**ФИРСОВА С. А., ПАКШИН А. В.**

# ОРГАНИЗАЦИЯ ПРОЦЕССА АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ WEB-ПРИЛОЖЕНИЙ НА БАЗЕ ФРЕЙМВОРКА SELENIUM

**Аннотация.** В статье описана разработанная авторами программная система, позволяющая выполнять автоматизированное тестирование web-приложений на базе фреймворка SELENIUM. Подробно рассмотрена архитектура программной системы, описан процесс построения тестовых сценариев и генерации отчетов по проведенному тестированию, приведен пример тестирования конкретного web-приложения.

**Ключевые слова:** автоматизированное тестирование, тестовые сценарии, архитектура программной системы, UML-диаграмма, Selenium.

**FIRSOVA S. A., PAKSHIN A.V.**

# ORGANIZATION OF THE PROCESS OF AUTOMATED TESTING OF WEB APPLICATIONS BASED ON THE SELENIUM FRAMEWORK

**Abstract.** The article describes a software system developed by the authors that allows automated testing of web applications based on the SELENIUM framework. The architecture of the software system is considered in detail, the process of building test scenarios and generating reports on the conducted testing is described, an example of testing a specific web application is given.

**Keywords:** automated testing, test scenarios, software system architecture, UML diagram, Selenium.

**Введение.** В современном мире программное обеспечение используется практически во всех сферах жизни, гигантские суммы тратятся на разработку разнообразных программ, востребованных в промышленности, бизнесе, индустрии развлечений, образовании и медицине [1]. Задача снижения стоимости разработки программного обеспечения и улучшения качества выпускаемой продукции является одной из наиболее актуальных в индустрии информационных технологий.

Автоматизация тестирования позволяет значительно сократить затраты компаний-разработчиков, сэкономить время и ресурсы, расходуемые на тестирование, снизить риск выпуска на рынок некачественного продукта. Поэтому технологии автоматизации тестирования набирают все большую популярность среди компаний, связанных с разработкой программных продуктов. Автоматизированное тестирование заключается в написании автотестов – программ, направленных на выполнение заложенных в них тестовых сценариев. Благодаря им разработчик может по истечению короткого периода времени получить информацию о том, были ли обнаружены в ходе выполнения данного сценария

ошибки или тест пройден успешно. Для разработки и запуска автотестов используются специальные системы автоматизированного тестирования. Главной идеей автоматизированного тестирования является полная замена человеческого труда. Программист единожды создает программу, которая проверяет все, подготовленные заранее, тестовые сценарии. Далее, сценарии только поддерживаются в актуальном рабочем состоянии. Создание таких программ позволяет снизить присутствие ручного тестирования в жизненном цикле разрабатываемого ПО [2].

Данный подход экономит время сотрудников отдела по контролю качества, а также ощутимо ускоряет работу команды разработчиков, и весь процесс разработки ПО в целом.

Недостатком автоматизации тестирования является то, что она применима только к работе с длинными проектами. Создание программы тестирования занимает много времени, и тратить это время на небольшие проекты просто бессмысленно. Как правило, автоматизированное тестирование эффективно применяется к относительно простым сценариям и тестам. Автоматизация сложных многошаговых сценариев может занять недопустимо много времени из-за непредвиденных нюансов или технических проблем [3]. Также может потребоваться много времени для поддержки тестов, при необходимости часто вносить в них правки после изменений в тестируемой функциональности. Несмотря на нюансы автоматического тестирования, этот тип не уступает, и компании внедряют его в свой рабочий процесс.

В данной статье рассматривается разработанная авторами программная система, предназначенная для построения тестовых сценариев и автоматизированного тестирования с их помощью различных web-приложений.

**Разработка архитектуры программной системы.** Архитектуру системы опишем в соответствии с рекомендациями Rational Unified Process. UML спецификация системы разделена на следующие части:

– Концептуальная модель. Данная модель описывает участников системы, основные сценарии и варианты использования системы;

– Логическая модель. Эта модель включает описание и диаграммы наиболее важных модулей системы, описания значимых классов, диаграммы последовательностей выполнения типовых задач и процессов приложения;

– Модель реализации. Эта модель включает описание разделения прототипа системы на отдельные компоненты, независимые задачи, подпрограммы, информационные и управляющие потоки и связи между элементами системы.

**Концептуальная модель** разработанной программной системы представлена диаграммой вариантов использования (см. рис. 1):


Рис. 1. Диаграмма вариантов использования.

На диаграмме вариантов использования показаны участники системы (актеры) и основные варианты работы системы (прецеденты):

– Тестировщик создает/редактирует сценарий тестирования web-приложения, используя возможности фреймворка Selenium Web Driver [4] и среды юнит-тестирования NUnit для языка C#. Сценарий сохраняется в виде исполняемого файла .exe для проведения последующих тестов.

– Программист запускает исполняемый файл, и система проводит автоматическое тестирование приложения. По завершении тестирования генерируется отчет, содержащий информацию о времени начала и завершения тестирования, количестве запущенных тестов, процент успешно пройденных тестов.

**Логическая модель** включает описание и диаграммы наиболее важных модулей системы, описания значимых классов, диаграммы последовательностей выполнения типовых задач и процессов приложения.

Диаграмма классов системы представлена на рисунке 2:

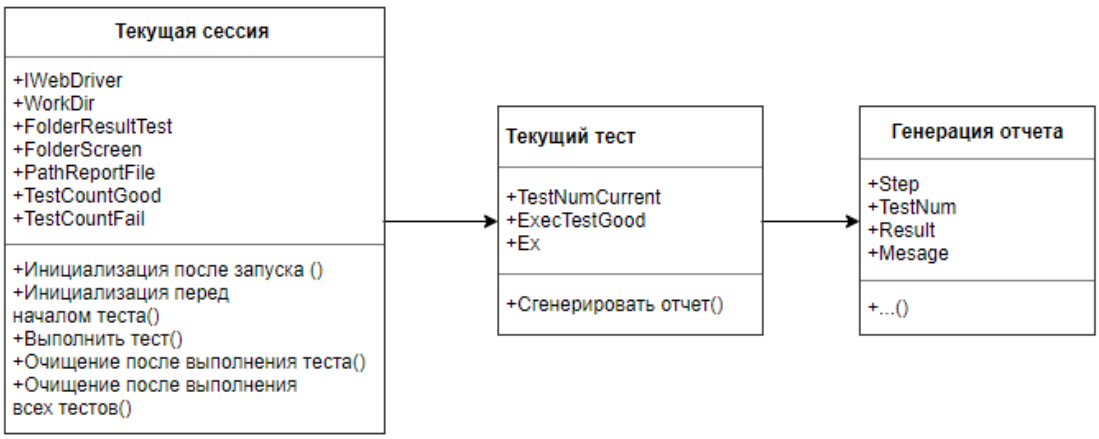


Рис. 2. Диаграмма классов.

Класс <Текущая сессия> содержит переменные для подсчета количества пройденных и не пройдённых тестов, адреса каталогов, а также методы инициации и проведения функционального тестирования:

– переменная <IWebDriver> служит указателем на интерфейс, через который пользователь управляет браузером;

– переменные <WorkDir>, <FolderResult>, <FolderScreen>, <PathReportFile> содержат информацию о путях расположения компонентов программы;

– переменные <TestCountGood>, <TestCountFail> являются счетчиками положительного и отрицательного прохождения тестов.

Класс <Текущий тест> содержит атрибуты, предоставляющие информацию о прохождении теста; инициирует и хранит такие переменные как:

– переменная <TestNumCurrent> хранит строку с наименованием протекающего тест-сценария.

– переменная <ExexTestGood> служит индикатором прохождения тестирования.

– переменная <Ex> является указателем на возникающее в случае выявления ошибки сообщение.

Класс <Генерация отчета> содержит атрибуты для создания отчета, инициирует и хранит такие переменные как:

– переменная <Step> которая является идентификатором этапа внесения информации: 0 – начало отчета, 1 – формирование отчета, -1 – завершение отчета.

– переменная <TestName> хранит строку с наименованием протекающего тест-сценария.

– переменная <Result> хранит результат прохождения тестирования: True-пройдено, False-не пройдено.

– переменная <Mesage> хранит строку с сообщением, выводимым при возникновении ошибки.

**Модель реализации** описывает разделение системы на отдельные компоненты, независимые задачи, подпрограммы, информационные и управляющие потоки и связи между элементами системы.

В разработанной программной системе в Visual Studio происходит компиляция кода на языке C#, который передается для исполнения утилите NUnit.consolerunner.

Скрипт теста содержит в себе команды, имитирующие взаимодействие пользователя с браузером Google Chrome при помощи драйвера Selenium WebDriver. При этом все внесенные или измененные данные записываются в существующую базу данных, связанную с тестируемым web-приложением. Диаграмма компонентов представлена на рисунке 3:

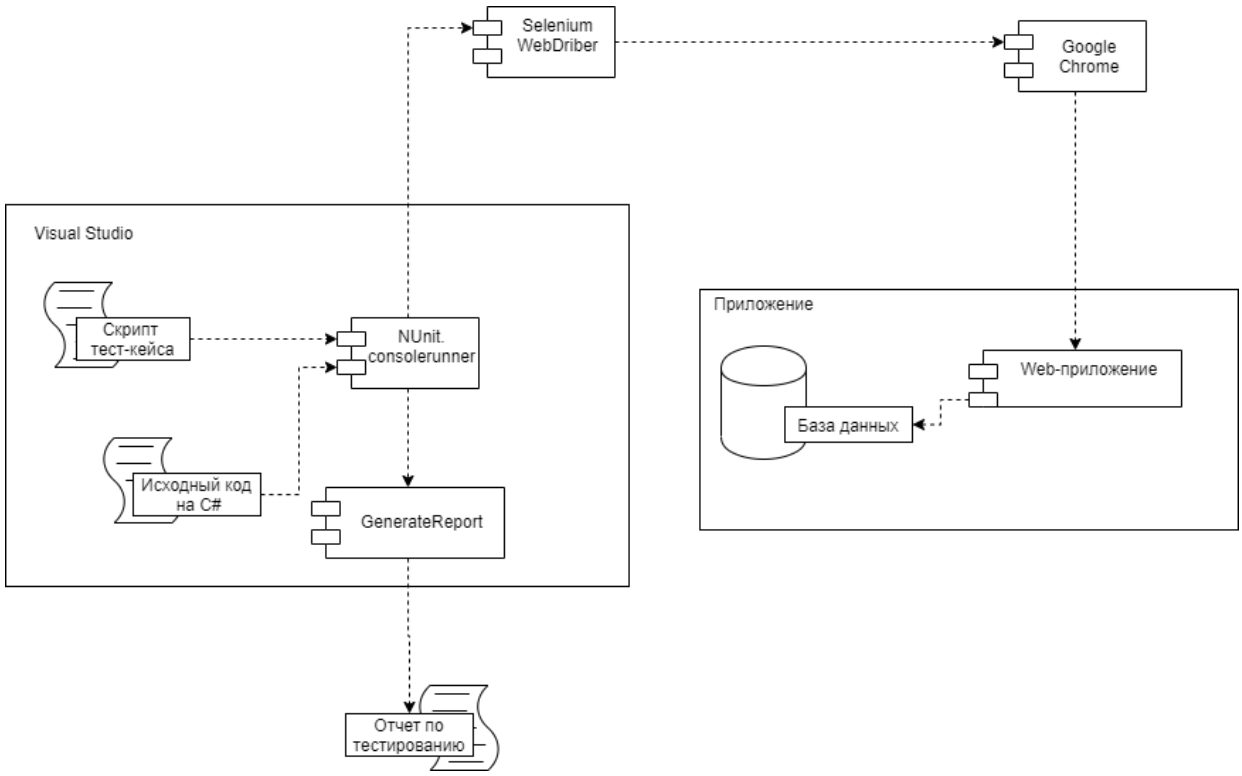


Рис. 3. Диаграмма компонентов.

Также во время выполнения тестирования, полученные результаты вносятся в формируемый при помощи метода <GeneraterReport> отчет, модель которого представлена на рис. 4:

**Тест начат 20.09.2021 12:15:16**

| Тест | Результат | Время | Снимок | Сообщение |
|---|---|---|---|---|
| S1 | Успех | 12:15:18 | - | |
| S2 | Успех | 12:15:23 | - | |
| S3 | Успех | 12:15:43 | open | |

Всего тесто запущено: 3 || **Успешно: 2** || **Провалено: 1** || Процент пройденных тестов: 66% ||
Тест завершен: 20.09.2021 12:17:35

Рис. 4. Модель отчета.

Отчет представляет собой HTML-таблицу, в первой строке которой размещается время начала тестирования. Первый столбец хранит название тестовых сценариев, второй – отвечает за результат проведения тестирования (Успех и Провал), третий – хранит время воспроизведения тестового сценария, которое привязано к системным часам операционной системы. В четвертый столбец (в случае провала тестирования) выводится ссылка на сделанный в момент возникновения критической ситуации скриншот окна браузера. Последний столбец содержит сообщение, выводимое при возникновении ошибки тестирования. В конце отчета выводится информация об общем количестве проведенных

тестов, количестве успешных и проваленных тестов, а также процент успешно пройденных тест-сценариев, а также время завершения тестирования.

**Программная реализация системы автоматизированного тестирования web-приложений.** Система построена на основе NUnit-фреймворка, предназначенного для модульного тестирования. NUnit поддерживает использование атрибутов, необходимых для распознавания тестов.

Атрибут [TestFixture] означает, что все помеченные им классы образуют тестовую структуру.

Атрибут [OneTimeSetUp] означает, что помеченные им классы будут вызываться единожды перед началом тестирования.

Атрибут [OneTimeTearDown] означает, что помеченные им классы будут вызываться единожды после окончания тестирования.

Атрибут [SetUp] означает, что помеченные им классы будут вызываться перед началом каждого теста.

Атрибут [TearDown] означает, что помеченные им классы будут вызываться после окончания каждого теста.

Атрибут [Test] означает, что все помеченные ими классы, являются непосредственно тестами.

Таким образом, проект представляет собой следующую структуру:

```
[TestFixture]
class Test
{       public static void Main(string[] args){}
                [OneTimeSetUp] public void TestFixtureSetUp(){}
                [OneTimeTearDown] public void TestFixtureTearDown(){}
                [SetUp]  public void SetUp(){}
                [TearDown] public void TearDown(){}
                [Test] public void TEST_1(){}
                [Test]  public void TEST_2(){}
}
```

Листинг 1. Структура проекта.

Для корректного запуска тестов при помощи исполняемого файла будем использовать утилиту nunit3-console.exe, которая подключается к проекту при помощи встроенного в VisualStudio менеджера пакетов NuGet. Функция запуска тестов имеет вид:

```
public static void Main(string[] args)
{
string path = Assembly.GetExecutingAssembly().Location;
NUnit.ConsoleRunner.Program.Main(new[] { path });
}
```

Листинг 2. Функция запуска тестов.

Метод <GetExecutingAssembly().Location> получает путь до расположения загрузочного файла, а метод Program.Main, принадлежащий пространству имен <NUnit.ConsoleRunner>, вызывает утилиту nunit3-console.exe которой передается путь до скомпилированного файла.

**Конструирование тест-кейсов.** Построение тест-кейсов начинается с объявления используемых глобальных переменных:

```
public string Sname = Randomizer.CreateRandomizer().GetString(8);
public string Slogin = Randomizer.CreateRandomizer().GetString(10);
public string Sphone =
                Randomizer.CreateRandomizer().GetString(9,"1234567890");
public string Spassword = Randomizer.CreateRandomizer().GetString(6);
public string ItemName;
public string ItemCount;
public string OrderItemCount =
                Randomizer.CreateRandomizer().GetString(2, "1234567890");
```

Листинг 3. Объявление глобальных переменных.

Переменные <Sname>, <Slogin>, <Spassword>, <Sphone> хранят случайно сгенерированную информацию о пользователе сайта, которая в дальнейшем будет использоваться для доступа к приложению.

Вид класса <TestFixtureSetUp()> представлен ниже на листинге 4. Класс <TestFixtureSetUp()> имеет атрибут [OneTimeSetUp], что означает, что он будет вызываться перед стартом тестирования. При помощи метода <CreateDirectory()> создаются новые каталоги, в которых будут храниться отчет и скриншоты, а метод <GenerateReport()> вызывает функцию генерации отчета.

Определяем <driver> как указатель переменной типа <ChromeDriver>, принадлежащего пакету <Selenium.WebDriver.ChromeDriver>. Это позволит нам управлять действиями браузера Google Chrome.

Метод <Manage()> позволяет управлять настройками драйвера, с его помощью устанавливается разрешение окна приложения “На весь экран” и время ожидания до появления следующего элемента в 5 секунд.

```
[OneTimeSetUp]
public void TestFixtureSetUp()
{
if (Directory.Exists(iFolderResultTest)) Directory.Delete(iFolderResultTest, true);
DirectoryInfo dr = Directory.CreateDirectory(iFolderResultTest);
DirectoryInfo ds = Directory.CreateDirectory(iFolderScreen);
GenerateReport(0, "0", true);
ChromeOptions options = new ChromeOptions();
options.AddArguments("--ignore-certificate-errors");
options.AddArguments("--ignore-ssl-errors");
driver = new ChromeDriver(iWorkDir, options);
driver.Manage().Window.Maximize();
driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(5);
}
```

Листинг 4. Класс <TestFixtureSetUp()>.

Класс <SetUp> имеет вид:

```
[SetUp]
public void SetUp()
{                driver.Navigate().GoToUrl("http://localhost/index.php");   }
```

Листинг 5. Класс <SetUp>.

При помощи метода <Navigate()> браузер переходит на страницу тестируемого приложения перед запуском каждого теста.

Объектом для проведения тестирования было выбрано web-приложение, созданное студенткой, обучающейся по направлению подготовки «Программная инженерия», и реализующее косметику посредством сети Интернет. Вид главной страницы приложения представлен на рисунке 5:

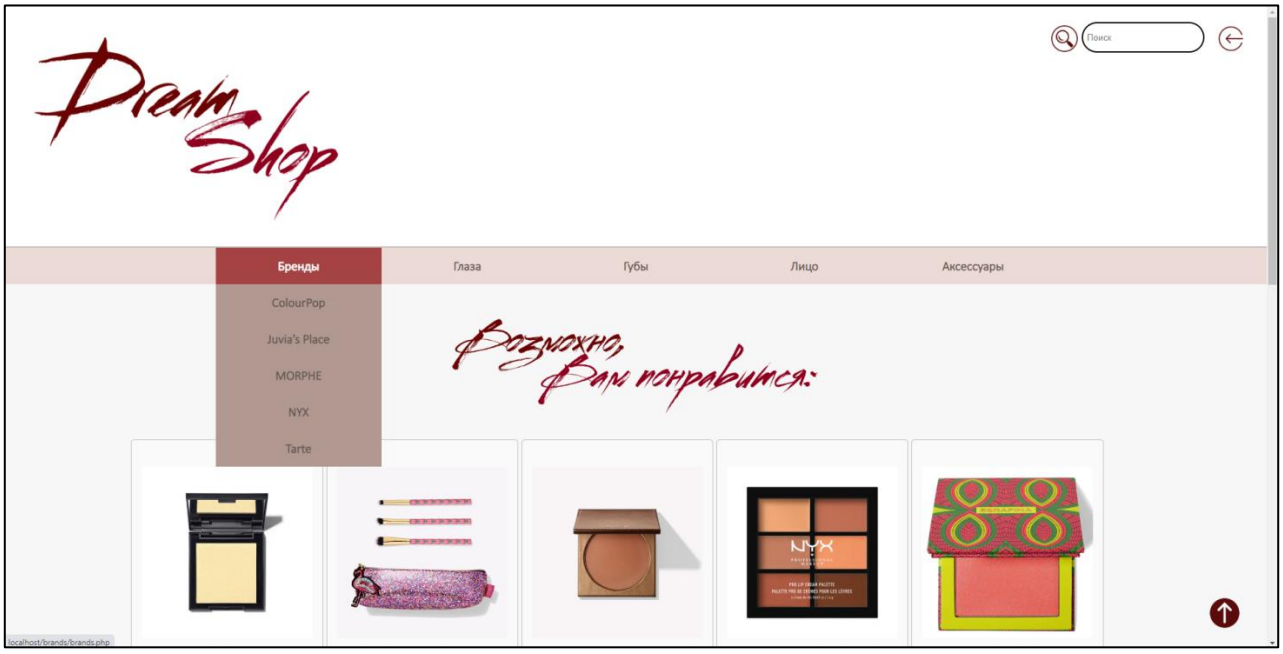


Рис. 5. Главная страница web-приложения.

Было создано 10 тест-кейсов, выполняющих проверку работоспособности основного функционала web-приложения:

- тест-кейс №1 – Проверка прохождения регистрации,
- тест-кейс №2 – Проверка прохождения авторизации,
- тест-кейс №3 – Проверка правильности отображения меню,
- тест-кейс №4 – Проверка правильности отображения товаров,
- тест-кейс №5 – Заказ товара,
- тест кейс №6 – Поиск товара,
- тест-кейс №7 – Проверка покупки повара,
- тест-кейс №8 – Проверка товара на складе,
- тест-кейс №9 – Редактирование цены товара администратором,
- тест-кейс №10 – Проверка совместимости с браузером Internet Explorer.

Подробно рассмотрим создание тестового сценария для тест-кейса №1 – Проверка прохождения регистрации.

Пакет Selenium.WebDriver предоставляет следующие методы взаимодействия с содержимым web-страницы:

Метод <FindElement()> находит на странице первый элемент, удовлетворяющий условиям поиска в зависимости от выбранного метода:

– By.Id() – поиск элемента происходит по наименованию идентифицирующего его поля Id;

– By.Name() – поиск элемента происходит по наименованию его поля Name;

– By.Xpath() – поиск элемента происходит при помощи языка запросов к элементам XML-документа XPATH

Метод <Click()> производит щелчок левой кнопкой мыши по выбранному элементу.

Метод <SendKeys()> производит ввод текста в выбранный элемент.

Код тест-сценария №1 имеет вид:

```
[Test]
public void TEST_1()
{       iTestName = "registration";
        try
{driver.FindElement(By.Id("login")).Click(); driver.FindElement(By.Id("registration")).Click();
 driver.FindElement(By.Name("FIO")).SendKeys(Sname);
 driver.FindElement(By.Name("login")).SendKeys(Slogin);
 driver.FindElement(By.Name("password")).SendKeys(Spassword);
 driver.FindElement(By.Name("phone")).SendKeys(Sphone);
 driver.FindElement(By.XPath("//button[@type='submit']")).Click();
 Assert.IsTrue(driver.FindElement(By.XPath("//*[contains(text(),'Вы успешно
зарегистрированы!')]")).Displayed);
GenerateReport(1, iTestName, true); iExecTestGood = true; }
catch (Exception ex)
{ GenerateReport(1, iTestName, false, ex.ToString()); iExecTestGood = false; }
Assert.True(iExecTestGood); }
```

Листинг 6. Код тест-сценария №1.

В результате проведения автоматизированного тестирования с помощью разработанной авторами программной системы, был получен отчет, представленный на рисунке 6. На рисунке видно, что «провалены» 4 сценария: Поиск товара, Проверка покупки товара, Проверка товара на складе и Проверка совместимости с браузером Internet Explorer:

**Тест начат 20.09.2021 12:15:16**

| Тест | Результат | Время | Снимок | Сообщение |
|---|---|---|---|---|
| Registration | Успех | 12:15:18 | - | |
| Autorization | Успех | 12:15:23 | - | |
| ListCount | Успех | 12:15:43 | - | |
| ItemOnPage Count | Успех | 12:16:01 | - | |
| Order | Успех | 12:16:18 | - | |
| ItemSearch | Провал | 12:16:29 | open | OpenQA.Selenium.NoSuchElementException: no such element: Unable to locate element: {"method":"link text","selector":"Juvia&#65533;s 5pcs Multicolored Eye Set"} (Session info: chrome=83.0.4103.106) в OpenQA.Selenium.Remote.RemoteWebDriver.UnpackAndThrowOnError(Response errorResponse) в OpenQA.Selenium.Remote.RemoteWebDriver.Execute(String driverCommandToExecute, Dictionary`2 parameters) в OpenQA.Selenium.Remote.RemoteWebDriver.FindElement(String mechanism, String value) в OpenQA.Selenium.Remote.RemoteWebDriver.FindElementByLinkText(String linkText) в OpenQA.Selenium.By.<>c__DisplayClass17_0.<LinkText>b__0(ISearchContext context) в OpenQA.Selenium.By.FindElement(ISearchContext context) в OpenQA.Selenium.Remote.RemoteWebDriver.FindElement(By by) в TESTING_SPACE.Test.TEST_6() в C:\Users\alexp\Desktop\Новая папка\Tests1\Program.cs:строка 225 |
| CheckOrder | Провал | 12:16:41 | open | OpenQA.Selenium.NoSuchElementException: no such element: Unable to locate element: {"method":"link text","selector":"Juvia&#65533;s 5pcs Multicolored Eye Set"} в OpenQA.Selenium.Remote.RemoteWebDriver.FindElementByLinkText(String linkText) в TESTING_SPACE.Test.TEST_7() в C:\Users\alexp\Desktop\Новая папка\Tests1\Program.cs:строка 248 |
| CheckItem Count | Провал | 12:17:03 | open | OpenQA.Selenium.NoSuchElementException: no such element: Unable to locate element: {"method":"xpath","selector":"//article[1]//child::img"} в OpenQA.Selenium.Remote.RemoteWebDriver.FindElementByXPath(String xpath) в TESTING_SPACE.Test.TEST_8() в C:\Users\alexp\Desktop\Новая папка\Tests1\Program.cs:строка 274 |
| Admin Change | Успех | 12:17:30 | - | |
| Internet ExplorerRun | Провал | 12:17:35 | open | OpenQA.Selenium.NoSuchElementException: Unable to find element with css selector == #logout в OpenQA.Selenium.Remote.RemoteWebDriver.FindElementById(String id) в TESTING_SPACE.Test.TEST_10() в C:\Users\alexp\Desktop\Новая папка\Tests1\Program.cs:строка 345 |

Всего тесто запущено: 10 || **Успешно: 6** || **Провалено: 4** || Процент пройденных тестов: 60% ||
Тест завершен: 20.09.2021 12:17:35

Рис. 6. Отчет о прохождении тестирования.

Благодаря сделанным при возникновении ошибки снимкам экрана (см., например, рисунок 7) были выявлены ошибки, сделанные разработчиком при написании кода web-приложения.

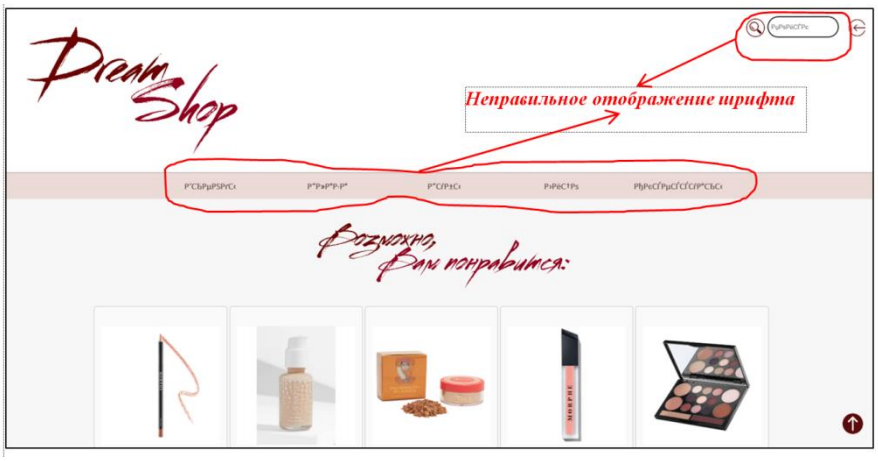


Рис. 7. Снимок экрана при возникновении ошибки тест-кейсе № 10.

После устранения выявленных ошибок было проведено повторное тестирование web-приложения, результаты которого представлены на рисунке 8:

**Тест начат 20.09.2021 12:15:16**

| Тест | Результат | Время | Снимок | Сообщение |
|---|---|---|---|---|
| Registration | Успех | 12:15:18 | - | |
| Autorization | Успех | 12:15:23 | - | |
| ListCount | Успех | 12:15:43 | - | |
| ItemOnPageCount | Успех | 12:16:01 | - | |
| Order | Успех | 12:16:18 | - | |
| ItemSearch | Успех | 12:16:29 | - | |
| CheckOrder | Успех | 12:16:41 | - | |
| CheckItemCount | Успех | 12:17:03 | - | |
| AdminChange | Успех | 12:17:30 | - | |
| InternetExplorerRun | Успех | 12:17:35 | - | |

Всего тесто запущено: 10 || **Успешно: 10** || **Провалено: 0** || Процент пройденных тестов: 100% ||
Тест завершен: 20.09.2021 12:17:35

Рис. 8. Отчет о повторном прохождении тестирования.

**Заключение.** В статье рассмотрена реализованная авторами программная система, построенная на базе фреймворка SELENIUM и предназначенная для проведения автоматизированного тестирования web-приложений. Разработанные в системе тест-кейсы могут успешно применяться для тестирования студенческих работ, выполняемых по дисциплине «Разработка web-приложений», а при согласовании с заказчиком и на основе его требований – и для тестирования коммерческих проектов.

СПИСОК ЛИТЕРАТУРЫ


1. Винокуров А. В., Лавлинская О. Ю. Уровни организации автоматизированного тестирования мобильных приложений для операционной системы ANDROID // Вестник Воронежского института высоких технологий. – 2020. – № 3 (34). – С. 22-26 [Электронный ресурс]. – Режим доступа: https://vivt.ru/downloads/vestnik/vestnik34.pdf (дата обращения: 15.09.2021).
2. Янгунаева Е. А., Янгунаев В. М. Сравнение автоматизированного и ручного подхода в тестировании веб-приложений // Научный альманах. – 2016. – № 1-1 (15). – С. 546-549 [Электронный ресурс]. – Режим доступа: https://ukonf.com/doc/na.2016.01.01.pdf (дата обращения: 15.09.2021).
3. Барвин С. К., Попов Д. В. Метрики автоматизированного тестирования web-приложения // Современные научные исследования и инновации. – 2019. – № 4 (96). – С. 4 [Электронный ресурс]. – Режим доступа: https://web.snauka.ru/issues/2019/04/89160 (дата обращения: 15.09.2021).

4. Янгунаев В. М., Янгунаева Е. А. Исследование средств семейства Selenium для автоматизированного тестирования веб-приложений // Вестник научных конференций. – 2016. – № 1-5 (5). – С. 218–219 [Электронный ресурс]. – Режим доступа: https://ukonf.com/doc/cn.2016.01.05.pdf (дата обращения: 15.09.2021).