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

ОТЧЕТ

по учебной практике

Тема: UI-тестирование сайта 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 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 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. Описание тестов

1. Создание новой issue:

Тест посвящён созданию новой issue. В нём осуществляется переход на страницу Issues репозитория, где с помощью кнопки New issue открывается форма создания задачи. Проверяется корректность работы следующих полей: обязательного поля Title, поля Description, а также кнопки Create issue, которая должна становиться активной при наличии заголовка. После создания проверяется, что новая issue появилась в списке и содержит введённые данные.

2. Добавление комментария к существующей issue:

Второй тест проверяет добавление комментария к существующей issue. Тест автоматически переходит на страницу задачи, вводит текст в поле Write, нажимает кнопку Comment, после чего осуществляется проверка появления комментария на странице конкретной issue, с точным соответствием текста.

3. Назначение исполнителя assignee на существующую issue:

Проверка на открытие меню Assignees, выбор нужного пользователя из выпадающего списка с чекбоксами, затем проверяется, что имя исполнителя отображается в соответствующем блоке задачи.

4. Добавление метки label к issue:

Тест проверяет добавление метки label к выбранной задаче. Осуществляется открытие меню Labels, выбор нужной метки из списка и её подтверждение. Также выполняется поиск несуществующей метки через поле фильтрации. В результате проверяется, что выбранная метка отображается как на странице issue, так и в общем списке задач.

5. Редактирование заголовка существующей issue:

В пятом тесте осуществляется проверяется функция редактирование заголовка issue. Осуществляется переход на страницу задачи, нажимается кнопка Edit , изменяется содержимое поля Title, и нажимается кнопка Save. После выполнения проверяется, что заголовок обновлён на странице и корректно отображается в списке всех issues.

6. Закрытие issue:

Нажимается кнопка Close issue, после чего проверяется наличие индикатора Closed рядом с заголовком, а также соответствующее изменение доступных действий, включая возможное скрытие поля комментариев.

7. Повторное открытие ранее закрытой issue:

Седьмой тест дополняет проверяет возможность повторного открытия ранее закрытой issue. На странице задачи нажимается кнопка Reopen issue, после чего проверяется, что статус Closed исчезает, а кнопка закрытия снова становится доступной.

8. Блокировка комментариев:

Пользователь нажимает кнопку Lock conversation и подтверждает действие в модальном окне, затем другой пользователь переходит на ту же issue и проверяет отсутствие возможности оставлять комментарии, а также наличие информационного сообщения о блокировке.

9. Закрепление issue в списке:

Девятый тест проверяет возможность закрепления issue в списке. Нажимается кнопка Pin issue, после чего задача должна отображаться вверху списка с иконкой закрепления pin.

10. Проверка выполнения действия без прав:

Тест направлен на проверку ограничений прав доступа. Пользователь, не обладающий необходимыми правами, переходит на страницу задачи. Проверяется, что такие элементы, как Edit, Close issue, Assignees, Labels, Pin issue и другие действия недоступны для взаимодействия.

2. ОПИСАНИЕ КЛАССОВ И МЕТОДОВ И UML ДИАГРАММА

2.1. Описание классов и методов

BaseComponent.java

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 вспомогательных методов для работы с элементами пользовательского интерфейса.

Класс содержит логгер log, что позволяет фиксировать действия, происходящие при взаимодействии с элементами в дочерних классах. 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 позволяет избежать ошибок, связанных с попыткой взаимодействия с элементами, которые ещё не отображены или уже скрыты.

BasePage.java

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*() 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 только записывает отладочную информацию, но может быть переопределён в наследниках.

BaseTest.java

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.

CommentComponent.java

Класс *CommentComponent* представляет собой компоненту для работы с отдельным комментарием на веб-странице и расширяет функциональность базового класса *BaseComponent*, наследуя от него методы и логирование. Этот класс инкапсулирует структуру и поведение визуального элемента комментария, позволяя обращаться к его частям и выполнять с ним действия.

Компонент инициализируется через конструктор, в который передаётся текст комментария. На его основе выполняется поиск основного контейнера комментария с использованием XPath, после чего внутри него инициализируются ссылки на вложенные элементы: имя автора, текст комментария, а также кнопки Редактировать и Удалить. Это позволяет избежать повторного поиска элементов при каждом обращении к ним. В момент инициализации в лог записывается сообщение, содержащее указанный текст комментария, что удобно для отладки.

Методы *getCommentText()* и *getAuthor()* возвращают текст комментария и имя его автора соответственно, предварительно ожидая появления соответствующего элемента на странице. Метод *isAuthoredBy()* позволяет сравнить имя автора комментария с ожидаемым значением, а *containsText()* - проверить наличие конкретной подстроки в тексте комментария.

Для взаимодействия с кнопками действия реализованы методы *clickEdit()* и *clickDelete()*, которые логируют нажатия кнопки и обеспечивают предварительную проверку видимости элементов перед кликом. Методы *isEditable()* и *isDeletable()* возвращают булевы значения, указывающие, доступны ли кнопки редактирования и удаления в текущем состоянии интерфейса.

Метод *isVisible()* использует унаследованную логику из *BaseComponent* и возвращает true, если сам комментарий отображается на странице.

HeaderComponent.java

Класс *HeaderComponent* представляет собой UI-компонент, отвечающий за взаимодействие с заголовком веб-страницы, в частности с навигационными элементами GitHub-интерфейса. Он расширяет базовый класс *BaseComponent*, унаследовав логирование и методы взаимодействия с элементами, и инкапсулирует все ключевые элементы заголовка, включая меню пользователя, глобальный поиск и кнопки профиля и выхода из аккаунта.

Компонент инициализирует множество элементов, используя XPath и CSS-селекторы с учетом возможной изменчивости интерфейса GitHub. Так, например, элемент *userMenu* описан с несколькими альтернативными локаторами. То же самое касается *signOutButton* и *profileLink*.

В конструкторе класса выводится лог о создании компонента. Метод search() реализует ввод поискового запроса в глобальное поле поиска в заголовке и отправку его нажатием клавиши Enter. Метод openUserMenu() позволяет открыть выпадающее меню пользователя, проверяя наличие доступных элементов по приоритету: *userMenu, userAvatar* или *dropdownCaret*.

Метод *clickProfile()* обеспечивает переход в профиль пользователя, предварительно открывая меню. Аналогично, метод *signOut()* реализует выход из аккаунта и также включает проверку наличия нужной кнопки. Оба метода сопровождаются логированием и защитой от ошибок за счёт проверок *exists()* и *shouldBe(visible)*.

Метод *clickGitHubLogo()* выполняет переход на главную страницу GitHub по клику на логотип, что может использоваться для возврата на стартовую страницу из любого состояния.

Два булевых метода - *isUserLoggedIn()* и *isUserMenuOpen()* служат для проверки текущего состояния интерфейса. Первый определяет, авторизован ли пользователь, по наличию одного из признаков интерфейса авторизации. Второй определяет, открыто ли выпадающее меню пользователя, что важно для пошагового управления тестовым сценарием.

IssueDetailsPage.java

Класс *IssueDetailsPage* - страница деталей issue (обращения, задачи) в веб-интерфейсе GitHub, она расширяет базовый класс *BasePage*. Он инкапсулирует все элементы, действия и проверки, связанные с отображением, управлением и взаимодействием с конкретным issue. Его цель — предоставить полный и удобный интерфейс для автотестов.

В классе определены ключевые элементы страницы: заголовок *issueTitle* и *issueTitleSticky*, описание *issueDescription*, статус *statusBadge* и кнопки управления: редактирование, закрытие, повторное открытие, закрепление, блокировка обсуждения, а также поля для добавления комментария. Используются CSS и XPath-селекторы из-за вариативности интерфейса GitHub.

В конструкторе происходит вызов *waitForPageLoad()* для ожидания появления статуса и заголовка для гарантии что дальнейшие действия выполняются на полностью загруженной странице.

Методы *getTitle()*, *getDescription()* и *getStatus()* возвращают основные данные issue. Методы *isOpen()* и *isClosed()* позволяют определить текущее состояние issue, проверяя наличие строк "Open" или "Closed" в статусе.

Методы *clickEdit()* и *editTitle()* позволяют выполнить редактирование заголовка issue: первый открывает режим редактирования, учитывая возможность доступа к кнопке через меню, а второй вводит новый заголовок и подтверждает изменение нажатием Enter.

Методы *clickClose()* и *clickReopen()* реализуют логику закрытия и повторного открытия issue, включая обработку сценариев с кнопками подтверждения или вложенными меню. Аналогично работают методы *clickPin()* и *clickLockConversation()*, отвечающие за закрепление и блокировку обсуждения.

Для взаимодействия с комментариями реализован метод *addComment()*, который вводит текст в поле и нажимает кнопку отправки. Метод *getCommentByText()* возвращает экземпляр *CommentComponent* - это позволяет

производить действия над конкретным комментарием, опираясь на его содержимое.

Метод *getIssueActions()* возвращает компонент *IssueActionsComponent*, если требуется доступ к вспомогательным действиям issue.

Также реализованы методы проверки: *isConversationLocked()* определяет, заблокировано ли обсуждение, *isCloseButtonDisabled()* — активна ли кнопка закрытия, *getNoPermissionTooltip()* — возвращает подсказку о недостатке прав.

CommentsTest.java

Класс CommentsTest предназначен для автоматизированной проверки функционала комментариев на страницах issues.

В начале класса определены три константы, загружаемые через *ConfigReader* из конфигурационного файла = это заголовок issue, её описание и текст комментария.

В классе реализовано два тестовых метода. Первый, *testAddCommentToIssue*, моделирует добавление комментария к новой issue. В ходе теста происходит авторизация пользователя, переход к тестовому репозиторию и вкладке Issues, создание новой issue путём заполнения заголовка и описания, и её публикация. Затем на открывшейся странице issue добавляется комментарий с помощью метода *addComment*.

Второй тест, testLockConversation, проверяет возможность блокировки обсуждения. Процедура авторизации и создания новой issue аналогична предыдущему тесту. После создания issue вызывается метод *clickLockConversation*, который активирует блокировку комментариев на уровне интерфейса. Далее с помощью *assertTrue* и метода *isConversationLocked* выполняется проверка, что обсуждение действительно было заблокировано.

IssueAssignmentTest

Класс IssueAssignmentTest - набор тестов, направленных на проверку функционала назначения пользователей и меток к задачам (issues) в репозитории.

В начале класса определены тестовые данные: заголовок и описание issue, имя пользователя для назначения, а также имена существующей и несуществующей метки.

Тест *testAssignUserToIssue* проверяет функционал назначения текущего пользователя на новую issue. После стандартной процедуры авторизации и создания issue осуществляется переход к блоку действий *issueActions*, из которого вызывается метод *assignYourself*. В тесте проверяется, что секция назначения Assignees видна и что пользователь успешно назначен.

Тест *testAddLabelToIssue* отвечает за проверку возможности присвоения метки задаче. После создания issue вызывается метод *selectLabel*, которому передаётся имя метки, и далее проверяется, что метка действительно появилась в интерфейсе задачи.

Тест testSearchNonExistentLabel фокусируется на поведении интерфейса при вводе несуществующего названия метки. После создания новой issue и открытия секции меток вызывается метод *searchLabel*, в который передаётся имя заведомо несуществующей метки. Далее с помощью методов *isCreateLabelSuggestionVisible* и *hasNoSearchResults* проверяется, что либо отображается предложение создать новую метку, либо интерфейс сообщает об отсутствии результатов.

IssueCreationTest.java

В данном тестовом классе IssueCreationTest реализованы три теста, направленных на проверку функционала создания и редактирования issues в интерфейсе репозитория.

Первый тест *testCreateIssueWithEmptyTitle* проверяет корректную обработку случая, когда пользователь пытается создать issue без заполнения

заголовка. После авторизации и перехода в репозиторий, открывается страница создания новой issue, где заполняется только описание, а заголовок остаётся пустым. Затем вызывается метод *clickCreateWithoutTitle*, и проверяется наличие ошибки валидации с помощью *assertTrue*.

Второй тест *testCreateValidIssue* проверяет успешное создание issue. Пользователь авторизуется, открывает репозиторий и переходит на вкладку Issues. На странице создания issue заполняются оба поля: заголовок *ISSUE_TITLE* и описание *ISSUE_DESCRIPTION*, затем происходит отправка формы. После этого с помощью методов *getTitle* и *isOpen* проверяется, что issue действительно создана с нужным заголовком и находится в статусе открытой. Далее выполняется переход обратно в список issues через главный экран, где с помощью *isIssueExists* проверяется наличие созданной issue в общем списке.

Третий тест testEditIssueTitle проверяет возможность редактирования заголовка уже созданной issue. После стандартной последовательности авторизации и создания issue, вызывается метод *editTitle* с новым значением *UPDATED_TITLE*. Затем с помощью *getTitle* проверяется, что заголовок был успешно обновлён.

IssueManagementTest.java

В классе IssueManagementTest представлены тесты, направленные на проверку функциональности управления issue в пользовательском интерфейсе: их закрытие, повторное открытие и закрепление.

Первый тест, *testCloseIssue*, проверяет возможность закрытия открытой задачи. После создания новой issue выполняется проверка, что задача находится в статусе "открыта". Затем вызывается метод *clickClose*, который инициирует закрытие задачи. Логгируется статус до и после операции, и выполняется проверка с помощью assertTrue, что issue действительно перешла в состояние *isClosed*.

Второй тест, *testReopenClosedIssue*, валидирует возможность переоткрытия ранее закрытой задачи. Сценарий начинается с создания новой

issue и её закрытия через *clickClose*, после чего проверяется, что статус изменён на закрытый. Затем вызывается метод *clickReopen*, и с помощью *assertTrue* проверяется, что issue снова находится в статусе "открыт".

Третий тест, *testPinIssue*, проверяет функциональность закрепления задачи. После стандартного процесса создания issue, вызывается метод *clickPin*, который закрепляет задачу, делая её видимой в верхней части списка issues.

PermissionsTest.java

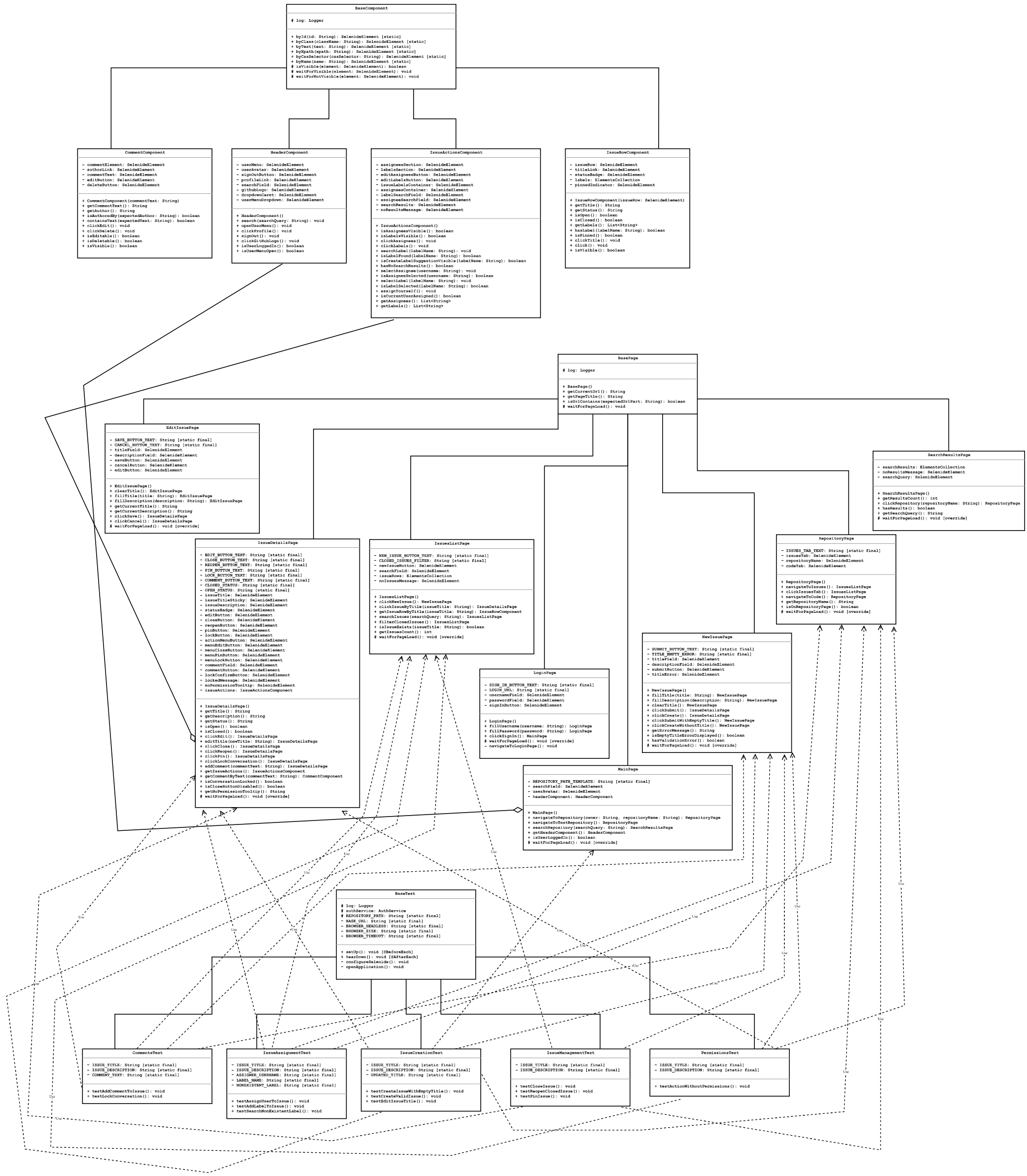
Класс *PermissionsTest* содержит тест, направленный на проверку ограничений прав доступа пользователя при взаимодействии с issues.

Тест *testActionWithoutPermissions* начинается с авторизации пользователя, перехода в тестовый репозиторий и создания новой issue с заданным заголовком и описанием, полученными из конфигурационного файла. После создания issue проверяется, что она находится в открытом состоянии *isOpen* и что заголовок совпадает с ожидаемым *getTitle*.

Основная часть теста посвящена проверке доступности определённых действий, связанных с управлением задачами. В блоке try-catch осуществляется попытка получить список доступных действий через метод *getIssueActions*. Если у текущего пользователя нет соответствующих прав (если он не является владельцем репозитория или имеет ограниченные разрешения), система выбрасывает исключение, которое перехватывается, и в лог записывается информация об ограничении прав доступа. В противном случае логируется сообщение о наличии прав на выполнение действий.

Тест имитирует ситуацию, при которой пользователь может столкнуться с ограничениями при работе с задачами, и проверяет корректную обработку таких случаев в пользовательском интерфейсе и логике приложения.

2.2 UML диаграмма


BaseComponent
log: Logger
+ byId(id: String): SelenideElement [static]
+ byClass(className: String): SelenideElement [static]
+ byText(text: String): SelenideElement [static]
+ byXpath(xpath: String): SelenideElement [static]
+ byCssSelector(cssSelector: String): SelenideElement [static]
+ byName(name: String): SelenideElement [static]
# isVisible(element: SelenideElement): boolean
# waitForVisible(element: SelenideElement): void
# waitForNotVisible(element: SelenideElement): void
CommentComponent
- commentElement: SelenideElement
- authorLink: SelenideElement
- commentText: SelenideElement
- editButton: SelenideElement
- deleteButton: SelenideElement
+ CommentComponent(commentText: String)
+ getCommentText(): String
+ getAuthor(): String
+ isAuthoredBy(expectedAuthor: String): boolean
+ containsText(expectedText: String): boolean
+ clickEdit(): void
+ clickDelete(): void
+ isEditable(): boolean
+ isDeletable(): boolean
+ isVisible(): boolean
HeaderComponent
- userMenu: SelenideElement
- userAvatar: SelenideElement
- signOutButton: SelenideElement
- profileLink: SelenideElement
- searchField: SelenideElement
- githubLogo: SelenideElement
- dropdownCaret: SelenideElement
- userMenuDropdown: SelenideElement
+ HeaderComponent()
+ search(searchQuery: String): void
+ openUserMenu(): void
+ clickProfile(): void
+ signOut(): void
+ clickGitHubLogo(): void
+ isUserLoggedIn(): boolean
+ isUserMenuOpen(): boolean
IssueActionsComponent
- assigneesSection: SelenideElement
- labelsSection: SelenideElement
- editAssigneesButton: SelenideElement
- editLabelsButton: SelenideElement
- issueLabelsContainer: SelenideElement
- assigneesContainer: SelenideElement
- labelSearchField: SelenideElement
- assigneeSearchField: SelenideElement
- searchResults: SelenideElement
- noResultsMessage: SelenideElement
+ IssueActionsComponent()
+ isAssigneesVisible(): boolean
+ isLabelsVisible(): boolean
+ clickAssignees(): void
+ clickLabels(): void
+ searchLabel(labelName: String): void
+ isLabelFound(labelName: String): boolean
+ isCreateLabelSuggestionVisible(labelName: String): boolean
+ hasNoSearchResults(): boolean
+ selectAssignee(username: String): void
+ isAssigneeSelected(username: String): boolean
+ selectLabel(labelName: String): void
+ isLabelSelected(labelName: String): boolean
+ assignYourself(): void
+ isCurrentUserAssigned(): boolean
+ getAssignees(): List<String>
+ getLabels(): List<String>
IssueRowComponent
- issueRow: SelenideElement
- titleLink: SelenideElement
- statusBadge: SelenideElement
- labels: ElementsCollection
- pinnedIndicator: SelenideElement
+ IssueRowComponent(issueRow: SelenideElement)
+ getTitle(): String
+ getStatus(): String
+ isOpen(): boolean
+ isClosed(): boolean
+ getLabels(): List<String>
+ hasLabel(labelName: String): boolean
+ isPinned(): boolean
+ clickTitle(): void
+ click(): void
+ isVisible(): boolean
BasePage
log: Logger
+ BasePage()
+ getCurrentUrl(): String
+ getPageTitle(): String
+ isUrlContains(expectedUrlPart: String): boolean
# waitForPageLoad(): void
EditIssuePage
- SAVE_BUTTON_TEXT: String [static final]
- CANCEL_BUTTON_TEXT: String [static final]
- titleField: SelenideElement
- descriptionField: SelenideElement
- saveButton: SelenideElement
- cancelButton: SelenideElement
- editButton: SelenideElement
+ EditIssuePage()
+ clearTitle(): EditIssuePage
+ fillTitle(title: String): EditIssuePage
+ fillDescription(description: String): EditIssuePage
+ getCurrentTitle(): String
+ getCurrentDescription(): String
+ clickSave(): IssueDetailsPage
+ clickCancel(): IssueDetailsPage
# waitForPageLoad(): void [override]
SearchResultsPage
- searchResults: ElementsCollection
- noResultsMessage: SelenideElement
- searchQuery: SelenideElement
+ SearchResultsPage()
+ getResultsCount(): int
+ clickRepository(repositoryName: String): RepositoryPage
+ hasResults(): boolean
+ getSearchQuery(): String
# waitForPageLoad(): void [override]
IssueDetailsPage
- EDIT_BUTTON_TEXT: String [static final]
- CLOSE_BUTTON_TEXT: String [static final]
- REOPEN_BUTTON_TEXT: String [static final]
- PIN_BUTTON_TEXT: String [static final]
- LOCK_BUTTON_TEXT: String [static final]
- COMMENT_BUTTON_TEXT: String [static final]
- CLOSED_STATUS: String [static final]
- OPEN_STATUS: String [static final]
- issueTitle: SelenideElement
- issueTitleSticky: SelenideElement
- issueDescription: SelenideElement
- statusBadge: SelenideElement
- editButton: SelenideElement
- closeButton: SelenideElement
- reopenButton: SelenideElement
- pinButton: SelenideElement
- lockButton: SelenideElement
- actionMenuButton: SelenideElement
- menuEditButton: SelenideElement
- menuCloseButton: SelenideElement
- menuPinButton: SelenideElement
- menuLockButton: SelenideElement
- commentField: SelenideElement
- commentButton: SelenideElement
- lockConfirmButton: SelenideElement
- lockedMessage: SelenideElement
- noPermissionTooltip: SelenideElement
- issueActions: IssueActionsComponent
+ IssueDetailsPage()
+ getTitle(): String
+ getDescription(): String
+ getStatus(): String
+ isOpen(): boolean
+ isClosed(): boolean
+ clickEdit(): IssueDetailsPage
+ editTitle(newTitle: String): IssueDetailsPage
+ clickClose(): IssueDetailsPage
+ clickReopen(): IssueDetailsPage
+ clickPin(): IssueDetailsPage
+ clickLockConversation(): IssueDetailsPage
+ addComment(commentText: String): IssueDetailsPage
+ getIssueActions(): IssueActionsComponent
+ getCommentByText(commentText: String): CommentComponent
+ isConversationLocked(): boolean
+ isCloseButtonDisabled(): boolean
+ getNoPermissionTooltip(): String
# waitForPageLoad(): void [override]
IssuesListPage
- NEW_ISSUE_BUTTON_TEXT: String [static final]
- CLOSED_ISSUES_FILTER: String [static final]
- newIssueButton: SelenideElement
- searchField: SelenideElement
- issueRows: ElementsCollection
- noIssuesMessage: SelenideElement
+ IssuesListPage()
+ clickNewIssue(): NewIssuePage
+ clickIssueByTitle(issueTitle: String): IssueDetailsPage
+ getIssueRowByTitle(issueTitle: String): IssueRowComponent
+ searchIssues(searchQuery: String): IssuesListPage
+ filterClosedIssues(): IssuesListPage
+ isIssueExists(issueTitle: String): boolean
+ getIssuesCount(): int
# waitForPageLoad(): void [override]
RepositoryPage
- ISSUES_TAB_TEXT: String [static final]
- issuesTab: SelenideElement
- repositoryName: SelenideElement
- codeTab: SelenideElement
+ RepositoryPage()
+ navigateToIssues(): IssuesListPage
+ clickIssuesTab(): IssuesListPage
+ navigateToCode(): RepositoryPage
+ getRepositoryName(): String
+ isOnRepositoryPage(): boolean
# waitForPageLoad(): void [override]
LoginPage
- SIGN_IN_BUTTON_TEXT: String [static final]
- LOGIN_URL: String [static final]
- usernameField: SelenideElement
- passwordField: SelenideElement
- signInButton: SelenideElement
+ LoginPage()
+ fillUsername(username: String): LoginPage
+ fillPassword(password: String): LoginPage
+ clickSignIn(): MainPage
# waitForPageLoad(): void [override]
- navigateToLoginPage(): void
NewIssuePage
- SUBMIT_BUTTON_TEXT: String [static final]
- TITLE_EMPTY_ERROR: String [static final]
- titleField: SelenideElement
- descriptionField: SelenideElement
- submitButton: SelenideElement
- titleError: SelenideElement
+ NewIssuePage()
+ fillTitle(title: String): NewIssuePage
+ fillDescription(description: String): NewIssuePage
+ clearTitle(): NewIssuePage
+ clickSubmit(): IssueDetailsPage
+ clickCreate(): IssueDetailsPage
+ clickSubmitWithEmptyTitle(): NewIssuePage
+ clickCreateWithoutTitle(): NewIssuePage
+ getErrorMessage(): String
+ isEmptyTitleErrorDisplayed(): boolean
+ hasValidationError(): boolean
# waitForPageLoad(): void [override]
MainPage
- REPOSITORY_PATH_TEMPLATE: String [static final]
- searchField: SelenideElement
- userAvatar: SelenideElement
- headerComponent: HeaderComponent
+ MainPage()
+ navigateToRepository(owner: String, repositoryName: String): RepositoryPage
+ navigateToTestRepository(): RepositoryPage
+ searchRepository(searchQuery: String): SearchResultsPage
+ getHeaderComponent(): HeaderComponent
+ isUserLoggedIn(): boolean
# waitForPageLoad(): void [override]
BaseTest
log: Logger
authService: AuthService
REPOSITORY_PATH: String [static final]
- BASE_URL: String [static final]
- BROWSER_HEADLESS: String [static final]
- BROWSER_SIZE: String [static final]
- BROWSER_TIMEOUT: String [static final]
+ setUp(): void [@BeforeEach]
+ tearDown(): void [@AfterEach]
- configureSelenide(): void
- openApplication(): void
CommentsTest
- ISSUE_TITLE: String [static final]
- ISSUE_DESCRIPTION: String [static final]
- COMMENT_TEXT: String [static final]
+ testAddCommentToIssue(): void
+ testLockConversation(): void
IssueAssignmentTest
- ISSUE_TITLE: String [static final]
- ISSUE_DESCRIPTION: String [static final]
- ASSIGNEE_USERNAME: String [static final]
- LABEL_NAME: String [static final]
- NONEXISTENT_LABEL: String [static final]
+ testAssignUserToIssue(): void
+ testAddLabelToIssue(): void
+ testSearchNonExistentLabel(): void
IssueCreationTest
- ISSUE_TITLE: String [static final]
- ISSUE_DESCRIPTION: String [static final]
- UPDATED_TITLE: String [static final]
+ testCreateIssueWithEmptyTitle(): void
+ testCreateValidIssue(): void
+ testEditIssueTitle(): void
IssueManagementTest
- ISSUE_TITLE: String [static final]
- ISSUE_DESCRIPTION: String [static final]
+ testCloseIssue(): void
+ testReopenClosedIssue(): void
+ testPinIssue(): void
PermissionsTest
- ISSUE_TITLE: String [static final]
- ISSUE_DESCRIPTION: String [static final]
+ testActionWithoutPermissions(): void
Use

3. ТЕСТИРОВАНИЕ

3.1. Демонстрация отработки тестов

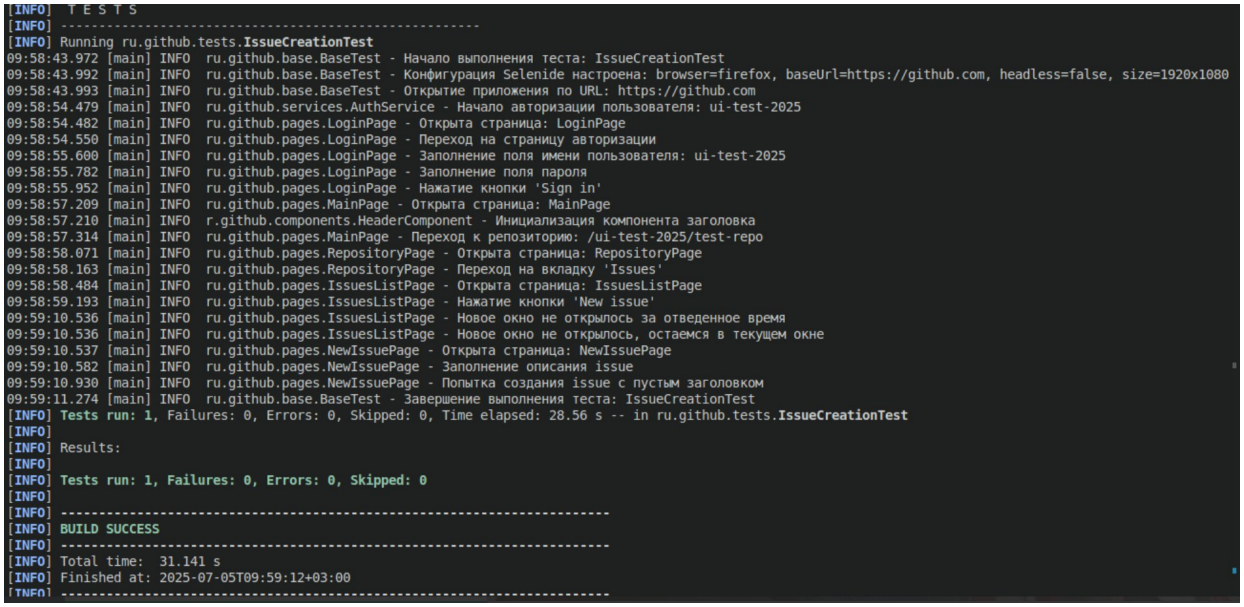


Рис. 1 выполнение теста IssueCreationTest

На картинке видим, что программа успешно открыла сайт GitHub.com, перешла во вкладку issues, нажала на кнопку создать новое задание (New issue) и попробовала создать задание с пустым заголовком, возникла ошибка создания, что было ожидаемым результатом и не вызывает ошибки.

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

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

Репозиторий с кодом // URL: https://github.com/postusername/ui-test-2025/tree/main