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

ОТЧЕТ

по учебной практике

Тема: UI-тестирование сайта Pinterest

Студенты гр. 3344 Бубякина Ю.В.
Вердин К.К.
Жаворонок Д.Н.

Руководитель Шевелева А.М

Санкт-Петербург
2025

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студенты Бубякина Ю.В., Вердин К.К., Жаворонок Д.Н.

Группа 3344

Тема практики: UI-тестирование сайта Pinterest

Задание на практику:

Создание архитектуры для реализации 10 различных UI-Тестов функционала сайта Pinterest. С использованием таких технологий как Java, Selenide (Selenium), Junit, Maven, логирование. Создание документации к этим тестам

Сроки прохождения практики: 25.06.2025 – 25.06.2025

Дата сдачи отчета: 07.04.2025

Дата защиты отчета: 07.04.2025

Студенты		Бубякина Ю.В. Вердин К.К. Жаворонок Д.Н.
Руководитель		Шевелева А.М

АННОТАЦИЯ

Целью данной практики является освоение автоматизации тестирования веб-приложений с использованием современных технологий: Java, Selenide (на основе Selenium), JUnit для написания тестов и Maven как системы управления проектом. В рамках практики разработано 10 автотестов на определенный функциональный блок выбранной системы (например, работа с постами в социальной сети). Работа ведётся в группах по 3 человека через GitHub, где каждый участник выполняет коммиты и несёт ответственность за определённую часть проекта. Тесты должны быть задокументированы в формате чеклиста с описанием действий, входных данных и ожидаемых результатов. Также предусмотрено логирование выполнения тестов. Итоговый отчет включает описание реализации, UML-диаграммы, демонстрацию работы тестов и заключение по результатам практики

SUMMARY

The goal of this practice is to master the automation of web application testing using modern technologies: Java, Selenide (based on Selenium), JUnit for writing tests, and Maven as a project management system. Within the scope of the practice, 10 automated tests have been developed for a specific functional block of a selected system (for example, working with posts in a social network). The work is carried out in groups of 3 people via GitHub, where each participant makes commits and is responsible for a certain part of the project. The tests must be documented in a checklist format, including descriptions of actions, input data, and expected results. Logging of test execution is also provided. The final report includes an implementation overview, UML diagrams, demonstration of test execution, and a conclusion based on the results of the practice.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: Разработка автоматизированных UI-тестов для веб-приложения с использованием современных инструментов тестирования.

Задачи:

1.Выбрать систему тестирования

2.Оформить чеклист в виде таблицы с описанием тестов.

3.Организовать работу в GitHub (репозиторий на группу из 3 человек с распределением задач в README.md).

4.Написать 10 автоматизированных тестов и документацию к ним для выбранной системы.

Использовать технологии: Java, Selenide (Selenium), JUnit 5, Maven, логирование.

Для тестирования был выбрал сайт Pinterest [1]. Выбор обусловлен тем, что это популярные и функционально насыщенный социальный интернет-сервис, фотохостинг, позволяющий пользователям добавлять в режиме онлайн изображения, помещать их в тематические коллекции и делиться ими с другими пользователями.

1. РЕАЛИЗУЕМЫЕ ТЕСТЫ

1.1. Описание тестов

1. **Проверка авторизации**

Проверяется поведение формы входа при различных вариантах ввода. Тест включает открытие модального окна, попытки входа с неверными данными (неверный пароль, некорректный формат email) и успешную авторизацию. Ожидаемые результаты — отображение соответствующих сообщений об ошибке и переход на главную страницу при корректных данных.

2. **Редактирование профиля**

Проверяется возможность изменения пользовательской информации в настройках профиля. Вводятся новые значения для имени, фамилии и описания, сохраняются, после чего выполняется перезагрузка страницы. Ожидается, что обновлённые данные корректно сохраняются и отображаются.

3. **Открытие и закрытие поста**

Тест проверяет корректность открытия поста из ленты и его закрытия. После клика по посту URL должен соответствовать ссылке элемента. При возврате назад пользователь должен видеть главную страницу без ошибок или перезагрузки.

4. **Поставить и убрать лайк на пост**

Проверяется возможность поставить и убрать лайк у поста. После нажатия на кнопку лайка и перезагрузки страницы меняется SVG-атрибут d элемента path, отражающий состояние кнопки. При повторном нажатии и обновлении страницы атрибут возвращается к исходному значению, что подтверждает отмену лайка.

5. **Сохранение поста в профиль**

Тест проверяет работу кнопки сохранения поста. После нажатия на кнопку её атрибут aria-label должен измениться, отражая новое состояние “Пост сохранен”.

6. **Добавление комментария**

Проверяется возможность оставить комментарий к посту с текстом и прикреплённым изображением. После отправки комментарий должен отображаться под постом с введённым текстом и загруженным изображением.

7. **Скачивание изображения**

Проверяется работа функции скачивания. После нажатия на кнопку «…» и выбора опции «Скачать пост» изображение должно успешно сохраняться на устройство.

8. **Поиск по запросу**

Проверяется корректность работы поиска. При вводе запроса и подтверждении поиска, среди первых 10 изображений хотя бы одно должно содержать в атрибуте aria-label ключевое слово из запроса.

9. **Скрытие поста из ленты**

Тест проверяет возможность скрыть пост по причине «Не интересует». После выбора этой опции появляется уведомление «Пин скрыт», подтверждающее успешное выполнение действия.

10. **Изменение сортировки досок**

Проверяется корректность работы функции сортировки досок в разделе «Сохранённые» профиля пользователя. Тест направлен на подтверждение того, что при выборе в выпадающем меню сортировки критерия «Последний добавленный пин» отображение досок обновляется и соответствует выбранному параметру сортировки.

Чеклист

№	Название теста	Шаги тестирования	Ожидаемый результат

1.	Проверка авторизации	1. Нажать на кнопку войти в хедере, чтобы появилась модалка Логина 2. Ввести логин и неверный пароль «aboba» 3.Нажать на кнопку войти в модалке 4. Удалить из инпутов email и password значения, ввести некорректный email и пароль 5. Нажать на кнопку войти 6. Удалить из инпутов email и password значения, ввести корректные email и пароль 7. Нажать на кнопку войти	При корректных данных — пользователь авторизован, окно авторизации закрыто, отображаются посты. При некорректном пароле должно появиться сообщение «Неверный пароль» При неправильном формате логина должно появиться сообщение «Похоже формат email неверен»
2.	Редактирование профиля	1.Перейти на главную страницу 2. Нажать на кнопку настройки (кнопка с иконкой шестерни слева снизу) 3.Нажать на кнопку с надписью «Настройки» 4.Перейти в раздел настроек профиля 5.В инпуты вводятся изначальные имя, фамилия и о себе, с добавленными к себе суффиксами	Профиль обновлён: новое имя фамилия и описание отображаются.

		6.Нажать на кнопку сохранить 7. Перезагрузить страницу	
3.	Открытие и закрытие поста	1.Перейти на главную страницу 2.Нажать на первый пост в ленте (самый верхний левый) 3.Нажать на кнопку назад, закрыв пост	После нажатия на пост url в поиске должен соответствовать href в <a> После нажатия на кнопку назад, главная страница должна отображаться
4.	Поставить и убрать лайк на пост	1.Перейти на главную страницу 2.Нажать на первый пост в ленте (самый верхний левый) 3.Нажать на кнопку Лайка 4. Перезагрузить страницу 5. Снова нажать на кнопку лайка 6. Перезагрузить страницу	После нажатия на кнопку лайка и перезагрузки страница аттрибуты d элемента path, который является дочерним элементом кнопки должны быть разные. При повторных нажатии и перегрузке атрибут d <path> должен совпасть с первоначальным

5.	Сохранение поста в профиль	1. Перейти на главную страницу 2.Нажать на первый пост в ленте (самый верхний левый) 3.Нажать на Сохранить	После нажатия на кнопку его aria-label должен измениться
6.	Добавление комментария	1.Перейти на главную страницу 2.Нажать на первый пост в ленте (самый верхний левый) 3.Ввести в инпут «Классный пост» 4.Нажать на кнопку «Прикрепить фото» 5.Загрузить фото test.png 6.Нажать на кнопку «Отправить»	Комментарий отображается под постом с текстом и изображением
7.	Скачивание изображения	1.Перейти на главную страницу 2.Нажать на первый пост в ленте (самый верхний левый) 3.Нажать на кнопку «…» 4.Нажать на кнопку «Скачать пост»	Изображение успешно скачано на устройство
8.	Поиск по запросу	1.Перейти на главную страницу 2.Нажать на поисковую строку сверху страницы	Хотя бы в 1 из 10 <img> будет атрибут aria-label,

		3. Ввести в неё «милые котики» 4.Нажать enter	содержащий один из ключей проверки
9.	Скрытие поста из ленты	1.Перейти на главную страницу 2.Нажать на первый пост в ленте (самый верхний левый) 3.Нажать на кнопку «…» 4. Выбрать причину: 'Не интересует'	Появляется уведомление 'Пин скрыт'
10.	Изменение сортировки досок	1. Перейти в профиль → 'Сохранённые' 2. Открыть выпадающее меню сортировки 3. Выбрать 'Последний добавленный пин'	Сортировка досок изменилась согласно выбранному критерию

2. ОПИСАНИЕ КЛАССОВ И МЕТОДОВ, UML ДИАГРАММА

2.1. Описание классов и методов

Для данной работы использовался паттерн Page Object Modal, для реализации которого были созданы классы элементов, страниц и тестов.

Классы элементов

public class BaseElement - Базовый класс для всех элементов пользовательского интерфейса. Инкапсулирует общую функциональность для работы с веб-элементами.

Поля:

protected static final int WAIT_SECONDS - константа, определяющая время ожидания рендера элемента.

protected final SelenideElement baseElement - экземпляр найденного элемента

Методы:

protected BaseElement(String xpath, String attributeValue) - конструктор базового элемента, принимает следующие аргументы: *xPath* - шаблон xPath для поиска элемента *attributeValue* - параметр для подстановки в шаблон Xpath

public boolean isDisplayed() - метод, проверяющий, отображается ли элемент на странице. Ожидает появление элемента в течение стандартного времени ожидания.

public String getAttribute(String attributeName) - метод, возвращающий значение указанного атрибута (*attributeName*) элемента

public class Article extends BaseElement - класс, представляющий элемент статьи на веб-странице, наследуется от базового класса элемента BaseElement. Инкапсулирует функциональность для работы со статьями.

Поля:

private static final String ARIA_LABEL_XPATH - константа. определяющая xPath к элементу по значению атрибута aria-label.

Методы:

public String getHref() - получает значение атрибута href элемента статьи

public String getAriaLabel() - Получает значение атрибута aria-label элемента статьи.

public void click() - выполняет клик по элементу статьи

Статичные фабрики:

public static Article byAriaLabel(String ariaLabel) - Находит элемент статьи по значению атрибута *aria-label*.

public static Article byXpath(String xPath) - Находит элемент статьи по *XPath*.

public class Button extends BaseElement - Класс для работы с элементами типа "кнопка". Предоставляет методы для поиска кнопок по различным атрибутам и выполнения действий над ними. Наследуется от базового класса элемента BaseElement.

Поля:

private static final String ID_XPATH - определяет xPath к элементу кнопки по атрибуту id

private static final String TEXT_XPATH - определяет xPath к элементу кнопки по атрибуту text

private static final String CLASS_XPATH - определяет xPath к элементу кнопки по атрибуту class

private static final String TYPE_XPATH - определяет xPath к элементу кнопки по атрибуту type

private static final String ARIA_LABEL_XPATH - определяет xPath к элементу кнопки по атрибуту aria-label

private static final String DATA_TEST_ID_XPATH - определяет xPath к элементу кнопки по атрибуту data-test-id

Методы:

public void click() - выполняет клик по кнопке.

Фабричные статичные методы:

public static Button byId(String id) - возвращает кнопку по идентификатору (*id*)

public static Button byText(String text) - находит кнопку по тексту внутри неё (*text*)

public static Button byClass(String className) - находит кнопку по названию класса (className)

public static Button byType(String type) - находит кнопку по её типу (type)

public static Button byAriaLabel(String ariaLabel) - Находит кнопку по значению атрибута aria-label.

public static Button byXPath(String xPath) - Находит кнопку по указанному XPath.

public static Button byDataTestId(String dataTestId) - находит кнопку по значению атрибута data-test-id.

public class H2 extends BaseElement - Класс для работы с текстовыми элементами. Предоставляет методы для поиска текстовых элементов и проверки их содержимого.

Поля:

private static final String ID_XPATH - определяет xPath к элементу h2 по значению атрибута id.

private static final String CLASS_XPATH = определяет xPath к элементу h2 по значению атрибута class.

Методы:

public String getText() - возвращает текст элемента.

Фабричные статичные методы:

public static H2 byId(String id) - Находит текстовый элемент по идентификатору.

public static H2 byClass(String className) - Находит текстовый элемент по названию класса.

public static H2 byXPath(String xpath) - Находит H2 по указанному XPath.

public class Image extends BaseElement - Класс для работы с элементами типа "изображение". Предоставляет методы для поиска изображений по различным атрибутам и выполнения действий над ними.

Поля:

private static final String ALT_XPATH

private static final String ELEMENT_TIMING_XPATH

private static final String HREF_ATTRIBUTE

private static final String SRC_ATTRIBUTE

являются константами для шаблонов xPath

Методы:

public void click() - Выполняет клик по изображению.

Фабричные статические методы:

public static Image byAlt(String alt) - Находит изображение по значению атрибута alt.

public class ImagePreview extends BaseElement - класс для работы с превью изображений. Представляет составной элемент, содержащий связанную статью и заголовок.

Поля:

private static final String ARTICLE_XPATH_SUFFIX

private static final String H2_XPATH_SUFFIX

являются константами для xPath суффиксов

private final Article article - элемент статьи;

private final H2 h2 элемент заголовка;

Методы:

public String getPreviewName() - получает заголовок превью изображения

public String getArticleAriaLabel() - Получает значение атрибута aria-label связанной статьи

public String getArticleHref() - Получает ссылку на связанную статью.

public void click() - Выполняет переход по ссылке превью изображения.

Фабричные статические методы:

p*ublic static ImagePreview byXpath(String xPath)* - Находит превью изображения по указанному *XPath*.

public class Input extends BaseElement - Класс для работы с элементами типа "поле ввода". Предоставляет методы для поиска полей ввода по различным атрибутам и выполнения действий над ними.

Поля:

private static final String ID_XPATH

private static final String NAME_XPATH

private static final String CLASS_XPATH

private static final String DATA_TEST_ID_XPATH

являются константами для xPath суффиксов

Методы:

public void clear() - Очищает поле ввода.

public void fill(String value) - Вводит указанное значение в поле ввода.

public void load(File file) - метод, позволяющий загружать файлы

public void click() - Кликает на элемент.

public void pressEnter() - Нажимает Enter для подтверждения.

Фабричные статичные методы:

public static Input byId(String id) - Находит поле ввода по идентификатору id.

public static Input byDataTestId(String id) - Находит поле ввода по dataTextId.

public static Input byName(String name) - Находит поле ввода по имени name.

public class LikeButton extends Button - Класс для работы с кнопкой "лайк". Расширяет функциональность базовой кнопки, добавляя специфичные для кнопки методы.

Поля:

private static final String PATH_XPATH_SUFFIX

private static final String DATA_TEST_ID_XPATH

являются константами для xPath суффиксов

private final Path path - элемент типа path

Методы:

public String getPathD() - Получает значение атрибута 'd' элемента path внутри кнопки. Используется для проверки состояния лайка (заполненный/пустой).

Фабричные статические методы:

public static LikeButton byDataTestId(String dataTestId) - Находит кнопку лайка по значению атрибута data-test-id.

public class MansoryContainer extends BaseElement - Класс для работы с Masonry-контейнером изображений. Представляет контейнер, содержащий превью изображений в виде сетки.

Поля:

private static final String CLASS_XPATH

private static final String IMAGE_PREVIEW_XPATH

являются константами для xPath суффиксов

Методы:

public String getHrefOfFirstImage() - Получает ссылку на статью первого превью изображения в контейнере.

public ImagePreview getNthImagePreview(int n) - Получает превью изображения по порядковому номеру.

public void clickOnImage() - Выполняет клик по первому превью изображения в контейнере.

public String getPreviewName() - Получает название первого превью изображения в контейнере.

public String getNthArticleAriaLabel(int n) - Получает значение атрибута aria-label статьи для n-го превью.

Фабричные статические методы:

public static MansoryContainer byClass(String className) - Находит Masonry-контейнер по названию класса.

public class Path extends BaseElement - Класс для работы с элементами SVG-пути. Предоставляет методы для взаимодействия с тегом <path/> в SVG-графике.

Методы:

public String getD() - Получает значение атрибута 'd' элемента пути. Атрибут 'd' содержит команды для построения SVG-пути.

Фабричные статические методы:

public static Path byXpath(String xPath) - Находит элемент пути по указанному XPath.

public class SearchHeader extends BaseElement - Класс для работы с шапкой поиска. Инкапсулирует функциональность элементов поисковой строки и кнопок в шапке приложения.

Поля:

private static final String CLASS_XPATH - константа, определяющая xPath по названию класса

private final Input searchInput - объект класса Input.

private final Button accountButton - кнопка аккаунта.

Методы:

public void openSearchBar() - Активирует поле поиска, выполняя по нему клик.

public void fillSearchText(String text) - Вводит текст в поле поиска.

public void pressEnter() - Выполняет поиск, нажимая Enter в поле поиска.

public void clickAccountButton() - Нажимает на кнопку аккаунта

Фабричные статические методы:

public static SearchHeader byId(String text) - Находит шапку поиска по идентификатору.

public class Text extends BaseElement - Класс для работы с текстовыми элементами. Предоставляет методы для поиска текстовых элементов и проверки их содержимого.

Поля:

private static final String ID_XPATH

private static final String CLASS_XPATH

private static final String TEXT_XPATH

являются константами для xPath суффиксов

Фабричные методы:

public static Text byId(String id) - Находит текстовый элемент по идентификатору.

public class TextArea extends BaseElement - Класс для работы с текстовыми элементами. Предоставляет методы для поиска текстовых элементов и проверки их содержимого.

Поля:

private static final String ID_XPATH

являются константами для xPath суффиксов

Методы:

public String getText() - Возвращает текст элемента.

public void fill(String value) - вводит текст в элемент

Фабричные методы:

public static TextArea byId(String id) - Находит текстовый элемент по идентификатору.

Классы страниц

public class BasePage - Базовый класс для всех страниц приложения. Инкапсулирует общую функциональность работы со страницами.

Поля:

private static final String BASE_ELEMENT_XPATH - базовый xpath страницы

protected final SelenideElement basePage - элемент страницы

protected final Class<? extends BasePage> pageClass - класс страницы

Методы:

protected BasePage(Class<? extends BasePage> pageClass, String uniqueElementIdentifier) - Конструктор базовой страницы.

pageClass - класс страницы (для рефлексии)

uniqueElementIdentifier - уникальный идентификатор элемента страницы

public <T extends BasePage> T refresh(Class<T> pageClass) - Обновляет текущую страницу. *pageClass* - класс страницы, которую нужно вернуть, *<T>* тип страницы, возвращает новый экземпляр обновлённой страницы

public <T extends BasePage> T page(Class<T> pageClass) - Создаёт экземпляр страницы указанного класса. *pageClass* - класс страницы, *<T>* - тип страницы

public class BoardsPage extends BasePage - Страница досок пользователя. Предоставляет функционал для работы с досками и их сортировкой.

Поля:

private final Button moreOptionsButton - элемент кнопки “показать ещё”.

private final Button alphabeticOrderButton - элемент кнопки сортировки по алфавиту.

private final Button lastAddedOrderButton - элемент кнопки сортировки по дате добавления.

private final MansoryContainer mansoryContainer - элемент контейнера с картинками.

Методы:

public void openSortingOptions() - Открывает меню дополнительных опций сортировки.

public void sortAlphabetically() - Выбирает сортировку досок по алфавиту

public void sortByLastAdded() - Выбирает сортировку досок по дате добавления (последние добавленные).

public String getFirstBoardName() - Получает название первого превью доски в контейнере.

public class LoginPage extends BasePage - Страница аутентификации пользователя. Предоставляет методы для взаимодействия с элементами входа.

Поля:

private final Button loginModalButton - кнопка открытия модал окна логина.

*private final Input loginInput*t - поле ввода логина

private final Input passwordInput - поле ввода пароля

private final Button submitButton - кнопка входа в аккаунт

private final Text incorrectPasswordText - текст, появляющийся при неверном пароле

private final Text incorrectEmailFormatText - текст, появляющийся при неверном формате e-mail.

Методы:

public <T extends BasePage> T openLoginModal(Class<T> nextPageClass) - Открывает модальное окно аутентификации. *nextPageClass* - класс возращаемой страницы, *<T>* - тип страницы

public <T extends BasePage> T enterLogin(String login, Class<T> nextPageClass) - Вводит логин в поле ввода.

login логин пользователя

nextPageClass класс возращаемой страницы

<T> тип страницы

public <T extends BasePage> T enterPassword(String password, Class<T> nextPageClass) - Вводит пароль в поле ввода.

password - пароль пользователя

nextPageClass класс возращаемой страницы

<T> тип страницы

public <T extends BasePage> T clickLoginButton(Class<T> nextPageClass) - Нажимает кнопку входа.

nextPageClass - класс возращаемой страницы

<T> - тип страницы

public boolean checkIsIncorrectPasswordMessageDisplayed() - Проверяет отображение сообщения о неверном пароле.

public boolean checkIsIncorrectEmailMessageDisplayed() -Проверяет отображение сообщения о неверном формате email.

public class MainPage extends BasePage - главная страница, на которую пользователь переходит после аутентификации

Поля:

private final Button undoActionButton - кнопка отмены действия

private final MansoryContainer mansoryContainer - элемент mansory контейнера

private final SearchHeader searchHeader - элемент поискового хэдэра

private final Button settingsModalButton - элемент кнопки, открывающей настройки

Методы:

public boolean isDisplayed() - Проверяет отображение главной страницы.

public String getHrefOfFirstImage() - Получает ссылку на статью первого изображения.

public <T extends BasePage> T clickOnFirstImage(Class<T> className) - Открывает статью первого изображения. Возвращает экземпляр страницы className

public void openSearchBar() - Активирует поле поиска.

public void fillSearchBar(String searchText) - Вводит текст в поле поиска.

public void submitSearch() - Выполняет поиск.

public void waitForSearchPageLoad(String expectedUrlPart) - Ожидает загрузку страницы по частичному совпадению URL.

public <T extends BasePage> T clickAccountButton(Class<T> className) - Открывает страницу аккаунта.

public String getNthAriaLabel(int n) - Получает значение атрибута aria-label для n-го элемента.

public void clickSettingsModalButton() - Открывает меню настроек.

public <T extends BasePage> T clickSettings(Class<T> className) - Открывает страницу настроек.

public boolean undoDisplayed() - Проверяет, появилась ли возможность отменить действие скрытия

public class PinPage extends BasePage - страница, отображающая пин(пост)

Поля

private final Button backButton - кнопка перехода на главную страницу

private final Button moreOptionsButton - кнопка дополнительных опций

private final Button hidePinButton - кнопка скрытия пина

private final Button notInterestedButton - кнопка “не интересно”

private final LikeButton likeButton - кнопка лайка

private final Button downloadPinButton - кнопка скачивания фото пина

private final Button saveButton - кнопка сохранения пина к себе

private final Button alreadySavedButton - кнопка отмены сохранения пина

private final Button choosePhotoButton - кнопка выбора фото для комментария

private final Input loadPhotoInput - поле выбора фото для комментария

private final SelenideElement commentTextInput - поле для ввода текста комментария

private final Button sendCommentButton - кнопка отправки комментария

private final H2 commentCount - поле, показывающее количество комментариев

public String getCurrentUrl() - Получает ссылку на эту страницу - изображение пина.

public <T extends BasePage> T clickBackButton(Class<T> className) - Возвращается на предыдущую страницу.

public void waitForPinPageLoad(String expectedUrlPart) - Ожидает загрузку страницы пина по частичному совпадению URL

public void clickMoreOptionsButton() - Открывает меню дополнительных опций.

public void clickHidePinButton() - Скрывает текущий пин.

public void clickChoosePhotoButton - Открывает окно загрузки изображения.

public <T extends BasePage> T clickNotInterestedButton(Class<T> className) - Отмечает пин как "Не интересно".

public void clickLikeButton() - Ставит или снимает лайк с поста.

public String getDLikeButton() - Получает данные SVG-пути для кнопки лайка

public void fillText(String text) - Вводит текст комментария.

public void clickSendCommentButton() - Отправляет комментарий.

public void loadPhoto(File file) - Загружает фото.

public void clickDownloadPinButton() - Скачивает текущий пин.

public void clickSaveButton() - Сохраняет текущий пин.

public boolean isPostSaved() - Проверяет, сохранен ли пин

public String getCommentCount() - Возвращает кол-во комментариев.

public class SettingsPage extends BasePage - страница настроек и редактирования информации о пользователе

Поля:

private final Input firstNameInput - поле ввода имени пользователя

private final Input lastNameInput - поле ввода фамилии пользователя

private final TextArea aboutTextArea - поле ввода информации о себе

private final Button saveButton - кнопка сохранения информации

private final Button cancelButton - кнопка сброса введённых данных

Методы:

public void fillFirstName(String firstName) - Вводит имя пользователя.

public void fillLastName(String lastName) - Вводит фамилию пользователя.

public void fillAbout(String about) - Вводит информацию "О себе".

public void clickSaveButton() - Сохраняет изменения профиля.

public String getFirstName() - Получает текущее значение имени пользователя.

public String getLastName() - Получает текущее значение фамилии пользователя.

public String getAbout() - Получает текущую информацию "О себе"

Классы тестов

public class BaseTest - Базовый класс для UI тестов. Содержит общие настройки для всех тестовых классов.

Методы:

@BeforeEach

public void setUp() - Конфигурация тестового окружения перед каждым тестом. Устанавливает параметры браузера, таймауты и базовые настройки.

@AfterEach

public void tearDown() - Закрывает веб-драйвер после каждого теста.

protected MainPage auth(String login, String password) - Выполняет аутентификацию пользователя.

private void configureBrowser() - настраивает параметры браузера.

private void configureLogging() - Настраивает систему логирования.

protected void initTestData() - Инициализирует тестовые данные перед каждым тестом. Загружает учетные данные из .env файла.

public class CommentTest extends BaseTest - Тестовый класс для проверки функциональности комментирования пинов. Проверяет возможность добавления комментария с текстом и изображением.

Поля:

private static final String COMMENT_TEXT - текст комментария

private static final String IMAGE_PATH - путь к изображению

Методы:

@Test

void shouldAddCommentWithTextAndImage() - Проверяет возможность добавления комментария с текстом и изображением.

public class EditProfileTest extends BaseTest - Тестовый класс для проверки функциональности редактирования профиля пользователя. Проверяет возможность изменения имени, фамилии и информации "О себе".

Поля:

private static final String NAME_SUFFIX - суффикс который будет добавлен к имени

private static final String SURNAME_SUFFIX - суффикс который будет добавлен к фамилии

private static final String ABOUT_TEXT - суффикс который будет к полю “О себе”

Методы:

@Test

public void shouldSuccessfullyEditUserProfile() - Проверяет возможность редактирования профиля пользователя.

public class HideTest extends BaseTest - Тестовый класс для проверки функциональности скрытия пинов. Проверяет возможность скрытия пина и отмены этого действия.

Методы:

@Test

public void checkHidePost() - Проверяет функциональность скрытия пина и отмены этого действия.

public class LikeTest extends BaseTest - Тестовый класс для проверки функциональности лайков. Проверяет постановку и снятие лайка, а также сохранение состояния после обновления страницы.

Методы:

public void shouldToggleLikeAndPersistStateAfterRefresh() - Проверяет функциональность лайка

public class LoginTest extends BaseTest - Тестовый класс для проверки функциональности аутентификации. Проверяет сценарии входа с неверными и верными учетными данными.

Поля:

private static final String FAKE_PASSWORD - неверный пароль

private static final String INCORRECT_FORMAT_EMAIL - неверный email

Методы:

public void checkAuth() - проверяет авторизацию

public class OpenClosePostTest extends BaseTest - Тестовый класс для проверки функциональности открытия и закрытия постов. Проверяет корректность перехода на страницу поста и возврата на главную страницу.

Методы:

public void shouldOpenPostAndReturnToMainPage() - проверяет функциональность открытия и закрытия поста

public class SavePostTest extends BaseTest - Тестовый класс для проверки функциональности сохранения пина.

Методы:

public void savePost() - проверяет сохранился ли пост

public class SearchTest extends BaseTest - Тестовый класс для проверки функциональности поиска. Проверяет поиск по запросу.

Поля:

private static final String SEARCH_STR - строка, которая вводится в поле поиска

private static final List<String> SEARCH_KEYWORDS - слова, по которым проверяется, был ли произведён поиск корректно

Методы:

public void checkSearch() - проверяет поиск по запросу

public class SortingTest extends BaseTest - Тестовый класс для проверки функциональности сортировки досок. Проверяет сортировку по нажатию.

Методы:

public void checkSort() - проверяет сортировку досок

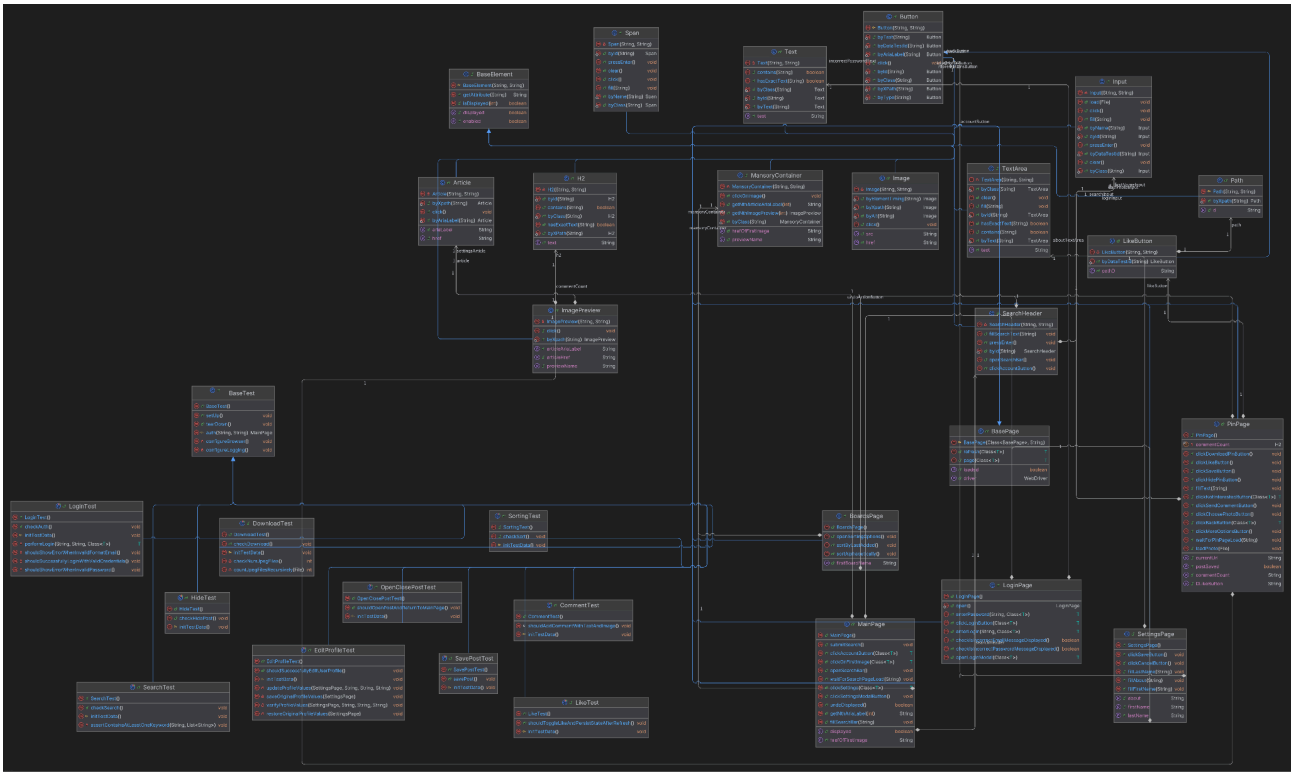
public class UploadTest extends BaseTest - Тестовый класс для проверки функциональности загрузки фото.

Методы:

public void checkUpload() - метод, проверяющий скачивается ли фото.

2.2. UML Диаграмма

При помощи плагинов и среды разработки INTELLIJ IDEA была создана UML диаграмма классов



3. ТЕСТИРОВАНИЕ

3.1. Работа тестов

Выполнение теста shouldAddCommentWithTextAndImage

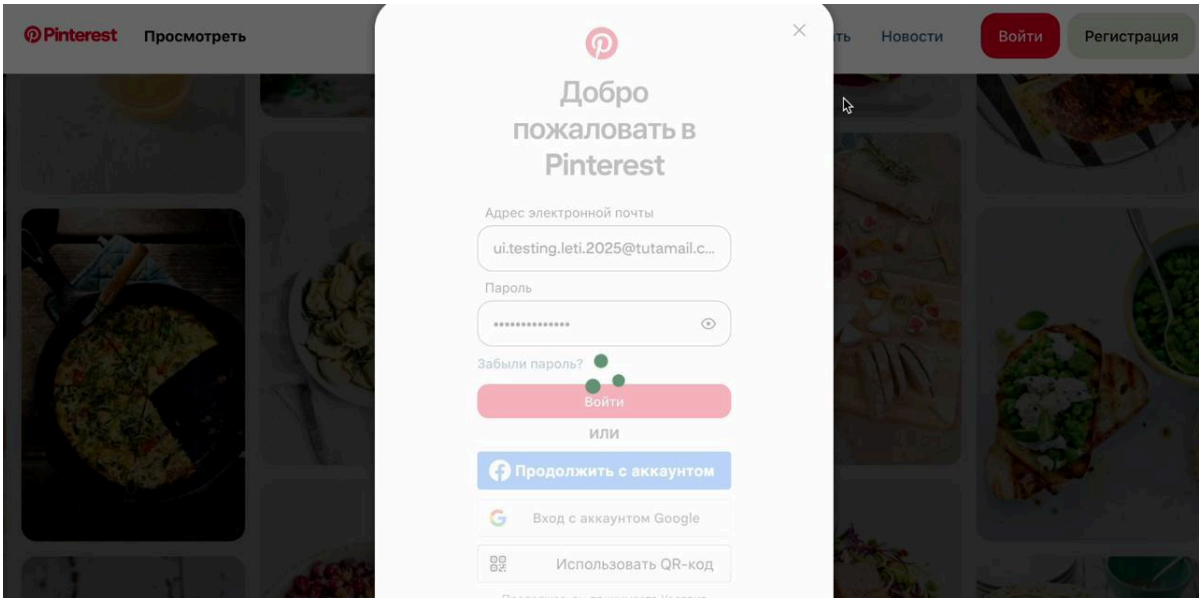


Рис. 3.1.1 - Авторизация на сайте

Наблюдаем, как пользователь вводит данные для входа в систему: логин и пароль. После ввода происходит нажатие на кнопку “Войти”, что инициирует процесс авторизации.

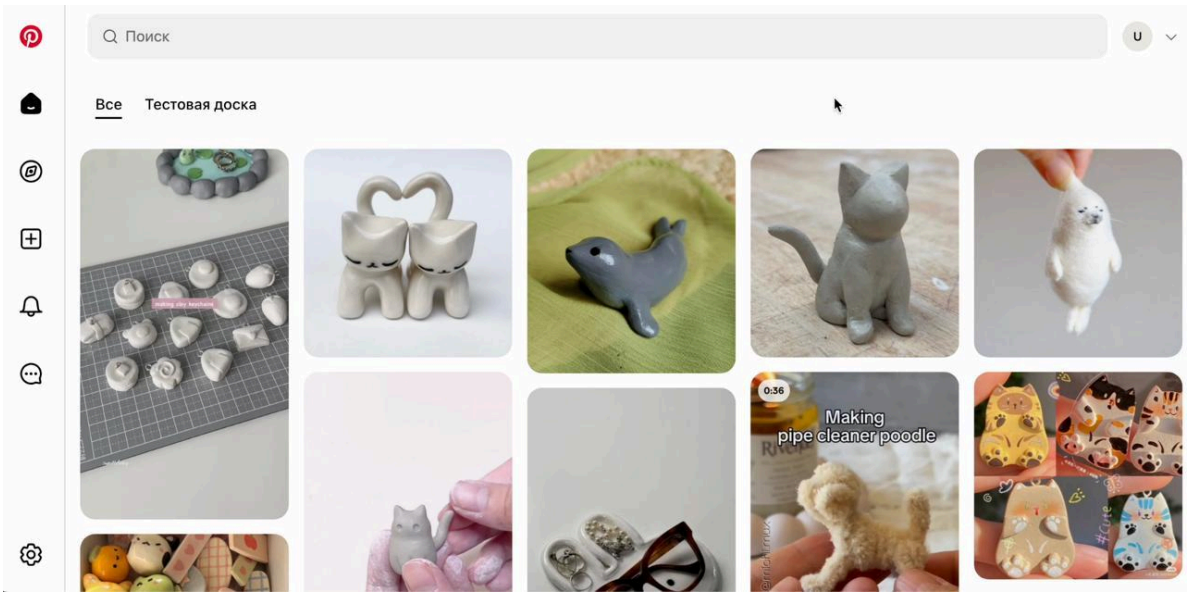


Рис. 3.1.2 - Главная страница сайта

После успешной авторизации пользователь попадает на главную страницу сайта Pinterest. Наблюдаем отображение ленты с постами — визуальным контентом, доступным для просмотра, взаимодействия и выбора для дальнейших действий.

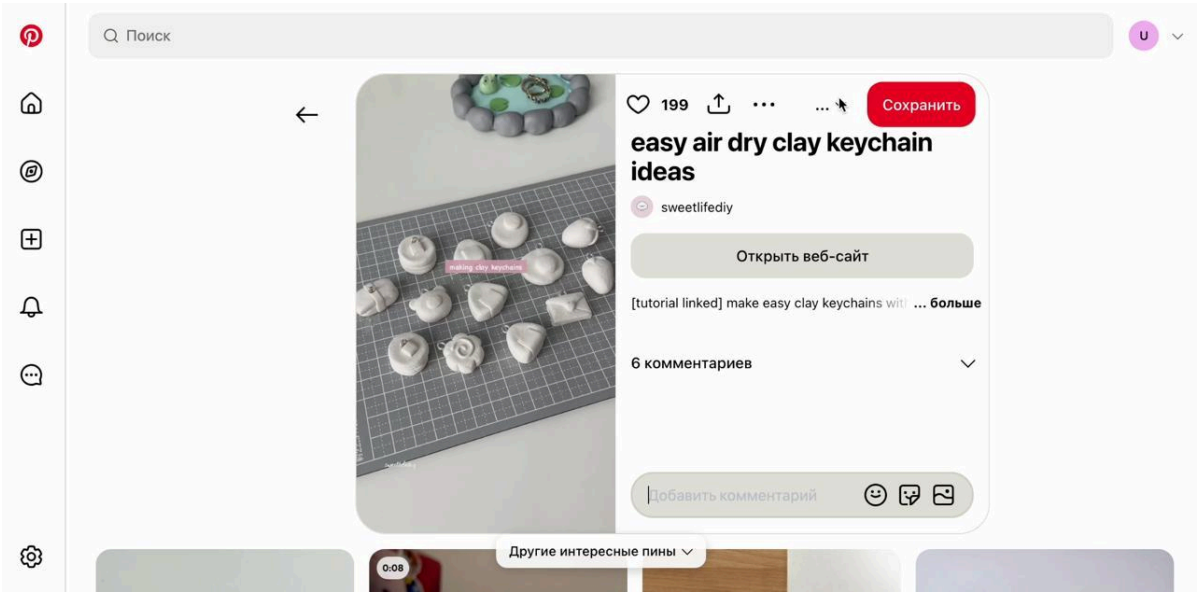


Рис. 3.1.3 - Страница первого поста с главной

Программа тестирования выбирает первый пост из ленты на главной странице. Наблюдаем переход на страницу поста, где отображается его подробное содержимое и доступны элементы для взаимодействия, включая поле добавления комментария.

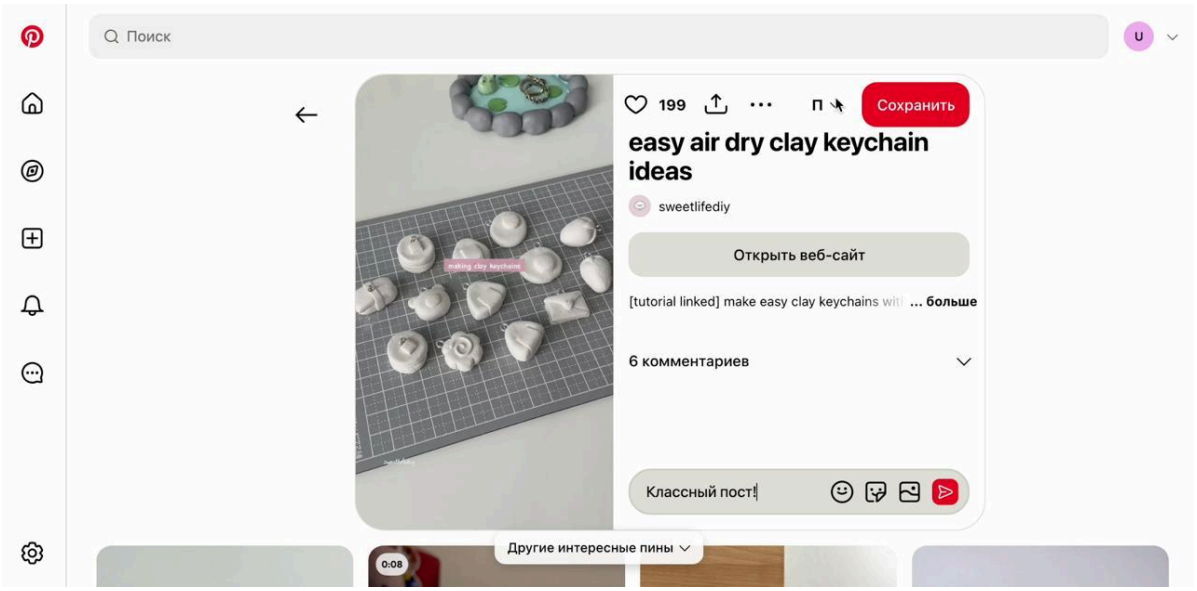


Рис. 3.1.4 - Ввод комментария

Программа тестирования находит поле для ввода комментария и автоматически вписывает текст: “Классный пост!”. Наблюдаем процесс симуляции пользовательского ввода

.

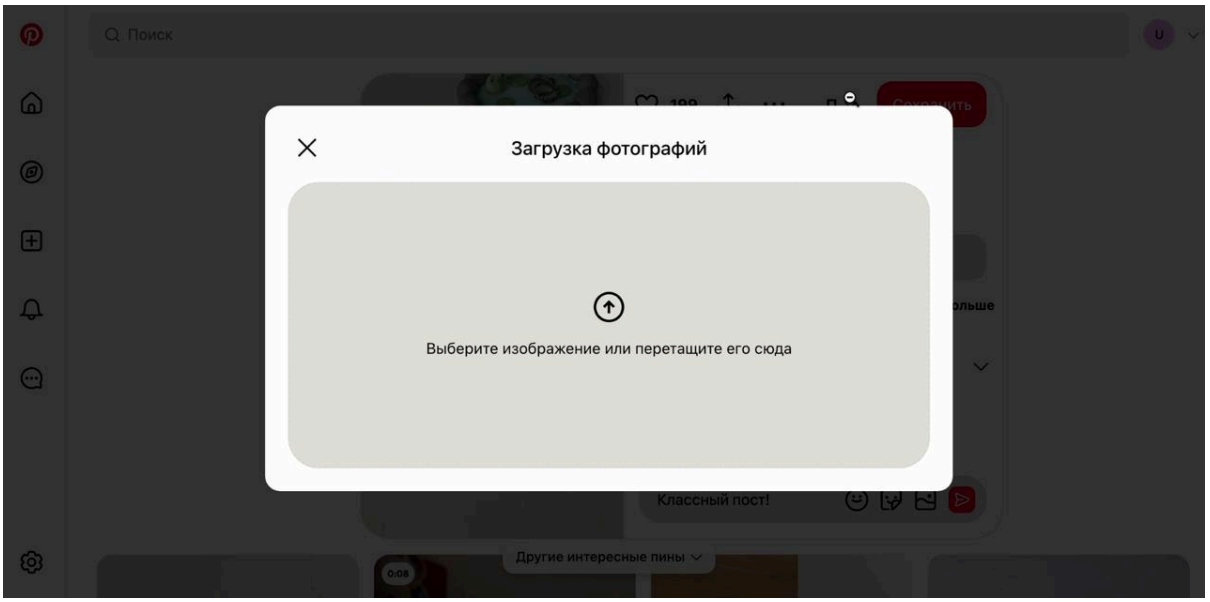


Рис. 3.1.5 - Окошко загрузки изображения

Программа тестирования инициирует действие по открытия окна для прикрепления изображения к комментарию.

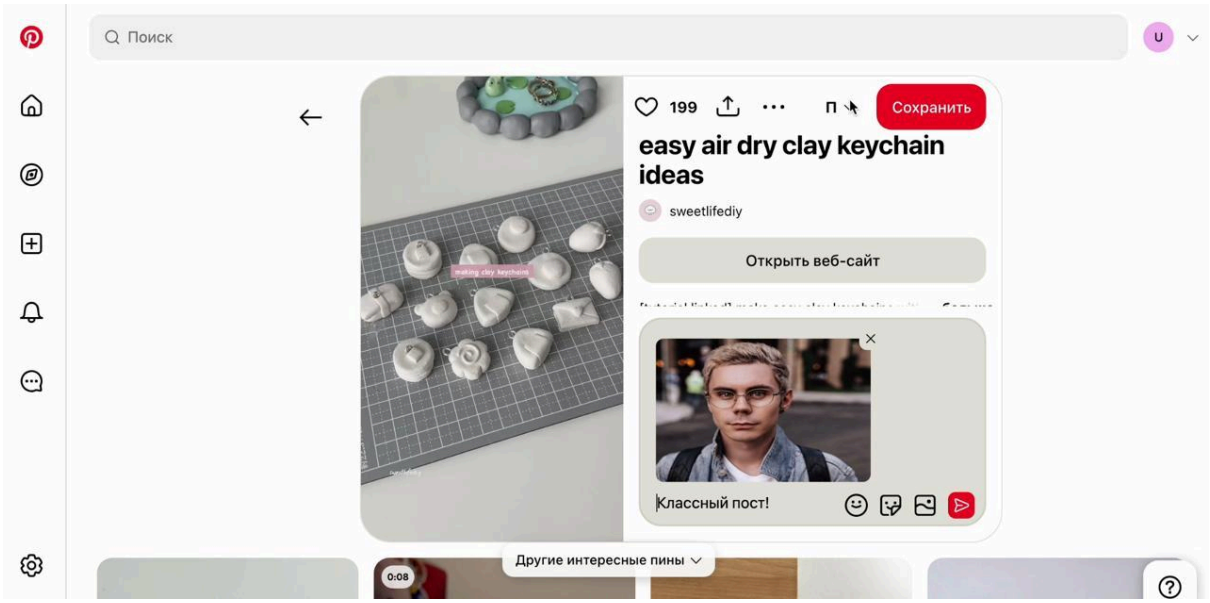


Рис. 3.1.6 - Изображение и текст прикреплены к комментарию

Окно загрузки закрывается, и программа тестирования проверяет, что изображение успешно прикреплено к комментарию вместе с текстом. Наблюдаем отображение загруженного изображения и введённого текста в поле комментария.

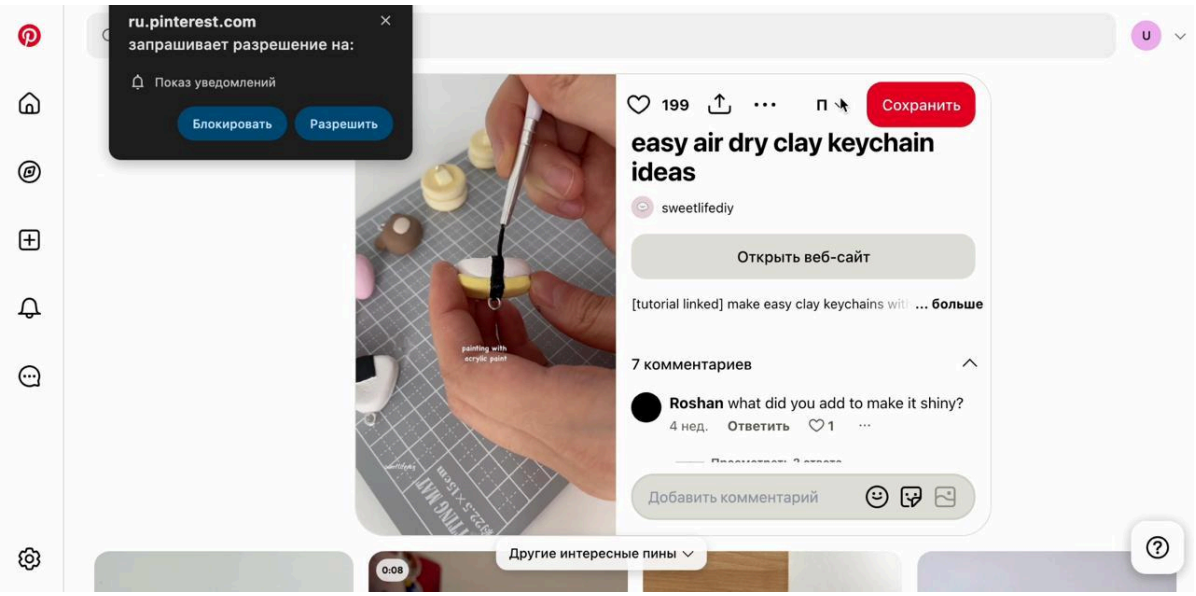


Рис. 3.1.7 - Отправка комментария, изменение количества комментариев у поста

Программа тестирования нажимает кнопку отправки комментария. Наблюдаем, как количество комментариев у поста увеличивается на один, подтверждая успешное добавление нового комментария с текстом и изображением.

ЗАКЛЮЧЕНИЕ

В ходе летней практики по автоматизации UI-тестирования веб-приложения Pinterest были достигнуты все поставленные цели и успешно решены задачи:

Разработка и запуск автотестов. Были созданы 8 автоматизированных тестов на основе Java, Selenide и JUnit 5, покрывающих ключевые пользовательские сценарии: авторизация, поиск и фильтрация товаров, работа с корзиной, управление избранным и создание смет. Каждый тест оформлен в виде чеклиста с подробным описанием шагов, входных данных и ожидаемых результатов.

Применение современных инструментов. Для управления проектом использовался Maven. Логирование выполнения тестов обеспечило наглядность отчётов и упростило отладку.

Проектирование структуры. Были реализованы удобные Page Object и элементные абстракции, что повысило читаемость и поддерживаемость кода. UML-диаграмма отразила архитектуру и помогла систематизировать классы и их взаимодействия.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. сайт Pinterest https://ru.pinterest.com/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: BaseElement.java

```
package com.example.elements;

import com.codeborne.selenide.SelenideElement;
import com.codeborne.selenide.ex.ElementNotFound;

import java.lang.reflect.UndeclaredThrowableException;
import java.time.Duration;

import static com.codeborne.selenide.Condition.visible;
import static com.codeborne.selenide.Selenide.$x;

/**
 * Базовый класс для всех элементов пользовательского интерфейса.
 * Инкапсулирует общую функциональность для работы с веб-элементами.
 */
public class BaseElement {
    protected static final int WAIT_SECONDS = 10;
    protected final SelenideElement baseElement;

    /**
     * Конструктор базового элемента.
     *
     * @param xpathTemplate шаблон XPath для поиска элемента
     * @param parameter     параметр для подстановки в шаблон XPath
     */
    protected BaseElement(String xpath, String attributeValue) {
        baseElement = $x(String.format(xpath, attributeValue));
```

```
    }

    /**
     * Проверяет, отображается ли элемент на странице.
     * Ожидает появление элемента в течение стандартного времени ожидания.
     *
     * @return true если элемент отображается, иначе false
     */
    public boolean isDisplayed() {
        try {
            return baseElement
                                                    .shouldBe(visible, Duration.ofSeconds(WAIT_SECONDS))
                    .isDisplayed();
        } catch (UndeclaredThrowableException | ElementNotFound e) {
            return false;
        }
    }

    /**
     * Проверяет, отображается ли элемент на странице с кастомным временем ожидания.
     *
     * @param seconds время ожидания в секундах
     * @return true если элемент отображается, иначе false
     */
    public boolean isDisplayed(int seconds) {
        try {
            baseElement.shouldBe(visible, Duration.ofSeconds(seconds));
            return baseElement.isDisplayed();
        } catch (UndeclaredThrowableException | ElementNotFound e) {
```

```
            return false;
        }
    }

    /**
     * Проверяет, доступен ли элемент для взаимодействия.
     *
     * @return true если элемент доступен, иначе false
     */
    public boolean isEnabled() {
        return baseElement.isEnabled();
    }

    /**
     * Возвращает значение указанного атрибута элемента.
     *
     * @param attributeName название атрибута
     * @return значение атрибута или null, если атрибут отсутствует
     */
    public String getAttribute(String attributeName) {
        return baseElement.getAttribute(attributeName);
    }
}
```

Название файла: Article.java

```
package com.example.elements.Article;

import com.example.elements.BaseElement;

/**
 * Класс, представляющий элемент статьи на веб-странице.
 * Инкапсулирует функциональность для работы со статьями.
 */
public class Article extends BaseElement {
            private static final String ARIA_LABEL_XPATH =
"//a[@aria-label='%s']";
```

```
private static final String HREF_ATTRIBUTE = "href";
private static final String ARIA_LABEL_ATTRIBUTE = "aria-label";

/**
 * Конструктор элемента статьи.
 *
 * @param xpathTemplate шаблон XPath для поиска элемента
 * @param parameter параметр для подстановки в шаблон XPath
 */
private Article(String xpathTemplate, String parameter) {
    super(xpathTemplate, parameter);
}

/**
 * Получает значение атрибута href элемента статьи.
 *
 * @return значение атрибута href
 */
public String getHref() {
    return baseElement.getAttribute(HREF_ATTRIBUTE);
}

/**
 * Получает значение атрибута aria-label элемента статьи.
 *
 * @return значение атрибута aria-label
 */
public String getAriaLabel() {
    return baseElement.getAttribute(ARIA_LABEL_ATTRIBUTE);
}
```

```
    /**
     * Выполняет клик по элементу статьи.
     */
    public void click() {
        baseElement.click();
    }

    /**
     * Находит элемент статьи по значению атрибута aria-label.
     *
     * @param ariaLabel значение атрибута aria-label
     * @return экземпляр элемента статьи
     */
    public static Article byAriaLabel(String ariaLabel) {
        return new Article(ARIA_LABEL_XPATH, ariaLabel);
    }

    /**
     * Находит элемент статьи по XPath.
     *
     * @param xPath XPath элемента
     * @return экземпляр элемента статьи
     */
    public static Article byXpath(String xPath) {
        return new Article("%s", xPath);
    }
}
```

Название файла: Button.java

```
package com.example.elements.Button;

import com.example.elements.BaseElement;
```

```
/**
 * Класс для работы с элементами типа "кнопка".
 * Предоставляет методы для поиска кнопок по различным атрибутам
 * и выполнения действий над ними.
 */
public class Button extends BaseElement {
    // Константы для шаблонов XPath
    private static final String ID_XPATH = "//button[@id='%s']";
                private    static    final    String    TEXT_XPATH    =
"//button[contains(text(),'%s')]";
    private static final String CLASS_XPATH = "//button[@class='%s']";
    private static final String TYPE_XPATH = "//button[@type='%s']";
             private   static   final   String   ARIA_LABEL_XPATH   =
"//button[@aria-label='%s']";
            private   static   final   String   DATA_TEST_ID_XPATH   =
"//button[@data-test-id='%s']";

    /**
     * Конструктор класса.
     *
     * @param xpathTemplate шаблон XPath для поиска элемента
     * @param parameter параметр для подстановки в шаблон XPath
     */
    protected Button(String xpathTemplate, String parameter) {
        super(xpathTemplate, parameter);
    }

    /**
     * Выполняет клик по кнопке.
     */
    public void click() {
```

```
    baseElement.click();
}

/**
 * Находит кнопку по идентификатору.
 *
 * @param id идентификатор кнопки
 * @return экземпляр класса Button
 */
public static Button byId(String id) {
    return new Button(ID_XPATH, id);
}

/**
 * Находит кнопку по тексту.
 *
 * @param text текст кнопки
 * @return экземпляр класса Button
 */
public static Button byText(String text) {
    return new Button(TEXT_XPATH, text);
}

/**
 * Находит кнопку по названию класса.
 *
 * @param className название класса
 * @return экземпляр класса Button
 */
public static Button byClass(String className) {
    return new Button(CLASS_XPATH, className);
```

```
}

/**
 * Находит кнопку по типу.
 *
 * @param type тип кнопки
 * @return экземпляр класса Button
 */
public static Button byType(String type) {
    return new Button(TYPE_XPATH, type);
}

/**
 * Находит кнопку по значению атрибута aria-label.
 *
 * @param ariaLabel значение атрибута aria-label
 * @return экземпляр класса Button
 */
public static Button byAriaLabel(String ariaLabel) {
    return new Button(ARIA_LABEL_XPATH, ariaLabel);
}

/**
 * Находит кнопку по указанному XPath.
 *
 * @param xPath XPath для поиска элемента
 * @return экземпляр класса Button
 */
public static Button byXPath(String xPath) {
    return new Button("%s", xPath);
}
```

```
    /**
     * Находит кнопку по значению атрибута data-test-id.
     *
     * @param dataTestId значение атрибута data-test-id
     * @return экземпляр класса Button
     */
    public static Button byDataTestId(String dataTestId) {
        return new Button(DATA_TEST_ID_XPATH, dataTestId);
    }
}
```

Название файла H2.java

```
package com.example.elements.H2;

import com.example.elements.BaseElement;

/**
 * Класс для работы с текстовыми элементами.
 * Предоставляет методы для поиска текстовых элементов и проверки их
 * содержимого.
 */
public class H2 extends BaseElement {
    private static final String ID_XPATH = "//h2[@id='%s']";
    private static final String CLASS_XPATH = "//h2[@class='%s']";
                private    static    final    String    TEXT_XPATH    =
"//h2[contains(text(),'%s')]";

    /**
     * Конструктор класса.
     *
     * @param xpath шаблон XPath для поиска элемента
```

```
     * @param param параметр для подстановки в шаблон XPath
     */
    private H2(String xpath, String param) {
        super(xpath, param);
    }

    /**
     * Проверяет, содержит ли элемент указанный текст.
     *
     * @param text текст для проверки
     * @return true если элемент содержит указанный текст, иначе false
     */
    public boolean contains(String text) {
        return baseElement.getText().contains(text);
    }

    /**
     * Проверяет точное соответствие текста элемента.
     *
     * @param text ожидаемый текст
     * @return true если текст элемента точно соответствует ожидаемому,
иначе false
     */
    public boolean hasExactText(String text) {
        return baseElement.getText().equals(text);
    }

    /**
     * Возвращает текст элемента.
     *
     * @return текст элемента
```

```
     */
    public String getText() {
        return baseElement.getText();
    }

    /**
     * Находит текстовый элемент по идентификатору.
     *
     * @param id идентификатор элемента
     * @return экземпляр класса H2
     */
    public static H2 byId(String id) {
        return new H2(ID_XPATH, id);
    }

    /**
     * Находит текстовый элемент по названию класса.
     *
     * @param className название класса
     * @return экземпляр класса H2
     */
    public static H2 byClass(String className) {
        return new H2(CLASS_XPATH, className);
    }

    /**
     * Находит H2 по указанному XPath.
     *
     * @param xpath XPath для поиска элемента
     * @return экземпляр класса H2
     */
```

```
    public static H2 byXPath(String xpath) {
        return new H2("%s", xpath);
    }
}
```

Название файла: Image.java

```
package com.example.elements.Image;

import com.example.elements.BaseElement;

/**
 * Класс для работы с элементами типа "изображение".
 * Предоставляет методы для поиска изображений по различным атрибутам
 * и выполнения действий над ними.
 */
public class Image extends BaseElement {
    // Константы для шаблонов XPath
    private static final String ALT_XPATH = "//img[@alt='%s']";
            private static final String ELEMENT_TIMING_XPATH =
"//img[@elementtiming='%s']";
    private static final String HREF_ATTRIBUTE = "href";
    private static final String SRC_ATTRIBUTE = "src";

    /**
     * Конструктор класса.
     *
     * @param xpathTemplate шаблон XPath для поиска элемента
     * @param parameter параметр для подстановки в шаблон XPath
     */
    private Image(String xpathTemplate, String parameter) {
        super(xpathTemplate, parameter);
    }
```

```
/**
 * Выполняет клик по изображению.
 */
public void click() {
    baseElement.click();
}

/**
 * Получает значение атрибута href изображения.
 * Примечание: используется, когда изображение является ссылкой.
 *
 * @return значение атрибута href
 */
public String getHref() {
    return baseElement.getAttribute(HREF_ATTRIBUTE);
}

/**
 * Получает значение атрибута src изображения.
 *
 * @return значение атрибута src
 */
public String getSrc() {
    return baseElement.getAttribute(SRC_ATTRIBUTE);
}

/**
 * Находит изображение по значению атрибута alt.
 *
 * @param alt значение атрибута alt
```

```
     * @return экземпляр класса Image
     */
    public static Image byAlt(String alt) {
        return new Image(ALT_XPATH, alt);
    }

    /**
     * Находит изображение по значению атрибута elementtiming.
     *
     * @param elementTiming значение атрибута elementtiming
     * @return экземпляр класса Image
     */
    public static Image byElementTiming(String elementTiming) {
        return new Image(ELEMENT_TIMING_XPATH, elementTiming);
    }

    /**
     * Находит изображение по указанному XPath.
     *
     * @param xPath XPath для поиска элемента
     * @return экземпляр класса Image
     */
    public static Image byXpath(String xPath) {
        return new Image("%s", xPath);
    }
}
```

Название файла: ImagePreview.java

```
package com.example.elements.ImagePreview;

import com.example.elements.Article.Article;
import com.example.elements.BaseElement;
```

```
import com.example.elements.H2.H2;


/**
 * Класс для работы с превью изображений.
  * Представляет составной элемент, содержащий связанную статью и заголовок.
 */
public class ImagePreview extends BaseElement {
    // Константы для XPath суффиксов
    private static final String ARTICLE_XPATH_SUFFIX = "//a";
    private static final String H2_XPATH_SUFFIX = "//h2";


    private final Article article;
    private final H2 h2;


    /**
     * Конструктор класса.
     *
     * @param xpathTemplate базовый XPath для превью изображения
     * @param parameter параметр для подстановки в шаблон XPath
     */
    private ImagePreview(String xpathTemplate, String parameter) {
        super(xpathTemplate, parameter);
                    this.article = Article.byXpath(xpathTemplate + ARTICLE_XPATH_SUFFIX);
        this.h2 = H2.byXPath(xpathTemplate + H2_XPATH_SUFFIX);
    }


    /**
     * Получает заголовок превью изображения.
     *
     * @return текст заголовка
```

```
     */
    public String getPreviewName() {
        return h2.getText();
    }

    /**
     * Получает значение атрибута aria-label связанной статьи.
     *
     * @return значение атрибута aria-label
     */
    public String getArticleAriaLabel() {
        return article.getAriaLabel();
    }

    /**
     * Получает ссылку на связанную статью.
     *
     * @return URL статьи
     */
    public String getArticleHref() {
        return article.getHref();
    }

    /**
     * Выполняет переход по ссылке превью изображения.
     */
    public void click() {
        article.click();
    }

    /**
```

```
     * Находит превью изображения по указанному XPath.
     *
     * @param xPath XPath для поиска элемента
     * @return экземпляр класса ImagePreview
     */
    public static ImagePreview byXpath(String xPath) {
        return new ImagePreview(xPath, "");
    }
}
```

Название файла: Input.java

```
package com.example.elements.Input;

import com.codeborne.selenide.Condition;
import com.codeborne.selenide.Selenide;
import com.example.elements.BaseElement;
import org.openqa.selenium.Keys;

import java.io.File;

import static com.codeborne.selenide.Selenide.$;

/**
 * Класс для работы с элементами типа "поле ввода".
 * Предоставляет методы для поиска полей ввода по различным атрибутам
 * и выполнения действий над ними.
 */
public class Input extends BaseElement {
    private static final String ID_XPATH = "//input[@id='%s']";
    private static final String NAME_XPATH = "//input[@name='%s']";
    private static final String CLASS_XPATH = "//input[@class='%s']";
```

```
            private static final String DATA_TEST_ID_XPATH =
"//input[@data-test-id='%s']";

    /**
     * Конструктор класса.
     *
     * @param xpath шаблон XPath для поиска элемента
     * @param param параметр для подстановки в шаблон XPath
     */
    private Input(String xpath, String param) {
        super(xpath, param);
    }

    /**
     * Очищает поле ввода.
     */
    public void clear() {
        while (!baseElement.getAttribute("value").equals("")) {
            baseElement.sendKeys(Keys.BACK_SPACE);
        }
    }

    /**
     * Вводит указанное значение в поле ввода.
     *
     * @param value значение для ввода
     */
    public void fill(String value) {
        clear();
        baseElement.sendKeys(value);
    }
```

```
    /**
     * Метод позволяющий загружать файлы
     */
    public void load(File file) {
            System.out.println("[INFO] Starting file upload for: " +
file.getName());
                    System.out.println("[DEBUG] File path: " +
file.getAbsolutePath());

        if (!file.exists()) {
                    String errorMsg = "[ERROR] File not found: " +
file.getAbsolutePath();
            System.err.println(errorMsg);
            throw new RuntimeException(errorMsg);
        }

        baseElement.sendKeys(file.getAbsolutePath());
                    System.out.println("[SUCCESS]: File uploaded." +
file.getName());
    }

    /**
     * Кликает на элемент.
     */
    public void click() {
        baseElement.click();
    }

    /**
     * Нажимает Enter для подтверждения.
     */
```

```
public void pressEnter() {
    baseElement.pressEnter();
}

/**
 * Находит поле ввода по идентификатору.
 *
 * @param id идентификатор поля ввода
 * @return экземпляр класса Input
 */
public static Input byId(String id) {
    return new Input(ID_XPATH, id);
}

/**
 * Находит поле ввода по dataTextId.
 *
 * @param dataTextId идентификатор поля ввода
 * @return экземпляр класса Input
 */
public static Input byDataTestId(String dataTextId) {
    return new Input(DATA_TEST_ID_XPATH, dataTextId);
}

/**
 * Находит поле ввода по имени.
 *
 * @param name имя поля ввода
 * @return экземпляр класса Input
 */
public static Input byName(String name) {
```

```
        return new Input(NAME_XPATH, name);
    }

    /**
     * Находит поле ввода по названию класса.
     *
     * @param text название класса
     * @return экземпляр класса Input
     */
    public static Input byClass(String text) {
        return new Input(CLASS_XPATH, text);
    }
}
```

Название файла: LikeButton.java

```
package com.example.elements.likebutton;

import com.example.elements.Button.Button;
import com.example.elements.path.Path;

/**
 * Класс для работы с кнопкой "лайк".
 * Расширяет функциональность базовой кнопки, добавляя специфичные для лайков методы.
 */
public class LikeButton extends Button {
    // Константы для шаблонов XPath
            private static final String PATH_XPATH_SUFFIX = "//*[name()='path']";
            private static final String DATA_TEST_ID_XPATH = "//button[@data-test-id='%s']";

    // Дочерний элемент
```

```
    private final Path path;

    /**
     * Конструктор класса.
     *
     * @param xpathTemplate шаблон XPath для поиска элемента
     * @param parameter параметр для подстановки в шаблон XPath
     */
    private LikeButton(String xpathTemplate, String parameter) {
        super(xpathTemplate, parameter);
                this.path  =  Path.byXpath(String.format(xpathTemplate,
parameter) + PATH_XPATH_SUFFIX);
    }

    /**
     * Получает значение атрибута 'd' элемента path внутри кнопки.
     * Используется для проверки состояния лайка (заполненный/пустой).
     *
     * @return значение атрибута 'd'
     */
    public String getPathD() {
        return path.getD();
    }

    /**
     * Находит кнопку лайка по значению атрибута data-test-id.
     *
     * @param dataTestId значение атрибута data-test-id
     * @return экземпляр класса LikeButton
     */
    public static LikeButton byDataTestId(String dataTestId) {
```

```
        return new LikeButton(DATA_TEST_ID_XPATH, dataTestId);
    }
}
```

Название файла: MansoryContainer.java

```
package com.example.elements.MansoryContainer;

import com.example.elements.BaseElement;
import com.example.elements.ImagePreview.ImagePreview;

/**
 * Класс для работы с Masonry-контейнером изображений.
 * Представляет контейнер, содержащий превью изображений в виде сетки.
 */
public class MansoryContainer extends BaseElement {
    private static final String CLASS_XPATH = "//div[@class='%s']";
            private static final String IMAGE_PREVIEW_XPATH =
"//*[@role=\"listitem\"][%d]";  // Updated with placeholder for index

    /**
     * Конструктор класса.
     *
     * @param xpath шаблон XPath для поиска элемента
     * @param param     параметр для подстановки в шаблон XPath
     */
    private MansoryContainer(String xpath, String param) {
        super(xpath, param);
    }

    /**
         * Получает ссылку на статью первого превью изображения в
контейнере.
     *
```

```
     * @return URL статьи
     */
    public String getHrefOfFirstImage() {
        return getNthImagePreview(1).getArticleHref();
    }

    /**
     * Получает превью изображения по порядковому номеру.
     *
     * @param n порядковый номер превью (начиная с 1)
     * @return экземпляр превью изображения
     * @throws IllegalArgumentException если индекс меньше 1
     */
    public ImagePreview getNthImagePreview(int n) {
        if (n < 1) {
                throw new IllegalArgumentException("Index must be 1 or
greater");
        }
         return ImagePreview.byXpath(String.format(IMAGE_PREVIEW_XPATH,
n));
    }

    /**
     * Выполняет клик по первому превью изображения в контейнере.
     */
     public void clickOnImage() {// Uses getNthImagePreview instead of
firstImagePreview
        getNthImagePreview(1).click();
    }

    /**
     * Получает название первого превью изображения в контейнере.
```

```
     *
     * @return название превью
     */
    public String getPreviewName() {
        return getNthImagePreview(1).getPreviewName();
    }

    /**
     * Получает значение атрибута aria-label статьи для n-го превью.
     *
     * @param n порядковый номер превью (начиная с 1)
     * @return значение атрибута aria-label
     */
    public String getNthArticleAriaLabel(int n) {
        return getNthImagePreview(n).getArticleAriaLabel();
    }

    /**
     * Находит Masonry-контейнер по названию класса.
     *
     * @param className название класса контейнера
     * @return экземпляр класса MansoryContainer
     */
    public static MansoryContainer byClass(String className) {
        return new MansoryContainer(CLASS_XPATH, className);
    }
}
```

Название файла Path.java

```
package com.example.elements.path;

import com.example.elements.BaseElement;
```

```
/**
 * Класс для работы с элементами SVG-пути.
  * Предоставляет методы для взаимодействия с тегом &lt;path&gt; в SVG-графике.
 */
public class Path extends BaseElement {
    private static final String D_ATTRIBUTE = "d";

    /**
     * Конструктор класса.
     *
     * @param xpathTemplate шаблон XPath для поиска элемента
     * @param parameter параметр для подстановки в шаблон XPath
     */
    protected Path(String xpathTemplate, String parameter) {
        super(xpathTemplate, parameter);
    }

    /**
     * Получает значение атрибута 'd' элемента пути.
     * Атрибут 'd' содержит команды для построения SVG-пути.
     *
     * @return значение атрибута 'd'
     */
    public String getD() {
        return getAttribute(D_ATTRIBUTE);
    }

    /**
     * Находит элемент пути по указанному XPath.
```

```
     *
     * @param xPath XPath для поиска элемента
     * @return экземпляр класса Path
     */
    public static Path byXpath(String xPath) {
        return new Path("%s", xPath);
    }
}
```

Название файла: SearchHeader.java

```
package com.example.elements.SearchHeader;


import com.example.elements.BaseElement;
import com.example.elements.Button.Button;
import com.example.elements.Input.Input;


/**
 * Класс для работы с шапкой поиска.
 * Инкапсулирует функциональность элементов поисковой строки и кнопок в шапке приложения.
 */
public class SearchHeader extends BaseElement {
    private static final String CLASS_XPATH = "//*[@id='%s']";


    private final Input searchInput = Input.byName("searchBoxInput");
                    private     final     Button     accountButton     =
Button.byXPath("//*[@id=\"HeaderContent\"]/div/div/div[2]/div/div/div/d
iv[2]/div/div/div/a");



    /**
     * Конструктор класса.
     *
```

```
     * @param xpath шаблон XPath для поиска элемента
     * @param param     параметр для подстановки в шаблон XPath
     */
    private SearchHeader(String xpath, String param) {
        super(xpath, param);
    }

    /**
     * Активирует поле поиска, выполняя по нему клик.
     */
    public void openSearchBar() {
        searchInput.click();
    }

    /**
     * Вводит текст в поле поиска.
     *
     * @param text текст для поиска
     */
    public void fillSearchText(String text) {
        searchInput.fill(text);
    }

    /**
     * Выполняет поиск, нажимая Enter в поле поиска.
     */
    public void pressEnter() {
        searchInput.pressEnter();
    }

    /**
```

```
     * Нажимает на кнопку аккаунта
     */
    public void clickAccountButton() {
        accountButton.click();
    }

    /**
     * Находит шапку поиска по идентификатору.
     *
     * @param text идентификатор элемента шапки
     * @return экземпляр класса SearchHeader
     */
    public static SearchHeader byId(String text) {
        return new SearchHeader(CLASS_XPATH, text);
    }

}
```

Название файла: Text.java

```
package com.example.elements.Text;

import com.example.elements.BaseElement;

/**
 * Класс для работы с текстовыми элементами.
 * Предоставляет методы для поиска текстовых элементов и проверки их
 * содержимого.
 */
public class Text extends BaseElement {
    private static final String ID_XPATH = "//span[@id='%s']";
    private static final String CLASS_XPATH = "//span[@class='%s']";
```

```
                private    static    final    String    TEXT_XPATH    =
"//span[contains(text(),'%s')]";


    /**
     * Конструктор класса.
     *
     * @param xpath шаблон XPath для поиска элемента
     * @param param параметр для подстановки в шаблон XPath
     */
    private Text(String xpath, String param) {
        super(xpath, param);
    }


    /**
     * Проверяет, содержит ли элемент указанный текст.
     *
     * @param text текст для проверки
     * @return true если элемент содержит указанный текст, иначе false
     */
    public boolean contains(String text) {
        return baseElement.getText().contains(text);
    }


    /**
     * Проверяет точное соответствие текста элемента.
     *
     * @param text ожидаемый текст
      * @return true если текст элемента точно соответствует ожидаемому,
иначе false
     */
    public boolean hasExactText(String text) {
        return baseElement.getText().equals(text);
```

```
}

/**
 * Возвращает текст элемента.
 *
 * @return текст элемента
 */
public String getText() {
    return baseElement.getText();
}

/**
 * Находит текстовый элемент по идентификатору.
 *
 * @param id идентификатор элемента
 * @return экземпляр класса Text
 */
public static Text byId(String id) {
    return new Text(ID_XPATH, id);
}

/**
 * Находит текстовый элемент по названию класса.
 *
 * @param className название класса
 * @return экземпляр класса Text
 */
public static Text byClass(String className) {
    return new Text(CLASS_XPATH, className);
}
```

```
    /**
     * Находит текстовый элемент по содержащемуся тексту.
     *
     * @param text текст элемента
     * @return экземпляр класса Text
     */
    public static Text byText(String text) {
        return new Text(TEXT_XPATH, text);
    }
}
```

Название файла TextArea.java

```
package com.example.elements.TextArea;

import com.example.elements.BaseElement;
import org.openqa.selenium.Keys;

/**
 * Класс для работы с текстовыми элементами.
 * Предоставляет методы для поиска текстовых элементов и проверки их
 * содержимого.
 */
public class TextArea extends BaseElement {
    private static final String ID_XPATH = "//textarea[@id='%s']";
                private static final String CLASS_XPATH =
"//textarea[@class='%s']";
                private static final String TEXT_XPATH =
"//textarea[contains(text(),'%s')]";

    /**
     * Конструктор класса.
     *
     * @param xpath шаблон XPath для поиска элемента
```

```
     * @param param параметр для подстановки в шаблон XPath
     */
    private TextArea(String xpath, String param) {
        super(xpath, param);
    }

    /**
     * Проверяет, содержит ли элемент указанный текст.
     *
     * @param text текст для проверки
     * @return true если элемент содержит указанный текст, иначе false
     */
    public boolean contains(String text) {
        return baseElement.getText().contains(text);
    }

    /**
     * Проверяет точное соответствие текста элемента.
     *
     * @param text ожидаемый текст
     * @return true если текст элемента точно соответствует ожидаемому,
иначе false
     */
    public boolean hasExactText(String text) {
        return baseElement.getText().equals(text);
    }

    /**
     * Возвращает текст элемента.
     *
     * @return текст элемента
```

```
     */
    public String getText() {
        return baseElement.getText();
    }

    public void clear() {
        while (!baseElement.getAttribute("value").equals("")) {
            baseElement.sendKeys(Keys.BACK_SPACE);
        }
    }

    public void fill(String value) {
        clear();
        baseElement.sendKeys(value);
    }

    /**
     * Находит текстовый элемент по идентификатору.
     *
     * @param id идентификатор элемента
     * @return экземпляр класса Text
     */
    public static TextArea byId(String id) {
        return new TextArea(ID_XPATH, id);
    }

    /**
     * Находит текстовый элемент по названию класса.
     *
     * @param className название класса
     * @return экземпляр класса Text
```

```
     */
    public static TextArea byClass(String className) {
        return new TextArea(CLASS_XPATH, className);
    }

    /**
     * Находит текстовый элемент по содержащемуся тексту.
     *
     * @param text текст элемента
     * @return экземпляр класса Text
     */
    public static TextArea byText(String text) {
        return new TextArea(TEXT_XPATH, text);
    }
}
```

Название файла BasePage.java

```
package com.example.pages;

import com.codeborne.selenide.Selenide;
import com.codeborne.selenide.SelenideElement;
import org.openqa.selenium.WebDriver;

import static com.codeborne.selenide.Selenide.$x;

/**
 * Базовый класс для всех страниц приложения.
 * Инкапсулирует общую функциональность работы со страницами.
 */
public class BasePage {
            private static final String BASE_ELEMENT_XPATH =
"//div[contains(@name,'%s')]";
```

```
    protected final SelenideElement basePage;
    protected final Class<? extends BasePage> pageClass;


    /**
     * Конструктор базовой страницы.
     *
     * @param pageClass                класс страницы (для рефлексии)
     * @param uniqueElementIdentifier уникальный идентификатор элемента страницы
     */
      protected BasePage(Class<? extends BasePage> pageClass, String uniqueElementIdentifier) {
                this.basePage  =  $x(String.format(BASE_ELEMENT_XPATH, uniqueElementIdentifier));
        this.pageClass = pageClass;
    }


    /**
     * Проверяет, загружена ли страница.
     *
     * @return true если базовый элемент страницы отображается, иначе false
     */
    public boolean isLoaded() {
        return basePage.isDisplayed();
    }


    /**
     * Обновляет текущую страницу.
     *
     * @param pageClass класс страницы, которую нужно вернуть
     * @param <T>       тип страницы
```

```
     * @return новый экземпляр обновлённой страницы
     */
    public <T extends BasePage> T refresh(Class<T> pageClass) {
        Selenide.refresh();
        return page(pageClass);
    }


    /**
     * Создаёт экземпляр страницы указанного класса.
     *
     * @param pageClass класс страницы
     * @param <T>       тип страницы
     * @return новый экземпляр страницы
     */
    public <T extends BasePage> T page(Class<T> pageClass) {
        try {
            return pageClass.getDeclaredConstructor().newInstance();
        } catch (Exception e) {
             throw new RuntimeException("Failed to create page instance
for " + pageClass.getSimpleName(), e);
        }
    }


    /**
     * Получает экземпляр WebDriver из Selenide
     *
     * @return текущий экземпляр WebDriver
     */
    public WebDriver getDriver() {
        return Selenide.webdriver().driver().getWebDriver();
```

```
    }
}
```

Название файла: BoardsPage.java

```
package com.example.pages;


import com.example.elements.Button.Button;

import com.example.elements.MansoryContainer.MansoryContainer;


/**
 * Страница досок пользователя.
 * Предоставляет функционал для работы с досками и их сортировкой.
 */
public class BoardsPage extends BasePage {

            private static final String MAIN_CONTAINER_XPATH =
"//*[@class=\"mainContainer\"]";

       private static final String MORE_OPTIONS_BUTTON_ARIA_LABEL =
"Упорядочить";

         private static final String ALPHABETIC_ORDER_BUTTON_ID =
"actionBarMenuButton-item-0";

         private static final String LAST_ADDED_ORDER_BUTTON_ID =
"actionBarMenuButton-item-2";

          private static final String MASONRY_CONTAINER_CLASS =
"masonryContainer";


    // Элементы страницы

                  private final Button moreOptionsButton =
Button.byAriaLabel(MORE_OPTIONS_BUTTON_ARIA_LABEL);

                private final Button alphabeticOrderButton =
Button.byId(ALPHABETIC_ORDER_BUTTON_ID);

                 private final Button lastAddedOrderButton =
Button.byId(LAST_ADDED_ORDER_BUTTON_ID);

              private final MansoryContainer mansoryContainer =
MansoryContainer.byClass(MASONRY_CONTAINER_CLASS);


    /**
```

```
     * Конструктор страницы досок.
     * Инициализирует страницу с валидационным XPath.
     */
    public BoardsPage() {
        super(BoardsPage.class, MAIN_CONTAINER_XPATH);
    }

    /**
     * Открывает меню дополнительных опций сортировки.
     */
    public void openSortingOptions() {
        moreOptionsButton.click();
    }

    /**
     * Выбирает сортировку досок по алфавиту.
     */
    public void sortAlphabetically() {
        alphabeticOrderButton.click();
    }

    /**
         * Выбирает сортировку досок по дате добавления (последние
добавленные).
     */
    public void sortByLastAdded() {
        lastAddedOrderButton.click();
    }

    /**
     * Получает название первого превью доски в контейнере.
```

```
     *
     * @return название доски
     */
    public String getFirstBoardName() {
        return mansoryContainer.getPreviewName();
    }

}
```

Название файла LoginPage.java

```
package com.example.pages;


import com.example.elements.Button.Button;
import com.example.elements.Input.Input;
import com.example.elements.Text.Text;


/**
 * Страница аутентификации пользователя.
 * Предоставляет методы для взаимодействия с элементами входа.
 */
public class LoginPage extends BasePage {

            private static final String LOGIN_BUTTON_XPATH =
"//*[@id=\"__PWS_ROOT__\"]/div[1]/div[1]/div[2]/div/div[2]/div[2]/butto
n";

           private static final String PAGE_VALIDATION_XPATH =
"//*[@id=\"fullpage\"]";


                 private final Button loginModalButton =
Button.byXPath(LOGIN_BUTTON_XPATH);
    private final Input loginInput = Input.byId("email");
    private final Input passwordInput = Input.byId("password");
    private final Button submitButton = Button.byType("submit");
                private final Text incorrectPasswordText =
Text.byId("password-error");
```

```
            private final Text incorrectEmailFormatText =
Text.byId("email-error");


    /**
     * Конструктор класса.
     * Инициализирует страницу с валидационным XPath.
     */
    public LoginPage() {
        super(LoginPage.class, PAGE_VALIDATION_XPATH);
    }


    /**
     * Открывает модальное окно аутентификации.
     *
     * @param nextPageClass класс возвращаемой страницы
     * @param <T>           тип страницы
     * @return текущий экземпляр страницы
     */
          public <T extends BasePage> T openLoginModal(Class<T>
nextPageClass) {
        loginModalButton.click();
        return page(nextPageClass);
    }


    /**
     * Вводит логин в поле ввода.
     *
     * @param login         логин пользователя
     * @param nextPageClass класс возвращаемой страницы
     * @param <T>           тип страницы
     * @return текущий экземпляр страницы
     */
```

```
     public <T extends BasePage> T enterLogin(String login, Class<T>
nextPageClass) {
        loginInput.fill(login);
        return page(nextPageClass);
    }

    /**
     * Вводит пароль в поле ввода.
     *
     * @param password      пароль пользователя
     * @param nextPageClass класс возвращаемой страницы
     * @param <T>           тип страницы
     * @return текущий экземпляр страницы
     */
       public <T extends BasePage> T enterPassword(String password,
Class<T> nextPageClass) {
        passwordInput.fill(password);
        return page(nextPageClass);
    }

    /**
     * Нажимает кнопку входа.
     *
     * @param nextPageClass класс возвращаемой страницы
     * @param <T>           тип страницы
     * @return экземпляр главной страницы
     */
         public <T extends BasePage> T clickLoginButton(Class<T>
nextPageClass) {
        submitButton.click();
        return page(nextPageClass);
    }
```

```
    /**
     * Проверяет отображение сообщения о неверном пароле.
     *
     * @return true если сообщение отображается, иначе false
     */
    public boolean checkIsIncorrectPasswordMessageDisplayed() {
        return incorrectPasswordText.isDisplayed();
    }

    /**
     * Проверяет отображение сообщения о неверном формате email.
     *
     * @return true если сообщение отображается, иначе false
     */
    public boolean checkIsIncorrectEmailMessageDisplayed() {
        return incorrectEmailFormatText.isDisplayed();
    }

    /**
     * Открывает страницу аутентификации.
     *
     * @return экземпляр страницы аутентификации
     */
    public static LoginPage open() {
        return new LoginPage();
    }
}
```

Название файла: MainPage.java

```
package com.example.pages;
```

```
import com.example.elements.Article.Article;

import com.example.elements.Button.Button;

import com.example.elements.MansoryContainer.MansoryContainer;

import com.example.elements.SearchHeader.SearchHeader;

import org.openqa.selenium.support.ui.ExpectedConditions;


import java.time.Duration;


import static com.codeborne.selenide.Selenide.Wait;


//Класс главной страницы, где уже есть картинки, наследуется от
BasePage


public class MainPage extends BasePage {

            private static final String MAIN_CONTAINER_XPATH =
"//*[@role='main']";

          private static final String MASONRY_CONTAINER_CLASS =
"masonryContainer";

    private static final String SEARCH_HEADER_ID = "HeaderContent";

     private static final String SETTINGS_BUTTON_ARIA_LABEL = "Другие
варианты";

          private static final String SETTINGS_ARTICLE_XPATH =
"//*[@id=\"VerticalNav-MoreOptions-Flyout-item-0\"]/div/a";


                  private    final    Button    undoActionButton    =
Button.byXPath("//*[@data-test-id=\"undo-action-btn\"]");

             private final MansoryContainer mansoryContainer =
MansoryContainer.byClass(MASONRY_CONTAINER_CLASS);

                 private    final    SearchHeader    searchHeader    =
SearchHeader.byId(SEARCH_HEADER_ID);

                private    final    Button    settingsModalButton    =
Button.byAriaLabel(SETTINGS_BUTTON_ARIA_LABEL);

                  private    final    Article    settingsArticle    =
Article.byXpath(SETTINGS_ARTICLE_XPATH);
```

```
/**
 * Конструктор главной страницы.
 * Инициализирует страницу с валидационным XPath.
 */
public MainPage() {
    super(MainPage.class, MAIN_CONTAINER_XPATH);
}

/**
 * Проверяет отображение главной страницы.
 *
 * @return true если страница отображается, иначе false
 */
public boolean isDisplayed() {
    return mansoryContainer.isDisplayed();
}

/**
 * Получает ссылку на статью первого изображения.
 *
 * @return URL статьи
 */
public String getHrefOfFirstImage() {
    return mansoryContainer.getHrefOfFirstImage();
}

/**
 * Открывает статью первого изображения.
 *
 * @return экземпляр страницы статьи
 */
```

```
    public <T extends BasePage> T clickOnFirstImage(Class<T> className)
{
        mansoryContainer.clickOnImage();
        return page(className);
    }

    /**
     * Активирует поле поиска.
     */
    public void openSearchBar() {
        searchHeader.openSearchBar();
    }

    /**
     * Вводит текст в поле поиска.
     *
     * @param searchText текст для поиска
     */
    public void fillSearchBar(String searchText) {
        searchHeader.fillSearchText(searchText);
    }

    /**
     * Выполняет поиск.
     */
    public void submitSearch() {
        searchHeader.pressEnter();
    }

    /**
     * Ожидает загрузку страницы по частичному совпадению URL.
```

```
     *
     * @param expectedUrlPart часть ожидаемого URL
     */
    public void waitForSearchPageLoad(String expectedUrlPart) {

Wait().withTimeout(Duration.ofSeconds(10)).until(ExpectedConditions.url
Contains(expectedUrlPart));
    }


    /**
     * Открывает страницу аккаунта.
     *
     * @return экземпляр страницы аккаунта
     */
         public <T extends BasePage> T clickAccountButton(Class<T>
className) {
        searchHeader.clickAccountButton();
        return page(className);
    }


    /**
     * Получает значение атрибута aria-label для n-го элемента.
     *
     * @param n порядковый номер элемента
     * @return значение атрибута aria-label
     */
    public String getNthAriaLabel(int n) {
        return mansoryContainer.getNthArticleAriaLabel(n);
    }


    /**
     * Открывает меню настроек.
```

```
     */
    public void clickSettingsModalButton() {
        settingsModalButton.click();
    }

    /**
     * Открывает страницу настроек.
     *
     * @return экземпляр страницы настроек
     */
    public <T extends BasePage> T clickSettings(Class<T> className) {
        settingsArticle.click();
        return page(className);
    }

    /**
     * Проверяет, появилась ли возможность отменить действие скрытия
     *
     * @return true или false соответственно
     */
    public boolean undoDisplayed() {
        return undoActionButton.isDisplayed();
    }
}
```

Название файла: PinPage.java

```
package com.example.pages;

import com.codeborne.selenide.SelenideElement;
import com.example.elements.Button.Button;
import com.example.elements.H2.H2;
import com.example.elements.Input.Input;
```

```
import com.example.elements.likebutton.LikeButton;

import org.openqa.selenium.support.ui.ExpectedConditions;


import java.io.File;

import java.time.Duration;


import static com.codeborne.selenide.Selenide.$x;

import static com.codeborne.selenide.Selenide.Wait;


public class PinPage extends BasePage {


    private final Button backButton = Button.byAriaLabel("Назад");

    private final Button moreOptionsButton = Button.byAriaLabel("Другие
действия");

                    private     final     Button     hidePinButton     =
Button.byXPath("//*[@data-test-id=\"pin-action-dropdown-hide\"]");

                private    final    Button    notInterestedButton    =
Button.byXPath("//*[@data-test-id=\"hide-reason\"][1]");

                    private     final     LikeButton     likeButton     =
LikeButton.byDataTestId("react-button");

                private    final    Button    downloadPinButton    =
Button.byXPath("//*[@data-test-id=\"pin-action-dropdown-download\"]");

    private final Button saveButton = Button.byAriaLabel("Сохранить");

     private final Button alreadySavedButton = Button.byAriaLabel("Пин
сохранен");

                private    final    Button    choosePhotoButton    =
Button.byAriaLabel("Выберите фото.");

                    private     final     Input     loadPhotoInput     =
Input.byDataTestId("photo-upload-input");

             private   final   SelenideElement   commentTextInput   =
$x("//*[@class='DraftEditor-editorContainer']/div");

                private    final    Button    sendCommentButton    =
Button.byAriaLabel("Опубликовать");

    private final H2 commentCount = H2.byId("comments-heading");
```

```
    /**
     * Конструктор страницы пина.
     * Инициализирует страницу с валидационным XPath.
     */
    public PinPage() {
                                                    super(PinPage.class,
"//*[@data-test-id=\"closeup-container\"]");
    }

    /**
     * Получает ссылку на эту страницу - изображение пина.
     *
     * @return URL изображения
     */
    public String getCurrentUrl() {
        return getDriver().getCurrentUrl();
    }

    /**
     * Возвращается на предыдущую страницу.
     *
     * @return экземпляр предыдущей страницы
     */
    public <T extends BasePage> T clickBackButton(Class<T> className) {
        backButton.click();
        return page(className);
    }

    /**
     * Ожидает загрузку страницы пина по частичному совпадению URL.
```

```
     *
     * @param expectedUrlPart часть ожидаемого URL
     */
    public void waitForPinPageLoad(String expectedUrlPart) {

Wait().withTimeout(Duration.ofSeconds(10)).until(ExpectedConditions.url
Contains(expectedUrlPart));
    }


    /**
     * Открывает меню дополнительных опций.
     */
    public void clickMoreOptionsButton() {
        moreOptionsButton.click();
    }


    /**
     * Скрывает текущий пин.
     */
    public void clickHidePinButton() {
        hidePinButton.click();
    }


    /**
     * Открывает окно загрузки изображения.
     */
    public void clickChoosePhotoButton() {
        choosePhotoButton.click();
    }


    /**
     * Отмечает пин как "Не интересно".
```

```
     */
      public <T extends BasePage> T clickNotInterestedButton(Class<T>
className) {
        notInterestedButton.click();
        return page(className);
    }

    /**
     * Ставит или снимает лайк с поста.
     */
    public void clickLikeButton() {
        likeButton.click();
    }

    /**
     * Получает данные SVG-пути для кнопки лайка.
     *
     * @return значение атрибута 'd'
     */
    public String getDLikeButton() {
        return likeButton.getPathD();
    }

    /**
     * Вводит текст комментария.
     *
     * @param text текст комментария
     */
    public void fillText(String text) {
        commentTextInput.sendKeys(text);
    }
```

```
/**
 * Отправляет комментарий.
 */
public void clickSendCommentButton() {
    sendCommentButton.click();
}

/**
 * Загружает фото.
 *
 * @param file файл для загрузки
 */
public void loadPhoto(File file) {
    loadPhotoInput.load(file);
}

/**
 * Скачивает текущий пин.
 */
public void clickDownloadPinButton() {
    downloadPinButton.click();
}

/**
 * Сохраняет текущий пин.
 */
public void clickSaveButton() {
    saveButton.click();
}
```

```
    /**
     * Проверяет, сохранен ли пин.
     *
     * @return true если пин сохранен, иначе false
     */
    public boolean isPostSaved() {
        return alreadySavedButton.isDisplayed();
    }

    /**
     * Возвращает кол-во комментариев.
     *
     * @return количество комментариев.
     */
    public String getCommentCount() {
        return commentCount.getText();
    }
}
```

Название файла SettingsPage.java

```
package com.example.pages;

import com.example.elements.Button.Button;
import com.example.elements.TextArea.TextArea;
import com.example.elements.Input.Input;

public class SettingsPage extends BasePage {
            private static final String PAGE_VALIDATION_XPATH =
"//*[@id=\"fullpage\"]";
    private static final String FIRST_NAME_INPUT_NAME = "first_name";
    private static final String LAST_NAME_INPUT_NAME = "last_name";
    private static final String ABOUT_TEXTAREA_ID = "about";
```

```
            private static final String SAVE_BUTTON_XPATH =
"//div[@data-test-id=\"done-button\"]/button";

           private static final String CANCEL_BUTTON_XPATH =
"//div[@data-test-id=\"cancel-button\"]/button";

    private static final String VALUE_ATTRIBUTE = "value";


    // Элементы страницы

                  private final Input firstNameInput =
Input.byName(FIRST_NAME_INPUT_NAME);

                   private final Input lastNameInput =
Input.byName(LAST_NAME_INPUT_NAME);

                 private final TextArea aboutTextArea =
TextArea.byId(ABOUT_TEXTAREA_ID);

                    private final Button saveButton =
Button.byXPath(SAVE_BUTTON_XPATH);

                   private final Button cancelButton =
Button.byXPath(CANCEL_BUTTON_XPATH);


    /**
     * Конструктор страницы настроек.
     * Инициализирует страницу с валидационным XPath.
     */
    public SettingsPage() {
        super(LoginPage.class, PAGE_VALIDATION_XPATH);
    }


    /**
     * Вводит имя пользователя.
     *
     * @param firstName имя пользователя
     */
    public void fillFirstName(String firstName) {
        firstNameInput.fill(firstName);
    }
```

```
/**
 * Вводит фамилию пользователя.
 *
 * @param lastName фамилия пользователя
 */
public void fillLastName(String lastName) {
    lastNameInput.fill(lastName);
}

/**
 * Вводит информацию "О себе".
 *
 * @param about информация о пользователе
 */
public void fillAbout(String about) {
    aboutTextArea.fill(about);
}

/**
 * Сохраняет изменения профиля.
 */
public void clickSaveButton() {
    saveButton.click();
}

/**
 * Отменяет изменения профиля.
 */
public void clickCancelButton() {
    cancelButton.click();
```

```
    }

    /**
     * Получает текущее значение имени пользователя.
     *
     * @return текущее имя
     */
    public String getFirstName() {
        return firstNameInput.getAttribute("value");
    }

    /**
     * Получает текущее значение фамилии пользователя.
     *
     * @return текущая фамилия
     */
    public String getLastName() {
        return lastNameInput.getAttribute("value");
    }

    /**
     * Получает текущую информацию "О себе".
     *
     * @return текущая информация "О себе"
     */
    public String getAbout() {
        return aboutTextArea.getText();
    }
}
```

Название файла: BaseTest.java

```
package com.example.tests;
```

```
import com.codeborne.selenide.Configuration;
import com.codeborne.selenide.FileDownloadMode;
import com.codeborne.selenide.Selenide;
import com.codeborne.selenide.logevents.SelenideLogger;
import com.example.pages.LoginPage;
import com.example.pages.MainPage;
import io.qameta.allure.selenide.AllureSelenide;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.openqa.selenium.chrome.ChromeOptions;

import static com.codeborne.selenide.Selenide.open;

/**
 * Базовый класс для UI тестов.
 * Содержит общие настройки для всех тестовых классов.
 */
public class BaseTest {

    /**
     * Конфигурация тестового окружения перед каждым тестом.
     * Устанавливает параметры браузера, таймауты и базовые настройки.
     */
    @BeforeEach
    public void setUp() {
        configureBrowser();
        configureLogging();
        open("/");
    }
```

```
/**
 * Закрывает веб-драйвер после каждого теста.
 */
@AfterEach
public void tearDown() {
    Selenide.closeWebDriver();
}

/**
 * Выполняет аутентификацию пользователя.
 *
 * @param username логин пользователя
 * @param password пароль пользователя
 */
protected MainPage auth(String login, String password) {
    LoginPage loginPage = LoginPage.open();
    loginPage = loginPage.openLoginModal(LoginPage.class);
    loginPage.enterLogin(login, LoginPage.class);
    loginPage.enterPassword(password, LoginPage.class);
    return loginPage.clickLoginButton(MainPage.class);
}

/**
 * Настраивает параметры браузера.
 */
private void configureBrowser() {
    Configuration.browser = "chrome";
    Configuration.browserSize = "1280x720";
    Configuration.pageLoadStrategy = "eager";
    Configuration.timeout = 100000;
    Configuration.pageLoadTimeout = 60000;
```

```
        Configuration.baseUrl = "https://ru.pinterest.com";

        ChromeOptions options = new ChromeOptions();
        options.addArguments(
                "--disable-gpu",
                "--no-sandbox",
                "--disable-dev-shm-usage",
                "--remote-allow-origins=*",
                "--disable-blink-features=AutomationControlled");
        Configuration.browserCapabilities = options;
        Configuration.downloadsFolder = "./downloads";
    }

    /**
     * Настраивает систему логирования.
     */
    private void configureLogging() {
        SelenideLogger.addListener(
                "AllureSelenide",
                new AllureSelenide()
                        .screenshots(true)
                        .savePageSource(true));
    }
}
```

Название файла CommentTest.page

```
package com.example.tests.commentTest;

import com.example.pages.MainPage;
import com.example.pages.PinPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
```

```
import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static com.codeborne.selenide.Selenide.Wait;


import java.io.File;

import java.time.Duration;


import static org.junit.jupiter.api.Assertions.*;


/**

 * Тестовый класс для проверки функциональности комментирования пинов.

  * Проверяет возможность добавления комментария с текстом и изображением.

 */

public class CommentTest extends BaseTest {

    private static final String COMMENT_TEXT = "Классный пост!";

                private static final String IMAGE_PATH = "./testAssets/commentTest.png";

          private static final Duration COMMENT_UPDATE_TIMEOUT = Duration.ofSeconds(10);


    private String login;

    private String password;


    /**

     * Инициализирует тестовые данные перед каждым тестом.

     * Загружает учетные данные из .env файла.

     */

    @BeforeEach

    protected void initTestData() {

        Dotenv dotenv = Dotenv.load();

        login = dotenv.get("USER_LOGIN");
```

```
        password = dotenv.get("USER_PASSWORD");
    }

    /**
         * Проверяет возможность добавления комментария с текстом и изображением.
     */
    @Test
    void shouldAddCommentWithTextAndImage() {
                            System.out.println("[INFO]   Начало   теста: shouldAddCommentWithTextAndImage");

        // Аутентификация и открытие главной страницы
        System.out.println("[ACTION] Аутентификация пользователя");
        MainPage mainPage = auth(login, password);
        System.out.println("[SUCCESS] Пользователь аутентифицирован");

        // Открытие первого пина
        System.out.println("[ACTION] Открытие первого пина");
        PinPage pinPage = mainPage.clickOnFirstImage(PinPage.class);
        System.out.println("[SUCCESS] Пин открыт");

        // Ввод текста комментария
            System.out.println("[ACTION] Ввод текста комментария: " + COMMENT_TEXT);
        pinPage.fillText(COMMENT_TEXT);
        System.out.println("[SUCCESS] Текст комментария введён");

        // Загрузка изображения
        System.out.println("[ACTION] Нажатие на кнопку выбора фото");
        pinPage.clickChoosePhotoButton();
         System.out.println("[ACTION] Загрузка изображения из пути: " + IMAGE_PATH);
```

```
        pinPage.loadPhoto(new File(IMAGE_PATH));

        System.out.println("[SUCCESS] Изображение загружено");


        // Отправка комментария

            System.out.println("[ACTION] Получение текущего количества
комментариев");

        String oldCount = pinPage.getCommentCount();

         System.out.println("[INFO] Старое количество комментариев: " +
oldCount);


                System.out.println("[ACTION] Нажатие кнопки отправки
комментария");

        pinPage.clickSendCommentButton();


             System.out.println("[WAIT] Ожидание обновления количества
комментариев...");

        Wait().withTimeout(COMMENT_UPDATE_TIMEOUT)

                                                          .until(d    ->
!pinPage.getCommentCount().equals(oldCount));

                System.out.println("[SUCCESS] Количество комментариев
обновилось");


        String newCount = pinPage.getCommentCount();

          System.out.println("[INFO] Новое количество комментариев: " +
newCount);


        assertNotEquals(oldCount, newCount);

         System.out.println("[ASSERTION PASSED] Количество комментариев
изменилось");


                        System.out.println("[INFO]   Завершение   теста:
shouldAddCommentWithTextAndImage");

    }

}
```

Название файла DownloadTest.java

```
package com.example.tests.DownloadTest;

import com.example.pages.MainPage;
import com.example.pages.PinPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
import org.junit.jupiter.api.Test;

import java.io.File;
import java.time.Duration;

import static com.codeborne.selenide.Selenide.Wait;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class DownloadTest extends BaseTest {
    private String login;
    private String password;
    private final String pathToDownloadDir = "./downloads";

    @Test
    public void checkDownload() {
        System.out.println("[INFO] Инициализация тестовых данных...");
        initTestData();

          System.out.println("[ACTION] Проверка количества файлов .jpeg
до скачивания...");
        int startJpegCount = checkNumJpegFiles(true);
           System.out.println("[INFO] Файлов .jpeg до скачивания: " +
startJpegCount);

        System.out.println("[ACTION] Авторизация пользователя...");
        MainPage mainPage = auth(login, password);
```

```
        System.out.println("[ACTION] Переход к первому пину...");
        PinPage pinPage = mainPage.clickOnFirstImage(PinPage.class);

                System.out.println("[ACTION] Открытие дополнительных опций...");
        pinPage.clickMoreOptionsButton();

        System.out.println("[ACTION] Начало загрузки пина...");
        pinPage.clickDownloadPinButton();

            System.out.println("[WAIT] Ожидание обновления количества картинок...");
        Wait().withTimeout(Duration.ofSeconds(5))
                                .until(d -> checkNumJpegFiles(false) != startJpegCount);
        System.out.println("[SUCCESS] Количество картинок обновилось");

          System.out.println("[ACTION] Проверка количества файлов .jpeg после скачивания...");
        int endJpegCount = checkNumJpegFiles(true);
          System.out.println("[INFO] Файлов .jpeg после скачивания: " + endJpegCount);

         System.out.println("[ASSERTION] Проверка, что количество файлов увеличилось");
           assertTrue(endJpegCount > startJpegCount, "[FAIL] Количество файлов после скачивания должно быть больше, чем до");

            System.out.println("[SUCCESS] Тест checkDownload успешно пройден");
    }

    /**
     * Инициализация тестовых данных.
```

```
     * Загружает учетные данные из .env файла.
     */
    protected void initTestData() {
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
        System.out.println("Логин и пароль загружены из .env");
    }

    private int checkNumJpegFiles(boolean log) {
        File downloadDir = new File(pathToDownloadDir);
        if (!downloadDir.exists()) {
                System.out.println("Папка для загрузок не найдена: " +
pathToDownloadDir);
            return 0;
        }
        int count = countJpegFilesRecursively(downloadDir);
           if (log) System.out.println("Найдено файлов .jpeg (включая
подкаталоги): " + count);
        return count;
    }

    /**
      * Рекурсивно считает все файлы с расширением .jpeg в каталоге и
подкаталогах.
     */
    private int countJpegFilesRecursively(File directory) {
        int count = 0;
        File[] filesAndDirs = directory.listFiles();
        if (filesAndDirs != null) {
            for (File file : filesAndDirs) {
                if (file.isDirectory()) {
```

```
                    count += countJpegFilesRecursively(file);
                } else if (file.isFile() &&
file.getName().toLowerCase().endsWith(".jpeg")) {
                    count++;
                }
            }
        }
        return count;
    }
}
```

Название файла : EditProfileTest.java

```
package com.example.tests.editProfileTest;

import com.example.pages.MainPage;
import com.example.pages.SettingsPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;

/**
 * Тестовый класс для проверки функциональности редактирования профиля
пользователя.
 * Проверяет возможность изменения имени, фамилии и информации "О
себе".
 */
public class EditProfileTest extends BaseTest {
    // Константы для тестовых данных
```

```
    private static final String NAME_SUFFIX = "I";
    private static final String SURNAME_SUFFIX = "g";
    private static final String ABOUT_TEXT = "testing";

    private String login;
    private String password;
    private String originalFirstName;
    private String originalLastName;
    private String originalAbout;

    /**
     * Инициализирует тестовые данные перед каждым тестом.
     * Загружает учетные данные из .env файла.
     */
    @BeforeEach
    protected void initTestData() {
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
    }

    /**
     * Проверяет возможность редактирования профиля пользователя.
     */
    @Test
    public void shouldSuccessfullyEditUserProfile() {
                            System.out.println("[INFO]   Начало   теста:
shouldSuccessfullyEditUserProfile");

        // Аутентификация и открытие главной страницы
        System.out.println("[ACTION] Аутентификация пользователя");
```

```
        MainPage mainPage = auth(login, password);
        System.out.println("[SUCCESS] Пользователь аутентифицирован");

        // Открытие страницы настроек
                System.out.println("[ACTION] Открытие модального окна
настроек");
        mainPage.clickSettingsModalButton();
        System.out.println("[ACTION] Переход на страницу настроек");
                                        SettingsPage settingsPage =
mainPage.clickSettings(SettingsPage.class);
        System.out.println("[SUCCESS] Страница настроек открыта");

        // Сохранение исходных значений
             System.out.println("[ACTION] Сохранение исходных значений
профиля");
        saveOriginalProfileValues(settingsPage);
                        System.out.println("[INFO] Исходное имя: " +
originalFirstName);
                    System.out.println("[INFO] Исходная фамилия: " +
originalLastName);
                  System.out.println("[INFO] Исходное 'О себе': " +
originalAbout);

        try {
            // Подготовка новых значений
            String newFirstName = originalFirstName + NAME_SUFFIX;
            String newLastName = originalLastName + SURNAME_SUFFIX;
            System.out.println("[INFO] Новое имя: " + newFirstName);
            System.out.println("[INFO] Новая фамилия: " + newLastName);
            System.out.println("[INFO] Новое 'О себе': " + ABOUT_TEXT);

            // Изменение данных профиля
            System.out.println("[ACTION] Изменение данных профиля");
```

```
                        updateProfileValues(settingsPage,  newFirstName,
newLastName, ABOUT_TEXT);

                System.out.println("[SUCCESS] Данные профиля обновлены и
сохранены");


            // Обновление страницы для проверки сохранения

               System.out.println("[ACTION] Обновление страницы настроек
для валидации");

            settingsPage = settingsPage.refresh(SettingsPage.class);


            // Проверка обновленных значений

              System.out.println("[ACTION] Проверка обновлённых значений
профиля");

                        verifyProfileValues(settingsPage,  newFirstName,
newLastName, ABOUT_TEXT);

                 System.out.println("[ASSERTION PASSED] Значения профиля
успешно обновлены и проверены");

        } finally {

            // Восстановление исходных значений

                   System.out.println("[ACTION] Восстановление исходных
значений профиля");

            restoreOriginalProfileValues(settingsPage);

                  System.out.println("[SUCCESS] Профиль восстановлен к
первоначальным значениям");

        }


                        System.out.println("[INFO]   Завершение   теста:
shouldSuccessfullyEditUserProfile");

    }


    /**

     * Сохраняет исходные значения профиля.

     *

     * @param settingsPage экземпляр страницы настроек

     */
```

```
    private void saveOriginalProfileValues(SettingsPage settingsPage) {
        originalFirstName = settingsPage.getFirstName();
        originalLastName = settingsPage.getLastName();
        originalAbout = settingsPage.getAbout();
    }

    /**
     * Обновляет значения профиля.
     *
     * @param settingsPage экземпляр страницы настроек
     * @param firstName    новое имя
     * @param lastName     новая фамилия
     * @param about        новая информация "О себе"
     */
     private void updateProfileValues(SettingsPage settingsPage, String
firstName,
                                      String lastName, String about) {
        settingsPage.fillFirstName(firstName);
        settingsPage.fillLastName(lastName);
        settingsPage.fillAbout(about);
        settingsPage.clickSaveButton();
    }

    /**
     * Проверяет значения профиля.
     *
     * @param settingsPage      экземпляр страницы настроек
     * @param expectedFirstName ожидаемое имя
     * @param expectedLastName  ожидаемая фамилия
     * @param expectedAbout     ожидаемая информация "О себе"
     */
```

```
     private void verifyProfileValues(SettingsPage settingsPage, String
expectedFirstName,
                                          String expectedLastName, String
expectedAbout) {
        String actualFirstName = settingsPage.getFirstName();
        String actualLastName = settingsPage.getLastName();
        String actualAbout = settingsPage.getAbout();


        assertAll("Проверка обновленных значений профиля",
                  () -> assertEquals(expectedFirstName, actualFirstName,
"Имя должно соответствовать"),
                    () -> assertEquals(expectedLastName, actualLastName,
"Фамилия должна соответствовать"),
                          () -> assertEquals(expectedAbout, actualAbout,
"Информация 'О себе' должна соответствовать")
        );
    }


    /**
     * Восстанавливает исходные значения профиля.
     *
     * @param settingsPage экземпляр страницы настроек
     */
             private   void   restoreOriginalProfileValues(SettingsPage
settingsPage) {
        settingsPage.fillFirstName(originalFirstName);
        settingsPage.fillLastName(originalLastName);
        settingsPage.fillAbout(originalAbout);
        settingsPage.clickSaveButton();
    }
}
```

Название файла HideTest.java

```
package com.example.tests.HideTest;
```

```
import com.example.pages.MainPage;
import com.example.pages.PinPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

/**
 * Тестовый класс для проверки функциональности скрытия пинов.
 * Проверяет возможность скрытия пина и отмены этого действия.
 */
public class HideTest extends BaseTest {
    private String login;
    private String password;

    /**
     * Проверяет функциональность скрытия пина и отмены этого действия.
     */
    @Test
    public void checkHidePost() {
        System.out.println("[INFO] Начало теста: checkHidePost");

        // Инициализация данных
        System.out.println("[ACTION] Инициализация учетных данных");
        initTestData();
         System.out.println("[SUCCESS] Данные инициализированы: login="
+ login);

        // Аутентификация
```

```
        System.out.println("[ACTION] Аутентификация пользователя");
        MainPage mainPage = auth(login, password);
        System.out.println("[SUCCESS] Аутентификация прошла успешно");

        // Получение ссылки на первый пин
        System.out.println("[ACTION] Получение href первого пина");
        String hrefPreview = mainPage.getHrefOfFirstImage();
        System.out.println("[INFO] href пина: " + hrefPreview);

        // Переход к пину
        System.out.println("[ACTION] Клик на первый пин");
        PinPage pinPage = mainPage.clickOnFirstImage(PinPage.class);
        System.out.println("[ACTION] Ожидание загрузки страницы пина");
        pinPage.waitForPinPageLoad(hrefPreview);
        System.out.println("[SUCCESS] Страница пина загружена");

        // Скрытие пина
        System.out.println("[ACTION] Нажатие кнопки 'Еще опции'");
        pinPage.clickMoreOptionsButton();
        System.out.println("[ACTION] Нажатие кнопки 'Скрыть пин'");
        pinPage.clickHidePinButton();

        // Подтверждение "Не интересно"
            System.out.println("[ACTION] Подтверждение скрытия: 'Не
интересно'");
        mainPage = pinPage.clickNotInterestedButton(MainPage.class);
        System.out.println("[SUCCESS] Возврат на главную страницу");

        // Проверка отображения кнопки "Отменить"
           System.out.println("[ASSERTION] Проверка отображения кнопки
'Отменить'");
        assertTrue(mainPage.undoDisplayed());
```

```
            System.out.println("[ASSERTION PASSED] Кнопка 'Отменить'
отображается");


        System.out.println("[INFO] Завершение теста: checkHidePost");
    }


    /**
     * Инициализация тестовых данных.
     * Загружает учетные данные из .env файла.
     */
    protected void initTestData() {
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
    }



}
```

Название файла LikeTest.java

```
package com.example.tests.likeTest;

import com.example.pages.MainPage;
import com.example.pages.PinPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotEquals;
```

```
/**
 * Тестовый класс для проверки функциональности лайков.
 * Проверяет постановку и снятие лайка, а также сохранение состояния после обновления страницы.
 */
public class LikeTest extends BaseTest {
    private String login;
    private String password;

    /**
     * Инициализирует тестовые данные перед каждым тестом.
     * Загружает учетные данные из .env файла.
     */
    @BeforeEach
    protected void initTestData() {
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
    }

    /**
     * Проверяет функциональность лайка:
     * 1. Постановка лайка и проверка изменения состояния
     * 2. Сохранение состояния после обновления страницы
     * 3. Снятие лайка и проверка возврата в исходное состояние
     */
    @Test
    public void shouldToggleLikeAndPersistStateAfterRefresh() {
                            System.out.println("[INFO] Начало теста: shouldToggleLikeAndPersistStateAfterRefresh");

        // Аутентификация и открытие главной страницы
```

```
        System.out.println("[ACTION] Аутентификация пользователя");
        MainPage mainPage = auth(login, password);
        System.out.println("[SUCCESS] Аутентификация прошла успешно");

        // Открытие первого пина
        System.out.println("[ACTION] Открытие первого пина");
        PinPage pinPage = mainPage.clickOnFirstImage(PinPage.class);
        System.out.println("[SUCCESS] Пин открыт");

        // Получение исходного состояния кнопки лайка
            System.out.println("[ACTION] Получение исходного состояния
лайка");
        String initialPathData = pinPage.getDLikeButton();
            System.out.println("[INFO] Исходное состояние кнопки: " +
initialPathData);

        // Постановка лайка
        System.out.println("[ACTION] Клик по кнопке лайка");
        pinPage.clickLikeButton();
        String likedPathData = pinPage.getDLikeButton();
              System.out.println("[INFO] Состояние после лайка: " +
likedPathData);

        // Проверка изменения состояния
              System.out.println("[ASSERTION] Проверка, что состояние
изменилось после лайка");
          assertNotEquals(initialPathData, likedPathData, "Данные пути
должны измениться после постановки лайка");
            System.out.println("[ASSERTION PASSED] Состояние изменилось
после лайка");

        // Обновление страницы
        System.out.println("[ACTION] Обновление страницы");
        pinPage = pinPage.refresh(PinPage.class);
```

```
        String refreshedPathData = pinPage.getDLikeButton();

         System.out.println("[INFO] Состояние после обновления страницы: " + refreshedPathData);



        // Проверка сохранения состояния

          System.out.println("[ASSERTION] Проверка, что лайк сохранился после обновления страницы");

         assertEquals(likedPathData, refreshedPathData, "Состояние лайка должно сохраниться после обновления страницы");

           System.out.println("[ASSERTION PASSED] Лайк сохранился после обновления");



        // Снятие лайка

        System.out.println("[ACTION] Снятие лайка");

        pinPage.clickLikeButton();

        String unlikedPathData = pinPage.getDLikeButton();

          System.out.println("[INFO] Состояние после снятия лайка: " + unlikedPathData);



        // Проверка возврата в исходное состояние

          System.out.println("[ASSERTION] Проверка возврата к исходному состоянию после снятия лайка");

           assertEquals(initialPathData, unlikedPathData, "Данные пути должны вернуться к исходному состоянию после снятия лайка");

           System.out.println("[ASSERTION PASSED] Состояние вернулось к исходному после снятия лайка");



                        System.out.println("[INFO] Завершение теста: shouldToggleLikeAndPersistStateAfterRefresh");

    }

}
```

Название файла LoginTest.java

```
package com.example.tests.LoginTest;


import com.example.pages.BasePage;
```

```
import com.example.pages.LoginPage;
import com.example.pages.MainPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

/**
 * Тестовый класс для проверки функциональности аутентификации.
 * Проверяет сценарии входа с неверными и верными учетными данными.
 */
public class LoginTest extends BaseTest {
    private static final String FAKE_PASSWORD = "aboba";
    private static final String INCORRECT_FORMAT_EMAIL = "aaaa";
    private String validLogin;
    private String validPassword;

    @Test
    public void checkAuth() {
        System.out.println("[INFO] Начало теста: checkAuth");

        // Инициализация данных
        System.out.println("[ACTION] Инициализация тестовых данных");
        initTestData();
         System.out.println("[SUCCESS] Данные инициализированы: login="
+ validLogin);

        // Открытие страницы входа
        System.out.println("[ACTION] Открытие страницы логина");
        LoginPage.open().openLoginModal(LoginPage.class);
```

```
        System.out.println("[SUCCESS] Модальное окно логина открыто");

        // Проверка ошибки при некорректном email
        shouldShowErrorWhenInvalidFormatEmail();

        // Проверка ошибки при неправильном пароле
        shouldShowErrorWhenInvalidPassword();

        // Проверка успешного входа
        shouldSuccessfullyLoginWithValidCredentials();

        System.out.println("[INFO] Завершение теста: checkAuth");
    }

    /**
     * Проверяет сценарий входа с неверным паролем.
     * Ожидает появление сообщения об ошибке.
     */
    private void shouldShowErrorWhenInvalidPassword() {
        System.out.println("[ACTION] Попытка входа с правильным логином и неправильным паролем");
        LoginPage loginPage = performLogin(validLogin, FAKE_PASSWORD, LoginPage.class);
        boolean isErrorShown = loginPage.checkIsIncorrectPasswordMessageDisplayed();
        System.out.println("[ASSERTION] Отображается сообщение об ошибке пароля: " + isErrorShown);
        assertTrue(isErrorShown);
        System.out.println("[ASSERTION PASSED] Ошибка при неверном пароле отображается корректно");
    }

    /**
```

```
     * Проверяет сценарий входа с неверным email неверного формата.
     * Ожидает появление сообщения об ошибке.
     */
    private void shouldShowErrorWhenInvalidFormatEmail() {
                System.out.println("[ACTION] Попытка входа с email в неправильном формате");
            LoginPage loginPage = performLogin(INCORRECT_FORMAT_EMAIL, FAKE_PASSWORD, LoginPage.class);
                                            boolean isErrorShown = loginPage.checkIsIncorrectEmailMessageDisplayed();
             System.out.println("[ASSERTION] Отображается сообщение об ошибке формата email: " + isErrorShown);
        assertTrue(isErrorShown);
         System.out.println("[ASSERTION PASSED] Ошибка при неправильном формате email отображается корректно");
    }

    /**
     * Проверяет сценарий входа с верными параметрами входа.
     * Ожидает выполнение аутентификации и прогрузку главной страницы.
     */
    private void shouldSuccessfullyLoginWithValidCredentials() {
         System.out.println("[ACTION] Попытка входа с валидными учетными данными");
           MainPage mainPage = performLogin(validLogin, validPassword, MainPage.class);
        boolean isMainPageVisible = mainPage.isDisplayed();
         System.out.println("[ASSERTION] Главная страница отображается: " + isMainPageVisible);
        assertTrue(isMainPageVisible);
               System.out.println("[ASSERTION PASSED] Успешный вход с валидными учетными данными");
    }

    /**
```

```
     * Инициализация тестовых данных.
     * Загружает учетные данные из .env файла.
     */
    protected void initTestData() {
        Dotenv dotenv = Dotenv.load();
        validLogin = dotenv.get("USER_LOGIN");
        validPassword = dotenv.get("USER_PASSWORD");
    }

    /**
     * Выполняет полный процесс аутентификации.
     *
     * @param login      логин пользователя
     * @param password      пароль пользователя
     * @param nextPageClass класс страницы
     * @return экземпляр класса страницы
     */
     private <T extends BasePage> T performLogin(String login, String
password, Class<T> nextPageClass) {
                            return   LoginPage.open().enterLogin(login,
LoginPage.class).enterPassword(password, LoginPage.class)
                .clickLoginButton(nextPageClass);
    }

}
```

Название файла: OpenClosePostTest.java

```
package com.example.tests.OpenClosePostTest;

import com.example.pages.MainPage;
import com.example.pages.PinPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
```

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

/**
 * Тестовый класс для проверки функциональности открытия и закрытия постов.
 * Проверяет корректность перехода на страницу поста и возврата на главную страницу.
 */
public class OpenClosePostTest extends BaseTest {
    private String login;
    private String password;

    /**
     * Инициализирует тестовые данные перед каждым тестом.
     * Загружает учетные данные из .env файла.
     */
    @BeforeEach
    protected void initTestData() {
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
    }

    /**
     * Проверяет функциональность открытия и закрытия поста:
     * 1. Открытие поста и проверка URL
     * 2. Возврат на главную страницу и проверка отображения
     */
    @Test
```

```
    public void shouldOpenPostAndReturnToMainPage() {

                            System.out.println("[INFO]   Начало   теста:
shouldOpenPostAndReturnToMainPage");


        // Аутентификация и открытие главной страницы

        System.out.println("[ACTION] Выполняется аутентификация");

        MainPage mainPage = auth(login, password);

           System.out.println("[SUCCESS] Аутентификация прошла успешно,
главная страница загружена");


        // Получение URL первого поста

         System.out.println("[ACTION] Получение href первого изображения
(поста)");

        String firstPostUrl = mainPage.getHrefOfFirstImage();

                 System.out.println("[INFO]  Получен  href  поста:  "  +
firstPostUrl);


        // Открытие страницы поста

        System.out.println("[ACTION] Переход к первому посту");

        PinPage pinPage = mainPage.clickOnFirstImage(PinPage.class);

        pinPage.waitForPinPageLoad(firstPostUrl);

        System.out.println("[SUCCESS] Страница поста загружена");


        // Проверка URL страницы поста

             System.out.println("[ASSERTION]  Проверка,  что  текущий  URL
содержит href поста");

        String currentUrl = pinPage.getCurrentUrl();

        boolean urlMatches = currentUrl.contains(firstPostUrl);

        System.out.println("[INFO] currentUrl = " + currentUrl);

         assertTrue(urlMatches, "URL страницы поста должен содержать URL
превью");

              System.out.println("[ASSERTION  PASSED]  URL  содержит  href
поста");
```

```
        // Возврат на главную страницу

          System.out.println("[ACTION] Нажатие кнопки назад (возврат на
главную страницу)");

        mainPage = pinPage.clickBackButton(MainPage.class);


        // Проверка отображения главной страницы

         System.out.println("[ASSERTION] Проверка, что главная страница
отображается");

        boolean isMainPageVisible = mainPage.isDisplayed();

               assertTrue(isMainPageVisible, "Главная страница должна
отображаться после возврата");

              System.out.println("[ASSERTION PASSED] Главная страница
отображается");


                        System.out.println("[INFO] Завершение теста:
shouldOpenPostAndReturnToMainPage");

    }

}
```

Название файла: SavePostTest.java

```
package com.example.tests.savePostTest;


import com.example.pages.MainPage;

import com.example.pages.PinPage;

import com.example.tests.BaseTest;

import io.github.cdimascio.dotenv.Dotenv;

import org.junit.jupiter.api.Test;


import static org.junit.jupiter.api.Assertions.assertTrue;


/**

 * Тестовый класс для проверки функциональности сохранения поста.

 */

public class SavePostTest extends BaseTest {
```

```
    private String login;
    private String password;

    @Test
    public void savePost() {
        System.out.println("[INFO] Начало теста: savePost");

        initTestData();

         System.out.println("[ACTION] Аутентификация и открытие главной страницы");
        MainPage mainPage = auth(login, password);
        System.out.println("[SUCCESS] Главная страница загружена");

        System.out.println("[ACTION] Переход к первому посту");
        PinPage pinPage = mainPage.clickOnFirstImage(PinPage.class);
        System.out.println("[SUCCESS] Страница поста загружена");

        System.out.println("[ACTION] Нажатие кнопки сохранения поста");
        pinPage.clickSaveButton();

            System.out.println("[ASSERTION] Проверка, что пост успешно сохранён");
        assertTrue(pinPage.isPostSaved(), "Пост должен быть сохранён");
        System.out.println("[ASSERTION PASSED] Пост сохранён");

        System.out.println("[INFO] Завершение теста: savePost");
    }

    /**
     * Инициализация тестовых данных.
     * Загружает учетные данные из .env файла.
```

```
     */
    protected void initTestData() {
        System.out.println("[INIT] Загрузка данных из .env");
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
        System.out.println("[INIT DONE] Данные загружены");
    }
}
```

Название файла: SearchTest.java

```
package com.example.tests.SearchTest;

import com.example.pages.MainPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
import org.junit.jupiter.api.Test;

import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertTrue;

/**
 * Тестовый класс для проверки функциональности поиска.
 * Проверяет поиск по запросу.
 */
public class SearchTest extends BaseTest {
    private static final String SEARCH_STR = "Милые котики";
    private static final List<String> SEARCH_KEYWORDS = Arrays.asList(
```

```
            "cat", "kit", "gat", "кот", "кош"
    );
    private String login;
    private String password;

    @Test
    public void checkSearch() {
        System.out.println("[INFO] Начало теста: checkSearch");

        initTestData();

        System.out.println("[ACTION] Аутентификация пользователя");
        MainPage mainPage = auth(login, password);
        System.out.println("[SUCCESS] Успешный вход в систему");

            System.out.println("[ACTION] Открытие и заполнение строки
поиска");
        mainPage.openSearchBar();
        mainPage.fillSearchBar(SEARCH_STR);
        mainPage.submitSearch();

           String encodedNoResultsTerm = URLEncoder.encode(SEARCH_STR,
StandardCharsets.UTF_8).replace("+", "%20");
        String expectedUrlPart = "?q=" + encodedNoResultsTerm;
         System.out.println("[WAIT] Ожидание загрузки страницы поиска с
URL частью: " + expectedUrlPart);
        mainPage.waitForSearchPageLoad(expectedUrlPart);

            System.out.println("[INFO] Сбор aria-label для первых 10
результатов");
        StringBuilder ariaLabels = new StringBuilder();
        for (int i = 1; i <= 10; i++) {
            String ariaLabel = mainPage.getNthAriaLabel(i);
```

```
            System.out.printf("[ARIA] #%d: %s%n", i, ariaLabel);
            ariaLabels.append(ariaLabel);
        }

        System.out.println("[ASSERTION] Проверка, что хотя бы один ключ найден в результатах");
                assertContainsAtLeastOneKeyword(ariaLabels.toString(), SEARCH_KEYWORDS);

            System.out.println("[SUCCESS] Тест checkSearch завершён успешно");
    }

    /**
     * Проверяет что в строке содержится хотя бы один кейворд.
     */
        private void assertContainsAtLeastOneKeyword(String text, List<String> keywords) {
        boolean found = keywords.stream()
                                                .anyMatch(keyword -> text.toLowerCase().contains(keyword.toLowerCase()));

        assertTrue(found, String.format(
                "[ERROR] Ни один из ожидаемых ключей (%s) не найден. Актуальный текст: %s",
                String.join(", ", keywords),
                text
        ));
    }

    /**
     * Инициализация тестовых данных.
     * Загружает учетные данные из .env файла.
     */
```

```
    protected void initTestData() {
            System.out.println("[INIT] Загрузка данных пользователя из
.env");
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
        System.out.println("[INIT DONE] Логин и пароль загружены");
    }
}
```

Название файла: SortingTest.java

```
package com.example.tests.SortingTest;

import com.example.pages.BoardsPage;
import com.example.pages.MainPage;
import com.example.tests.BaseTest;
import io.github.cdimascio.dotenv.Dotenv;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

/**
 * Тестовый класс для проверки функциональности сортировки досок.
 * Проверяет сортировку по нажатию.
 */
public class SortingTest extends BaseTest {
    private String login;
    private String password;

    @Test
    public void checkSort() {
```

```
        System.out.println("[INFO] Начало теста: checkSort");


        initTestData();


        System.out.println("[ACTION] Вход в систему");

        MainPage mainPage = auth(login, password);

        System.out.println("[SUCCESS] Вход выполнен");


        System.out.println("[ACTION] Переход на страницу досок");

                                        BoardsPage     boardsPage     =
mainPage.clickAccountButton(BoardsPage.class);


        // Сортировка по алфавиту

        System.out.println("[ACTION] Сортировка досок по алфавиту");

        boardsPage.openSortingOptions();

        boardsPage.sortAlphabetically();


        String previewName = boardsPage.getFirstBoardName();

            System.out.printf("[ASSERTION] Проверка: ожидаем 'Котята',
получили '%s'%n", previewName);

             assertEquals("Котята", previewName, "После сортировки по
алфавиту первой должна быть доска 'Котята'");


        // Сортировка по дате добавления

               System.out.println("[ACTION]  Сортировка  досок  по  дате
последнего добавления");

        boardsPage.openSortingOptions();

        boardsPage.sortByLastAdded();


        previewName = boardsPage.getFirstBoardName();

            System.out.printf("[ASSERTION] Проверка: ожидаем 'Тестовая
доска', получили '%s'%n", previewName);

          assertEquals("Тестовая доска", previewName, "После сортировки
по последнему добавлению первой должна быть 'Тестовая доска'");
```

```
                System.out.println("[SUCCESS] Тест checkSort завершён
успешно");
    }

    /**
     * Инициализация тестовых данных.
     * Загружает учетные данные из .env файла.
     */
    protected void initTestData() {
        System.out.println("[INIT] Загрузка данных из .env");
        Dotenv dotenv = Dotenv.load();
        login = dotenv.get("USER_LOGIN");
        password = dotenv.get("USER_PASSWORD");
        System.out.println("[INIT DONE] Данные загружены");
    }
}
```