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 8 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

Шуменков А.П.

Студенты гр. 3341 _____ Анисимов Д.А.

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 8 автотестов на определенный функциональный блок выбранной системы (например, работа с постами в социальной сети). Работа ведётся в группах по два 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, 8 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 two 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. Написать 8 автоматизированных тестов для выбранной системы. Использовать технологии: Java, Selenide (Selenium), JUnit 5, Maven, логирование.
2. Организовать работу в GitHub (репозиторий на группу из 2 человек с распределением задач в README.md).
3. Оформить чеклист в виде таблицы с описанием тестов.

Для тестирования был выбран сайт СТД Петрович [1]. 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.

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".

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)".

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".

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".

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".

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.

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.

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.

ВТОРОЙ РАЗДЕЛ

2.1. Описание классов и методов

BaseElement.java

Класс: BaseElement - 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 - Представляет кнопку на странице.

Наследует: BaseElement.

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 - Поле ввода текста

Наследует: BaseElement

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 - Таблица на странице.
Наследует: BaseElement

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.

BasePage.java

Класс: BasePage - 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) - 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) - 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) - 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) - 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) - 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) - Список созданных смет.

Методы:

List<String> getEstimateIds() – получить список ID смет.

void deleteEstimate(String id) – удалить смету.

FavoritesPage.java

Класс: FavoritesPage (extends BasePage) - 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) - 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) - 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.

BaseTest.java

Класс: BaseTest - 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) - Тесты авторизации.

Методы:

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

CartTest.java

Класс: CartTest (extends BaseTest) - 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) - Тест создания сметы.

Методы:

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

FavoritesTest.java

Класс: FavoritesTest (extends BaseTest) - Тесты избранного.

Методы:

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

ProductPageTest.java

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

Методы:

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

SearchTest.java

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

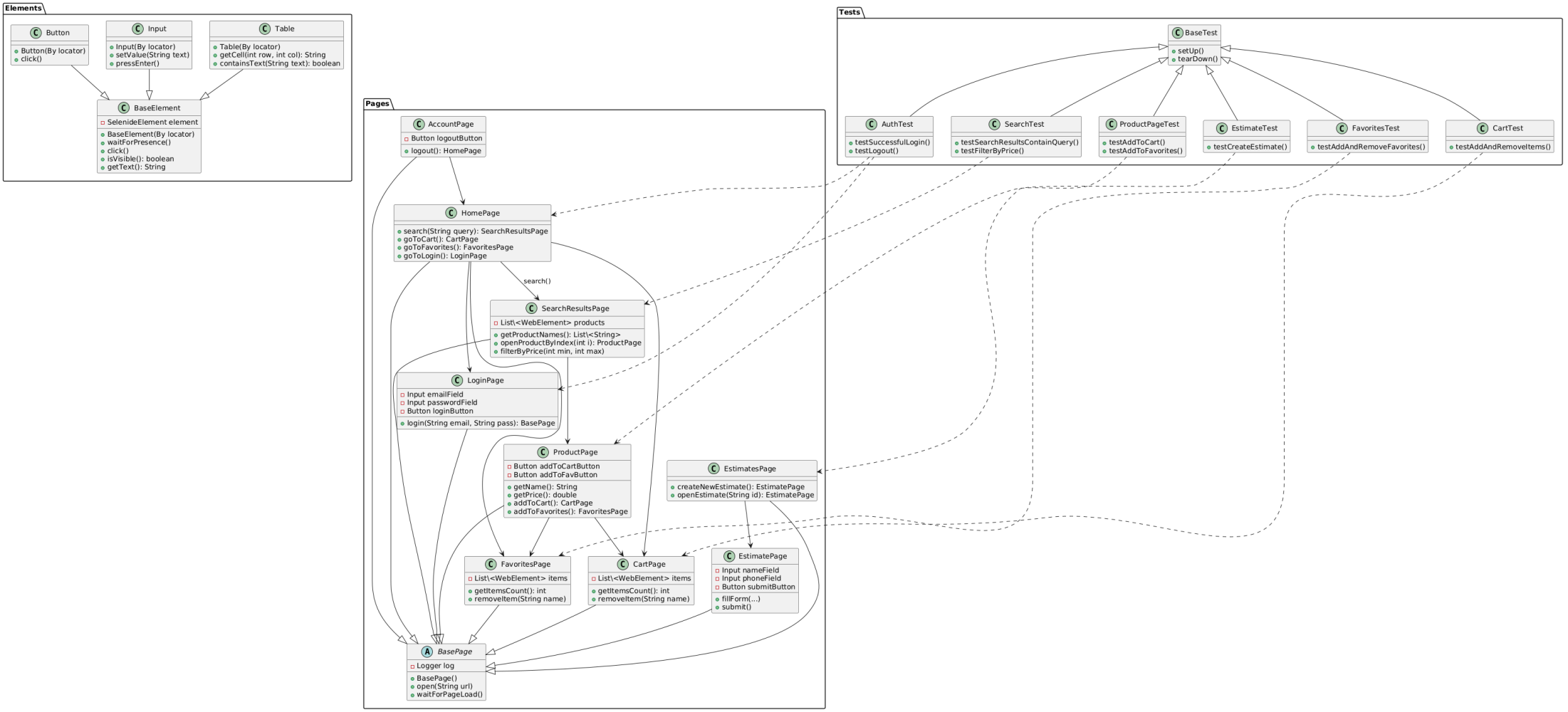
Методы:

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

Разработанный программный код см. в приложении А.

2.2. UML-диаграмма

Нами была создана UML-диаграмма (рис 1.), с помощью сайта [2]. Для этого был написан код (см. приложение А).



3. ТРЕТИЙ РАЗДЕЛ

1.1. Тестирование одного теста

Для описания выбран тест «Авторизация»

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

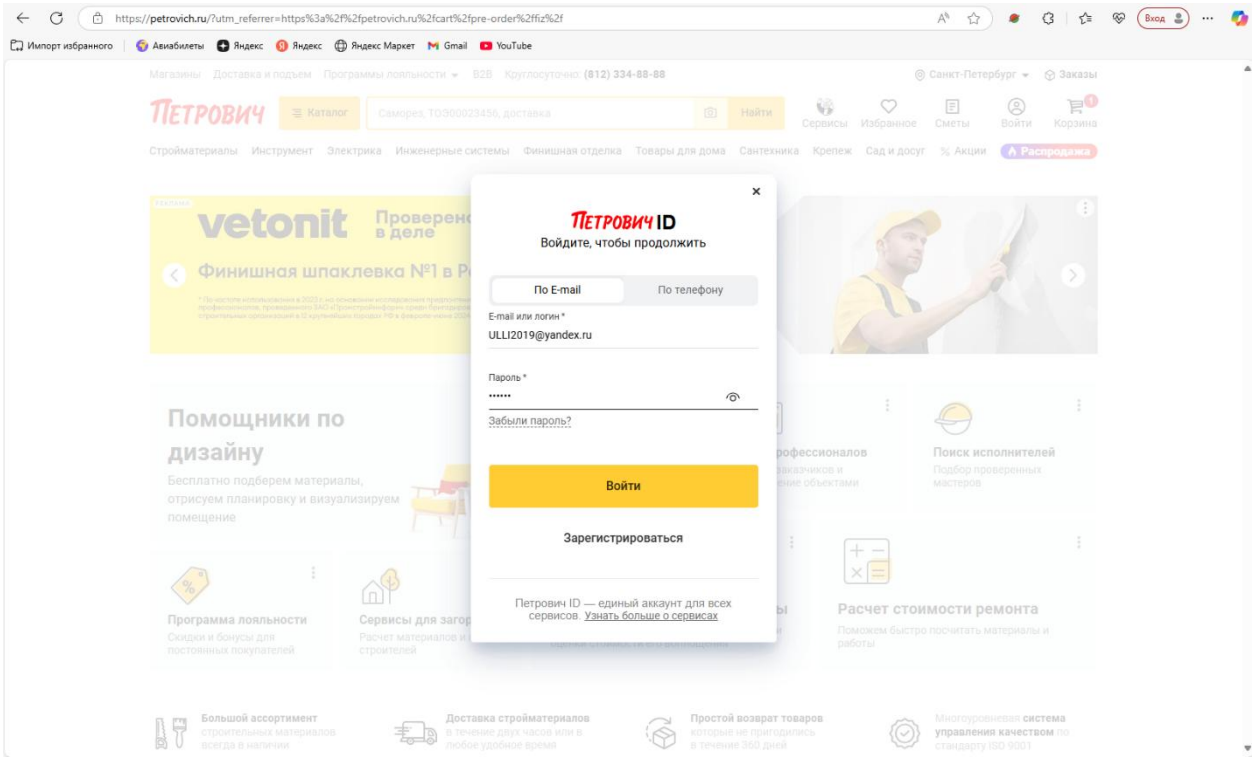


Рисунок 2 - Авторизация

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

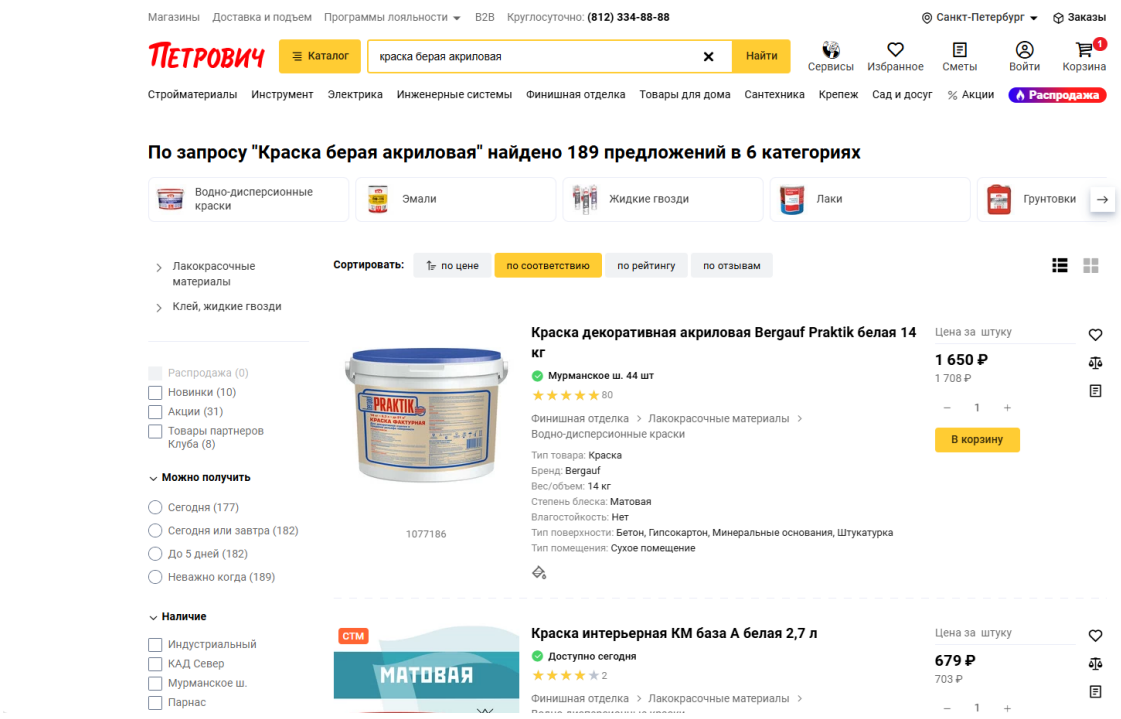


Рисунок 3 – Ввод товара в строку поиска

3. Программа выставляет диапазон цены (рис.4).

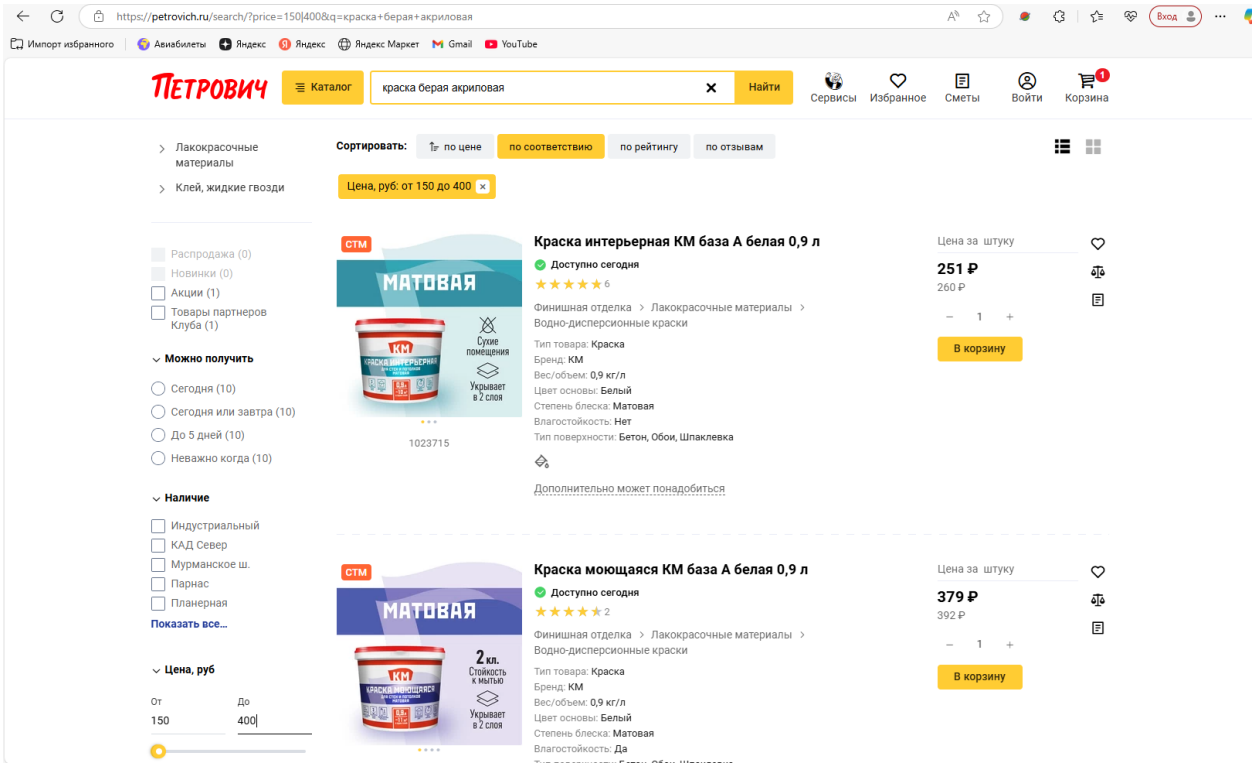


Рисунок 4 – Высиавление диапазона

ЗАКЛЮЧЕНИЕ

В ходе летней практики по автоматизации UI-тестирования веб-приложения «Петрович» 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, для ведения кода — GitHub (коллективная работа двух студентов). Логирование выполнения тестов обеспечило наглядность отчётов и упростило отладку.

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СТД Петрович https://petrovich.ru/
2. www.plantuml.com – сайт для создания Uml – диаграмм.

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;

/**
 * Базовый абстрактный класс-обёртка над {@link SelenideElement}.
 * Содержит общие методы ожидания, проверки видимости, клика и получения
текста,
 * а также логгирование действий с элементами.
 */
public abstract class BaseElement {

    /** Оборачиваемый элемент Selenide. */
    protected final SelenideElement element;

    /** Логгер для всех наследников. */
    protected final Logger logger = LoggerFactory.getLogger(getClass());

    /**
     * Проверяет, отображается ли элемент на странице.
     *
     * <p>Метод ждёт видимости элемента до 2 секунд. Если за это время
     * элемент не становится видимым, возвращается {@code false} без
     * выбрасывания исключения.</p>
     *
      * @return {@code true}, если элемент виден; {@code false} — в
противном случае
     */
    public boolean isDisplayed() {
        try {
            element.shouldBe(Condition.visible, Duration.ofSeconds(2));
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    /**
     * Кликает по элементу.
     *
      * <p>Перед кликом метод ждёт, пока элемент станет видимым (до 2
секунд),
     * а затем логирует факт клика.</p>
     */
    public void click() {
```

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

    /**
     * Возвращает текст элемента.
     *
     * <p>Сначала ожидает видимости элемента до 2 секунд, затем
     * считывает и возвращает текстовое содержимое.</p>
     *
     * @return текст, содержащийся в элементе
     */
    public String getText() {
        element.shouldBe(Condition.visible, Duration.ofSeconds(2));
        return element.getText();
    }

    /**
     * Конструктор базового элемента.
     *
     * @param element экземпляр {@link SelenideElement}, который будет
обёрнут
     */
    protected BaseElement(SelenideElement element) {
        this.element = element;
    }
}
```

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 {

    /**
     * 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");
    }

    private Input(SelenideElement element) {
        super(element);
    }
}
```

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.$;

/**
 * Обёртка над HTML-таблицей или любым контейнером с табличными данными.
 * Позволяет удобно получать количество строк и проверять,
 * содержит ли таблица заданный текст.
 */
public class Table extends BaseElement {

    /**
        *  Создаёт  {@code  Table},  оборачивая  переданный  {@link
SelenideElement}.
     *
     * @param element элемент, представляющий таблицу
     */
    public Table(SelenideElement element) {
        super(element);
    }

    /**
     * Быстрый статический конструктор по CSS-селектору.
     *
     * @param cssSelector CSS-селектор таблицы
     * @return объект {@code Table}, привязанный к найденному элементу
     */
    public static Table byCss(String cssSelector) {
        return new Table($(cssSelector));
    }
```

```
    /**
     * Возвращает количество строк таблицы.
     *
      * <p>Считает элементы с ролями {@code [role=row]} или {@code
tr}.</p>
     *
     * @return число строк
     */
    public int getRowCount() {
        return element.$$("[role=row], tr").size();
    }

    /**
     * Проверяет, содержит ли таблица указанный текст.
     *
     * @param text искомая строка
     * @return {@code true}, если текст найден; иначе {@code false}
     */
    public boolean isTextContains(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;

/**
 * Базовый абстрактный класс для всех Page Object-страниц.
 * <p>
 * Содержит общие утилиты для:
 * <ul>
 *      <li>Получения текущего количества товаров в корзине и
избранном.</li>
 *   <li>Открытия страниц по относительному URL с логгированием.</li>
 * </ul>
 * Каждый наследник автоматически получает настроенный {@link Logger}.
 */
public abstract class BasePage {

    /** Логгер конкретного класса-наследника. */
    protected final Logger logger = LoggerFactory.getLogger(getClass());

    /**
     * Возвращает количество товаров в корзине.
     *
      * <p>Метод читает текст из элемента <code>&lt;span class="cart-
count"&gt;</code>.
     * Если текст пустой, считается, что товаров нет.</p>
     *
```

```
     * @return число товаров в корзине
     */
    public int getCartCount() {
        String text = Selenide.$("span.cart-count").getText();
        return text.isEmpty() ? 0 : Integer.parseInt(text.trim());
    }

    /**
     * Возвращает количество товаров в избранном.
     *
         *   <p>Берёт   текст   из   <code>&lt;span   class="favorites-
count"&gt;</code>.
     * Пустая строка трактуется как <code>0</code>.</p>
     *
     * @return число товаров в избранном
     */
    public int getFavoritesCount() {
        String text = Selenide.$("span.favorites-count").getText();
        return text.isEmpty() ? 0 : Integer.parseInt(text.trim());
    }

    /**
     * Открывает страницу по относительному URL.
     *
     * <p>Автоматически добавляет ведущий слэш при необходимости,
     * записывает действие в лог и использует {@link Selenide#open}.</p>
     *
       *  @param  relativeUrl  путь  вида  <code>"/profile"</code>  или
<code>"profile"</code>
     */
    protected void openPage(String relativeUrl) {
        logger.info("Opening page: {}", relativeUrl);
        String url = relativeUrl.startsWith("/") ? relativeUrl : "/" +
relativeUrl;
        Selenide.open(url);
    }
}
```

Название файла: HomePage.java

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

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

/**
 * Страница главного экрана сайта.
 */
public class HomePage extends BasePage {
     private  static  final  String  SEARCH_INPUT ="#page-header-search >
div > div > input";
    private static final String LOGIN_LINK ="Войти";

    private final Input  searchInput = Input.byCss(SEARCH_INPUT);
```

```
    private final Button loginLink   = Button.byText(LOGIN_LINK);

    /**
     * Открывает главную страницу.
     */
    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() {
        return loginLink.isDisplayed();
    }
}
```

Название файла: LoginPage.java

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

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

/**
 * Страница логина пользователя.
 */
public class LoginPage extends BasePage {
    // XPaths элементов страницы логина вынесены в константы
        private   static   final   String   XPATH_USERNAME_INPUT   =
"/html/body/div[6]/div/div/div[2]/div[2]/div/div[2]/div[1]/form/div[1]
/div/div/label/input";
```

```
        private static final String XPATH_PASSWORD_INPUT =
"/html/body/div[6]/div/div/div[2]/div[2]/div/div[2]/div[1]/form/div[2]
/div/div/label/input";
        private static final String XPATH_LOGIN_BUTTON =
"/html/body/div[6]/div/div/div[2]/div[2]/div/div[2]/div[1]/form/button
[1]";

    // Элементы страницы логина
            private final Input usernameInput =
Input.byXpath(XPATH_USERNAME_INPUT);
            private final Input passwordInput =
Input.byXpath(XPATH_PASSWORD_INPUT);
            private final Button loginButton =
Button.byXpath(XPATH_LOGIN_BUTTON);

    /**
     * Выполняет вход в систему и возвращает страницу аккаунта.
     * @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();
        return new AccountPage();
    }
}
```

Название файла: AccountPage.java

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

import com.petrovich.ui.elements.Button;

/**
 * Страница «Личный кабинет».
 * <p>
 * Предоставляет функции выхода из системы и проверки,
 * что пользователь авторизован.
 * </p>
 */
public class AccountPage extends BasePage {
    // Константа для текста кнопки выхода
    private static final String LOGOUT_BUTTON_TEXT = "Выход";

    // Элемент кнопки «Выход»
            private final Button logoutButton =
Button.byText(LOGOUT_BUTTON_TEXT);

    /**
     * Выполняет выход пользователя из аккаунта.
     *
```

```
     * <p>Метод кликает по кнопке «Выход», пишет запись в лог
     * и возвращает объект главной страницы.</p>
     *
     * @return {@link HomePage} — объект главной страницы после выхода
     */
    public HomePage logout() {
        logger.info("Logging out");
        logoutButton.click();
        return new HomePage();
    }

    /**
     * Проверяет, отображается ли кнопка «Выход»,
     * тем самым косвенно подтверждая, что пользователь авторизован.
     *
     * @return {@code true}, если кнопка видна; иначе {@code false}
     */
    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.$;

/**
 * Страница товара.
 */
public class ProductPage extends BasePage {

    private static final String ADD_TO_CART_TEXT    = "В корзину";
    private static final String FAVORITE_BUTTON_TEXT = "Избранное";
     private static final String TECH_SPECS_TABLE_CSS = "table.tech-
specs";

    // Элементы страницы товара
              private      final      Button      addToCartButton      =
Button.byText(ADD_TO_CART_TEXT);
              private      final      Button      favoriteButton       =
Button.byText(FAVORITE_BUTTON_TEXT);
            private     final     Table      techSpecsTable           =
Table.byCss(TECH_SPECS_TABLE_CSS);

    /**
     * Открыть страницу товара по артикулу.
     */
    public ProductPage open(String productCode) {
```

```
        openPage("/product/" + productCode);
        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).
     */
    public String getTitle() {
        SelenideElement title = $("h1").shouldBe(Condition.visible);
        return title.getText();
    }

    /**
     * Текст цены.
     */
    public String getPrice() {
                                    SelenideElement        price        =
$("[class*=price]").shouldBe(Condition.visible);
        return price.getText();
    }

    /**
     * Описание товара.
     */
    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".
     */
    public boolean hasReviewsSection() {
        return $("*[class*=reviews], *:contains('Отзывы')").exists();
    }

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

    /**
     * Видна ли картинка с подстрокой imageName в пути.
     */
    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 {
```

```
          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");

          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");

    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));

        $("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();
        $("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() {
        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")

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

     private static final String QTY_INPUT_XPATH =
         "/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";
    /** Открыть корзину */
    public CartPage open() {
        openPage("/cart");
        return this;
    }

    /** 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(QTY_INPUT_XPATH);

        // 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 value = findCartItem(code)
                    .$("input.quantity")
                    .getValue();
        return Integer.parseInt(value);
    }

    /** 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 total = $("span.cart-total")
                      .getText()
                      .replaceAll("\\D", "");
        return total.isEmpty() ? 0 : Integer.parseInt(total);
    }

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

}
```

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 {
    private static final String  CREATE_BTN = "Создать новую смету";

    private final Button createBtn = Button.byText(CREATE_BTN);

    /**
```

```
     * Открывает страницу со списком смет.
     */
    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) {
        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());

    private static final String NAME_INPUT_NAME        = "estimateName";
    private static final String ADD_BUTTON_TEXT         = "Добавить";
    private static final String ARTICLE_INPUT_NAME      = "article";
    private static final String QUANTITY_INPUT_NAME     = "quantity";
    private static final String SAVE_BUTTON_TEXT        = "Сохранить";

    // Элементы формы сметы
    private final Input  nameInput     = Input.byName(NAME_INPUT_NAME);
    private final Button addBtn        = Button.byText(ADD_BUTTON_TEXT);
              private      final      Input       articleInput       =
Input.byName(ARTICLE_INPUT_NAME);
              private      final      Input       quantityInput      =
Input.byName(QUANTITY_INPUT_NAME);
    private final Button saveBtn       = Button.byText(SAVE_BUTTON_TEXT);
```

```
    /** 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 {
    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) {
        return $(byText(code)).exists();
    }
}
```

Название файла: BaseTest.java

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

import com.codeborne.selenide.Configuration;
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("/");
        }
    }

    @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.*;

/**
 * Тест-класс для проверки сценария авторизации и выхода из системы.
 */
public class AuthTest extends BaseTest {

    /** Тестовый e-mail для входа в систему. */
    private static final String TEST_LOGIN    = "ULLI2019@yandex.ru";
    /** Тестовый пароль для входа в систему. */
    private static final String TEST_PASSWORD = "password";

    /**
     * Выполняет полный сценарий:
     * 1. Открытие главной страницы.
     * 2. Переход на страницу логина.
     * 3. Авторизация с заданными данными.
     * 4. Проверка успешного входа.
     * 5. Выход из аккаунта.
     * 6. Проверка возврата на главную.
     */
    @Test
    public void testLoginLogout() {
        // 1. Открываем главную страницу
        HomePage home = new HomePage().open();

        // 2. Переходим на страницу авторизации
        LoginPage login = home.goToLoginPage();

       // 3. Вводим логин и пароль, кликаем "Войти" и переходим в личный
кабинет
        AccountPage account = login.login(TEST_LOGIN, TEST_PASSWORD);

         // 4. Проверяем, что кнопка "Выход" отображается, значит вход
выполнен
           assertTrue(account.isLoggedIn(), "Пользователь не вошёл в
систему");

        // 5. Нажимаем "Выход" и получаем главную страницу
        HomePage after = account.logout();

          // 6. Проверяем, что ссылка "Войти" снова видна на главной
странице
            assertTrue(after.isLoginLinkVisible(), "Ссылка «Войти» не
отображается после выхода");
    }
}
```

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.*;

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: CartRemoveTest.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.*;

/**
 * Тест удаления товара из корзины и проверки обновления счётчика.
 */
public class CartRemoveTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testRemoveProductFromCart() {
        // Добавляем товар, чтобы можно было его удалить
        ProductPage product = new ProductPage().open(code);
        product.addToCart();
        CartPage cart = new CartPage().open();
        int before = cart.getCartCount();

        // Удаляем товар и проверяем, что он исчез и счётчик уменьшился
        cart.removeProduct(code);
        assertFalse(cart.isProductInCart(code));
        assertEquals(before - 1, cart.getCartCount());
    }
}
```

Название файла: CartAddTest.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.*;

/**
 * Тест добавления товара в корзину и проверка счётчика.
 */
public class CartAddTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testAddProductToCart() {
        // Открываем страницу товара и фиксируем текущее число товаров
в корзине
        ProductPage product = new ProductPage().open(code);
        int before = product.getCartCount();

        // Добавляем товар и проверяем счётчик в шапке
        product.addToCart();
        assertEquals(before + 1, product.getCartCount());

        // Переходим в корзину и проверяем наличие и количество товара
        CartPage cart = new CartPage().open();
        assertTrue(cart.isProductInCart(code));
        assertEquals(1, cart.getProductQuantity(code));
    }
```

```
}
```

Название файла: CartUpdateTest.java

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

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

/**
 * Тест изменения количества товара в корзине и проверки корректности
суммы.
 */
public class CartUpdateQuantityTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testChangeProductQuantity() {
        CartPage cart = new CartPage().open();

        // Изменяем количество товара и проверяем корректность значения
и суммы
        cart.updateProductQuantity(code, 5);
        assertEquals(5, cart.getProductQuantity(code));
        assertEquals(5 * 615, cart.getTotalPrice());

        // Удаляем товар, установив количество в 0
        cart.updateProductQuantity(code, 0);
        assertFalse(cart.isProductInCart(code));
    }
}
```

Название файла: EstimateTest.java

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

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.*;

/**
 * Тест для проверки создания и проверки новой сметы.
 */
public class EstimateTest extends BaseTest {
    /** Код товара для добавления в смету. */
    private final String code = "104843";

    @Test
    public void testCreateEstimate() {
        // Название новой сметы
        String estimateName = "Тестовая смета";
```

```
        // 1. Устанавливаем количество товара в корзине
        CartPage cart = new CartPage().open();
        cart.updateProductQuantity(code, 2);

        // 2. Открываем список смет и переходим к созданию новой
        EstimatesPage list = new EstimatesPage().open();
        EstimatePage estimate = list.clickCreateNewEstimate();

        // 3. Заполняем форму: имя сметы и добавляем товар
        estimate.setName(estimateName);
        estimate.addProduct(code, 2);

        // 4. Сохраняем смету и возвращаемся к списку
        estimate.save();
        list.open();

         // 5. Проверяем, что в созданной смете есть нужный товар с правильным количеством
       assertTrue(estimate.containsProduct(code, 2), "Смете не добавлен ожидаемый товар");

        // 6. Проверяем корректность общей суммы
        assertEquals(2 * 615, estimate.getTotal(), "Неправильная общая сумма в смете");

        // 7. Убеждаемся, что смета отображается в списке
         assertTrue(list.isEstimatePresent(estimateName), "Новая смета не найдена в списке");
    }
}
```

Название файла: FavoritesAddTest.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 static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selectors.byText;
import com.codeborne.selenide.Condition;
import java.time.Duration;

/**
 * Тесты, связанные с добавлением товара в избранное и проверкой счётчика.
 */
public class FavoritesAddTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testAddFavorite() {
```

```
          // Открываем страницу товара и получаем начальное значение
счётчика
        ProductPage product = new ProductPage().open(code);
        int before = product.getFavoritesCount();

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

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

Название файла: FavoritesRemooveTest.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 static com.codeborne.selenide.Selenide.$;
import static com.codeborne.selenide.Selectors.byText;
import com.codeborne.selenide.Condition;
import java.time.Duration;

/**
 * Тесты, связанные с удалением товара из избранного и проверкой
счётчика.
 */
public class FavoritesRemoveTest extends BaseTest {
    private final String code = "104843";

    @Test
    public void testRemoveFavorite() {
        // Убедимся, что товар в избранном (или добавим его)
        ProductPage product = new ProductPage().open(code);
        if (product.getFavoritesCount() == 0) {
            product.addToFavorites();
            new FavoritesPage().open(); // обновляем страницу
        }
        FavoritesPage fav = new FavoritesPage().open();
        int before = product.getFavoritesCount();

        // Удаляем из избранного
        product.open(code).removeFromFavorites();
        $("span.favorites-count")
```

```
                .shouldHave(Condition.text(String.valueOf(before - 1)),
Duration.ofSeconds(2));
        assertEquals(before - 1, product.getFavoritesCount());

        // Проверяем, что товар исчез из списка избранного
        FavoritesPage favAfter = fav.open();
        $(byText(code))
            .shouldNot(Condition.exist, Duration.ofSeconds(2));
        assertFalse(favAfter.containsProduct(code));
    }
}
```

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 product = new ProductPage().open(code);
        int before = product.getFavoritesCount();

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

```
        assertEquals(before, product.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());
    }
}
```