МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление 02.03.01 Математика и компьютерные науки

Отчет по дисциплине

«Методы тестирования программного обеспечения»

«Автоматизированное тестирование»

Студент: гр. 5130201/10101 Гуй Цици

Преподаватель: Курочкин Михаил Александрович

Санкт-Петербург, 2024

Содержание

Введение

В лабораторных работах по автоматизированному тестированию требуется:

- С помощью фреймворка JUnit разработать unit-тесты к методам предоставленного класса Калькулятор. Необходимо протестировать 4 метода калькулятора;
- Используя фреймворк TestNG и методы библиотеки Selenium WebDriver реализовать 2 теста для предоставленного сайта в соответствии с заданием;
- Выполнить задания из второй лабораторной работы, используя шаблон проектирования тестов Page Object;
- Выполнить задания из третьей лабораторной работы с помощью шаблона Steps и, используя, Jenkins CI, создать задачу, которая будет запускать тесты из данной работы и тест, переделанный так, чтобы он не проходил. Также требуется создавать отчет о выполнении тестирования с помощью Allure Report.

1 Описание автоматизации тестирования

1.1 TestNG

TestNG – это фреймворк для тестирования Java-приложений. Особенности TestNG:

- Поддержка аннотаций;
- Поддержка тестирования интегрированных классов, например, по умолчанию нет необходимости создавать новый экземпляр класса теста для каждого метода тестирования;
- Разделение тестового кода времени компиляции и информации о конфигурации, данных времени выполнения.
- Гибкая конфигурация во время выполнения;
- Наличие «тестовых групп»;
- Поддержка зависимых методов тестирования, параллельного тестирования, нагрузочного тестирования и частичного отказа;
- Гибкий плагин API;
- Поддержка многопоточного тестирования.

Написание теста обычно состоит из трех этапов:

1. Реализация бизнес-логики теста и добавление аннотации TestNG в код;
2. Добавление информации о тесте, например, имя класса, группы, в которой его необходимо запускать и т. д.);
3. Запуск TestNG.

1.2 JUnit

JUnit — это фреймворк для языка программирования Java, предназначенный для автоматического тестирования программ. Его основное назначение — модульное тестирование.

Основные отличия JUnit от TestNG:

- Аннотации Before и After для осуществления действий до и после тестирования, являются общими для методов и классов;
- Группировка происходит отдельной аннотацией Suite, а не через "pom.xml";
- Параметризация через аннотации ParametrizedTest и CsvSource.

1.3 Selenium WebDriver

Selenium WebDriver — библиотека для управления браузерами, основной продукт комплекта Selenium. Представляет из себя семейство драйверов для разных браузеров (Firefox, Edge, Google Chrome/Chromium, Internet Explorer, Safari, Opera) и набор клиентских библиотек на разных языках программирования для работы с драйверами. WebDriver поддерживает работу с языками Java, .Net (C#), Python, Ruby, JavaScript.

WebDriver напрямую отправляет команды браузеру, используя его API и получает результаты тестирования, то есть используется способ взаимодействия с браузером, максимально близкий к действиям обычного пользователя.

Для работы с Webdriver необходимо иметь 3 основных программных компонента:

- Браузер, работу которого пользователь хочет автоматизировать. Это реальный браузер определённой версии, установленный на определённой ОС и имеющий свои настройки;
- Для управления браузером необходим драйвербраузера. Драйвер является веб-сервером, который запускает браузер и отправляет ему команды, а также закрывает его;

- Тест, который содержит набор команд на определённом языке программирования для драйвера браузера.

Основными сущностями в Selenium WebDriver являются:

1. WebDriver – самая важная сущность, ответственная за управление браузером. Основной ход теста строится именно вокруг экземпляра этой сущности;

2. WebElement – представляет собой абстракцию над веб-элементом, например, кнопкой,ссылкой, полем ввода и т.п. WebElement инкапсулирует методы для взаимодействия пользователя с элементами и получения их текущего статуса;

3. By – абстракция над локатором веб-элемента. Этот класс инкапсулирует информацию о селекторе, например, CSS-селектор, а также сам локатор элемента, то есть всю информацию, необходимую для того, чтобы найти нужный элемент на странице.

1.4 Page Object

Page Object - один из наиболее полезных и используемых архитектурных решений в автоматизации. Данный шаблон проектирования помогает инкапсулировать работу с отдельными элементами страницы, что позволяет использовать одни и те же объекты страниц в различных тестах, и, следовательно, позволяет уменьшить количество кода и его поддержку. использовать в различных тестах. Например, если изменился дизайн одной из страниц, то возникает необходимость внести изменения только в соответствующий класс, описывающий эту страницу.

Основные преимущества использование паттерна Page Object:

- Разделение кода тестов и кода описания страниц;

- Объединение всех действий по работе с веб-страницей в одном месте.

Паттерн Page Object в Selenium реализован с помощью библиотеки PageFactory и класса страницы. Page Object представляет собой отдельный класс, содержащий локаторы элементов, методы для работы с ними и конструктор принимающий в качестве параметра объект WebDriver. Методы класса Page Object могут возвращать объекты других Page Object классов.

1.5 Jenkins

Jenkins — программная система с открытым исходным кодом на Java, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения.

Непрерывная интеграция (Continuous Integration, CI) – это процесс разработки программного обеспечения, смысл которого заключается в постоянном соединении рабочих копий в общую линию разработки, и выполнении постоянных автоматизированных сборок проекта для быстрого выявления возможных ошибок и решения интеграционных проблем.

Jenkins позволяет автоматизировать часть процесса разработки программного обеспечения, в котором не обязательно участие человека, обеспечивая функции непрерывной интеграции. Может собирать проекты с использованием Apache Ant и Apache Maven, а также выполнять произвольные сценарии оболочки и пакетные файлы Windows. Сборка может быть запущена разными способами, например, по событию фиксации изменений в системе управления версиями, по расписанию, по запросу на определённый URL, после завершения другой сборки в очереди. Кроме того, возможности Jenkins могут быть расширены с помощью плагинов.

Основные преимущества Jenkins:

- Режим работы сразу в двух и более средах;
- Повышенная надежность развертываемого программного обеспечения;

- Уменьшение ошибок, связанных с человеческим фактором;
- Уменьшение затрат на персонал;
- Упрощение рабочего процесса за счеттого,что не возникает необходимости нанимать дорогостоящую команду опытных специалистов, т.к. с Jenkins справится небольшая группа сотрудников без специальной квалификации.

1.6 Allure Framework

Allure Framework – популярный инструмент построения отчётов автотестов, упрощающий их анализ. Это гибкий и легкий инструмент, который позволяет получить не только краткую информацию о ходе выполнения тестов, но и предоставляет всем участникам производственного процесса максимум полезной информации из повседневного выполнения автоматизированных тестов.

Разработчикам и тестировщикам использование отчетов Allure позволяет сократить жизненный цикл дефекта: падения тестов могут быть разделены на дефекты продукта и дефекты самого теста, что сокращает затраты времени на анализ дефекта и его устранение. Также к отчету могут быть прикреплены логи, обозначены тестовые шаги, добавлены вложения с разнообразным контентом, получена информация о таймингах и времени выполнения тестов.

Кроме того, Allure-отчеты поддерживают взаимодействие с системами непрерывной интеграции и баг-трекинговыми системами, что позволяет всегда держать под рукой нужную информацию о прохождении тестов и дефектах.

Тест-менеджерам Allure дает общее представление о работоспособности проекта, позволяет понять, какая часть проекта покрыта тестами, как сгруппированы дефекты, какова общая динамока качества проекта.

2 Результаты выполнения лабораторных работ

2.1 Лабораторная работа №1

В данной работе было необходимо разработать unit-тесты к предоставленной библиотеке calculator-1.0.jar, которая содержит класс Калькулятор с методами, соответствующими операциям калькулятора. Было необходимо протестировать 4 операции. Тесты должны быть написаны с использованием фреймворка JUnit.

В результате выполнения лабораторной работы было реализовано 8 классов: AddLongTest, AddDoubleTest, DivLongTest, DivDoubleTest, MulLongTest, MulDoubleTest, SubLongTest, SubDoubleTest. Всего было реализовано 96 тестов.

```
class AddDoubleTest extends AbstractCalculatorTest {
    @ParameterizedTest
    @ValueSource(doubles = { 0.0, -1.0, 1.0, 12345.67, -12345.67 })
    void testAdditionWithZero(double input) {
        assertEquals(input, calculator.sum(input, 0.0), DELTA);
    }

    @ParameterizedTest
    @ValueSource(doubles = { -1.0, 1.0, 12345.67, -12345.67 })
    void testAdditionWithSelf(double input) {
        assertEquals(input * 2, calculator.sum(input, input), DELTA);
    }

    @ParameterizedTest
    @CsvSource({ "5.7, 2.3, 3.4", "-2.5, -1.2, -1.3", "0.0, -5.5, 5.5" })
    void testGeneralCorrectness(double expected, double a, double b) {
        assertEquals(expected, calculator.sum(a, b), DELTA);
    }
}
```

```
20 class AddLongTest extends AbstractCalculatorTest {
21     @ParameterizedTest
22     @ValueSource(longs = { 0, -1, 1, 12345, -12345 })
23     void testAdditionWithZero(long input) {
24         assertEquals(input, calculator.sum(input, 0));
25     }
26 
27     @ParameterizedTest
28     @ValueSource(longs = { 0, -1, 1, 12345, -12345 })
29     void testAdditionWithSelf(long input) {
30         assertEquals(input * 2, calculator.sum(input, input));
31     }
32 
33     @ParameterizedTest
34     @CsvSource({ "5, 2, 3", "-2, -1, -1", "0, -5, 5" })
35     void testGeneralCorrectness(long expected, long a, long b) {
36         assertEquals(expected, calculator.sum(a, b));
37     }
38 }
39 class DivDoubleTest extends AbstractCalculatorTest {
40     @ParameterizedTest
41     @ValueSource(doubles = { 0.0, -1.0, 1.0, 12345.67, -12345.67 })
42     void testDivisionByZero(double input) {
43         assertThrows(NumberFormatException.class, () -> calculator.div(input, 0.0)
       );
44     }
45 
46     @ParameterizedTest
47     @ValueSource(doubles = { -1.0, 1.0, 12345.67, -12345.67 })
48     void testDivisionByOne(double input) {
49         assertEquals(input, calculator.div(input, 1.0), DELTA);
```

```
50      }
51
52      @ParameterizedTest
53      @CsvSource({ "4.0, 20.0, 5.0", "-6.0, 36.0, -6.0", "-9.0, -72.0, 8.0" })
54      void testGeneralCorrectness(double expected, double a, double b) {
55          assertEquals(expected, calculator.div(a, b), DELTA);
56      }
57  }
58  class DivLongTest extends AbstractCalculatorTest {
59      @ParameterizedTest
60      @ValueSource(longs = { 0, -1, 1, 12345, -12345 })
61      void testDivisionByZero(long input) {
62          assertThrows(NumberFormatException.class, () -> calculator.div(input, 0));
63      }
64
65      @ParameterizedTest
66      @ValueSource(longs = { 0, -1, 1, 12345, -12345 })
67      void testDivisionByOne(long input) {
68          assertEquals(input, calculator.div(input, 1));
69      }
70
71      @ParameterizedTest
72      @CsvSource({ "4, 20, 5", "-6, 36, -6", "-9, -72, 8" })
73      void testGeneralCorrectness(long expected, long a, long b) {
74          assertEquals(expected, calculator.div(a, b));
75      }
76  }
77  class MulDoubleTest extends AbstractCalculatorTest {
78      @ParameterizedTest
79      @ValueSource(doubles = { 0.0, -1.0, 1.0, 12345.67, -12345.67 })
80      void testMultiplicationWithZero(double input) {
```

```
81          assertEquals(0.0, calculator.mult(input, 0.0), DELTA);
82      }
83
84      @ParameterizedTest
85      @ValueSource(doubles = { -1.0, 1.0, 12345.67, -12345.67 })
86      void testMultiplicationWithOne(double input) {
87          assertEquals(input, calculator.mult(input, 1.0), DELTA);
88      }
89
90      @ParameterizedTest
91      @CsvSource({ "6.0, 2.0, 3.0", "1.0, -1.0, -1.0", "0.0, -5.5, 0.0" })
92      void testGeneralCorrectness(double expected, double a, double b) {
93          assertEquals(expected, calculator.mult(a, b), DELTA);
94      }
95  }
96  class MulLongTest extends AbstractCalculatorTest {
97      @ParameterizedTest
98      @ValueSource(longs = { 0, -1, 1, 12345, -12345 })
99      void testMultiplicationWithZero(long input) {
100         assertEquals(0, calculator.mult(input, 0));
101     }
102
103     @ParameterizedTest
104     @ValueSource(longs = { -1, 1, 12345, -12345 })
105     void testMultiplicationWithOne(long input) {
106         assertEquals(input, calculator.mult(input, 1));
107     }
108
109     @ParameterizedTest
110     @CsvSource({ "6, 2, 3", "1, -1, -1", "0, -5, 0" })
111     void testGeneralCorrectness(long expected, long a, long b) {
```

```
112         assertEquals(expected, calculator.mult(a, b));
113     }
114 }
115 class SubDoubleTest extends AbstractCalculatorTest {
116     @ParameterizedTest
117     @ValueSource(doubles = { 0.0, -1.0, 1.0, 12345.67, -12345.67 })
118     void testSubtractionWithZero(double input) {
119         assertEquals(input, calculator.sub(input, 0.0), DELTA);
120     }
121 
122     @ParameterizedTest
123     @ValueSource(doubles = { -1.0, 1.0, 12345.67, -12345.67 })
124     void testSubtractionWithSelf(double input) {
125         assertEquals(0.0, calculator.sub(input, input), DELTA);
126     }
127 
128     @ParameterizedTest
129     @CsvSource({ "1.1, 3.2, 2.1", "0.0, -1.1, -1.1", "-10.5, -5.5, 5.0" })
130     void testGeneralCorrectness(double expected, double a, double b) {
131         assertEquals(expected, calculator.sub(a, b), DELTA);
132     }
133 }
134 class SubLongTest extends AbstractCalculatorTest {
135     @ParameterizedTest
136     @ValueSource(longs = { 0, -1, 1, 12345, -12345 })
137     void testSubtractionWithZero(long input) {
138         assertEquals(input, calculator.sub(input, 0));
139     }
140 
141     @ParameterizedTest
142     @ValueSource(longs = { 0, -1, 1, 12345, -12345 })
```

```
143     void testSubtractionWithSelf(long input) {
144         assertEquals(0, calculator.sub(input, input));
145     }
146
147     @ParameterizedTest
148     @CsvSource({ "1, 3, 2", "0, -1, -1", "-10, -5, 5" })
149     void testGeneralCorrectness(long expected, long a, long b) {
150         assertEquals(expected, calculator.sub(a, b));
151     }
152 }
```

Результаты тестов представлены:

Рис. 1: Результаты тестирования

2.2 Лабораторная работа №2

В данной лабораторной работе необходимо использовать фреймворк TestNG для того, чтобы протестировать корректность работы библиотеки SeleniumWebDriver на примере сайта `https://jdi-testing.github.io/jdi-light/index.html` с использованием браузера Chrome. Необходимо реализовать два теста, каждый из которых проверяет корректность отображения конкретной страницы этого сайта. Ниже приведены представленные для работы тестовые сценарии.

Упражнение 1

Для этого упражнения используйте SoftAsserts.

#	Шаг	Данные	Ожидаемый результат
1	Open test site by URL	https://jdi-testing.github.io/jdi-light/index.html	Test site is opened
2	Assert Browser title	"Home Page"	Browser title equals "Home Page"
3	Perform login	username: Roman pass: Jdi1234	User is logged
4	Assert Username is loggined	"ROMAN IOVLEV"	Name is displayed and equals to expected result
5	Assert that there are 4 items on the header section are displayed and they have proper texts	"HOME", "CONTACT FORM", "SERVICE", "METALS & COLORS"	Menu buttons are displayed and have proper texts
6	Assert that there are 4 images on the Index Page and they are displayed	4 images	Images are displayed
7	Assert that there are 4 texts on the Index Page under icons and they have proper text	4 texts below of each image	Texts are displayed and equal to expected
8	Assert that there is the iframe with “Frame Button” exist	%	The iframe exists
9	Switch to the iframe and check that there is “Frame Button” in the iframe	%	The “Frame Button” exists
10	Switch to original window back	%	Driver has focus on the original window
11	Assert that there are 5 items in the Left Section are displayed and they have proper text	“Home”, “Contact form”, “Service”, “Metals & Colors”, “Elements packs”	Left section menu items are displayed and have proper text
12	Close Browser	%	Browser is closed

Рис. 2: Тестовый сценарий 1

Задаине 2

#	Шаг	Данные	Ожидаемый результат
1	Open test site by URL	https://jdi-testing.github.io/jdi-light/index.html	Test site is opened
2	Assert Browser title	"Home Page"	Browser title equals "Home Page"
3	Perform login	username: Roman pass: Jdi1234	User is logged
4	Assert User name in the left-top side of screen that user is loggined	ROMAN IOVLEV	Name is displayed and equals to expected result
5	Open through the header menu Service -> Different Elements Page	%	Page is opened
6	Select checkboxes	Water, Wind	Elements are checked
7	Select radio	Selen	Element is checked
8	Select in dropdown	Yellow	Element is selected
9	Assert that • -> for each checkbox there is an individual log row and value is corresponded to the status of checkbox • -> for radio button there is a log row and value is corresponded to the status of radio button • -> for dropdown there is a log row and value is corresponded to the selected value.	%	Log rows are displayed and • -> checkbox name and its status are corresponding to selected • -> radio button name and it status is corresponding to selected • -> dropdown name and selected value is corresponding to selected
10	Close Browser		Browser is closed

Рис. 3: Тестовый сценарий 2

В результате выполнения были реализованы три класса: DriverSerup, SubTaskA и SubTaskB. Для запуска этих классов был создан Maven профиль.

```
public class DriverSetup {
    protected static WebDriver driver;

    @BeforeClass
    public void setup() {
        System.setProperty("webdriver.chrome.driver", "src/test/resources/
    chromedriver");
        System.setProperty("webdriver.http.factory", "jdk-http-client");
        driver = new ChromeDriver();
        driver.manage().window().maximize();

        // 1. Open test site by URL
        driver.navigate().to("https://jdi-testing.github.io/jdi-light/index.html")
    ;

        // 3. Perform login
        driver.findElement(
                By.cssSelector("html>body>header>div>nav>ul.uui-navigation.navbar-
    nav.navbar-right>li>a>span")).click();
        driver.findElement(By.id("name")).sendKeys("Roman");
        driver.findElement(By.id("password")).sendKeys("Jdi1234");
        driver.findElement(By.id("login-button")).click();
    }

    @AfterClass
    public void exit() {
        // 10. Close Browser
        driver.close();
    }
```

```
}
public class SubTaskA extends DriverSetup {
    @Test
    public void testTask() {
        SoftAssert softAssert = new SoftAssert();

        // 2. Assert browser title
        softAssert.assertEquals(driver.getTitle(), "Home Page");

        // 4. Assert Username is logged in
        softAssert.assertEquals(driver.findElement(By.id("user-name")).getText(),
    "ROMAN IOVLEV");

        // 5. Assert that there are 4 items on the header section are displayed,
    and they have proper texts
        List<WebElement> headerItems = driver.findElement(By.cssSelector("ul.uui-
    navigation.nav.navbar-nav.m-l8"))
                .findElements(By.xpath("./child::*"));
        final int EXPECTED_HEADER_ITEMS_SIZE = 4;
        softAssert.assertEquals(headerItems.size(), EXPECTED_HEADER_ITEMS_SIZE);
        headerItems.forEach(item -> softAssert.assertTrue(item.isDisplayed()));
        softAssert.assertEquals(headerItems.stream().map(WebElement::getText).
    toList(),
                List.of("HOME", "CONTACT FORM", "SERVICE", "METALS & COLORS"));

        // 6. Assert that there are 4 images on the Index Page, and they are
    displayed
        List<WebElement> benefitImages = driver.findElements(By.className("benefit
    -icon"));
        final int EXPECTED_BENEFIT_IMAGES_SIZE = 4;
        softAssert.assertEquals(benefitImages.size(), EXPECTED_BENEFIT_IMAGES_SIZE
```

```
);
52          benefitImages.forEach(image -> softAssert.assertTrue(image.isDisplayed()))
    ;
53
54          // 7. Assert that there are 4 texts on the Index Page under icons, and
    they have proper text
55          List<WebElement> benefitTexts = driver.findElements(By.className("benefit-
    txt"));
56          final int EXPECTED_BENEFIT_TEXTS_SIZE = 4;
57          softAssert.assertEquals(benefitTexts.size(), EXPECTED_BENEFIT_TEXTS_SIZE);
58          benefitTexts.forEach(text -> softAssert.assertTrue(text.isDisplayed()));
59          softAssert.assertEquals(
60                  benefitTexts.stream()
61                          .map(WebElement::getText)
62                          .map(text -> text.replace("\n", " "))
63                          .toList(),
64                  List.of(
65                          "To include good practices and ideas from successful EPAM
    project",
66                          "To be flexible and customizable",
67                          "To be multiplatform",
68                          "Already have good base (about 20 internal and some
    external projects), wish to get more..."));
69
70          // 8. Assert that there is the iframe with Frame Button exist
71          final String EXPECTED_IFRAME_SRC = "https://jdi-testing.github.io/jdi-
    light/frame-button.html";
72          softAssert.assertEquals(driver.findElement(By.tagName("iframe")).
    getAttribute("src"),
73                  EXPECTED_IFRAME_SRC);
74
```

```
75          // 9.  Switch to the iframe and check that there is Frame Button in the
    iframe
76          driver.switchTo().frame("frame");
77          final String EXPECTED_FRAME_VALUE = "Frame Button";
78          softAssert.assertEquals(driver.findElement(By.id("frame-button")).
    getAttribute("value"),
79                  EXPECTED_FRAME_VALUE);
80
81          // 10. Switch to original window back
82          driver.switchTo().defaultContent();
83
84          // 11. Assert that there are 5 items in the Left Section are displayed,
    and they have proper text
85          List<WebElement> leftSectionItems = driver.findElement(By.cssSelector("ul.
    sidebar-menu.left"))
86                  .findElements(By.xpath("./child::*"));
87          leftSectionItems.forEach(item -> softAssert.assertTrue(item.isDisplayed())
    );
88          softAssert.assertEquals(leftSectionItems.stream().map(WebElement::getText)
    .toList(), List.of(
89                  "Home", "Contact form", "Service", "Metals & Colors", "Elements
    packs"));
90
91          // Assert all in soft assert
92          softAssert.assertAll();
93      }
94  }
95  public class SubTaskB extends DriverSetup {
96      @Test
97      public void testBrowserTitle() {
98          // 2. Assert browser title
```

```
99          assertEquals(driver.getTitle(), "Home Page");
100     }
101
102     @Test
103     public void testLogin() {
104         // 4. Assert User name in the left-top side of screen that user is
    loggined
105         assertEquals(driver.findElement(By.id("user-name")).getText(), "ROMAN
    IOVLEV");
106     }
107
108     @Test
109     public void testElements() {
110         // 5. Open through the header menu Service -> Different Elements Page
111         driver.findElement(By.cssSelector("body>header>div>nav>ul.uui-navigation.
    nav.navbar-nav.m-l8>li>a>span"))
112                 .click();
113         driver.findElement(By.xpath("html/body/header/div/nav/ul[1]/li[3]/ul/li
    [8]/a")).click();
114
115         // 6. Select checkboxes
116         driver.findElements(By.className("label-checkbox")).stream()
117                 .filter(checkbox -> checkbox.getText().equals("Water") || checkbox
    .getText().equals("Wind"))
118                 .forEach(WebElement::click);
119
120         // 7. Select radio
121         driver.findElements(By.className("label-radio")).stream()
122                 .filter(radio -> radio.getText().equals("Selen"))
123                 .forEach(WebElement::click);
124
```

```
125         // 8. Select in dropdown
126         driver.findElements(By.tagName("option")).stream()
127                 .filter(option -> option.getText().equals("Yellow"))
128                 .forEach(WebElement::click);
129 
130         // 9. Assert logs
131         final int LOGS_BEGIN_INDEX = 9;
132         List<String> logLines = Arrays.stream(driver.findElement(By.cssSelector("
    ul.panel-body-list.logs")).getText()
133                 .split("\n")).map(line -> line.substring(LOGS_BEGIN_INDEX))
134                 .toList();
135         assertEquals(logLines, List.of(
136                 "Colors: value changed to Yellow",
137                 "metal: value changed to Selen",
138                 "Wind: condition changed to true",
139                 "Water: condition changed to true"));
140     }
141 }
```

Результаты тестов:

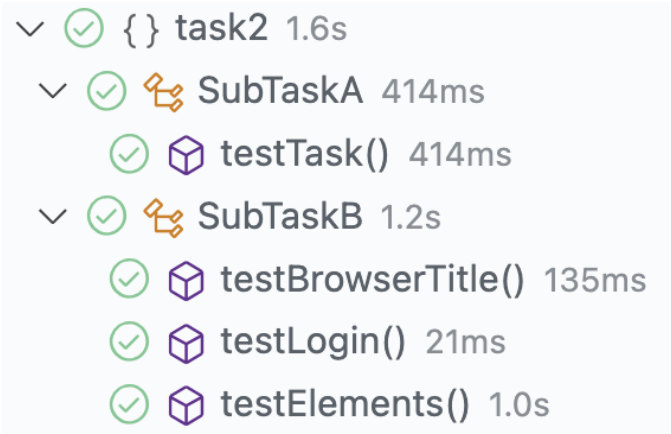

Рис. 4: Результаты тестирования сайта

2.3 Лабораторная работа №3

В данной работе необходимо произвести рефакторинг тестов из второй лабораторной работы, используя шаблон проектирования тестов Page Object. То есть нужно создать объекты страниц и отделить код работы с элементами страницы от работы с веб-драйвером.

В результате выполнения работы были реализованы классы HomePage, DifferentElementsPage, IFrame, представляющие 3 различных веб-страницы, и классы Header и LeftSection, представляющие 2 веб-элемента страницы HomePage, которые используются в процессе тестирования. После реализации необходимых средств для работы с вебстраницами был изменен код классов, содержащих тесты.

```
1  public class DifferentElementsPage {
2      private static final Integer LOGS_BEGIN_INDEX = 9;
3
4      @FindBy(className = "label-checkbox")
5      private List<WebElement> checkboxes;
6
7      @FindBy(className = "label-radio")
8      private List<WebElement> radios;
9
10     @FindBy(tagName = "option")
11     private List<WebElement> dropdownOptions;
12
13     @FindBy(css = "ul.panel-body-list.logs")
14     private WebElement logs;
15
16     public DifferentElementsPage(WebDriver webDriver, HomePage homePage) {
17         homePage.getHeaderSection().clickServiceDropdown();
18         homePage.getHeaderSection().clickDifferentElements();
19         PageFactory.initElements(webDriver, this);
```

```
20      }
21
22      public void selectCheckbox(String name) {
23          this.checkboxes.stream().filter(checkbox -> checkbox.getText().equals(name
        ))
24                  .forEach(WebElement::click);
25      }
26
27      public void selectRadio(String name) {
28          this.radios.stream().filter(radio -> radio.getText().equals(name))
29                  .forEach(WebElement::click);
30      }
31
32      public void selectDropdownOption(String name) {
33          this.dropdownOptions.stream().filter(option -> option.getText().equals(
        name))
34                  .forEach(WebElement::click);
35      }
36
37      public List<String> getLogs() {
38          return Arrays.stream(logs.getText().split("\n"))
39                  .map(log -> log.substring(LOGS_BEGIN_INDEX)).toList();
40      }
41  }
42  public class DriverSetup {
43      protected static WebDriver webDriver;
44      protected static HomePage homePage;
45
46      @BeforeClass
47      public static void setup() {
48          System.setProperty("webdriver.chrome.driver", "src/test/resources/
```

```
    chromedriver");
49          System.setProperty("webdriver.http.factory", "jdk-http-client");
50          Properties properties = new Properties();
51          try {
52              properties.load(Files.newInputStream(Path.of("src/test/resources/data.
    properties")));
53          } catch (IOException error) {
54              throw new RuntimeException(error);
55          }
56
57          webDriver = new ChromeDriver();
58          webDriver.manage().window().maximize();
59
60          homePage = new HomePage(webDriver, properties.getProperty("site.url"));
61
62          // 3. Perform login
63          homePage.performLogin(
64                  properties.getProperty("user.name"),
65                  properties.getProperty("user.password"));
66      }
67
68      @AfterClass
69      public static void exit() {
70          // 10. Close browser
71          webDriver.close();
72      }
73  }
74  public class ExpectedData {
75      public static final String SITE_NAME = "Home Page";
76      public static final String LOGGED_USER_NAME = "ROMAN IOVLEV";
77      public static final List<String> HEADER_SECTION_ITEMS = List.of(
```

```
        "HOME", "CONTACT FORM", "SERVICE", "METALS & COLORS"
    );
    public static final List<String> BENEFIT_ICONS = List.of(
        "To include good practices\nand ideas from successful\nEPAM project",
        "To be flexible and\ncustomizable",
        "To be multiplatform",
        "Already have good base\n(about 20 internal and\nsome external projects),\nwish to get more..."
    );
    public static final List<String> LEFT_SECTION_ITEMS = List.of(
        "Home", "Contact form", "Service", "Metals & Colors", "Elements packs"
    );
    public static final List<String> DIFFERENT_ELEMENTS_LOGS = List.of(
        "Colors: value changed to Yellow",
        "metal: value changed to Selen",
        "Wind: condition changed to true",
        "Water: condition changed to true"
    );
}
public class HeaderSection {
    @FindBy(css = "body > header > div > nav > ul.uui-navigation.nav.navbar-nav.m-l8 > li")
    private List<WebElement> items;

    @FindBy(css = "body > header > div > nav > ul.uui-navigation.nav.navbar-nav.m-l8 > li > a > span")
    private WebElement serviceDropdown;

    @FindBy(css = "body > header > div > nav > ul.uui-navigation.nav.navbar-nav.m-l8 > li.dropdown.open > ul > li:nth-child(8) > a")
    private WebElement differentElements;
```

```
105
106     public HeaderSection(WebDriver webDriver) {
107         PageFactory.initElements(webDriver, this);
108     }
109
110     public void clickServiceDropdown() {
111         this.serviceDropdown.click();
112     }
113
114     public void clickDifferentElements() {
115         this.differentElements.click();
116     }
117
118     public List<WebElement> getItems() {
119         return this.items;
120     }
121
122     public Integer getItemsSize() {
123         return this.items.size();
124     }
125
126     public List<String> getItemsNames() {
127         return this.items.stream().map(WebElement::getText).toList();
128     }
129 }
130 public class HomePage {
131     private final WebDriver webDriver;
132     private final HeaderSection headerSection;
133     private final LeftSection leftSection;
134
135     @FindBy(id = "name")
```

```
136     private WebElement loginUsername;
137
138     @FindBy(id = "password")
139     private WebElement loginPassword;
140
141     @FindBy(css = "html > body > header > div > nav > ul.uui-navigation.navbar-nav
        .navbar-right > li > a > span")
142     private WebElement loginDropdownButton;
143
144     @FindBy(id = "login-button")
145     private WebElement loginButton;
146
147     @FindBy(id = "user-name")
148     private WebElement username;
149
150     @FindBy(className = "benefit-icon")
151     private List<WebElement> benefitIcons;
152
153     @FindBy(className = "benefit-txt")
154     private List<WebElement> benefitIconsTexts;
155
156     @FindBy(tagName = "iframe")
157     private WebElement iframe;
158
159     public HomePage(WebDriver webDriver, String url) {
160         this.webDriver = webDriver;
161
162         // 1. Open test site by URL
163         this.webDriver.navigate().to(url);
164
165         PageFactory.initElements(this.webDriver, this);
```

```
166         headerSection = new HeaderSection(webDriver);
167         leftSection = new LeftSection(webDriver);
168     }
169 
170     public void performLogin(String username, String password) {
171         this.loginDropdownButton.click();
172         this.loginUsername.sendKeys(username);
173         this.loginPassword.sendKeys(password);
174         this.loginButton.click();
175     }
176 
177     public String getTitle() {
178         return this.webDriver.getTitle();
179     }
180 
181     public String getLoggedName() {
182         return this.username.getText();
183     }
184 
185     public HeaderSection getHeaderSection() {
186         return this.headerSection;
187     }
188 
189     public Integer getBenefitIconsSize() {
190         return this.benefitIcons.size();
191     }
192 
193     public List<WebElement> getBenefitIcons() {
194         return this.benefitIcons;
195     }
196 
```

```
197     public Integer getBenefitIconsTextsSize() {
198         return this.benefitIconsTexts.size();
199     }
200 
201     public List<String> getBenefitIconsTextsStrings() {
202         return this.benefitIconsTexts.stream().map(WebElement::getText).toList();
203     }
204 
205     public LeftSection getLeftSection() {
206         return this.leftSection;
207     }
208 
209     public WebElement getFrame() {
210         return this.iframe;
211     }
212 
213     public WebElement getFrameButton() {
214         return new IFrame(this.webDriver).getFrameButton();
215     }
216 
217     public void switchToOriginalWindow() {
218         this.webDriver.switchTo().defaultContent();
219     }
220 }
221 public class IFrame {
222     @FindBy(id = "frame-button")
223     private WebElement frameButton;
224 
225     public IFrame(WebDriver webDriver) {
226         webDriver.switchTo().frame("frame");
227         PageFactory.initElements(webDriver, this);
```

```
228     }
229
230     public WebElement getFrameButton() {
231         return this.frameButton;
232     }
233 }
234 public class LeftSection {
235     @FindBy(css = "#mCSB_1_container > ul > li")
236     private List<WebElement> items;
237
238     public LeftSection(WebDriver webDriver) {
239         PageFactory.initElements(webDriver, this);
240     }
241
242     public List<WebElement> getItems() {
243         return this.items;
244     }
245
246     public Integer getItemSize() {
247         return this.items.size();
248     }
249
250     public List<String> getItemsNames() {
251         return this.items.stream().map(WebElement::getText).toList();
252     }
253 }
254 public class SubTaskA extends DriverSetup {
255     @Test
256     public void testTaskA() {
257         SoftAssert softAssert = new SoftAssert();
258
```

```
259         // 2. Assert browser title
260         softAssert.assertEquals(homePage.getTitle(), ExpectedData.SITE_NAME);
261 
262         // 4. Assert user is logged in
263         softAssert.assertEquals(homePage.getLoggedName(), ExpectedData.
    LOGGED_USER_NAME);
264 
265         // 5. Assert that there are 4 items on the header section are displayed ,
    and they have proper texts
266         softAssert.assertEquals((int) homePage.getHeaderSection().getItemsSize(),
    (int) ExpectedData.HEADER_SECTION_ITEMS.size());
267         softAssert.assertEquals(homePage.getHeaderSection().getItemsNames(),
    ExpectedData.HEADER_SECTION_ITEMS);
268         homePage.getHeaderSection().getItems().forEach(item -> {
269             softAssert.assertTrue(item.isDisplayed());
270         });
271 
272         // 6. Assert that there are 4 images on the Index Page, and they are
    displayed
273         softAssert.assertEquals((int) homePage.getBenefitIconsSize(), (int)
    ExpectedData.BENEFIT_ICONS.size());
274         homePage.getBenefitIcons().forEach(icon -> {
275             softAssert.assertTrue(icon.isDisplayed());
276         });
277 
278         // 7. Assert that there are 4 texts on the Index Page under icons, and
    they have proper text
279         softAssert.assertEquals((int) homePage.getBenefitIconsTextsSize(), (int)
    ExpectedData.BENEFIT_ICONS.size());
280         softAssert.assertEquals(homePage.getBenefitIconsTextsStrings(),
    ExpectedData.BENEFIT_ICONS);
```

```
281
282         // 8. Assert that there is an iframe with frame button exists
283         softAssert.assertTrue(homePage.getFrame().isDisplayed());
284
285         // 9. Switch to the iframe and check that there is Frame Button in the
    iframe
286         softAssert.assertTrue(homePage.getFrameButton().isDisplayed());
287
288         // 10. Switch to original window back
289         homePage.switchToOriginalWindow();
290
291         // 11. Assert that there are 5 items in the Left Section are displayed,
    and they have proper text
292         softAssert.assertEquals((int) homePage.getLeftSection().getItemSize(), (
    int) ExpectedData.LEFT_SECTION_ITEMS.size());
293         homePage.getLeftSection().getItems().forEach(item -> {
294             softAssert.assertTrue(item.isDisplayed());
295         });
296         softAssert.assertEquals(homePage.getLeftSection().getItemsNames(),
    ExpectedData.LEFT_SECTION_ITEMS);
297
298         softAssert.assertAll();
299     }
300 }
301 public class SubTaskB extends DriverSetup {
302     @Test
303     public void testBrowserTitle() {
304         // 2. Assert browser title
305         assertEquals(homePage.getTitle(), ExpectedData.SITE_NAME);
306     }
307
```

```
308     @Test
309     public void testLogin() {
310         // 4. Assert user is logged in
311         assertEquals(homePage.getLoggedName(), ExpectedData.LOGGED_USER_NAME);
312     }
313 
314     @Test
315     public void testElements() {
316         // 5. Open through the header menu Service -> Different Elements Page
317         DifferentElementsPage differentElementsPage = new DifferentElementsPage(
    webDriver, homePage);
318 
319         // 6. Select checkboxes
320         differentElementsPage.selectCheckbox("Water");
321         differentElementsPage.selectCheckbox("Wind");
322 
323         // 7. Select radio
324         differentElementsPage.selectRadio("Selen");
325 
326         // 8. Select in dropdown
327         differentElementsPage.selectDropdownOption("Yellow");
328 
329         // 9. Assert logs
330         assertEquals(differentElementsPage.getLogs(), ExpectedData.
    DIFFERENT_ELEMENTS_LOGS);
331     }
332 }
```

Результаты тестов:

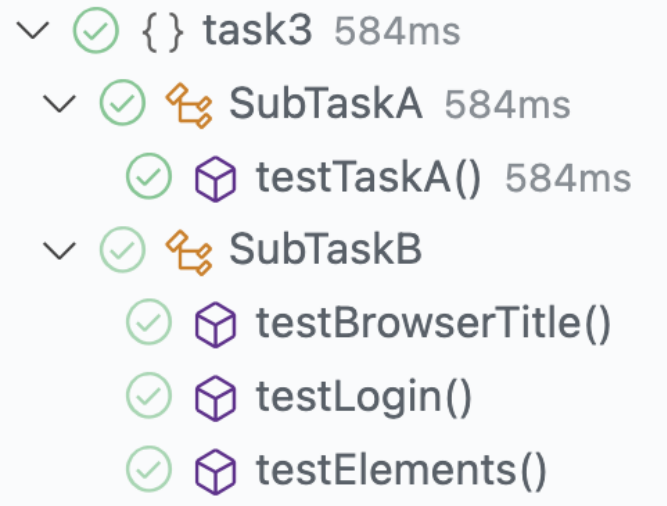

Рис. 5: Результаты тестирования сайта

2.4 Лабораторная работа №4

В данной лабораторной работе необходимо произвести рефакторинг тестов из лабораторной работы 3 с использованием шаблона программирования Steps. То есть необходимо создать еще один уровень между кодом тестов и кодом взаимодействия с веб-страницами и веб-элементами. Классы с шагами будут отвечать за вызов определенных методов для действий на странице и за запуск проверок. Таким образом, код теста будет состоять из вызовов реализованных шагов.

Кроме того, необходимо создать третий тест на основе теста из предыдущей работы, который бы содержал ошибки и оказался не пройденным.

Также необходимо создать задачу в Jenkins, которая будет запускать полученные тесты и создавать отчет с помощью Allure Report.

```
1 public class Action extends StepsSetup {
2     public Action(WebDriver webDriver, Properties properties) {
3         super(webDriver, properties);
4     }
```

```
5
6       @Step("Navigating to Home Page")
7       public void navigateToHomePage() {
8           homePage = new HomePage(webDriver, properties.getProperty("site.url"));
9       }
10
11      @Step("Logging in")
12      public void performLogin() {
13          homePage.performLogin(
14                  properties.getProperty("user.name"),
15                  properties.getProperty("user.password"));
16      }
17
18      @Step("Switching to the original window")
19      public void switchToOriginalWindow() {
20          homePage.switchToOriginalWindow();
21      }
22
23      @Step("Navigating to different elements page")
24      public void navigateToDifferentElementsPage() {
25          differentElementsPage = new DifferentElementsPage(webDriver, homePage);
26      }
27
28      @Step("Selecting checkboxes")
29      public void selectCheckboxes(String... names) {
30          Stream.of(names).forEach(name -> differentElementsPage.selectCheckbox(name
        ));
31      }
32
33      @Step("Selecting radio")
34      public void selectRadio(String name) {
```

```
35          differentElementsPage.selectRadio(name);
36      }
37
38      @Step("Selecting dropdown option")
39      public void selectDropdownOption(String name) {
40          differentElementsPage.selectDropdownOption(name);
41      }
42  }
43  public class Assertion extends StepsSetup {
44      public Assertion(WebDriver webDriver, Properties properties) {
45          super(webDriver, properties);
46      }
47
48      @Step("Asserting Home Page's title")
49      public void assertHomePageTitle(String expectedTitle) {
50          assertEquals(homePage.getTitle(), expectedTitle);
51      }
52
53      @Step("Asserting user is logged in")
54      public void assertUserLoggedIn(String expectedUsername) {
55          assertEquals(homePage.getLoggedName(), expectedUsername);
56      }
57
58      @Step("Asserting Header Seciton items properties")
59      public void assertHeaderSectionItemsProperties(
60              Integer expectedItemsSize,
61              List<String> expectedItemsNames) {
62          SoftAssert softAssert = new SoftAssert();
63          softAssert.assertEquals(homePage.getHeaderSection().getItemsSize(),
    expectedItemsSize);
64          softAssert.assertEquals(homePage.getHeaderSection().getItemsNames(),
```

```
    expectedItemsNames);
65          homePage.getHeaderSection().getItems().forEach(item -> {
66              softAssert.assertTrue(item.isDisplayed());
67          });
68          softAssert.assertAll();
69      }
70
71      @Step("Asserting Index Page images properties")
72      public void assertIndexPageImages(Integer expectedImagesSize) {
73          SoftAssert softAssert = new SoftAssert();
74          softAssert.assertEquals(homePage.getBenefitIconsSize(), expectedImagesSize
    );
75          homePage.getBenefitIcons().forEach(icon -> {
76              softAssert.assertTrue(icon.isDisplayed());
77          });
78          softAssert.assertAll();
79      }
80
81      @Step("Asserting Index Page texts properties")
82      public void assertIndexPageTexts(
83              Integer expectedTextsSize,
84              List<String> expectedTextsStrings) {
85          SoftAssert softAssert = new SoftAssert();
86          softAssert.assertEquals(homePage.getBenefitIconsTextsSize(),
    expectedTextsSize);
87          softAssert.assertEquals(homePage.getBenefitIconsTextsStrings(),
    expectedTextsStrings);
88          softAssert.assertAll();
89      }
90
91      @Step("Asserting iframe existence")
```

```
92      public void assertFrameExsitence() {
93          assertTrue(homePage.getFrame().isDisplayed());
94      }
95
96      @Step("Asserting Frame Button existence")
97      public void assertFrameButtonExistence() {
98          assertTrue(homePage.getFrameButton().isDisplayed());
99      }
100
101     @Step("Asserting Left Section properties")
102     public void assertLeftSectionProperties(
103             Integer expectedItemsSize,
104             List<String> expectedItemsNames) {
105         SoftAssert softAssert = new SoftAssert();
106         softAssert.assertEquals(homePage.getLeftSection().getItemSize(),
    expectedItemsSize);
107         softAssert.assertEquals(homePage.getLeftSection().getItemsNames(),
    expectedItemsNames);
108         homePage.getLeftSection().getItems().forEach(item -> {
109             softAssert.assertTrue(item.isDisplayed());
110         });
111         softAssert.assertAll();
112     }
113
114     @Step("Asserting logs")
115     public void assertLogs(List<String> logs) {
116         assertEquals(differentElementsPage.getLogs(), logs);
117     }
118 }
119 @Feature("First task using Steps")
120 public class SubTaskA extends DriverSetup {
```

```
121     @Story("Testing the Home Page")
122     @Test
123     public void testTaskA() {
124         // 2. Assert browser title
125         assertion.assertHomePageTitle(ExpectedData.SITE_NAME);
126 
127         // 4. Assert user is logged in
128         assertion.assertUserLoggedIn(ExpectedData.LOGGED_USER_NAME);
129 
130         // 5. Assert that there are 4 items on the header section are displayed,
    and they have proper texts
131         assertion.assertHeaderSectionItemsProperties(
132                 ExpectedData.HEADER_SECTION_ITEMS.size(),
133                 ExpectedData.HEADER_SECTION_ITEMS);
134 
135         // 6. Assert that there are 4 images on the Index Page, and they are
    displayed
136         assertion.assertIndexPageImages(ExpectedData.BENEFIT_ICONS.size());
137 
138         // 7. Assert that there are 4 texts on the Index Page under icons, and
    they have proper text
139         assertion.assertIndexPageTexts(ExpectedData.BENEFIT_ICONS.size(),
    ExpectedData.BENEFIT_ICONS);
140 
141         // 8. Assert that there is the iframe with frame button exists
142         assertion.assertFrameExsitence();
143 
144         // 9. Switch to the iframe and check that there is Frame Button in the
    iframe
145         assertion.assertFrameButtonExistence();
146 
```

```
147         // 10. Switch to original window back
148         action.switchToOriginalWindow();
149 
150         // 11. Assert that there are 5 items in the Left Section are displayed,
    and they have proper text
151         assertion.assertLeftSectionProperties(
152                 ExpectedData.LEFT_SECTION_ITEMS.size(),
153                 ExpectedData.LEFT_SECTION_ITEMS);
154     }
155 }
156 @Feature("Second task using Steps")
157 public class SubTaskB extends DriverSetup {
158     @Story("Testing the Different Elements page")
159     @Test
160     public void testTaskB() {
161         // 2. Assert browser title
162         assertion.assertHomePageTitle(ExpectedData.SITE_NAME);
163 
164         // 4. Assert user is logged in
165         assertion.assertUserLoggedIn(ExpectedData.LOGGED_USER_NAME);
166 
167         // 5. Open through the header header menu Service -> Different Elements
    Page
168         action.navigateToDifferentElementsPage();
169 
170         // 6. Select checkboxes
171         action.selectCheckboxes("Water", "Wind");
172 
173         // 7. Select radio
174         action.selectRadio("Selen");
175 
```

```
176         // 8. Select item in dropdown
177         action.selectDropdownOption("Yellow");
178 
179         // 9. Assert logs
180         assertion.assertLogs(ExpectedData.DIFFERENT_ELEMENTS_LOGS);
181     }
182 }
183 public class BadTask extends DriverSetup {
184     @Test
185     public void testBadTask() {
186         // 2. Assert browser title - a random stirng :)
187         assertion.assertHomePageTitle("A WRONG SITENAME");
188     }
189 }
```

Результаты тестов:

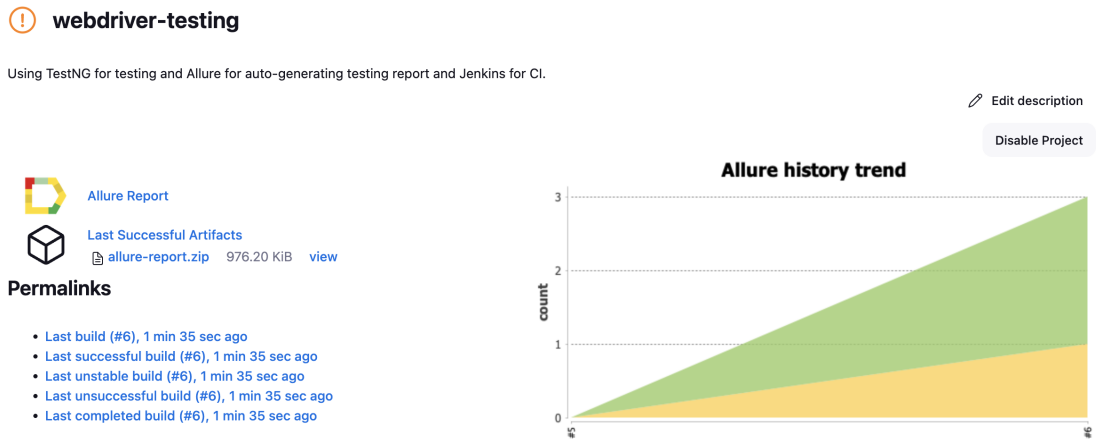

Рис. 6: Задача в Jenkins

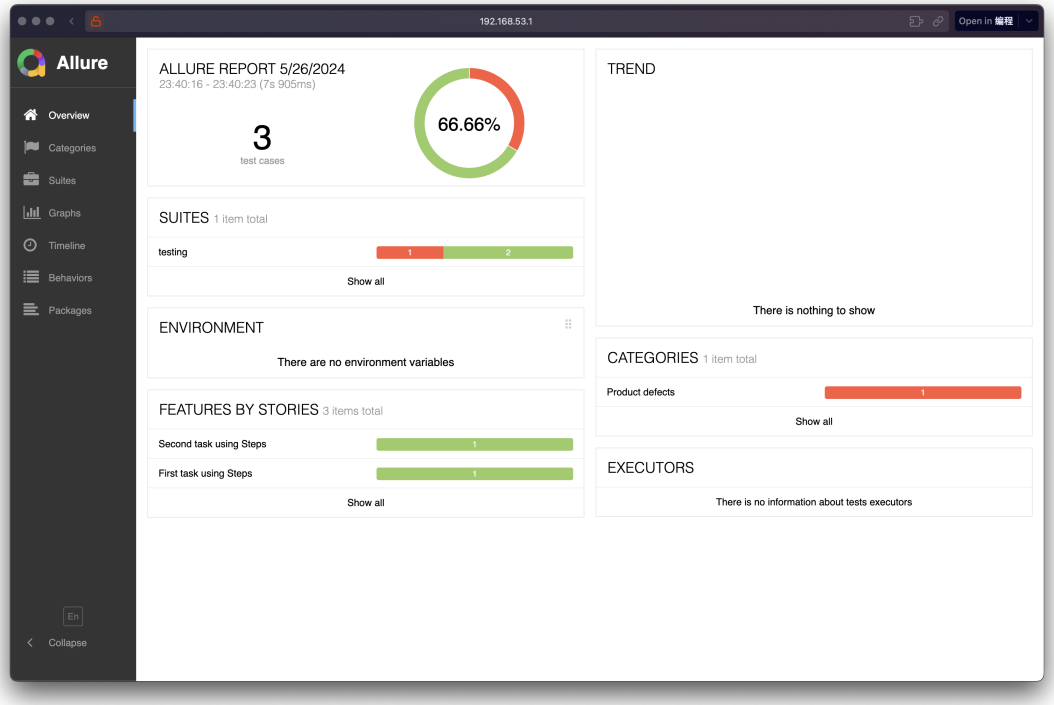

Рис. 7: Сформированный отчет в Allure

Заключение

В результате прохождения курса были изучены такие инструменты для автоматизации тестирования ПО, как фреймворки JUnit, TestNG, библиотека для управления браузерами Selenium WebDriver, шаблоны проектирования автоматизированных тестов Page Object и Steps, программная система сборки проектов Jenkins, фреймворк для создания отчетов Allure Framework.

Полученные знания были применены на практике в процессе выполнения 4 лабораторных работ. Все работы были выполнены успешно.

Тестирование, выполняемое с помощью специализированного ПО, позволяет ускорить процесс тестирования, не понижая при этом его качество, устранить влияние человеческого фактора, а также достаточно быстро и легко получить наглядные результаты тестирования с помощью составления отчетов. Используемые при автоматизированном тестировании шаблоны проектирования позволяют формализовать структуру кода тестов.

Список литературы

[1] Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер. Изд. 3-е. — Санкт-Петербург : Диалектика, 2012. — 272 с.

[2] Курс лекций по атоматизированному тестированию. Башарина Е.А. Шукшин И.Д.