КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня магістра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

ВИКОРИСТАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ В ПРОЦЕСІ КОНТРОЛЮ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Виконала студентка 2-го курсу магістратури
Марія КУШНІРЕНКО

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Оксана ШКІЛЬНЯК

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 10 » травня 2023 р.,
протокол № 9
Завідувач кафедри
Олександр ПРОВОТАР

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 75 сторінок, 41 ілюстрація, 24 джерела посилання.

ALLURE, BDD, CUCUMBER, JENKINS, MAVEN, SELENIDE, ІНСТРУМЕНТ ТЕСТУВАННЯ, КОНТРОЛЬ ЯКОСТІ, ПЛАГІН, ФРЕЙМВОРК.

Об’єктом роботи є дослідження інструментальних засобів, що використовуються в процесі контролю якості програмного забезпечення. Предметом роботи є фреймворк автоматизації тестування з наступними інтеграціями: Cucumber, Selenide, AssertJ, Jenkins, Allure Report.

Метою роботи є створення власного комплексного фреймворку з автоматизації тестування з використанням вищезазначених інструментів.

Методи розроблення: створення комплексного рішення за допомогою використання популярних інструментів з метою створення цілісного продукту. Інструменти розроблення: інтегроване середовище розробки IntelliJ IDEA, мова програмування Java, засіб збірки Maven, інструмент, який підтримує розробку, орієнтовану на поведінку Cucumber, засіб автоматизації вебзастосунків Selenide, бібліотека тверджень AssertJ, інструмент CI/CD Jenkins, інструмент звітування про результати тестування Allure.

Результати роботи: розглянуто і проаналізовано типи інструментів автоматизації з прикладами, обрано найкращі варіанти для подальшого практичного використання в роботі, і з їх допомогою розроблений фреймворк для автоматизації тестування, продемонстрована його робота на прикладі автоматизації тестування базового функціоналу інтернет-магазину «ROZETKA».

Розроблений фреймворк для автоматизації тестування, може використовуватися як основа під час впровадження автоматизації тестування вебзастосунків на будь-якому реальному проєкті.

ЗМІСТ

РЕФЕРАТ.....2
ЗМІСТ.....3
СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ.....6
ВСТУП.....7
РОЗДІЛ 1. ПРОЦЕС КОНТРОЛЮ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....10
1.1 Різниця між забезпеченням та контролем якості (QA & QC).....10
1.2 Типи інструментів, що використовуються в процесі контролю якості програмного забезпечення.....12
1.2.1 Тестування вебдодатків.....12
1.2.2 Тестування мобільних додатків.....12
1.2.3 Тестування безпеки.....14
1.2.4 Автоматизація тестування інфраструктури.....15
1.2.5 Управління тестовими випадками.....16
1.2.6 Відстеження дефектів.....16
1.2.7 Звітування про автоматизоване тестування.....17
1.2.8 Тестування продуктивності та навантаження.....19
1.2.9 Полегшення читання та написання коду з автоматизації.....20
РОЗДІЛ 2. ОГЛЯД ІНСТРУМЕНТІВ, ЩО ВИКОРИСТОВУЮТЬСЯ В ПРОЦЕСІ КОНТРОЛЮ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....21
2.1 Теоретичні відомості про Cucumber.....22
2.1.1 Розробка, орієнтована на поведінку (BDD).....22
2.1.2 Синтаксичні правила Gherkin.....25

2.1.3 Визначення кроків..... 26
2.2 Теоретичні відомості про Selenide..... 27
2.2.1 Selenide та Selenium.....27
2.2.1.1 Особливості Selenium.....28
2.2.1.2 Особливості Selenide..... 30
2.2.2 Порівняння Selenium та Selenide на прикладі.....31
2.2.3 Вибір Selenide..... 31
2.3 Автоматизація в неперервній інтеграції за допомогою Jenkins.....32
2.3.1 Неперервна інтеграція.....32
2.3.2 Роль автоматизованих тестів у CI..... 33
2.3.3 Інструмент Jenkins..... 33
2.3.3.1 Переваги Jenkins..... 34
2.3.3.2 Недоліки Jenkins..... 35
2.4 Теоретичні відомості про Allure Report.....35
РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ СЕРВІСІВ ТА ІНСТРУМЕНТІВ У ПРОЦЕСІ КОНТРОЛЮ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..... 37
3.1 Використання Cucumber.....37
3.1.1 Підключення Cucumber до проєкту..... 37
3.1.2 Ключові слова та синтаксис Cucumber.....38
3.1.3 Визначення кроків Cucumber.....45
3.2 Використання Selenide..... 46
3.2.1 Підключення Selenide до проєкту..... 46
3.2.2 Основні принципи Selenide.....46
3.2.3 Ядро Selenide.....48
3.2.4 Перевага над Selenium.....49

3.3 Використання AssertJ..... 51
3.3.1 Підключення AssertJ до проєкту..... 51
3.3.2 Порівняння AssertJ з JUnit на прикладі..... 51
3.4 Використання Jenkins..... 52
3.4.1 Встановлення Jenkins..... 52
3.4.2 Запуск Cucumber сценаріїв за допомогою командного рядка... 55
3.4.3 Конфігурування Jenkins Job..... 57
3.4.4 Запуск Jenkins Job..... 62
3.4.5 Інтеграція Jenkins із Slack..... 63
3.5 Використання Allure Report..... 65
3.5.1 Встановлення Allure..... 65
3.5.2 Підключення Allure для роботи з Cucumber 7..... 65
3.5.3 Підключення Allure в Jenkins..... 67
3.5.4 Огляд згенерованого Allure Report..... 67
ВИСНОВКИ..... 72
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ..... 73

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AJAX (Asynchronous JavaScript And XML) − асинхронний JavaScript та XML;

API (Application Programming Interface) – інтерфейс програмування застосунків;

BDD (Behaviour-Driven Development) − керована поведінкою розробка;

CI/CD (Continuous Integration / Continuous Delivery) − неперервна інтеграція / неперервна доставка;

CSS (Cascading Style Sheets) − каскадні таблиці стилів;

DevOps − акронім від англійських слів development і operations;

QA(Quality Assurance) – забезпечення якості продукту;

QC(Quality Control) – контроль якості продукту;

UI (User Interface) − інтерфейс користувача;

URL (Uniform Resource Locator) − єдиний вказівник на ресурс;

ІТ − інформаційні технології;

ПК − персональний комп'ютер;

ШІ – штучний інтелект.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Популярність гнучких методологій розробки, таких як Agile, спонукає до активного розвитку інструментів автоматизованого тестування. Особливістю таких методологій є часті зміни в продукті та необхідність бути впевненим, що після них він працює як очікувалося. Для цього потрібно регулярно проводити регресійне тестування, щоб перевірити, що після змін система лишилась цілісною і працює як очікувалось. Таких тестів на реальних проєктах є доволі багато, це можуть бути сотні або навіть тисячі тестових сценаріїв, які пройти вручну майже не реально за помірний проміжок часу. Тому для такого тестування зазвичай використовують автоматизацію. При цьому має бути правильно налаштований процес та підібрані технології, щоб розбір та моніторинг автоматизованих сценаріїв потребував мінімум зусиль і давав швидку та максимально повну інформацію про стан системи. Саме для цього на проєктах використовуються фреймворки, в які інтегровані різноманітні інструменти автоматизації тестування.

Актуальність роботи та підстави для її виконання. Наявність великої кількості різноманітних інструментів автоматизації тестування робить актуальним завдання класифікації таких інструментів, а також виокремлення лідерів у кожній із категорій. Для всіх фахівців забезпечення якості є необхідним вміння користуватися такими інструментами, а для фахівців, які претендують на посади вище середнього також необхідне вміння інтегрувати ці засоби у тестовий фреймворк. Саме тому доцільно проводити дослідження доступних інструментів для автоматизації тестування та в результаті цього дослідження створити власну систему з використанням найбільш популярних із них.

Мета й завдання роботи. Метою цієї роботи є дослідження різноманітних інструментів автоматизації тестування, класифікація таких засобів, виділення лідерів ринку. Кінцевим результатом цього дослідження має бути власноруч створений фреймворк автоматизації тестування з використанням найпопулярніших технологій. Для досягнення мети було сформовано наступний перелік завдань:

- уточнити сенс понять контролю та забезпечення якості;
- класифікувати інструменти автоматизації тестування та визначити найактуальніші в кожній категорії;
- обрати декілька найпопулярніших інструментів і детально ознайомитися із їхніми можливостями та особливостями;
- розробити власний фреймворк автоматизації тестування з використанням розглянутих інструментів.

Об'єкт, методи й засоби розроблення. Об’єктом розроблення фреймворку автоматизованого тестування є налаштування процесу забезпечення якості за допомогою автоматизації тестування на прикладі тестування базової функціональності вебсайту «ROZETKA».

Розробці даного фреймворку передувала категоризація засобів, які використовуються в процесі контролю якості програмного забезпечення та виокремлення лідерів ринку. Основою для цього стало дослідження наявних інструментів автоматизації тестування.

Під час розробки системи використана ітераційна модель, у результаті якої процес розробки було розділено на наступні ітерації:

- підключення Cucumber та додавання мануальних сценаріїв;
- підключення Selenide та опис класів сторінок;
- додавання бібліотеки AssertJ та імплементація кроків;
- інтеграція з Jenkins;
- підключення Allure Report.

Інструментами створення системи було обрано інтегроване середовище розробки IntelliJ IDEA, мова програмування Java, засіб збірки Maven, інструмент, який підтримує розробку, орієнтовану на поведінку Cucumber, засіб автоматизації вебзастосунків Selenide, бібліотека тверджень AssertJ, інструмент CI/CD Jenkins, інструмент звітування про результати тестування Allure.

Можливі сфери застосування. Розроблений комплексний фреймворк автоматизації тестування може застосовуватися як основа під час впровадження автоматизації тестування вебзастосунків на будь-якому проєкті.

Взаємозв'язок з іншими роботами. За методами розробки та інструментальними засобами робота є продовженням кваліфікаційної роботи на здобуття освітнього рівня бакалавра на тему «Автоматизація тестування вебзастосунків».

РОЗДІЛ 1. ПРОЦЕС КОНТРОЛЮ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Термін «забезпечення якості» (англ. Quality Assurance або QA) [1] іноді використовують для позначення тестування. Проте поняття QA та тестування – це не одне і те саме, хоча вони й пов’язані, їх об’єднує ширша концепція – управління якістю. Управління якістю включає всі види діяльності, які спрямовують і контролюють організацію процесу забезпечення якості. Серед інших видів діяльності управління якістю включає як забезпечення, так і контроль якості (англ. Quality Control або QC).

1.1 Різниця між забезпеченням та контролем якості (QA & QC)

Розробка якісного ПЗ, а також підтримка якості [2] – це головне в життєвому циклі будь-якого проєкту. Тому необхідно дотримуватися певних стандартів, щоб задовольнити вимоги замовника або користувача програмного забезпечення.

Процес забезпечення якості охоплює розробку, створення та випуск програмного забезпечення з мінімальною кількістю дефектів для кінцевого користувача. QA використовує такі методи та техніки для всіх учасників процесу [3], щоб отримати максимально якісний програмний продукт. Основними завданнями забезпечення якості є:

- перевірка вимог до програмного забезпечення;
- врахування ризиків;
- планування QA процесів;
- підготовка тестової документації, середовища та даних;
- тестування;

– оцінка результатів тестування, звітність.

До QA відноситься структурованість процесу виконання роботи над проєктом, тобто присутній план із завданнями, які повинен виконати кожен член команди. Перед тим як почати процес контролю якості мають бути з'ясовані стандарти якості та обсяг тестування.

До процесу QC (Quality Control) належить аналіз результатів тестування, виявлення та усунення дефектів. Контроль якості допомагає з'ясувати стан та якість продукту в конкретний момент часу. Завданнями QC є:

– перевірка якості програмного забезпечення в контексті можливості надати його кінцевому користувачу;
– перевірка відповідності вимогам даної версії програмного продукту.

У випадку, якщо виявлені критичні дефекти чи нема впевненості в їх відсутності, розробка продовжується до моменту виправлення всіх значних недоліків.

Саме по собі тестування не гарантує якість програмного продукту. Воно може лише контролювати виконання вимог до програмного продукту.

Тобто, контроль якості та тестування є підмножинами множини забезпечення якості. Основною особливістю QA процесів є те, що вони починаються ще до початку імплементації програмного продукту, тобто ще на підготовчій аналітичній фазі. QC починається лише тоді, коли програмний продукт має вже якусь проміжну версію чи хоча б частково реалізований функціонал.

Тестувальники зі свого боку мають перевіряти як працюють частини програмного продукту ізольовано, а також їхню взаємодію у кінцевому вигляді. Тобто більшість тестувальників найбільш активну участь приймають саме в процесі контролю якості.

1.2 Типи інструментів, що використовуються в процесі контролю якості програмного забезпечення

Оскільки в сучасних програмних продуктах процес контролю якості є доволі складним і об'ємним його складно уявити без використання спеціальних сервісів та інструментів. Далі буде розглянуто різноманітні типи таких допоміжних програм, які допомагають максимально зручно та якісно налаштувати процеси забезпечення та контролю якості для конкретних областей застосування.

1.2.1 Тестування вебдодатків

Тестування вебдодатків, також відоме як вебтестування [4], – це перевірка функціональності вебдодатків перед їх запуском у робочому середовищі. Інструменти тестування вебдодатків надають важливу інформацію та дані не лише для тестувальників, а і для інших членів команди: розробників, аналітиків, адміністраторів серверів. Прикладами таких інструментів є: Selenium, Selenide, TestComplete, Katalon Studio та інші. Дані інструменти були детально проаналізовані у кваліфікаційній роботі на освітній рівень бакалавра на тему «Автоматизація тестування вебзастосунків».

1.2.2 Тестування мобільних додатків

Інструменти тестування мобільних додатків [5] – це інструменти, призначені для того, щоб допомогти розробникам і тестувальникам у тестуванні мобільних додатків на різних пристроях, операційних системах та умовах мережі. Інструменти тестування мобільних додатків надають низку

можливостей тестування, включаючи функціональне тестування, тестування продуктивності, сумісності та безпеки.

Інструменти тестування мобільних додатків автоматизують процес тестування, дозволяючи тестувальникам заощаджувати час і зусилля та виявляти проблеми та помилки, які можуть вплинути на роботу кінцевих користувачів. Також вони можуть покращити комунікацію між командами, забезпечуючи точність, надійність і вичерпність результатів тестування. Інструменти тестування мобільних додатків відіграють вирішальну роль у забезпеченні високої якості мобільних додатків, відповідності очікуванням кінцевих користувачів і забезпечення позитивного досвіду користувача.

Найпопулярнішими станом на 2023 рік є наступні інструменти тестування мобільних додатків:

- BrowserStack – надає можливість тестувати на понад трьох тисячах реальних пристроях і браузерах;
- Headspin – підходить для тестування з використанням геолокації на основі ШІ та тестуванням продуктивності з тисячами пристроїв;
- Kobiton – мобільна платформа тестування, створена спеціально для підприємств;
- Apptim – дозволяє перевірити продуктивність мобільних програм у CI/CD;
- Ranorex Studio – найкраще підходить для потужного кросплатформового тестування графічного інтерфейсу;
- Perfecto – хмарне тестування з можливостями написання сценаріїв без знання програмування;
- Eggplant – інструмент автоматизації тестування за допомогою ШІ з найзручнішим інтерфейсом користувача;
- BitBar – підходить для тестування безпеки на хмарних і локальних пристроях;

– Appium – найкращий фреймворк із відкритим кодом для додатків iOS, Android та Windows.

1.2.3 Тестування безпеки

Інструменти тестування безпеки [6] допомагають організаціям захистити свою IT-інфраструктуру від зловмисних атак і вразливостей програмного забезпечення. Багато платформ автоматично забезпечують постійний моніторинг, щоб спеціалісти з безпеки та розробники могли випереджати кіберзагрози.

Доступний широкий спектр інструментів із багатьма додатковими функціями, які дають змогу командам DevOps визначати пріоритет безпеки під час розробки програмного забезпечення.

Найпопулярнішими інструментами тестування безпеки є:

– Google Nogotofail – інструмент тестування безпеки мережі для виявлення відомих уразливостей;
– SQLMap – програмне забезпечення для тестування на проникнення для виявлення вразливостей SQL-ін’єкції;
– Snyk – зручна для розробників платформа безпеки з семантичним аналізом коду в реальному часі;
– Intruder – хмарний сканер уразливостей з інструментом автоматичного відстеження IP-адрес;
– SonarQube – додаток для безперервної перевірки коду, чудово підходить для дотримання стандартів якості.

1.2.4 Автоматизація тестування інфраструктури

Автоматизація інфраструктури [7] – це процес, який використовує різні технології для виконання завдань. Він спрямований на обмеження людського втручання в керування компонентами IT-інфраструктури.

Інструменти автоматизації інфраструктури допомагають процесу шляхом встановлення, налаштування та підтримки IT-інфраструктури. Він розгортає апаратне забезпечення, програмне забезпечення, операційні системи, мережеві елементи та компоненти даних для процесу.

Інструменти автоматизації інфраструктури зменшують ризик людської помилки. Це підвищує ефективність IT-операцій і персоналу в IT-середовищі.

Найдосконалішими вважаються наступні інструменти:

- Ansible – інструмент з відкритим кодом, автоматизує функції для розгортання та налаштування IT-компонентів у програмі;
- SaltStack – високошвидкісний інструмент, що оптимально керує та налаштовує IT-інфраструктуру;
- Docker – один із найкращих інструментів автоматизації інфраструктури, який використовується для контейнеризації. Платформа допомагає виконувати кілька програм на одному сервері, не впливаючи одна на одну;
- Jenkins – один із найшвидших інструментів автоматизації інтеграції. Це інструмент на основі Java, який використовується як система контролю версій, наприклад GitHub або SVN;
- Monit – це один із найпростіших інструментів автоматизації інфраструктури з відкритим кодом.

1.2.5 Управління тестовими випадками

Управління тестуванням [8] у сфері тестування програмного забезпечення належить до процесів планування, організації та контролю діяльності з тестування. Управління тестуванням програмного забезпечення передбачає створення та виконання стратегій тестування, визначення планів тестування та тестових випадків, щоб забезпечити відповідність програмного забезпечення бажаним стандартам якості.

Ефективне керування тестуванням має вирішальне значення для успіху проєкту та передбачає співпрацю з усіма зацікавленими сторонами продукту, щоб забезпечити спрощений та ефективний процес тестування.

Список найкращих інструментів керування тестами:

- TestRail – універсальний інструмент керування тестуванням із налаштованими інформаційними панелями та широкою інтеграцією;
- Tuskr – популярний засіб завдяки доступному повному набору функцій керування тестами;
- Xray – додаток для керування тестами Jira, який використовує типи завдань Jira для керування артефактами тестування;
- TestLink – найкращий варіант з відкритим кодом.

1.2.6 Відстеження дефектів

Програмне забезпечення для відстеження помилок [9] – це інструмент, який використовується групами розробників програмного забезпечення для виявлення, відстеження та керування помилками та проблемами у програмному забезпеченні. Він забезпечує систематичний підхід до керування помилками, дозволяючи розробникам визначати пріоритети та ефективно розв'язувати проблеми.

Програмне забезпечення для відстеження помилок зазвичай містить такі функції, як звіти про помилки, відстеження проблем, співпраця, показники та звіти, а також інтеграцію з іншими інструментами в процесі розробки програмного забезпечення. Це допомагає покращити співпрацю, пришвидшити усунення помилок, покращити гарантію якості, підвищити прозорість і забезпечити кращу звітність.

Нижче наведені найчастіше використовувані інструменти відстеження помилок:

- BugHerd – програмне забезпечення для відстеження помилок для тестування вебсайтів;
- GitHub – безкоштовне легке програмне забезпечення;
- Jira Software – для відстеження помилок за використання методології Agile;
- Katalon TestOps – для відстеження помилок із готовими звітами для CI/CD;
- LogRocket – для відстеження помилок, що впливають на розробку зовнішнього програмного забезпечення;
- Bird Eats Bug – розширення браузера Chrome для відстеження помилок;
- Bugzilla – найкраще програмне забезпечення для відстеження помилок з відкритим кодом.

1.2.7 Звітування про автоматизоване тестування

Своєчасна інформація про результати тестування та прогрес є ключем до того, що будь-які помилки, виявлені в програмному забезпеченні, виправляються якнайшвидше та без шкоди для якості кінцевого продукту. Хоча

всі популярні засоби автоматизації тестування мають вбудовані звіти, проте вони не завжди надають повну інформацію про тестування. У таких ситуаціях на допомогу приходять інструменти звітування про автоматизоване тестування [10] – спеціалізовані системи, які допомагають відстежувати охоплення тестуванням, результати тестування, співвідношення успішних і неуспішних тестів і багато іншого.

Наведемо приклади найкращих інструментів звітності про тестування, які допоможуть зрозуміти, наскільки цифровий продукт готовий до випуску та чи відповідає він початковим вимогам.

Allure – це гнучка платформа з відкритим кодом для створення звітів, яка підтримує багато мов програмування. Чудово підходить для командної співпраці над проєктом, оскільки надає інформацію про тестування програмного забезпечення в стислій формі, яку можуть зрозуміти навіть нетехнічні спеціалісти. Інструмент можна встановити для Linux, Mac OS і Windows.

Testomatio – це передова система керування тестуванням, яка поєднує в собі функції фреймворків автоматизації тестування та інструментів звітності про тестування. Створює звіти, які дозволяють відстежувати хід перевірки якості в реальному часі та оцінювати результати в кінці тестового запуску.

ReportPortal – це вебсервіс, який може значно підвищити ефективність автоматизованих тестів. Легко інтегрується з популярними фреймворками тестування, дозволяючи накопичувати всі результати тестів, створювати докладні звіти та проводити глибокий аналіз результатів тестування на їх основі.

Extent Reporting Framework – це бібліотека звітів, яка пропонує користувачам новий погляд на їхні тести за допомогою чітких звітів про виконання тестів і аналітики на їх основі. Інструмент дозволяє створювати зрозумілі інтерактивні звіти для чіткої візуалізації результатів тестування.

Robot framework – це платформа автоматизації тестування на основі ключових слів, яку можна використовувати для роботизованої автоматизації процесів. Інструмент підходить для BDD, розробки на основі приймальних тестів і приймального тестування. Наприкінці тестування інструмент створює звіти про результати тестування у форматі HTML.

1.2.8 Тестування продуктивності та навантаження

Інструменти тестування навантаження [11] – це хмарне програмне забезпечення, яке допомагає тестувати продуктивність або навантаження сайту чи програми. Тестування навантаження – це різновид тестування продуктивності, яке визначає продуктивність системи в умовах реального навантаження. Результати тестування допомагають визначити, як поводиться програма, коли до неї звертаються декілька користувачів одночасно.

Навантажувальне тестування зазвичай визначає:

- максимальну продуктивність програми;
- чи достатньо поточної інфраструктури для запуску програми;
- стійкість програми щодо пікового навантаження користувача;
- кількість одночасних користувачів, яких може підтримувати програма;
- масштабованість.

Тестування навантаження виконується для визначення поведінки системи як у нормальних, так і в пікових умовах. Платформи тестування допомагають у цьому процесі. Надамо короткі відомості про найпопулярніші платформи для цього:

- Headspin – підходить для показників відстеження на основі ШІ;
- WebLOAD – підходить для параметризації та перевірки відповідей;
- Apache JMeter – Java-додаток з відкритим вихідним кодом для функціонального тестування навантаження;

- LoadFocus – підходить для тестування без коду;
- Parasoft Load Test – ідеальний для початківців;
- Neotys Neoload – підходить для складних програм із графічним інтерфейсом користувача;
- Micro Focus LoadRunner – підходить для локальних проєктних програм і тестування навантаження;
- Load Impact – підходить для навантажувального тестування API.

1.2.9 Полегшення читання та написання коду з автоматизації

Для покращення структурування, зручності написання та подальшого розуміння автоматизованих тестових сценаріїв було розроблено багато різноманітних інструментів. Прикладами таких інструментів є:

- Cucumber [12] – це інструмент, який підтримує розробку, орієнтовану на поведінку (BDD);
- RestAssured [13] – це Java бібліотека для спрощення тестування служб на основі REST, створених на основі HTTP Builder. RestAssured підтримує запити POST, GET, PUT, DELETE, OPTIONS, PATCH і HEAD і може використовуватися для підтвердження та перевірки відповіді на ці запити;
- AssertJ [14] – надає багатий та інтуїтивно зрозумілий набір строго типізованих тверджень, що використовуються для модульного тестування (з JUnit, TestNG або будь-якою іншою тестовою структурою).

РОЗДІЛ 2. ОГЛЯД ІНСТРУМЕНТІВ, ЩО ВИКОРИСТОВУЮТЬСЯ В ПРОЦЕСІ КОНТРОЛЮ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі будуть детально розібрані приклади деяких сервісів, назви яких згадувались в першому розділі, а саме: Cucumber, Selenide, Jenkins, Allure Report. Було прийняте рішення зробити детальний розбір саме цих сервісів з огляду на наступні критерії:

- Популярність та актуальність. Необхідно, щоб набутий у рамках даної роботи досвід був релевантним та міг бути використаний в майбутньому на реальних проєктах.
- Інтерфейс користувача. Інструмент має естетично виглядати, бути інтуїтивно зрозумілим та не містити значних недоліків.
- Зручність використання. На скільки легко навчитися користуватися даним інструментом, на скільки він є комфортним для використання.
- Особливості та функціональність. Чи надає даний інструмент достатній обсяг функцій, щоб задовольнити потреби типового реального проєкту.
- Інтеграція. Має бути достатньо легко впровадити його використання в проєкті. Значною мірою на цей показник впливає наявність та якість інформації про це в мережі Інтернет, а також сумісність даного інструмента з іншими.
- Співвідношення ціни та якості. Наскільки ціна відповідає функціям, можливостям і варіантам використання. Оскільки даний проєкт не є комерційним, то за цим критерієм було обрано лише безкоштовні варіанти.

2.1 Теоретичні відомості про Cucumber

Cucumber [12] − це інструмент, який підтримує розробку, орієнтовану на поведінку (BDD). Розробка орієнтована на поведінку (BDD) − це процес розробки програмного забезпечення, для підтримки якого було створено Cucumber.

Cucumber зчитує виконувані специфікації, написані звичайним текстом, і перевіряє, що програмне забезпечення виконує те, що зазначено в цих специфікаціях. Специфікації складаються з кількох прикладів або сценаріїв. Наприклад:

Сценарій: Іван вгадує слово

Нехай Олег загадує слово

Коли Іван робить припущення

Тоді Олег підтверджує або спростовує припущення

Кожен сценарій − це список кроків, які необхідно виконати Cucumber. Cucumber перевіряє, що програмне забезпечення відповідає специфікації та створює звіт із зазначенням успіху або невдачі для кожного сценарію.

2.1.1 Розробка, орієнтована на поведінку (BDD)

BDD (Behaviour-Driven Development) – це спосіб роботи команд розробки програмного забезпечення, який усуває непорозуміння між людьми бізнесу та технічними людьми за допомогою наступних підходів:

– заохочення співпраці між ролями для формування спільного розуміння проблеми, яку потрібно вирішити;

- роботи на швидких та малих ітераціях для збільшення зворотного зв’язку;
- створення системної документації, яка автоматично перевіряє на відповідність поведінку системи.

Якщо зосереджувати спільну роботу на конкретних реальних прикладах, які ілюструють, як має поводитися система, то ці приклади допомагають пройти шлях від концепції до реалізації в процесі безперервної співпраці.

Для впровадження BDD команда вже має використовувати якусь гнучку методологію розробки програмного забезпечення, планувати роботу з невеликими наборами змін, наприклад, історії користувача. BDD не є альтернативою наявного гнучкого процесу розробки, він лише покращує його.

BDD зробить команду спроможнішою виконувати гнучкі обіцянки: своєчасні надійні випуски робочого програмного забезпечення, яке відповідає потребам організації, що постійно оновлюються, а це вимагає певних зусиль і дисципліни.

Потрібно мати можливість швидко реагувати на відгуки користувачів і виконувати лише мінімальну роботу, необхідну для задоволення цих потреб.

BDD заохочує працювати на швидких ітераціях, безперервно розбиваючи проблеми користувача на невеликі шматки, які можуть проходити через процес розробки якомога швидше.

Типова діяльність BDD – це триетапний ітераційний процес:

- невелика майбутня зміна в системі – історії користувача, обговорення конкретних прикладів нової функціональності для з'ясування та узгодження деталей того, що планується реалізувати;
- документування прикладів очікуваної роботи системи способом, який можна в перспективі покрити автоматизованими тест-кейсами, і перевірка, чи узгоджуються вони з існуючою функціональністю;

– реалізація поведінки, описаної в кожному задокументованому прикладі та автоматизація відповідних тестових сценаріїв.

Ідея полягає в тому, щоб зробити кожну зміну невеликою і швидко повторювати ітераційний процес, за потреби повертаючись на початковий рівень щоразу, коли знадобиться більше інформації. Кожного разу під час автоматизації та реалізації нового прикладу, коли додається щось важливе у систему, є готовність швидко та ефективно на це відреагувати.

Тобто, ітераційний процес BDD складається з методів виявлення, формулювання та автоматизації (рисунок 1).

Рисунок 1 – Схема виявлення, формулювання та автоматизації

З часом задокументовані приклади стають активом, який дає змогу команді продовжувати впевнено та швидко вносити зміни в систему. Код відображає документацію, а документація відображає спільне розуміння командою предметної області. Це спільне розуміння постійно розвивається.

2.1.2 Синтаксичні правила Gherkin

Щоб Cucumber розумів сценарії, вони повинні дотримуватися деяких основних синтаксичних правил, які називаються Gherkin.

Gherkin − це набір граматичних правил, який робить звичайний текст достатньо структурованим для розуміння Cucumber.

Gherkin має кілька цілей (рисунок 2):

– однозначна специфікація;
– автоматизоване тестування за допомогою Cucumber;
– документування очікуваної поведінки системи.

Рисунок 2 − Єдине джерело істини

Граматика Cucumber існує в різних варіантах для багатьох розмовних мов, щоб команда могла використовувати ключові слова рідною мовою.

Для того, щоб покращити доступність для людей, які добре володіють лише рідною мовою, ключові слова було перекладено. Деякі мови можуть мати більше ніж один переклад для будь-якого ключового слова. Приклади перекладів ключових слів мови Gherkin можна знайти за посиланням: https://cucumber.io/docs/gherkin/languages.

Gherkin документи зберігаються у файлах із розширенням feature в репозиторії разом із файлами, в яких описана реалізація кроків на якійсь із мов програмування.

2.1.3 Визначення кроків

Визначення кроків пов’язують кроки Gherkin з програмним кодом. Визначення кроку виконує дію, яку має виконати крок. Тобто, визначення кроків жорстко пов'язує специфікацію з реалізацією (рисунок 3).

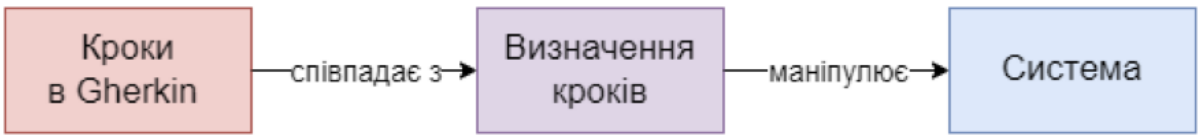

Рисунок 3 – Діаграма визначення кроків

Визначення кроків можна написати багатьма мовами програмування. Вираз визначення кроку може бути регулярним або Cucumber виразом. При використанні регулярних виразів кожна група захоплення з відповідності буде передана як аргументи до методу визначення кроку.

Визначення кроків не мають жорсткого зв'язку з конкретним feature файлом або сценарієм. Ім'я файлу, класу або пакета визначення кроку не

впливає на те, яким крокам Gherkin воно відповідатиме. Єдине, що має значення, – це вираз визначення кроку.

2.2 Теоретичні відомості про Selenide

Selenide [15] − це фреймворк для автоматизованого тестування вебзастосунків, в основі якого міститься Selenium WebDriver. Selenide має наступні переваги в порівнянні із Selenium WebDriver [16]:

- зручний інтерфейс для роботи із WebDriver (Selenide автоматично запускає і вимикає браузер за необхідності);
- витончений API, за допомогою якого можна писати компактний та зрозумілий код;
- вбудовані методи очікування, які значно підвищують стабільність тестів та зменшують кількість роботи з написання таких методів власноруч;
- автоматизовані знімки екрана у випадку, коли тест-кейс завершився падінням.

Тобто, Selenide дозволяє автоматизатору сфокусуватися на бізнес-логіці, а фреймворк допоможе із більшістю технічних моментів.

2.2.1 Selenide та Selenium

Selenium та Selenide [17] − це два різних терміни, які можуть здаватися ідентичними на перший погляд. Проте, вони відрізняються один від одного, і зрозуміти цю відмінність можна розібравши детально особливості кожного із них.

2.2.1.1 Особливості Selenium

Selenium має інструмент відтворення для створення функціональних тестів, автоматизатору потрібно вивчити мову тестових сценаріїв під назвою Selenium IDE, щоб працювати з ним.

Selenium Testing Framework постачається з мовою тестового домену Selenese для написання тестів кількома мовами програмування, такими як: Groovy, C#, Java, PHP, Perl, Python, Scala і Ruby.

Перевагою написання автоматизованих тестових сценаріїв за допомогою Selenium є те, що можна запускати їх у сучасних веббраузерах, і це дозволяє проводити тестування на різних операційних системах, таких як Windows, macOS та Linux.

Selenium є повністю відкритим програмним продуктом, який складається із:

- Selenium IDE;
- Selenium Grid;
- Selenium Server.

Selenium IDE − це плагін, який підтримує Mozilla Firefox для запису та відтворення певних дій користувача. Він відрізняється від Selenium Server або Selenium Grid тим, що працює лише з локальними веббраузерами.

Selenium Grid можна встановити на ПК і віддалено взаємодіяти з іншими, якщо вони встановили Selenium Server. Для того, щоб створити сеанс, потрібно перейти до ПК за допомогою Grid. Після цього він визначить, скільки часу потрібно для завантаження машини. Виходячи з цього, він пропонує необхідні команди. Потрібно пам’ятати, що майже всі машини взаємодіють із Selenium Grid, і він працює на різних операційних системах та з різними веббраузерами.

Selenium Server − це програма, яка допомагає керувати веббраузерами відповідно до оригінального блоку драйверів. Selenium Server має бути на тому ж комп’ютері, крім того, має бути веббраузер, щоб почати роботу на ньому. За допомогою Selenium Server можна легко керувати веббраузерами на певному локальному ПК.

Вебдрайвер є найкращим інструментом, але це не інструмент тестування. Selenium Web Driver має кілька тестових бібліотек, але вони не вирішують наступні проблеми:

– незручність написання UI тестів;
– нестабільність тестів, що викликана динамічним вмістом, JavaScript, Ajax, Time out тощо.

Автоматизацією тестування інтерфейсу користувача можна покрити модульні, функціональні, інтеграційні сценарії тощо. У випадку з вебдодатком пишуться скрипти, які відкривають сторінку сайту в новому браузері та починають натискати кнопки та перевіряти елементи на сторінці.

AJAX [18] − це підхід до побудови користувацьких інтерфейсів вебзастосунків, який полягає в тому, що вебсторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер, а звідти довантажує необхідні користувачу дані.

Проблематикою, пов'язаною з Ajax та Time out є те, що тестові сценарії час від часу будуть падати через нижчу ніж зазвичай швидкодію сервера. Це може бути пов'язано із різними причинами, такими як зміни на сервері, велике навантаження на сервер тощо. JavaScript стає повільним, деякі запити Ajax займають багато часу і деякі інші процеси виконуються одночасно з тестовими сценаріями, які займають простір центрального процесора сервера.

У випадку використання Selenium для подолання описаної проблеми використовуються методи «sleep» або «wait_until» у тестах. Проте ці методи не використовуються для кожного елемента за замовчуванням, тому є доволі

висока ймовірність того, що тести можуть почати падати в будь-якому місці, якщо той чи інший елемент не відобразиться моментально, як ми цього очікували. Варто зазначити, що в ідеальному світі під час написання тестів основна увага має бути зосереджена на бізнес-логіці без необхідності турбуватися про time out, явні та неявні очікування, життєвий цикл WebDriver тощо.

Варто зауважити, що Selenium WebDriver не є інструментом тестування, це інструмент керування браузером. Тому через інструмент Selenium створено кілька інструментів, таких як FluentLenium, Fluent-selenium, HTMLElements, Thucydides, Yandex, Watir-webdriver. Selenide є одним із них і він створений для написання стислого, виразного, чистого коду та стабільних тестів інтерфейсу користувача на Java.

2.2.1.2 Особливості Selenide

Selenide використовується для написання програмних кодів для автоматизації тестування, допомагає створювати та надсилати HTTP-запити на Server/Grid. Однією з основних цілей використання є створення сценаріїв для легкої перевірки вебдодатків. Він розроблений спеціально для створення скриптів, які можуть тестувати роботу вебпродуктів: виявлення необхідних вебоб’єктів, перевірку виконання подій, роботу з UI тощо.

Selenide пропонує вбудоване рішення проблем Ajax/Timeouts. Це реалізовано наступним чином: кожен метод повинен почекати кілька секунд перед тим як повернути помилку про те, що об'єкт не знайдено чи його стан не відповідає очікуваному. Тобто, в більшості випадків нема потреби використовувати команди sleeps та waits.

Розглянемо, як приклад, наступний рядок коду:

```
$("#title").shouldHave(text("Menu"));
```

У цьому рядку Selenide перевіряє, чи містить елемент заголовка текст «Menu». Якщо ні, то елемент буде динамічно оновлюватись протягом деякого часу і знову перевірятись на наявність у ньому тексту. Для більшості вебпрограм Timeout за замовчуванням становить 4 секунди.

2.2.2 Порівняння Selenium та Selenide на прикладі

У випадку використання Selenium Web Driver потрібно написати:

By user = By.name("user");

driver.findElement(user).click();

При використанні Selenide можна просто написати:

By user = By.byName("user");

$(user).click();

Можна побачити, що під час використання Selenide користувачу не потрібно безпосередньо взаємодіяти із вебдрайвером. Також Selenide має вбудований механізм очікування, тому не потрібно в явному вигляді прописувати очікування виклику Ajax. Тобто, можна очікувати відносно стабільного автоматизованого тестування програм з викликами Ajax.

2.2.3 Вибір Selenide

Багато компаній використовують інструмент Selenide замість Selenium WebDriver, оскільки Selenide спеціально розроблений для автоматизації тестування з використанням Selenium WebDriver. Ще одним аргументом за використання Selenide замість Selenium є хороший API для роботи з вебдрайвером.

У кваліфікаційній роботі на практиці скористаємось Selenide, оскільки він спеціально розроблений для автоматизації тестування, має хороший API та інші переваги в порівнянні з Selenium. Недарма даний інструмент обрали багато відомих компаній, такі як MasterCard, IBM, Google та інші.

2.3 Автоматизація в неперервній інтеграції за допомогою Jenkins

2.3.1 Неперервна інтеграція

Неперервна інтеграція (англ. Continuous Integration *або CI*) [19] є фундаментальною практикою для розробки сучасного програмного забезпечення. Вона полягає в тому, що розробники інтегрують свій код у загальне сховище принаймні раз на день. Досвідчені команди розробників часто інтегрують свій код багато разів на день. Іншою важливою частиною CI є автоматичні перевірки, які відбуваються після того, як інженери інтегрують свій код: сервер CI запускає всі автоматизовані тести та інші перевірки, такі як перевірки безпеки, статичний аналіз тощо.

CI вирішує проблему «інтеграційного пекла» − тобто розробники працювали б ізольовано тижнями або навіть місяцями та лише потім намагалися інтегрувати свою роботу. Це неймовірно ускладнило інтеграцію, було багато злиття та логічних конфліктів. У результаті інженери боялися інтеграції, відкладали її до останнього можливого моменту, що лише погіршувало проблему.

Іншою важливою перевагою CI є перевірки, які виконуються з кожним комітом, що може виявити проблеми з кодом ще до того, як він потрапить у виробництво.

2.3.2 Роль автоматизованих тестів у CI

CI − це практика, коли розробники регулярно інтегрують свій код у спільне сховище. Щоб переконатися, що інтеграція пройшла успішно, існує набір автоматизованих тестів для оцінки цілісності коду. Автоматичний запуск тестів є одним із ключових принципів CI.

Ідея практики CI полягає в тому, щоб кожен у команді мав останню робочу версію коду. Ось чому варто інтегруватися кілька разів на день. Після додавання автоматизованих тестів у конвеєр розробки, продуктивність розробників підвищується. Розробники можуть вносити зміни в код з більшою впевненістю, оскільки автотести перевіряють їхні зміни. Проте тести мають виконуватися якомога швидше, щоб оперативно надати відгук. Інакше виправлення та пошук проблем може зайняти багато часу, що значно погіршить продуктивність.

Важливо також зазначити, що CI не пропагує використання будь-якого конкретного інструменту, оскільки найбільше значення мають самі практики та культура. Але щоб отримати краще уявлення про те, як використовувати автоматизоване тестування в конвеєрі CI, в рамках даної кваліфікаційної роботи розглянемо як це можна зробити за допомогою Jenkins.

2.3.3 Інструмент Jenkins

Jenkins − це інструмент CI/CD з відкритим кодом. Можна використовувати його потужність, щоб автоматизувати завдання, що необхідні для створення, тестування та навіть розгортання програм.

Jenkins є одним із найпопулярніших інструментів у своїй категорії. Велика частина цієї популярності пояснюється його розширюваністю. Завдяки

архітектурі плагінів Jenkins можна змусити працювати з великою кількістю інструментів і служб. Користувачі можуть використовувати наявну екосистему плагінів або навіть створити власний.

Слово «фреймворк» сьогодні доволі часто використовують як синонім слова «інструмент». З цих міркувань Jenkins можна вважати системою автоматизації. Проте, вживаючи точнішу термінологію, краще називати Jenkins «сервером автоматизації» відповідно до визначення з офіційного сайту Jenkins [20].

Щоб налаштувати CI/CD у Jenkins, потрібно використовувати його конвеєр.

Jenkins Pipeline (або просто «Pipeline») [19] − це набір плагінів, який підтримує реалізацію та інтеграцію конвеєрів безперервної доставки в Jenkins.

Jenkins дозволяє виражати кроки процесу CI/CD за допомогою коду Pipeline DSL (спеціальна мова). Тому конфігурація стає простим файлом, який можна додати до репозиторію керування версіями та відстежувати внесені зміни до нього.

Тестувальники, фахівці з контролю якості або будь-які професіонали, відповідальні за тестування програмного забезпечення (якими, в ідеалі, повинні бути всі технічні члени команди), використовують Jenkins для налаштування конвеєра, у якому тести автоматично виконуються щоразу, коли зміна коду надсилається до центрального сховища.

2.3.3.1 Переваги Jenkins

Розширювана архітектура Jenkins значною мірою відповідає за його популярність, але це не єдина його сильна сторона. Наведемо перелік інших переваг Jenkins:

– просте інсталювання та швидке налаштування;

- гнучкість використання, тобто його можна використовувати просто як сервер CI або як повне рішення для неперервного розгортання програми;
- безкоштовний інструмент з відкритим кодом;
- велика спільнота користувачів, яка забезпечує наявність достатньої кількості матеріалів, і в разі потреби можна доволі легко знайти фахівця з таким досвідом;
- безліч плагінів, які допоможуть автоматизувати тестування, в тому числі плагіни, що можуть відображати та підсумовувати результати.

2.3.3.2 Недоліки Jenkins

Не існує ідеального інструменту, і Jenkins не є винятком. Ось кілька недоліків цього засобу:

- надмірність плагінів: іноді важко обрати потрібний варіант, особливо для початківців, коли є кілька плагінів для однієї мети;
- недосконалий інтерфейс користувача, який може здатися інтуїтивно незрозумілим і заплутаним, особливо для новачків;
- функції керування користувачами є дещо примітивними, деякі більш розширені функції важко або неможливо виконати;
- низька якість деяких плагінів, зумовлена їх кількістю, багато з них працюють некоректно або вже не підтримуються.

2.4 Теоретичні відомості про Allure Report

Allure Framework [21] − це гнучкий інструмент звітності про тестування, який не тільки показує стисле представлення того, що було протестовано в акуратній формі вебзвіту, а й дозволяє кожному, хто бере участь у процесі

розробки, отримувати максимум корисної інформації з щоденних запусків тестів.

З точки зору розробників та інженерів по забезпеченню якості Allure Report скорочує загальний життєвий цикл дефектів: падіння тестових випадків можна розділити на потенційні дефекти та проблеми із тестами, також можна налаштувати журнали, вкладення, час, історію та інтеграцію з системами відстеження помилок, тому відповідальні розробники та тестувальники матимуть всю необхідну інформацію під рукою.

З точки зору менеджерів, Allure надає чітку «загальну картину» того, які функції були охоплені, де згруповані дефекти, як виглядає графік виконання та багато інших зручних речей. Модульність і розширюваність Allure забезпечує можливість налаштувати його відповідно до власних потреб.

РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ СЕРВІСІВ ТА ІНСТРУМЕНТІВ У ПРОЦЕСІ КОНТРОЛЮ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даному розділі буде описана робота із Cucumber, Selenide, AssertJ, Jenkins та Allure Report: як під'єднати дані фреймворки до проєкту та як їх використовувати.

3.1 Використання Cucumber

Cucumber доступний для більшості популярних мов програмування, таких як Java, JavaScript, C++, Kotlin, Ruby, Scala, Go і деяких інших. У рамках даної кваліфікаційної роботи буде розглянуто використання Cucumber з мовою програмування Java використовуючи інтегроване середовище розробки IntelliJ IDEA, плагін IntelliJ IDEA Cucumber для Java та засіб збірки Maven.

У цьому підрозділі будуть розкриті наступні теми: додавання Cucumber до проєкту, розгляд синтаксису Gherkin та написання сценаріїв перевірки інтернет-магазину ROZETKA відповідно до цього синтаксису.

3.1.1 Підключення Cucumber до проєкту

Cucumber-JVM опубліковано в центральному репозиторії Maven, тобто можна встановити Cucumber для роботи з Java, додавши залежність до pom.xml файлу (рисунок 4).

```
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.11.2</version>
    <scope>test</scope>
</dependency>
```

Рисунок 4 – Залежність, що додається у pom файл
для підключення Cucumber

3.1.2 Ключові слова та синтаксис Cucumber

Gherkin використовує набір спеціальних ключових слів, щоб надати структуру та значення виконуваним специфікаціям. Кожне ключове слово перекладено багатьма розмовними мовами, в даній роботі буде використано англійську мову, оскільки вона є міжнародною та найчастіше використовується в IT індустрії.

Більшість рядків у документі Gherkin починаються з одного з ключових слів. Коментарі дозволені лише на початку нового рядка в будь-якому місці feature файлу. Вони починаються з нуля або більшої кількості пробілів, за якими йде знак решітки, а за нею вміст коментаря.

Для відступів можна використовувати пробіли або табуляції. Рекомендований рівень відступу – два пробіли.

Кожен рядок, який не є порожнім, має починатися з ключового слова Gherkin. Виняток становлять лише описи функцій і сценаріїв.

Основними ключовими словами є:

– Feature;
– Example (або Scenario);

– Given, When, Then, And, But для кроків (або *);
– Background;
– Scenario Outline (або Scenario Template);
– Examples (або Scenarios).

Також є кілька другорядних ключових слів:

– """ (рядки документів);
– | (таблиці даних);
– @ (теги);
– # (коментарі).

Мета ключового слова Feature – надати високорівневий опис функції програмного забезпечення та згрупувати пов’язані сценарії. Першим основним ключовим словом у документі Gherkin завжди має бути Feature, за ним двокрапка і короткий текст, що описує функцію. Можна додати текст довільної форми внизу Feature для більш детального опису. Ці рядки опису ігноруються Cucumber під час виконання, але доступні для звітів. Назва та необов’язковий опис не мають особливого значення, їх мета – забезпечити місце для документування важливих аспектів функції, таких як коротке пояснення та список бізнес-правил (загальні критерії прийняття).

Опис вільного формату для Feature закінчується, коли рядок починається з ключових слів Background або Example, Scenario Outline і тому подібних. Можна розміщувати теги вище Feature для групування пов’язаних функцій, незалежно від структури файлів і каталогів. Проте слово Feature у feature файлі можна мати лише одне. Описи у довільній формі (як для Feature) також можна розмістити під Example/Scenario, Background, Scenario Outline.

Сценарій – це конкретний приклад, який ілюструє бізнес-правило. Він складається зі списку кроків. Ключове слово Scenario є синонімом ключового

слова Example. Дозволено мати скільки завгодно кроків, але рекомендовано 3-5 кроків для кожного прикладу. Занадто багато кроків призведе до того, що приклад втратить свою виразну силу як специфікацію та документацію. Окрім специфікації та документації прикладом є також тест. Загалом, приклади є виконуваною специфікацією системи.

Приклади мають наступний шаблон:

– опис початкового контексту (Given кроки);
– опис події (When кроки);
– опис очікуваного результату (Then кроки).

Кожен крок починається з Given, When, Then, And або But. Cucumber виконує кожен крок сценарію по одному в тій послідовності, в якій їх було записано. Коли Cucumber намагається виконати крок, він шукає відповідне визначення кроку для виконання. Під час пошуку визначення кроку ключові слова не враховуються. Це означає, що не можна мати Given, When, Then, And або But крок з тим самим текстом, що й інший крок.

Given кроки використовуються для опису початкового контексту системи. Коли Cucumber виконує Given крок, він налаштовує систему у чітко визначеному стані, наприклад, створює та налаштовує об’єкти або додає дані до тестової бази даних. Метою Given кроків є переведення системи у відомий стан до того, як користувач (або зовнішня система) почне взаємодіяти з системою (на When етапах).

When кроки використовуються для опису події або дії. Це може бути особа, яка взаємодіє з системою, або це може бути подія, ініційована іншою системою. Рекомендовано використовувати лише один When крок для кожного сценарію.

Then кроки використовуються для опису очікуваного результату. Визначення кроку із Then має використовувати твердження, щоб порівняти фактичний результат (що система насправді робить) з очікуваним результатом (те, що система має зробити). Результат повинен бути очевидним, наприклад: звіт, інтерфейс користувача, повідомлення, а не поведінка, глибоко захована в системі (наприклад, запис у базі даних).

Повторювані послідовні Given, When або Then, варто замінювати на And і But. Gherkin також підтримує використання зірочки замість будь-якого зі звичайних ключових слів кроку. Це може бути корисно, коли є деякі кроки, які фактично є списком речей, тож можливо виразити це як пункти.

У випадку, коли присутні повторювальні ті самі Given кроки в усіх сценаріях у файлі Feature, це свідчить про те, що ці кроки не є суттєвими для опису сценаріїв, тому їх можна згрупувати та винести у Background розділ. Background дозволяє додати деякий контекст до наступних сценаріїв. Він може містити один або кілька Given кроків, які виконуються перед кожним сценарієм. Background розміщується перед першим Scenario або Example, на тому ж рівні відступу.

Відповідно до правил, описаних у цьому підрозділі, в рамках практичної частини кваліфікаційної роботи було описано тестові випадки у feature файлах, для тестування функціональності на сайті інтернет-магазину ROZETKA.

У файлі, що містить перевірки функціональності пошуку, було описано наступні перевірки (рисунок 5):

- URL містить слово по якому виконується пошук;
- всі результати містять слово за яким виконувався пошук;
- у випадку пошуку по неправильному слову отримуємо відповідне повідомлення.

SearchGoods.feature

```
Feature: Verify search goods functionality

  Background:
    Given I open ROZETKA home page

  Scenario: Verify that URL contains search word
    When I set IPhone value in Search field and press Enter
    Then I verify that URL contains IPhone

  Scenario: Verify that search results contains search word
    When I set iPhone value in Search field and press Enter
    Then I verify that search result contains iPhone word

  Scenario: Verify catalog empty message is displayed
    When I set wrongSearchWord value in Search field and press Enter
    Then I see catalog empty message
```

Рисунок 5 – Опис сценаріїв для тестування функціональності пошуку

У файлі, що містить перевірки функціональності фільтрування товарів та сортування результатів пошуку, було описано наступні перевірки (рисунок 6):

– фільтрація по мінімальній ціні;
– фільтрація по максимальній ціні;
– сортування від дешевих до дорогих;
– сортування від дорогих до дешевих;
– фільтрація по обсягу пам’яті.

FilteringAndSortingInCatalog.feature

```
Feature: Verify filtering and sorting functionality in catalog

  Background:
    Given I open ROZETKA home page
    And I set iPhone value in Search field and press Enter

  Scenario: Verify that all goods prices greater than min cost
    When I set 30000 value in min cost field and press OK
    Then I verify that all prices greater than 30000

  Scenario: Verify that all goods prices less than max cost
    When I set 30000 value in max cost field and press OK
    Then I verify that all prices less than 30000

  Scenario: Verify cheap first sorting option of price
    When I choose cheap first sort option
    Then I verify that all goods are sorted in ascending order of price

  Scenario: Verify expensive first sorting option of price
    When I choose expensive first sort option
    Then I verify that all goods are sorted in descending order of price

  Scenario: Verify 512 GB memory filtering
    When I choose 512GB option in memory filter
    Then I verify that search result contains 512GB word
```

Рисунок 6 – Опис сценаріїв для тестування фільтрації та сортування каталогу товарів

У файлі, що містить перевірки функціональності кошика, було описано наступні перевірки (рисунок 7):

– зміна значення лічильника кількості товарів після додавання товару;
– відповідність ціни та назви товару в кошику каталогу;
– подвоєння сумарної ціни на товар за умови збільшення кількості товару в кошику на один;
– сумарної ціни декількох різних товарів у кошику;
– видалення всіх товарів із кошика.

Cart.feature

0 results

```
Feature: Verify cart functionality

  Background:
    Given I open ROZETKA home page
    And I set iPhone value in Search field and press Enter

  Scenario: Verify cart header counter for one good in the cart
    When I click add to cart button for the 1 good
    Then I verify that cart header counter equals 1

  Scenario: Verify cart title and price in the cart
    And I store first goods title in catalog
    And I store first goods price in catalog
    When I click add to cart button for the 1 good
    And I open cart using header
    Then I verify that goods title in the cart equals to title in the catalog
    Then  I verify sum goods price in the cart with 1 the same goods

  Scenario: Verify sum in cart after changing goods count
    And I store first goods price in catalog
    When I click add to cart button for the 1 good
    And I open cart using header
    Then I verify sum goods price in the cart with 1 the same goods
    When I increment goods count in the cart
    Then I verify sum goods price in the cart with 2 the same goods
    When I decrement goods count in the cart
    Then I verify sum goods price in the cart with 1 the same goods
```

Рисунок 7 – Опис сценаріїв для тестування функціонала кошика

```
28
29  Scenario: Verify sum price of different goods in cart
30    And I store first goods price in catalog
31    And I store second goods price in catalog
32    When I click add to cart button for the 1 good
33    And I click add to cart button for the 2 good
34    And I open cart using header
35    Then I verify sum goods price in the cart with two different goods
36
37  Scenario: Verify delete all goods from cart
38    When I click add to cart button for the 1 good
39    And I open cart using header
40    And I open goods actions tab and choose delete button
41    Then I verify that empty cart icon present
```

Рисунок 7 – Опис сценаріїв для тестування функціонала кошика (продовження)

3.1.3 Визначення кроків Cucumber

Визначення кроку – це метод Java з виразом, який пов’язує його з одним або кількома кроками Gherkin. Коли Cucumber виконує крок Gherkin у сценарії, він шукатиме відповідне визначення кроку для виконання.

Вираз визначення кроку може бути регулярним або Gherkin виразом. У регулярних виразах, кожна група захоплення з відповідності буде передана як аргументи до методу визначення кроку. Якщо вираз групи захоплення ідентичний одному із зареєстрованих типів параметрів, regexp рядок буде перетворено до того, як він буде переданий методу визначення кроку.

Більш детальна інформація про визначення кроків та їх приклади буде подана у наступному підрозділі, оскільки для визначення кроків у даній роботі було використано Selenide.

3.2 Використання Selenide

3.2.1 Підключення Selenide до проєкту

Selenide підключається за допомогою додавання залежності в pom файл (рисунок 8).

```
<dependency>
    <groupId>com.codeborne</groupId>
    <artifactId>selenide</artifactId>
    <version>6.13.0</version>
    <scope>test</scope>
</dependency>
```

Рисунок 8 – Залежність, що додається у pom файл
для підключення Selenide

3.2.2 Основні принципи Selenide

Практичне використання Selenide базується на трьох основних кроках.

Можливість відкрити сторінку за допомогою методу open. Аргументом цього методу є посилання на ту сторінку, яку потрібно відкрити у веббраузері (рисунок 9).

```
1 usage
String homePageUrl = "https://rozetka.com.ua/";

3 usages  admin
@Given("I open ROZETKA home page")
public void openHomePage() { open(homePageUrl); }
```

Рисунок 9 – Приклад реалізації кроку, що відкриває сторінку

Можливість виконати дію стосовно якогось елемента (рисунок 10).

```
1 usage   admin
@And("^I increment goods count in the cart$")
public void incrementGoodsCountInTheCart() {
    cartElement.getAddOneMoreGood().click();
}
```

Рисунок 10 – Приклад реалізації кроку, що збільшує кількість вибраних товарів у кошику

Можливість перевірити чи виконується умова щодо якогось елемента (рисунок 11).

```
1 usage   admin
public void emptyChartVisible() {
    emptyChart.shouldBe(visible);
}
```

Рисунок 11 – Приклад реалізації методу, що перевіряє наявність елемента порожнього кошика

Selenide API складається з невеликої кількості класів. Розробники Selenide пропонують припинити читати документацію, відкрити свою улюблену IDE і просто почати друкувати. Достатньо просто набрати: $(selector). і IDE запропонує всі можливі опції (рисунок 12).

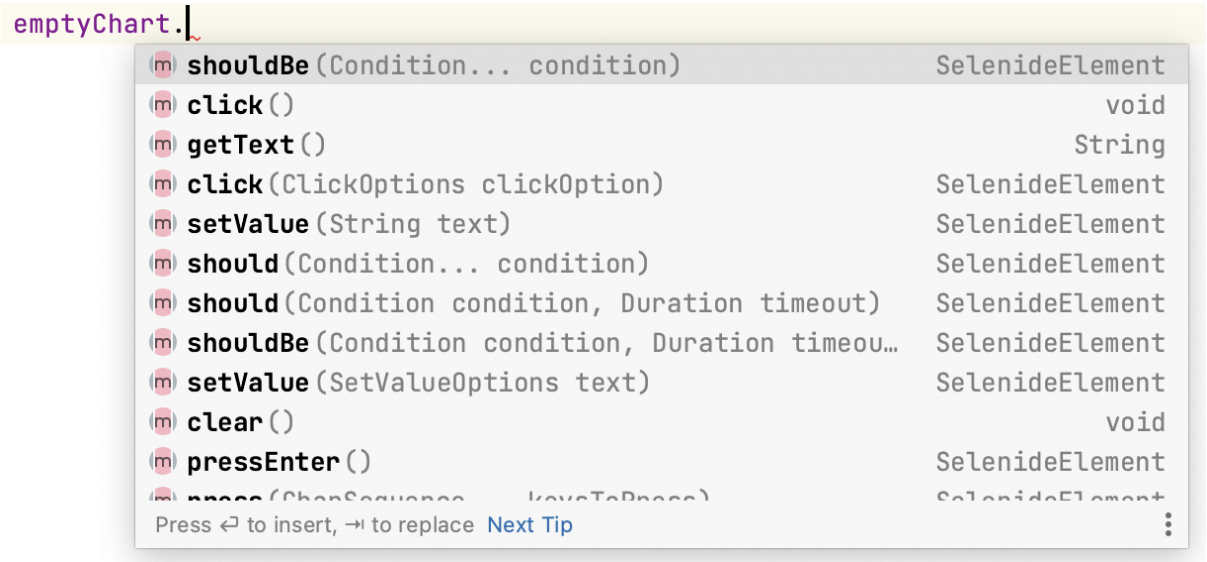

Рисунок 12 – Selenide API: просто почни писати

Тобто, варто використовувати всю силу сучасних засобів розробки замість того, щоб забивати голову документацією!

3.2.3 Ядро Selenide

Розглянемо ядро бібліотеки Selenide. Основні методи – це open, $, $$.

open(URL) – відкриття тестованої сторінки програми (головна відправна точка Selenide).

$(String cssSelector) – повертає об’єкт типу SelenideElement, який представляє перший знайдений за CSS селектором елемент на сторінці.

$$(String cssSelector) – повертає об’єкт типу ElementsCollection, який представляє колекцію всіх елементів знайдених за CSS селектором.

$x(String xpath) – повертає перший SelenideElement по xpath локатору.

$$x(String xpath) – повертає об’єкт типу ElementsCollection, який представляє колекцію всіх елементів, знайдених за xpath селектором.

Клас SelenideElement описує елемент, знайдений на сторінці. Його об'єкт можна, наприклад, отримати за допомогою команди $. Більшість дій також

повертають об'єкт SelenideElement, дозволяючи використовувати ланцюжки викликів (рисунок 13).

```
5 usages   admin
@And("^I set (.*) value in Search field and press Enter$")
public void setValueInSearchFieldAndPressEnter(String searchWord) {
    headerElement.getSearchField().setValue(searchWord).pressEnter();
}
```

Рисунок 13 – Приклад використання ланцюжків викликів
за допомогою Selenide

Рекомендовано статично імпортувати використовувані умови, щоб покращити читабельність коду (рисунок 14).

```
import static com.codeborne.selenide.Condition.visible;
```

Рисунок 14 – Статичний імпорт умови

3.2.4 Перевага над Selenium

Більшість операцій над елементами, отриманими за допомогою $ та $$ в Selenide, мають вбудовані неявні очікування залежно від контексту. Це дозволяє в більшості випадків не відволікатися на явні очікування завантаження елементів при тестуванні динамічних вебдодатків. Для прикладу порівняємо той самий тест-кейс, автоматизований за допомогою Selenide (рисунок 15) та Selenium (рисунок 16).

```
Scenario: Verify catalog empty message is displayed
  When I set wrongSearchWord value in Search field and press Enter
  Then I see catalog empty message
```

Рисунок 15 – Кейс автоматизований за допомогою Selenide:
а – опис тестового випадку в feature файлі

```
5 usages   admin
@And("^I set (.*) value in Search field and press Enter$")
public void setValueInSearchFieldAndPressEnter(String searchWord) {
    headerElement.getSearchField().setValue(searchWord).pressEnter();
}
```

Рисунок 15 – Кейс автоматизований за допомогою Selenide:
б – реалізація кроку пошуку на сайті за пошуковим словом

```
@Then("^I see catalog empty message$")
public void assetThatSearchResultsContainsSearchWord() {
    searchResultsPage.catalogEmptyVisible();
}

public void catalogEmptyVisible() {
    catalogEmpty.shouldBe(visible);
}
```

Рисунок 15 – Кейс автоматизований за допомогою Selenide:
в – реалізація кроку перевірки того, що кошик порожній

```
@Test
void checkThatSearchWithWrongSearchWord() {
    String wrongSearchWord = ConfProperties.getProperty("wrongSearchWord");
    getHeaderElement().inputSearchAndClickSearchButton(wrongSearchWord);
    getBaseElement().waitForPageReadyState();
    Assertions.assertTrue(getSearchResultPage().catalogEmptyMessageExist());
}
```

Рисунок 16 – Тестовий випадок з використанням Selenium

Як можна бачити із цих прикладів, Selenide не вимагає очікування у явному вигляді, на відміну від Selenium.

3.3 Використання AssertJ

Cucumber не постачається з бібліотекою тверджень, а тверджень, що пропонує Selenide, не достатньо для повноцінного функціонування фреймворку автоматизації. Добре, що разом із Cucumber та Selenium можна вільно використовувати будь-яку бібліотеку тверджень на вибір. В даній кваліфікаційній роботі скористаємось AssertJ.

3.3.1 Підключення AssertJ до проєкту

AssertJ підключається до Maven проєкту за допомогою додавання залежності в pom файл (рисунок 17).

```
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.24.2</version>
    <scope>test</scope>
</dependency>
```

Рисунок 17 – Залежність, що додається у pom файл
для підключення AssertJ

3.3.2 Порівняння AssertJ з JUnit на прикладі

AssertJ [22] – це бібліотека Java, яка надає багатий набір тверджень і корисних повідомлень про помилки, покращує читабельність тестового коду та є надзвичайно простою у використанні.

Для прикладу порівняємо твердження перевірки наявності певного тексту в посиланні за допомогою JUnit та за допомогою AssertJ.

З використанням JUnit:

```
assertTrue(WebDriverRunner.url().contains(searchWord.toLowerCase()));
```

З використанням AssertJ:

```
assertThat(WebDriverRunner.url()).containsIgnoringCase(searchWord);
```

Як можна побачити, AssertJ має наступні переваги:

- Скориставшись методом containsIgnoringCase вже не потрібно застосовувати метод toLowerCase для слова. Це не єдине таке застосування, бо AssertJ має багатий набір різноманітних варіантів порівнянь, наприклад, isCloseTo, isOdd, isBetween, matches та інші. Вичерпний перелік таких методів можна знайти в документації [22].
- Під час падіння перевірок з використанням AssertJ повідомлення про помилку будуть давати вичерпну відповідь, тобто будуть в собі містити причину цього падіння, який був очікуваний результат, а який актуальний. Водночас падіння з використанням JUnit будуть лише повідомляти нам, що очікувалось true, а отримали false.
- В решті решт це читабельність та розуміння коду. Доволі зручно, коли параметром методу assertThat завжди є реальний результат і ми його порівнюємо з очікуваним за допомогою окремого методу (containsIgnoringCase в нашому випадку) і не думаємо в якій послідовності ставити очікуваний і актуальний результат, наприклад, як в методі assertEquals із JUnit.

3.4 Використання Jenkins

3.4.1 Встановлення Jenkins

Jenkins можна встановити за допомогою менеджера пакетів Homebrew.

Для встановлення Homebrew було виконано наступну команду в терміналі (рисунок 18).

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Рисунок 18 – Команда для встановлення Homebrew

Для встановлення останньої версії Jenkins в терміналі було виконано команду brew install jenkins-lts.

Для запуску Jenkins сервісу було виконано команду brew services start jenkins-lts.

Після запуску сервісу Jenkins потрібно перейти на http://localhost:8080 (рисунок 19), щоб завершити встановлення [23].

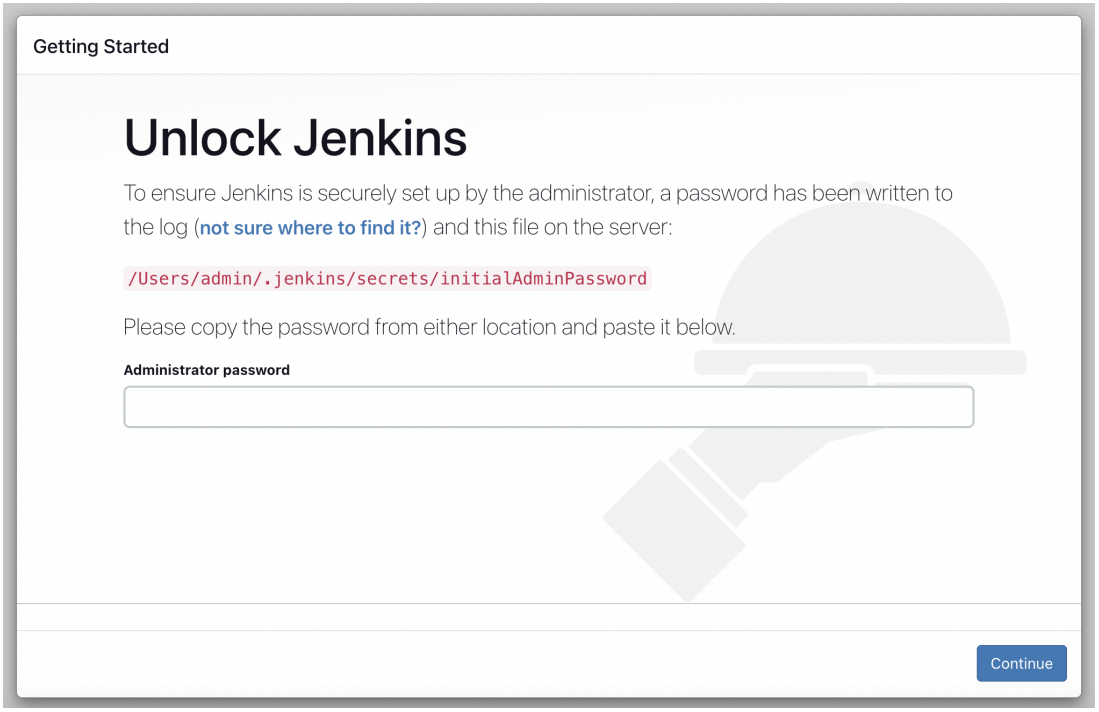

Рисунок 19 – Сторінка, яка буде відображена за посиланням http://localhost:8080 на даному етапі встановлення

Якщо скопіювати виділений червоним текст і в терміналі використати команду cat, можна отримати початковий пароль.

Після використання отриманого пароля на сторінці Unlock Jenkins можна переходити до налаштування Jenkins і встановлювати плагіни (рисунок 20). Виберемо «Install suggested plugins», щоб встановити найпопулярніші плагіни, які пропонують розробники Jenkins.

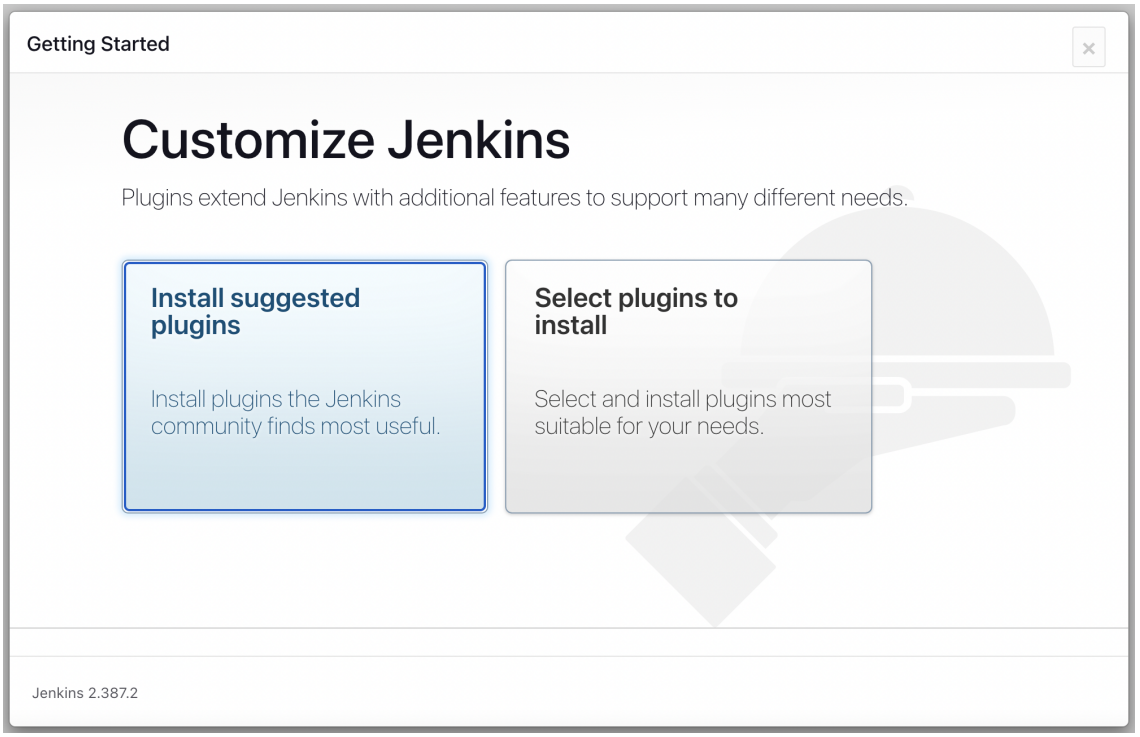

Рисунок 20 – Сторінка, з якої можна розпочати встановлення плагінів для Jenkins

Після вибору опції «Install suggested plugins» розпочнеться процес встановлення плагінів (рисунок 21).

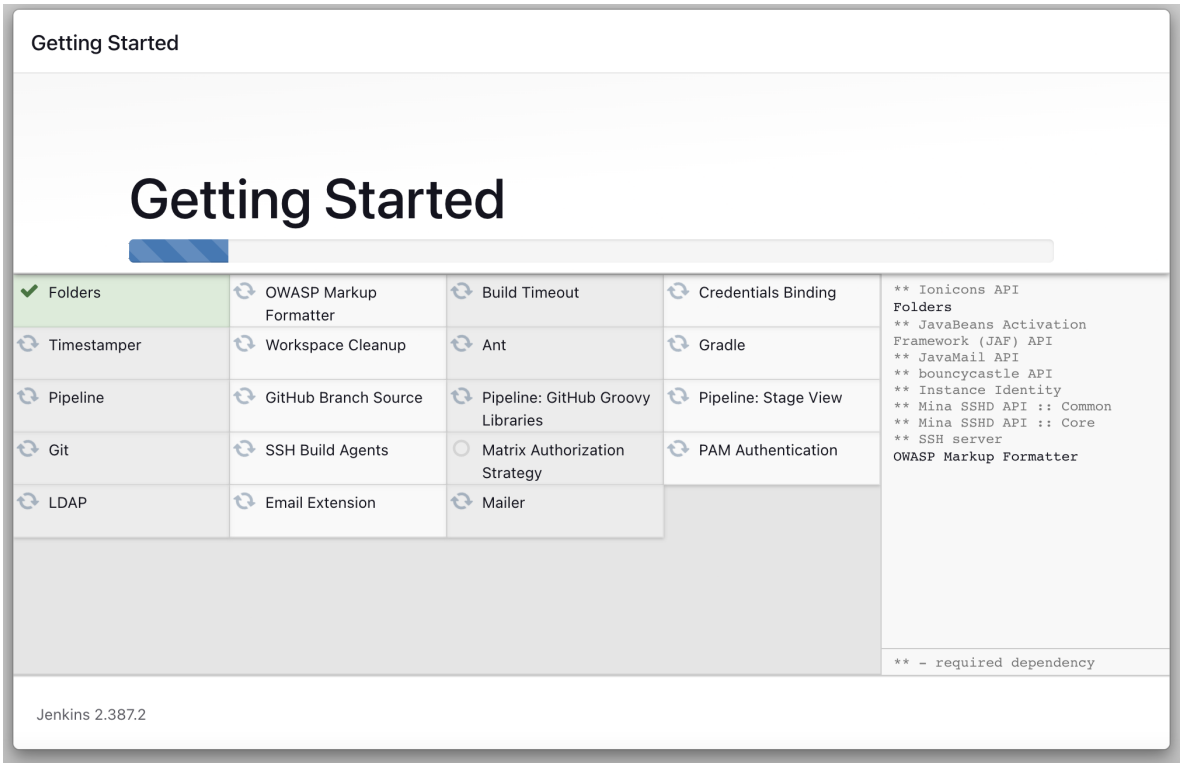

Рисунок 21 – Процес встановлення Jenkins плагінів

Після завершення процесу встановлення плагінів потрібно створити першого користувача-адміністратора. Далі визначити посилання для Jenkins. Оскільки в даній роботі буде використовуватись Jenkins лише на локальній машині, то лишимо посилання http://localhost:8080/.

Після цього початкове налаштування Jenkins завершено.

3.4.2 Запуск Cucumber сценаріїв за допомогою командного рядка

Перш ніж розпочинати конфігурування Jenkins Job для запуску Cucumber сценаріїв, потрібно налаштувати можливість запуску Cucumber сценаріїв за допомогою командного рядка. На Jenkins нема IntelliJ IDEA, яка буде автоматично підставляти потрібні плагіни для запуску тестових сценаріїв. Jenkins оперує лише консольними командами роботи із Maven, тому потрібно

вручну додати потрібні плагіни до pom.xml файлу та створити Runner, за допомогою якого можна запускати тестові випадки з командного рядка.

Для запуску Cucumber сценаріїв за допомогою командного рядка було додано org.apache.maven.plugins:maven-surefire-plugin плагін, а також залежності io.cucumber:cucumber-junit та junit:junit. Зауважимо, що всі залежності io.cucumber в межах одного проєкту мають мати однакову версію.

Також було додано CucumberRunnerTest (рисунок 22) для запуску тестових сценаріїв. Зауважимо, що для того, щоб під час виконання команди mvn test Maven розпізнав клас як ранер, потрібно, щоб назва класу закінчувалась словом Test.

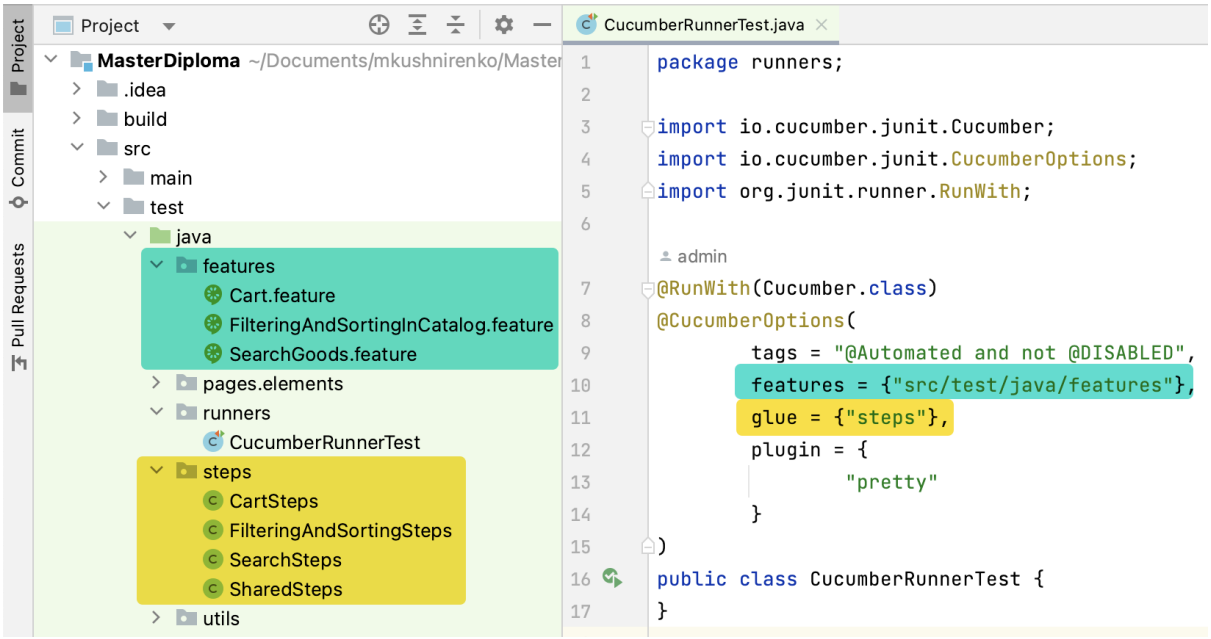

Рисунок 22 – Клас CucumberRunnerTest

Параметр tags є своєрідним способом фільтрації тестових випадків по тегу. Він визначає якими тегами потрібно чи не потрібно запускати тестові випадки незалежно від положення сценарію в контексті feature файлів та директорій.

Параметр features відповідає за те, з якої директорії чи конкретного feature файлу потрібно запускати тестові сценарії.

Параметр glue відповідає за те, з якої директорії чи конкретного класу потрібно підставляти імплементацію кроків, за допомогою яких були описані тестові сценарії, які містяться у feature файлах, зазначених у параметрі features.

Параметр plugin вказує які конкретні плагіни мають бути використані для даного запуску, наприклад, це може бути підключення Allure Report, деталі якого будуть описані в наступних підрозділах. А поки можна додати простий Cucumber плагін pretty, який виводить в консоль всі виконувані Cucumber кроки та коментарем їхню імплементацію (рисунок 23).

```
@Automated
Scenario: Verify that search results contains search word # src/test/java/features/SearchGoods.feature:12
  Given I open ROZETKA home page                          # steps.SharedSteps.openHomePage()
  When I set iPhone value in Search field and press Enter # steps.SearchSteps.setValueInSearchFieldAndPressEnter(java.lang.String)
  Then I verify that search result contains iPhone word   # steps.SearchSteps.assetThatSearchResultsContainsSearchWord(java.lang.String)
```

Рисунок 23 – Вивід консолі з використанням Cucumber плагіна pretty

3.4.3 Конфігурування Jenkins Job

У цьому підрозділі буде описано створення та конфігурування Jenkins Job, з допомогою якої можна запускати Cucumber сценарії з визначеного файлу чи директорії та з певним фільтром по тегах.

Для того, щоб налаштувати Jenkins Job для запуску автоматизованих сценаріїв, для початку її потрібно створити. Для цього потрібно обрати New item пункт на головній сторінці Jenkins. Ввести назву проєкту, в нашому випадку це буде RunFeatureFilesByFolder, та обрати тип Freestyle project і натиснути кнопку «OK».

У результаті вищеописаних дій користувач опиняється на сторінці конфігурування новоствореної RunFeatureFilesByFolder Job.

На цій сторінці можна обрати багато різноманітних параметрів, причому список наявних параметрів можна значно розширити за допомогою Jenkins плагінів.

У налаштуваннях додамо опис, який коротко описує для чого буде використовуватися даний проєкт (рисунок 24).

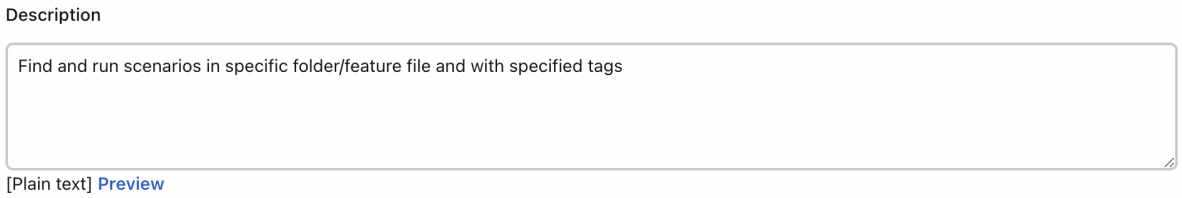

Рисунок 24 – Налаштування опису Jenkins Job

Також варто додати параметри, згідно з якими будуть видалятися старі збірки, щоб не перевантажувати сховище неактуальними даними. Нижче наведена конфігурація означає, що збірки будуть зберігатися не більше семи днів, а максимальна кількість збірок не має перевищувати десяти (рисунок 25).

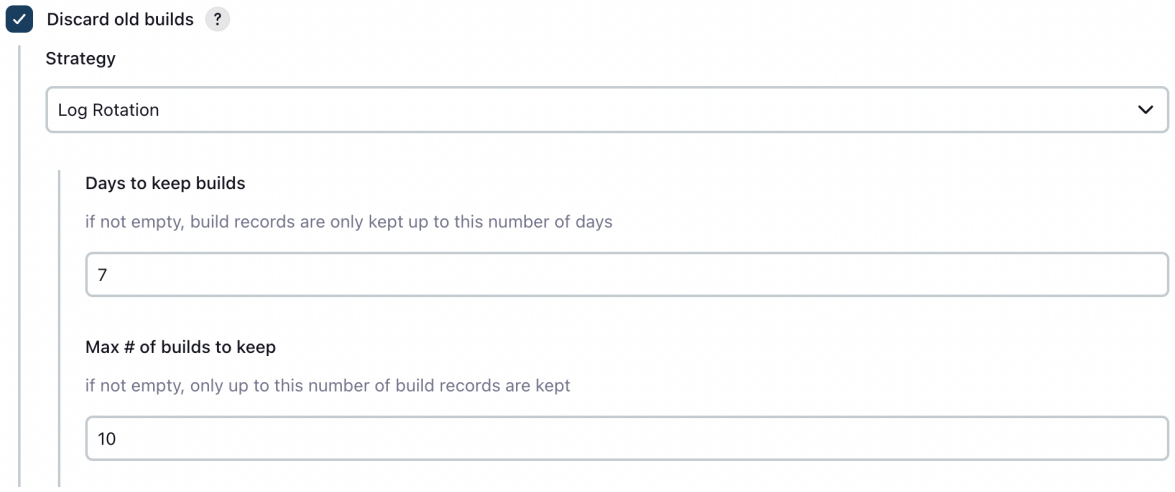

Рисунок 25 – Налаштування автоматичного видалення застарілих збірок

Додамо параметр TEST_DIR типу вибору зі списку, за допомогою якого можна обирати з якого файлу чи директорії потрібно запускати тестові сценарії (рисунок 26).

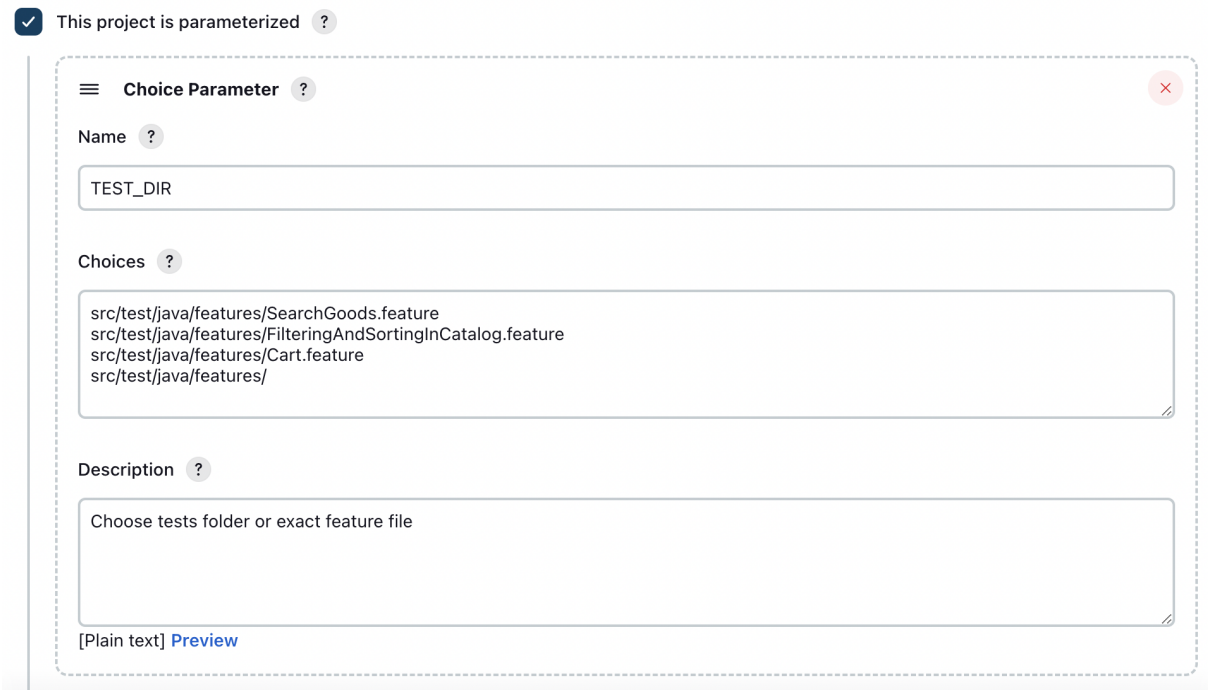

Рисунок 26 – Додавання параметра типу вибору зі списку TEST_DIR

Параметризуємо можливі фільтри для запуску сценаріїв по значеннях тегів за допомогою рядкового параметра TAG (рисунок 27).

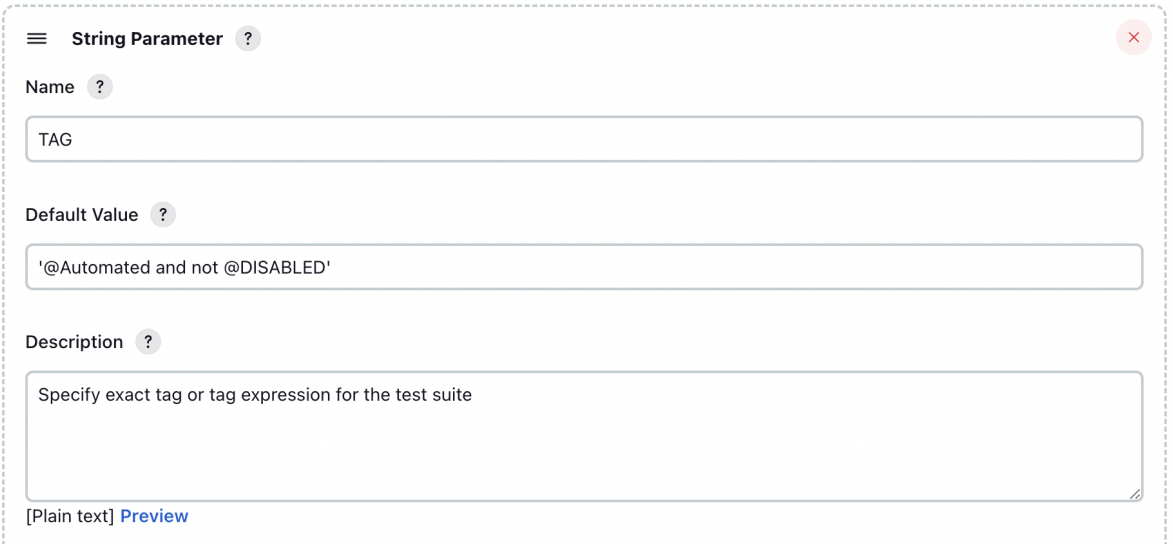

Рисунок 27 – Додавання рядкового параметра TAG

Визначимо, з якого репозиторію Git та якої гілки мають запускатися сценарії. Оскільки для даної роботи використовується публічний репозиторій, то вказувати креденшионали не потрібно (рисунок 28).

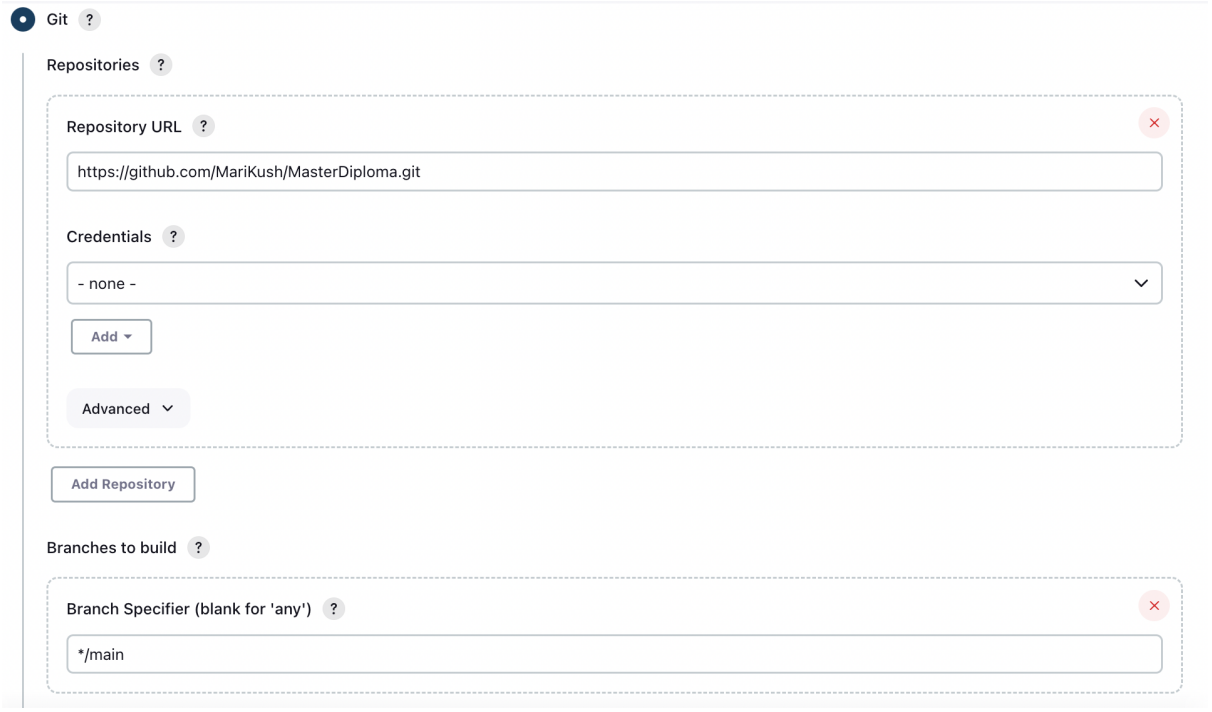

Рисунок 28 – Додавання інформації про Git репозиторій

Налаштуємо щогодинний автоматичний запуск тестових сценаріїв за допомогою cron виразу (рисунок 29).

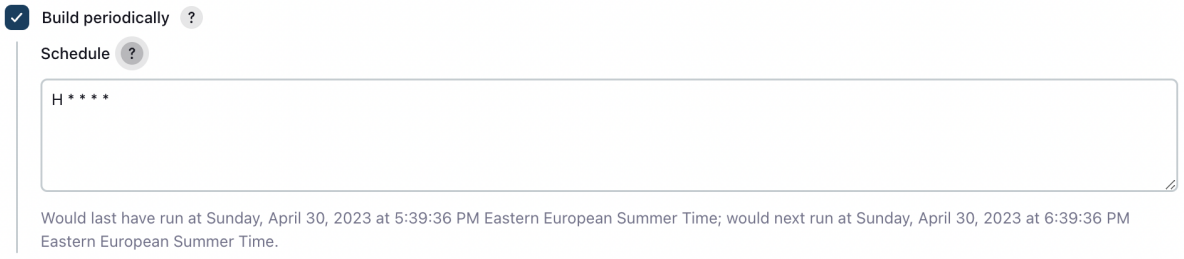

Рисунок 29 – Налаштування щогодинного запуску

Визначимо умову і час, після яких збірка має бути автоматично припинена у зв'язку з підозрою на зависання (рисунок 30).

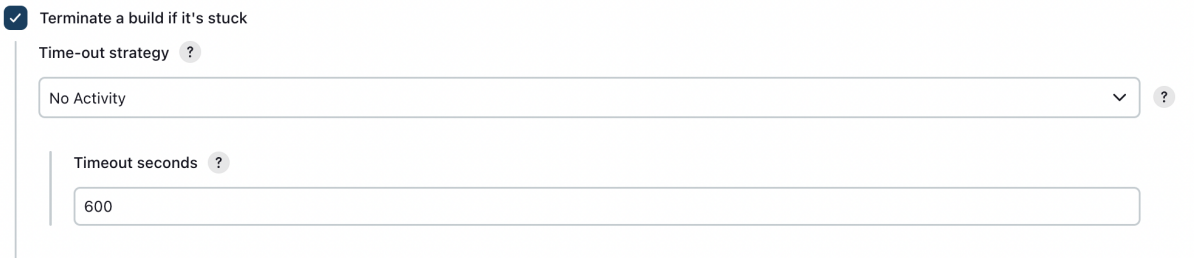

Рисунок 30 – Умови, за яких збірка має бути припинена

Врешті-решт, щоб запрацювала вся вищеописана конфігурація потрібно додати виклик Maven команд. Перед цим потрібно в конфігураційних інструментах визначити версію Maven і ця версія має збігатися із законфігурованою версією в Jenkins Job (рисунок 31).

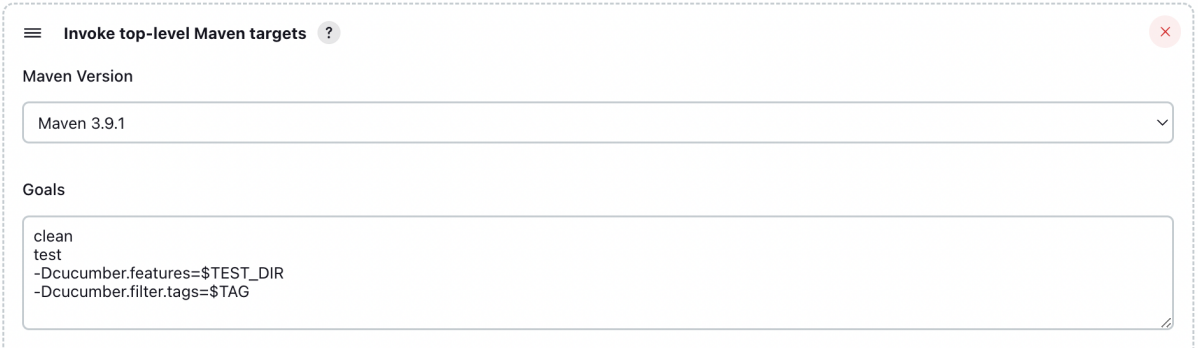

Рисунок 31 – Команди Maven для запуску тестових сценаріїв із параметрами

Як можна побачити з рисунка, в командах Maven ми спершу очищаємо попередні збірки, потім викликаємо команду для запуску тестів, куди передаємо вище законфігуровані динамічні параметри, а саме: визначення тестової директорії та набору тегів.

3.4.4 Запуск Jenkins Job

Коли конфігурація готова, то для запуску тестів потрібно відкрити RunFeatureFilesByFolder Jenkins Job та вибрати пункт меню «Build with Parameters». На сторінці вибору параметрів для запуску можна за бажанням змінити параметри (рисунок 32). Коли фінальна версія параметрів обрана, потрібно натиснути кнопку «Build», після натискання даної кнопки відбудеться запуск сконфігурованих Maven команд.

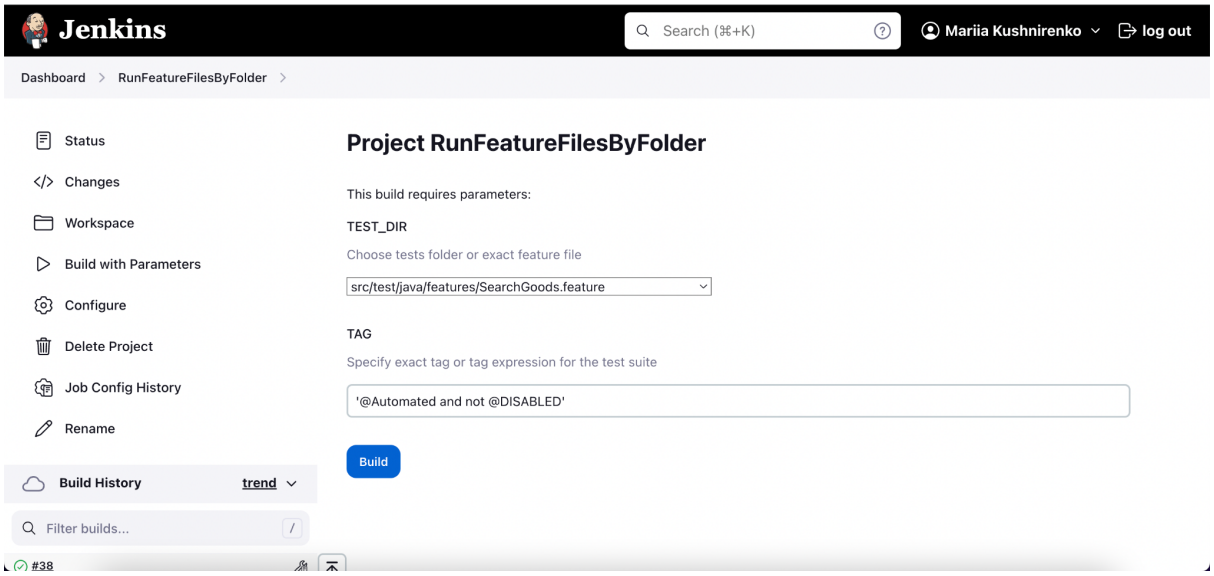

Рисунок 32 – Вибір параметрів перед запуском Jenkins Job

Найпростіший спосіб подивитися результати запуску – це в обраному запуску відкрити пункт меню «Console Output» і проаналізувати консоль логи. Оскільки в CucumberRunnerTest було додано плагін pritty, то в консолі можна побачити які тести запускалися, з якими кроками, які методи використовують ці кроки. Також там можна знайти інформацію про кількість запущених, пропущених, вдалих і невдалих тестів. У випадку падіння там можна побачити трасування стека.

3.4.5 Інтеграція Jenkins із Slack

Для інтеграції Jenkins із Slack потрібно інсталювати Global Slack Notifier Plugin [24] в Jenkins.

Для автентифікації в Slack за допомогою цього плагіна потрібно створити власну спеціальну «Slack App».

Для створення програми потрібно перейти за посиланням https://api.slack.com/apps і натиснути «Create an App».

Вибрати назву програми, наприклад «Jenkins», і робочу область, у яку її буде встановлено.

Після натискання «Create App» потрібно обрати пункт «Permissions», щоб перейти на сторінку «OAuth & Permissions».

Потім у підрозділі «Scopes» додати chat:write.

У верхній частині сторінки потрібно натиснути «Install to Workspace», це згенерує необхідний токен для Slack аутентифікації в Jenkins. Потрібно зберегти цей токен.

Тепер треба перейти в Jenkins і відкрити конфігурацію Slack у «Manage Jenkins → Configure System».

У полі Credential потрібно додати нові креденшионали із типом «Secret text» і значенням токена із попереднього кроку та дати їм назву. У нашому випадку назва буде slack-bot-tocken.

Також потрібно визначити ідентифікатор каналу/учасника за замовчуванням, це буде канал #rozetka-test-results і обов'язково відмітити пташкою пункт «Custom slack app bot user» (рисунок 33).

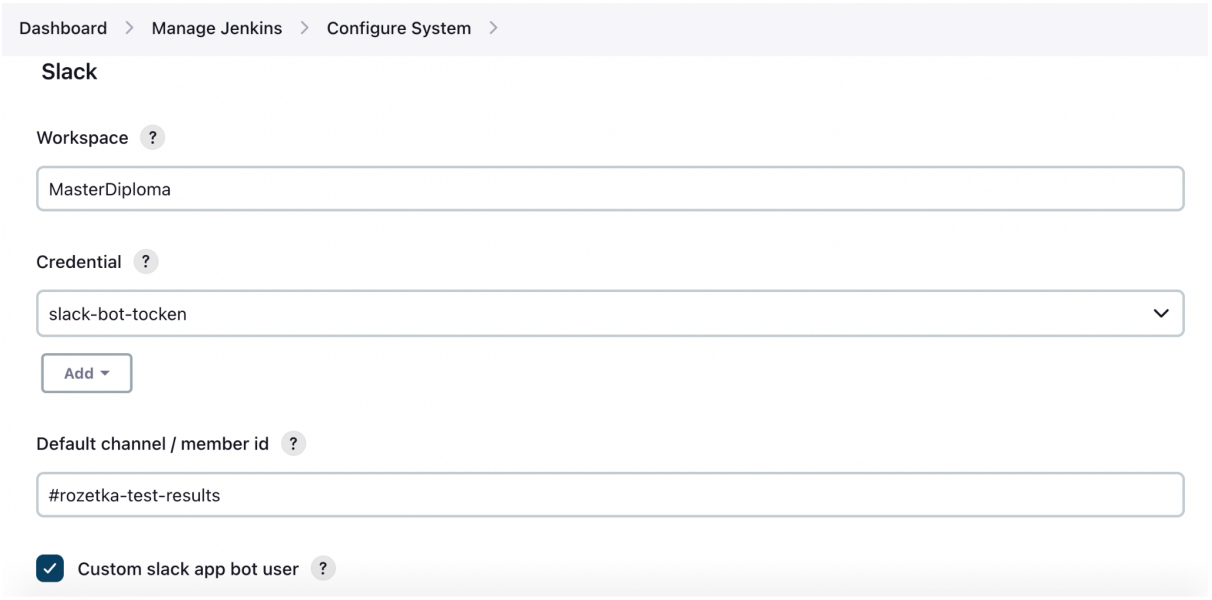

Рисунок 33 – Конфігурування Jenkins для роботи з Slack

У Slack потрібно запросити Jenkins App в канал, у який мають надходити сповіщення. Після цього перевірка з'єднання у Jenkins має бути «Success», у зазначений Slack канал комунікації за замовчуванням має бути надіслане тестове повідомлення.

Тепер можна додати інтеграцію із Slack у RunFeatureFilesByFolder Jenkins job. Для цього потрібно відкрити конфігурацію і в розділі «Post-build Actions» додати Slack Notifications.

Там можна вказати, якими мають бути причини надсилання повідомлення, текст та отримувача повідомлення. Причиною надсилання повідомлення було обрано падіння або успіх запуску. В текст додано значення параметрів із якими відбувся конкретний запуск. Отримувачем є канал rozetka-test-results.

Результати даної конфігурації можна побачити в каналі rozetka-test-results (рисунок 34).

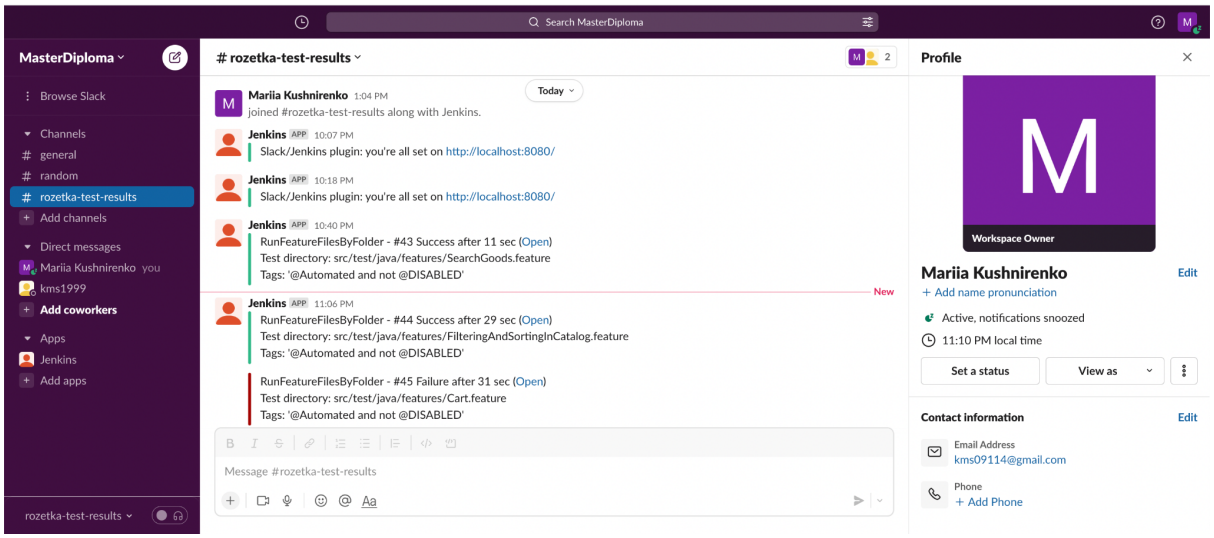

Рисунок 34 – Jenkins надсилає повідомлення у канал Slack

3.5 Використання Allure Report

3.5.1 Встановлення Allure

Для Mas OS автоматичне встановлення Allure доступне за допомогою Homebrew. Для цього потрібно виконати команду brew install allure.

Перевірити чи успішно було встановлено Allure можна за допомогою команди allure --version.

3.5.2 Підключення Allure для роботи з Cucumber 7

У залежності від версії Cucumber потрібно додавати різні версії бібліотек для роботи Allure із Cucumber. Оскільки в даній роботі використовується найновіша 7 версія Cucumber, то і версія Allure бібліотеки повинна мати відповідний індекс в назві.

Для підключення Allure для Cucumber 7 потрібно додати io.qameta.allure:allure-cucumber7-jvm залежність та розширити наявний

org.apache.maven.plugins:maven-surefire-plugin плагін конфігурацією argLine з aspectj javaagent та залежністю org.aspectj:aspectjweaver у pom файлі.

Варто зауважити, що Allure несумісний з найновішими мінорними версіями Cucumber 7. Тому для того, щоб запрацювала конфігурація, довелось понизити версію Cucumber із 7.11.1 до 7.2.3.

Крім цього в CucumberRunnerTest потрібно додати плагін io.qameta.allure.cucumber7jvm.AllureCucumber7Jvm.

Додавання знімків екрана у випадку падіння тестових випадків не є влаштованою функцією Allure Report. Тому, щоб забезпечити дану логіку необхідно дописати метод, який це реалізує.

Додамо метод attachScreenshotIfFailed з анотацією After у клас SharedSteps (рисунок 35). Анотація After означатиме, що даний метод буде викликатися після кожного сценарію.

```
@After
public void attachScreenshotIfFailed(Scenario scenario) {
    if (scenario.isFailed()) {
        byte[] screenshot = ((TakesScreenshot)WebDriverRunner.getWebDriver()).getScreenshotAs(OutputType.BYTES);
        Allure.addAttachment( name: "Failed Screenshot", new ByteArrayInputStream(screenshot));
    }
}
```

Рисунок 35 – Метод, який додає зображення екрана в Allure Report, за умови падіння сценарію

Тепер для всіх наступних запусків CucumberRunnerTest буде генеруватися Allure Report. Він буде зберігатися у новоствореній директорії allure-results, яка знаходиться в корені проєкту. Щоб відкрити Allure Report, потрібно запустити команду allure serve allure-results із кореневої директорії проєкту і як результат цієї команди Allure Report згенерується і відкриється у браузері, який є браузером ПК за замовчуванням.

3.5.3 Підключення Allure в Jenkins

Для того, щоб інтегрувати Allure Report в Jenkins, спершу потрібно інсталювати плагін Allure для Jenkins.

Потім потрібно відкрити Global Tool Configuration і в налаштуваннях додати Allure Commandline, причому його версія має збігатися із версією Allure в pom файлі.

Тепер в конфігурації RunFeatureFilesByFolder секції Post-build Actions можна додати Allure Report.

У результаті даної конфігурації для кожного запуску цієї Jenkins job буде генеруватися Allure Report (рисунок 36). Розбір прикладу Allure Report буде наведено в наступному підрозділі.

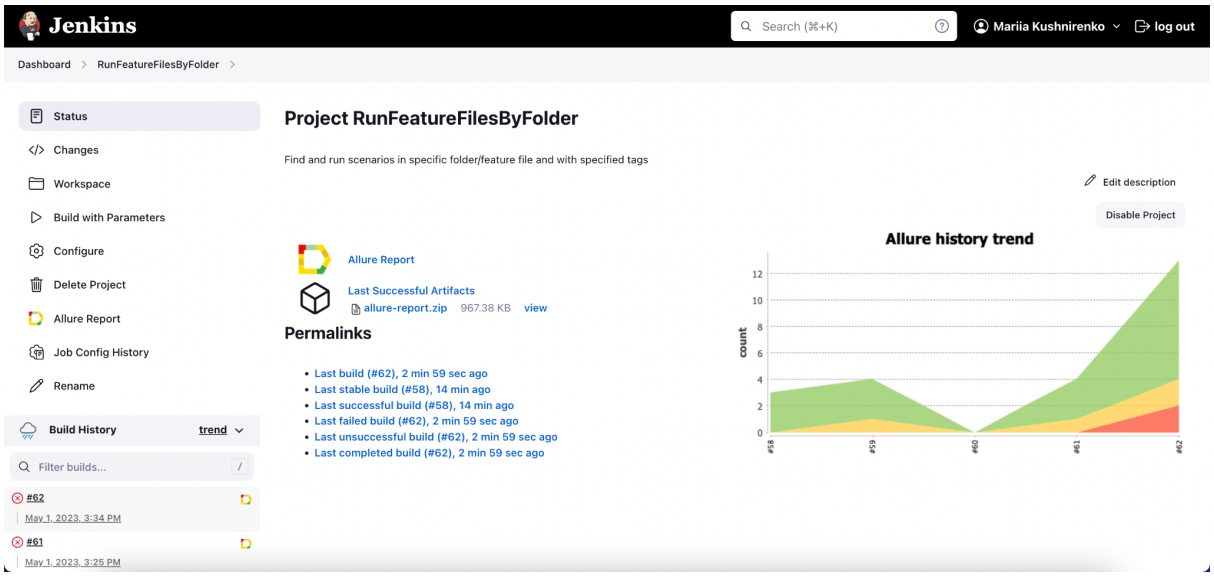

Рисунок 36 – Вигляд RunFeatureFilesByFolder після додавання Allure Report

3.5.4 Огляд згенерованого Allure Report

Типовий Allure Report складається з вкладки «Overview», панелі навігації, кількох вкладок для різних типів представлення тестових даних і сторінок

тестових випадків для кожного окремого тесту. Кожен Allure Report підтримується деревоподібною структурою даних, яка представляє процес виконання тесту. Всі деревоподібні представлення, включаючи «Categories», «Behaviors» і «Packages», підтримують фільтрацію та сортування.

Сторінка «Overview» є початковою сторінкою кожного звіту, на ній можна побачити різноманітні інформаційні панелі (рисунок 37).

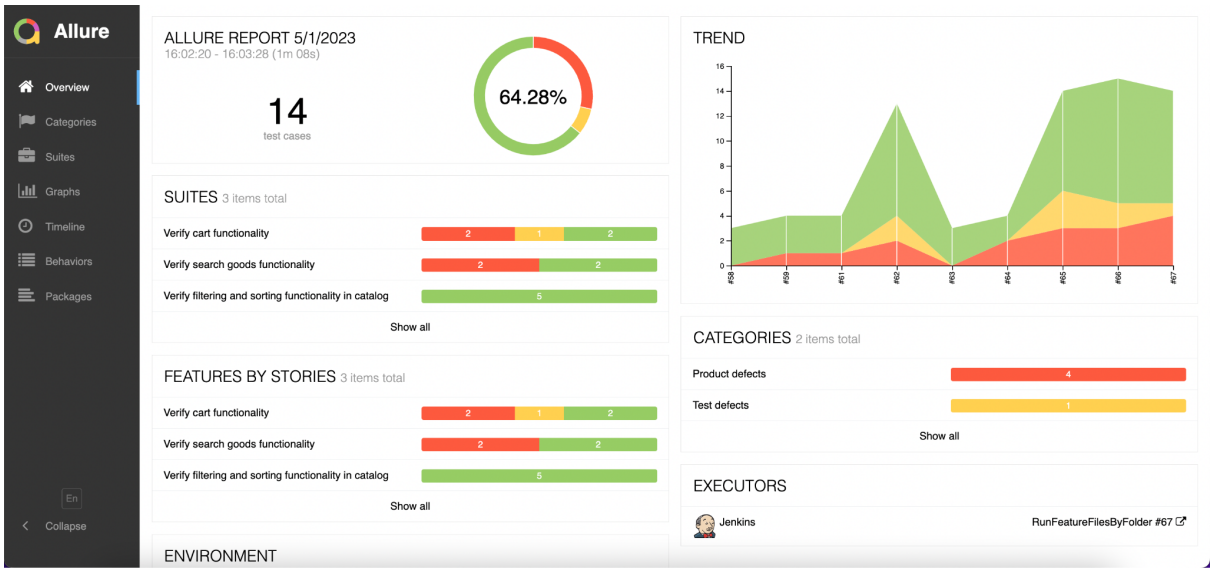

Рисунок 37 – Сторінка «Overview» Allure Report

Сторінка «Overview» містить кілька стандартних віджетів, що представляють основні характеристики тестового середовища: загальну статистику звіту, результати тестів відповідно до файлу, в якому вони знаходяться, історичний тренд на графіку і тому подібне. Віджети на цій сторінці можна перетягувати та налаштовувати.

Вкладка «Categories» класифікує дефекти, цю класифікацію зручно використовувати при розборі результатів тестування. Її можна налаштувати для власних потреб шляхом додавання файлу categories.json в директорію allure-results перед генеруванням звіту.

Як можна побачити на рисунку 38, в нашому запуску є продуктові та тестові дефекти. Перша категорія − це потенційні проблеми в системі, а друга − це скоріш за все проблеми із кодом автоматизації.

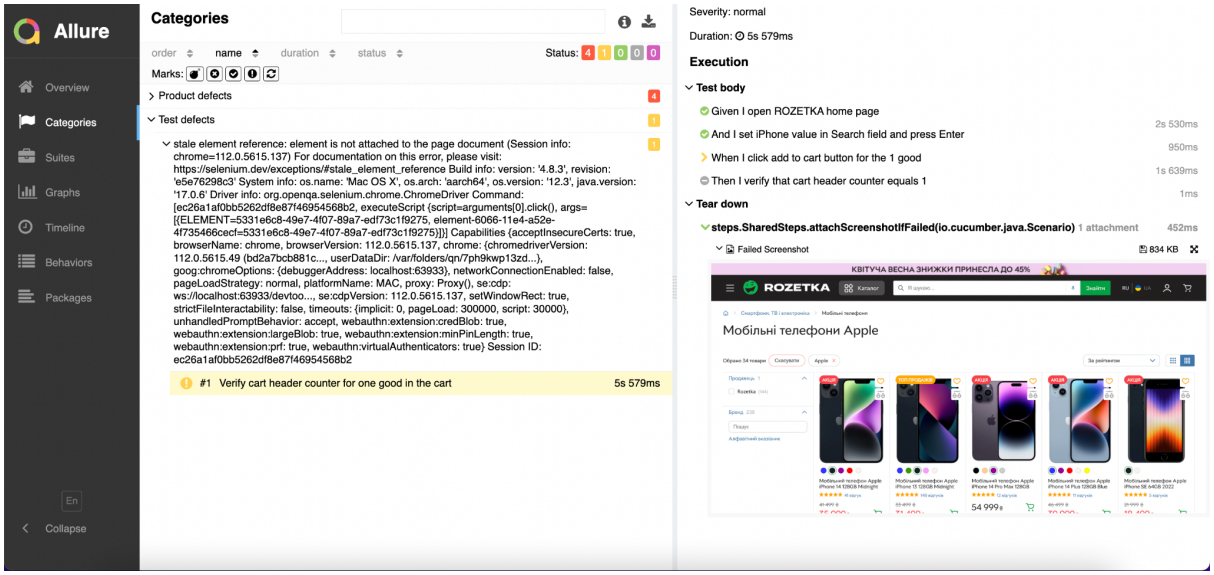

Рисунок 38 − Вкладка «Categories» Allure Report

Зі сторінок, де є посилання на конкретний тестовий сценарій можна перейти на сторінку тестового випадку, натиснувши на його посилання. В результаті праворуч від основного вмісту сторінки відкриється підсторінка, яка міститиме наступні дані пов’язані з тестом: кроки, виконані під час тесту, час, вкладення, мітки категоризації тесту, описи та посилання. Також у випадку падіння тестового сценарію до звіту додається знімок екрана браузера у момент падіння тесту (рисунок 38).

На сторінці «Suites» можна побачити структурне представлення виконаних тестів відповідно до файлів в яких вони знаходяться (рисунок 39).

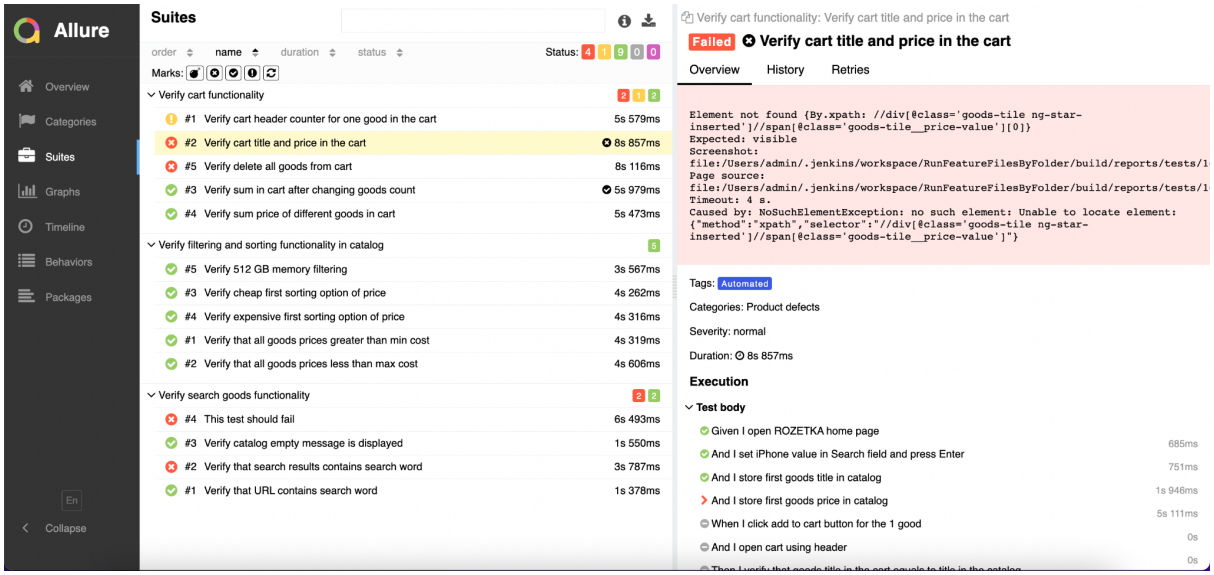

Рисунок 39 – Вкладка «Suites» Allure Report

Графіки дозволяють переглядати різні статистичні дані, зібрані з даних тестування: розподіл статусів, діаграми тривалості й тому подібне (рисунок 40).

Рисунок 40 – Вкладка «Graphs» Allure Report

Вкладка «Timeline» (рисунок 41) візуалізує ретроспективу виконання тестів, адаптери Allure збирають точні часові дані тестів, на цій вкладці вони впорядковані відповідно до їхньої послідовної або паралельної структури синхронізації.

Рисунок 41 − Вкладка «Timeline» Allure Report

Для підходу, керованого поведінкою, «Behaviors» вкладка групує результати тестування відповідно до тегів Epic, Feature та Story.

Вкладка «Packages» представляє деревоподібний макет результатів тестування, згрупованих за різними пакетами.

Вкладки «Behaviors» та «Packages» будуть подібні до вкладки «Suites» у нашому випадку.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було:

- уточнено сенс понять контролю та забезпечення якості;
- класифіковано інструменти автоматизації тестування;
- наведено приклади найактуальніших інструментів автоматизації тестування із кожної із категорій;
- обрано найпопулярніші інструменти й описано їхні можливості та особливості;
- розроблено власний фреймворк автоматизації тестування з використанням розглянутих інструментів.

При розробці системи була використана ітераційна модель. Інструментарієм було обрано інтегроване середовище розробки IntelliJ IDEA, мова програмування Java, засіб збірки Maven, інструмент, який підтримує розробку, орієнтовану на поведінку Cucumber, засіб автоматизації вебзастосунків Selenide, бібліотека тверджень AssertJ, інструмент CI/CD Jenkins, інструмент звітування про результати тестування Allure.

Популярність гнучких методологій розробки спонукає до активного розвитку інструментів автоматизованого тестування. Розроблений фреймворк автоматизації тестування може застосовуватися як основа під час впровадження автоматизації тестування вебзастосунків на будь-якому проєкті. Надалі можливе доопрацювання даної системи і її розширення шляхом розгортання Jenkins на віддаленому сервері, інтегруванням з системами управління тестовими випадками, відстеження дефектів та інструментами тестування мобільних пристроїв.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Certified Tester. Foundation Level (CTFL) Syllabus. Version 2018 v3.1.1. – 93с.
2. Ron Patton. Software Testing – Sams Publishing, 2nd edition, 2005. – 408с.
3. Lisa Crispin. Agile Testing: A Practice Guide for Testers and Agile Teams, 1st Edition, 2009. – 577с.
4. dotcom tools. Web Application Testing Tools [Електронний ресурс] – Режим доступу: https://www.dotcom-tools.com/web-application-testing
5. QA Lead. 10 Best Mobile Application Testing Tools in 2023[Електронний ресурс] – Режим доступу: https://theqalead.com/tools/best-mobile-application-testing-tools/
6. QA Lead. 10 Best Security Testing Tools For QA In 2023 [Електронний ресурс] – Режим доступу: https://theqalead.com/tools/best-security-testing-tools/
7. HiTechNectar. Top 9 Infrastructure Automation Tools [Електронний ресурс] – Режим доступу: https://www.hitechnectar.com/blogs/infrastructure-automation-tools/
8. QA Lead. 10 Best Test Management Tools Of 2023 [Електронний ресурс] – Режим доступу: https://theqalead.com/tools/best-test-management-tools/
9. QA Lead. 10 Best Defect Tracking Tools in 2023 [Електронний ресурс] – Режим доступу: https://theqalead.com/tools/defect-tracking-tools/
10. Testomat.io. 19 Best Report Testing Tools in 2023. Comprehensive List [Електронний ресурс] – Режим доступу: https://testomat.io/blog/best-report-testing-tools-in-comprehensive-list/

11. QA Lead. 10 Best Load Testing Tools For Web Applications In 2023 [Електронний ресурс] – Режим доступу: https://theqalead.com/tools/load-testing-tools/
12. Офіційний сайт Cucumber [Електронний ресурс] – Режим доступу: https://cucumber.io/
13. Офіційний сайт RestAssured [Електронний ресурс] – Режим доступу: https://rest-assured.io/
14. GitHub. README.md для AssertJ [Електронний ресурс] – Режим доступу: https://github.com/assertj/assertj
15. Офіційний сайт Selenide [Електронний ресурс] – Режим доступу: https://selenide.org/
16. QualityAssuranceGroup. Переваги Selenide [Електронний ресурс] – Режим доступу: https://qagroup.com.ua/publications/benefits-selenide/
17. BugRaptors. Selenide Vs. Selenium – A Detailed Comparison [Електронний ресурс] – Режим доступу: https://www.bugraptors.com/blog/selenide-vs-selenium
18. XUL. Ajax Tutorial [Електронний ресурс] – Режим доступу: https://www.xul.fr/en-xml-ajax.html
19. Testim. How to Put Quality in the Build With Jenkins Test Automation [Електронний ресурс] – Режим доступу: https://www.testim.io/blog/jenkins-test-automation/
20. Офіційний сайт Jenkins [Електронний ресурс] – Режим доступу: https://www.jenkins.io/
21. QA Meta. Allure Framework [Електронний ресурс] – Режим доступу: https://docs.qameta.io/allure/
22. Документація з AssertJ [Електронний ресурс] – Режим доступу: https://assertj.github.io/doc/

23. MacMiniVaul. Installing Jenkins on macOS [Електронний ресурс] – Режим доступу: https://www.macminivault.com/installing-jenkins-on-macos/
24. YouTube. How to Send Slack Notifications From Jenkins [Електронний ресурс] – Режим доступу: https://www.youtube.com/watch?v=EDVZli8GdUM&ab_channel=CloudBeesTV