SOFTWARE VERIFICATION, VALIDATION AND TESTING

TESTING DOCUMENTATION

AuctionApp

Prepared by:

Admir Krilašević

Matej Mujezinović

Proposed to:

Samed Jukić, Assist. Prof. Dr.

Aldin Kovačević, Teaching Assistant

Date of submission

16/01/2022

Table of Contents

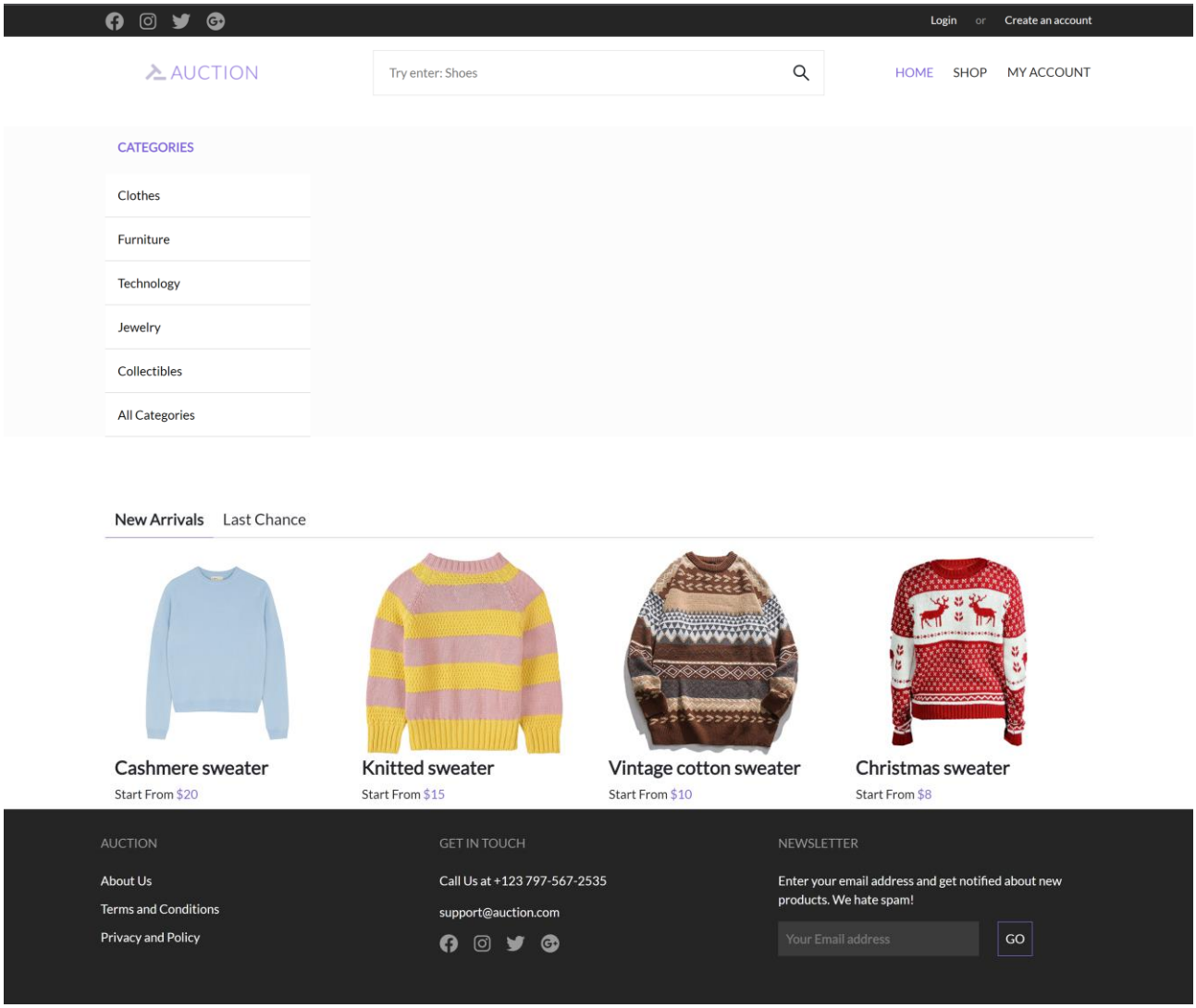
1. Introduction

1.1. About the Project

AuctionApp is an **e-commerce** web application that provides consumer-to-consumer sales based entirely on **bidding**. There are 2 types of users in the app: **unregistered** users, who can navigate through the app and view all products, and **registered** users, who can act as sellers and/or bidders. This application is an internship project, which is still being developed, so not all features are implemented implying that not all features will be tested. The application is available at the following link: https://auctionapp.krilasevic.me (since the application has separate client and server sides, in case the items do not load it should be checked that the server is running https://auction-app-abh-server.herokuapp.com).

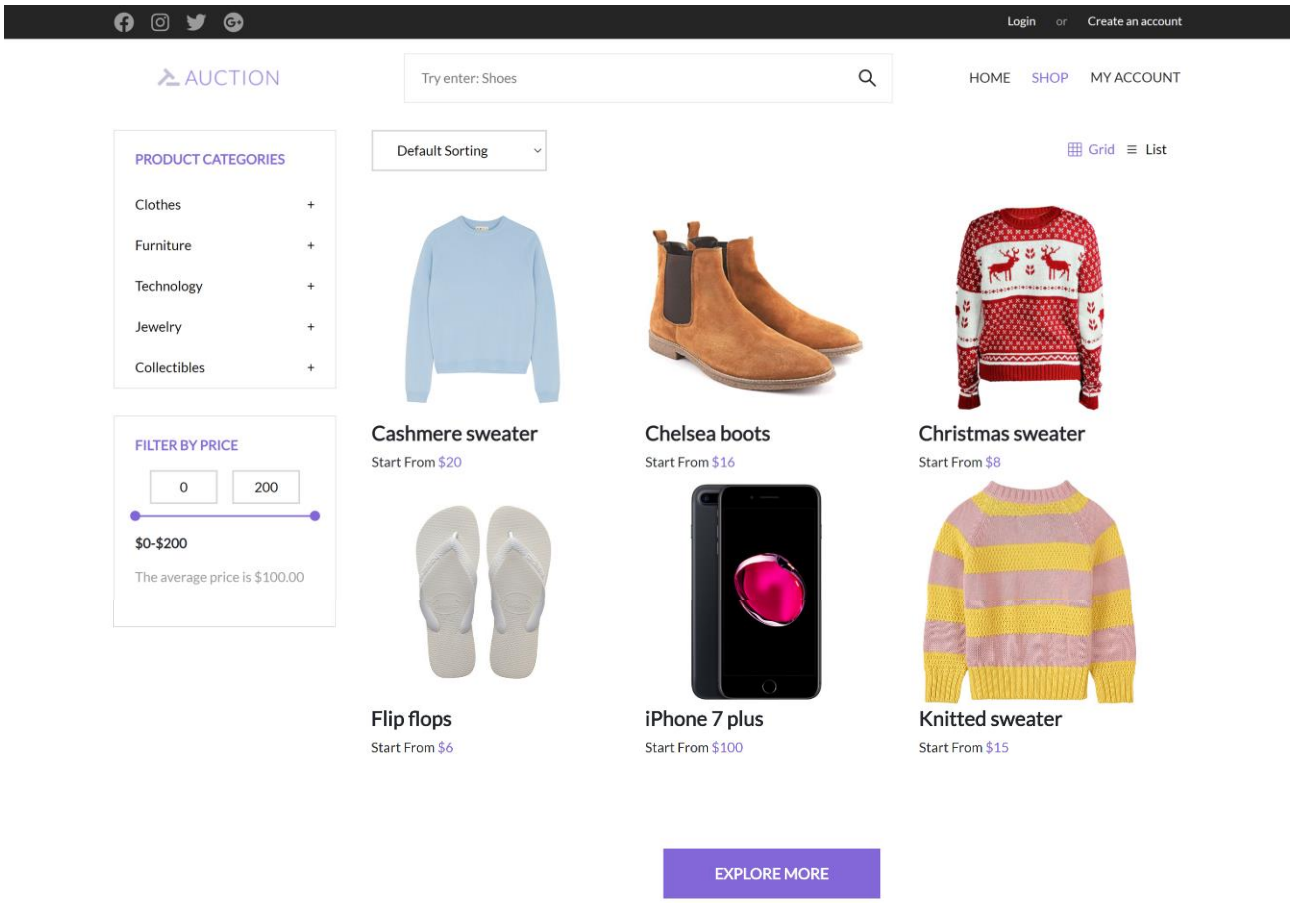
1.2. Project Functionalities and Screenshots

Home Page:

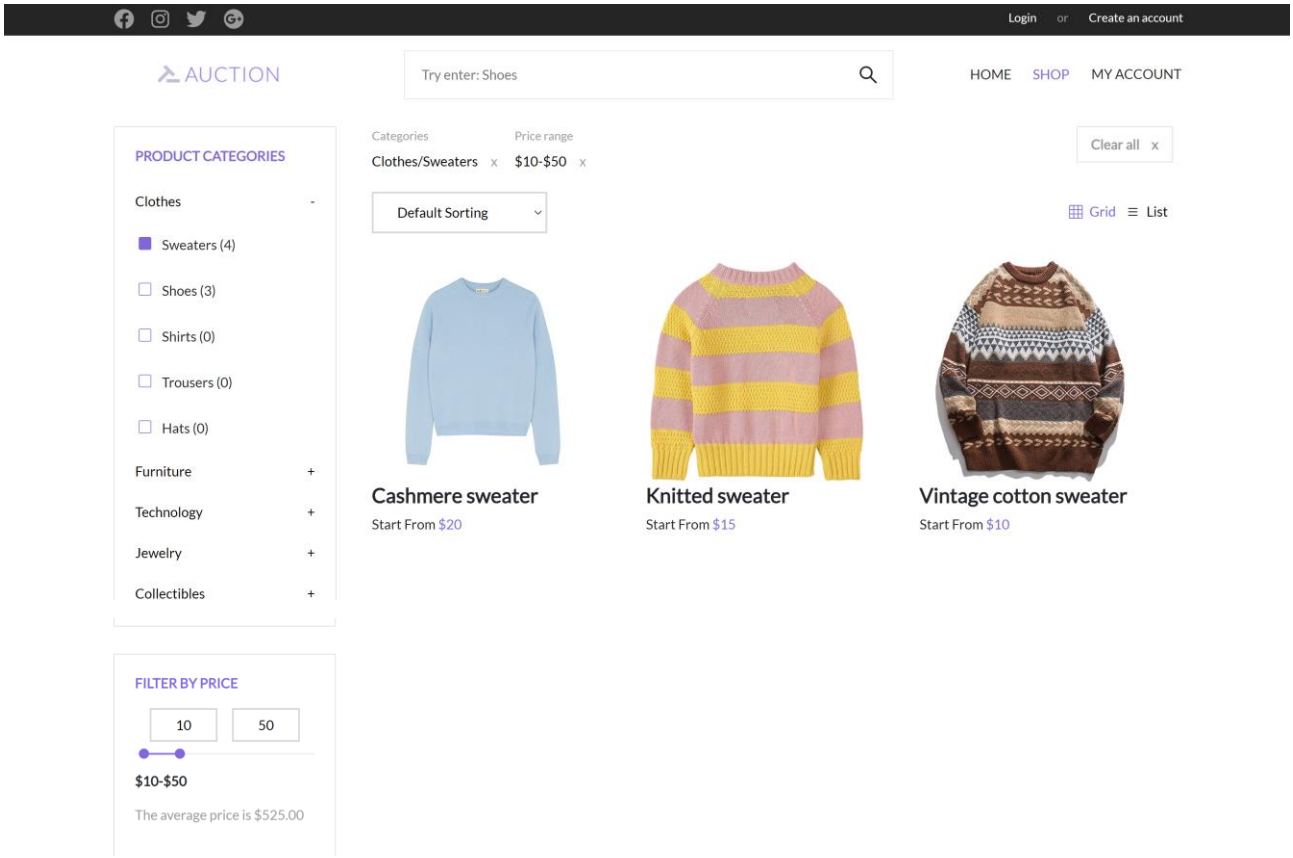


The page layout consists of the header, with social media links, as well as links to login and registration pages, below which is the navigation bar, with a logo that also serves as the home page link, a search bar, and three main pages withing the application, page content, and footer with informational links, contact information and newsletter subscription. The page seen above is the landing page (home page), and its content consists of two parts – categories section and tab section. The categories section lists all the existing categories of items, and upon clicking on any of them the user would be redirected to the shop page, with an applied filter for the clicked category. There is also an "All Categories" button, which performs the same action as the "Shop" button in the navigation bar. Below, the tab sections consists of "New Arrivals" and "Last Chance" items, both of which are implemented with infinite scroll functionality. New Arrivals displays items sorted by the starting date of their auction, while Last Chance displays items sorted by their auction end date.

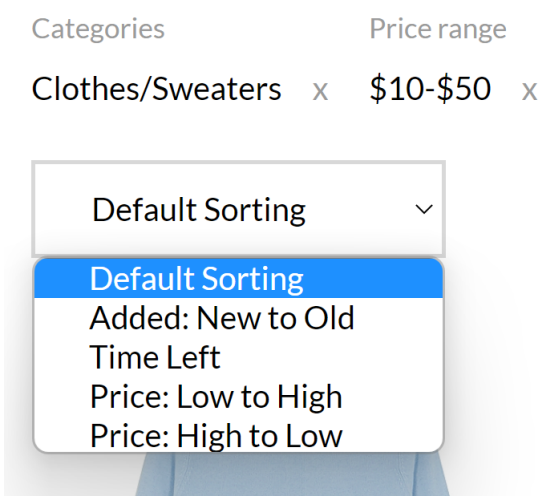
Shop Page:



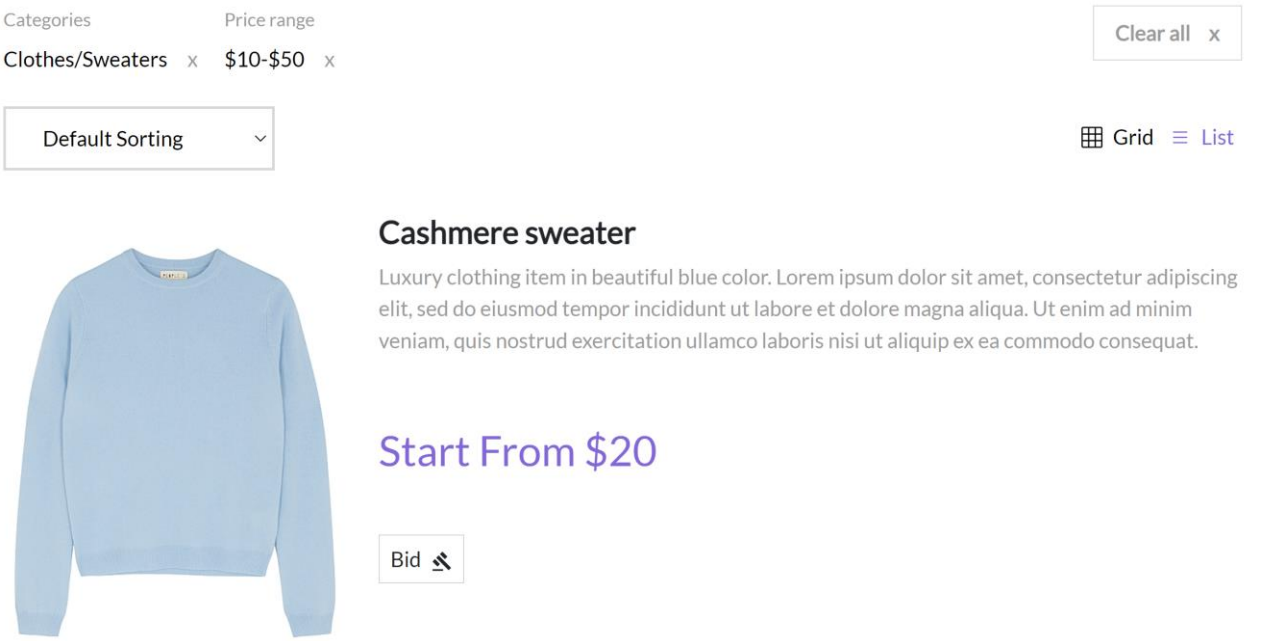
The shop page has filter menus of the left side, options for sorting and view switching on the top, and the items in the center. Initially 6 items are displayed and the “Explore More” button loads additional items on click. The button is not displayed if there are no additional items to show. By default, sorting is alphabetical, items from all categories are display, and the initial price range is from $0 to $200.



The categories menu allows users to choose which category/categories they want to browse. On click, a filter for that category is applied and all active filters are shown at the top, with options to deselect each filter individually, as well as clear all filters. When a category is clicked, it is also expanded and displays all existing subcategories, as well as the number of items within each subcategory. Multiple categories and/or subcategories can be selected at the same time. Price filter can be either adjusted using the two input fields, or the slider below them.

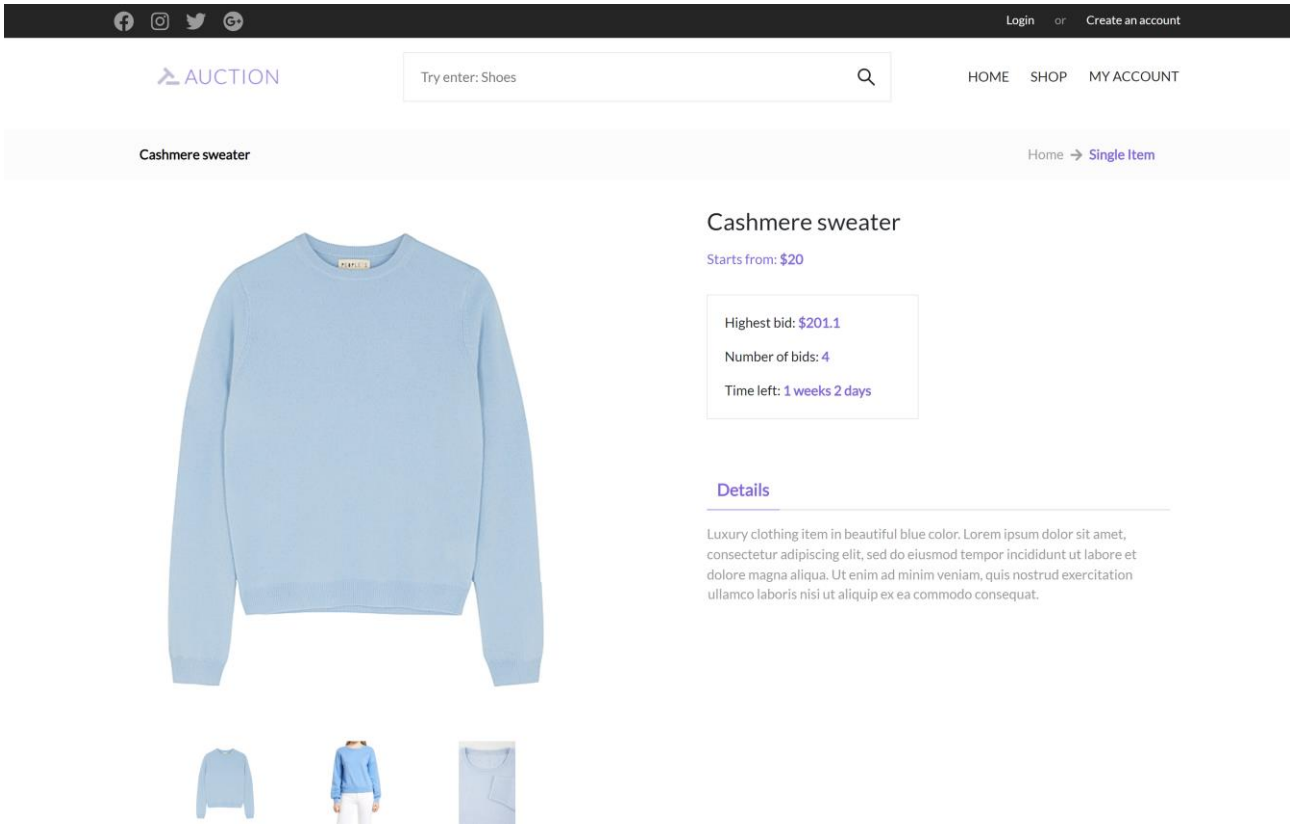


Sorting has the following 4 options, where “Added: New to Old” uses the same sorting as “New Arrivals” earlier, and “Time Left” same as “Last Chance”.

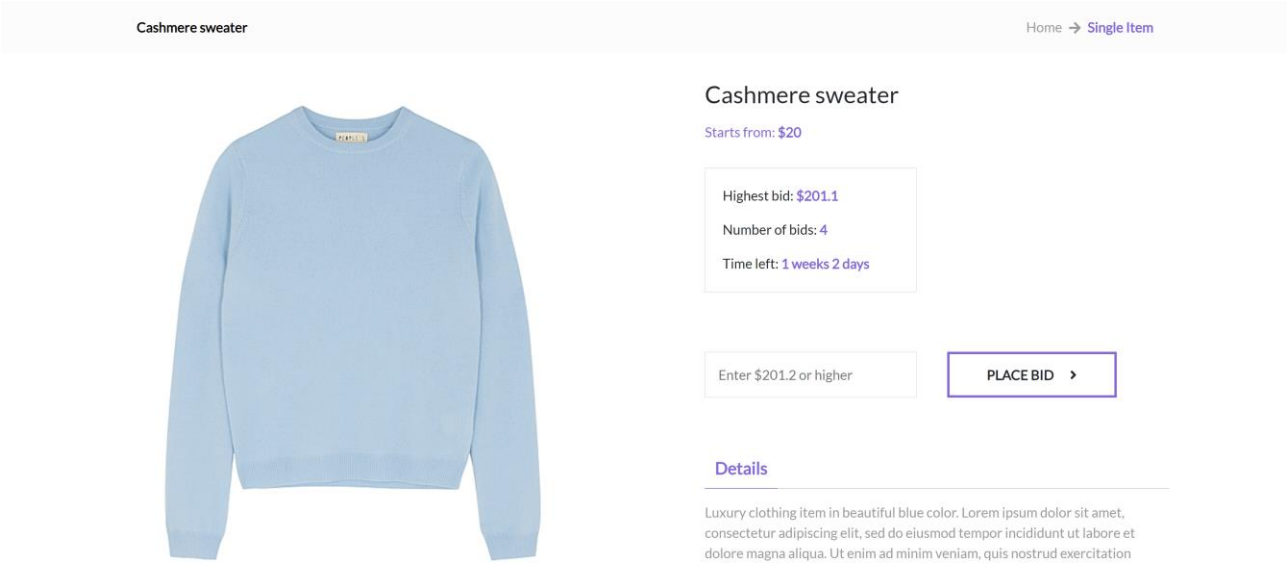


There is an option to switch to a list view, which gives more details about the item itself. It is important to note that when switching between views occurs, the filters should remain the same, i.e. the first item in a grid view should be the same as the first item in a list view.

Item Page:

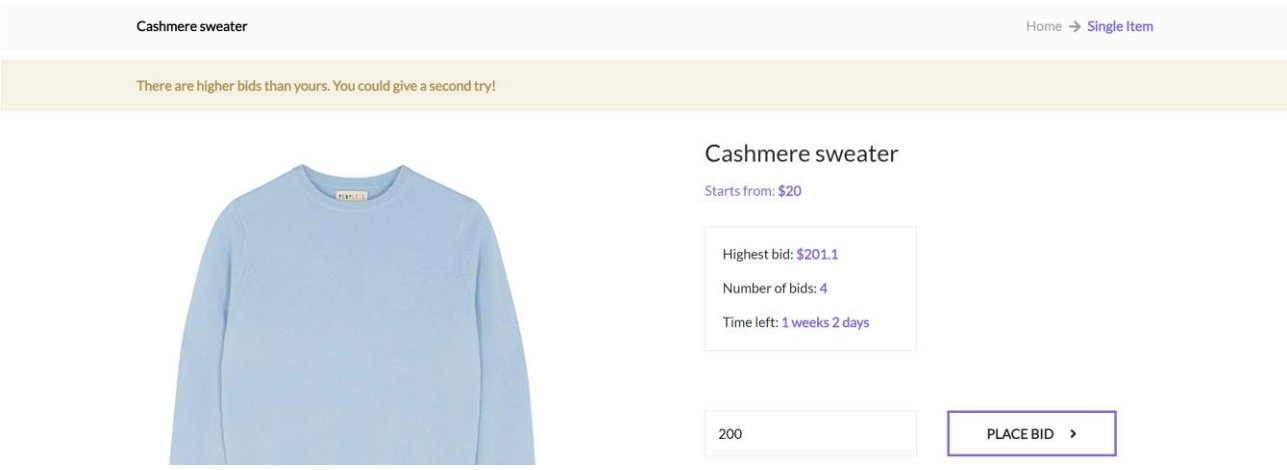


The item page is an overview of the item being sold and is the place where the bidding process takes place. As mentioned earlier, unregistered users (guest users) do have the chance to browse items, but the option of placing a bid is not available to them. However, other information about the item is still displayed. The page consists of a header that has the item name and the breadcrumbs – path taken to reach the item. In the context of this application, it can be from the home page or the shop page. The page content has an image gallery on the left side, and item information on the right side.

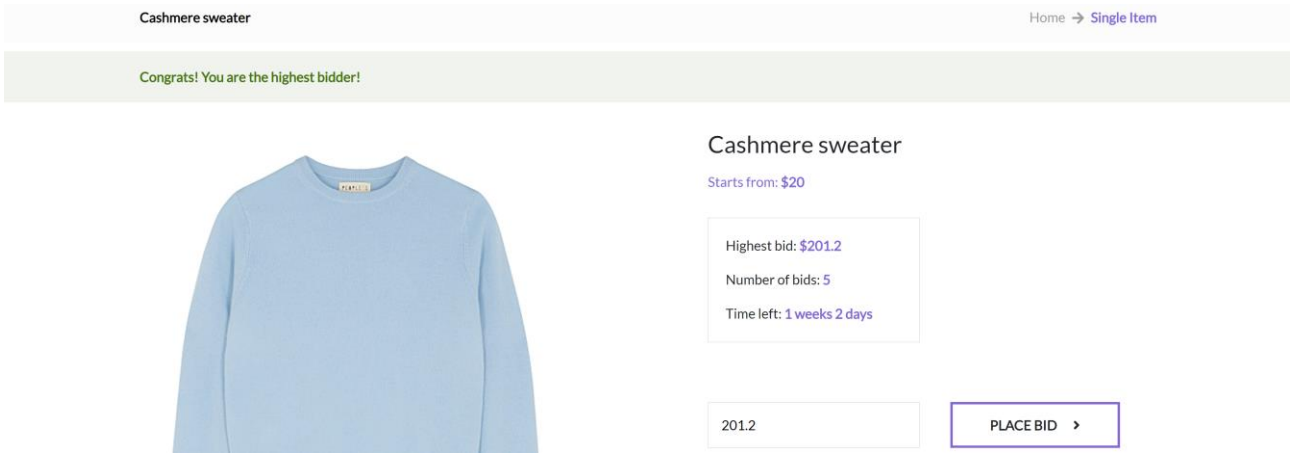


A registered and logged-in user has the option to place a bid for the item, since the input field and button are displayed to them. The placeholder of the input field should suggest an amount that is higher than the starting price, in case there are no bids, or higher than the highest bid otherwise. Upon clicking the “Place Bid” button, the bid amount entered should be validated, i.e. checked whether the amount is indeed higher as suggested by the placeholder. In accordance with the validation response, a message should be shown. If the process is successful, the information about the highest bid and the number of bids should be updated.

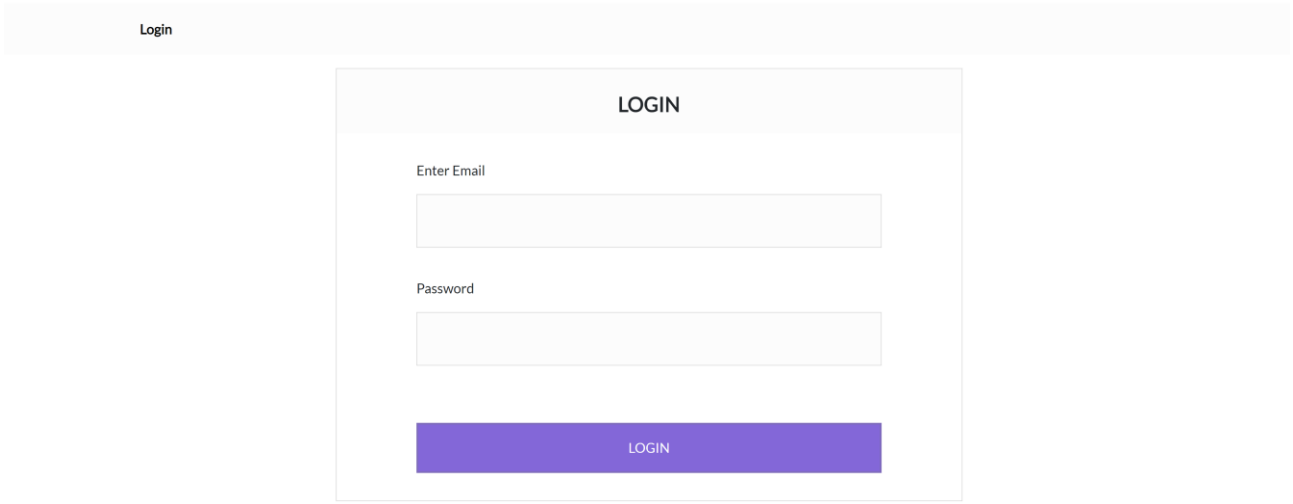
Error message:



Success message with updated information:

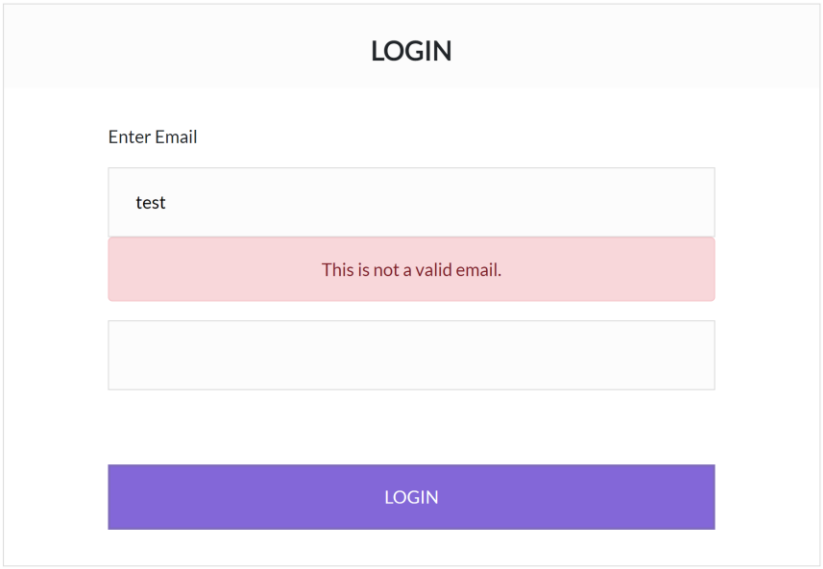


Login Page:



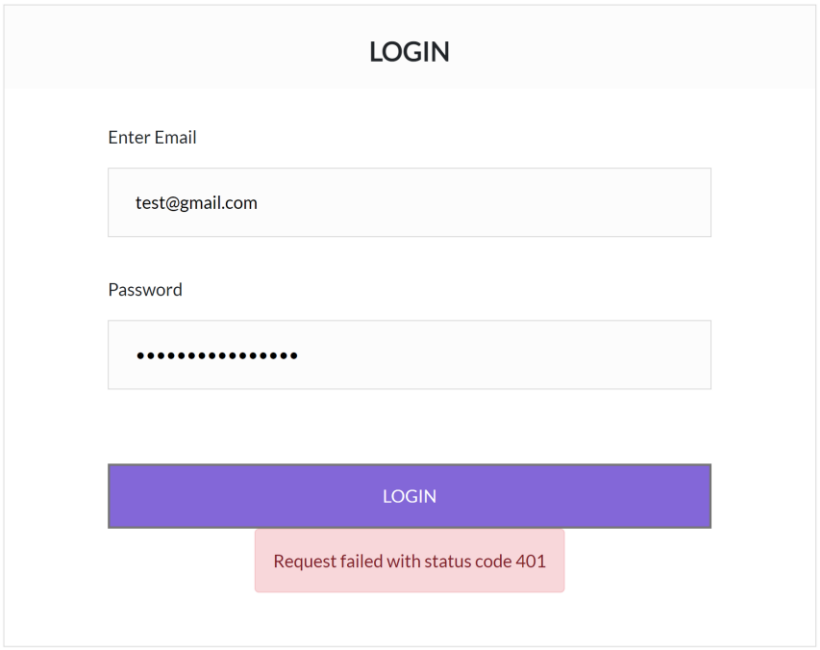
The page consists of the same header seen before, and a form with two input fields that are required for login – email and password.

Alert (invalid):



In case that an invalid email is entered, an alert message should appear and the login button should be disabled.

Alert (unauthorized):

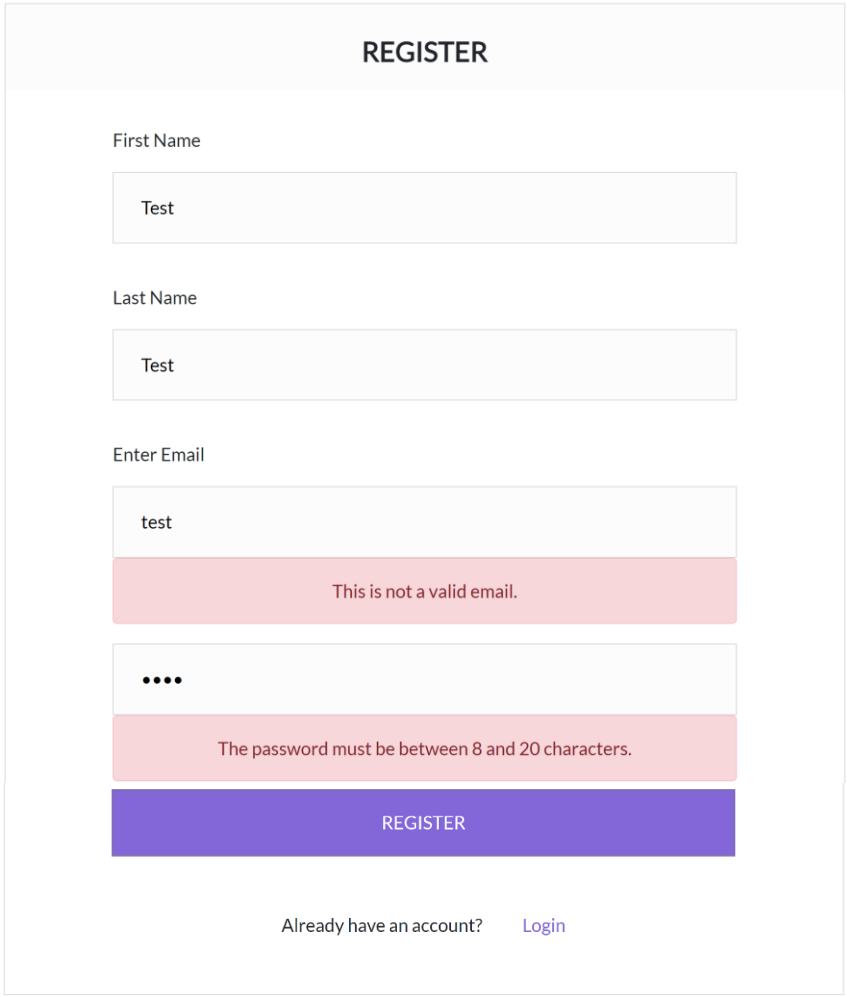


In case that valid information is entered, the login button will be enabled. However, if the information is incorrect, meaning that the credentials are wrong, then after the login button is clicked another alert will appear.

Log Out:

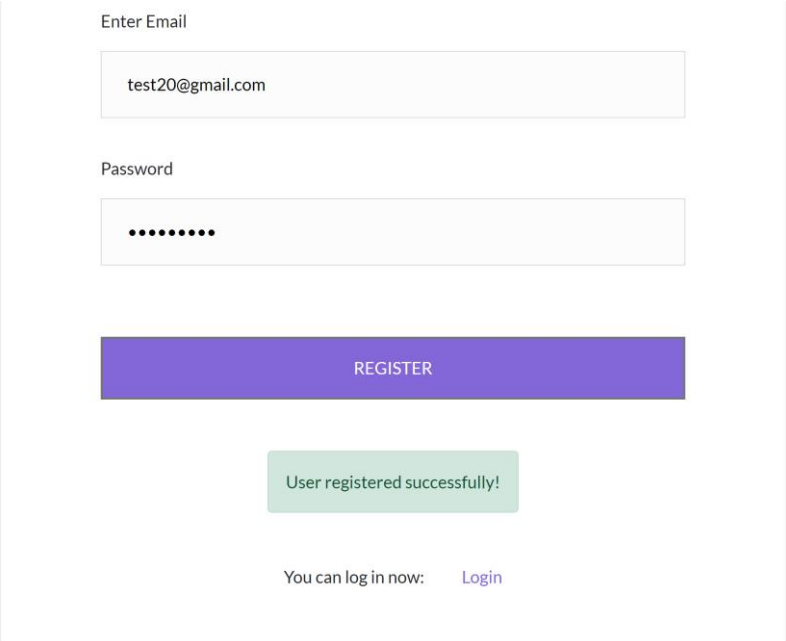
In case of a successful login, the user is redirected to the home page, and in the header instead of “Login” or “Create an account” buttons, the “Log Out” button is shown.

Registration:



The Register page has the identical layout as the Login page, with two additional fields for the first and last name, as well as a link leading to the Login page at the bottom. The email validation work in the same way, and the password input is validated as well now, indicating it should be between 8 and 20 characters. In case of error, the register button is disabled.

Successful register:



If all information is valid and the register button is clicked, a success message will appear.
The descriptions and images provided above summarize all the features which will be tested in this project.

2. Test Plan

2.1. Scope

As mentioned earlier, this project is still under development, so certain features will not be covered with this testing plan. Those are: searching – although a **search** bar is visible on the page it is not functional and the same applies for the **newsletter** box in the footer, **account page** – even though some parts of this page are implemented, it is not complete and therefore will not be included in the test, **social media links** in the header – although they are placed on the website for visual purposes, the links do not have an actual address behind them since this is a hypothetical product.

2.2. Testing Environment and Tools

Testing will be conducted using Java and jUnit and Selenium as supporting tools for automation testing. Besides this, the browser used is Google Chrome, indicating the need for a corresponding driver. Lastly, the IDE used for writing the code will be Eclipse (Admir) and IntelliJ (Matej).

3. Test Execution

3.1. Header and Footer Interaction

User clicks on any of the links/buttons located in the header, navigation bar and footer.

3.1.1. Test Header Links

Test Name: Test header links				
Description: Checking that every button leads to a correct link				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click on each link except the social media links 2. Assert that the user is taken to the correct page by checking the URL	**Test Data:**	**Expected Result:** The user is taken to the correct new page when clicking on the link.	**Actual Result:** The user is taken to the correct new page when clicking on the link.	**Status:** PASS
Notes: Social media links are nonexistent and thus not tested				

```
@Test
void testHeaderLinks() throws InterruptedException {
    WebElement loginLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    loginLink.click();
    assertEquals("https://auctionapp.krilasevic.me/login", webDriver.getCurrentUrl());

    WebElement registerLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[2]"));
    registerLink.click();
    assertEquals("https://auctionapp.krilasevic.me/register", webDriver.getCurrentUrl());
}
```

3.1.2. Test Navigation Bar Links

Test Name: Test navigation bar links				
Description: Checking that every button leads to a correct link				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps:	**Test Data:**	**Expected Result:**	**Actual Result:**	**Status:**

1. Click on each link including the logo and the on-hover dropdown “Account” menu 2. Assert that the user is taken to the correct page by checking the URL		The user is taken to the correct new page when clicking on the link.	The user is taken to the correct new page when clicking on the link.	PASS
Notes: Testing the logo link implied testing that it leads to the same location as the “Home” link				

```
@Test
void testNavbarLinks() throws InterruptedException {
    WebElement logoLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[1]/a/img"));
    logoLink.click();
    assertEquals("https://auctionapp.krilasevic.me/home", webDriver.getCurrentUrl());

    WebElement homeLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/a[1]"));
    homeLink.click();
    assertEquals("https://auctionapp.krilasevic.me/home", webDriver.getCurrentUrl());

    WebElement shopLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/a[2]"));
    shopLink.click();
    assertEquals("https://auctionapp.krilasevic.me/shop/0", webDriver.getCurrentUrl());

    WebElement accountLink = webDriver.findElement(By.xpath("/html/body/div/div/div[2]/div/div[3]/div/button/a"));
    accountLink.click();
    assertEquals("https://auctionapp.krilasevic.me/register", webDriver.getCurrentUrl());

    Actions action = new Actions(webDriver);
    action.moveToElement(accountLink).perform();
    Thread.sleep(1000);

    WebElement accountProfileLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/div/div/a[1]/a"));
    accountProfileLink.click();
    assertEquals("https://auctionapp.krilasevic.me/register", webDriver.getCurrentUrl());

    action.moveToElement(accountLink).perform();
    Thread.sleep(1000);

    WebElement accountSellerLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/div/div/a[2]/a"));
    accountSellerLink.click();
    assertEquals("https://auctionapp.krilasevic.me/register", webDriver.getCurrentUrl());

    action.moveToElement(accountLink).perform();
    Thread.sleep(1000);

    WebElement accountBidsLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/div/div/a[3]/a"));
    accountBidsLink.click();
    assertEquals("https://auctionapp.krilasevic.me/register", webDriver.getCurrentUrl());

    action.moveToElement(accountLink).perform();
    Thread.sleep(1000);

    WebElement accountSettingsLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/div/div/a[4]/a"));
    accountSettingsLink.click();
    assertEquals("https://auctionapp.krilasevic.me/register", webDriver.getCurrentUrl());
}
```

3.1.3. Test Footer Links

Test Name: Test footer links
Description: Checking that every button leads to a correct link

Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click on each link in the first footer column 2. Assert that the user is taken to the correct page by checking the URL	**Test Data:**	**Expected Result:** The user is taken to the correct new page when clicking on the link.	**Actual Result:** The user is taken to the correct new page when clicking on the link.	**Status:** PASS
Notes:				

```
@Test
void testFooterLinks() throws InterruptedException {
    WebElement aboutUsLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[4]/div/div/div[1]/div/a[1]"));
    aboutUsLink.click();
    assertEquals("https://auctionapp.krilasevic.me/about", webDriver.getCurrentUrl());

    WebElement termsAndConditionsLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[4]/div/div/div[1]/div/a[2]"));
    termsAndConditionsLink.click();
    assertEquals("https://auctionapp.krilasevic.me/terms", webDriver.getCurrentUrl());

    WebElement privacyAndPolicyLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[4]/div/div/div[1]/div/a[3]"));
    privacyAndPolicyLink.click();
    assertEquals("https://auctionapp.krilasevic.me/privacy", webDriver.getCurrentUrl());
}
```

3.2. Login and Registration

User clicks on the login and registration links and fills in the respective forms, taking into account both valid and invalid information. User logs out.

3.2.1. Test Login Link on Register Page

Test Name: Test login link on register page				
Description: Check if the login link on the register page leads to the correct URL.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click create an account. 2. Click login. 3. Check if the login URL is correct.	**Test Data:**	**Expected Result:** The login link leads to the login page.	**Actual Result:** The login link led to the login page.	**Status:** PASS
Notes:				

```java
@Test
void testLoginLinkOnRegisterPage() throws InterruptedException {
    WebElement register = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[2]"));
    register.click();
    Thread.sleep( millis: 2000);

    JavascriptExecutor js = (JavascriptExecutor) webDriver;
    js.executeScript( script: "window.scrollBy(0,500)",  ...args: "");
    Thread.sleep( millis: 2000);

    WebElement loginLink = webDriver.findElement(By.className("Forms_linkToLogin__VHCBU"));
    loginLink.click();
    Thread.sleep( millis: 2000);

    assertEquals( expected: "https://auctionapp.krilasevic.me/login", webDriver.getCurrentUrl());
}
```

3.2.2. Test Registration – Success

Test Name: Test registration - success				
Description: Check if user can register a fresh account.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click create an account. 2. Send adequate keys to register form. 3. Click register. 4. Check if user registered successfully.	**Test Data:** Name: matej Surname: mujezinovic Email: matej.mujezinovic@gmail.com Password: 123456789	**Expected Result:** The user registers successfully.	**Actual Result:** The user registers successfully.	**Status:** PASS
Notes: The account registered with this data will be used for further login testing.				

```java
@Test
void testRegisterSuccessful() throws InterruptedException {
    WebElement register = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[2]"));
    register.click();
    Thread.sleep( millis: 2000);

    WebElement name = webDriver.findElement(By.name("name"));
    WebElement surname = webDriver.findElement(By.name("surname"));
    WebElement email = webDriver.findElement(By.name("email"));
    WebElement password = webDriver.findElement(By.name("password"));
    name.sendKeys( ...keysToSend: "newName");
    surname.sendKeys( ...keysToSend: "newSurname");
    email.sendKeys( ...keysToSend: "newName.newSurname.test@gmail.com");
    password.sendKeys( ...keysToSend: "123456789");
    Thread.sleep( millis: 2000);

    JavascriptExecutor js = (JavascriptExecutor) webDriver;
    js.executeScript( script: "window.scrollBy(0,500)", ...args: "");
    Thread.sleep( millis: 2000);

    WebElement registerButton = wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/button")));
    registerButton.click();
    Thread.sleep( millis: 2000);

    WebElement successAlert = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/div[6]/div"));
    assertEquals( expected: "User registered successfully!", successAlert.getText());
}
```

3.2.3. Test Registration – Disabled

Test Name: Test registration - disabled				
Description: Check if guest user is alerted in case register form is not correct.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click create an account. 2. Send inadequate keys to register form. 4. Check if user received an alert. 5. Check if register button is disabled.	**Test Data:** Name: matej Surname: matej Email: matej Password: matej	**Expected Result:** The register button will be disabled.	**Actual Result:** The register button was disabled.	**Status:** PASS
Notes:				

```java
@Test
void testRegisterDisabled() throws InterruptedException {
    WebElement register = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[2]"));
    register.click();
    Thread.sleep( millis: 2000);

    WebElement name = webDriver.findElement(By.name("name"));
    WebElement surname = webDriver.findElement(By.name("surname"));
    WebElement email = webDriver.findElement(By.name("email"));
    WebElement password = webDriver.findElement(By.name("password"));

    name.sendKeys( ...keysToSend: "matej");
    surname.sendKeys( ...keysToSend: "matej");
    email.sendKeys( ...keysToSend: "matej");
    password.sendKeys( ...keysToSend: "matej");

    Actions action = new Actions(webDriver);
    action.moveToElement(email).click().perform();

    WebElement emailAlert = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/div[4]/div/div"));
    WebElement passwordAlert = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/div[5]/div/div"));
    assertEquals( expected: "This is not a valid email.", emailAlert.getText());
    assertEquals( expected: "The password must be between 8 and 20 characters.", passwordAlert.getText());

    WebElement registerButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/button"));
    assertFalse(registerButton.isEnabled());
}
```

3.2.4. Test Login – Success

Test Name: Test login - success				
Description: Check if user can log into his account.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page. User must have a registered account to log in.				
Test Steps: 1. Click login. 2. Send adequate email and password keys to login form. 3. Click login button. 4. Check if user logged in by seeing if “Log Out” button appears.	**Test Data:** Email: matej.mujezinovic@gmail.com Password: 123456789	**Expected Result:** User logs in and log out button appears.	**Actual Result:** User logs in and log out button appears.	**Status:** PASS
Notes:				

```java
@Test
void testLoginSuccessful() throws InterruptedException {
    WebElement login = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    login.click();
    Thread.sleep( millis: 2000);

    WebElement email = webDriver.findElement(By.name("email"));
    WebElement password = webDriver.findElement(By.name("password"));
    email.sendKeys( ...keysToSend: "matej.mujezinovic@gmail.com");
    password.sendKeys( ...keysToSend: "123456789");
    Thread.sleep( millis: 2000);

    WebElement loginButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/button"));
    loginButton.click();
    Thread.sleep( millis: 2000);

    WebElement logout = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a"));
    assertEquals( expected: "Log Out", logout.getText());
    logout.click();
    Thread.sleep( millis: 2000);
}
```

3.2.5. Test Login – Disabled

Test Name: Test login - disabled				
Description: Check if user is alerted in case login form is not correct.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click login. 2. Send inadequate email and password keys to login form. 3. Check if user received an alert. 4. Check if login button is disabled.	**Test Data:** Email: bademail Password: wrongpassword	**Expected Result:** Login button is disabled.	**Actual Result:** Login button is disabled.	**Status:** PASS
Notes:				

```java
@Test
void testLoginDisabled() throws InterruptedException {
    WebElement login = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    login.click();
    Thread.sleep( millis: 2000);

    WebElement email = webDriver.findElement(By.name("email"));
    WebElement password = webDriver.findElement(By.name("password"));
    email.sendKeys( ...keysToSend: "bademail");
    password.sendKeys( ...keysToSend: "wrongpassword");
    WebElement invalidEmailField = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/div[2]/div/div"));
    assertEquals( expected: "This is not a valid email.", invalidEmailField.getText());

    WebElement loginButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/button"));
    assertFalse(loginButton.isEnabled());
}
```

3.2.6. Test Login - Unauthorized

Test Name: Test login - unauthorized				
Description: Check if login request fails with inadequate login details.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click login. 2. Send incorrect email and password keys to register form. 3. Click login button. 4. Check if login request fails.	**Test Data:** Email: matej @gmail.com Password: wrongpassword	**Expected Result:** Login fails and alert is displayed.	**Actual Result:** Login fails and alert is displayed.	**Status:** PASS
Notes:				

```
@Test
void testLoginUnauthorized() throws InterruptedException {
    WebElement login = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    login.click();
    Thread.sleep( millis: 2000);

    WebElement email = webDriver.findElement(By.name("email"));
    WebElement password = webDriver.findElement(By.name("password"));
    email.sendKeys( ...keysToSend: "matej@gmail.com");
    password.sendKeys( ...keysToSend: "wrongpassword");

    WebElement loginButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/button"));
    loginButton.click();
    Thread.sleep( millis: 2000);

    WebElement unauthorizedAlert = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/div[4]/div"));
    assertEquals( expected: "Request failed with status code 401", unauthorizedAlert.getText());
}
```

3.2.7. Test Log Out

Test Name: Test log out
Description: Check if user can log out.
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.

Test Steps:	Test Data:	Expected Result:	Actual Result:	Status:
1. Click login. 2. Send correct email and password keys to login page. 3. Click login button. 4. Check if user sees logout option. 5. Click logout. 6. Click login again. 7. Check if the login URL is correct.	Email: matej.mujezinovic@gmail.com Password: 123456789	User logs out and is able to access login page again.	User logs out and is able to access login page again.	PASS
Notes:				

```java
@Test
void testLogout() throws InterruptedException {
    WebElement login = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    assertEquals( expected: "Login", login.getText());
    login.click();
    Thread.sleep( millis: 2000);

    WebElement email = webDriver.findElement(By.name("email"));
    WebElement password = webDriver.findElement(By.name("password"));
    email.sendKeys( ...keysToSend: "matej.mujezinovic@gmail.com");
    password.sendKeys( ...keysToSend: "123456789");

    WebElement loginButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/button"));
    loginButton.click();
    Thread.sleep( millis: 2000);

    WebElement logout = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a"));
    assertEquals( expected: "Log Out", logout.getText());
    logout.click();
    Thread.sleep( millis: 2000);

    WebElement newLogin = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    assertEquals( expected: "Login", newLogin.getText());
    Thread.sleep( millis: 2000);

    newLogin.click();
    assertEquals( expected: "https://auctionapp.krilasevic.me/login", webDriver.getCurrentUrl());
}
```

3.3. Home Page Interaction

User, while on the home page, clicks on the categories listed in the categories section. User switches between tabs in the tab section. User views the new arrivals and last chance items.

3.3.1. Test Categories Section

Test Name: Test categories section

Description: Checking that on every category click, the shop page is opened with correct filter applied

Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.

Test Steps:	**Test Data:**	**Expected Result:**	**Actual Result:**	**Status:**
1. Click on each category listed in the categories section of the landing page 2. Assert that the user is taken to the shop page, and the name of the clicked category is listed as the active filter 3. Click on “All Categories” button and assert that there are no active filters		The user is taken to the shop page with correct filter (or no filter) applied.	The user is taken to the shop page with correct filter (or no filter) applied.	PASS

Notes:

```
@Test
void testCategoriesSection() throws InterruptedException {
    Thread.sleep(2000);
    List<WebElement> categories = webDriver.findElements(By.className("CategoriesSection_categoryButton__2oY_0"));

    for (int i = 0; i < categories.size()-1; i++) {
        Thread.sleep(2000);

        List<WebElement> refreshedCategories = webDriver.findElements(By.className("CategoriesSection_categoryButton__2oY_0"));
        String categoryName = refreshedCategories.get(i).getText();
        refreshedCategories.get(i).click();

        Thread.sleep(2000);

        WebElement categoryActiveFilter = webDriver.findElement(By.xpath("/html/body/div/div/div[3]/div/div/div[2]/div[1]/div/div/button"));
        String categoryFilterName = categoryActiveFilter.getText().split(" ")[0];
        assertEquals(categoryName, categoryFilterName);

        webDriver.navigate().back();
    }

    Thread.sleep(2000);
    List<WebElement> refreshedCategories = webDriver.findElements(By.className("CategoriesSection_categoryButton__2oY_0"));
    refreshedCategories.get(refreshedCategories.size()-1).click();
    String allCategoriesUrl = webDriver.getCurrentUrl();

    WebElement shopLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/a[2]"));
    shopLink.click();
    String shopUrl = webDriver.getCurrentUrl();

    assertEquals(shopUrl, allCategoriesUrl);
}
```

3.3.2. Test Tab Switching

Test Name: Test tab switching				
Description: Checking that on tab selection, the selected tab changes appearance and becomes the active tab				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Asserts that “New Arrivals” is the active tab by default 2. Click on the “Last Chance” tab and assert that it is active 3. Go back to the first tab and make the same assertion	**Test Data:**	**Expected Result:** On tab click, the selected tab is marked as active	**Actual Result:** On tab click, the selected tab is marked as active	**Status:** PASS
Notes:				

```
@Test
void testTabSwitching() {
    WebElement newArrivalsTab = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/ol/li[1]"));
    WebElement lastChanceTab = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/ol/li[2]"));

    assertEquals("tab-list-item tab-list-active", newArrivalsTab.getAttribute("class"));

    lastChanceTab.click();
    lastChanceTab = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/ol/li[2]"));
    assertEquals("tab-list-item tab-list-active", lastChanceTab.getAttribute("class"));

    newArrivalsTab.click();
    newArrivalsTab = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/ol/li[1]"));
    assertEquals("tab-list-item tab-list-active", newArrivalsTab.getAttribute("class"));

}
```

3.3.3. Test New Arrivals

Test Name: Test new arrivals				
Description: Checking that items are correctly sorted				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps:	**Test Data:**	**Expected Result:**	**Actual Result:**	**Status:**

1. Click and open the first three items within the “New Arrivals” tab 2. Retrieve the “Time Left” value for each of them 3. Assert that they are in descending order		Items are sorted by the time remaining for their auction, in descending order	Items are sorted by the time remaining for their auction, in descending order	PASS

Notes: Since the actual start date of the auction is not displayed on the website, it was assumed that there is a fixed interval between a start date and an end date, indicating that descending end dates (descending values of time left for auction) are equivalent to ascending start dates, which is actually what “New Arrivals” means

```
@Test
void testNewArrivals() throws InterruptedException {
    int[] hoursLeft = new int[3];
    WebElement timeLeftForItem;
    String timeLeftWithText;
    int timeLeftInHours;

    Thread.sleep(2000);
    WebElement firstItem = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div/div/div/div/div[1]/div/a[1]/img"));
    firstItem.click();
    hoursLeft[0] = getTimeLeft();
    webDriver.navigate().back();

    Thread.sleep(2000);
    WebElement secondItem = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div/div/div/div/div[2]/div/a[1]/img"));
    secondItem.click();
    hoursLeft[1] = getTimeLeft();
    webDriver.navigate().back();

    Thread.sleep(2000);
    WebElement thirdItem = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div/div/div/div/div[3]/div/a[1]/img"));
    thirdItem.click();
    hoursLeft[2] = getTimeLeft();

    assertTrue(hoursLeft[0] > hoursLeft[1]);
    assertTrue(hoursLeft[1] > hoursLeft[2]);
}
```

3.3.4. Test Last Chance

Test Name: Test last chance				
Description: Checking that items are correctly sorted				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps:	**Test Data:**	**Expected Result:**	**Actual Result:**	**Status:**
1. Click and open the first three items within the “Last Chance” tab		Items are sorted by the time remaining	Items are sorted by the time remaining	PASS

2. Retrieve the “Time Left” value for each of them 3. Assert that they are in ascending order		for their auction, in ascending order	for their auction, in ascending order	
Notes:				

```
@Test
void testLastChance() throws InterruptedException {

    int[] hoursLeft = new int[3];

    clickLastChanceTab();
    Thread.sleep(2000);
    WebElement firstItem = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div/div/div/div/div[1]/div/a[1]/img"));
    firstItem.click();
    hoursLeft[0] = getTimeLeft();
    webDriver.navigate().back();

    clickLastChanceTab();
    Thread.sleep(2000);
    WebElement secondItem = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div/div/div/div/div[2]/div/a[1]/img"));
    secondItem.click();
    hoursLeft[1] = getTimeLeft();
    webDriver.navigate().back();

    clickLastChanceTab();
    Thread.sleep(2000);
    WebElement thirdItem = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div/div/div/div/div[3]/div/a[1]/img"));
    thirdItem.click();
    hoursLeft[2] = getTimeLeft();

    assertTrue(hoursLeft[0] < hoursLeft[1]);
    assertTrue(hoursLeft[1] < hoursLeft[2]);
}
```

3.3.5. Helper Methods

Below are helper methods used in this scenario, added in order to reduce code redundancy.

```
void clickLastChanceTab() throws InterruptedException {
    Thread.sleep(2000);
    WebElement lastChanceTab = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/ol/li[2]"));
    lastChanceTab.click();
}

int getTimeLeft() throws InterruptedException {
    Thread.sleep(2000);
    WebElement timeLeftForItem = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div[2]/div[1]/p[3]/span"));
    String timeLeftWithText = timeLeftForItem.getText();
    int timeLeftInHours = convertTimeLeftToHours(timeLeftWithText);
    return timeLeftInHours;
}

int convertTimeLeftToHours(String timeLeft) {
    String[] elements = timeLeft.split("\\s+");
    int hours;
    System.out.println(elements[1]);
    if (elements[1].equals("weeks")) {
        hours = Integer.parseInt(elements[0]) * 168 + Integer.parseInt(elements[2]) * 24;
        return hours;
    } else if (elements[1].equals("days")) {
        hours = Integer.parseInt(elements[0]) * 24 + Integer.parseInt(elements[2]);
        return hours;
    } else {
        hours = Integer.parseInt(elements[0]);
        return hours;
    }
}
```

3.4. Shop Page Interaction

User, while on the shop page, uses the categories, subcategories and price filters. User selects different sorting options. User switches between the grid and list view. User removes some and/or all active filters.

3.4.1. Test Explore More Button

Test Name: Test explore more button				
Description: Checking that additional items are loaded when the button is clicked				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Go to the shop page 2. Retrieve the number of items currently displayed 3. Scroll down 4. Click on the “Explore More” button 5. Assert that the number of items has increased	**Test Data:**	**Expected Result:** On button click, the number of displayed items is increased	**Actual Result:** On button click, the number of displayed items is increased	**Status:** PASS
Notes:				

```java
@Test
void testExploreMore() throws InterruptedException {
    openShopPage();

    List<WebElement> listedItems = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
    int noOfItemsBefore = listedItems.size();

    JavascriptExecutor js = (JavascriptExecutor) webDriver;
    js.executeScript("window.scrollBy(0,500)", "");
    Thread.sleep(2000);

    WebElement exploreMoreButton = webDriver.findElement(By.className("ShopPageItems_exploreMoreButton__1nQKr"));
    exploreMoreButton.click();
    Thread.sleep(2000);

    listedItems = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
    int noOfItemsAfter = listedItems.size();

    assertTrue(noOfItemsBefore < noOfItemsAfter);
}
```

3.4.2. Test Category Filters

Test Name: Test category filters
Description: By using the category filter menu, checking that on each category click the correct filter is applied, and then removed
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.

Test Steps:	Test Data:	Expected Result:	Actual Result:	Status:
1. Go to the shop page For each category displayed: 2. Retrieve category name 3. Click on category 4. Retrieve active filter name 5. Assert that the category and filter name are equal 6. Click on the filter to remove it 7. In the end, assert that no filters are displayed		On category click, the corresponding filter is applied and removing each filter is functional so in the end there are no active filters	On category click, the corresponding filter is applied and removing each filter is functional so in the end there are no active filters	PASS

Notes:

```
@Test
void testCategoryFilters() throws InterruptedException {
    openShopPage();

    List<WebElement> categories = webDriver.findElements(By.className("Category_collapsibleCategory__3wERX"));

    for (int i = 0; i < categories.size(); i++) {
        WebElement currentCategory = categories.get(i);
        String categoryName = currentCategory.getText().split("\\s+")[0];
        currentCategory.click();

        WebElement categoryActiveFilter = webDriver.findElement(By.xpath("/html/body/div/div/div[3]/div/div/div[2]/div[1]/div/div/button"));
        String categoryFilterName = categoryActiveFilter.getText().split(" ")[0];
        assertEquals(categoryName, categoryFilterName);

        currentCategory.click();
        categoryActiveFilter.click();
    }

    assertThrows(NoSuchElementException.class,
            () -> {webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div/div/div[2]/div[1]/div/div/p"));});
}
```

3.4.3. Test Subcategory Filters

Test Name: Test subcategory filters
Description: By using the category filter menu, checking that subcategories are displayed on category click, that correct subcategory filter is applied and correct number of items shown, and on filter removal, that the checkbox used to apply the subcategory filter is no longer checked

Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.

Test Steps:	**Test Data:**	**Expected Result:**	**Actual Result:**	**Status:**
1. Go to the shop page 2. Click on the first category that appears and retrieve its name For each subcategory displayed: 3. Retrieve subcategory name and number of items 4. Click on the checkbox 5. Retrieve active filter name 6. Assert that the filter corresponds to the category and subcategory name 7. If items exist within a subcategory, check that the correct number is displayed 8. Click on the filter to remove it 9. Check that the checkbox is not selected after removal 10. In the end, assert that no filters are displayed		On category click, subcategories are shown. On subcategory checkbox click, the corresponding filter is applied and correct number of items is shown. Removing each filter is functional and deselects the checkbox. In the end there are no active filters.	On category click, subcategories are shown. On subcategory checkbox click, the corresponding filter is applied and correct number of items is shown. Removing each filter is functional and deselects the checkbox. In the end there are no active filters.	PASS

Notes:

```java
@Test
void testSubcategoryFilters() throws InterruptedException {
    openShopPage();

    WebElement category = webDriver.findElement(By.className("Category_collapsibleCategory__3wERX"));
    String categoryName = category.getText().split("\\s+")[0];
    category.click();

    List<WebElement> subcategories = webDriver.findElements(By.className("Subcategory_subcategoryItem__1tROv"));

    for (int i = 0; i < subcategories.size(); i++) {
        WebElement currentSubcategory = subcategories.get(i);
        String subcategoryName = currentSubcategory.getText().split("\\s+")[0];
        String noOfItemsInSubcategoryString = currentSubcategory.getText().split("\\s+")[1];
        int noOfItemsInSubcategory = Integer.parseInt(noOfItemsInSubcategoryString.replaceAll("[^0-9]", ""));
        WebElement currentSubcategoryCheckbox = currentSubcategory.findElement(By.tagName("input"));
        currentSubcategoryCheckbox.click();

        WebElement categoryActiveFilter = webDriver.findElement(By.xpath("/html/body/div/div/div[3]/div/div/div[2]/div[1]/div/div/button"));
        String categoryFilterName = categoryActiveFilter.getText().split(" ")[0];
        assertEquals(categoryName + "/" + subcategoryName, categoryFilterName);

        if (noOfItemsInSubcategory > 0) {
            Thread.sleep(2000);
            List<WebElement> itemsInSubcategory = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
            assertEquals(noOfItemsInSubcategory, itemsInSubcategory.size());
        }

        categoryActiveFilter.click();
        assertFalse(currentSubcategoryCheckbox.isSelected());
    }

    assertThrows(NoSuchElementException.class,
            () -> {webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div/div/div[2]/div[1]/div/div/p"));});
}
```

3.4.4. Test Price Filter

Test Name: Test price filter				
Description: By using the price filter slider, checking that correct price range and price filter are displayed, as well as checking that items indeed fall within the selected price range				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Go to the shop page 2. Use the price filter slider to adjust the minimum and maximum price to some arbitrary values 3. Retrieve those values from the input fields above the slider 4. Check that the price range is correct	**Test Data:**	**Expected Result:** Using the slider, the price range can be adjusted. Correct range, average price and filter are displayed. Items displayed fall within the selected range.	**Actual Result:** Using the slider, the price range can be adjusted. Correct range, average price and filter are displayed. Items displayed fall within the selected range.	**Status:** PASS

5. Check that the average price is correct 6. Check that the applied price filter is correct 7. Sort elements by price, from low to high 8. Check that the price of the first item is greater than or equal to the minimum price 9. Check that the price of the last item is greater than or equal to the maximum price				

Notes:

```
@Test
void testPriceFilter() throws InterruptedException {
    openShopPage();
    selectPrices();

    WebElement minPriceFilterInput = webDriver.findElement(By.className("PriceMenu_minPriceInput__2GC2p"));
    WebElement maxPriceFilterInput = webDriver.findElement(By.className("PriceMenu_maxPriceInput__2RsAW"));
    double minPrice = Double.parseDouble(minPriceFilterInput.getAttribute("value"));
    double maxPrice = Double.parseDouble(maxPriceFilterInput.getAttribute("value"));

    WebElement priceRange = webDriver.findElement(By.className("PriceMenu_priceRange__3eMWN"));
    String expectedPriceRange = "$" + minPrice + "-$" + maxPrice;
    assertEquals(expectedPriceRange, priceRange.getText());

    WebElement averagePrice = webDriver.findElement(By.className("PriceMenu_priceAverage__1lTg9"));
    double avgPriceDouble = ((minPrice + maxPrice) / 2);
    DecimalFormat decimalFormat = new DecimalFormat("0.00");
    String avgPrice = "The average price is $" + decimalFormat.format(avgPriceDouble);
    assertEquals(avgPrice, averagePrice.getText());

    WebElement priceActiveFilter = webDriver.findElement(By.xpath("/html/body/div/div/div[3]/div/div/div[2]/div[1]/div/div/button"));
    String priceFilterValue = priceActiveFilter.getText().split(" ")[0];
    assertEquals(expectedPriceRange, priceFilterValue);

    Select sortOptions = new Select(webDriver.findElement(By.className("ShopPageItems_sortDropdown__2zRR6")));
    sortOptions.selectByVisibleText("Price: Low to High");
    Thread.sleep(2000);

    List<WebElement> listedItems = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
    WebElement startingPriceContainerFirst = listedItems.get(0).findElement(By.className("Item_startPrice__1QWpu"));
    WebElement startingPriceContainerLast = listedItems.get(listedItems.size()-1).findElement(By.className("Item_startPrice__1QWpu"));
    Double startingPriceFirst = Double.parseDouble(startingPriceContainerFirst.getText().substring(12));
    Double startingPriceLast = Double.parseDouble(startingPriceContainerLast.getText().substring(12));

    assertTrue(startingPriceFirst >= minPrice);
    assertTrue(startingPriceLast <= maxPrice);
}
```

3.4.5. Test Selecting Multiple Filters and Clearing All

Test Name: Test selecting multiple filters and clearing all

Description: Select all three possible types of filters and check that when clear all is clicked, all of them are removed				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Go to the shop page 2. Select first two categories to apply filter 3. Select first subcategory to apply filter 4. Use the price slider to apply filter 5. Click the “Clear All” button 6. Assert that no active filters are displayed	**Test Data:**	**Expected Result:** After “Clear All” is clicked, no filters are displayed as active.	**Actual Result:** After “Clear All” is clicked, no filters are displayed as active.	**Status:** PASS
Notes:				

```java
@Test
void testMultipleFiltersAndClearAll() throws InterruptedException {
    openShopPage();

    List<WebElement> categories = webDriver.findElements(By.className("Category_collapsibleCategory__3wERX"));
    String categoryName1 = categories.get(0).getText().split("\\s+")[0];
    String categoryName2 = categories.get(1).getText().split("\\s+")[0];
    categories.get(0).click();
    categories.get(1).click();

    WebElement subcategory = webDriver.findElement(By.className("Subcategory_subcategoryItem__1tROv"));
    String subcategoryName = subcategory.getText().split("\\s+")[0];
    WebElement subcategoryCheckbox = subcategory.findElement(By.tagName("input"));
    subcategoryCheckbox.click();

    JavascriptExecutor js = (JavascriptExecutor) webDriver;
    js.executeScript("window.scrollBy(0,300)", "");
    Thread.sleep(2000);

    selectPrices();
    WebElement minPriceFilterInput = webDriver.findElement(By.className("PriceMenu_minPriceInput__2GC2p"));
    WebElement maxPriceFilterInput = webDriver.findElement(By.className("PriceMenu_maxPriceInput__2RsAW"));
    double minPrice = Double.parseDouble(minPriceFilterInput.getAttribute("value"));
    double maxPrice = Double.parseDouble(maxPriceFilterInput.getAttribute("value"));

    js.executeScript("window.scrollBy(0,-300)", "");
    Thread.sleep(2000);

    List<WebElement> allActiveFilters = webDriver.findElements(By.className("ActiveFilters_filter__xHUuV"));
    String[] activeFilterNames = new String[3];
    for (int i = 0; i < allActiveFilters.size(); i++) {
        activeFilterNames[i] = allActiveFilters.get(i).getText().split(" ")[0];
    }

    String expectedCategoryFilterName = categoryName2;
    String expectedSubcategoryFilterName = categoryName1 + "/" + subcategoryName;
    String expectedPriceFilterName = "$" + minPrice + "-$" + maxPrice;

    assertEquals(expectedCategoryFilterName, activeFilterNames[0]);
    assertEquals(expectedSubcategoryFilterName, activeFilterNames[1]);
    assertEquals(expectedPriceFilterName, activeFilterNames[2]);

    WebElement clearAllButton = webDriver.findElement(By.className("ActiveFilters_clearAllButton__3lJdT"));
    clearAllButton.click();

    assertThrows(NoSuchElementException.class,
            () -> {webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div/div/div[2]/div[1]/div/div/p"));});
}
```

3.4.6. Test Selecting Multiple Filters and Sorting

Test Name: Test selecting multiple filters and sorting				
Description: Check whether sorting options are applied correctly, in combination with applied filters				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Go to the shop page 2. Select first two categories to apply filter	**Test Data:**	**Expected Result:** Default sorting is alphabetical. Selecting options to sort by price	**Actual Result:** Default sorting is alphabetical. Selecting options to sort by price order	**Status:** PASS

3. Select first subcategory to apply filter 4. Use the price slider to apply filter 5. Retrieve the names of first three items displayed 6. Check if they are in alphabetical order 7. Select the “Price: Low to High” sorting option 8. Retrieve the starting prices for first three items displayed 9. Compare them and assert that they are in ascending order 10. Select the “Price: High to Low” sorting option 11. Retrieve the starting prices for first three items displayed 12. Compare them and assert that they are in descending order		order the items in a correct way.	the items in a correct way.	
Notes: Default sorting should be alphabetical. Sorting by “Added: New to Old” and “Time Left” correspond to “New Arrivals” and “Last Chance”, which have already been tested. Alphabetical sorting is the default				

```java
@Test
void testMultipleFiltersAndSorting() throws InterruptedException {
    openShopPage();
    selectMultipleFilters();

    String[] names = new String[3];
    List<WebElement> listedItems = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
    for (int i = 0; i < 3; i++) {
        WebElement item = listedItems.get(i);
        WebElement nameContainer = item.findElement(By.className("Item_title__3so5b"));
        names[i] = nameContainer.getText();
    }

    assertTrue(names[0].compareTo(names[1]) < 0);
    assertTrue(names[1].compareTo(names[2]) < 0);

    Select sortOptions = new Select(webDriver.findElement(By.className("ShopPageItems_sortDropdown__2zRR6")));
    sortOptions.selectByVisibleText("Price: Low to High");
    Thread.sleep(2000);

    Double[] prices = new Double[3];
    listedItems = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
    for (int i = 0; i < 3; i++) {
        WebElement item = listedItems.get(i);
        WebElement startingPriceContainer = item.findElement(By.className("Item_startPrice__1QWpu"));
        prices[i] = Double.parseDouble(startingPriceContainer.getText().substring(12));
    }

    assertTrue(prices[0] < prices[1]);
    assertTrue(prices[1] < prices[2]);

    sortOptions.selectByVisibleText("Price: High to Low");
    Thread.sleep(2000);

    prices = new Double[3];
    listedItems = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
    for (int i = 0; i < 3; i++) {
        WebElement item = listedItems.get(i);
        WebElement startingPriceContainer = item.findElement(By.className("Item_startPrice__1QWpu"));
        prices[i] = Double.parseDouble(startingPriceContainer.getText().substring(12));
    }

    assertTrue(prices[0] > prices[1]);
    assertTrue(prices[1] > prices[2]);
}
```

3.4.7. Test Selecting Multiple Filters and Switching Views

Test Name: Test selecting multiple filters and switching views				
Description: Check whether buttons to switch views function, and whether additional elements are present in list view				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Go to the shop page	**Test Data:**	**Expected Result:** Default view is grid view. Switching views	**Actual Result:** Default view is grid view. Switching views changed	**Status:** PASS

2. Select first two categories to apply filter 3. Select first subcategory to apply filter 4. Use the price slider to apply filter 5. Check that the default view is the grid view 6. Retrieve the name of the first item displayed 7. Check that after clicking the list view button, the button becomes active 8. Retrieve the name of the first item displayed 9. Check that the names are the same 10. Check that the description and bid button appear in list view		changed grid/list buttons to active. The first item is the same in both views. Description and bid button are shown in list view.	grid/list buttons to active. The first item is the same in both views. Description and bid button are shown in list view.	
Notes:				

```
@Test
void testMultipleFiltersAndGridListSwitching() throws InterruptedException {
    openShopPage();
    selectMultipleFilters();

    List<WebElement> listedItemsGridView = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));
    WebElement firstItemGridView = listedItemsGridView.get(0);
    WebElement secondItemGridView = listedItemsGridView.get(1);
    String firstItemGridViewName = firstItemGridView.findElement(By.className("Item_title__3so5b")).getText();
    String secondItemGridViewName = secondItemGridView.findElement(By.className("Item_title__3so5b")).getText();

    WebElement gridViewButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div/div/div[2]/div[2]/div/button[1]"));
    WebElement listViewButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div/div/div[2]/div[2]/div/button[2]"));

    assertEquals("ShopPageItems_gridListButtonActive__26AAN", gridViewButton.getAttribute("class"));

    listViewButton.click();
    assertEquals("ShopPageItems_gridListButtonActive__26AAN", listViewButton.getAttribute("class"));

    List<WebElement> listedItemsListView = webDriver.findElements(By.className("ListItem_itemContainer__2s8p_"));
    WebElement firstItemListView = listedItemsListView.get(0);
    WebElement secondItemListView = listedItemsListView.get(1);
    String firstItemListViewName = firstItemListView.findElement(By.className("ListItem_title__1l8Nk")).getText();
    String secondItemListViewName = secondItemListView.findElement(By.className("ListItem_title__1l8Nk")).getText();

    assertEquals(firstItemGridViewName, firstItemListViewName);
    assertEquals(secondItemGridViewName, secondItemListViewName);

    WebElement description = firstItemListView.findElement(By.className("ListItem_description__3ZsnJ"));
    WebElement bidButton = firstItemListView.findElement(By.className("ListItem_bidButton__3b21V"));

    assertTrue(description.isDisplayed());
    assertTrue(bidButton.isDisplayed());
}
```

3.4.8. Helper Methods

Below are helper methods used in this scenario, added in order to reduce code redundancy.

```
void selectMultipleFilters() throws InterruptedException {
    List<WebElement> categories = webDriver.findElements(By.className("Category_collapsibleCategory__3wERX"));
    categories.get(0).click();
    categories.get(1).click();

    WebElement subcategory = webDriver.findElement(By.className("Subcategory_subcategoryItem__1tROv"));
    WebElement subcategoryCheckbox = subcategory.findElement(By.tagName("input"));
    subcategoryCheckbox.click();

    JavascriptExecutor js = (JavascriptExecutor) webDriver;
    js.executeScript("window.scrollBy(0,300)", "");
    Thread.sleep(2000);

    selectPrices();
    js.executeScript("window.scrollBy(0,-300)", "");
    Thread.sleep(2000);
}

void selectPrices() {
    WebElement minPriceSlider = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div/div/div[1]/div[2]/span/span[3]"));
    Actions moveMinSlider = new Actions(webDriver);
    moveMinSlider
        .moveToElement(minPriceSlider)
        .clickAndHold(minPriceSlider)
        .moveByOffset(10, 0)
        .release().perform();

    WebElement maxPriceSlider = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div/div/div[1]/div[2]/span/span[4]"));
    Actions moveMaxSlider = new Actions(webDriver);
    moveMaxSlider
        .moveToElement(maxPriceSlider)
        .clickAndHold(maxPriceSlider)
        .moveByOffset(-150, 0)
        .release().perform();
}

void openShopPage() throws InterruptedException {
    WebElement shopLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/a[2]"));
    shopLink.click();
    Thread.sleep(2000);
}
```

3.5. Item Page Interaction

Guest user, while on the item page, can view information about the product, but cannot bid.

Logged-in user can take part in the bidding process, which has validation.

3.5.1. Test Item Header and Title

Test Name: Test item header and title				
Description: Check if the item name in header and title match.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click shop page. 2. Iterate shop elements (items). 3. Click item. 4. Check if item name is the same in the title and header.	**Test Data:**	**Expected Result:** Item name is the same.	**Actual Result:** Item name is the same.	**Status:** PASS
Notes:				

```
@Test
void testItemHeaderAndTitle() throws InterruptedException {
    openShopPage();
    List<WebElement> itemList = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));

    for (int i = 1; i < itemList.size() + 1; i++) {
        //Locate xpath of next element in iteration
        String xpath = String.format("/html/body/div/div/div[3]/div/div/div[2]/div[3]/div[%d]", i);
        webDriver.findElement(By.xpath(xpath)).click();
        Thread.sleep( millis: 1000);

        WebElement headerName = webDriver.findElement(By.xpath("/html/body/div[1]/div/div[3]/div[1]/p"));
        WebElement itemName = webDriver.findElement(By.xpath("/html/body/div[1]/div/div[3]/div[2]/div/div/div[2]/h3"));
        assertEquals(headerName.getText(), itemName.getText());
        webDriver.navigate().back();
        Thread.sleep( millis: 1000);
    }
}
```

3.5.2. Test Breadcrumbs

Test Name: Test Breadcrumbs				
Description: Check if the URL of the previous page is correct.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click home page item. 2. Check that previous page was home. 3. Click shop page. 4. Click shop page item. 5. Check that previous page was shop.	**Test Data:**	**Expected Result:** Previous pages are displayed correctly.	**Actual Result:** Previous page of shop item is incorrect.	**Status:** FAIL
Notes:				

```java
@Test
void testBreadcrumbs() throws InterruptedException {
    Thread.sleep( millis: 2000);
    WebElement homePageItem = webDriver.findElement(By.className("Item_itemContainer__uYG6J"));
    homePageItem.click();
    Thread.sleep( millis: 2000);

    WebElement previousPageInBreadcrumbs = webDriver.findElement(By.className("ItemOverview_breadcrumbsLink__1bkCB"));
    assertEquals( expected: "Home", previousPageInBreadcrumbs.getText());

    openShopPage();

    WebElement shopPageItem = webDriver.findElement(By.className("Item_itemContainer__uYG6J"));
    shopPageItem.click();
    Thread.sleep( millis: 2000);

    previousPageInBreadcrumbs = webDriver.findElement(By.className("ItemOverview_breadcrumbsLink__1bkCB"));
    assertEquals( expected: "Shop", previousPageInBreadcrumbs.getText());
}
```

3.5.3. Test Ability to Place Bid

Test Name: Test ability to place bid				
Description: Check if a bid can be placed on any item. Only a logged in user can bid.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click shop page. 2. Iterate shop elements (items). 3. Click item. 4. Check if bid container is hidden. 5. Login into existing account. 6. Check if bid container is displayed.	**Test Data:**	**Expected Result:** Bid container is hidden until the user logs in.	**Actual Result:** Bid container is hidden until the user logs in.	**Status:** PASS
Notes:				

```java
@Test
void testIfUserCanPlaceBid() throws InterruptedException{
    openShopPage();
    List<WebElement> itemList = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));

    for (int i = 1; i < itemList.size() + 1; i++) {
        String xpath = String.format("/html/body/div/div/div[3]/div/div/div[2]/div[3]/div[%d]", i);
        webDriver.findElement(By.xpath(xpath)).click();
        Thread.sleep( millis: 1000);
        assertThrows(NoSuchElementException.class,
                () -> {webDriver.findElement(By.xpath("/html/body/div/div/div[3]/div[2]/div/div[1]/div[2]/div[2]/div[2]/button"));});
        webDriver.navigate().back();
        Thread.sleep( millis: 1000);
    }

    doLogin();
    openShopPage();

    for(int i = 1; i < itemList.size() + 1; i++) {
        String xpath = String.format("/html/body/div/div/div[3]/div/div/div[2]/div[3]/div[%d]", i);
        webDriver.findElement(By.xpath(xpath)).click();
        Thread.sleep( millis: 1000);
        WebElement bidButton = webDriver.findElement(By.xpath("/html/body/div/div/div[3]/div[2]/div/div[1]/div[2]/div[2]/div[2]/button"));
        assertEquals( expected: "PLACE BID", bidButton.getText());
        webDriver.navigate().back();
        Thread.sleep( millis: 1000);
    }
    doLogout();
}
```

3.5.4. Test Bid Placeholder

Test Name: Test bid placeholder				
Description: Check if the appropriate bid value is displayed in the placeholder.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Click shop page. 2. Iterate shop elements (items). 3. Click item. 4. Check if placeholder bid value is adequate.	**Test Data:**	**Expected Result:** Starting bid is correct.	**Actual Result:** Starting bid is correct.	**Status:** PASS
Notes: Value displayed in the placeholder should be 0.1 higher than the starting price if there are no bids, and otherwise 0.1 higher than the highest bid.				

```java
@Test
void testBidPlaceholder() throws InterruptedException{
    doLogin();
    openShopPage();
    List<WebElement> itemList = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));

    for(int i = 1; i < itemList.size() + 1; i++) {
        String xpath = String.format("/html/body/div/div/div[3]/div/div/div[2]/div[3]/div[%d]", i);
        webDriver.findElement(By.xpath(xpath)).click();
        Thread.sleep( millis: 1000);

        Double startBid = Double.parseDouble(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div[1]/div[2]/p/span")).getText().substring(1));
        Double highestBid = Double.parseDouble(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div[1]/div[2]/div[1]/p[1]/span")).getText().substring(1));
        Double placeholderBid = Double.parseDouble(webDriver.findElement(By.cssSelector("input.ItemOverview_bidInput__3ic45")).getAttribute( name: "placeholder").split( regex: " ")[1].substring(1));
        Boolean verifyStartBid;

        if (placeholderBid > startBid && placeholderBid > highestBid) {
            verifyStartBid = true;
        } else {
            verifyStartBid = false;
        }

        assertTrue(verifyStartBid);
        webDriver.navigate().back();
        Thread.sleep( millis: 1000);
    }
    doLogout();
}
```

3.5.5. Test Bidding Process

Test Name: Test bidding process				
Description: Check if a bid can be made. Bid must be higher than the highest and starting bid. After placing a bid, the number of bids and the highest bid should update.				
Pre-condition(s): Before starting with each scenario, the server is started. Every test starts from the home page.				
Test Steps: 1. Log in 2. Click shop page. 3. Iterate shop elements (items). 4. Click item. 5. Send invalid bid. 6. Click bid button. 7. Check for bid fail alert. 8. Check that number of bids did not update. 9. Send valid bid. 10. Click bid button. 11. Check if bid is submitted. 12. Check that number of bids and highest bid are updated.	**Test Data:**	**Expected Result:** After entering a valid bid, it is submitted and the number of bids and the highest bid are updated.	**Actual Result:** After entering a valid bid, it is submitted and the number of bids and the highest bid are updated.	**Status:** PASS
Notes:				

```java
@Test
void testBiddingProcess() throws InterruptedException{
    doLogin();
    openShopPage();
    List<WebElement> itemList = webDriver.findElements(By.className("Item_itemContainer__uYG6J"));

    for(int i = 1; i < 2; i++) {
        String xpath = String.format("/html/body/div/div/div[3]/div/div/div[2]/div[3]/div[%d]", i);
        webDriver.findElement(By.xpath(xpath)).click();
        Thread.sleep( millis: 3000);

        int noOfBids = Integer.parseInt(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div/div[2]/div[1]/p[2]/span")).getText());


        Double startBid = Double.parseDouble(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div[1]/div[2]/p/span")).getText().substring(1));
        Double highestBid = Double.parseDouble(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div[1]/div[2]/div[1]/p[1]/span")).getText().substring(1));
        Double valueToUse = getBiggerDouble(startBid, highestBid);
        Double lesserBid = valueToUse - 0.1;
        Double higherBid = valueToUse + 0.1;

        WebElement bidPlaceholder = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/div/div[1]/div[2]/div[2]/div[1]/input"));
        WebElement bidButton = webDriver.findElement(By.xpath("/html/body/div/div/div[3]/div[2]/div/div[1]/div[2]/div[2]/div[2]/button"));

        bidPlaceholder.sendKeys(String.valueOf(lesserBid));
        bidButton.click();
        Thread.sleep( millis: 3000);

        WebElement failAlert = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]"));
        assertEquals( expected: "There are higher bids than yours. You could give a second try!", failAlert.getText());
        int noOfBidsAfterFail = Integer.parseInt(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[3]/div/div[1]/div[2]/div[1]/p[2]/span")).getText());
        assertEquals(noOfBids, noOfBidsAfterFail);
        Thread.sleep( millis: 3000);

        bidPlaceholder.clear();
        bidPlaceholder.sendKeys(String.valueOf(higherBid));
        bidButton.click();
        Thread.sleep( millis: 3000);

        WebElement alertSuccess = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]"));
        assertEquals( expected: "Congrats! You are the highest bidder!", alertSuccess.getText());

        highestBid = Double.parseDouble(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[3]/div/div[1]/div[2]/div[1]/p[1]/span")).getText().substring(1));
        int noOfBidsAfterSuccess = Integer.parseInt(webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[3]/div/div[1]/div[2]/div[1]/p[2]/span")).getText());
        assertEquals(highestBid, higherBid);
        assertEquals( expected: noOfBids + 1, noOfBidsAfterSuccess);


        webDriver.navigate().back();
        Thread.sleep( millis: 1000);
    }
    doLogout();
}
```

3.5.6. Helper Methods

Below are helper methods used in this scenario, added in order to reduce code redundancy.

```
void openShopPage() throws InterruptedException {
    WebElement shopLink = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div/div[3]/a[2]"));
    shopLink.click();
    Thread.sleep( millis: 2000);
}

Double getBiggerDouble(Double a, Double b) {
    if(a > b) {
        return a;
    } else {
        return b;
    }
}

void doLogin() throws InterruptedException {
    WebElement login = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    login.click();
    Thread.sleep( millis: 2000);
    WebElement email = webDriver.findElement(By.name("email"));
    WebElement password = webDriver.findElement(By.name("password"));
    email.sendKeys( ...keysToSend: "matej.mujezinovic@gmail.com");
    password.sendKeys( ...keysToSend: "123456789");
    WebElement loginButton = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[3]/div[2]/form/button"));
    loginButton.click();
    Thread.sleep( millis: 2000);
}

void doLogout() throws InterruptedException {
    WebElement logout = webDriver.findElement(By.xpath("//*[@id=\"root\"]/div/div[1]/nav/p/a[1]"));
    logout.click();
    Thread.sleep( millis: 2000);
}
```

4. Conclusion

4.1. Testing Summary

Testing Tool	Total Tests	Passed Tests	Failed Tests
jUnit/Selenium	26	25	1

Failed Test: 3.5.2. Test Breadcrumbs

4.2. Final Thoughts

We were very happy to have the opportunity to test a website that one of us created. We covered every important functionality of the web shop giving it a decent amount of code coverage and quality assurance. A great majority of the tests have passed, indicating that the web application does operate in a very functional and correct way. However, even for the tests that have passed, there are some elements which could be improved, in terms of better UI experience, e.g. the message after an unauthorized login could be “Wrong email or password” instead of “Request failed with status code

401”. The failure of the breadcrumbs test was a nice discovery and unveiled a bug that the developer was unaware of before, displaying the importance of testing during the development process.