МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: UI-тестирование онлайн-магазина Tramp Sport

Студент гр. 2303 _____ Борисов В.О.

Студент гр. 2304 _____ Карпунин К.А.

Студент гр. 2303 _____ Царегородцев М.А.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2024

ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ

Студент Борисов В.О.

Группа 2303

Студент Карпунин К.А.

Группа 2304

Студент Царегородцев М.А.

Группа 2303

Тема практики: UI-тестирование онлайн-магазина *Tramp Sport*.

Задание на практику: Тестирование пользовательского интерфейса онлайн-магазина *Tramp Sport* (https://tramp-sport.ru/).

Сроки прохождения практики: 27.06.2024 – 09.07.2024

Дата сдачи отчета: 09.07.2024

Дата защиты отчета: 09.07.2024

Студент гр. 2303		Борисов В.О.
Студент гр. 2304	_____	Карпунин К.А.
Студент гр. 2303	_____	Царегородцев М.А.
Руководитель		Шевелева А.М.
	_____	

АННОТАЦИЯ

В данном проекте осуществляется *UI* тестирование онлайн магазина *Tramp Sport* (*https://tramp-sport.ru/*). В качестве инструментов используются язык программирования *Java*, система сборки *Maven*, фреймворки *JUnit* и *Selenide*, среда разработки *IntelliJ Idea Community/Professional Edition*. Тестируется функционал работы пользователя с корзиной и избранным, осуществление сортировки и фильтрации каталогов, а также система пользовательской авторизации. Проект разрабатывался с использованием системы контроля версий *git* и платформы *GitHub*. Для документирования кода был выбран генератор документации *JavaDoc*.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Изучение *UI*-тестирования с использованием *Java*, *Maven*, *Junit*, *Selenide*. Реализация веб-тестирования функционала онлайн-магазина *Tramp Sport* (*https://tramp-sport.ru/*).

Задачи

- Создание системы классов элементов страниц онлайн-магазина
- Создание системы классов страниц онлайн-магазина
- Определение различных блоков/модулей тестов, направленных на исследование функционала онлайн-магазина
- Реализация тестов, рассматривающих различные случаи работы онлайн-магазина
- Реализация системы логирования
- Создание документации к реализованным системам и тестам
- Создание *UML*-диаграмм к реализованным системам и тестам

1. ОБЩАЯ ЧАСТЬ

В данном проекте представлено тестирование функций онлайн-магазина. Реализованный программный код находится в находится в репозитории на *GitHub,* ссылка на который лежит в списке использованных источников. Чек-лист тестов находится в Приложении А проекта. UML-диаграммы реализованных классов находятся в Приложении Б.

Проект состоит из следующих пакетов:

1. *elements* - пакет с классами, соответствующими элементам на странице. В данных классах действия происходят на уровне взаимодействия с *xpath*-ми и объектами *SelenideElement*.
2. *pages* - пакет с классами, соответствующими страницам. Каждый класс отражает свою страницу, его методы - действия на этой странице, которые выполняются при тестировании. В данном пакете сосредоточены основные методы, используемые при тестировании.
3. *tests* - пакет с классами, соответствующими тестам. Каждый класс отвечает за свой тест. Классы также сгруппированы по отдельным пакетам (разделение на блоки тестов). Используя методы страниц, осуществляют непосредственное тестирование функций онлайн-магазина.
4. *utils* - пакет с классами, не имеющими прямого отношения к страницам, элементам и тестированию. Содержит классы, обеспечивающие вспомогательный функционал.

Кроме вышеупомянутых пакетов, проект содержит класс *Launcher*, ответственный за запуск тестов, а также директорию *resources*, в которой хранятся конфигурационные файлы.

2. РЕАЛИЗАЦИЯ ОБЩИХ КЛАССОВ

2.1. Элементы

- *class BaseElement* - базовый класс элемента страницы, от него наследуются все остальные классы элементов страниц. Содержит методы, общие для всех элементов страниц, а также *protected* поле для хранения элемента страницы типа *SelenideElement*.

Поля класса:

1. *protected final SelenideElement element* - поле для хранения элемента страницы.

Методы класса:

1. *public BaseElement(String xpath)* - метод-конструктор класса, принимает на вход строку с *xpath*-м элемента страницы.
2. *public BaseElement(SelenideElement element)* - метод-конструктор класса, принимает на вход элемент страницы типа *SelenideElement*.
3. *public String describe()* - метод, который возвращает описание элемента страницы. Оно включает в себя блок, теги и содержимое элемента.
4. *public boolean exists()* - метод, который проверяет, что элемент существует на странице.
5. *public boolean isDisplayed()* - метод, который проверяет, доступен ли элемент для просмотра.

- *class ButtonElement* - класс, соответствующий кнопке на странице. Содержит методы для работы с элементом кнопки. Наследуется от класса *BaseElement*.

Методы класса:

1. *public ButtonElement(String xpath)* - метод-конструктор класса, принимает на вход строку с *xpath*-ом элемента на странице. Использует конструктор родительского класса.
2. *public void click()* - метод, осуществляющий нажатие кнопки.

- *class TextElement* - класс, соответствующий элементу с текстом. Содержит методы для работы с элементом с текстом. Наследуется от класса *BaseElement*.

Методы класса:

1. *public TextElement(String xpath)* - метод-конструктор класса, принимает на вход строку с *xpath*-ом элемента на странице. Использует конструктор родительского класса.
2. *public String text()* - метод, возвращающий видимый текст элемента страницы.

- *class TextFieldElement* - класс, соответствующий полю для ввода текста на странице. Содержит методы для работы с текстовым полем. Наследуется от класса *BaseElement*.

Методы класса:

1. *public TextFieldElement(String xpath)* - метод-конструктор класса, принимает на вход строку с *xpath*-ом элемента на странице. Использует конструктор родительского класса.

2. *public void setValue(String value)* - метод, который вводит передаваемое значение *value* в текстовое поле.
3. *public void pressEnter()* - метод, который выполняет нажатие клавиши *Enter*.

- *class DescribableElement* - класс, соответствующий объекту на странице, имеющим описание. Наследуется от класса *BaseElement*.

Методы класса:

1. *public DescribableElement(String xpath)* - метод-конструктор класса, принимает на вход строку с *xpath*-м элемента на странице. Использует конструктор родительского класса, принимающий xpath.
2. *public DescribableElement(SelenideElement element)* - метод-конструктор класса, принимает на вход элемент страницы типа *SelenideElement*. Использует конструктор родительского класса.
3. *public static ArrayList<DescribableElement> getAllElements(String xpath)* - метод, который создает массив элементов по *xpath-y*, то есть берет все элементы, удовлетворяющие данному *xpath*-у, после чего каждый из элементов типа *SelenideElement* делает элементом класса *DescribableElement*.

2.2. Страницы

- *class BasePage* - базовый класс страницы, от него наследуются все остальные классы страниц. Содержит методы, общие для всех страниц, а также приватное поле для хранения *URL*-адреса страницы.

Поля класса:

1. *private final String pageAddress* - поле для хранения *URL*-адреса страницы.

Методы класса:

1. *public BasePage(String pageAddress)* - метод-конструктор класса, принимает на вход строку с *URL*, записывает ее в поле.
2. *public String openPage()* - метод, который открывает страницу класса по ее адресу, хранящемуся в соответствующем поле. Возвращает строку с адресом страницы.
3. *protected DescribableElement getDescribableElement(String xpath)* - метод, который создает экземпляр класса *DescribableElement* и возвращает его. Принимает на вход *xpath* нужного элемента.
4. *protected ButtonElement getButtonElement(String xpath)* - метод, который создает экземпляр класса *ButtonElement* и возвращает его. Принимает на вход *xpath* нужного элемента.
5. *protected TextElement getTextElement(String xpath)* - метод, который создает экземпляр класса *TextElement* и возвращает его. Принимает на вход *xpath* нужного элемента.
6. *protected TextFieldElement getTextFieldElement(String xpath)* - метод, который создает экземпляр класса *TextFieldElement* и возвращает его. Принимает на вход *xpath* нужного элемента.

- *ProductSetPage* - класс, соответствующий странице, содержащей набор товаров. Наследуется от класса *BasePage*.

Поля класса: текстовая константа, являющаяся *xpath*-м соответствующего элемента на странице.

1. *private static final String EMPTINESS_MESSAGE_XPATH*

Методы класса:

1. *public ProductSetPage(String pageAddress)* - метод-конструктор класса. Принимает на вход строку с *URL*, записывает ее в поле.

2. *protected boolean isEmpty(String message)* - метод, проверяющий отсутствие товаров на странице. Принимает на вход сообщение, соответствующее странице без товаров. Возвращает *true*, если страница пустая, иначе - *false*.

3. *protected ProductPage getFirstProductPage(String productXpath) throws Exception* - метод, возвращающий страницу первого товара. Принимает на вход *xpath* элемента товара. Возвращает массив страниц товаров каталога. Выбрасывает исключение, если у элемента товара нет *href* ссылки на страницу товара.

4. *protected ArrayList<ProductPage> getProductPages(String ProductXpath) throws Exception* - метод, формирующий массив страниц товаров каталога. Принимает на вход *xpath* элемента товара. Возвращает массив страниц товаров каталога. Выбрасывает исключение, если у элемента товара нет *href* ссылки на страницу товара.

- *class ProductPage* - класс, соответствующий странице товара на сайте.

  Наследуется от класса *BasePage*. Содержит методы работы с элементами страницы товара.

Поля класса: текстовые константы, являющиеся *xpath*-ми соответствующих элементов на странице.

1. private final static String PRODUCT_COLOR_XPATH

2. private final static String PRODUCT_ARTICLE_XPATH

3. private final static String ADD_TO_WISHLIST_BUTTON_XPATH

4. private final static String ADD_TO_CART_BUTTON_XPATH

5. private final static String PRODUCT_SIZE_BUTTON_XPATH

6. private final static String CONFIRM_BUTTON_XPATH

7. private final static String PRODUCT_TYPE_XPATH

Методы класса:

1. *public ProductPage(String pageAddress)* - метод-конструктор класса, использует конструктор родительского класса, на вход которому подается адрес страницы товара *pageAddress*.
2. *public String getColor() throws Exception* - метод, возвращающий цвет товара класса *ProductPage.* Выбрасывает исключение в случае ошибки.
3. *public boolean isSpecificType(String type)* - метод класса, проверяющий тип товара. Принимает на вход тип, с которым нужно сравнить тип текущего товара. Возвращает *true* если товар соответствующего типа, иначе вернет *false*.
4. *public String getArticle()* - метод класса для получения артикула товара. Ничего не принимает на вход. Возвращает артикул товара.
5. *public void clickFavouritesButton()* - метод класса, осуществляющий нажатие на кнопку "Добавить в избранное" на странице товара.

6. *public void addToShoppingCart()* - метод класса, осуществляющий нажатие на кнопку "Добавить в корзину" на странице товара.

- *class CartPage* - класс, соответствующий странице корзины пользователя.

  Наследуется от класса *ProductSetPage*. Содержит методы работы с элементами страницы корзины.

  Поля класса: текстовые константы, являющиеся *xpath*-ми соответствующих элементов на странице, и сообщения для их поиска на странице.
  1. private static final String EMPTINESS_MESSAGE

  2. private static final String AMOUNT_XPATH

  3. private static final String INCREASE_AMOUNT_BUTTON_XPATH

  4. private static final String DELETE_BUTTON_XPATH

  5. private static final String PRODUCT_XPATH

  Методы класса:

1. *public CartPage() throws IOException* - метод-конструктор класса, использует конструктор родительского класса, на вход которому подается *URL*-адрес данной страницы.
2. *public int getAmountOfFirstProduct() throws Exception* - метод класса, возвращающий количество экземпляров первого товара в корзине. Выбрасывает исключение в случае, если у товара нет значения количества. Создает и использует экземпляр класса *DescribableElement* с помощью метода *getDescribableElement()*.

3. *public void increaseAmountOfFirstProduct(int count)* - метод класса, увеличивающий количество экземпляров первого товара в корзине. Принимает на вход число, соответствующее увеличению. Создает и использует экземпляр класса *ButtonElement* с помощью метода *getButtonElement()*.

4. *public void removeFirstProduct()* - метод класса, удаляющий первый товар из корзины. Создает и использует экземпляр класса *ButtonElement* с помощью метода *getButtonElement()*.

5. *public boolean isEmpty()* - метод класса, проверяющий отсутствие товаров в корзине. Использует метод родительского класса, передавая в него нужное сообщение для поиска на странице.

6. *public ProductPage getFirstProductPage() throws Exception* - метод, возвращающий страницу первого товара. Выкидывает исключение, что у элемента товара нет *href* ссылки на страницу товара. Использует метод родительского класса, передавая в него нужный *xpath* элемента.

- *class WishlistPage* - класс, соответствующий пользовательской странице избранного. Наследуется от класса *ProductSetPage*. Содержит методы работы с элементами страницы избранного.

  Поля класса: текстовые константы, являющиеся *xpath*-ми соответствующих элементов на странице, и сообщения для их поиска на странице.

  1. private static final String EMPTINESS_MESSAGE

  2. private static final String PRODUCT_XPATH

  Методы класса:

1. *public WishlistPage() throws IOException* - метод-конструктор класса, использует конструктор родительского класса, на вход которому подается URL данной страницы.
2. *public boolean isEmpty()* - метод класса, проверяющий отсутствие товаров в избранном. Использует метод родительского класса, передавая в него нужное сообщение для поиска на странице.
3. *public ProductPage getFirstProductPage() throws Exception* - метод, возвращающий страницу первого товара. Выбрасывает исключение у элемента товара нет *href* ссылки на страницу товара

- *class LoginPage* - класс, соответствующий странице авторизации пользователя.

  Наследуется от класса *BasePage*. Содержит методы работы с элементами страницы авторизации.

  Поля класса: текстовые константы, являющиеся *xpath*-ми соответствующих элементов на странице.

1. *private final static String EMAIL_TEXTFIELD_XPATH*
2. *private final static String PASSWORD_TEXTFIELD_XPATH*
3. *private final static String SUBMIT_BUTTON_XPATH*
4. *private final static String WARNING_WINDOW_XPATH*

  Методы класса:

1. *public LoginPage() throws IOException* - метод-конструктор класса, использует конструктор родительского класса, на вход которому подается *URL*-адрес данной страницы.
2. *public void authorize(String email, String password)* - метод, который авторизует пользователя по входным данным: почте *email* и паролю *password*.
3. *public boolean hasWarningMessage()* - метод класса, проверяющий существование сообщения об ошибке. Возвращает *true* при наличии сообщения об ошибке на странице, иначе вернет *false*.

- *class CataloguePage* - класс, соответствующий странице каталога на сайте. Наследуется от класса *ProductSetPage*. Содержит методы работы с элементами страницы каталога.

  Поля класса: текстовые константы, являющиеся *xpath*-ми соответствующих элементов на странице, и константа нуля.

  1. private static final String FILTER_BUTTON_XPATH

  2. private static final String PRICE_SORT_BUTTON_XPATH

  3. private static final String PRICE_TEXTFIELD_XPATH

  4. private static final String PRODUCTS_LIMIT_XPATH

  5. private static final String PRODUCT_XPATH

  6. private static final String PRODUCTS_LIMIT_PAGES_XPATH

  7. private static final String PRODUCT_PRICE_XPATH

  8. private static final Double DOUBLE_NULL

Методы класса:

1. *public CataloguePage(String pageAddress)* - метод-конструктор класса, использующий конструктор родительского класса, на вход которому подается *URL*-адрес данной страницы.
2. *public void chooseFilter(String filterTitle)* - метод, который выбирает соответствующий фильтр в меню фильтров каталога.
3. *public void sortProductsByPrice()* - метод класса, который в меню каталога “Сортировать по:” выбирает сортировку по возрастанию цены.
4. *public void setPriceLimit(int limit)* - метод класса, устанавливающий в соответствующем меню каталога ограничение по цене товаров.
5. *public int getNumberOfProductsLimit()* - метод, возвращающий текущее ограничение на количество товаров на одной странице каталога.
6. *public int getNumberOfProducts()* - метод класса, возвращающий количество товаров на текущей странице каталога, не считая раскупленные товары.
7. *public ArrayList<ProductPage> getProductPages()* - метод класса, возвращающий страницы *ProductPage* всех товаров на текущей странице каталога, не считая раскупленные товары.
8. *public ProductPage getFirstProductPage()* - метод, возвращающий страницу *ProductPage*, которая соответствует первому товару на текущей странице каталога, не считая раскупленные товары.
9. *public ArrayList<CataloguePage> getPagesWithNumberOfProductsLimits()* - метод, который возвращает страницы каталогов со всеми возможными ограничениями на количество отображаемых товаров.
10. *public Pair<Boolean, ArrayList<Double>> checkSortedPrices()* - метод, проверяющий упорядоченность цен товаров в порядке неубывания, не считая

раскупленные товары. Метод возвращает пару из *boolean* значения и массива цен.

11. *public Pair<Boolean, ArrayList<Double>> checkLimitedPrices(int limit)* - метод, проверяющий ограниченность цен товаров, не считая раскупленные товары. Метод возвращает пару из *boolean* значения и массива цен.

- *class ClothesCataloguePage* - класс, соответствующий странице каталога одежды на сайте. Наследуется от класса *CataloguePage*.

Методы класса:

1. *public ClothesCataloguePage() throws IOException* - метод-конструктор класса, использующий конструктор родительского класса, на вход которому подается *URL*-адрес данной страницы.

- *class BackpacksCataloguePage* - класс, соответствующий странице каталога рюкзаков на сайте. Наследуется от класса *CataloguePage*.

Методы класса:

1. *public BackpacksCataloguePage()* - метод-конструктор класса, использующий конструктор родительского класса, на вход которому подается *URL*-адрес данной страницы.

2.3. Тесты

- *class BaseTest* - класс - базовый тест, от которого наследуются остальные.

Поля класса:

1. protected static Logger logger - встроенный логгер тестов.

2. private static final String BROWSER - используемый браузер.

3. private static final int PAGE_LOAD_TIMEOUT - время ожидания загрузки страницы.

4. private static final int TIMEOUT - время ожидания элемента страницы.

5. private static final String PROP_PATH - путь до файла *config.properties*.

Методы класса:

1. *void setUpWebDriver()* - метод для настройки веб-драйверов

2. *void setUpLogger(String loggerName) throws IOException* - метод для настройки логгера. На вход принимает имя логгера.

3. *public void init(TestInfo testInfo) throws IOException* - инициализация теста. Выполняется перед запуском каждого теста.

4. *public void tearDown()* - завершение теста. Выполняется по завершении каждого теста.

- *class FavouritesTest* - класс для тестов, относящихся к функционалу избранного. Наследуется от *BaseTest*.

Методы класса:

1. *public void addFavouritesTest()* - метод-тест для функции добавления товара в избранное.
2. *public void removeFavouritesTest()* - метод-тест для функции удаления товара из избранного.

- *class ShoppingCartTest* - класс для тестов, относящихся к функционалу избранного. Наследуется от *BaseTest*.

Методы класса:

1. *public void addToShoppingCartTest()* - метод-тест для функции добавления товара в корзину.
2. *public void increaseAmountOfProduct()* - метод-тест для функции увеличения количества экземпляров товара в корзине.
3. *public void removeFromShoppingCart()* - метод-тест для функции удаления товара из корзины.

- *class AuthorizationTest* - класс, содержащий все тесты авторизации пользователя. Наследуется от *BaseTest*.

Методы класса:

1. private final static String EXPECTED_URL - URL ожидаемой страницы.

2. private static String USER_EMAIL - поле почты пользователя.

3. private static String USER_PASSWORD - поле пароля пользователя.

Методы класса:

1. public static void initializeUser() throws IOException - метод, считывающий данные пользователя из файла *config.properties* и заносящий их в поля класса. Выполняется один раз перед всеми тестами.

2. public void correctAuthorizationTest() - проверяет корректность авторизация на сайте при вводе правильных данных пользователя.

3. *public void incorrectLoginAuthorizationTest()* - проверяет, что авторизация на сайте провалится при вводе неправильной почты.

4. public void incorrectPasswordAuthorizationTest() - проверяет, что авторизация на сайте провалится при вводе правильной почты и неправильного пароля.

5. public void noLoginAuthorizationTest() - проверяет, что авторизация на сайте не произойдет, если не будет введена почта.

- *class FilterTest* - класс, содержащий все тесты фильтрации товаров в каталоге.

  Наследуется от *BaseTest*.

Поля класса:

1. *private final static String COLOR* - цвет товаров для фильтрации.
2. *private final static String TYPE* - тип товаров для фильтрации.
3. *private final static int PRICE* - цена товаров для фильтрации.

Методы класса:

1. *public void colorFilterTest()* - метод-тест для функции фильтрации товаров в каталоге по цвету.
2. *public void typeFilterTest()* - метод-тест для функции фильтрации товаров в каталоге по типу.
3. *public void priceRangeTest()* - метод-тест для функции фильтрации товаров в каталоге по цене.

- *class NumberOfProductsLimitTest* - класс, содержащий тесты меню выбора количества отображаемых на странице товаров. Наследуется от *BaseTest*.

  Методы класса:

  1. public void limitNumberOfProductsTest() - проверка корректности количества при всех возможных значениях меню выбора количества отображаемых на странице товаров.

- *class SortTest* - класс, содержащий тесты меню сортировки каталога. Наследуется от *BaseTest*.

  Методы класса:

  1. public void priceSortTest() - проверка корректности порядка товаров в случае сортировки по цене.

2.4. Утилиты

- *class DescriptionParser* - класс, для обработки описания элемента страницы.

Методы класса:

1. *public static HashMap<String, String> parse(String str)* - метод, разделяющий строку-описание элемента на части и формирующий из них хэш-таблицу, с помощью которой можно получить значения всех пунктов описания.

- *record Pair<T, S>(T first, S second)* - класс для хранения двух объектов произвольных классов.

- *class RandomGenerator* - класс для получения случайных объектов.

Методы класса:

1. *public static int randInt(int from, int to)* - метод для генерации случайного целого числа из выбранного диапазона. Принимает на вход границы генерации.
2. *public static String randomEmail()* - метод для генерации адреса случайной электронной почты.
3. *public static String randomPassword()* - метод для генерации случайного пароля.

- PropertiesReader - класс для считывания необходимых данных из файла конфигурации.

Поля класса:

1. *public final static String PROP_PATH* - путь до файла *config.properties*

Методы класса:

1. *public static String getPropertyByKey(String key) throws IOException* - метод считывания данных из файла конфигурации. Возвращает ключ, по которому производится считывание. Возвращается значение *value* по ключу *key*.

2.5. Конфигурационные файлы

- *config.properties* - файл типа *properties*, в котором хранятся конфигурационные константы проекта, доступ к которым осуществляется по ключам. Включает в себя следующие пары ключ-значение:

  1. email=Tsar26Max@yandex.ru

  2. password=SimplePassword123

  3. url_authorization=https://tramp-sport.ru/index.php?route=account/login

  4. url_cart=https://tramp-sport.ru/cart/

  5. url_wishlist=https://tramp-sport.ru/wishlist/

  6. url_backpacks_catalogue=https://tramp-sport.ru/catalog/ryukzaki/

  7. url_clothes_catalogue=https://tramp-sport.ru/catalog/odezhda/

- *logging.properties* - файл типа *properties*, который содержит в себе формат логов. Включает в себя пару ключ-значение:

  1. *java.util.logging.SimpleFormatter.format=[%1$tc] %4$s: %5$s%n*

2.6. Launcher

- *class Launcher* - класс, отвечающий за запуск тестов. Предоставляет возможность запускать тесты блоками или все тесты сразу.

  Сначала печатается строка-подсказка, которая показывает варианты входных данных, от них зависит, какие тесты будут запущены. Далее считывается строка символов. В *switch-case* проверяется, что введенные данные соответствует возможным опциям.

  Затем создается объект класса *Launcher* (встроенный в библиотеку *org.junit.platform.launcher.core.LauncherFactory*). Для этого объекта создается приемник (*listener*) - экземпляр класса *SummaryGeneratingListener,* который позволяет выводить информацию о запущенных тестах. После этого происходит создание запроса с использованием *LauncherDiscoveryRequestBuilder*. В соответствии с введенными данными запускаются тесты, отвечающие определенным тегам (блокам) или все тесты. По завершении тестов выводится информация о них.

3. БЛОКИ ТЕСТОВ

3.1. Тестирование страницы каталога товаров

- ColorFilterTest

  Данный тест проверяет корректность выводимых товаров в случае использования фильтра по цвету товара. Задается строка-константа *COLOR*, которая содержит цвет, по которому будет происходить фильтрация.

  Создается страница каталога рюкзаков *cataloguePage*. У страницы вызывается метод *chooseFilter* с параметром *COLOR*, который устанавливает фильтр. Затем вызывается метод *getProductPages*, который возвращает массив объектов *productPage* - страниц товаров из каталога.

  Каждая такая страница открывается, после чего вызывается метод *getColor*, который возвращает цвет товара. С помощью *assertEquals* происходит сравнение строки, возвращаемой методом *getColor*, и строки *COLOR*.

  Тест завершается успешно, если каждый товар имеет цвет *COLOR*.

- TypeFilterTest

  Данный тест проверяет корректность выводимых товаров в случае использования фильтра по типу товара. Задается строка-константа *TYPE*, которая содержит тип, по которому будет происходить фильтрация.

  Создается страница каталога рюкзаков *cataloguePage*. У страницы вызывается метод *chooseFilter* с параметром *TYPE*, который устанавливает фильтр. Затем вызывается метод *getProductPages*, который возвращает массив объектов *productPage* - страниц товаров из каталога.

  Каждая такая страница открывается, после чего вызывается метод *isSpecificType* с параметром *TYPE*, который возвращает *true*, если товар имеет тип *TYPE*,

иначе *false*. С помощью *assertTrue* происходит проверка значения, возвращаемого методом *isSpecificType*.

Тест завершается успешно, если каждый товар имеет тип *TYPE*.

- PriceRangeTest

  Данный тест проверяет корректность выводимых товаров в случае ограничения ценового диапазона сверху. Задается число-константа *PRICE*, которое будет являться верхней границей ценового диапазона.

  Создается страница каталога рюкзаков *cataloguePage*. У страницы вызывается метод *setPriceLimit* с параметром *PRICE*, который устанавливает фильтр. Затем вызывается метод *checkLimitedPrices* с параметром *PRICE*, который возвращает *true*, если все товары каталога имеют цену не больше *PRICE*, иначе *false*. Также метод возвращает массив цен проверенных товаров. С помощью *assertTrue* происходит проверка значения, возвращаемого методом *checkLimitedPrices*.

  Тест завершается успешно, если цена каждого товара в каталоге не превышает *PRICE*.

- PriceSortTest

  Данный тест проверяет корректность порядка выводимых товаров в каталоге в случае применения сортировки по возрастанию цены.

  Создается страница каталога рюкзаков *cataloguePage*. У страницы вызывается метод *sortProductsByPrice*. Затем вызывается метод *checkSortedPrices*, который возвращает *true*, если цены товаров каталога упорядочены в порядке неубывания, иначе *false*. Также метод возвращает массив цен проверенных товаров. С помощью *assertTrue* происходит проверка значения, возвращаемого методом *checkSortedPrices*.

Тест завершается успешно, если цены товаров каталога отсортированы в порядке неубывания.

- LimitNumberOfProductsTest

  Данный тест проверяет корректность количества выводимых товаров на одной странице каталога в зависимости от выбранного значения в поле “Показывать:”.

  Создается страница каталога рюкзаков *cataloguePage*. У страницы вызывается метод *getPagesWithNumberOfProductsLimits*, который возвращает страницы каталогов со всеми возможными ограничениями на количество отображаемых товаров. Каждая такая страница открывается, после чего вызываются методы *getNumberOfProductsLimit*, который возвращает текущее ограничение на количество товаров *limit*, и *getNumberOfProducts*, который возвращает количество товаров на странице *numberOfProducts*. С помощью *assertTrue* происходит проверка того, что значение *numberOfProducts* не превышает *limit*.

  Тест завершается успешно, если во всех случаях количество товаров на странице не превышает соответствующее ограничение.

3.2. Тестирование функций работы с корзиной и избранным

- AddFavoritesTest

  Данный тест проверяет корректность функции добавления товара в избранное.

  Открывается страница каталога одежды *cataloguePage*. После создается страница первого товара *productPage*, она открывается. На этой странице происходит добавление товара в избранное методом *clickFavouritesButton()*. Осуществляется получение артикула товара методом *getArticle()*. Это значение записывается в переменную *expectedArticle*. Далее создается *wishlistPage* и открывается соответствующая страница. Записывается *productPage* объект

первого элемента страницы. Страница первого элемента открывается, и с помощью метода *getArticle()* значение артикула записывается в переменную *actualArticle*. *assertEquals* сравнивает переменные *expectedArticle* и *actualArticle*. В случае возникновения исключения (некоторые методы выбрасывают их) тест считается проваленным.

Тест завершается успешно, если значения *expectedArticle* и *actualArticle* совпадают.

- AddToShoppingCartTest

  Данный тест проверяет корректность функции добавления товара в корзину пользователя.

  Открывается страница каталога одежды *cataloguePage*. После создается страница первого товара *productPage*, она открывается. Товар добавляется в корзину методом *addToShoppingCart()*. Затем значение артикула товара записывается в переменную *expectedArticle* с помощью метода *getArticle()*. Создается объект *cartPage*, и открывается соответствующая страница. На этой странице в переменную *productPage* записывается значение страницы первого товара в корзине. Вызывается метод openPage(), на новой странице методом *getArticle()* в переменную *actualArticle* записывается значение. Происходит сравнение *expectedArticle* и *actualArticle* в *assertEquals*. В случае возникновения исключения (некоторые методы выбрасывают их) тест считается проваленным.

  Тест завершается успешно, если значения *expectedArticle* и *actualArticle* совпадают.

- IncreaseAmountOfProduct

Данный тест проверяет корректность функции увеличения экземпляров товара в корзине.

Открывается страница каталога одежды *cataloguePage*. После создается страница первого товара *productPage*, она открывается. Товар добавляется в корзину методом *addToShoppingCart()*. Создается объект *cartPage*, и открывается соответствующая страница. На этой странице считывается начальное количество экземпляров товара методом *getAmountOfFirstProduct(),* оно записывается в переменную amount. После с помощью метода *randInt()* генерируется целое число от 1 до 4. Это число записывается в переменную *increaseValue*. Метод *increaseAmountOfFirstProduct()* нажимает на кнопку увеличения *increaseValue* раз. В переменную *newAmount* записывается новое количество экземпляров. *assertEquals()* сравнивает значения *amount+increaseValue* и *newAmount*. В случае возникновения исключения (некоторые методы выбрасывают их) тест считается проваленным.

Тест завершается успешно, если значение *amount+increaseValue* оказывается равным *newAmount*.

- RemoveFavouritesTest

Данный тест проверяет корректность функции удаления товара из избранного.

Открывается страница каталога одежды *cataloguePage*. После создается страница первого товара *productPage*, она открывается. На этой странице происходит добавление товара в избранное методом *clickFavouritesButton()*. Далее создается *wishlistPage* и открывается соответствующая страница. Записывается *productPage* объект первого элемента страницы. Страница первого элемента открывается, повторное нажатие на кнопку добавления в избранное методом *clickFavouritesButton()* удаляет товар из избранного. Открывается страница избранного. Метод *isEmpty()* проверяет пуста ли

страница, результат подается в *assertTrue*. Тест провалится, если страница пуста. В случае возникновения исключения (некоторые методы выбрасывают их) тест считается проваленным.

Тест завершится успешно, если страница избранного будет пуста.

- RemoveFromShoppingCartTest

  Данный тест проверяет корректность работы функции удаления товара из корзины пользователя.

  Открывается страница каталога одежды *cataloguePage*. После создается страница первого товара *productPage*, она открывается. Товар добавляется в корзину методом *addToShoppingCart()*. Создается объект *cartPage*, и открывается соответствующая страница. Метод *removeFirstProduct()* осуществляет удаление первого товара из корзины. Метод *isEmpty()* проверяет пуста ли страница, результат подается в *assertTrue*. Тест провалится, если страница пуста. В случае возникновения исключения (некоторые методы выбрасывают их) тест считается проваленным.

  Тест завершится успешно, если страница корзины будет пуста.

3.3. Тестирование авторизации на сайте

- CorrectAuthorizationTest

  Данный тест проверяет корректность аутентификации на сайте при вводе правильных данных пользователя. Данные пользователя считаются правильными, если он был зарегистрирован на сайте с данной почтой и паролем.

Создается и открывается страница авторизации на сайте *LoginPage*. Метод *authorize()* производит ввод правильной почты и пароля в соответствующие поля на странице, после чего нажимается кнопка “Войти”. Затем считывается *URL*-адрес страницы, на которую произведен переход. Если *URL*-адрес страницы совпадает с адресом страницы личного кабинета *EXPECTED_URL*, то тест считается успешно пройденным. Тест провалится, если адрес страницы окажется отличным от адреса личного кабинета.

- IncorrectLoginAuthorizationTest

  Данный тест проверяет, что аутентификации на сайте провалится при вводе неправильных данных пользователя, а именно почты. Данные пользователя считаются неправильными, если они не были до этого зарегистрированы на сайте.

  Создается и открывается страница авторизации на сайте *LoginPage*. Генерируется случайная электронная почта с помощью библиотеки *Faker*, это гарантирует то, что почта не будет существовать в реальности и принадлежать какому-либо пользователю. Метод *authorize()* производит ввод неправильной почты и правильного пароля в соответствующие поля на странице, после чего нажимается кнопка “Войти”. Затем производится проверка существования сообщения об ошибке на странице. Если сообщение об ошибке существует на странице, то тест считается успешно пройденным. Тест провалится, если на странице не окажется сообщения об ошибке.

- IncorrectPasswordAuthorizationTest

  Данный тест проверяет, что аутентификации на сайте провалится при вводе правильной почты и неправильного пароля.

Создается и открывается страница авторизации на сайте *LoginPage*. Генерируется случайный пароль с помощью библиотеки *Faker*. Метод *authorize()* производит ввод правильной почты и неправильного пароля в соответствующие поля на странице, после чего нажимается кнопка “Войти”. Затем производится проверка существования сообщения об ошибке на странице. Если сообщение об ошибке существует на странице, то тест считается успешно пройденным. Тест провалится, если на странице не окажется сообщения об ошибке.

- NoLoginAuthorizationTest

Данный тест проверяет, что аутентификации на сайте не произойдет, если не будет введена почта в соответствующее текстовое поле на странице.

Создается и открывается страница авторизации на сайте *LoginPage*. Метод *authorize()* производит ввод правильного пароля в соответствующее поле на странице, при этом оставляя поле *E-mail* пустым, после чего нажимается кнопка “Войти”. Затем считывается *URL*-адрес страницы, на которую произведен переход (при корректном сценарии на сайте возникает окно, уведомляющее о том, что следует заполнить поле, при этом перехода на другие страницы не возникает). Если *URL*-адрес страницы не совпадает с адресом страницы личного кабинета *EXPECTED_URL*, то тест считается успешно пройденным. Тест провалится, если адрес страницы окажется равен адресу личного кабинета.

4. ИНСТРУКЦИЯ ПО ЗАПУСКУ ПРОГРАММЫ

Предусмотрено несколько вариантов запуска проекта.

Первый вариант: запуск метода *main* класса *Launcher*. Далее потребуется выбрать один из четырех режимов:

search - для запуска тестов страницы каталога товаров.

profile - для запуска тестов функций работы с корзиной и избранным.

authorization - для запуска тестов авторизации на сайте.

all - для запуска всех тестов.

Второй вариант: избирательный запуск отдельных тестов, для этого в классе теста нужно выбрать метод, помеченный аннотацией *@Test*.

5. ПРИМЕР ТЕСТИРОВАНИЯ

Продемонстрируем тестирование на примере теста *CorrectAuthorizationTest*.

1. При запуске теста создается и открывается страница авторизации *LoginPage*.

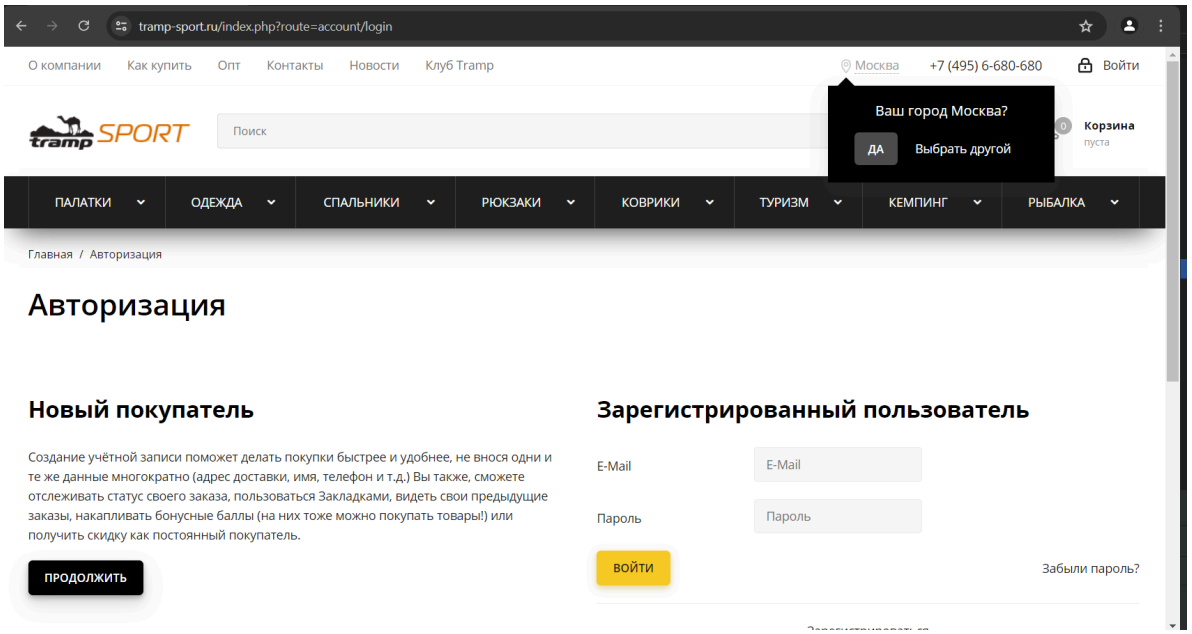

Рисунок 1 - Открытие страницы авторизации на странице *LoginPage*

2. Метод *authorize()* производит ввод правильной почты и пароля в соответствующие поля на странице, после чего нажимается кнопка “Войти”.

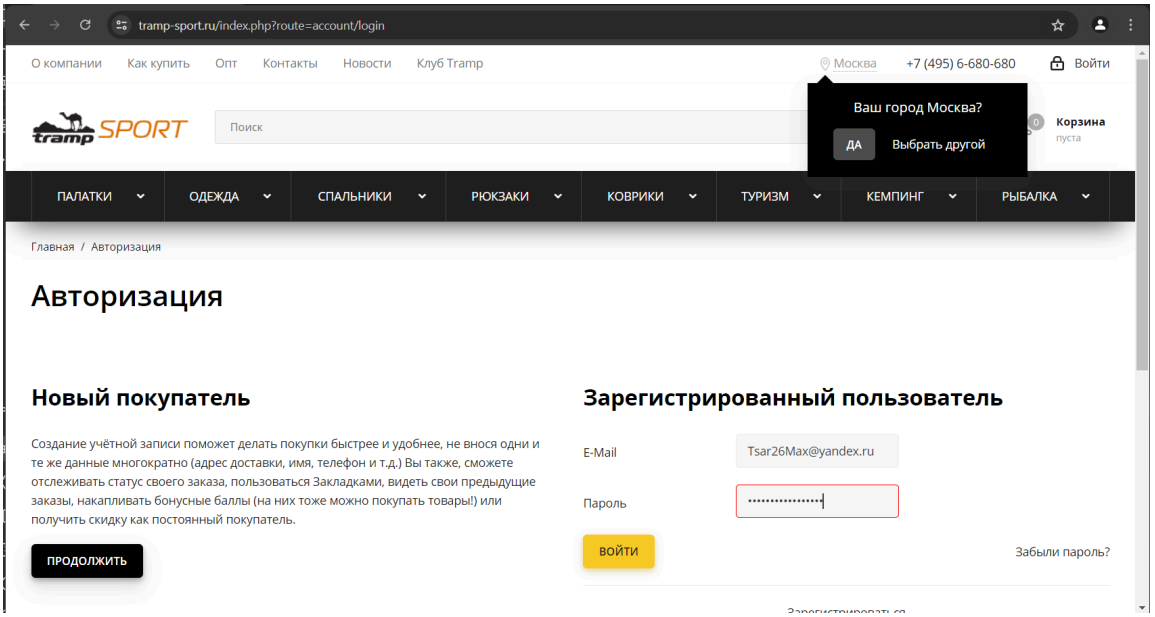

Рисунок 2 - Ввод корректных данных на странице *LoginPage*

3. Затем считывается *URL*-адрес страницы, на которую произведен переход. Если *URL*-адрес страницы совпадает с адресом страницы личного кабинета *expectedUrl*, то тест считается успешно пройденным. Тест провалится, если адрес страницы окажется отличным от адреса личного кабинета.

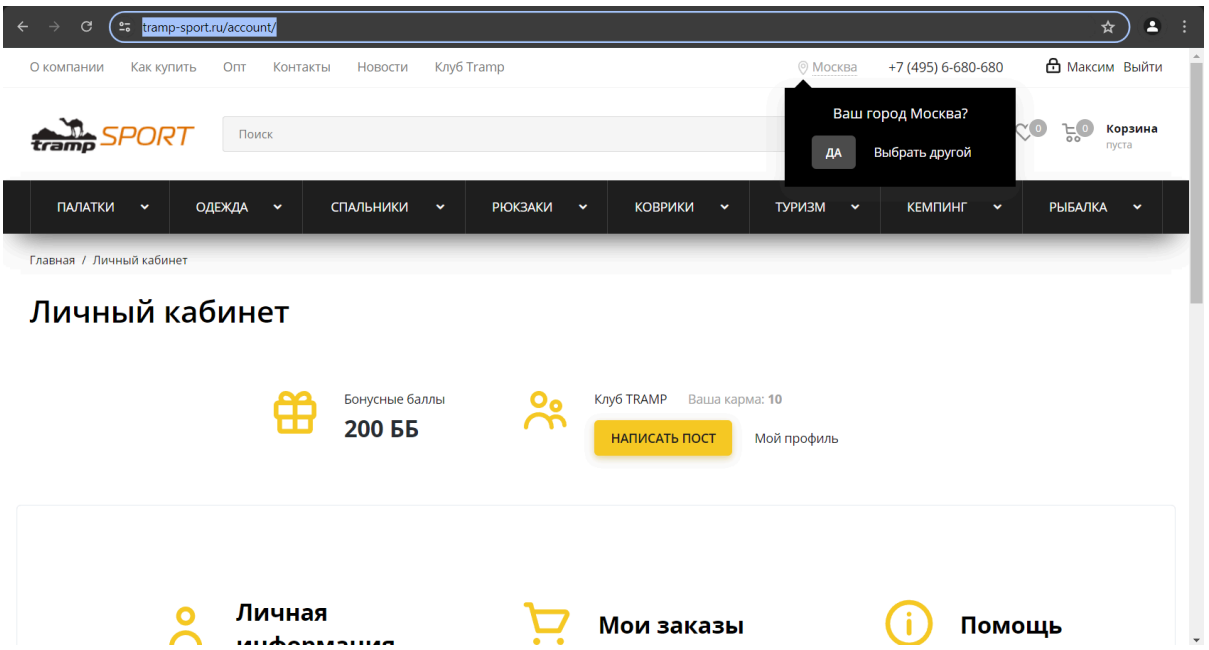

Рисунок 3 - Переход на страницу личного кабинета, выделенный *URL*-адрес страницы является правильным адресом личного кабинета

Лог теста:

```
[сб июл. 06 02:13:15 MSK 2024] INFO: Open login page https://tramp-sport.ru/index.php?route=account/login

[сб июл. 06 02:13:17 MSK 2024] INFO: Start authorization

[сб июл. 06 02:13:17 MSK 2024] CONFIG: Email: Tsar26Max@yandex.ru

[сб июл. 06 02:13:17 MSK 2024] CONFIG: Password: SimplePassword123

[сб июл. 06 02:13:17 MSK 2024] INFO: Check for results

[сб июл. 06 02:13:17 MSK 2024] CONFIG: Current url: https://tramp-sport.ru/account/
```

[сб июл. 06 02:13:17 MSK 2024] INFO: Current url should be https://tramp-sport.ru/account/

[сб июл. 06 02:13:17 MSK 2024] FINE: Test passed

ЗАКЛЮЧЕНИЕ

В рамках проекта было проведено *UI*-тестирование сайта *Tramp Sport*. Тестирование включало проверку корректности работы различных функций сайта, таких как взаимодействие с корзиной и избранным, фильтрация и сортировка товаров, а также авторизация пользователя на сайте.

С использованием фреймворка *JUnit* были проведено модульное тестирование разработанного проекта. Были реализованы блоки тестов, проверяющие корректную работу систем исследуемого сайта. Тесты показали, что выбранные функции сайта работают корректно.

Фреймворк *Selenide* использовался для настройки веб-браузера, а также для взаимодействия со страницами и их элементами.

С помощью системы сборки *Maven* к проекту были подключены сторонние библиотеки и зависимости.

Работа велась с использованием системы контроля версий *git* на платформе *GitHub*. К реализованному проекту была написана документация с помощью генератора документации *JavaDoc*.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *https://github.com/ShevelevaAnna/SP_24_3* - репозиторий на *GitHub*, содержащий программный код данного проекта.
2. *https://selenide.org/documentation.html* - документация *Selenide*.
3. *https://junit.org/junit5/docs/current/user-guide/* - документация *JUnit* 5.
4. *https://maven.apache.org/guides/index.html* - документация *Maven*.

ПРИЛОЖЕНИЕ А

ЧЕК-ЛИСТ ТЕСТОВ

Ниже представлено описание тестов, реализованных в рамках данной работы. Описание теста включает в себя название теста, входные данные, последовательность действий, выполняемых в тесте, и ожидаемый результат.

Объектом тестирования является интернет-магазин Tramp Sport (https://tramp-sport.ru/).

Тесты разделены по блокам.

I. Тестирование страницы каталога товаров

- ColorFilterTest

  Данный тест проверяет корректность выводимых товаров в случае использования фильтра по цвету товара.

  В качестве входных данных принимается строка, содержащая название цвета товаров, имеющегося в фильтрах. В данном случае используется строка “Синий”.

  Тест совершается под неавторизованным пользователем.

  Последовательность действий в тесте:

  1. Открытие страницы каталога “Рюкзаки” (https://tramp-sport.ru/catalog/ryukzaki/)
  2. Выбор цвета “Синий” в меню фильтров
  3. Для каждого товара, оставшегося в каталоге:
     - Открытие страницы товара

- Проверка цвета товара

Ожидается, что все товары, оставшиеся в каталоге, будут иметь цвет “Синий”.

- TypeFilterTest

Данный тест проверяет корректность выводимых товаров в случае использования фильтра по типу товара.

В качестве входных данных принимается строка, содержащая название типа товаров, имеющегося в фильтрах. В данном случае используется строка “аптечка”.

Тест совершается под неавторизованным пользователем.

Последовательность действий в тесте:

1. Открытие страницы каталога “Рюкзаки” (https://tramp-sport.ru/catalog/ryukzaki/)
2. Выбор типа “аптечка” в меню фильтров
3. Для каждого товара, оставшегося в каталоге:
    - Открытие страницы товара
    - Проверка типа товара

Ожидается, что все товары, оставшиеся в каталоге, будут иметь тип “аптечка”.

- PriceRangeTest

Данный тест проверяет корректность выводимых товаров в случае ограничения ценового диапазона сверху.

В качестве входных данных принимается число, которое будет являться верхней границей ценового диапазона. В данном случае используется число 5000.

Тест совершается под неавторизованным пользователем.

Последовательность действий в тесте:

1. Открытие страницы каталога “Рюкзаки” (https://tramp-sport.ru/catalog/ryukzaki/)
2. Ввод числа 5000 в соответствующее поле в меню фильтров
3. Для каждого товара, оставшегося в каталоге:
    - Проверка цены товара

Ожидается, что все товары, оставшиеся в каталоге, будут иметь цену, не превышающую 5000.

- PriceSortTest

Данный тест проверяет корректность порядка выводимых товаров в каталоге в случае применения сортировки по возрастанию цены.

Тест совершается под неавторизованным пользователем.

Последовательность действий в тесте:

1. Открытие страницы каталога “Рюкзаки” (https://tramp-sport.ru/catalog/ryukzaki/)
2. Выбор способа сортировки “цене” с помощью поля “Сортировать по:”
3. Для каждого товара, оставшегося в каталоге:
    - Проверка цены товара

Ожидается, что цена каждого товара в каталоге будет не меньше, чем цена предыдущего товара, т. е. товары отсортированы по возрастанию.

- LimitNumberOfProductsTest

Данный тест проверяет корректность количества выводимых товаров на одной странице каталога в зависимости от выбранного значения в поле “Показывать:”.

Тест совершается под неавторизованным пользователем.

Последовательность действий в тесте:

1. Открытие страницы каталога “Рюкзаки” (https://tramp-sport.ru/catalog/ryukzaki/)
2. Для каждого возможного значения в поле “Показывать:”
    - Установка этого значения
    - Проверка количества товаров на странице каталога

Ожидается, что количество товаров на одной странице каталога не будет превышать выбранное значение в поле “Показывать:”

II. Тестирование функций корзины и страницы избранных

- AddFavoritesTest

В данном тесте рассматривается функция добавления товара в избранное. В качестве такого товара рассматривается товар из каталога “Одежда”.

Тест совершается под неавторизованным пользователем.

Последовательность действий пользователя:

1. Открытие страницы каталога “Одежда” (https://tramp-sport.ru/catalog/odezhda/)
2. Открытие страницы первого товара
3. Нажатие кнопки добавления в избранное
4. Переход на страницу избранных товаров (https://tramp-sport.ru/wishlist/)
5. Открытие страницы первого товара в избранном

Ожидается, что артикулы выбранного в каталоге товара и товара в избранном совпадут, т. е. товар добавлен в избранное.

- AddToShoppingCartTest

В данном тесте рассматривается функция добавления элемента в корзину. В качестве такого товара рассматривается товар из каталога “Одежда”.

Тест совершается под неавторизованным пользователем.

Последовательность действий пользователя:

1. Открытие страницы каталога “Одежда” (https://tramp-sport.ru/catalog/odezhda/)
2. Открытие страницы первого товара
3. Добавление товара
    - Нажатие кнопки “Добавить в корзину”
    - Выбор первого размера в списке размеров
    - Нажатие кнопки “Добавить”
4. Переход на страницу корзины (https://tramp-sport.ru/cart/)

5. Переход на страницу первого товара

Ожидается, что артикулы выбранного в каталоге товара и товара в корзине совпадут, т. е. товар добавлен в корзину.

- IncreaseAmountOfProductTest

В данном тесте рассматривается функция увеличения количества экземпляров товара в корзине.

Тест совершается под неавторизованным пользователем.

Последовательность действий пользователя:

1. Открытие страницы каталога “Одежда” (https://tramp-sport.ru/catalog/odezhda/)
2. Открытие страницы первого товара
3. Добавление товара
    - Нажатие кнопки “Добавить в корзину”
    - Выбор первого размера в списке размеров
    - Нажатие кнопки “Добавить”
4. Переход на страницу корзины (https://tramp-sport.ru/cart/)
5. Нажатие на кнопку увеличения количества товаров (нажатие происходит случайное число раз - от 1 до 4)

Ожидается, что количество экземпляров станет равным числу нажатий на кнопку плюс 1.

- RemoveFavouritesTest

  Тестирование функции удаления товара из избранного.

  Тест совершается под неавторизованным пользователем.

Последовательность действий пользователя:

1. Открытие страницы каталога “Одежда” (https://tramp-sport.ru/catalog/odezhda/)
2. Открытие страницы первого товара
3. Нажатие кнопки добавления в избранное
4. Переход на страницу избранных товаров (https://tramp-sport.ru/wishlist/)
5. Нажатие на кнопку удаления товара из избранного

Ожидается, что избранное будет пусто, т.е. будет присутствовать соответствующая надпись.

- RemoveFromShoppingCartTest

  В данном тесте рассматривается функция удаления элемента из корзины.

  Тест совершается под неавторизованным пользователем.

Последовательность действий пользователя:

1. Открытие страницы каталога “Одежда” (https://tramp-sport.ru/catalog/odezhda/)
2. Открытие страницы первого товара
3. Добавление товара
   - Нажатие кнопки “Добавить в корзину”

- Выбор первого размера в списке размеров
- Нажатие кнопки “Добавить”

4. Переход на страницу корзины (https://tramp-sport.ru/cart/)
5. Нажатие кнопки удаления из корзины

Ожидается, корзина будет пуста, т.е. будет присутствовать соответствующая надпись.

III. Тестирование страницы авторизации

- CorrectAuthorizationTest

В данном тесте рассматривается корректная авторизация на сайте с правильно введенными логином и паролем.

Для этого изначально произведена регистрация на сайте с использованием почты “Tsar26Max@yandex.ru” и пароля “SimplePassword123”.

Последовательность действий теста:

1. Открытие страницы авторизации на сайте (https://tramp-sport.ru/index.php?route=account/login)
2. Введение в поле “E-Mail” почты “Tsar26Max@yandex.ru”
3. Введение в поле “Пароль” пароля “SimplePassword123”
4. Нажатие кнопки “Войти”

Ожидается, что пользователь перейдет на страницу личного кабинета.

- IncorrectLoginAuthorizationTest

В данном тесте рассматривается некорректная авторизация на сайте - попытка авторизации через почту и пароль, которые не были зарегистрированы на сайте до этого.

Почта сгенерирована случайным образом, пароль - “SimplePassword123”.

Последовательность действий теста:

1. Открытие страницы авторизации на сайте (https://tramp-sport.ru/index.php?route=account/login)
2. Введение в поле “E-Mail” случайно сгенерированной почты
3. Введение в поле “Пароль” пароля “SimplePassword123”
4. Нажатие кнопки “Войти”

Ожидается, что на текущей странице появится сообщение об ошибке и пользователь не перейдет в личный кабинет.

- IncorrectPasswordAuthorizationTest

В данном тесте рассматривается некорректная авторизация на сайте - попытка авторизации через зарегистрированную на сайте почту и неправильный пароль.

Пароль сгенерирован случайным образом, почта - “Tsar26Max@yandex.ru”.

Последовательность действий теста:

1. Открытие страницы авторизации на сайте (https://tramp-sport.ru/index.php?route=account/login)
2. Введение в поле “E-Mail” почты “Tsar26Max@yandex.ru”
3. Введение в поле “Пароль” случайно сгенерированного пароля

4. Нажатие кнопки “Войти”

Ожидается, что на текущей странице появится сообщение об ошибке и пользователь не перейдет в личный кабинет.

- NoLoginAuthorizationTest

В данном тесте рассматривается некорректная авторизация на сайте - попытка авторизации без ввода почты.

Пароль - “SimplePassword123”.

Последовательность действий теста:

1. Открытие страницы авторизации на сайте (https://tramp-sport.ru/index.php?route=account/login)
2. Введение в поле “Пароль” пароля “SimplePassword123”
3. Нажатие кнопки “Войти”

Ожидается, что пользователь не перейдет на страницу личного кабинета, а останется на текущей.

ПРИЛОЖЕНИЕ Б

UML-ДИАГРАММЫ

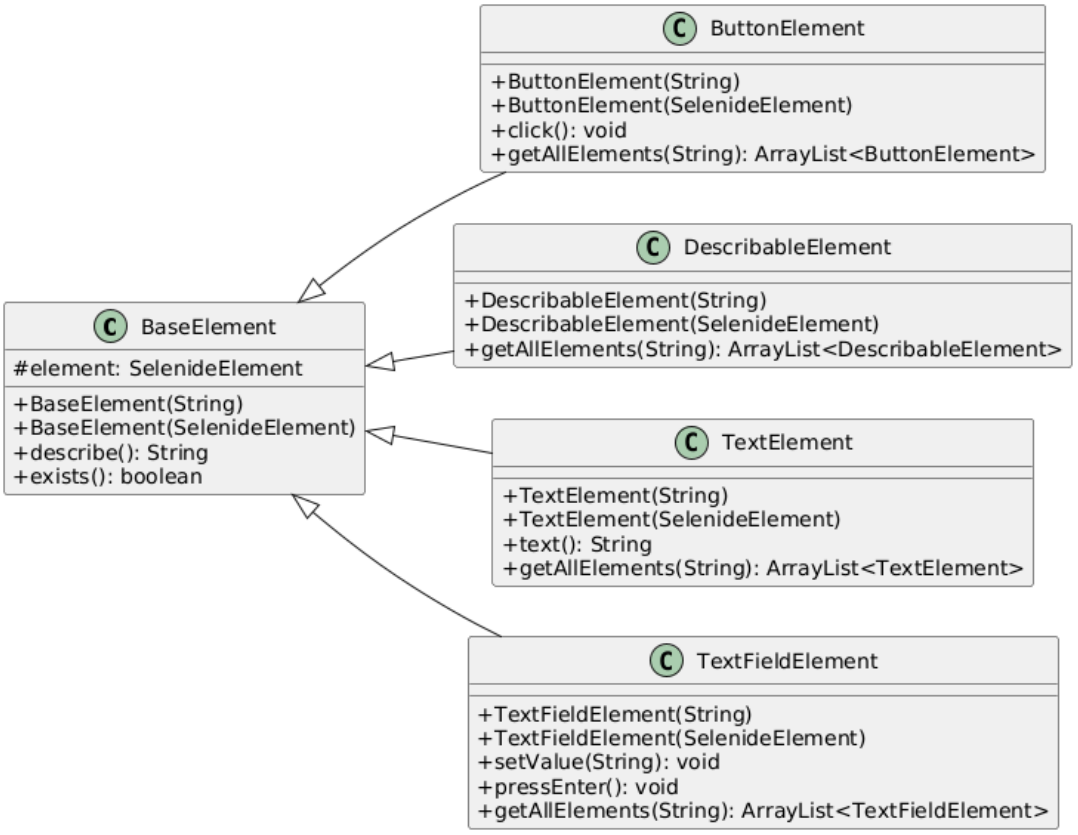

Рисунок 1 - UML-диаграмма пакета elements

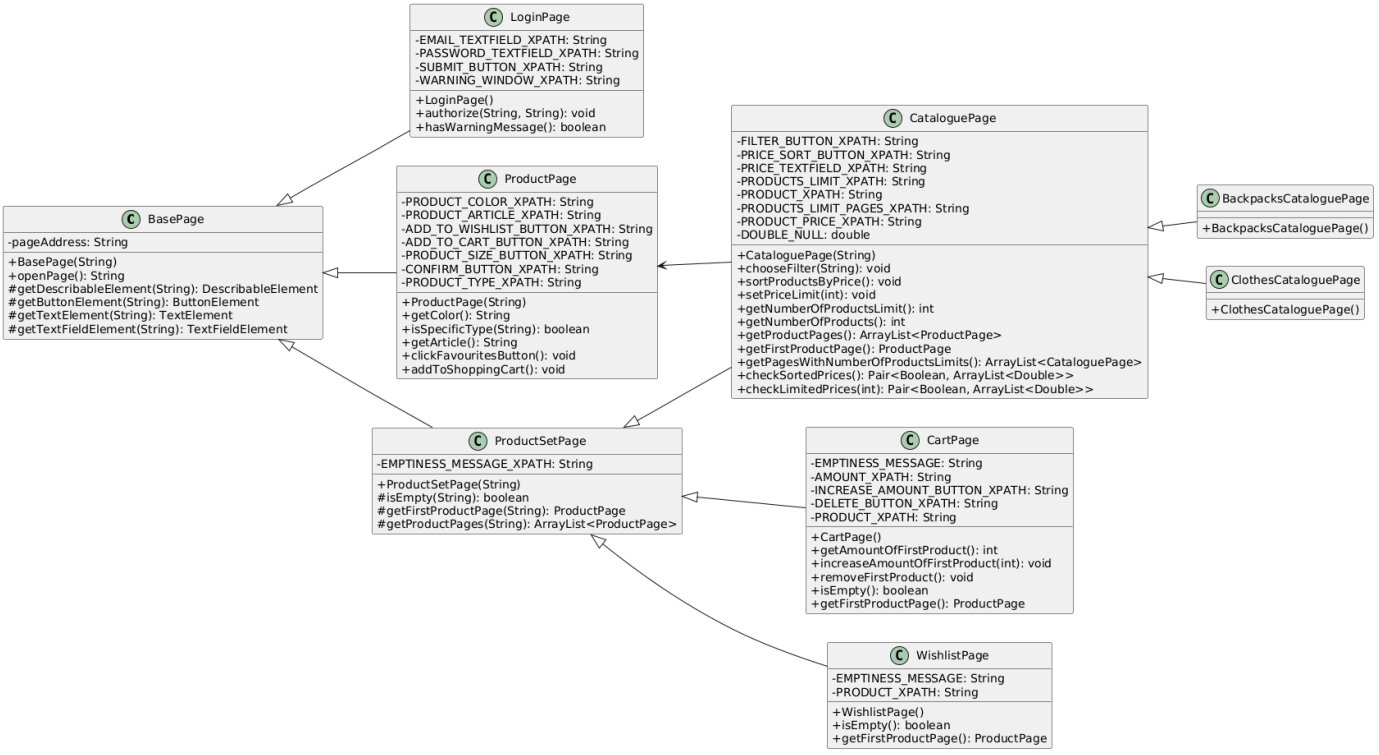

Рисунок 2 - UML-диаграмма пакета pages

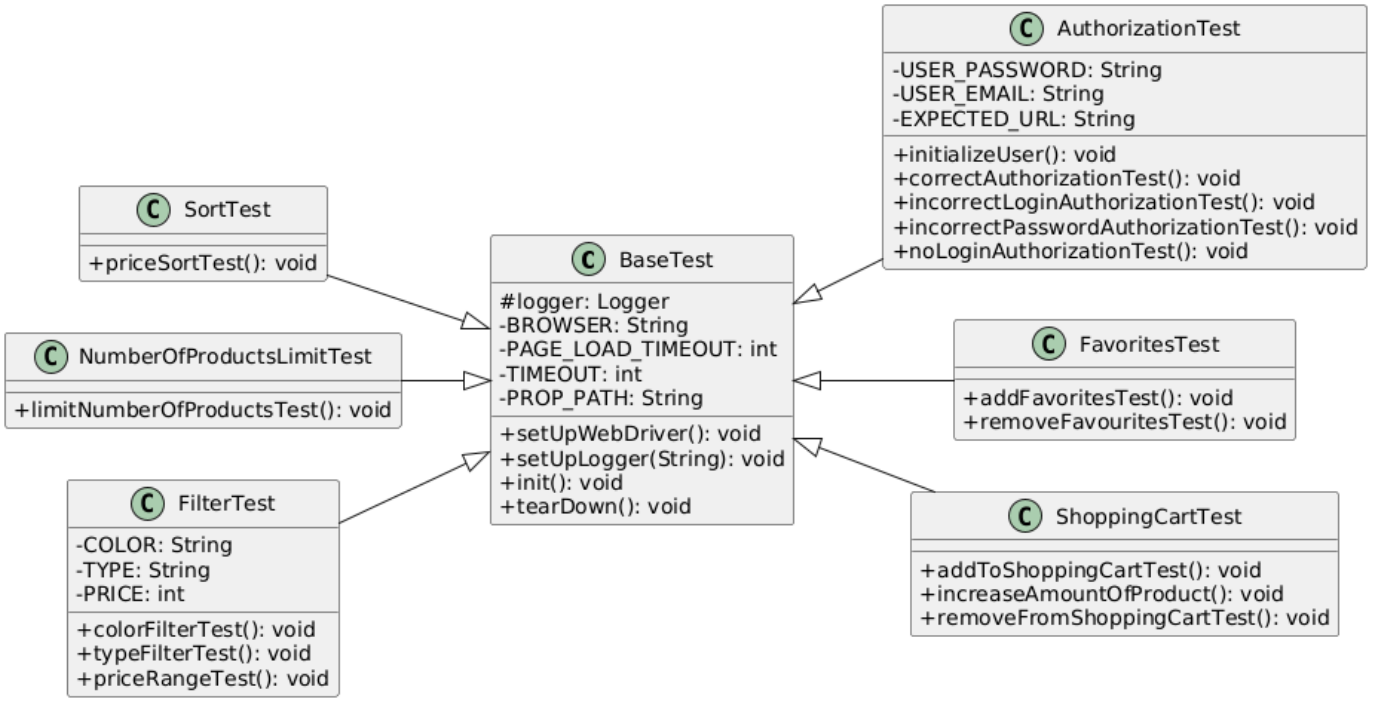

Рисунок 3 - UML-диаграмма пакета tests

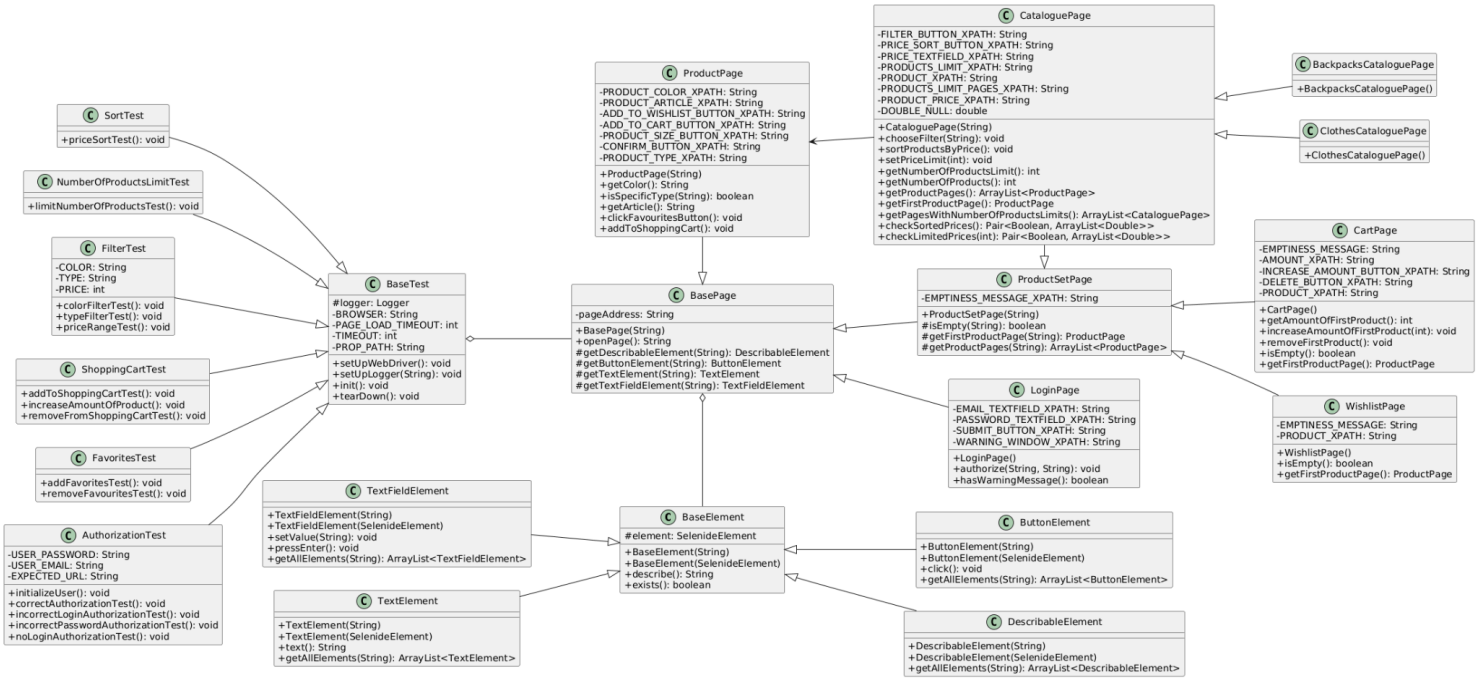

Рисунок 4 - общая UML-диаграмма