|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | МИНОБРНАУКИ РОССИИ | МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение*  *высшего образования*  ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** | *овательное учреждение*  *высшего образования*  ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** | *Федеральное государственное бюджетное образовательное учреждение*  *высшего образования*  ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |



Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

ПРАКТИЧЕСКАЯ РАБОТА №4

по дисциплине

**«Тестирование и верификация программного обеспечения»**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы ИКБО-36-22 | Утенков Ю. Ю. |
| **Проверил:** | ассистент Петрова А. А. |

Москва 2024 г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc181472864)

[ЧАСТЬ 1. План тестирования 3](#_Toc181472865)

[ЧАСТЬ 2. Автоматизация тестирования Web-приложения 7](#_Toc181472866)

[СЦЕНАРИЙ №1 8](#_Toc181472867)

[СЦЕНАРИЙ №2 8](#_Toc181472868)

[СЦЕНАРИЙ №3 10](#_Toc181472869)

[ЗАКЛЮЧЕНИЕ 10](#_Toc181472870)

[ПРИЛОЖЕНИЕ 10](#_Toc181472871)

# ВВЕДЕНИЕ

На данный момент актуальным стандартом IEEE для документации по тестированию программного обеспечения и систем является IEEE Std 829-2019.

Стандарт IEEE Std 829 определяет требования к содержанию, стилю и оформлению тестовой документации. Он содержит рекомендации по написанию тестовых планов, сценариев, процедур, отчётов и других документов, связанных с тестированием.

Этот стандарт может быть полезен для разработчиков, тестировщиков и всех, кто участвует в процессе разработки программного обеспечения.

# ЧАСТЬ 1. План тестирования

Для выполнения данного шага было выбрано приложение, разработанное в практической работе №3, и было подробно описано для него каждый пункт плана тестирования.

1. **Идентификатор тестового плана**  
   TP-MelodyGen-001
2. **Ссылки на используемые документы**

* Спецификация требований к модулю генерации аккордов и мелодий
* Документация по коду и его структуре
* Документация по юнит-тестам модуля pytest

1. **Введение**  
   Этот тестовый план описывает тестирование модуля генерации мелодий по аккордам. Программа принимает аккорды, введенные пользователем, и генерирует мелодию на основе случайного выбора нот из аккордов. Цель — убедиться, что программа корректно создает аккорды, генерирует мелодии и обрабатывает исключительные случаи.
2. **Тестируемые элементы**

* Класс Chord
* Метод get\_random\_note класса Chord
* Класс MelodyGenerator
* Метод generate\_melody класса MelodyGenerator
* Функция create\_chord
* Основная функция main

1. **Проблемы риска тестирования ПП**

* Возможные ошибки в обработке некорректных данных (например, ввод неверных нот)
* Ошибки в работе генератора мелодий при использовании различных аккордов
* Нарушение последовательности генерации мелодии при использовании списка аккордов

1. **Особенности или свойства, подлежащие тестированию**

* Создание аккордов с верными нотами (мажорных и минорных)
* Генерация случайных нот из аккорда
* Обработка некорректного ввода данных (например, неверное качество аккорда)
* Генерация мелодии заданной длины с корректной последовательностью аккордов

1. **Особенности (свойства), не подлежащие тестированию**

* Сложные музыкальные структуры (например, добавление септаккордов или нот расширения)
* Генерация мелодий по заданной гамме или тональности
* Производительность на больших объемах данных, так как программа ориентирована на небольшие мелодии

1. **Подход**  
   Тестирование будет выполняться с использованием библиотеки pytest для выполнения юнит-тестов, описанных в коде. Будет проверено, что программа генерирует правильные аккорды, случайные ноты и корректно обрабатывает ошибки ввода. Сценарии будут выполняться вручную и автоматически.
2. **Критерии смоук-тестирования**  
   Программа должна:

* Создавать аккорды заданного качества (мажорные/минорные) для любых корневых нот.
* Корректно генерировать случайные ноты из аккордов.
* Обрабатывать некорректные значения аккордов и качеств (например, выбрасывать исключение).

1. **Критерии прохождения тестов**  
   Тесты считаются пройденными, если:

* Все аккорды создаются с верными нотами.
* Мелодия генерируется с использованием случайных нот, выбранных из каждого аккорда.
* Некорректный ввод данных вызывает ожидаемые исключения.
* Тесты, которые не содержат ожидаемых ошибок, завершаются успешно.

1. **Критерии приостановки и возобновления работ**

* Работы приостанавливаются, если встречается критическая ошибка, препятствующая выполнению тестов.
* Работы возобновляются после исправления критических ошибок в коде.

1. **Тестовая документация**

* Тест-кейсы для проверки каждого метода и функции.
* Описание ожидаемых результатов тестирования для каждого сценария.
* Лог ошибок и документация по выполнению тестов.

1. **Основные задачи тестирования**

* Проверка правильности создания аккордов
* Тестирование генерации случайных нот из аккордов
* Проверка обработки исключительных ситуаций
* Проверка генерации мелодии заданной длины

1. **Необходимый персонал и обучение**

* Разработчик, знакомый с Python и библиотекой pytest.
* Разработчик – QA-инженер для выполнения тестов и проверки результатов.

1. **Требования среды**

* Операционная система: Windows, macOS, или Linux
* Python версии 3.8 и выше
* IDE или редактор кода с установленным локально или глобально на компьютере интерпретатора python
* Установленный модуль тестирования pytest

1. **Распределение ответственности**

* Разработчик отвечает за написание и корректировку тестов.
* QA-инженер отвечает за выполнение тестов, анализ логов ошибок и отчетность.

1. **График работ (календарный план)**

* 1 день на подготовку тестов
* 1 день на запуск тестов и фиксацию ошибок
* 1 день на анализ результатов и отчетность

1. **Риски и непредвиденные обстоятельства**

* Риск изменения требований к функционалу программы
* Возможность несоответствия библиотек и зависимостей, что может вызвать ошибки окружения

1. **Утверждение плана тестирования**

* План утверждается преподавателем по дисциплине ТиВПО.

1. **Глоссарий**

* *Аккорд* — сочетание нот, используемое в музыкальной композиции.
* *Мелодия* — последовательность нот, выбранных для формирования музыкального мотива.
* *Мажорный аккорд* — это группа трех звуков (или нот), которые вместе образуют аккорд с положительной эмоциональной нагрузкой и обычно ассоциируются с радостью или оптимизмом.
* *Минорный аккорд* — это группа трех звуков (или нот), которая отличается от мажорного аккорда по высоте второй степени и обычно ассоциируется с более серьезным или грустным настроем.
* *Юнит-тест* — тестирование отдельных модулей программы.
* *Pytest* — инструмент тестирования для Python, использующий автодетекцию тестов.

# ЧАСТЬ 2. Автоматизация тестирования Web-приложения

Для выполнения данного шага было выполнено:

1. Был выбран программный продукт selenium ide для разработки скриптов для тестирования Web-приложения;
2. Был выбран один Web-ресурс – NASA.GOV, для проведения тестирования;
3. Разработано три скрипта для проверки корректности работы веб-ресурса;
4. Скриншоты результатов проведения тестирования представлены на рисунках ниже.

Все сценарии тестирования проводились в браузере Mozilla Firefox.

## **СЦЕНАРИЙ №1**

Для данного сценария было выбрано соответствие текста на главной странице, посвящённой рекомендуемым последним новостям, тестирование проводилось по данному URL: <https://www.nasa.gov/>

Успешное прохождение данного сценария представлено на рисунке 1.

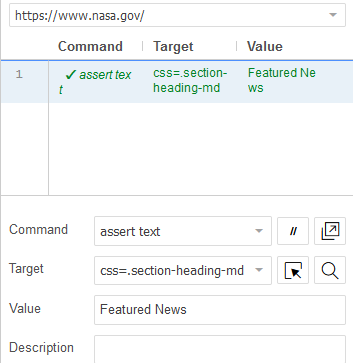


Рисунок 1 – успешное прохождение первого сценария

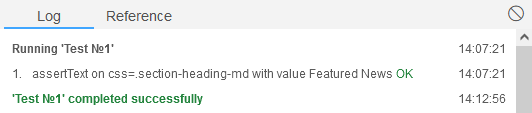


Рисунок 2 – логи первого сценария, сообщающие об успешном прохождении его

## **СЦЕНАРИЙ №2**

В данном сценарии есть три теста, проверяющие нажатие определённых чекбоксов по данному URL: <https://www.nasa.gov/missions/>

Успешное прохождение данного сценария представлено на рисунке 2.

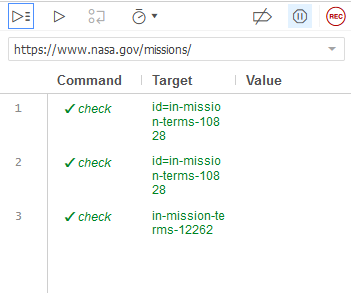


Рисунок 2 – успешное прохождение второго сценария с тремя тестами



Рисунок 3 – отображение первого чекбокса с заданным id в html через инспектор



Рисунок 4 – отображение второго чекбокса с заданным id в html через инспектор



Рисунок 5 – отображение третьего чекбокса с заданным id в html через инспектор

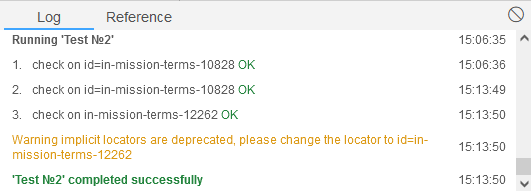


Рисунок 6 – логи второго сценария, сообщающие об успешном прохождении его

## **СЦЕНАРИЙ №3**

В данном сценарии проверялось правильность перехода на другую страницу сайта, базовый URL: <https://science.nasa.gov/universe/>

Успешное прохождение данного сценария представлено на рисунке 7.

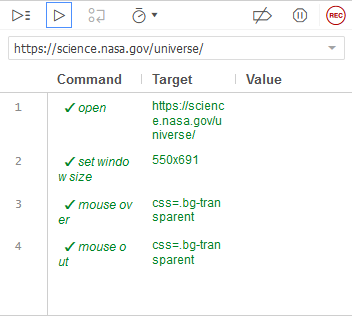


Рисунок 7 – успешное прохождение третьего сценария с 4 тестами

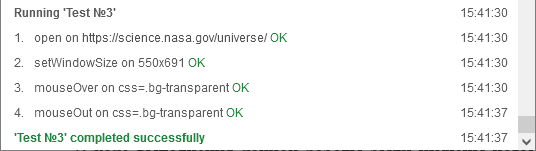


Рисунок 8 – логи третьего сценария, сообщающие об успешном прохождении его

# ЗАКЛЮЧЕНИЕ

В данной работы был получен навык работы с расширением selenuim ide для тестирования веб – ресурсов на определённые действия.

# ПРИЛОЖЕНИЕ

Код, для тестирования веб – приложений представлен на листингах 1 – 3.

|  |
| --- |
| // Generated by Selenium IDE import org.junit.Test; import org.junit.Before; import org.junit.After; import static org.junit.Assert.\*; import static org.hamcrest.CoreMatchers.is; import static org.hamcrest.core.IsNot.not; import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import org.openqa.selenium.firefox.FirefoxDriver; import org.openqa.selenium.chrome.ChromeDriver; import org.openqa.selenium.remote.RemoteWebDriver; import org.openqa.selenium.remote.DesiredCapabilities; import org.openqa.selenium.Dimension; import org.openqa.selenium.WebElement; import org.openqa.selenium.interactions.Actions; import org.openqa.selenium.support.ui.ExpectedConditions; import org.openqa.selenium.support.ui.WebDriverWait; import org.openqa.selenium.JavascriptExecutor; import org.openqa.selenium.Alert; import org.openqa.selenium.Keys; import java.util.\*; import java.net.MalformedURLException; import java.net.URL; public class Test1Test {  private WebDriver driver;  private Map<String, Object> vars;  JavascriptExecutor js;  @Before  public void setUp() {  driver = new FirefoxDriver();  js = (JavascriptExecutor) driver;  vars = new HashMap<String, Object>();  }  @After  public void tearDown() {  driver.quit();  }  @Test  public void test1() {  assertThat(driver.findElement(By.cssSelector(".section-heading-md")).getText(), is("Featured News"));  } } |

Листинг 1 – код, для тестирования первого сценария

|  |
| --- |
| // Generated by Selenium IDE import org.junit.Test; import org.junit.Before; import org.junit.After; import static org.junit.Assert.\*; import static org.hamcrest.CoreMatchers.is; import static org.hamcrest.core.IsNot.not; import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import org.openqa.selenium.firefox.FirefoxDriver; import org.openqa.selenium.chrome.ChromeDriver; import org.openqa.selenium.remote.RemoteWebDriver; import org.openqa.selenium.remote.DesiredCapabilities; import org.openqa.selenium.Dimension; import org.openqa.selenium.WebElement; import org.openqa.selenium.interactions.Actions; import org.openqa.selenium.support.ui.ExpectedConditions; import org.openqa.selenium.support.ui.WebDriverWait; import org.openqa.selenium.JavascriptExecutor; import org.openqa.selenium.Alert; import org.openqa.selenium.Keys; import java.util.\*; import java.net.MalformedURLException; import java.net.URL; public class Test2Test {  private WebDriver driver;  private Map<String, Object> vars;  JavascriptExecutor js;  @Before  public void setUp() {  driver = new FirefoxDriver();  js = (JavascriptExecutor) driver;  vars = new HashMap<String, Object>();  }  @After  public void tearDown() {  driver.quit();  }  @Test  public void test2() {  {  WebElement element = driver.findElement(By.id("in-mission-terms-10828"));  if (!element.isSelected()) {  element.click();  }  }  {  WebElement element = driver.findElement(By.id("in-mission-terms-10873"));  if (!element.isSelected()) {  element.click();  }  }  {  WebElement element = driver.findElement(By.id("in-mission-terms-12262"));  if (!element.isSelected()) {  element.click();  }  }  } } |

Листинг 2 – код, для тестирования второго сценария

|  |
| --- |
| // Generated by Selenium IDE import org.junit.Test; import org.junit.Before; import org.junit.After; import static org.junit.Assert.\*; import static org.hamcrest.CoreMatchers.is; import static org.hamcrest.core.IsNot.not; import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import org.openqa.selenium.firefox.FirefoxDriver; import org.openqa.selenium.chrome.ChromeDriver; import org.openqa.selenium.remote.RemoteWebDriver; import org.openqa.selenium.remote.DesiredCapabilities; import org.openqa.selenium.Dimension; import org.openqa.selenium.WebElement; import org.openqa.selenium.interactions.Actions; import org.openqa.selenium.support.ui.ExpectedConditions; import org.openqa.selenium.support.ui.WebDriverWait; import org.openqa.selenium.JavascriptExecutor; import org.openqa.selenium.Alert; import org.openqa.selenium.Keys; import java.util.\*; import java.net.MalformedURLException; import java.net.URL; public class Test3Test {  private WebDriver driver;  private Map<String, Object> vars;  JavascriptExecutor js;  @Before  public void setUp() {  driver = new FirefoxDriver();  js = (JavascriptExecutor) driver;  vars = new HashMap<String, Object>();  }  @After  public void tearDown() {  driver.quit();  }  @Test  public void test3() {  driver.get("https://science.nasa.gov/universe/");  driver.manage().window().setSize(new Dimension(550, 691));  {  WebElement element = driver.findElement(By.cssSelector(".bg-transparent"));  Actions builder = new Actions(driver);  builder.moveToElement(element).perform();  }  {  WebElement element = driver.findElement(By.tagName("body"));  Actions builder = new Actions(driver);  builder.moveToElement(element, 0, 0).perform();  }  } } |

Листинг 3 – код, для тестирования третьего сценария

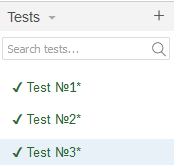


Рисунок 9 – все пройденные сценарии