
        {
            this.STEP($»Removing sub, from Archive view - Company Name: [{
currentSub.CompanyName }]»);
            this.LoginAsAdministrator();
            ViewContractorsPage.Navigation.Archived.ClickAgain();
            WaitForViewContractors();

this.GoToProfileContractorsPage(GetContractorIdByCompanyName(currentSub.Comp
anyName));
            ExecuteRemoveSub();
            return this;
        }
        public ViewContractorsPage WaitForViewContractors()
        {
            var js = WebDriver as IJavaScriptExecutor;

            ViewContractorsPage.WaitFor(p => p.ContractorsGrid)
                .WaitForVisible()
```

```
            .TryWaitUntil(g => js.ExecuteScript(«return
jQuery.active»).ToString().Equals(«0»));

            return ViewContractorsPage;
        }
        public AccountInfo GetAccountInfo(Sub sub, UsersType ut = UsersType.All)
        {
            return googleSheet.getAccountsInfoList(ut: ut).FirstOrDefault(c =>
c.CompanyName == sub.GetOwnerName());
        }
        public AccountInfo GetAccountInfo(string companyName, UsersType ut =
UsersType.All)
        {
            return googleSheet.getAccountsInfoList(ut: ut).FirstOrDefault(c =>
c.CompanyName == companyName);
        }
        private void ClickOnNavigation(TypeOfSubsRow type)
        {
            switch (type)
            {
                case TypeOfSubsRow.NotPaid:
                    {
                        var t = ViewContractorsPage.Navigation.IsNotPaid;
                        t.WaitForVisible().TryClick();
                        t.WaitUntilActive();
                        break;
                    }
                case TypeOfSubsRow.All:
                    {
                        var t = ViewContractorsPage.Navigation.AllProfiles;
                        t.WaitForVisible().TryClick();
                        t.WaitUntilActive();
                        break;
                    }

            }
        }
    }
}
```

ДОДАТОК Г

Ілюстративний матеріал до захисту магістерської кваліфікаційної роботи

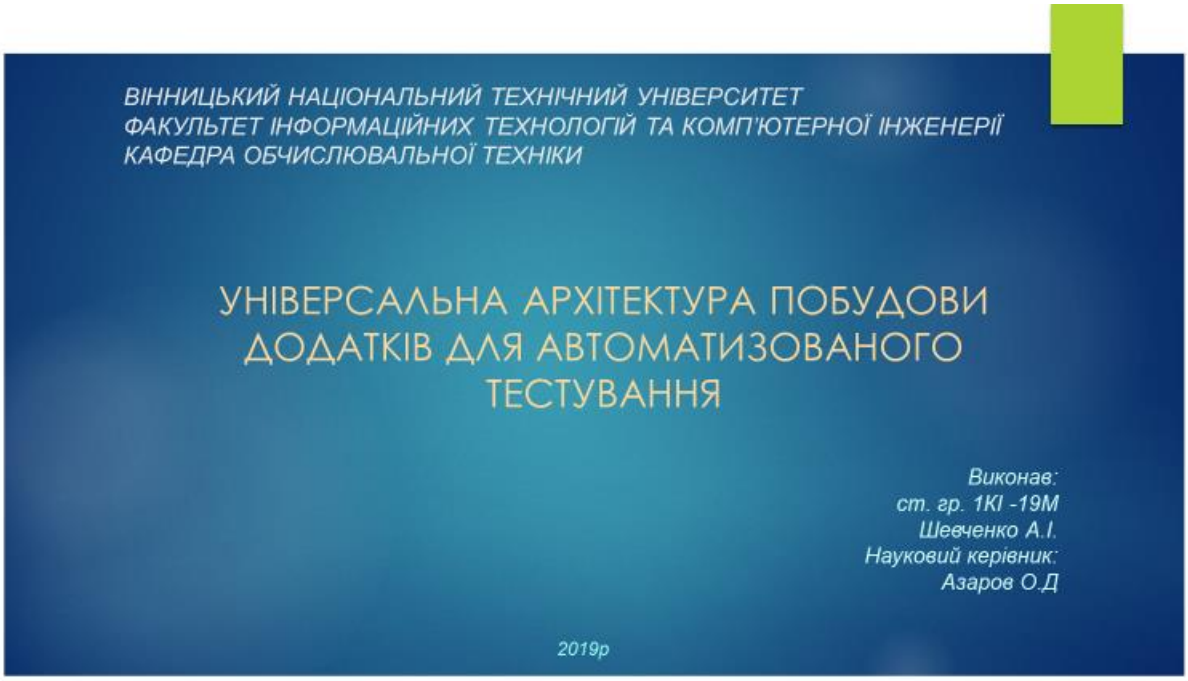


Рисунок Г.1 — Тема, автор, науковий керівник бакалаврської дипломної роботи

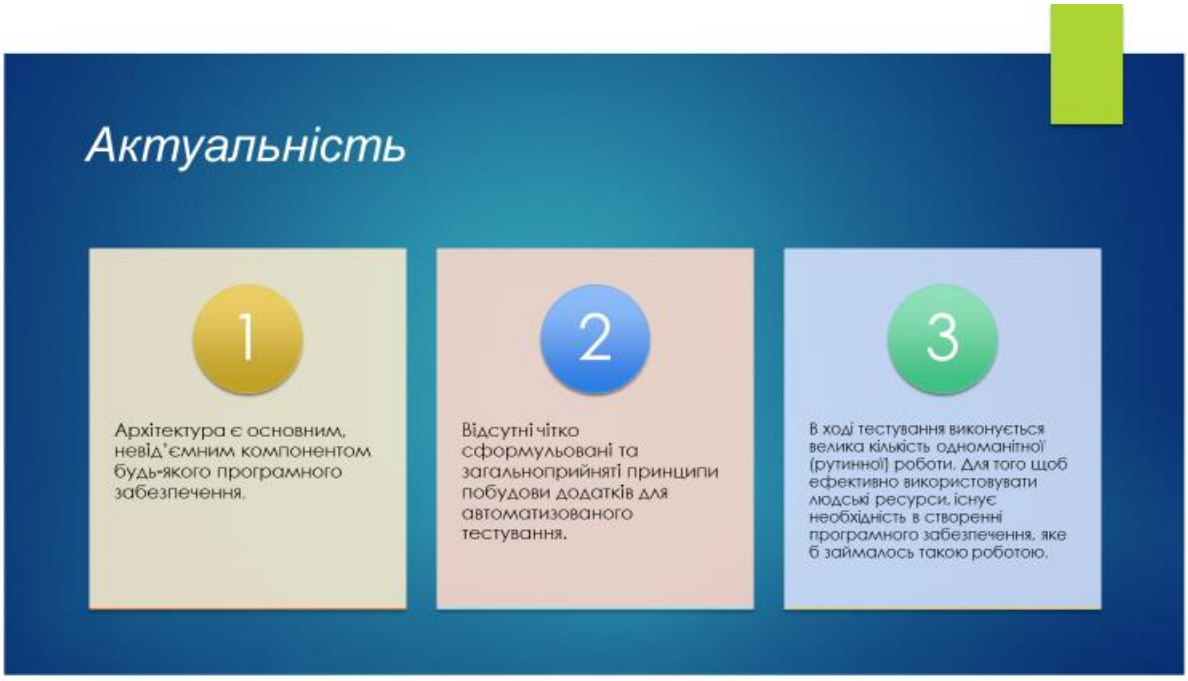


Рисунок Г.2 — Актуальність роботи

Рисунок Г.3 — Об'єкт дослідження, предмет дослідження, мета

Рисунок Г.4 — Вирішені задачі

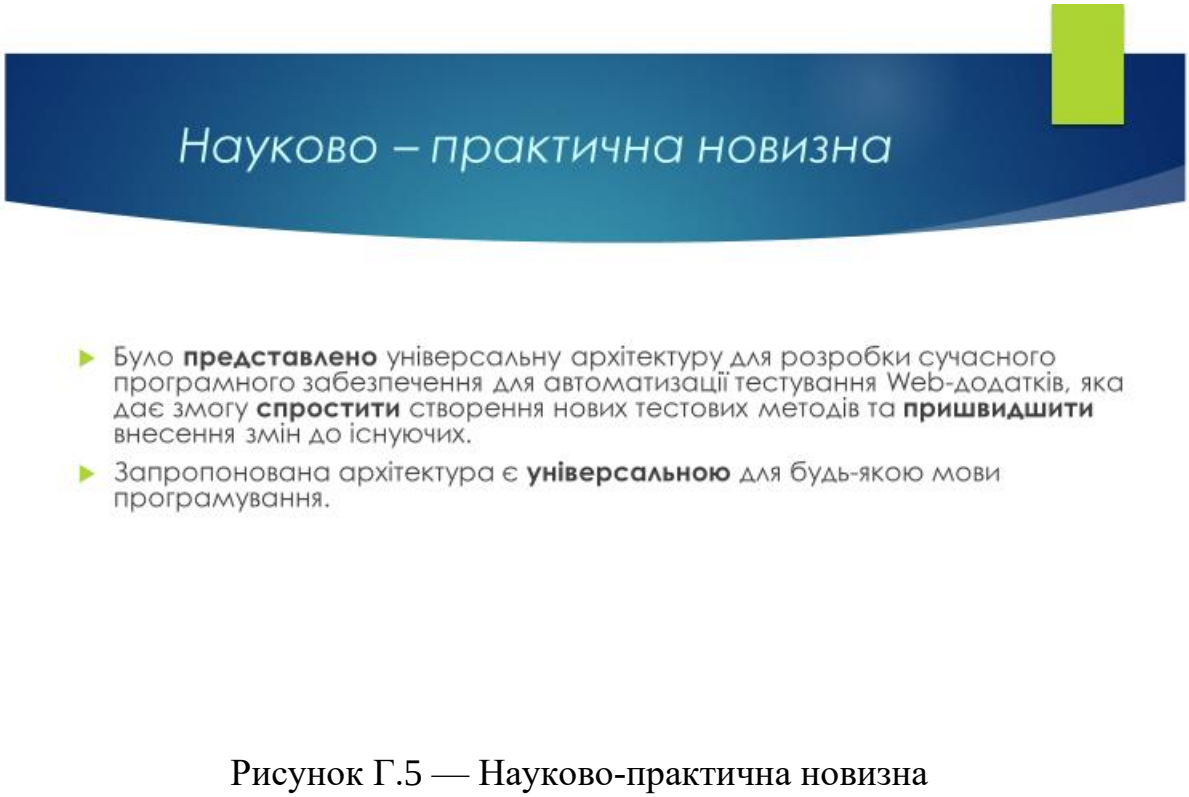


Рисунок Г.5 — Науково-практична новизна

Рисунок Г.6 — Засоби створення ПЗ

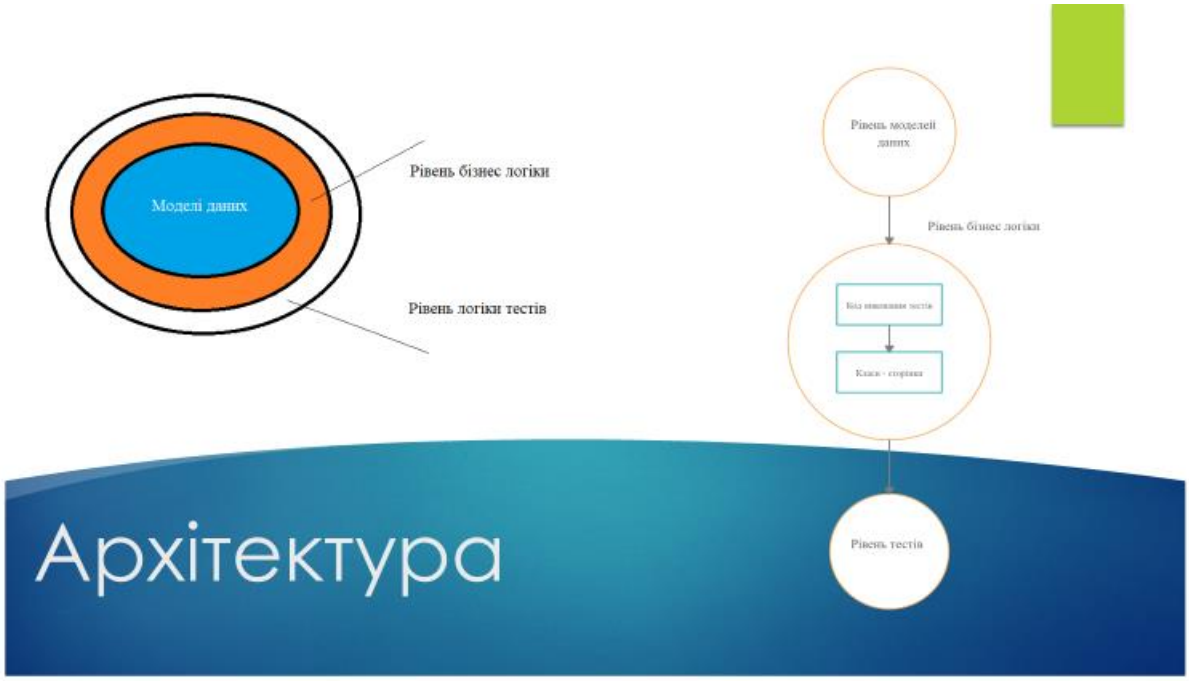


Рисунок Г.7 — Схема архітектури програмного забезпечення

Рисунок Г.8 — Опис архітектури

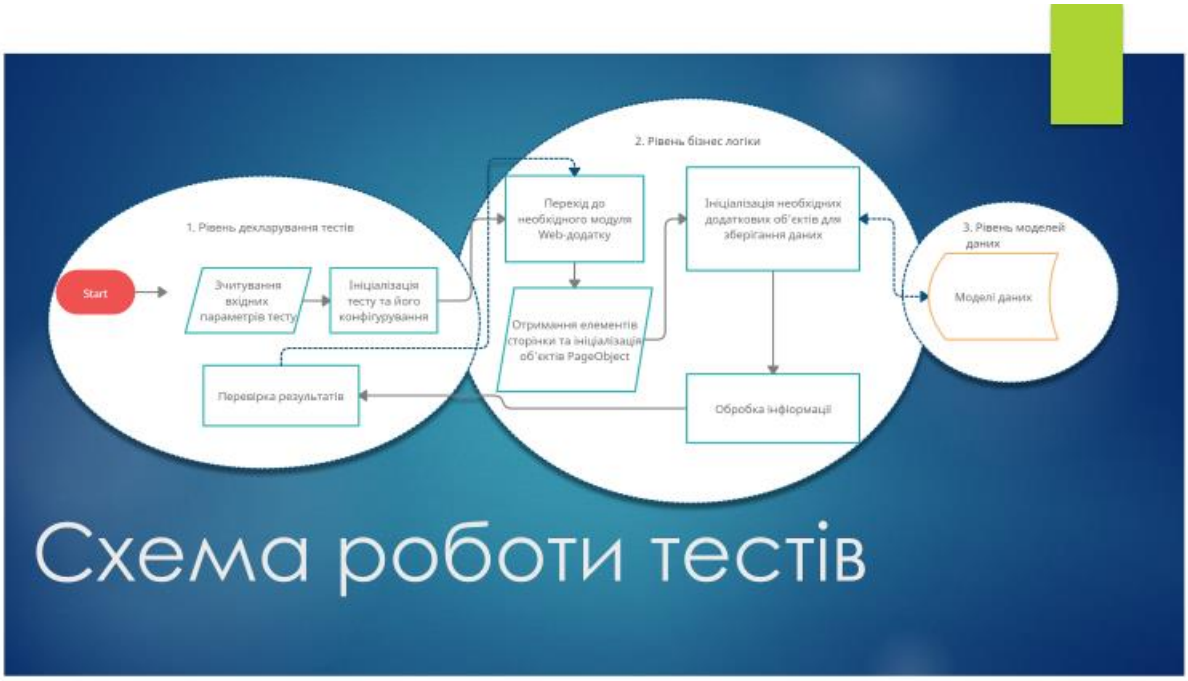


Рисунок Г.9 — Схема роботи тестів

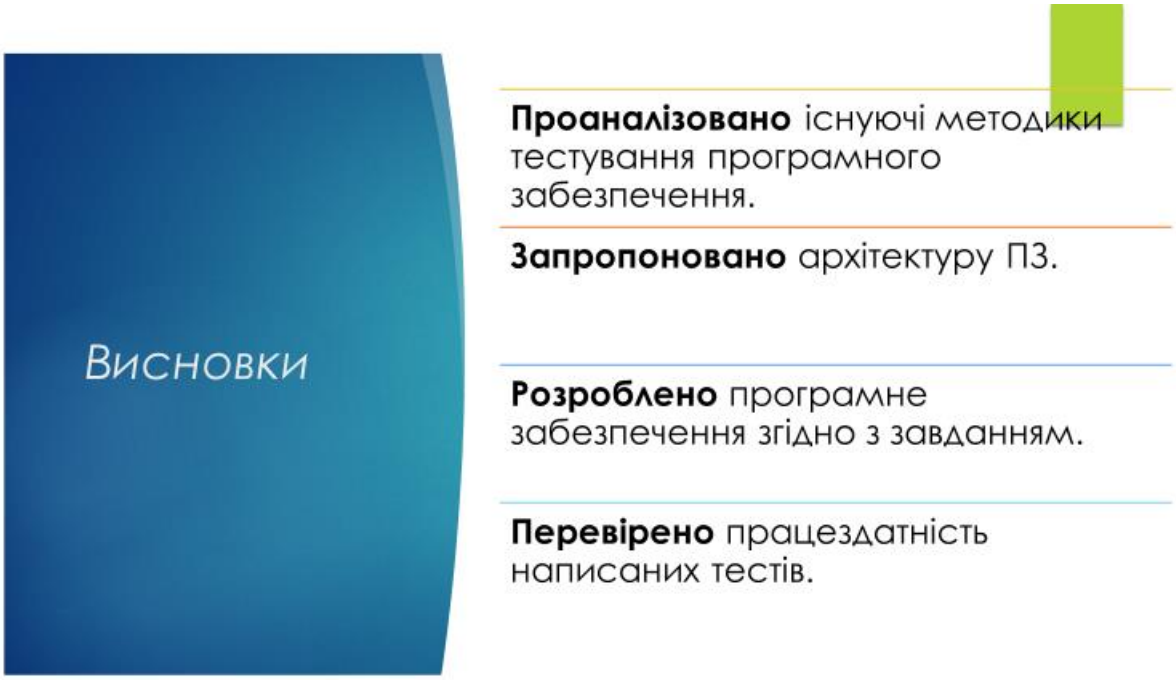


Рисунок Г.10 — Висновки

Завідувач кафедри ОТ, д.т.н., професор	_____	О. Д. Азаров
Керівник роботи, к.т.н., доцент	_____	О. Д. Азаров
Рецензент, к.т.н., зав.каф.	_____	В. В. Карпінець
Нормоконтроль, к. т. н., доцент	_____	С. І. Швець
Виконавець, студент групи 2КІ-15Б	_____	А.І. Шевченко