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

ОТЧЕТ

По «UI-тестирование» практике

Тема: Тестирование СТД Петрович

Шуменков А.П.

Студенты гр. 3341 _____ Анисимов Д.А.

Руководитель _____ Шевелева А.М

Санкт-Петербург
2025

ЗАДАНИЕ

НА «UI-ТЕСТИРОВАНИЕ» ПРАКТИКУ

Студенты: Анисимов Д.А. Шуменков А.П.

Группа: 3341

Тема практики: Тестирование СТД Петрович

Задание на практику:

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

Сроки прохождения практики: 25.06.2025 – 08.07.2025

Дата сдачи отчета: 07.07.2025

Дата защиты отчета: 07.07.2025

		Анисимов Д.А.
Студенты	_____	Шуменков А.П.
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 три 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 three 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	5
1.	Первый раздел	6
1.1.	Чеклист	6
2.	Второй раздел	8
2.1.	Описание классов и методы	8
2.2.	UML-диаграмма	0
3.	Третий раздел	0
3.1.	Тестирование одного теста	0
3.2.	Успешная обработка всех тестов	0
	Заключение	0
	Список использованных источников	0
	Приложение А. Название приложения	0

ВВЕДЕНИЕ

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

Задачи:

1. Написать 10 автоматизированных тестов для выбранной системы.
2. Использовать технологии: Java, Selenide (Selenium), JUnit 5, Maven, логирование.
3. Организовать работу в GitHub (репозиторий на группу из 3 человек с распределением задач в README.md).
4. Оформить чеклист в виде таблицы с описанием тестов.

Для тестирования был выбран сайт СТД Петрович (https://petrovich.ru/). Выбор обусловлен тем, что это популярная и функционально насыщенная торговая площадка с широким ассортиментом товаров и развитой системой поиска, фильтрации и навигации. Такие особенности позволяют протестировать разнообразные сценарии взаимодействия пользователя с веб-интерфейсом, включая авторизацию, поиск, фильтрацию, добавление товаров в корзину и сравнение.

1. ПЕРВЫЙ РАЗДЕЛ

1.1. Чеклист

Тест 1 «Авторизация (вход и выход)»

Шаги теста

1. Перейти на страницу входа.
2. Ввести в поле "Логин " значение " *здесь будет логин* ".
3. Ввести в поле "Пароль" значение " *здесь будет пароль* ".
4. Нажать кнопку "Войти".
5. Нажать ссылку "Выход".

Ожидаемый результат

1. Пользователь успешно входит: отображается личный кабинет;
2. после выхода — перенаправление на главную страницу.

Тест 2 «Поиск товаров»

Шаги теста

1. Перейти на главную страницу.
2. Ввести в поле поиска значение "Краска акриловая белая".
3. Нажать Enter.
4. В фильтре "Цена от" ввести "150", в фильтре "Цена до" ввести "400", нажать "Применить".
5. Выбрать сортировку "по цене (возрастание)".

Ожидаемый результат

1. В результатах отображаются товары с названием, содержащим "Краска акриловая белая".
2. Все цены товаров в диапазоне 150 400 руб.
3. Товары упорядочены по возрастанию цены.

Тест 3 «Добавление товара в корзину»

Шаги теста

Предусловие: через API создать товар с артикулом "104843", названием "Штукатурка гипсовая Knauf МП-75 машинная 30 кг", ценой 615 руб.

1. Перейти на страницу товара 104843.
2. Нажать кнопку "В корзину".

Ожидаемый результат

1. Счётчик в шапке корзины увеличился на 1;
2. в корзине отображается "Штукатурка гипсовая Knauf МП-75 машинная 30 кг" в количестве 1.

Тест 4 «Изменение количества товара в корзине»
Шаги теста

Предусловие: товар 104843 в корзине в количестве 1.
1. Перейти на страницу корзины.
2. В поле "Количество" напротив товара ввести "5", нажать "Обновить".
3. В поле "Количество" напротив товара ввести "0", нажать "Обновить".

Ожидаемый результат

1. После шага 2: количество товара обновилось на 5, итоговая сумма = 5 × 615 = 3075 руб.
2. После шага 3: товар 104843 удалён из корзины, корзина пуста.

Тест 5 «Создание сметы»
Шаги теста

Предусловие: через API в корзину добавить товар 104843 в количестве 2.

1. Перейти в раздел "Сметы".
2. Нажать "Создать новую смету".
3. Ввести в поле "Название сметы" значение "Тестовая смета".
4. Нажать "Добавить товар", ввести "104843" в поле "Артикул", нажать "Добавить", ввести "2" в поле "Количество".
5. Нажать "Сохранить".

Ожидаемый результат

1. Смета "Тестовая смета" создана. В смете товар 104843 (2 шт.).
2. Итоговая сумма = 2 × 615 = 1230 руб.

Тест 6 «Удаление из корзины»

Шаги теста

Предусловие: через API добавить в корзину товар 104843 в количестве 1.

1. Перейти на страницу корзины.
2. Нажать иконку удаления напротив товара 104843.

Ожидаемый результат

1. Товар 104843 удалён из корзины; счётчик в шапке корзины уменьшился на 1;
2. итоговая сумма пересчитана без учёта 104843.

Тест 7 «Добавление и удаление из избранного»

Шаги теста

Предусловие: через API создать товар с артикулом "104843" и названием "Штукатурка гипсовая Knauf МП-75 машинная 30 кг".

1. Перейти на страницу товара 104843.
2. Нажать иконку "Избранное".
3. Нажать иконку "Избранное" снова.

Ожидаемый результат

1. После шага 2: счётчик избранного увеличился на 1;
2. Товар 104843 присутствует в списке избранного.
3. После шага 3: счётчик избранного уменьшился на 1;
4. Товар 104843 отсутствует в списке избранного.

Тест 8 «Просмотр карточки товара

Шаги теста

Предусловие: через API создать товар с артикулом "104843", названием "Штукатурка гипсовая Knauf МП-75 машинная 30 кг", ценой 615 руб.,

описанием "Гипсовая машинная штукатурка Knauf МП-75", фото "mp75.jpg" добавить в тестовые данные.

1. Перейти на страницу товара 104843.

Ожидаемый результат

1. На карточке товара отображаются:
   – Название "Штукатурка гипсовая Knauf МП-75 машинная 30 кг";
   – Фото "mp75.jpg"; – Цена "615 руб.";
   – Описание "Гипсовая машинная штукатурка Knauf МП-75";
   – Технические характеристики (таблица);
   – Разделы с отзывами и информацией о доставке видны.

2. ВТОРОЙ РАЗДЕЛ

2.1. Описание классов и методов

Элементы (package elements)

BaseElement.java

- **Класс:** BaseElement
  Описание: Базовый класс для всех UI-элементов, инкапсулирует взаимодействие через Selenide.
- **Поля:**

- By locator – локатор элемента на странице.
- SelenideElement element – экземпляр найденного элемента (инициализируется по locator).
- Logger logger – логгер для записи действий.

- **Конструктор:**

- BaseElement(By locator) – сохраняет локатор и инициализирует element.

- **Методы:**

- void click() – клик по элементу.
- void setValue(String value) – установка текста в элемент (для input).
- String getText() – получение текста элемента.

Button.java

- **Класс:** Button
  Наследует: BaseElement
  Описание: Представляет кнопку на странице.
- **Статические фабрики:**

- static Button byText(String text) – находит кнопку по видимому тексту.
- static Button byLocator(By locator) – находит кнопку по любому локатору.

- **Методы (унаследованные):**

- void click() – клик по кнопке.

Input.java

- **Класс:** Input
  Наследует: BaseElement
  Описание: Поле ввода текста.
- **Статические фабрики:**

- static Input byLocator(By locator) – найти по локатору.

- **Методы:**

- void setValue(String value) – ввод текста.
- void pressEnter() – имитация нажатия клавиши Enter.
- String getValue() – получить текущее значение поля.

Table.java

- **Класс:** Table
  Наследует: BaseElement
  Описание: Таблица на странице.
- **Методы:**

- List<SelenideElement> getRows() – получить все строки таблицы.
- SelenideElement getRow(int index) – получить конкретную строку по индексу.
- String getCellValue(int rowIndex, int columnIndex) – получить значение ячейки.

Страницы (package page)

BasePage.java

- **Класс:** BasePage
  Описание: Базовые действия для всех страниц.
- **Поля:**

- String baseUrl – базовый URL приложения.
- Logger logger – логгер.

- **Методы:**

- void open(String path) – открыть страницу baseUrl + path.
- String getCurrentUrl() – вернуть текущий URL.
- String getTitle() – получить заголовок страницы.

HomePage.java

- **Класс:** HomePage (extends BasePage)
  Описание: Главная страница сайта.
- **Поля:**

- Button signInButton – кнопка «Войти».
- Input searchInput – поле поиска.

- **Методы:**

- HomePage open() – открыть главную страницу (/).
- HomePage search(String query) – ввести запрос в searchInput и нажать Enter.

LoginPage.java

- **Класс:** LoginPage (extends BasePage)
  Описание: Страница авторизации.
- **Поля:**

- Input usernameInput – поле для логина.
- Input passwordInput – поле для пароля.

- Button submitButton – кнопка «Войти» (подтвердить).

- **Методы:**

- LoginPage open() – открыть страницу входа (/login).
- void login(String username, String password) – заполнить поля и нажать submitButton.

AccountPage.java

- **Класс:** AccountPage (extends BasePage)
  Описание: Личный кабинет пользователя.
- **Методы:**

- boolean isLoggedIn() – проверить, что пользователь авторизован.
- CartPage goToCart() – перейти в корзину.
- FavoritesPage goToFavorites() – перейти в избранное.

CartPage.java

- **Класс:** CartPage (extends BasePage)
  Описание: Страница корзины.
- **Поля:**

- Button removeItemButton(By productLocator) – кнопка удаления товара.
- Input quantityInput(By productLocator) – поле изменения количества.
- Span cartCount – элемент отображения количества товаров.

- **Методы:**

- CartPage open() – открыть страницу /cart.
- int getCartCount() – получить число товаров в корзине.
- void removeProduct(String productId) – удалить товар по ID.
- void changeQuantity(String productId, int qty) – изменить количество товара.

EstimatePage.java

- **Класс:** EstimatePage (extends BasePage)
  Описание: Страница создания сметы.
- **Методы:**

- EstimatePage open() – открыть /estimate.
- void fillEstimateForm(Map<String, String> data) – заполнить поля формы сметы.
- void submit() – отправить форму.

EstimatesPage.java

- **Класс:** EstimatesPage (extends BasePage)
  Описание: Список созданных смет.
- **Методы:**

- List<String> getEstimateIds() – получить список ID смет.
- void deleteEstimate(String id) – удалить смету.

FavoritesPage.java

- **Класс:** FavoritesPage (extends BasePage)
  Описание: Страница избранного.
- **Поля:**

- Span favoritesCount – элемент отображения количества избранного.
- Button removeFavoriteButton(By productLocator) – убрать из избранного.

- **Методы:**

- FavoritesPage open() – открыть /favorites.
- int getFavoritesCount() – получить число избранных товаров.
- void removeFavorite(String productId) – удалить.

ProductPage.java

- **Класс:** ProductPage (extends BasePage)
  Описание: Страница конкретного товара.
- **Поля:**

- Span price – элемент цены.
- Div description – описание товара.
- Button addToCartButton – кнопка «В корзину».
- Button addToFavoritesButton – кнопка «В избранное».

- **Методы:**

- ProductPage open(String productId) – открыть /product/{id}.
- void addToCart() – клик по addToCartButton.
- void addToFavorites() – клик по addToFavoritesButton.
- String getPrice() – получить текст цены.
- String getDescription() – получить описание.

SearchResultsPage.java

- **Класс:** SearchResultsPage (extends BasePage)
  Описание: Страница результатов поиска.
- **Поля:**

- Input priceFromInput – поле «Цена от».
- Input priceToInput – поле «Цена до».
- Button sortPriceAscButton – кнопка сортировки по возрастанию цены.
- ElementsCollection<SelenideElement> productCards – коллекция карточек товаров.

- **Методы:**

- SearchResultsPage filterByPrice(int min, int max) – установить диапазон цен и дождаться хотя бы одной карточки.
- SearchResultsPage sortByPriceAscending() – клик по sortPriceAscButton и ожидание появления карточек.
- List<Integer> getPrices() – извлечь цены из productCards.

Тесты (package tests)

BaseTest.java

- **Класс:** BaseTest
  Описание: Общая конфигурация тестового фреймворка.
- **Методы:**

- @BeforeEach void setup() – инициализация WebDriver/Selenide.
- @AfterEach void teardown() – завершение сессии.
- BasePage getPage(Class<? extends BasePage> pageClass) – создать инстанс страницы.

AuthTest.java

- **Класс:** AuthTest (extends BaseTest)
  Описание: Тесты авторизации.
- **Методы:**

- @Test void testSuccessfulLogin() – проверяет вход с валидными данными.

CartTest.java

- **Класс:** CartTest (extends BaseTest)
  Описание: Тесты корзины.
- **Методы:**

- @Test void testAddProductToCart() – добавить товар и проверить счётчик.
- @Test void testChangeProductQuantity() – изменить количество товара.
- @Test void testRemoveProductFromCart() – удалить товар и проверить счётчик.

EstimateTest.java

- **Класс:** EstimateTest (extends BaseTest)
  Описание: Тест создания сметы.
- **Методы:**

- @Test void testCreateEstimate() – заполнение формы и проверка появления новой сметы.

FavoritesTest.java

- **Класс:** FavoritesTest (extends BaseTest)
  Описание: Тесты избранного.
- **Методы:**

- @Test void testAddAndRemoveFavorite() – добавить в избранное и затем удалить.

ProductPageTest.java

- **Класс:** ProductPageTest (extends BaseTest)
  Описание: Проверка элементов страницы товара.
- **Методы:**

- @Test void testProductDetailsPageElements() – убедиться, что отображаются цена и описание.

SearchTest.java

- **Класс:** SearchTest (extends BaseTest)
  Описание: Проверка фильтрации и сортировки.
- **Методы:**

- @Test void testSearchWithFilterAndSort() – выполнить поиск, отфильтровать по цене, проверить результаты и сортировку.

2.2. UML-диаграмма

Нами была создана UML-диаграмма (рис 1.), с помощью сайта www.plantuml.com. Для этого был написан код (см. приложение А).

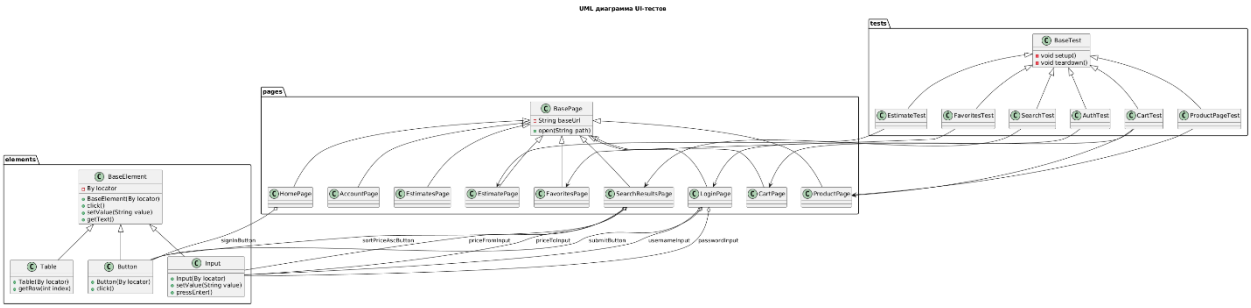

Рисунок 1 - UML-диаграмма

3. ТРЕТИЙ РАЗДЕЛ

1.1. Тестирование одного теста

Для описания выбран тест «Авторизация»

1. Программа заходит на сайт СТД Петрович и проходит регистрацию на сайте. Вводит почту и пароль(рис.2).

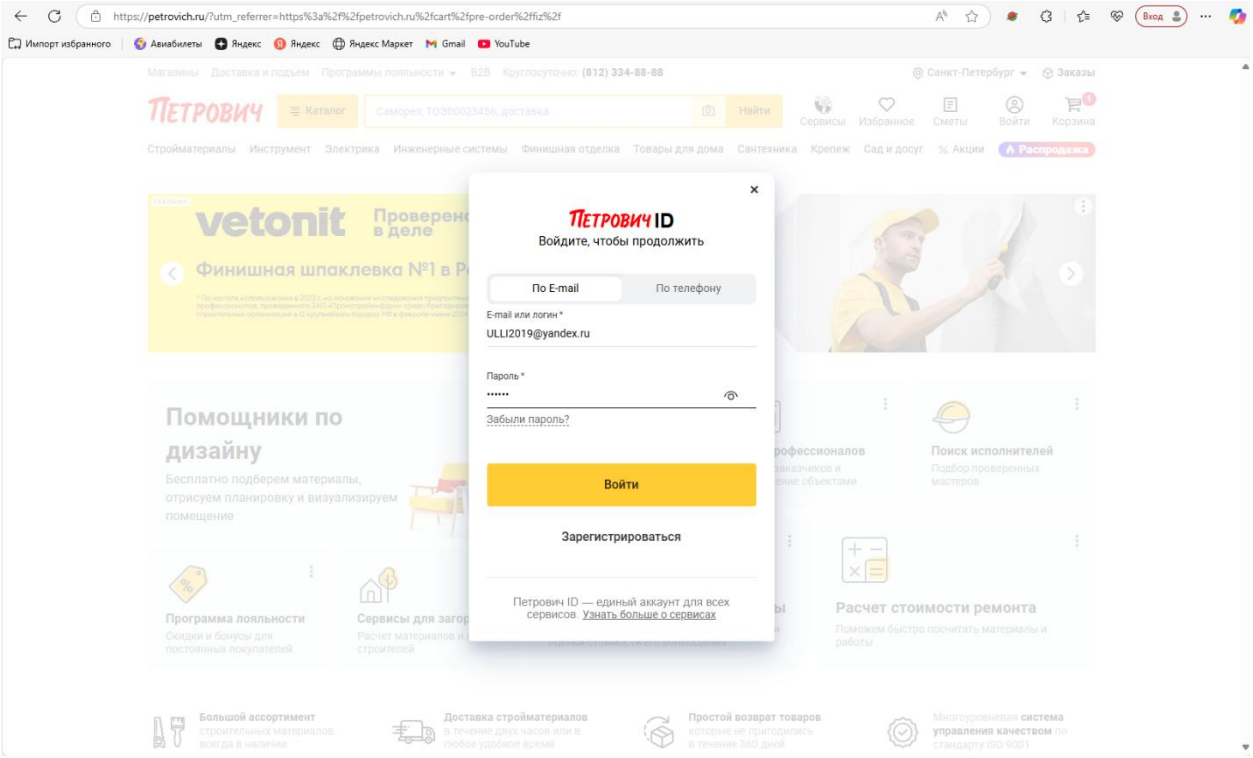

Рисунок 2 - Авторизация

2. Далее в поисковой строке вводится запрос «краска берая акриловая», клик по кнопке «Найти», выбор товара по запросу (рис.3)

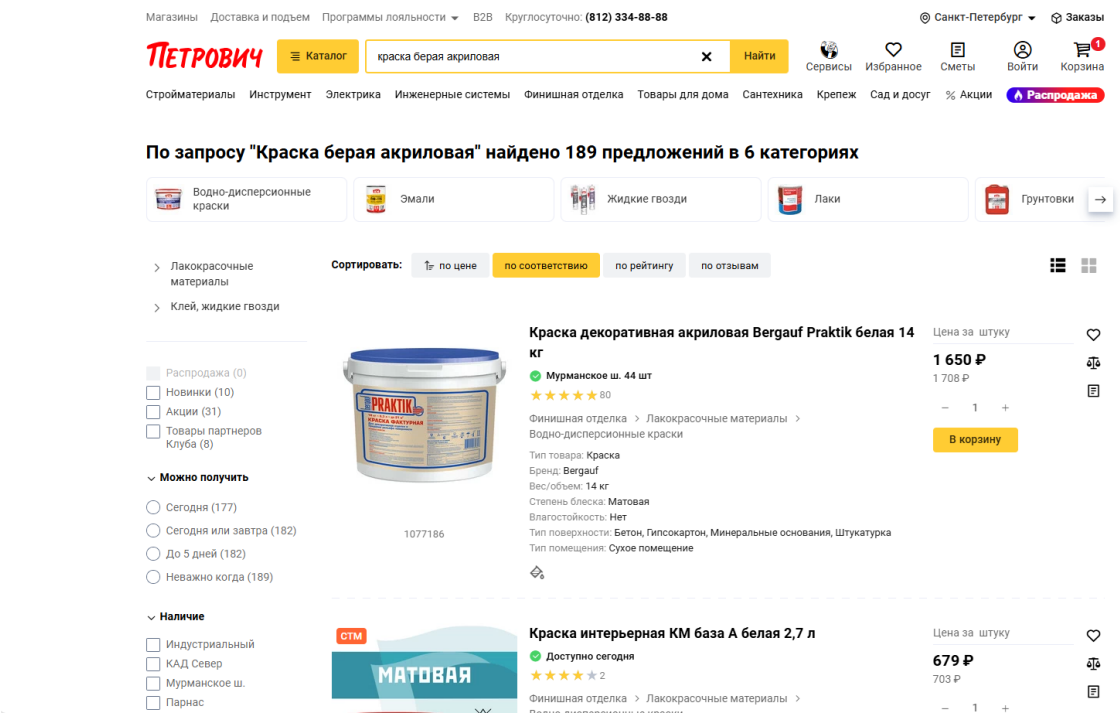

Рисунок 3 – Ввод товара в строку поиска

3. Программа выставляет диапазон цены (рис.4)

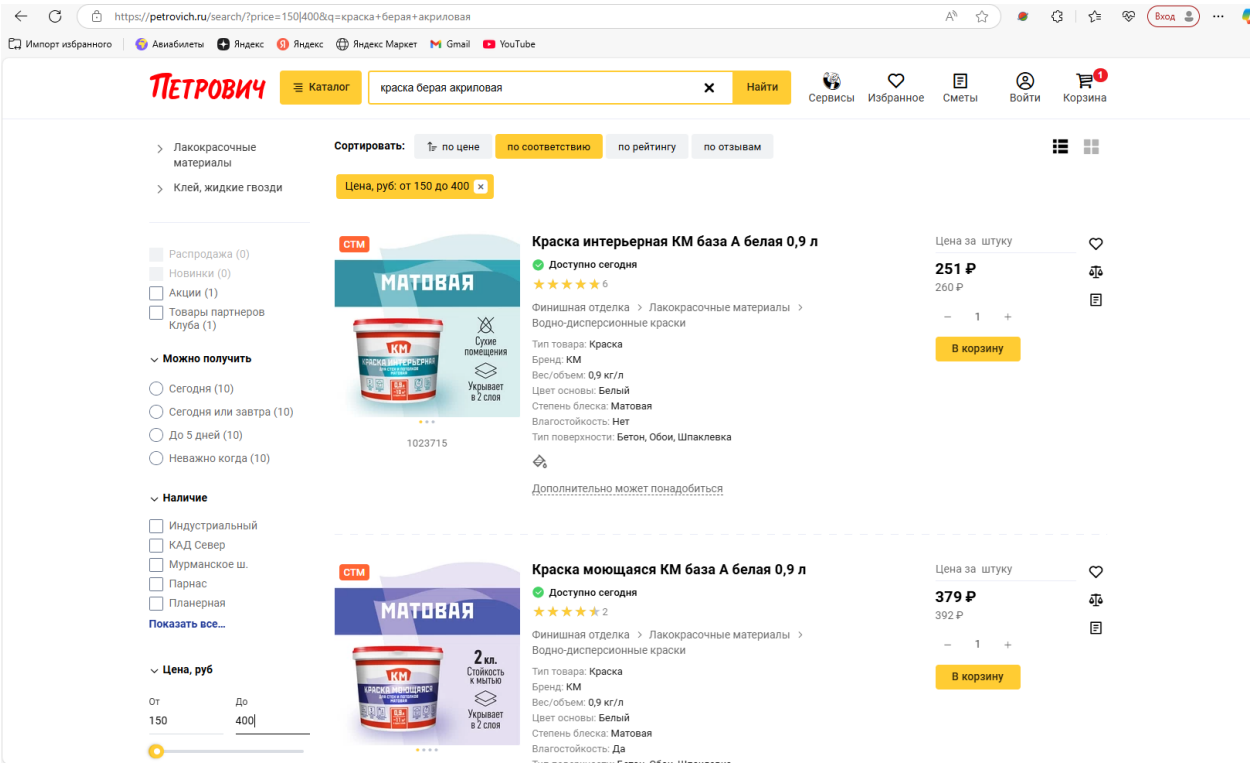

Рисунок 4 – Высиавление диапазона

ЗАКЛЮЧЕНИЕ

Кратко подвести итоги, проанализировать соответствие поставленной цели и полученного результата.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. «Программирование на Java»

2. СТД Петрович https://petrovich.ru/

3. https://www.selenium.dev/documentation/webdriver/ui_testing/
   (Официальная документация Selenium по тестированию пользовательского интерфейса)

4. Мэйвороно, Энтони
   Test-Driven Development with Selenium — Packt Publishing, 2021.
   (Подробное описание работы с UI-тестами через Selenium и Selenide

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: BaseElement.java

```
package com.petrovich.ui.elements;

import com.codeborne.selenide.SelenideElement;
import com.codeborne.selenide.Condition;
import java.time.Duration;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public abstract class BaseElement {
    protected final SelenideElement element;
    protected final Logger logger = LoggerFactory.getLogger(getClass());

    protected BaseElement(SelenideElement element) {
        this.element = element;
    }

    public boolean isDisplayed() {
        try {
            element.shouldBe(Condition.visible, Duration.ofSeconds(2));
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    public void click() {
                                        element.shouldBe(Condition.visible,
Duration.ofSeconds(2)).click();
        logger.info("Clicked on element: {}", element);
    }

    public String getText() {
        element.shouldBe(Condition.visible, Duration.ofSeconds(2));
        return element.getText();
    }
}
```

Название файла: Input.java

```
package com.petrovich.ui.elements;

import com.codeborne.selenide.SelenideElement;
import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selenide.$x;
/**
 * Обёртка над SelenideElement для работы с полями ввода.
 */
public class Input extends BaseElement {
    private Input(SelenideElement element) {
        super(element);
    }
```

```
    /**
     * Поиск по XPath-выражению.
     */
    public static Input byXpath(String xpath) {
        return new Input($x(xpath));
    }

    /**
     * Поиск по значению атрибута id.
     */
    public static Input byId(String id) {
        return new Input($("#" + id));
    }

    /**
     * Поиск по значению атрибута name.
     */
    public static Input byName(String name) {
        return new Input($("input[name='" + name + "']"));
    }

    /**
     * Поиск по произвольному CSS-селектору.
     */
    public static Input byCss(String cssSelector) {
        return new Input($(cssSelector));
    }

    /**
     * Очистка и ввод текста в поле.
     */
    public void setValue(String text) {
        element.clear();
        element.setValue(text);
        logger.info("Input value set to: {}", text);
    }

    /**
     * Нажатие Enter в поле.
     */
    public void pressEnter() {
        element.pressEnter();
        logger.info("Pressed Enter in input");
    }
}
```

Название файла: Button.java

```
package com.petrovich.ui.elements;

import com.codeborne.selenide.SelenideElement;
import com.codeborne.selenide.Selectors;
import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selenide.$x;
```

```
/**
 * Обёртка над SelenideElement для работы с кнопками.
 */
public class Button extends BaseElement {
    private Button(SelenideElement element) {
        super(element);
    }

    /**
     * Поиск кнопки по точному тексту.
     */
    public static Button byText(String text) {
        return new Button($(Selectors.byText(text)));
    }

    /**
     * Поиск кнопки по CSS-селектору.
     */
    public static Button byCss(String cssSelector) {
        return new Button($(cssSelector));
    }

    /**
     * Поиск кнопки по XPath-выражению.
     */
    public static Button byXpath(String xpath) {
        return new Button($x(xpath));
    }
}
```

Название файла: Table.java

```
package com.petrovich.ui.elements;

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

public class Table extends BaseElement {
    public Table(SelenideElement element) {
        super(element);
    }

    public static Table byCss(String cssSelector) {
        return new Table($(cssSelector));
    }

    public int getRowCount() {
        return element.$$("[role=row], tr").size();
    }

    public boolean contains(String text) {
        return element.getText().contains(text);
    }
}
```

Название файла: BasePage.java

```
package com.petrovich.ui.pages;

import com.codeborne.selenide.Selenide;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public abstract class BasePage {
    protected final Logger logger = LoggerFactory.getLogger(getClass());

    protected void openPage(String relativeUrl) {
        logger.info("Opening page: {}", relativeUrl);
        String url = relativeUrl.startsWith("/") ? relativeUrl : "/" +
relativeUrl;
        Selenide.open(url);
    }

    public int getCartCount() {
        String text = Selenide.$("span.cart-count").getText();
        return text.isEmpty() ? 0 : Integer.parseInt(text.trim());
    }

    public int getFavoritesCount() {
        String text = Selenide.$("span.favorites-count").getText();
        return text.isEmpty() ? 0 : Integer.parseInt(text.trim());
    }
}
```

Название файла: HomePage.java

```
package com.petrovich.ui.pages;

import com.petrovich.ui.elements.Input;
import com.petrovich.ui.elements.Button;

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

/**
 * Страница главного экрана сайта.
 */
public class HomePage extends BasePage {
    // TODO: Через DevTools скопируйте точный CSS-селектор поля поиска
и при необходимости замените его здесь
     private final Input searchInput = Input.byCss("#page-header-
search > div > div > input");

    // TODO: Уточните точный текст кнопки «Войти» в UI (например, из
ссылки в хедере)
    private final Button loginLink   = Button.byText("Войти");

    /**
     * Открывает главную страницу.
     * TODO: Проверьте, что относительный URL ("/") соответствует тому,
что используется в приложении.
```

```
     */
    public HomePage open() {
        openPage("/");
        return this;
    }

    /**
     * Выполняет поиск по запросу и переходит на страницу результатов.
      * @param query текст для поиска, скопированный/введённый в поле
поиска
     */
    public SearchResultsPage searchFor(String query) {
        logger.info("Searching for: {}", query);
        searchInput.setValue(query);
        searchInput.pressEnter();
        return new SearchResultsPage();
    }

    /**
     * Переходит на страницу логина.
     */
    public LoginPage goToLoginPage() {
        loginLink.click();
        return new LoginPage();
    }

    /**
     * Проверяет, видна ли ссылка «Войти».
     * @return true, если элемент отображается на странице
     */
    public boolean isLoginLinkVisible() {
           // TODO: Метод передаёт ту же строку, что и loginLink
(Button.byText), убедитесь в её корректности
        return loginLink.isDisplayed();
    }
}
```

Название файла: LoginPage.java

```
package com.petrovich.ui.pages;

import com.petrovich.ui.elements.Input;
import com.petrovich.ui.elements.Button;
import static com.codeborne.selenide.Selenide.sleep;

/**
 * Страница логина пользователя.
 */
public class LoginPage extends BasePage {
    // Элементы страницы логина
                    private        Input        usernameInput        =
Input.byXpath("/html/body/div[7]/div/div/div[2]/div[2]/div/div[2]/div[
1]/form/div[1]/div/div/label/input");
                    private        Input        passwordInput        =
Input.byXpath("/html/body/div[7]/div/div/div[2]/div[2]/div/div[2]/div[
1]/form/div[2]/div/div/label/input");
```

```
                    private        Button        loginButton         =
Button.byXpath("/html/body/div[7]/div/div/div[2]/div[2]/div/div[2]/div
[1]/form/button[1]");

    /**
     * Выполняет вход в систему и возвращает страницу аккаунта.
     * Добавлена задержка перед возвратом, чтобы дать странице время на
обработку.
     * @param username логин пользователя
     * @param password пароль пользователя
     * @return AccountPage после успешного входа
     */
    public AccountPage login(String username, String password) {
        logger.info("Attempting to login with username: {}", username);
        usernameInput.setValue(username);
        passwordInput.setValue(password);
        loginButton.click();

        // Ждём 1 секунду, чтобы страница успела обновиться
        sleep(1000);

        return new AccountPage();
    }
}
```

Название файла: AccountPage.java

```
package com.petrovich.ui.pages;

import com.petrovich.ui.elements.Button;

public class AccountPage extends BasePage {
    private final Button logoutButton = Button.byText("Выход");

    public HomePage logout() {
        logger.info("Logging out");
        logoutButton.click();
        return new HomePage();
    }

    public boolean isLoggedIn() {
        return logoutButton.isDisplayed();
    }
}
```

Название файла: ProductPage.java

```
package com.petrovich.ui.pages;

import com.petrovich.ui.elements.Button;
import com.petrovich.ui.elements.Table;
import com.codeborne.selenide.Condition;
import com.codeborne.selenide.SelenideElement;

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

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

/**
 * Страница товара.
 */
public class ProductPage extends BasePage {
    private final Button addToCartButton = Button.byText("В корзину");

    // TODO: Найдите в DevTools селектор кнопки добавления в избранное
(CSS-класс, data-атрибут и т.п.)
    private final Button favoriteButton  = Button.byText("Избранное");

     // TODO: В DevTools найдите таблицу технических характеристик и
скопируйте её уникальный CSS-селектор
     private final Table techSpecsTable    = Table.byCss("table.tech-
specs");

    /**
     * Открыть страницу товара по артикулу.
       * TODO: Проверьте, что относительный путь "/product/{code}"
соответствует маршруту в приложении.
     */
    public ProductPage open(String productCode) {
        openPage("/product/" + productCode);//нужно здесь прописать что
должен делать класс open а не ссылаться на openPage.
        return this;
    }

    /**
     * Добавить товар в корзину.
     */
    public void addToCart() {
        logger.info("Adding product to cart");
        addToCartButton.click();
    }

    /**
     * Получить текущее число в шапке корзины.
     */
    public int getCartCount() {
        return super.getCartCount();
    }

    /**
     * Добавить товар в избранное.
     */
    public void addToFavorites() {
        logger.info("Adding product to favorites");
        favoriteButton.click();
    }

    /**
     * Убрать товар из избранного.
     */
    public void removeFromFavorites() {
        logger.info("Removing product from favorites");
```

```
        favoriteButton.click();
    }

    /**
     * Заголовок товара (h1).
      * TODO: Убедитесь, что заголовок расположен в теге h1 через
DevTools.
     */
    public String getTitle() {
        SelenideElement title = $("h1").shouldBe(Condition.visible);
        return title.getText();
    }

    /**
     * Текст цены, например "615 ₽".
          *   TODO:   Уточните   точный   селектор   элемента   цены
(например .price__value) через DevTools.
     */
    public String getPrice() {
                                    SelenideElement        price        =
$("[class*=price]").shouldBe(Condition.visible);
        return price.getText();
    }

    /**
     * Описание товара.
     * TODO: Найдите контейнер описания товара (например div.product-
description) и скопируйте селектор.
     */
    public String getDescription() {
                        SelenideElement    desc    =    $("div.product-
description").shouldBe(Condition.visible);
        return desc.getText();
    }

    /**
     * Есть ли таблица технических характеристик.
     */
    public boolean hasTechnicalSpecifications() {
        return techSpecsTable.isDisplayed();
    }

    /**
     * Есть ли секция "Отзывы".
     * TODO: Уточните селектор секции отзывов (например id="reviews" или
класс reviews-section).
     */
    public boolean hasReviewsSection() {
        return $("*[class*=reviews], *:contains('Отзывы')").exists();
    }

    /**
     * Есть ли секция "Доставка".
     * TODO: Уточните селектор секции доставки (например div.delivery-
info) через DevTools.
     */
```

```
    public boolean hasDeliveryInfoSection() {
       return $("*[class*=delivery], *:contains('Доставка')").exists();
    }

    /**
     * Видна ли картинка с подстрокой imageName в пути.
     * TODO: Передавайте сюда имя файла или часть пути картинки, как
указано в атрибуте src.
     */
    public boolean isImageDisplayed(String imageName) {
        return $("img[src*='" + imageName + "']").exists();
    }
}
```

Название файла: SearchResultsPage.java

```
package com.petrovich.ui.pages;

import com.codeborne.selenide.Condition;
import com.codeborne.selenide.ElementsCollection;
import com.petrovich.ui.elements.Input;
import com.petrovich.ui.elements.Button;

import java.time.Duration;

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

/**
 * Страница результатов поиска.
 */
public class SearchResultsPage extends BasePage {

     // TODO: Скопируйте через DevTools селектор поля "Цена от"
(placeholder или id) для точного совпадения
          private    final    Input    priceFromInput              =
Input.byXpath("/html/body/div[2]/main/div/div/div[2]/aside/div/div[4]/
div/div/div[1]/div[1]/label/input");

     // TODO: Скопируйте через DevTools селектор поля "Цена до"
(placeholder 'До' или другой уникальный атрибут)
          private    final    Input    priceToInput                =
Input.byXpath("/html/body/div[2]/main/div/div/div[2]/aside/div/div[4]/
div/div/div[1]/div[2]/label/input");

     // TODO: Скопируйте через DevTools текст/селектор для кнопки
сортировки по цене по возрастанию
    private final Button sortPriceAscButton = Button.byText("по цене");

    /**
     * Задать диапазон цен и применить фильтр.
     * @param min минимальная цена
     * @param max максимальная цена
     */
    public SearchResultsPage filterByPrice(int min, int max) {
        logger.info("Filtering by price {}-{}", min, max);
```

```
        priceFromInput.click();
        priceFromInput.setValue(String.valueOf(min));

        priceToInput.click();
        priceToInput.setValue(String.valueOf(max));

        // TODO: При необходимости поменяйте селектор "div.product-card"
на тот, что появляется при загрузке результатов
        $("div.product-card")
            .should(Condition.appear, Duration.ofSeconds(2));

        return this;
    }

    /**
     * Отсортировать результаты по возрастанию цены.
     */
    public SearchResultsPage sortByPriceAscending() {
        logger.info("Sorting by price ascending");
        sortPriceAscButton.click();
       // TODO: Уточните селектор карточки товара для проверки появления
после сортировки
        $("div.product-card")
            .should(Condition.appear, Duration.ofSeconds(2));
        return this;
    }

    /**
     * Считать все цены товаров на странице.
     * @return массив цен (int), без символов валюты
     */
    public int[] getAllProductPrices() {
        // Ищем все элементы span, класс которых содержит 'pt-price'
       ElementsCollection elems = $$("div.product-card span[class*='pt-
price']");
        return elems.texts().stream()
                                                .mapToInt(text    ->
Integer.parseInt(text.replaceAll("\\D", "")))
                    .toArray();
    }

    /**
     * Считать все заголовки товаров на странице.
     */
    public ElementsCollection getAllProductTitles() {
         // TODO: Уточните селектор заголовков, например "h2.product-
title"
        return $$("div.product-card h2.product-title");
    }
}
```

Название файла: CartPage.java

```
// src/main/java/com/petrovich/ui/pages/CartPage.java
package com.petrovich.ui.pages;
```

```
import com.codeborne.selenide.Condition;
import com.codeborne.selenide.SelenideElement;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.time.Duration;
import  static  com.codeborne.selenide.Selenide.$x;            //  для
относительных XPath вызовов
import  static  com.codeborne.selenide.Selenide.$;             //  для
$("span.cart-total")
import static com.codeborne.selenide.Selenide.open;

public class CartPage extends BasePage {
            private     static     final     Logger     logger     =
LoggerFactory.getLogger(CartPage.class);

    /** Открыть корзину */
    public CartPage open() {
        openPage("/cart");
        return this;
    }

    /** Найти карточку товара по коду */
    private SelenideElement findCartItem(String code) {
        return $x(
          "//*[contains(@class,'cart-item') " +
          " and .//*[contains(text(),'" + code + "')]]"
        );
    }

    /** Есть ли товар в корзине */
    public boolean isProductInCart(String code) {
        return findCartItem(code).exists();
    }


    public CartPage updateProductQuantity(String code, int qty) {
        SelenideElement item = findCartItem(code);

        // 1) Находим поле количества через XPath внутри карточки
                                  SelenideElement       qtyInput       =
item.$x("/html/body/div[2]/main/div/div/div/div/div[2]/div/div/div/div
[3]/div[4]/div[2]/div[1]/div/div/div/input");

        // 2) Кликаем, очищаем и вводим новое значение
                                    qtyInput.shouldBe(Condition.visible,
Duration.ofSeconds(2)).click();
        qtyInput.clear();
        qtyInput.setValue(String.valueOf(qty));


        logger.info("Updated {} qty to {}", code, qty);
        return this;
    }
```

```
    /** Получить текущее количество товара */
    public int getProductQuantity(String code) {
        String v = findCartItem(code)
                    .$("input.quantity")
                    .getValue();
        return Integer.parseInt(v);
    }

    /** Удалить товар из корзины */
    public CartPage removeProduct(String code) {
        findCartItem(code)
          .$x(".//button[contains(@class,'remove-item')]")
          .click();
        logger.info("Removed {} from cart", code);
        return this;
    }

    /** Общая сумма корзины */
    public int getTotalPrice() {
        String t = $("span.cart-total")
                     .getText()
                     .replaceAll("\\D", "");
        return t.isEmpty() ? 0 : Integer.parseInt(t);
    }
}
```

Название файла: EstimatesPage.java

```
package com.petrovich.ui.pages;

import com.petrovich.ui.elements.Button;
import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selectors.byText;

/**
 * Страница списка смет.
 */
public class EstimatesPage extends BasePage {
    // TODO: В DevTools найдите кнопку «Создать новую смету» и скопируйте
её точный текст для локатора
     private final Button createBtn = Button.byText("Создать новую
смету");

    /**
     * Открывает страницу со списком смет.
     * TODO: Проверьте в адресной строке браузера относительный путь
(например "/estimates") и при необходимости скорректируйте.
     */
    public EstimatesPage open() {
        openPage("/estimates");
        return this;
    }

    /**
     * Нажимает кнопку создания новой сметы.
     * Возвращает страницу создания сметы.
```

```
     */
    public EstimatePage clickCreateNewEstimate() {
        createBtn.click();
        return new EstimatePage();
    }

    /**
     * Проверяет, что на странице присутствует смета с указанным именем.
     * @param name точное название сметы, скопированное с UI (например,
из заголовка или списка)
     */
    public boolean isEstimatePresent(String name) {
        // TODO: при вызове передавайте сюда текст названия сметы точно
так же, как он отображён на странице
        return $(byText(name)).exists();
    }
}
```

Название файла: EstimatePage.java

```
package com.petrovich.ui.pages;

import com.petrovich.ui.elements.Input;
import com.petrovich.ui.elements.Button;
import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selenide.$x;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class EstimatePage extends BasePage {
    private final Logger logger = LoggerFactory.getLogger(getClass());

    // Поля формы создания сметы
    private Input nameInput     = Input.byName("estimateName");
    private Button addBtn       = Button.byText("Добавить");
    private Input articleInput  = Input.byName("article");
    private Input quantityInput = Input.byName("quantity");
    private Button saveBtn      = Button.byText("Сохранить");

    /** Установить название сметы */
    public void setName(String name) {
        logger.info("Setting estimate name: {}", name);
        nameInput.setValue(name);
    }

    /** Добавить товар по артикулу и количеству */
    public void addProduct(String code, int qty) {
        logger.info("Adding {} x{} to estimate", code, qty);
        articleInput.setValue(code);
        addBtn.click();
        quantityInput.setValue(String.valueOf(qty));
    }

    /** Сохранить смету */
    public void save() {
        logger.info("Saving estimate");
```

```
        saveBtn.click();
    }

    /** Проверить, что в смете есть товар с указанным кодом и количеством
*/
    public boolean containsProduct(String code, int qty) {
        String xpath = String.format(
             "//*[contains(text(), '%s') and contains(text(), '%d')]",
code, qty
        );
        return $x(xpath).exists();
    }

    /** Получить итоговую сумму сметы */
    public int getTotal() {
                          String     text     =     $("span.estimate-
total").getText().replaceAll("\\D", "");
        return text.isEmpty() ? 0 : Integer.parseInt(text);
    }
}
```

Название файла: FavoritesPage.java

```
package com.petrovich.ui.pages;

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

/**
 * Страница избранных товаров.
 */
public class FavoritesPage extends BasePage {
    // TODO: Проверьте в адресной строке относительный путь к странице
избранного (например, "/favorites")
    public FavoritesPage open() {
        openPage("/favorites");
        return this;
    }

    /**
      * Проверяет, что на странице избранного присутствует товар с
указанным кодом.
     * @param code точный код товара, скопированный из UI (например, из
карточки товара или списка)
     */
    public boolean containsProduct(String code) {
        // TODO: Вызовите this метод с тем же текстом, что отображается
на странице
        return $(byText(code)).exists();
    }
}
```

Название файла: BaseTest.java

```
// src/test/java/com/petrovich/ui/tests/BaseTest.java
package com.petrovich.ui.tests;
```

```
import com.codeborne.selenide.Configuration;
import com.codeborne.selenide.Selenide;
import com.codeborne.selenide.WebDriverRunner;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterAll;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

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

public abstract class BaseTest {
            protected     static     final     Logger     logger     =
LoggerFactory.getLogger(BaseTest.class);

    @BeforeAll
    public static void setup() {
        Configuration.browserSize      = "1920x1080";
        Configuration.baseUrl          = "https://petrovich.ru";
        Configuration.pageLoadStrategy = "eager";
        Configuration.timeout           = 10_000;
        logger.info("Starting UI tests");
    }

    @BeforeEach
    public void cleanState() {
        // Гарантируем привязку WebDriver к текущему потоку
        if (!WebDriverRunner.hasWebDriverStarted()) {
            open("/");
        }
        //Selenide.clearBrowserCookies();
        //Selenide.clearBrowserLocalStorage();
    }

    @AfterAll
    public static void tearDown() {
        logger.info("UI tests completed");
    }
}
```

Название файла: AuthTest.java

```
package com.petrovich.ui.tests;

import com.petrovich.ui.pages.HomePage;
import com.petrovich.ui.pages.LoginPage;
import com.petrovich.ui.pages.AccountPage;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static com.codeborne.selenide.Selenide.sleep;

public class AuthTest extends BaseTest {
    private static final String TEST_LOGIN    = "ULLI2019@yandex.ru";
    private static final String TEST_PASSWORD = "s5GK4V";
```

```
    @Test
    public void testLoginLogout() {

        HomePage home = new HomePage().open(); // откр баз стр
        sleep(2000);
        LoginPage login = home.goToLoginPage(); // переход на автор
        sleep(6000);
         AccountPage account = login.login(TEST_LOGIN, TEST_PASSWORD);
//введение пароля loginPage
        //Работает верно

        //assertTrue(account.isLoggedIn());
        //HomePage after = account.logout();
        //assertTrue(after.isLoginLinkVisible());
    }
}
```

Название файла: SearchTest.java

```
package com.petrovich.ui.tests;

import com.petrovich.ui.pages.HomePage;
import com.petrovich.ui.pages.SearchResultsPage;
import org.junit.jupiter.api.Test;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;
import com.codeborne.selenide.Condition;
import java.time.Duration;
import static com.codeborne.selenide.Selenide.$;

public class SearchTest extends BaseTest {

    @Test
    public void testSearchWithFilterAndSort() {
        String query = "Краска акриловая белая";

        // Открываем главную страницу и выполняем поиск
        SearchResultsPage page = new HomePage().open()
            .searchFor(query);

        // Применяем фильтр по цене и сортировку
        page.filterByPrice(150, 400)
            .sortByPriceAscending();

          // Проверяем, что результаты не пустые и отсортированы по
возрастанию цен в заданном диапазоне
        int[] prices = page.getAllProductPrices();
         assertTrue(prices.length > 0, "Search results should not be
empty");
        for (int i = 1; i < prices.length; i++) {
            assertTrue(prices[i] >= prices[i - 1],
                       String.format("Prices not sorted: %d >= %d?",
prices[i], prices[i - 1]));
```

```
            assertTrue(prices[i] >= 150 && prices[i] <= 400,
                               String.format("Price %d is outside range
[150,400]", prices[i]));
        }

        // Проверяем, что названия товаров содержат поисковый запрос
        List<String> titles = page.getAllProductTitles().texts();
       assertFalse(titles.isEmpty(), "Titles list should not be empty");
        for (String title : titles) {
            assertTrue(title.contains(query),
                      String.format("Title '%s' does not contain query",
title));
        }
    }
}
```

Название файла: CartTest.java

```
package com.petrovich.ui.tests;

import com.petrovich.ui.pages.ProductPage;
import com.petrovich.ui.pages.CartPage;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CartTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testAddProductToCart() {
        ProductPage p = new ProductPage().open(code);
        int before = p.getCartCount();
        p.addToCart();
        assertEquals(before+1, p.getCartCount());
        CartPage cart = new CartPage().open();
        assertTrue(cart.isProductInCart(code));
        assertEquals(1, cart.getProductQuantity(code));
    }

    @Test
    public void testChangeProductQuantity() {

        CartPage c = new CartPage().open();
        c.updateProductQuantity(code, 5);
        assertEquals(5, c.getProductQuantity(code));
        assertEquals(5*615, c.getTotalPrice());
        c.updateProductQuantity(code, 0);
        assertFalse(c.isProductInCart(code));
    }

    @Test
    public void testRemoveProductFromCart() {
        ProductPage p = new ProductPage().open(code);
        p.addToCart();
        CartPage c = new CartPage().open();
        int before = c.getCartCount();
```

```
        c.removeProduct(code);
        assertFalse(c.isProductInCart(code));
        assertEquals(before-1, c.getCartCount());
    }
}
```

Название файла: EstimateTest.java

```
package com.petrovich.ui.tests;

import com.petrovich.ui.pages.ProductPage;
import com.petrovich.ui.pages.CartPage;
import com.petrovich.ui.pages.EstimatesPage;
import com.petrovich.ui.pages.EstimatePage;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class EstimateTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testCreateEstimate() {
        String estimateName = "Тестовая смета";


        CartPage c = new CartPage().open();
        c.updateProductQuantity(code, 2);

        EstimatesPage list = new EstimatesPage().open();
        EstimatePage e = list.clickCreateNewEstimate();
        e.setName(estimateName);
        e.addProduct(code, 2);
        e.save();

        list.open();

        assertTrue(e.containsProduct(code, 2));
        assertEquals(2*615, e.getTotal());
        assertTrue(list.isEstimatePresent(estimateName));
    }
}
```

Название файла: FavoritesTest.java

```
package com.petrovich.ui.tests;

import com.petrovich.ui.pages.ProductPage;
import com.petrovich.ui.pages.FavoritesPage;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import com.codeborne.selenide.Condition;
import java.time.Duration;
import static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selectors.byText;

public class FavoritesTest extends BaseTest {
```

```
    private final String code = "104843";

    @Test
    public void testAddAndRemoveFavorite() {
          // Открываем страницу товара и получаем начальное значение
счётчика избранного
        ProductPage p = new ProductPage().open(code);
        int before = p.getFavoritesCount();

        // Добавляем в избранное и ждём обновления счётчика
        p.addToFavorites();
        $("span.favorites-count")
               .shouldHave(Condition.text(String.valueOf(before + 1)),
Duration.ofSeconds(2));
        assertEquals(before + 1, p.getFavoritesCount());

        // Переходим в избранное и ждём появления товара
        FavoritesPage fav = new FavoritesPage().open();
        $(byText(code))
            .shouldBe(Condition.visible, Duration.ofSeconds(2));
        assertTrue(fav.containsProduct(code));

        // Удаляем из избранного и ждём обновления счётчика
        p.open(code).removeFromFavorites();
        $("span.favorites-count")
                    .shouldHave(Condition.text(String.valueOf(before)),
Duration.ofSeconds(1));
        assertEquals(before, p.getFavoritesCount());

         // Ждём, пока товар исчезнет из списка избранного и проверяем
отсутствие
        FavoritesPage favAfter = fav.open();
        $(byText(code))
            .shouldNot(Condition.exist, Duration.ofSeconds(2));
        assertFalse(favAfter.containsProduct(code));
    }
}
```

Название файла: ProductPageTest.java

```
package com.petrovich.ui.tests;

import com.petrovich.ui.pages.ProductPage;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import com.codeborne.selenide.Condition;
import java.time.Duration;
import static com.codeborne.selenide.Selenide.$;

public class ProductPageTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testProductDetailsPageElements() {
        String expectedTitle = "Штукатурка гипсовая Knauf МП-75 машинная
30 кг";
```

```
       String expectedDescriptionSnippet = "Гипсовая машинная штукатурка
Knauf МП-75";

        // Открываем страницу товара
        ProductPage p = new ProductPage().open(code);

        // Ждём появления заголовка и проверяем его
        $("h1").shouldBe(Condition.visible, Duration.ofSeconds(2));
        assertEquals(expectedTitle, p.getTitle());

        // Ждём отображения цены и проверяем значение без валюты
                        $("[class*=price]").shouldBe(Condition.visible,
Duration.ofSeconds(2));
        assertEquals("615", p.getPrice().replaceAll("\\D", ""));

        // Ждём описания и проверяем, что содержится нужный фрагмент
               $("div.product-description").shouldBe(Condition.visible,
Duration.ofSeconds(2));
         assertTrue(p.getDescription().contains(expectedDescriptionSnip
pet));

       // Ждём появления изображения товара по фрагменту src и проверяем
                       $("img[src*='mp75']").shouldBe(Condition.visible,
Duration.ofSeconds(2));
        assertTrue(p.isImageDisplayed("mp75"));

        // Проверяем таблицу технических характеристик
                       $("table.tech-specs").shouldBe(Condition.visible,
Duration.ofSeconds(2)); // TODO: уточнить селектор
        assertTrue(p.hasTechnicalSpecifications());

        // Проверяем секцию "Отзывы"
                                                $("*[class*=reviews],
*:contains('Отзывы')").shouldBe(Condition.visible,
Duration.ofSeconds(2)); // TODO: уточнить селектор
        assertTrue(p.hasReviewsSection());

        // Проверяем секцию "Доставка"
                                              $("*[class*=delivery],
*:contains('Доставка')").shouldBe(Condition.visible,
Duration.ofSeconds(2)); // TODO: уточнить селектор
        assertTrue(p.hasDeliveryInfoSection());
    }
}
```