МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: UI-тестирование сайта https://github.com

Студент гр. 3388 _____ Павлов А.Р

Студент гр. 3384 _____ Траксель В.С.

Студентка гр. 3384 _____ Мокрушина В.Л.

Руководитель _____ Шевелева А.М.

Санкт-Петербург

2025

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Павлов А.Р. группы 3388

Студент Траксель В.С. группы 3384

Студентка Мокрушина В.Л. группы 3384

Тема практики: ui-тестирование сайта

Задание на практику:

Командная разработка тестов для сайта https://github.com, тестирование раздела issues.

Сроки прохождения практики: 25.06.2024 – 08.07.2024

Дата сдачи отчета: 05.07.2024

Дата защиты отчета: 05.07.2024

Студент	_____	Павлов А.Р.
Студент	_____	Траксель В.С.
Студентка	_____	Мокрушина В.Л.
Руководитель	_____	Шевелева А.М.

АННОТАЦИЯ

Цель практики — изучение и отработка навыков UI-тестирования веб-приложений на примере сайта https://github.com, с фокусом на функциональность вкладки issues. В ходе практики были реализованы тесты основных сценариев работы с issue: создание новой задачи, добавление комментариев, назначение исполнителей, установка меток, редактирование заголовка, закрытие и повторное открытие issue. Также проверялась блокировка комментариев, закрепление задач в списке и ограничения действий для пользователей без прав. Каждый тест включал переход на соответствующую страницу, взаимодействие с элементами интерфейса и проверку корректности отображения изменений и сообщений об ошибках. Практика позволила закрепить навыки автоматизации UI-тестов и проверки прав доступа пользователей.

SUMMARY

The purpose of the practice is to study and practice the skills of UI testing of web applications using the example of a website https://github.com , with a focus on the functionality of the issues tab. During the practice, tests of the main scenarios for working with the issue were implemented: creating a new task, adding comments, assigning performers, setting labels, editing the title, closing and reopening the issue. Blocking comments, pinning tasks in the list, and restricting actions for users without rights were also checked. Each test included navigating to the appropriate page, interacting with interface elements, and checking whether changes and error messages were displayed correctly. The practice allowed us to consolidate the skills of automating UI tests and verifying user access rights.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Выполнение работы должно сопровождаться следующими пунктами:

Github: каждая группа из 3х человек работает в своем репозитории. Каждый участник группы должен делать коммиты. При этом в readme необходимо прописать кто за что отвечает.

Чеклист: подробное описание созданных тестов в одном файле, лучше всего в виде таблицы оформлять. Что должно быть:

- название системы, которую тестируете
- название тестов (например, Проверка добавления комментариев к посту)
- входные данные (по возможности) – это данные для входа в систему.
- описание теста – то, как это бы делал пользователь. Например, авторизация под пользователем, открытие меню «Профиль», открытие вкладки «Все посты», нажатие на кнопку «Комментарий» под постом «Тестовый пост», заполнение поля комментария значением «Комментарий» и т.д.
- описание ожидаемого результата – комментарий успешно отправился, изображение прикреплено.

Нужно написать 10 тестов на команду по одному определенному блоку / функционалу системы. Например, работа с постами в ВКонтакте – создание поста, поделиться постом на своей странице, добавить комментарий к посту, лайкнуть пост, поделиться постом в сообщении, удалить пост, закрепить пост, добавить в архив, отключить комментарии. Выбирайте систему, куда можете войти.

Технологии: Java, Selenide (Selenium), Junit, Maven, логирование.

1. РЕАЛИЗУЕМЫЕ ТЕСТЫ

1.1. Чеклист

1. Создание новой issue:
Переход на страницу issues репозитория, нажатие кнопки "New issue", заполнение полей title и description, нажатие кнопки "Create".

2. Добавление комментария к существующей issue:

Переход на страницу указанной issue, заполнение поля comment, нажатие кнопки "Comment".

3. Назначение исполнителя (assignee) на существующую issue:

Переход на страницу указанной issue, нажатие action button "Assignees", выбор тестового пользователя в выпадающем чекбоксе.

4. Добавление метки (label) к issue:

Переход на страницу указанной issue, нажатие action button "Labels", выбор тестовой метки в выпадающем чекбоксе. Проверка существования метки: ввод в поле поиска метки названия несуществующей метки.

5. Редактирование заголовка существующей issue:

Переход на страницу указанной issue, нажатие кнопки "Edit", заполнение поля title, нажатие кнопки "Save"

6. Закрытие issue:

Переход на страницу указанной issue, нажатие кнопки "Close issue".

7. Повторное открытие ранее закрытой issue:

Переход на страницу указанной issue, нажатие кнопки "Reopen issue".

8. Блокировка комментариев:

Первый пользователь: переход на страницу указанной issue, нажатие кнопки "Lock conversation", нажатие кнопки "Lock conversation" в появившемся pop up окне. Второй пользователь: переход на страницу указанной issue.

9. Закрепление issue в списке:

Переход на страницу указанной issue, нажатие кнопки "Pin issue".

10. Проверка выполнения действия без прав:

Второй пользователь: переход на страницу указанной issue.

2. ОПИСАНИЕ КЛАССОВ И МЕТОДОВ И UML ДИАГРАММА

2.1. Описание классов и методов

Класс *BaseComponent* представляет собой абстрактный базовый класс для всех UI-компонентов в проекте, реализующем автоматизированное тестирование веб-интерфейса с использованием библиотеки Selenide. Его основная цель - предоставить набор универсальных вспомогательных методов для работы с элементами пользовательского интерфейса.

Класс содержит логгер log, что позволяет фиксировать действия, происходящие при взаимодействии с элементами в дочерних классах. Все методы получения элементов инкапсулируют типовые способы поиска: по id, class, тексту, xpath, CSS-селектору и имени. Эти методы статичны, так как не зависят от состояния экземпляра компонента.

Помимо этого, класс содержит методы для проверки видимости элементов (isVisible) и ожидания их появления или исчезновения на странице (waitForVisible и waitForNotVisible). Это критически важно при написании стабильных UI-тестов, так как позволяет управлять синхронизацией и избежать ошибок, связанных с попыткой взаимодействия с элементами, которые ещё не отображены или уже скрыты.

Класс *BasePage* представляет собой абстрактный базовый класс для всех страниц, реализованных с использованием библиотеки Selenide. Он инкапсулирует общее поведение и служит основой для всех конкретных страниц.

В конструкторе класса автоматически выполняется открытие страницы - в лог записывается сообщение с именем текущего класса.

Метод getCurrentUrl() возвращает текущий URL активной страницы, используя API Selenide, а метод getPageTitle() - заголовок страницы, что позволяет тестам проверять правильность навигации и содержимого. Метод `isUrlContains(String expectedUrlPart)` предоставляет удобный способ

удостовериться, что текущий URL содержит ожидаемый фрагмент, например, при проверке редиректов или параметров запроса.

Также в классе присутствует защищённый метод waitForPageLoad(), который по умолчанию лишь записывает отладочную информацию, но может быть переопределён в наследниках для реализации специфических условий ожидания полной загрузки страницы.

Класс *BaseTest* служит абстрактной основой для всех тестов в проекте и инкапсулирует типовые действия, необходимые для подготовки и завершения тестового окружения. Его основная задача это управлять конфигурацией среды, запуском браузера, авторизацией и завершением тестов.

Перед каждым тестом выполняется метод setUp(), помеченный аннотацией @BeforeEach. В нём логгируется начало выполнения теста, создаётся экземпляр сервиса авторизации AuthService, вызывается метод configureSelenide(), настраивающий параметры окружения, и далее происходит открытие стартовой страницы приложения через метод openApplication().

Метод configureSelenide() читает параметры из конфигурационного файла при помощи утилиты ConfigReader, включая базовый URL, размер окна браузера, таймаут ожидания и режим headless. Здесь также устанавливаются значения по умолчанию, если параметры отсутствуют. Помимо этого, отключается сохранение исходного кода страницы, включается создание скриншотов и задаётся папка для отчётов. Вся информация о конфигурации логгируется для контроля.

Метод openApplication() открывает стартовую страницу по заранее определённому URL и также пишет об этом в лог.

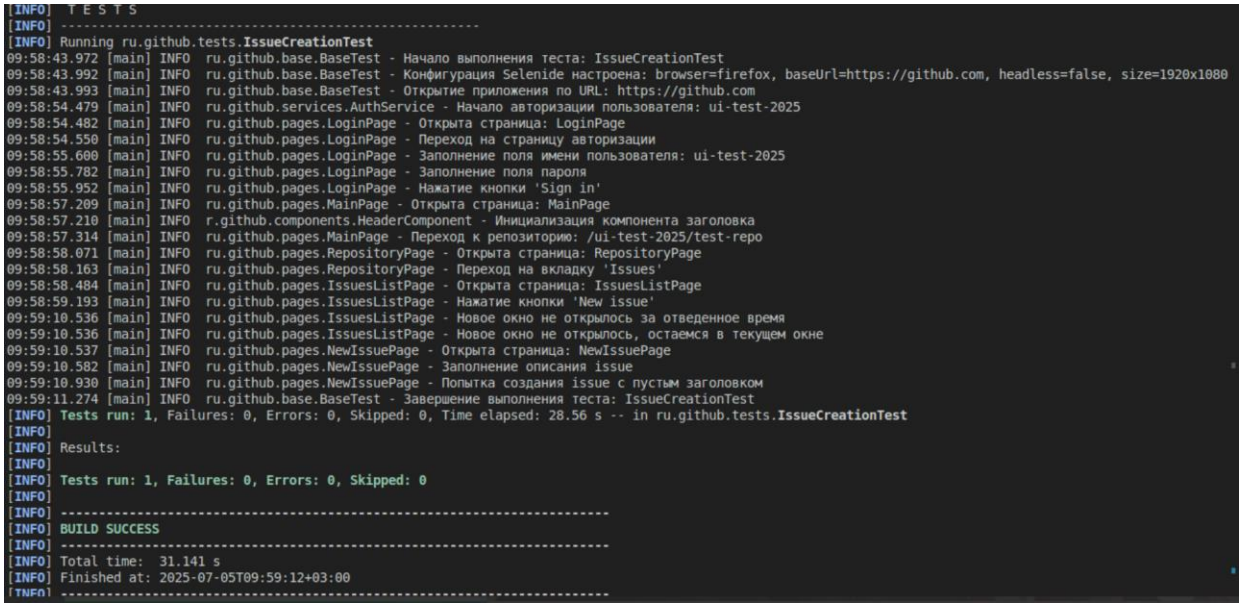
После выполнения каждого теста запускается метод tearDown(), помеченный @AfterEach, который закрывает веб-драйвер, завершая сессию браузера, и логгирует завершение работы теста.

Также класс содержит статическое поле REPOSITORY_PATH, формируемое на основе настроек из testData. Использование AuthService упрощает авторизацию в системе.

2.2. UML диаграмма

3. ТЕСТИРОВАНИЕ

3.1. Демонстрация отработки тестов



ЗАКЛЮЧЕНИЕ

В ходе практики была реализована система автоматизированного UI-тестирования функциональности вкладки ***Issues*** на сайте https://github.com. Цель практики — освоение инструментов и подходов к UI-тестированию веб-приложений, а также закрепление навыков работы с библиотекой ***Selenide*** и паттерном ***Page Object Model (POM***).

В качестве основы использовался язык ***Java*** и библиотека ***Selenide***, использующую технологию ***Selenium WebDriver***. Архитектура проекта строилась по шаблону ***POM***, где каждая веб-страница и её элементы выделены в отдельные классы. Это повысило читаемость и помогло избежать лишних повторений кода, а также упростило поддержку тестов при изменении структуры интерфейса. Проект был организован как Maven-проект — это позволило централизованно управлять зависимостями, структурой каталогов и конфигурацией сборки. Благодаря ***Maven*** стало возможным удобно управлять версиями библиотек, использовать конфигурацию на разных устройствах, а также гарантировать воспроизводимость сборки. Эта технология наиболее важна для проектов, где автоматизация тестирования является неотъемлемой частью разработки и сопровождения ПО.

Тесты охватили широкий спектр сценариев использования: от создания новой ***issue*** и добавления комментариев, до блокировки обсуждений и проверки прав доступа. Для каждого сценария был написан отдельный тест-кейс, содержащий чёткие пошаговые действия и проверки результата, соответствующие ожидаемому поведению UI. Были протестированы как базовые, так и составные элементы интерфейса: кнопки, поля ввода, чекбоксы, выпадающие списки и даже сообщения об ошибках.

Особое внимание было уделено правильному построению классов элементов с наследованием от ***BaseElement*** и страниц от ***BasePage***, а также внедрению сервисов и базовых классов для автоматического ввода логина и настройки тестов. Таким образом были выведена логика взаимодействия с интерфейсом от логики самих тестов и улучшена масштабируемость проекта.

Анализируя полученные результаты, можно сказать, что в ходе работы удалось не только автоматизировать тестирование заданного функционала, но и выстроить оптимальную архитектуру для итогового проекта. Разработанный набор тестов не только выполняет проверку корректности работы вкладки ***Issues***, но и демонстрирует принципы профессиональной организации UI-тестирования в современной разработке.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация для работы с технологией Selenium // Наименование сайта. URL: https://www.selenium.dev/documentation/ (дата обращения: 02.07.2025).

2. Документация для работы с ресурсом github // Наименование сайта. URL: https://docs.github.com/ru (дата обращения: 02.07.2025).

3. Статья по работе с Java Selenium // Наименование сайта. URL: https://uproger.com/chit-list-selenium-java (дата обращения: 02.07.2025).

4. Статья о ui-тестировании / Шевелева А.М.