МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(МОСКОВСКИЙ ПОЛИТЕХ)**

Факультет информационных технологий

Кафедра «Инфокогнитивные технологии»

**КУРСОВОЙ ПРОЕКТ**

на тему: «Автоматизация тестирования ПО»

Направление подготовки 09.03.03 «Прикладная информатика»

Профиль «Корпоративные информационные системы»

**Выполнил:**

студент группы 221-361

Макарова Дарья Антоновна

|  |  |
| --- | --- |
| 15.06.2024 |  |
|  | (подпись) |

Москва 2024

**ВВЕДЕНИЕ**

Целью данной курсовой работы является изучение существующих принципов и подходов к автоматизации тестирования, приобретение навыков и изучение методов автоматизированного тестирования программного обеспечения (ПО) посредством выполнения заданий курсового проекта, а также получение навыков использования популярных средств автоматизации тестирования на начальном профессиональном уровне, формирования и выполнения автоматизированных тестов.

В рамках данной курсовой работы будут решены следующие задачи:

* Выполнение заданий по тестированию фронтенда: тестирование "LambdaTest Sample App", тестирование страницы расписания на сайте Мосполитеха, тестирование Яндекс.Маркета.
* Выполнение индивидуального задания по тестированию фронтенда (тестирование сайта BMW Россия).
* Изучение материалов лекционных и практических занятий из курса LMS «Введение в автоматизированное тестирование»

Для выполнения данных задач будут использованы следующие технологии и инструменты:

* Язык программирования Java.
* Среда разработки IntelliJ IDEA.
* Maven для управления зависимостями и сборкой проекта.
* Selenium WebDriver и паттерн PageObject для автоматизации тестирования фронтенда.
* Фреймворк Maven для автоматизации сборки проектов
* AssertJ для утверждений в тестах.
* Фреймворков на выбор: JUnit 4.
* Log4J для логирования действий в автотестах.
* Allure для генерации отчетов по запущенным тестам.

**2 Теория тестирования ПО**

**Тестирование программного обеспечения** — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование — это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Цель тестирования — проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.

Тестирование ПО проводится для следующих целей:

* Для проверки соответствия требованиям.
* Для обнаружения проблем на более ранних этапах разработки и предотвращения повышения стоимости продукта.
* Обнаружения вариантов использования, которые не были предусмотрены при разработке. А также взгляд на продукт со стороны пользователя.
* Повышения лояльности к компании и продукту, т.к. любой обнаруженный дефект негативно влияет на доверие пользователей.

Все виды работ проводимых тестировщиком можно условно разделить на два[1]:

* Функциональное тестирование — тестирование ПО главная цель которого это проверка реализуемости функциональных требований приложения, т.е. способность приложения в заданных критериях решать возложенные на него (на приложение) задачи.
* Нефункциональное тестирование ПО — в первую очередь проверка на соответствие не функциональным требованиям:
  + Удобство (В основном производиться оценка удобства для пользователей)
  + Маштабируемость (проверяется как вертикальная так и горизонтальная маштабируемость тестируемого приложения)
  + Производительность (Способность работы приложения при различных нагрузках)
  + Безопасность (Защита пользовательских данных, защита данных приложения, стойкость на взлом)
  + Портируемость (Совместимость и переносимость приложения для и под различные окружения, платформы и т.д.)
  + Надежность (Поведение системы при различных непредвиденных ситуациях, способность обработки нестандартных действий пользователя)

В свою очередь, эти виды тестирования ПО имеют по множеству разнообразных разбиений по особенностям тестирования. Так же тестирование может еще подразделяться на уровни тестирования, которые в той или иной мере могут пресекаться между собой. Подробнее с этим можно ознакомится на рисунке 1.

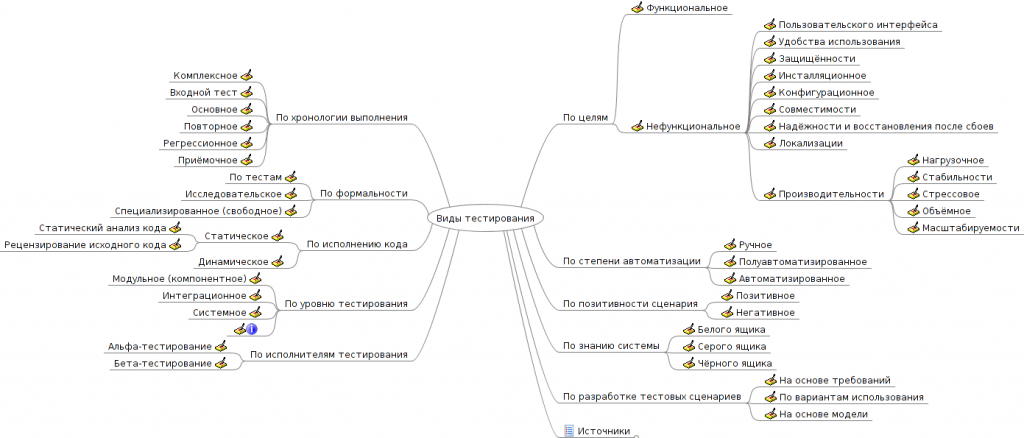


Рисунок – Виды тестирования

Автоматизация тестирования – это процесс использования программных средств для выполнения тестовых сценариев. Автоматизация тестирования позволяет улучшить качество и скорость тестирования, а также сократить затраты на тестирование.

Когда тестирование выполняется вручную, это может быть очень трудоемким процессом. Например, если тестирование включает множество шагов, которые необходимо повторить несколько раз, это может занять много времени и сил. Кроме того, при ручном тестировании возникает риск ошибок, так как человек может пропустить какую-то деталь или не заметить ошибку. Автоматизация позволяет сократить время тестирования, увеличить точность тестирования и уменьшить риск ошибок.

В целом, все тест-кейсы можно автоматизировать. Однако, не все тест-кейсы целесообразно автоматизировать. Следующие типы тестовых сценариев полезно автоматизировать:

* Тесты, которые необходимо повторять несколько раз
* Тесты, которые сложны для выполнения вручную
* Тесты, которые занимают много времени
* Тесты, которые требуют большого объема входных данных
* Тесты, которые требуют быстрого выявления ошибок

Существует множество инструментов для автоматизации тестирования, открытых и закрытых, бесплатных и платных. Вот некоторые из них:

* Selenium — наиболее популярный инструмент для автоматизации веб-приложений. Selenium можно использовать для написания тестов на многих языках программирования, включая Java, Python, C# и Ruby.
* Appium — инструмент для автоматизации тестирования мобильных приложений на платформах Android и iOS. Appium использует WebDriver протокол для автоматизации действий на устройствах.
* JMeter — инструмент для тестирования производительности и нагрузочного тестирования веб-приложений. JMeter может использоваться для тестирования различных протоколов, включая HTTP, HTTPS, FTP, SMTP и JDBC.
* TestComplete — инструмент для автоматизации тестирования десктопных, мобильных и веб-приложений. TestComplete предоставляет множество готовых модулей для автоматизации тестирования, а также позволяет написать свои собственные скрипты на языках JavaScript, Python и VBScript.
* Robot Framework — фреймворк для автоматизации тестирования, позволяющий писать тесты на многих языках программирования, включая Python, Java, .NET и Perl. Robot Framework поддерживает различные протоколы, такие как HTTP, FTP, SSH, JDBC и другие.

**3 Практическая часть**

Page Object - один из наиболее полезных и используемых архитектурных решений в автоматизации. Данный шаблон проектирования помогает инкапсулировать работу с отдельными элементами страницы, что позволяет уменьшить количество кода и его поддержку. Если, к примеру, дизайн одной из страниц изменён, то нам нужно будет переписать только соответствующий класс, описывающий эту страницу.

Основные преимущества:

* Разделение кода тестов и описания страниц
* Объединение всех действий по работе с веб-странией в одном месте

Selenium WebDriver — инструмент для автоматизации действий веб-браузера. В большинстве случаев используется для тестирования [Web-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), но этим не ограничивается. В частности, он может быть использован для решения рутинных задач администрирования сайта или регулярного получения данных из различных источников (сайтов).

Selenium WebDriver — это в первую очередь набор библиотек для различных языков программирования. Эти библиотеки используются для отправки [HTTP](https://ru.wikipedia.org/wiki/HTTP) запросов драйверу (отсюда и название WebDriver), с помощью протокола JsonWireProtocol[[5]](https://ru.wikipedia.org/wiki/Selenium#cite_note-5), в которых указано действие, которое должен совершить браузер в рамках текущей сессии. Примерами таких команд могут быть команды нахождения элементов по локатору, переход по ссылкам, парсинг текста страницы/элемента, нажатие кнопок или переход по ссылкам на странице [веб-сайта](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82). Существуют как официальные привязки библиотеки к популярным языкам программирования, так и любительские.

Для начала работы в рамках реализации паттерна PageObject были созданы следующие классы менеджеры: DriverManager, InitManager, PageManager и TestPropertiesManager. Листинг кода представлен ниже:

Класс DriverManager:

package org.example.appline.framework.managers;  
  
import org.example.appline.framework.utils.PropsConst;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.chrome.ChromeDriver;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Set;  
  
public class DriverManager {  
 private static DriverManager *INSTANCE* = null;  
 private WebDriver driver;  
 private TestPropertiesManager testPropertiesManager = TestPropertiesManager.*getInstance*();  
  
  
  
 private DriverManager() {  
 // Приватный конструктор, чтобы предотвратить создание экземпляров класса извне  
 }  
  
 public static DriverManager getInstance() {  
 if (*INSTANCE* == null) {  
 *INSTANCE* = new DriverManager();  
 }  
 return *INSTANCE*;  
 }  
  
 public WebDriver getDriver() {  
 if (driver == null) {  
 System.*setProperty*("webdriver.chrome.driver", testPropertiesManager.getProperty(PropsConst.*PATH\_CHROME\_DRIVER\_WINDOWS*));  
 driver = new ChromeDriver();  
 }  
 return driver;  
 }  
  
 public void quitDriver() {  
 if (driver != null) {  
 driver.quit();  
 driver = null;  
 }  
 }  
 public void switchWindow() {  
 Set<String> allTabs = driver.getWindowHandles();  
 List<String> tabList = new ArrayList<>(allTabs);  
  
 // Определение текущего окна  
 String currentWindow = driver.getWindowHandle();  
 int currentIndex = tabList.indexOf(currentWindow);  
  
 // Определение следующего окна  
 if (tabList.size() > 1) {  
 int nextIndex = (currentIndex + 1) % tabList.size();  
 String nextWindow = tabList.get(nextIndex);  
 driver.switchTo().window(nextWindow);  
 }  
 }  
  
  
  
 public int seeAllPages(){  
 return driver.getWindowHandles().size();  
 }  
}

Класс InitManager:

package org.example.appline.framework.managers;  
  
import org.example.appline.framework.utils.PropsConst;  
  
import java.util.concurrent.TimeUnit;  
  
import static org.example.appline.framework.utils.PropsConst.*PAGE\_LOAD\_TIMEOUT*;  
  
  
public class InitManager {  
 private static final TestPropertiesManager *props* = TestPropertiesManager.*getInstance*();  
 private static final DriverManager *driverManager* = DriverManager.*getInstance*();  
 public static void initFramework() {  
 *driverManager*.getDriver().manage().window().maximize();  
 *driverManager*.getDriver().manage().timeouts().implicitlyWait(Integer.*parseInt*(*props*.getProperty(PropsConst.*IMPLICITLY\_WAIT*)), TimeUnit.*SECONDS*);  
 *driverManager*.getDriver().manage().timeouts().pageLoadTimeout(Integer.*parseInt*(*props*.getProperty(*PAGE\_LOAD\_TIMEOUT*)), TimeUnit.*SECONDS*);  
 }  
 public static void quitFramework() {  
 *driverManager*.quitDriver();  
 }  
}

Класс PageManager:

package ru.appline.framework.managers;  
  
import ru.appline.framework.pages.task1.FirstPage1;  
import ru.appline.framework.pages.task2.DMamiPage;  
import ru.appline.framework.pages.task2.FirstPage2;  
import ru.appline.framework.pages.task2.ShedulePage;  
import ru.appline.framework.pages.task3.FirstPage3;  
import ru.appline.framework.pages.task3.LaptopsPage;  
import ru.appline.framework.pages.task4.\*;  
  
  
public class PageManager {  
  
 private static PageManager *pageManager*;  
  
 private FirstPage1 firstPage1;  
  
   
 private PageManager() {  
 }  
  
  
 public static PageManager getInstance() {  
 if (*pageManager* == null) {  
 *pageManager* = new PageManager();  
 }  
 return *pageManager*;  
 }  
  
  
 public FirstPage1 getFirstPage1() {  
 if (firstPage1 == null) {  
 firstPage1 = new FirstPage1();  
 }  
 return firstPage1;  
 }  
  
 }

Класс TestPropertiesManager:

package ru.appline.framework.managers;  
  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Properties;  
  
public class TestPropManager {  
 private final Properties properties = new Properties();  
 private static TestPropManager *INSTANCE* = null;  
  
 private TestPropManager() {  
 loadApplicationProperties();  
 loadCustomProperties();  
 }  
 public static TestPropManager getInstance() {  
 if (*INSTANCE* == null) {  
 *INSTANCE* = new TestPropManager();  
 }  
 return *INSTANCE*;  
 }  
 private void loadApplicationProperties() {  
 String nameFile = System.*getProperty*("propFile", "application");  
 try {  
 properties.load(new FileInputStream(  
 new File("src/main/resources/" +  
 nameFile + ".properties")));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 private void loadCustomProperties() {  
 properties.forEach((key, value) -> System.*getProperties*()  
 .forEach((customUserKey, customUserValue) -> {  
 if (key.toString().equals(customUserKey.toString()) &&  
 !value.toString().equals(customUserValue.toString())) {  
 properties.setProperty(key.toString(), customUserValue.toString());  
 }  
 }));  
 }  
 public String getProperty(String key, String defaultValue) {  
 return properties.getProperty(key, defaultValue);  
 }  
 public String getProperty(String key) {  
 return properties.getProperty(key);  
 }  
}

Классы-менеджеры играют важную роль в реализации паттерна PageObject. Основными преимуществами использования классов-менеджеров в PageObject являются:

* Централизованное управление конфигурацией: Классы-менеджеры отвечают за загрузку и хранение конфигурационных параметров, необходимых для работы тестов. Это может включать настройки подключения к веб-приложению, пользовательские учетные данные, URL-адреса и другие данные, которые нужно использовать в тестах.
* Повторное использование конфигурации: Классы-менеджеры позволяют легко повторно использовать одну и ту же конфигурацию во множестве тестов, что повышает последовательность и единообразие в автоматизации.
* Гибкость и масштабируемость: Если требуется изменить какую-либо конфигурацию, то необходимо внести изменения только в один класс-менеджер, а не во все тесты, использующие эту конфигурацию.
* Разделение ответственности: Классы-менеджеры отделяют логику загрузки и управления конфигурацией от бизнес-логики, описанной в PageObject. Это способствует более чёткому разделению ответственности и улучшает модульность кода.
* Единая точка доступа: Другие компоненты (например, PageObject) могут обращаться к классам-менеджерам для получения необходимых настроек, что делает код более читаемым и упрощает поддержку.

**3.1 Тестирование списка дел "LambdaTest Sample App"**

**3.1.1 Классы BasePage и FirstPage1**

Далее в рамках выполнения данного задания и реализации паттерна PageObject необходимо создать классы страниц. Для начала реализуем класс BasePage, в котором будут описываться основные свойства страницы.

Класс BasePage:

package ru.appline.framework.pages.task1;  
  
import ru.appline.framework.managers.DriverManager;  
import ru.appline.framework.managers.PageManager;  
import ru.appline.framework.managers.TestPropManager;  
import ru.appline.utils.PropConst;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.support.PageFactory;  
  
public class BasePage {  
 protected WebDriver driver;  
 private TestPropManager testPropManager = TestPropManager.*getInstance*();  
 protected PageManager pageManager = PageManager.*getInstance*();  
 protected DriverManager driverManager = DriverManager.*getInstance*();  
 public BasePage() {  
 driver = DriverManager.*getInstance*().getDriver();  
 driver.get(testPropManager.getProperty(PropConst.*LAMDA\_URL*));  
 PageFactory.*initElements*(driver, this);  
 }  
}

Далее реализуем класс FirstPage1, который будет описывать все действия и элементы, которые будут происходить на конкретной странице, в нашем случае это страница списка дел "LambdaTest Sample App"

Класс FirstPage1:

package ru.appline.framework.pages.task1;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.Keys;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
  
import java.util.List;  
  
public class FirstPage1 extends BasePage {  
  
 @FindBy(xpath = "//span[@class='ng-binding']")  
 private WebElement upText;  
  
 @FindBy(xpath = "//ul[@class='list-unstyled']/li/span")  
 private List<WebElement> spans;  
  
 @FindBy(xpath = "//input[@type='checkbox']")  
 private List<WebElement> checkboxes;  
  
 @FindBy(id = "sampletodotext")  
 private WebElement inputField;  
  
 @FindBy(xpath = "//ul/li[-1]/span")  
 private WebElement newSpan;  
  
 @FindBy(xpath = "//ul/li[-1]/input")  
 private WebElement newCheckbox;  
  
  
 @Step("Проверить, что присутствует текст: “5 of 5 remaining”")  
 public FirstPage1 getUpText() {  
 Assert.*assertEquals*("Не тот текст", "5 of 5 remaining", upText.getText());  
 return pageManager.getFirstPage1();  
 }  
  
 @Step("Проверить, что первый элемент списка не зачеркнут")  
 public FirstPage1 getSpanClass(int a) {  
 Assert.*assertTrue*(a + " элемент списка зачеркнут", spans.get(a).isDisplayed() && spans.get(a).getAttribute("class").contains("false"));  
 return pageManager.getFirstPage1();  
 }  
  
 @Step("Проверка, есть ли у первого элемента галочка")  
 public FirstPage1 isCheckboxSelected(int a) {  
 Assert.*assertFalse*("Чекбокс уже выбран", checkboxes.get(a).isSelected());  
 return pageManager.getFirstPage1();  
 }  
  
 @Step("Поставить галочку у первого элемента")  
 public FirstPage1 clickCheckbox(int a) {  
 checkboxes.get(a).click();  
 Assert.*assertTrue*(a + " чекбокс не выбран", checkboxes.get(a).isSelected());  
 return pageManager.getFirstPage1();  
 }  
  
 public void addItem(String text) {  
 inputField.sendKeys(text);  
 inputField.sendKeys(Keys.*ENTER*);  
 }  
  
 public int getRemainingCount() {  
 String remainingCountText = upText.getText().split(" ")[0];  
 return Integer.*parseInt*(remainingCountText);  
 }  
  
 @Step("Повторить шаги 3, 4 для остальных элементов списка")  
 public FirstPage1 clickAllCheckboxes() {  
  
 for(int i = 1; i < checkboxes.size(); i++){  
 getSpanClass(i);  
 clickCheckbox(i);  
 }  
 int itemCountAfter = pageManager.getFirstPage1().getRemainingCount();  
 Assert.*assertEquals*("Отображаемое число оставшихся элементов не уменьшилось до 0!", 0, itemCountAfter);  
 return pageManager.getFirstPage1();  
 }  
  
  
  
 @Step("Добавить новый элемент списка")  
 public FirstPage1 addNewElement(){  
 String newItemText = "Sixth item";  
 addItem(newItemText);  
 checkboxes.add(newCheckbox);  
 spans.add(newSpan);  
 int id = spans.size()-1;  
 getSpanClass(id);  
 clickCheckbox(id);  
 Assert.*assertTrue*("Добавленный элемент списка не зачеркнут.", spans.get(id).isDisplayed() && spans.get(id).getAttribute("class").contains("true"));  
 return pageManager.getFirstPage1();  
 }  
  
}

**3.1.2 Классы BaseTests и FirstTest**

Следующие классы, которые мы должны реализовать - BaseTests и Tests

1. BaseTests:

* Этот класс служит базовым классом для всех тестовых классов внутри пакета.
* Он содержит общую логику, необходимую для подготовки и выполнения тестов, такую как:
* Инициализация вебдрайвера
* Настройка браузера
* Управление сессией браузера
* Инициализация PageObject
* Общие методы для выполнения действий или проверок
* Класс BaseTests обеспечивает единообразие и повторное использование этой общей логики во всех тестовых классах.

Tests:

* Этот класс (или набор классов) содержит реализацию непосредственно тестовых сценариев.
* Здесь описываются последовательности действий, выполняемые на страницах веб-приложения, и проверки ожидаемых результатов.
* Тестовые методы в этих классах используют PageObject для взаимодействия с элементами приложения, что повышает читаемость и поддерживаемость кода.
* Классы Tests наследуют от BaseTests, получая доступ к общей функциональности, необходимой для выполнения тестов.

Класс BaseTests:

package task1;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.junit.After;  
import org.junit.AfterClass;  
import org.junit.Before;  
import org.junit.BeforeClass;  
import org.openqa.selenium.OutputType;  
import org.openqa.selenium.TakesScreenshot;  
import org.openqa.selenium.io.FileHandler;  
import ru.appline.framework.managers.DriverManager;  
import ru.appline.framework.managers.InitManager;  
import ru.appline.framework.managers.PageManager;  
  
import java.io.File;  
import java.io.IOException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
  
public class BaseTests {  
 protected DriverManager driverManager = DriverManager.*getInstance*();  
 protected PageManager pageManager = PageManager.*getInstance*();  
 private static final Logger *logger* = LogManager.*getLogger*(BaseTests.class);  
  
 @Before  
 public void before() {  
 InitManager.*initFramework*();  
 driverManager.getDriver();  
 *logger*.info("Driver initialized");  
 pageManager.getFirstPage1();  
 *logger*.info("Тест начат");  
 }  
  
 @AfterClass  
 public static void tearDown() {  
 InitManager.*quitFramework*();  
 *logger*.info("Framework quit");  
 }  
  
 protected void takeScreenshot(String testName) {  
 File scrFile = ((TakesScreenshot) driverManager.getDriver()).getScreenshotAs(OutputType.*FILE*);  
 String timestamp = new SimpleDateFormat("yyyyMMddHHmmss").format(new Date());  
 try {  
 FileHandler.*copy*(scrFile, new File("src/test/screenshots/" + testName + "\_" + timestamp + ".png"));  
 *logger*.info("Скриншот сохранён: " + testName + "\_" + timestamp + ".png");  
 } catch (IOException e) {  
 *logger*.error("Ошибка при сохранении скриншота: " + e.getMessage());  
 }  
 }  
  
}

Класс FirstTest:

package task1;  
  
import org.apache.log4j.LogManager;  
import org.apache.log4j.Logger;  
import org.junit.\*;  
import org.junit.rules.TestName;  
import org.junit.rules.TestWatcher;  
import org.junit.runner.Description;  
  
  
public class FirstTest extends BaseTests {  
 private static final Logger *logger* = LogManager.*getLogger*(FirstTest.class);  
 @Rule  
 public TestName testName = new TestName();  
  
 @Rule  
 public TestWatcher testWatcher = new TestWatcher() {  
 @Override  
 protected void failed(Throwable e, Description description) {  
 takeScreenshot(testName.getMethodName());  
 }  
 };  
 @Test  
 public void test() {  
 pageManager.getFirstPage1().getUpText()  
 .getSpanClass(0)  
 .isCheckboxSelected(0)  
 .clickCheckbox(0)  
 .clickAllCheckboxes()  
 .addNewElement();  
 }  
}

Как видно по листингу кода, в этих классах также реализовано логгирование с помощью Log4J и сохранение скриншотов в указанной директории при падении тестов. Посмотрим на результаты тестирования через Allure(рис. 2).

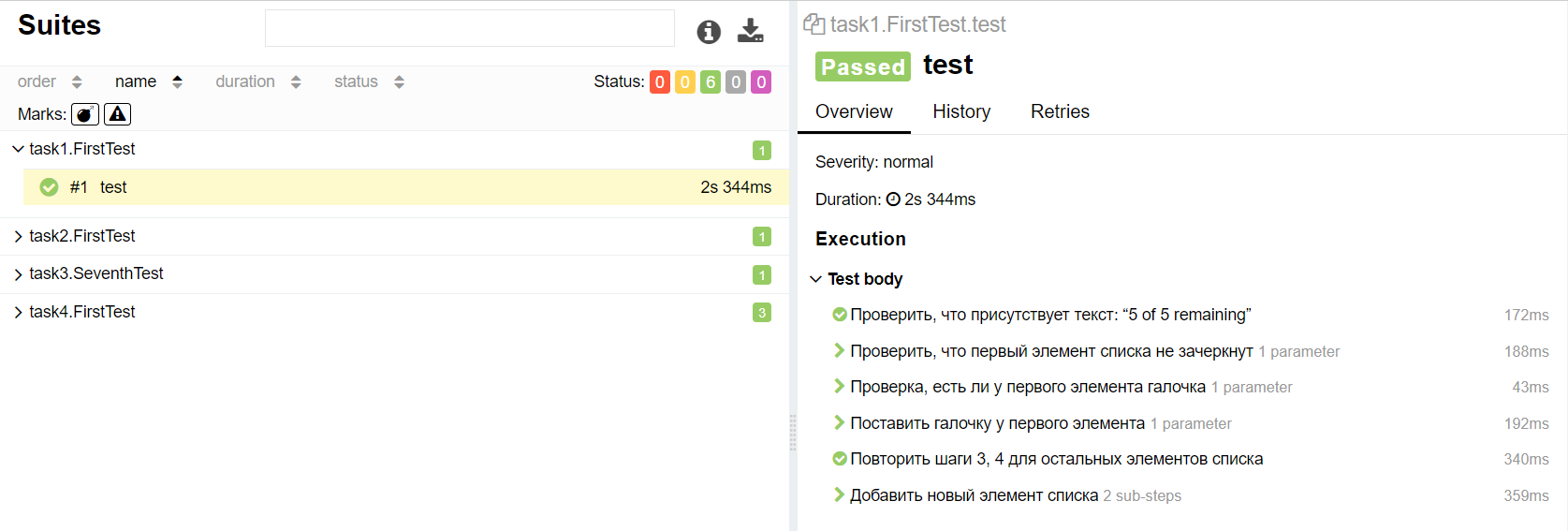


Рисунок – Отчет Allure по первому заданию

**3.2 Тестирование страницы расписания на сайте Мосполитеха**

**3.2.1 Классы страниц**

В данном тест-кейсе необходимо реализовать переход от одной страницы к другой. На примере листинга кода класса страницы из первой главы можно было заметить, что методы для работы с объектами на странице возвращают элемент класса страницы. В том случае это не было необходимостью, потому что мы работали в рамках одной страницы. В данном случае это будет необходимо для грамотного перехода от страницы к странице, допустим если в методе нажимается кнопка/ссылка на странице, которая перебрасывает на другую страницу, мы возвращаем в этом методе класс новой страницы, где уже и продолжаем работу. BasePage будет одинаковым в каждом пакете, а листинг классов всех страниц, необходимых для реализации Тест-кейса 2 представлен ниже.

Класс FirstPage2:

package ru.appline.framework.pages.task2;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.Keys;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import ru.appline.utils.PropConst;  
  
  
public class FirstPage2 extends BasePage{  
  
 @FindBy(xpath = "//a[contains(@href, \"raspisaniya\")]")  
 private WebElement shedule;  
  
 @FindBy(xpath = "//h2[contains(@class, \"main-hero-content-slide\_\_heading\")]")  
 private WebElement title;  
  
 public void loadFirstPage() {  
 driver.get(testPropManager.getProperty(PropConst.*MOSPOLYTECH\_URL*));  
 }  
  
  
  
 @Step("Нажать на кнопку Расписания")  
 public ShedulePage clickOnSchedule(){  
 Assert.*assertEquals*("Главная страница не прогрузилась", "Флагман проектного обучения в России", title.getText());  
 shedule.click();  
 return pageManager.getShedulePage();  
 }  
  
  
 public void loadHomePage() {  
 driver.get(testPropManager.getProperty(PropConst.*MOSPOLYTECH\_URL*));  
  
 }  
}

Класс ShedulePage:

package ru.appline.framework.pages.task2;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.Keys;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
  
  
public class ShedulePage extends BasePage{  
  
 @FindBy(xpath = "//a[@href = \"https://rasp.dmami.ru/\"]")  
 private WebElement lookOnWeb;  
  
 @FindBy(xpath = "//h1[contains(@class, \"h1 hero\_\_title\")]")  
 private WebElement title;  
  
  
 @Step("В разделе “Расписания занятий” нажать “Смотрите на сайте”")  
 public DMamiPage clickOnLookOnWeb(){  
 Assert.*assertEquals*("Страница расписаний не прогрузилась", "Расписания", title.getText());  
 lookOnWeb.click();  
 driverManager.switchWindow();  
 Assert.*assertEquals*("Новая вкладка не была открыта", 2, driverManager.seeAllPages());  
 return pageManager.getDMamiPage();  
 }  
  
}

Класс DMamiPage:

package ru.appline.framework.pages.task2;  
  
import io.qameta.allure.Step;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.junit.Assert;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
  
import java.time.DayOfWeek;  
import java.time.LocalDate;  
import java.time.format.TextStyle;  
import java.util.List;  
import java.util.Locale;  
  
  
  
public class DMamiPage extends BasePage{  
  
 @FindBy(xpath = "//input[@placeholder='группа ...']")  
 private WebElement inputGroup;  
  
 @FindBy(xpath = "//div[@class='found-groups row not-print']/div")  
 private List<WebElement> group;  
  
 @FindBy(xpath = "//div[@class=\"message not-print\"]")  
 private WebElement shedule;  
  
 private static final Logger *logger* = LogManager.*getLogger*(DMamiPage.class);  
  
  
  
  
 @Step("Ввести номер группы в поле поиска и выбрать предложенную группу")  
 public DMamiPage selectGroup(){  
 //ввести  
 inputGroup.sendKeys("23А-241");  
 Assert.*assertEquals*("В результатах поиска отображается несколько групп", 1, group.size());  
  
 //нажать на группу  
 group.get(0).click();  
 //Assert.assertEquals("Выведено расписание другой группы.", "23А-241", shedule.getText());  
 return pageManager.getDMamiPage();  
 }  
  
 public String getCurrentDayOfWeek() {  
 LocalDate currentDate = LocalDate.*now*();  
 DayOfWeek dayOfWeek = currentDate.getDayOfWeek();  
  
 Locale russianLocale = new Locale("ru", "RU");  
 String dayOfWeekInRussian = dayOfWeek.getDisplayName(TextStyle.*FULL*, russianLocale);  
  
 dayOfWeekInRussian = dayOfWeekInRussian.substring(0, 1).toUpperCase() + dayOfWeekInRussian.substring(1);  
  
 return dayOfWeekInRussian;  
 }  
  
 @Step("Проверка цвета текущего дня")  
 public DMamiPage checkOfColor(){  
 try {  
 WebElement todayElement = driverManager.getDriver().findElement(By.*xpath*("//div[contains(@class, 'schedule-day schedule-day\_today')]//div[contains(@class, 'bold schedule-day\_\_title') and text()='"+getCurrentDayOfWeek()+"']"));  
 Assert.*assertNotNull*("Текущий день недели не выделен цветом", todayElement);  
 WebElement todaydivElement = driverManager.getDriver().findElement(By.*xpath*("//div[contains(@class, 'schedule-day schedule-day\_today')]"));  
 String backgroundColor = todaydivElement.getCssValue("background-color");  
 Assert.*assertEquals*("Цвет фона блока 'schedule-day\_today' не равен #e2ffd9", "rgba(226, 255, 217, 1)", backgroundColor);  
  
 Thread.*sleep*(2000);  
 } catch (Exception e){  
 *logger*.info("Сегодня воскресенье! Текущий день не может быть выделен цветом.");  
 }  
 return pageManager.getDMamiPage();  
 }  
}

**2.2. Классы тестов**

Как и в первой главе и во всех последующих, где мы будем рассматривать тестирование страниц, классы с тестами будут состоять из классов BaseTests и FirstTest

Класс BaseTests:

package task2;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.BeforeClass;  
import org.openqa.selenium.OutputType;  
import org.openqa.selenium.TakesScreenshot;  
import org.openqa.selenium.io.FileHandler;  
import ru.appline.framework.managers.DriverManager;  
import ru.appline.framework.managers.InitManager;  
import ru.appline.framework.managers.PageManager;  
  
import java.io.File;  
import java.io.IOException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
  
public class BaseTests {  
 protected DriverManager driverManager = DriverManager.*getInstance*();  
 protected static PageManager *pageManager* = PageManager.*getInstance*();  
 private static final Logger *logger* = LogManager.*getLogger*(BaseTests.class);  
  
 @BeforeClass  
 public static void beforeClass() {  
 InitManager.*initFramework*();  
 *pageManager*.getFirstPage2().loadHomePage();  
  
 *logger*.info("Тест начат");  
 }  
  
 @Before  
 public void setUp() {  
 driverManager.getDriver();  
 *logger*.info("Driver initialized");  
 }  
  
 @After  
 public void tearDown() {  
 InitManager.*quitFramework*();  
 *logger*.info("Framework quit");  
 }  
  
 protected void takeScreenshot(String testName) {  
 File scrFile = ((TakesScreenshot) driverManager.getDriver()).getScreenshotAs(OutputType.*FILE*);  
 String timestamp = new SimpleDateFormat("yyyyMMddHHmmss").format(new Date());  
 try {  
 FileHandler.*copy*(scrFile, new File("src/test/screenshots/" + testName + "\_" + timestamp + ".png"));  
 *logger*.info("Скриншот сохранён: " + testName + "\_" + timestamp + ".png");  
 } catch (IOException e) {  
 *logger*.error("Ошибка при сохранении скриншота: " + e.getMessage());  
 }  
 }  
}

Класс FirstTest:

package task2;  
  
import org.apache.log4j.LogManager;  
import org.apache.log4j.Logger;  
import org.junit.Rule;  
import org.junit.Test;  
import org.junit.rules.TestName;  
import org.junit.rules.TestWatcher;  
import org.junit.runner.Description;  
  
public class FirstTest extends BaseTests {  
 private static final Logger *logger* = LogManager.*getLogger*(task1.FirstTest.class);  
 @Rule  
 public TestName testName = new TestName();  
  
 @Rule  
 public TestWatcher testWatcher = new TestWatcher() {  
 @Override  
 protected void failed(Throwable e, Description description) {  
 takeScreenshot(testName.getMethodName());  
 }  
 };  
  
 @Test  
 public void test() {  
 *pageManager*.getFirstPage2()  
 .clickOnSchedule()  
 .clickOnLookOnWeb()  
 .selectGroup();  
  
  
 }  
}

Ну и конечно посмотрим на Allure отчёт по тестированию страницы расписания Московского Политеха

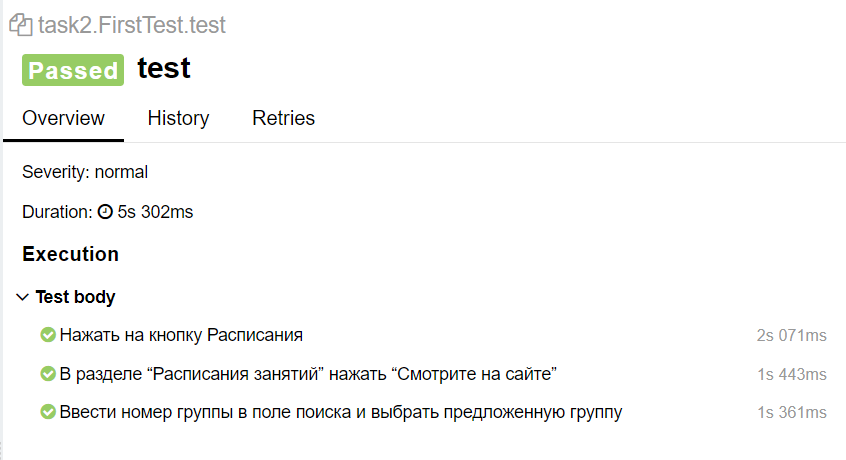


Рисунок – Отчет Allure по второму заданию

**3.3 Тестирование Яндекс.Маркета. Проверка фильтра по уцененным товарам**

**3.3.1 Классы страниц**

Для реализации Тест-Кейса 3 и выполнения данного задания нам понадобиться всего 2 класса страниц (не считая BasePage), их листинг представлен ниже.

Класс FirstPage3:

package ru.appline.framework.pages.task3;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.interactions.Actions;  
import org.openqa.selenium.support.FindBy;  
import ru.appline.utils.PropConst;  
  
public class FirstPage3 extends BasePage{  
  
 @FindBy(xpath = "//button[@class = '\_30-fz button-focus-ring Hkr1q \_1pHod \_2rdh3 \_3rbM-']")  
 private WebElement catalogButton;  
  
 @FindBy(xpath = "//span[contains(text(), 'Ноутбуки и компьютеры')]")  
 private WebElement laptopsAndPC;  
  
 @FindBy(xpath = "//a[text()='Серверы и СКС']")  
 private WebElement button;  
  
  
  
 @FindBy(xpath = "//a[text()='Ноутбуки']/..")  
 private WebElement laptops;  
  
 @FindBy(xpath = "//h1[@class = '\_3lpeU \_6tyDq \_1ea6I \_2Imo\_']")  
 private WebElement laptopsTitle;  
  
  
  
 public void loadFirstPage() {  
 driverManager.getDriver().get(testPropManager.getProperty(PropConst.*YANDEX\_URL*));  
 }  
  
 @Step("В меню “Каталог” выбрать категорию: Ноутбуки и компьютеры -> Ноутбуки и планшеты -> Ноутбуки")  
 public LaptopsPage clickOnLaptops() throws InterruptedException {  
  
 Thread.*sleep*(15000); //на капчу  
 catalogButton.click();  
 Thread.*sleep*(2000);  
 actions.moveToElement(laptopsAndPC).perform();  
 Thread.*sleep*(2000);  
  
 actions.moveToElement(button).perform();  
  
 js.executeScript("window.scrollBy(1000, 0)");  
 Thread.*sleep*(2000);  
 js.executeScript("window.scrollTo(0, 0)");  
  
  
 Thread.*sleep*(2000);  
 laptops.click();  
 Thread.*sleep*(1000);  
 Assert.*assertEquals*("Страница 'Ноутбуки' не открылась ", "Ноутбуки", laptopsTitle.getText());  
 return pageManager.getLaptopsPage();  
 }  
}

Класс LaptopsPage:

package ru.appline.framework.pages.task3;  
  
import io.qameta.allure.Step;  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.junit.Assert;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
  
import java.util.List;  
  
public class LaptopsPage extends BasePage{  
 private static final Logger *logger* = LogManager.*getLogger*(LaptopsPage.class);  
  
 @FindBy(xpath = "//div[@data-auto-themename=\"listDetailed\"]")  
 private List<WebElement> laptopItems;  
  
 @FindBy(xpath = "//div[@data-filter-value-id=\"resale\_resale\"]")  
 private WebElement filterResale;  
  
 @FindBy(xpath = "//div[@data-auto-themename=\"listDetailed\"]")  
 private List<WebElement> filterLaptopItems;  
  
  
 @Step("Вывести в лог первые 5 найденных товаров (название, цену и плашку “Уценка” при наличии)")  
 public LaptopsPage infoToLogs(){  
 *logger*.info("3 ШАГ ТЕСТ-КЕЙСА");  
  
  
 for (int i = 0; i < 5 && i < laptopItems.size(); i++) {  
  
 String name = laptopItems.get(i).findElement(By.*xpath*(".//h3[@data-auto='snippet-title']")).getText();  
 String price = laptopItems.get(i).findElement(By.*xpath*(".//span[@data-auto='snippet-price-current']")).getText();  
 boolean resale = !laptopItems.get(i).findElements(By.*xpath*(".//div[@data-zone-name = 'resale-badge']")).isEmpty();  
 if(resale){  
 *logger*.info("Ноутбук " + (i + 1) + ": \nНазвание - " + name + ", \nЦена - " + price + ", Уценка.");  
  
 }  
 else {  
 *logger*.info("Ноутбук " + (i + 1) + ": \nНазвание - " + name + ", \nЦена - " + price + ", Новый.");  
 }  
  
 }  
  
 return pageManager.getLaptopsPage();  
 }  
  
 @Step("В меню фильтров \"Состояние товара\" выбрать \"Уценка\"")  
 public LaptopsPage selectResale(){  
 filterResale.click();  
 try {  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 for (int i = 0; i < 10 && i < laptopItems.size(); i++) {  
  
 String name = filterLaptopItems.get(i).findElement(By.*xpath*(".//h3[@data-auto='snippet-title']")).getText();  
 boolean resale = !filterLaptopItems.get(i).findElements(By.*xpath*(".//div[@data-zone-name = 'resale-badge']")).isEmpty();  
 Assert.*assertTrue*(name + " не относится к уценке.", resale);  
  
 }  
  
 return pageManager.getLaptopsPage();  
 }  
  
  
}

**3.3.2 Классы тестов**

Для реализации тест кейса 3 задания 7 варианта использовались схожие с предыдущими классами BaseTests и SeventhTest. Листинг кода представлен ниже.

Класс SeventhTest:

package task3;  
  
import org.apache.log4j.LogManager;  
import org.apache.log4j.Logger;  
import org.junit.Rule;  
import org.junit.Test;  
import org.junit.rules.TestName;  
import org.junit.rules.TestWatcher;  
import org.junit.runner.Description;  
import task1.FirstTest;  
  
public class SeventhTest extends BaseTests{  
 private static final Logger *logger* = LogManager.*getLogger*(FirstTest.class);  
 @Rule  
 public TestName testName = new TestName();  
  
 @Rule  
 public TestWatcher testWatcher = new TestWatcher() {  
 @Override  
 protected void failed(Throwable e, Description description) {  
 takeScreenshot(testName.getMethodName());  
 }  
 };  
 @Test  
 public void test() throws InterruptedException {  
 *pageManager*.getFirstPage3()  
 .clickOnLaptops()  
 .infoToLogs()  
 .selectResale();  
  
  
 }  
}

Также посмотрим на результат работы. Помимо Allure-отчёта, посмотрим на логи, поскольку мы должны были вывести первые 5 ноутбуков, их цены из списка и уценены ли они.

Логи теста 3:

06:37:57.437 [main] INFO ru.appline.framework.pages.task3.LaptopsPage - 3 ШАГ ТЕСТ-КЕЙСА

06:38:02.734 [main] INFO ru.appline.framework.pages.task3.LaptopsPage - Ноутбук 1:

Название - 15.6" Ноутбук UOHUO Lightbook, Intel Celeron J4125 (2.0 ГГц), RAM 12 ГБ, SSD 512 ГБ, Intel HD Graphics 600, Windows 10 Pro,

Цена - 26 450 ₽, Новый.

06:38:08.040 [main] INFO ru.appline.framework.pages.task3.LaptopsPage - Ноутбук 2:

Название - Ноутбук 13" с двумя сенсорными экранами, Intel N100, DDR5 16ГБ, SSD 256ГБ,

Цена - 59 301 ₽, Новый.

06:38:13.268 [main] INFO ru.appline.framework.pages.task3.LaptopsPage - Ноутбук 3:

Название - Ноутбук с сенсорным экраном 11,6 дюйма, 256 ГБ, откидывается на 360 градусов,

Цена - 23 265 ₽, Новый.

06:38:18.456 [main] INFO ru.appline.framework.pages.task3.LaptopsPage - Ноутбук 4:

Название - VANWIN 11.6" сенсорный экран ноутбук, процессор Intel, 8GB RAM, 256GB SSD, Windows Pro, серебристый, ноутбук для работы и учебы,

Цена - 21 288 ₽, Новый.

06:38:23.679 [main] INFO ru.appline.framework.pages.task3.LaptopsPage - Ноутбук 5:

Название - VANWIN Ноутбук 15.6", Intel N95, RAM 16 ГБ, SSD 512 ГБ, Intel HD Graphics, Windows Pro, серый, Русская раскладка,

Цена - 34 926 ₽, Новый.

Также посмотрим на Allure-отчёт (рис. 4):

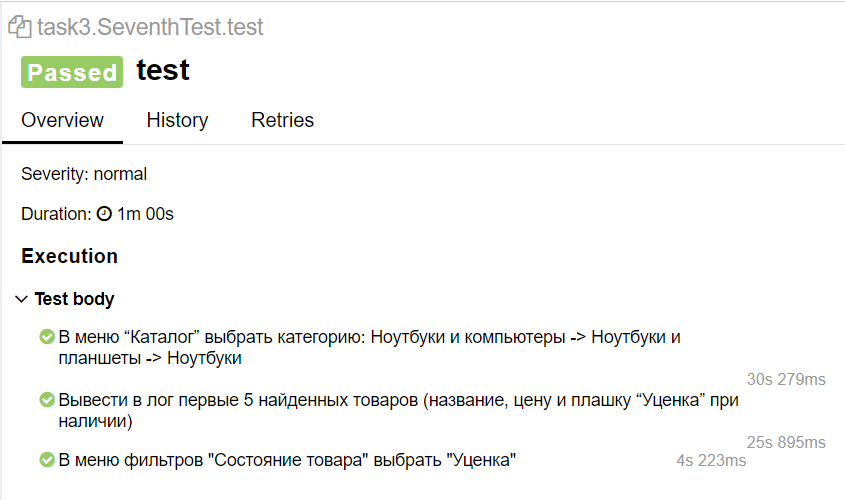


Рисунок - Allure-отчёт выполнения проверки фильтрации товаров по производителю на Яндекс.Маркете

**3.4 Тестирование сайта labirint.ru**

**3.4.1 Тест-кейсы**

В рамках задания нужно было придумать 3 тест-кейса, на основе которых тестировался функционал выбранного сайта. Тест-кейсы представлены ниже.

Тест-кейс 1 – Проверка популярных книг по количеству оценок

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: https://www.labirint.ru/ | Открывается главная страница Лабиринта |
| 1. В меню перейти в раздел «Рейтинги» | Открывается страница рейтингов по продажам |
| 1. Нажать на ссылку «Оценки» | Открывается страница рейтингов по продажам |
| 1. В выпадающем списке выбрать «Сначала популярные» | Книги расположены по убыванию количества оценок(проверить оценки первых 6 книг на убывание) |

Тест-кейс 2 – Проверка фильтрации поиска

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: https://www.labirint.ru/ | Открывается главная страница Лабиринта |
| 1. В меню перейти в раздел «Школа» → «Классы» → «9» | Откроется страница учебных пособий для 9 классов |
| 1. В меню фильтров «Для экзаменов» выбрать «ОГЭ» |  |
| 1. Выбрать цену до 600 рублей |  |
| 1. Нажать кнопку «Показать» | В выведенном списке у каждой книги цена меньше 600 рублей без учета скидки, а в названии есть число 9 и «ОГЭ» (проверить не менее 6 товаров). |

Тест-кейс 3 – Тестирование раздела «Рецензии»

|  |  |
| --- | --- |
| **Шаг** | **Ожидаемый результат** |
| 1. Перейти по ссылке: https://www.labirint.ru/ | Открывается главная страница Лабиринта |
| 1. В меню перейти в раздел «Клуб» → «Журнал» → «Рецензии читателей» | Открывается страница с рецензиями |
| 1. Нажать на выпадающее меню «Все товары» и выбрать «По автору» | Появится поле ввода |
| 1. Ввести «Лондон Джек» | В результатах отображается только искомый автор |
| 1. Нажать на найденного автора в результатах поиска | У всех книг показанных книг автор Джек Лондон |

**3.4.2 Классы страниц**

Каждый тест-кейс начинается с главной страницы, откуда потом переходит на нужный раздел из верхнего меню Листинг страниц представлен ниже.

Класс FirstPage4:

package ru.appline.framework.pages.task4;  
  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import ru.appline.utils.PropConst;  
  
public class FirstPage4 extends BasePage {  
  
 @FindBy(xpath = "//li[@data-event-content=\"Рейтинги\"]")  
 private WebElement ratingsMenu;  
  
 @FindBy(xpath = "//h1[contains(text(), \"Рейтинг: лучшие книги мира 2024 года\")]")  
 private WebElement ratingsTitle;  
  
 @FindBy(xpath = "//li[@data-event-content=\"Школа\" and contains(@class, '768')]")  
 private WebElement schoolMenu;  
  
 @FindBy(xpath = "//a[@href=\"/school/?klass[]=9#right\"]")  
 private WebElement schoolMenuClass9;  
  
 @FindBy(xpath = "//input[@name=\"klass[]\" and @value=\"9\"]")  
 private WebElement schoolClass9Checkbox;  
  
 @FindBy(xpath = "//li[@data-event-content=\"Клуб\"]")  
 private WebElement clubMenu;  
  
 @FindBy(xpath = "//a[@href=\"/reviews/\" and @class=\"b-sub-menu-sub-title\"]")  
 private WebElement clubMenuReviews;  
  
 @FindBy(xpath = "//li[contains(@class, \"b-stab-e-slider-item-m-active\")]/a")  
 private WebElement reviewsSlider;  
  
 @Step("В меню перейти в раздел «Рейтинги»")  
 public RatingsPage moveToRatings(){  
 ratingsMenu.click();  
 Assert.*assertEquals*("Страница рейтингов не загрузилась", "Рейтинг: лучшие книги мира 2024 года", ratingsTitle.getText());  
 return pageManager.getRatingsPage();  
 }  
  
 @Step("В меню перейти в раздел «Школа» → «Классы» → «9»")  
 public SchoolClass9Page moveToSchoolClass9(){  
 actions.moveToElement(schoolMenu).perform();  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 schoolMenuClass9.click();  
 Assert.*assertTrue*("Страница книг для 9 классов не открылась.", schoolClass9Checkbox.isSelected());  
 return pageManager.getSchoolClass9Page();  
 }  
  
 @Step("В меню перейти в раздел «Клуб» → «Журнал» → «Рецензии читателей»")  
 public ReviewsPage moveToReviews(){  
 actions.moveToElement(clubMenu).perform();  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 clubMenuReviews.click();  
 Assert.*assertEquals*("Страница рецензий не открылась.", "Рецензии", reviewsSlider.getText());  
 return pageManager.getReviewsPage();  
 }  
  
  
  
 public void loadFirstPage() {  
 driverManager.getDriver().get(testPropManager.getProperty(PropConst.*LABIRINT\_URL*));  
 }  
  
}

Класс RatingsPage:

package ru.appline.framework.pages.task4;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import ru.appline.utils.PropConst;  
  
public class RatingsPage extends BasePage {  
  
 @FindBy(xpath = "//a[@href=\"/rating/votes/\"]")  
 private WebElement ratingsScore;  
  
 @FindBy(xpath = "//a[@class=\"red pushstate\"]")  
 private WebElement classRatingsSales;  
  
 @Step("Нажать на ссылку «Оценки»")  
 public RatingsScorePage moveToRatingsScore(){  
 ratingsScore.click();  
 try {  
 Thread.*sleep*(5000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 Assert.*assertNotEquals*("Страница рейтинга по оценкам не прогрузилась.", "Оценки", classRatingsSales);  
 return pageManager.getRatingsScorePage();  
 }  
  
}

Класс RatingsScorePage:

package ru.appline.framework.pages.task4;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import ru.appline.utils.PropConst;  
  
import java.util.List;  
  
public class RatingsScorePage extends BasePage {  
  
  
 @FindBy(xpath = "//span[@class=\"sorting-value menu-open\"]")  
 private WebElement sortMenu;  
  
 @FindBy(xpath = "//a[@href=\"?order=popularity&way=forward\"]")  
 private WebElement popularItemSortMenu;  
  
 @FindBy(xpath = "//div[@class=\"product-padding\"]")  
 private List<WebElement> books;  
  
 @Step("В выпадающем списке выбрать «Сначала популярные»")  
 public RatingsScorePage assertFilter(){  
 Assert.*assertEquals*("Фильтр по умолчанию не \"популярные\"", "популярные", sortMenu.getText());  
 int score;  
 int lastScore = 5000;  
 for(int i = 0; i<6 && i<books.size(); i++){  
 score = Integer.*parseInt*(books.get(i).findElement(By.*xpath*(".//span[@class=\"countvotesbg\"]")).getText());  
 Assert.*assertTrue*("Порядок книг при использованных филтрах не соблюден.", lastScore > score);  
 }  
 return pageManager.getRatingsScorePage();  
 }  
  
}

Класс ReviewsPage:

package ru.appline.framework.pages.task4;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import ru.appline.utils.PropConst;  
  
import java.util.List;  
  
public class ReviewsPage extends BasePage{  
  
 @FindBy(xpath = "//span[contains(@class, \"navisort-part-4\")]")  
 private WebElement navisort;  
  
 @FindBy(xpath = "//a[@data-url=\"/reviews/?itemtype=authors\"]")  
 private WebElement navisortByAuthor;  
  
 @FindBy(xpath = "//span[@class=\"navisort-autocats-inputs\"]/input[@type=\"text\"]")  
 private WebElement authorField;  
  
 @FindBy(xpath = "//li[contains(@class, \"ui-state-focus\")]")  
 private WebElement focusAuthor;  
  
 @FindBy(xpath = "//a[@class=\"book-qtip\"]")  
 private List<WebElement> booksTitle;  
  
 @FindBy(xpath = "//input[@name=\"itemtype\"]")  
 private WebElement hiddenField;  
  
 @Step("Нажать на выпадающее меню «Все товары» и выбрать «По автору»")  
 public ReviewsPage selectSortByAuthor(){  
 navisort.click();  
 navisortByAuthor.click();  
 Assert.*assertEquals*("Поле для ввода авторов не появилось.", "authors", hiddenField.getAttribute("value"));  
 return pageManager.getReviewsPage();  
 }  
  
 @Step("Ввести «Лондон Джек»")  
 public ReviewsPage inputAuthor(){  
 authorField.sendKeys("Лондон Джек");  
 Assert.*assertEquals*("Среди выведенных авторов нет Джека Лондона", "Лондон Джек", focusAuthor.getText());  
 return pageManager.getReviewsPage();  
 }  
  
 @Step("Нажать на найденного автора в результатах поиска")  
 public ReviewsPage sortByAuthor(){  
 focusAuthor.click();  
 try {  
 Thread.*sleep*(3000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 for(int i = 0; i < booksTitle.size(); i++){  
 String title = booksTitle.get(i).getText();  
 Assert.*assertTrue*("Книга " + title + " не имеет автора 'Джек Лондон'.", title.contains("Лондон"));  
 }  
 return pageManager.getReviewsPage();  
 }  
}

Класс SchoolClass9Page:

package ru.appline.framework.pages.task4;  
  
import io.qameta.allure.Step;  
import org.junit.Assert;  
import org.openqa.selenium.By;  
import org.openqa.selenium.Keys;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import ru.appline.utils.PropConst;  
  
import java.util.List;  
  
public class SchoolClass9Page extends BasePage {  
  
 @FindBy(xpath = "//input[@name=\"examtype[]\" and @value=\"2\"]/..")  
 private WebElement ogeCheckbox;  
  
 @FindBy(xpath = "//input[@id=\"section-search-form-price\_max\"]")  
 private WebElement maxPriceInput;  
  
 @FindBy(xpath = "//input[@class=\"btn btn-primary btn-small\" and @value=\"Показать\"]")  
 private WebElement buttonSubmit;  
  
 @FindBy(xpath = "//div[@class=\"product-padding\"]")  
 private List<WebElement> books;  
  
 @Step("В меню фильтров «Для экзаменов» выбрать «ОГЭ»")  
 public SchoolClass9Page filterOge(){  
 ogeCheckbox.click();  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 return pageManager.getSchoolClass9Page();  
 }  
  
 @Step("Выбрать цену до 600 рублей")  
 public SchoolClass9Page filterPrice(){  
 maxPriceInput.sendKeys("600");  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 return pageManager.getSchoolClass9Page();  
 }  
  
 @Step("Нажать кнопку «Показать»")  
 public SchoolClass9Page clickSubmitButton(){  
 buttonSubmit.click();  
 try {  
 Thread.*sleep*(5000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 for(int i = 0; i<6 && i<books.size(); i++){  
 String title = books.get(i).findElement(By.*xpath*(".//span[@class=\"product-title\"]")).getText();  
 int price = Integer.*parseInt*(books.get(i).findElement(By.*xpath*(".//span[@class=\"price-gray\"]")).getText());  
 Assert.*assertTrue*("Книга " + title + " не соответсвует выбранным фильтрам.",  
 title.contains("ОГЭ") && price < 600);  
 }  
 return pageManager.getSchoolClass9Page();  
 }  
}

**3.4.3 Классы тестов**

В данном задании в рамках одного файла мы осуществляем 3 тест-кейса, потому в файл BaseTests вносятся изменения по анотациям Before, After, BeforeClass, AfterClass.

Класс FirstTest:

package task4;  
  
  
import org.apache.log4j.LogManager;  
import org.apache.log4j.Logger;  
import org.junit.Rule;  
import org.junit.Test;  
import org.junit.rules.TestName;  
import org.junit.rules.TestWatcher;  
import org.junit.runner.Description;  
  
public class FirstTest extends BaseTests {  
 private static final Logger *logger* = LogManager.*getLogger*(FirstTest.class);  
 @Rule  
 public TestName testName = new TestName();  
  
 @Rule  
 public TestWatcher testWatcher = new TestWatcher() {  
 @Override  
 protected void failed(Throwable e, Description description) {  
 takeScreenshot(testName.getMethodName());  
 }  
 };  
  
 @Test  
 public void test1() {  
 *pageManager*.getFirstPage4()  
 .moveToRatings()  
 .moveToRatingsScore()  
 .assertFilter();  
  
  
 }  
  
 @Test  
 public void test2() {  
 *pageManager*.getFirstPage4()  
 .moveToSchoolClass9()  
 .filterOge()  
 .filterPrice()  
 .clickSubmitButton();  
  
  
 }  
  
 @Test  
 public void test3() {  
 ;*pageManager*.getFirstPage4()  
 .moveToReviews()  
 .selectSortByAuthor()  
 .inputAuthor()  
 .sortByAuthor();  
  
  
 }  
}

Класс BaseTests:

package task4;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.junit.After;  
import org.junit.AfterClass;  
import org.junit.Before;  
import org.junit.BeforeClass;  
import org.openqa.selenium.OutputType;  
import org.openqa.selenium.TakesScreenshot;  
import org.openqa.selenium.io.FileHandler;  
import ru.appline.framework.managers.DriverManager;  
import ru.appline.framework.managers.InitManager;  
import ru.appline.framework.managers.PageManager;  
  
import java.io.File;  
import java.io.IOException;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
  
public class BaseTests {  
 protected DriverManager driverManager = DriverManager.*getInstance*();  
 protected static PageManager *pageManager* = PageManager.*getInstance*();  
 private static final Logger *logger* = LogManager.*getLogger*(task1.BaseTests.class);  
  
 @Before  
 public void before() {  
 InitManager.*initFramework*();  
 driverManager.getDriver();  
 *logger*.info("Driver initialized");  
 *pageManager*.getFirstPage4().loadFirstPage();  
 *logger*.info("Тест начат");  
 }  
  
 @AfterClass  
 public static void tearDown() {  
 InitManager.*quitFramework*();  
 *logger*.info("Framework quit");  
 }  
  
 protected void takeScreenshot(String testName) {  
 File scrFile = ((TakesScreenshot) driverManager.getDriver()).getScreenshotAs(OutputType.*FILE*);  
 String timestamp = new SimpleDateFormat("yyyyMMddHHmmss").format(new Date());  
 try {  
 FileHandler.*copy*(scrFile, new File("src/test/screenshots/" + testName + "\_" + timestamp + ".png"));  
 *logger*.info("Скриншот сохранён: " + testName + "\_" + timestamp + ".png");  
 } catch (IOException e) {  
 *logger*.error("Ошибка при сохранении скриншота: " + e.getMessage());  
 }  
 }  
  
}

Результаты 3 тест-кейсов представлены на рисунках 5-7.

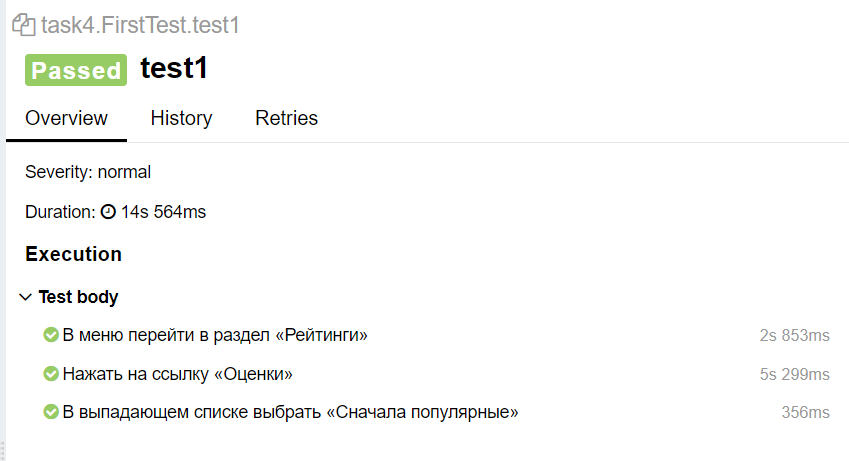


Рисунок – Отчет Allure по 1 тест-кейсу

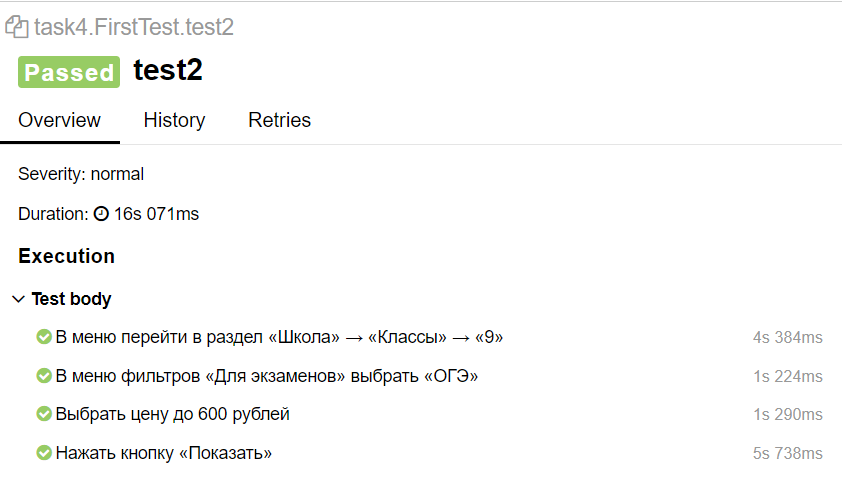


Рисунок – Отчет Allure по 2 тест-кейсу

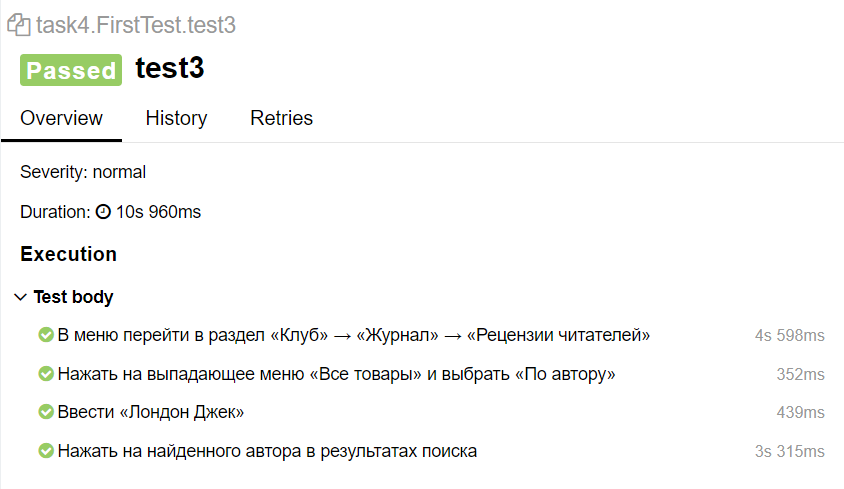


Рисунок – Отчет Allure по 3 тест-кейсу

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы были рассмотрены и реализованы основные принципы и методы автоматизированного тестирования программного обеспечения. Тестирование программного обеспечения является неотъемлемой частью процесса разработки, обеспечивая качество, надежность и соответствие продукта заявленным требованиям. Были изучены цели тестирования, включающие выявление дефектов, оценку производительности и предотвращение ошибок на ранних стадиях разработки. Рассмотрены виды тестирования, включая модульное, интеграционное, системное, приемочное и другие. Каждое из этих тестирований имеет свои цели и области применения, что позволяет комплексно оценить качество программного продукта.

Автоматизированное тестирование играет ключевую роль в современных процессах разработки ПО, позволяя значительно сократить время и затраты на тестирование, повысить качество продукта и обеспечить его соответствие требованиям пользователей. Автоматизация тестирования обеспечивает возможность регулярного выполнения тестов, что позволяет оперативно выявлять и устранять дефекты на различных этапах разработки. Были рассмотрены различные методы автоматизированного тестирования, такие как запись и воспроизведение действий пользователя, скриптовое тестирование и модельное тестирование. Каждый из методов имеет свои преимущества и области применения. В работе использовались популярные инструменты и фреймворки для автоматизации тестирования, такие как Selenium для тестирования веб-приложений, JUnit и TestNG для модульного тестирования на Java, RestAssured для тестирования REST API и Allure для генерации отчетов о выполнении тестов.

В рамках курсового проекта были выполнены задания по тестированию различных веб-приложений, включая тестирование страниц расписания на сайте Мосполитеха и страницы автомобилей с пробегом на сайте BMW. Были созданы тестовые сценарии, реализованы автотесты и сгенерированы отчеты по их выполнению. Полученные результаты подтверждают эффективность использования автоматизированного тестирования для повышения качества программного обеспечения и сокращения времени на его проверку. Автоматизация тестирования является важным инструментом в арсенале разработчиков и тестировщиков программного обеспечения. Ее использование позволяет обеспечить высокое качество продукта, повысить эффективность работы команды и сократить временные затраты на тестирование.

Полученные навыки и знания в области автоматизированного тестирования являются ценным опытом, который можно применять в будущей профессиональной деятельности для разработки и поддержки качественного программного обеспечения. Таким образом, выполненная курсовая работа позволила не только углубить теоретические знания в области тестирования ПО, но и получить практический опыт в использовании современных инструментов и методов автоматизации тестирования.

**Список литературы и интернет-ресурсов**

1. Принципы юнит-тестирования. — СПб.: Питер, 2021. — 320 с.: ил. — (Серия «Для профессионалов»).

2. Как перейти из ручного тестирования в автоматизированное [Электронный ресурс] // Habr: Свободная энциклопедия. URL: https://habr.com/ru/companies/yandex\_praktikum/articles/772240/ (дата обращения: 16.06.24)

3. Официальный сайт Selenide [Электронный ресурс]. – Режим доступа: https://ru.selenide.org (дата обращения 16.06.24).

4. Виды тестирования ПО [Электронный ресурс] // QA evolution. URL: <https://qaevolution.ru/testirovanie-po/vidy-testirovaniya-po/> (дата обращения: 16.06.24)

5. Allure Framework [Электронный ресурс] // GitHub. URL: <https://github.com/allure-framework> (дата обращения: 16.06.24)