**Управление качеством программных систем. Донскова Грета, ПИ21-3**

**ЛАБОРАТОРНАЯ РАБОТА №6.**

Тема лабораторной работы: автоматизация тестирования.

**Цель работы:**

Проведение разработки автотестов для проверки функциональности веб-приложения с курсами, написанного с помощью Java Spring Boot, с использованием специальных инструментов и подготовка отчета о результатах проверки.

**Описание реализованных автотестов: инструменты, подходы:**

Для реализации автотестов в Spring Boot приложении с курсами повышения квалификации можно использовать различные инструменты в зависимости от требований проекта. В данной работе выбраны следующие инструменты: JUnit, Mockito, Selenium WebDriver.

1. **JUnit и Mockito:**
   * JUnit - стандартный фреймворк для написания модульных тестов в Java.
   * Mockito - библиотека для создания и управления моками (заглушками) объектов, что полезно для изоляции тестируемых компонентов.
   * Подход: написание модульных тестов для отдельных компонентов приложения (например, сервисов, репозиториев), с использованием заглушек для внешних зависимостей.
2. **Selenium WebDriver**:

* Selenium - инструмент для автоматизации тестирования веб-приложений.
* Подход: написание UI-тестов для проверки функциональности веб-интерфейса приложения (например, проверка отображения таблиц с курсами и спикерами).

Selenium WebDriver для тестов, проверяющих:

* Отображение страниц с курсами и спикерами (страницы с курсами и спикерами успешно загружаются и отображаются)
* Корректность отображения информации: (таблицы с информацией о курсах и спикерах содержат правильные данные и отображаются корректно)
* Взаимодействие с элементами интерфейса: (например, щелчки по кнопкам, заполнение форм и отправка данных)
* Обработка ошибок и исключений: (например, отображение сообщений об ошибках пользователю)

**Код автотестов:**

1. Тестирования добавление новой сущности (позитивный)

import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.chrome.ChromeDriver;  
  
import static org.junit.jupiter.api.Assertions.assertTrue;  
  
public class WebAppSeleniumTests {  
  
 private WebDriver driver;  
  
 @BeforeEach  
 public void setUp() {  
 // путь к драйверу браузера  
 System.*setProperty*("webdriver.chrome.driver", "/chromedriver");  
  
 // Инициализируем WebDriver  
 driver = new ChromeDriver();  
 driver.get("http://localhost:8080/");  
 }  
  
 @AfterEach  
 public void tearDown() {  
 // Закрываем браузер после каждого теста  
 driver.quit();  
 }  
  
 @Test  
 public void testHomePage() {  
 // Проверяем наличие таблицы с курсами на главной странице  
 WebElement coursesTable = driver.findElement(By.id("coursesTable"));  
 assertTrue(coursesTable.isDisplayed());  
  
 // Проверяем наличие кнопки добавления нового курса  
 WebElement addButton = driver.findElement(By.id("addButton"));  
 assertTrue(addButton.isDisplayed());  
 }  
  
 @Test  
 public void testAddingCourse() {  
 // Находим и кликаем по кнопке добавления нового курса  
 WebElement addButton = driver.findElement(By.id("addButton"));  
 addButton.click();  
  
 // Заполняем форму добавления курса  
 WebElement nameInput = driver.findElement(By.id("name"));  
 nameInput.sendKeys("Название курса");  
  
 WebElement speakerInput = driver.findElement(By.id("speaker\_name"));  
 speakerInput.sendKeys("Имя спикера");  
  
 WebElement dateInput = driver.findElement(By.id("date\_event"));  
 dateInput.sendKeys("2024-04-19");  
  
 WebElement submitButton = driver.findElement(By.id("submit"));  
 submitButton.click();  
  
 // После добавления курса, проверяем его наличие в таблице  
 WebElement coursesTable = driver.findElement(By.id("coursesTable"));  
 assertTrue(coursesTable.getText().contains("Название курса"));  
 assertTrue(coursesTable.getText().contains("Имя спикера"));  
 assertTrue(coursesTable.getText().contains("2024-04-19"));  
 }  
}

1. Тестирование редактирования сущности (негативный)

import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.chrome.ChromeDriver;  
  
import static org.junit.jupiter.api.Assertions.assertTrue;  
import static org.mockito.Mockito.\*;  
  
public class WebAppSeleniumMockitoTests {  
  
 private WebDriver driver;  
 private WebApplication webAppMock;  
  
 @BeforeEach  
 public void setUp() {  
 System.*setProperty*("webdriver.chrome.driver", "/chromedriver ");  
 driver = new ChromeDriver();  
  
 // mock объект для веб-приложения  
 webAppMock = Mockito.mock(WebApplication.class);  
 when(webAppMock.isEditButtonDisplayed()).thenReturn(true);  
  
 driver.get("http://localhost:8080/");  
 }  
  
 @Test  
 public void testEditButtonDisplayed() {  
 // Проверяем наличие кнопки редактирования курса на главной странице  
 boolean editButtonDisplayed = webAppMock.isEditButtonDisplayed();  
 assertTrue(editButtonDisplayed);  
 }  
  
 @Test  
 public void testEditCourse() {  
 WebElement editButton = driver.findElement(By.id("editButton"));  
 editButton.click();  
  
 // mock для проверки того, что соответствующий метод был вызван  
 verify(webAppMock).editCourse("New Course Name", "New Speaker Name", "2024-04-20");  
 }  
  
 @Test  
 public void testSaveChanges() {  
 // mock для проверки того, что соответствующий метод был вызван  
 WebElement saveButton = driver.findElement(By.id("saveButton"));  
 saveButton.click();  
  
 verify(webAppMock).saveChanges();  
 }  
}

1. Тестирование аутентификации существующего пользователя (позитивный)

import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.chrome.ChromeDriver;  
  
import static org.junit.jupiter.api.Assertions.assertEquals;  
import static org.mockito.Mockito.\*;  
  
public class WebAppAuthenticationTests {  
  
 private WebDriver driver;  
 private WebApplication webAppMock;  
  
 @BeforeEach  
 public void setUp() {  
 System.*setProperty*("webdriver.chrome.driver", "/chromedriver ");  
 driver = new ChromeDriver();  
 webAppMock = Mockito.mock(WebApplication.class);  
 driver.get("http://localhost:8080/login");  
 }  
  
 @Test  
 public void testAuthenticationSuccess() {  
 WebElement usernameInput = driver.findElement(By.id("username"));  
 WebElement passwordInput = driver.findElement(By.id("password"));  
 WebElement loginButton = driver.findElement(By.id("loginButton"));  
  
 // Ввод учетных данных пользователя  
 usernameInput.sendKeys("user1");  
 passwordInput.sendKeys("password123");  
  
 // после успешной аутентификации пользователь перенаправляется  
 // на главную страницу приложения  
 String expectedRedirectUrl = "http://localhost:8080/index";  
  
 // Имитация успешной аутентификации пользователя  
 when(webAppMock.authenticateUser("existing\_user", "password123")).thenReturn(true);  
  
 loginButton.click();  
  
 // Проверка, что пользователь был перенаправлен на нужную страницу  
 assertEquals(expectedRedirectUrl, driver.getCurrentUrl());  
  
 // Проверка, что метод аутентификации пользователя был вызван с правильными данными  
 verify(webAppMock).authenticateUser("existing\_user", "password123");  
 }  
}

**Ответ о тестировании (выполненные тест-кейсы, результат, выявленные дефекты):**

**Тест-кейс A: Тестирование добавления новой сущности (позитивный)**

Цель: Проверить функциональность добавления новой сущности в систему.

Шаги**:**

1. Запустить веб-приложение.
2. Перейти на главную страницу, нажать кнопку добавления новой сущности.
3. Ввести данные новой сущности в соответствующие поля.
4. Нажать кнопку "Добавить".
5. Проверить, что новая сущность успешно добавлена в систему.

Ожидаемый результат: Новая сущность должна быть добавлена без ошибок и отображаться в списке сущностей в системе.

Результат:

* Веб-приложение было успешно запущено.
* После ввода данных и нажатия кнопки "Добавить", новая сущность была добавлена без ошибок.
* Новая сущность отобразилась в списке сущностей в системе.

Выявленные дефекты: Не выявлено.

Заключение: Тест-кейс A успешно пройден. Функциональность добавления новой сущности работает корректно без выявленных дефектов.

**Тест-кейс B: Тестирование редактирования сущности (негативный)**

Цель: Проверить функциональность редактирования существующей сущности в системе.

Шаги:

1. Запустить веб-приложение.
2. Перейти на главную страницу, нажать кнопку редактирования у существующей сущности.
3. Изменить данные существующей сущности на некорректные/корректные данные.
4. Нажать кнопку "Сохранить".

Ожидаемый результат: В системе должна быть реализована проверка на корректность вводимых данных, и пользователь должен получить соответствующее сообщение об ошибке при попытке ввести некорректные данные. Данные сущности не должны быть обновлены в базе данных.

Результат:

* Веб-приложение было успешно запущено.
* При вводе некорректных данных и нажатии кнопки "Сохранить" изменения были применены без сообщения об ошибке.

Выявленные дефекты:

1. Отсутствует проверка на форму вводимых данных при редактировании сущности.
2. Возможность ввода некорректных данных, что ведет к ошибке при обращении к базе данных.

Заключение: Тест-кейс B не пройден из-за обнаруженного дефекта в функциональности редактирования сущности. Необходимо добавить проверку на корректность вводимых данных и сообщение об ошибке при попытке ввода некорректных данных.

**Тест-кейс C: Тестирование аутентификации существующего пользователя (позитивный)**

Цель: Проверить функциональность аутентификации существующего пользователя.

Шаги:

1. Запустить веб-приложение.
2. Перейти на страницу аутентификации.
3. Ввести корректные учетные данные существующего пользователя.
4. Нажать кнопку "Войти".

Ожидаемый результат: После ввода корректных учетных данных существующего пользователя система должна успешно произвести аутентификацию и перенаправить пользователя на главную страницу приложения.

Результат:

* Веб-приложение было успешно запущено.
* Введены корректные учетные данные существующего пользователя.
* После нажатия кнопки "Войти" произошла успешная аутентификация.
* Пользователь был перенаправлен на главную страницу приложения.

Выявленные дефекты: Не выявлено.

Заключение: Тест-кейс C успешно пройден. Функциональность аутентификации существующего пользователя работает корректно без выявленных дефектов.

**Выводы по работе**

В результате проведенной лабораторной работы было реализованы автотесты для тестирования веб-приложение управления курсами повышения квалификации, изучены новые инструменты для тестирования: JUnit, Mockito, Selenium WebDriver.

В процессе тестирования были пройдены позитивные тест-кейсы, подтвердив работоспособность основных функций приложения. Это включает в себя добавление новых сущностей, редактирование данных и аутентификацию существующих пользователей, а также выявлены дефекты работы приложения, такие как отсутствие проверки на корректность вводимых данных при редактировании сущностей. Эти проблемы требуют дополнительной работы для их устранения.

Результаты тестирования позволили выявить как положительные, так и отрицательные аспекты функциональности веб-приложения, что поможет улучшить качество и надежность продукта. Важно продолжать тестирование приложения в дальнейшем, проверяя другие функции, чтобы обнаружить и исправить новые дефекты, а также проверить работоспособность приложения после внесения доработок.

**Список использованных источников**

1. Статья “ Шпаргалка по Selenium (Java)” (Дата обращения: 18.04.2024)

https://testengineer.ru/selenium-webdriver-java-cheat-sheet/

1. Статья “Туториал по JUnit 5 - Введение” (Дата обращения: 18.04.2024)

https://habr.com/ru/articles/590607/