# Testing Evaluations Of an E-Commerce Website

## Manual Testing:

Analyzing the outlook of the website we can look into developing some Black Box testing techniques as the documentation is not provided. Beginning with various simple unit tests to verify and/or validate single functionalities one at a time to start and further develop methods to determine optimal functionality.
It is also necessary to determine how different components are behaving in respect to each other so it is essential to implement integration testing with the entire structure in mind.

### Determining Various Test cases for Manual testing:

General Cases:

- ➢ Verify the compatibility of the website with it being able to load on different browser
- ➢ Validate the responsiveness across different platforms and devices(mobiles, PC, Tablets)
- ➢ Look for clear and complete Nav bar

Homepage Tests:

- ➢ Check for the hover effect on relevant elements
- ➢ Check the functionality of various Themes of the website(light,dark,default)
- ➢ Check searching functionality for searching various products and displaying thier information

Search Query Tests:

- ➢ Check for related object searching
- ➢ See if search produces accurate results(implemented example in Automated testing)
- ➢ Verify functionality of filters on basis on product/prices

Payments Test Cases:

- ➢ Test the different payment methods available on the website, such as credit cards,PayPal, or any other payment gateway.
- ➢ Validate that payment transactions are processed securely and without any errors.

- ➢ Verify that order confirmation emails are sent to customers after successful payment.

Shopping Cart Test Cases:

- ➢ Verify that products are added and removed from the cart correctly.
- ➢ Test the quantity update feature in the cart.
- ➢ Validate the subtotal, taxes, and shipping calculations on the cart page.

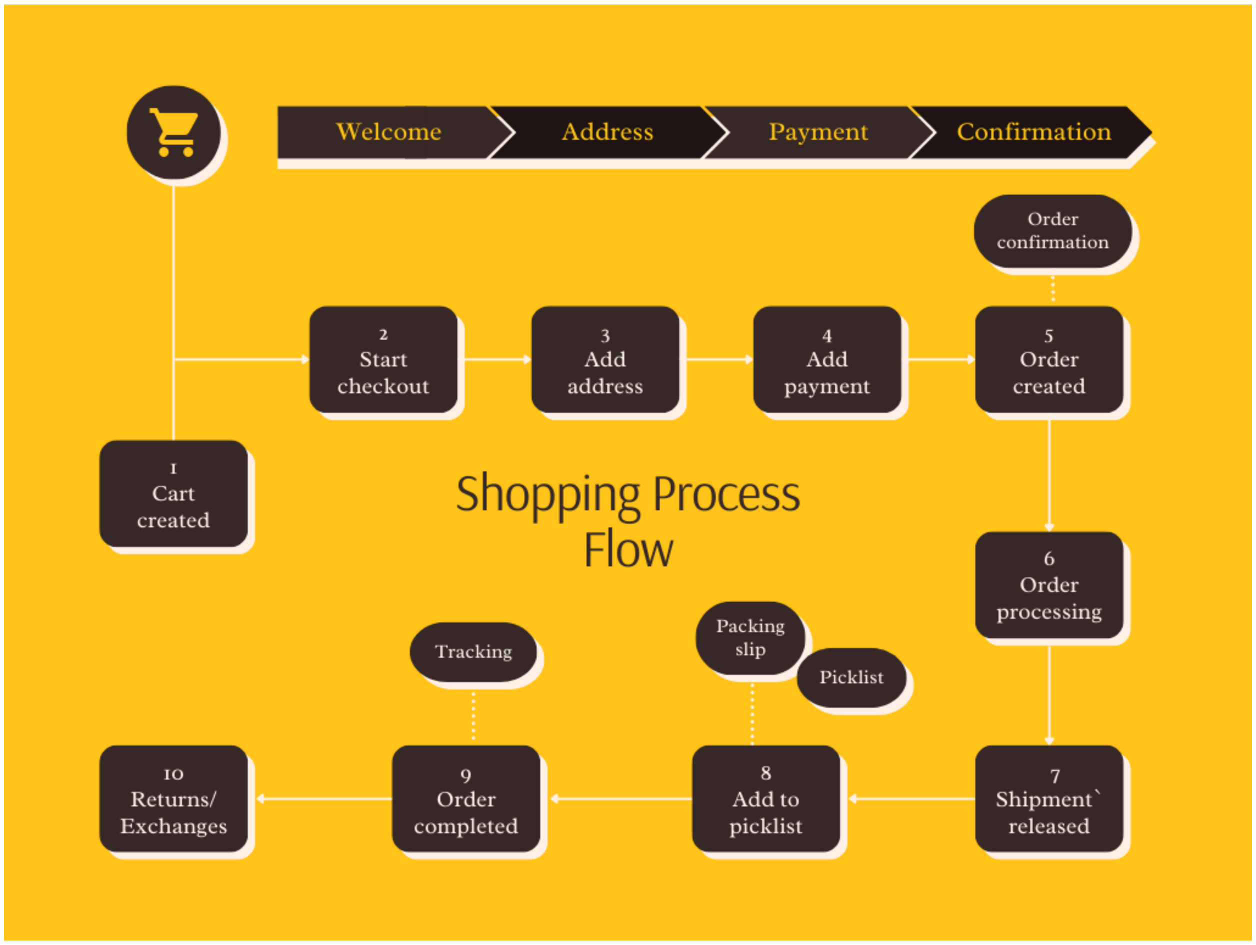
Registration Page Test Cases:

- ➢ Test the user registration process, including form validation and error handling.
- ➢ Validate that user accounts are created successfully and can be logged in to.
- ➢ Verify that the account confirmation email is sent after registration.
- ➢ Verify non-existent email entered is detected and invalid fo signing up

## Example Test Plan:

We can implement an example test plan to efficiently test the entirety of the test cases involving various stages and phases.This can help us find errors by detecting them in what phases they occur in and help debug and counter them.

Test Planning and Design is the initial phase of manual testing, where the scope and objectives of the testing efforts are clearly defined. This involves understanding the requirements and specifications of the application to develop comprehensive test cases and scenarios. These test cases detail the steps to be followed, the expected results, and any specific conditions to be tested. Additionally, this phase includes preparing the necessary test data and setting up the test environment, ensuring that all conditions mirror the production environment as closely as possible.



## <u>Phase 1: Welcome</u>

<u>Home Page and Landing Site Test Coverage:</u>

➢ Ensure the responsiveness of the home page on various devices (mobile, tablet, desktop).
➢ Test for interface issues to ensure a smooth user experience from the start.
➢ Validate the correctness of login and sign-up functionalities.
➢ Test for correct Landing page transition from product windows.

<u>Cart Initialization:</u>

➢ Proceed by adding items to the cart for further testing.
➢ Enable products ot be added to cart and disable unavailable products.

## **Phase 2: Address**

Adding Items to Cart:

- ➢ Ensure logical functionalities of the cart operate correctly.
- ➢ Empty Cart featuring Null cannot proceed to next stages.

Address Input:

- ➢ Validate that a valid address can be added.
- ➢ Ensure the correct handling of country selection.
- ➢ Verify the accuracy of Pincodes and additional contact information.

## **Phase 3: Payment**

Payment Gateway Testing:

- ➢ Test various payment gateways for secure and reliable transactions.
- ➢ Ensure that payments are atomic and isolated within the database.
- ➢ Verify that transactions do not result in incorrect overcharging or undercharging.

Price Handling:

- ➢ Confirm the correct calculation and display of prices, delivery costs, and taxes.
- ➢ Responsive calculation in the event of removal or addition of products
- ➢ Handling discount and offers.

## **Phase 4: Confirmation**

Order Verification:

- ➢ Send a verification message to the user upon order confirmation.
- ➢ Ensure correct attributes are correlated to their respective values and features.

Order Cancellation and Return:

- ➢ Validate the consistency of the database in handling order cancellations and returns.
- ➢ Keep accurate logs for tracking order history and changes.

These Steps are an efficient way to analyse and access different types of tests which can cover most aspects and potential faults of the system. Implementing these can give us an overall understanding of the system in entirety and help deter most possibilities of defects. These Phase and stages cover various different system testing features which increases the overall functional quality of the website.

## **Results:**

The results of manual testing provide a comprehensive overview of the application's functionality, usability, and reliability. During the testing process, both positive outcomes and defects are meticulously recorded, offering valuable insights into the software's quality

Positive:

- Website highly responsive towards shifting of landing pages.
- Cart logical functionalities are well operated.
- Search Query provides accurate results.
- Registration and verification gateways are fully functional.

Defects:

- While signing up incorrect emails are not detected. This can lead to sending verification codes to email addresses that do not exist.
- The price range filtering does not accurately represent the range of products from low to high.
- Language changing button does to change language of the page

## Automated Testing:

Automated testing involves using specialized software tools to execute pre-scripted tests on a software application before it is released into production. This method of testing is essential for increasing the efficiency, effectiveness, and coverage of the testing process, allowing for quicker detection of defects and continuous integration.

Stack Used: Python , Selenium, ChromeDriver.

## Test Case 1:

To verify that the correct landing page is opened by checking the page title of a specific URL. It ensures that the page title matches the expected value.

## Description

- ➢ Using Selenium to automate the process of opening a webpage and verifying that the title matches the expected value.
- ➢ It also manages handling exceptions and ensuring the browser closes after the test, whether it passes or fails.
- ➢ Scraping the HTML of the page and checking the <title> tag of the Landing Page is same as our target.
- ➢ If the <title> tag is same our test is successful therefore the desired page has loaded.
- ➢ This test is essential for making sure that while development the pages do not get mixed up and is tested upon thier <title> tags.

## Implementation:

Initialize webdriver: Create a new instance of Chrome driver to open the source Page

```
driver = webdriver.Chrome(service=service, options=chrome_options)
driver.get("https://vercel-commerce.oramasearch.com/")
```

Verify Page Title: Wait until the page title is loaded or time out after 10 seconds.

```
try:
    WebDriverWait(driver, 10).until(
        EC.title_is("Next.js Commerce")
    )
```

Get and verify this page:

```
 # Verify title
    assert page_title == "Next.js Commerce", f"Expected title to be
'Next.js Commerce' but got '{page_title}'"
    print("Test Passed: Page title is correct.")
except Exception as e:
    print(f"Test Failed: {e}")
```

## Test Results:

The script above provides a positive result to the Test case of verifying the desired Landing page which is determined by checking the <title> tag of the page from the HTML of the web page. Hence we can assure that the webpage `"https://vercel-commerce.oramasearch.com/"` delivers the correct and valid Landing Page.

## Test Case 2:

Test to check the Price sorting Filter and see if values are in correct order from low to high price ranges or if a valid price is not available show error

Description:

- ➢ Find and identify the price elements of the product on the page
- ➢ Check if thetre are atleast two prices of the products of compare from
- ➢ If lesser than two not enough elements to compare from
- ➢ Remove the $ sign from the price and convert to float type variable
- ➢ Now compare the price elements to ensure they are in the desired order
- ➢ If the price comparison doesnot match the filter as given return error

➢ In the case of the price elements not being able to be compared or matched throw error.

Implementation:
Initialize Webdriver: Instantiate the Chrome webdriver and open the target URL

```
driver = webdriver.Chrome(service=service, options=chrome_options)
driver.get("https://vercel-commerce.oramasearch.com/search/shirts?sort=pri
ce-asc")
```

Locate and verify Price Elements: Wait for price elements to load on the page.

```
try:
    price_elements = WebDriverWait(driver, 10).until(
        EC.presence_of_all_elements_located((By.CSS_SELECTOR,
"p.flex-none.rounded-full.bg-blue-600.p-2.text-white"))
```

Check if there are enough price elements: There must be atleast two elements to compare

```
 # if less than 2 prices, cannot compare
    if len(price_elements) < 2:
        print("Pricing error: Not enough price elements found.")
```

Convert Price elements to float: Remove the $ sign from the price text and convert to float values to compare.

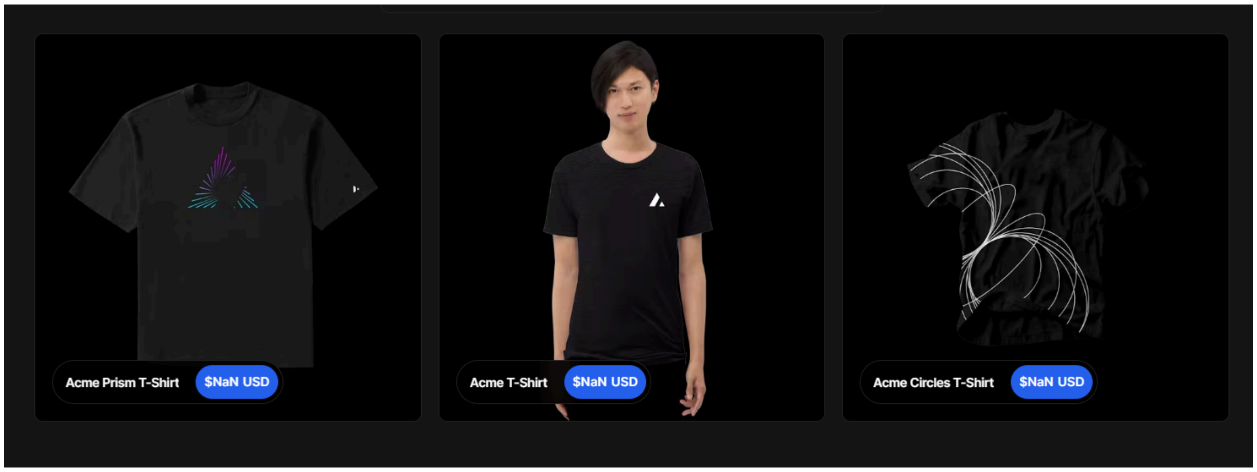
```
else:
        #Conv price elements to float
        price_1 = price_elements[0].text.strip("$")
        price_2 = price_elements[1].text.strip("$")

        try:
            price_1_value = float(price_1)
            price_2_value = float(price_2)

            # Compare the prices
            if price_1_value < price_2_value:
                print("Price filtering is correct.")
            else:
                print("Incorrect price filtering.")
        except ValueError:
            print("Pricing error: Unable to determine the price values.")
```

## Test Results:

The script checks the the price filtering functionality of the website and compares if the the price elements are in the correct order from low to high.There have to be atleast 2 price elements to be able to compare from. To check the price elements we also have to search through the HTML and remove the $ sign from the  by location via CSS_selector. Also checks the price element for being a valid comparable element so that the Price filter can be valid. However the resulting script showed an Error to unable to determine the prices as the webpage displayed invalid incomparable elements.



## Test Case 3:

Test for implementing a seach bar analysis by searcing an element and verifying results
Description:

- ➢ Open a webpage using the Chrome Webdriver
- ➢ Navigate to find the search bar using the subsequent tas found it the page's HTML script
- ➢ Enter a base seach item for running a test
- ➢ Submit the search and find the products related to the search query
- ➢ Wait and verify if the products are of the related query or incorrect

Implementation:

Initialize the Chrome webdriver:

```
driver = webdriver.Chrome(service=service, options=chrome_options)
driver.get("https://vercel-commerce.oramasearch.com/search/shirts?sort=pri
ce-asc")
```

Locate and Identify the Search Input Element:

```
Try:
    search_input = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR,
"input[name='search']"))
    )
```

Enter the Seach Query for the base element:

```
    search_term = "jacket" # Enter the search element
    search_input.send_keys(search_term)
    # Submit
    search_input.submit()
```

Wait for search results to load:

```
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR, "div.grid"))
    print(f"Search for '{search_term}' works and results are displayed.")
```

Check and Validate the Search Query:

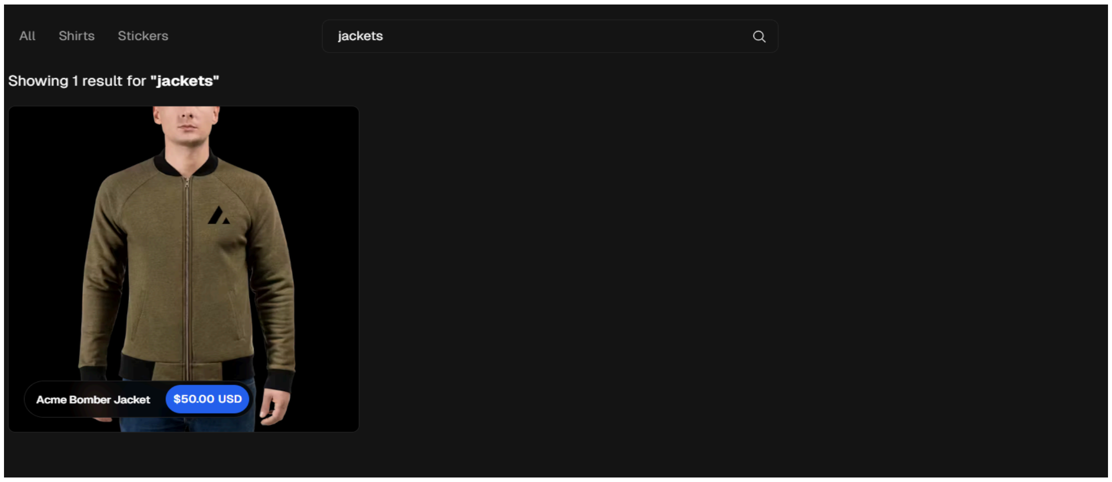
```
 results = driver.find_elements(By.CSS_SELECTOR, "div.grid div")
    assert len(results) > 0, "No search results found!"
    print(f"Test Passed: Search results for '{search_term}' are displayed
correctly.")
```

Error for when Test fails:

```
except Exception as e:
    print(f"Test Failed: {e}")
```

## Test Results:

This tests checks the search bar functionality to search and look for valid queries and of the correct product is produced in the output.The results of this test case shows us the positive outcome and valid product searches which shows the correct functionality of the the search bar.



Showing 1 result for "jackets"

## Test Case 4:

Test to check if clicking on product leads to correct page with the product info

Description:

- ➢ The script opens the web page and checks for a specific product to be available as a window which on click opens to the page of the product's information
- ➢ It finds and identifies the HTML tag related to the class of the product
- ➢ It clicks the product link related which leads to the page containing the information about the product
- ➢ Checks for the specific meta tag to confirm the correct page has loaded
- ➢ Prints success or errors messages based on the outcome of the execution

Implementation:

Initialize Chrome Webdriver:

```
driver = webdriver.Chrome(service=service, options=chrome_options)
driver.get("https://vercel-commerce.oramasearch.com/search/shirts?sort=pri
ce-asc")
```

Load the Product Link: Wait until the page is loaded then locate the product link

```
try:

    shirt_anchor = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR,
"a[href='/product/acme-geometric-circles-t-shirt']"))
    )
```

Click on the Link and verify the product page:

```
shirt_anchor.click()
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR,
"meta[property='og:title'][content='Acme Circles T-Shirt | Acme Store']"))
    )
    print("The anchor tag leads to the correct landing page.")
```

Handle Exceptions:

```
except Exception as e:
    print(f"Test Failed: {e}")
```

## Test Results:

This test is used to chrck if the products appearing on the landing site as clickable product windows are redirecting to the correct page which displays the information of the product.The result of this test indicates the positive behaviour that the website behaves responsively and displays the correct product of T-shirt.

## Test Case 5:

Test to check the logical functionality of the cart on the web page ensuring that the cart shows an empty message when there are no items and that the remove button is not present or clickable when the cart is empty.

Description:

- ➢ The script opens the webpage and waits for the cart icon to be clickable
- ➢ Once clicked it waits for the empty cart message to be displayed.
- ➢ Verify the empty cart message text to confirm the cart is empty
- ➢ Checks that the remove button is not present or clickable, as you cannot remove items when the cart is empty
- ➢ Prints success or error message based on the execution

Implementation:

Initialize the Chrome Webdriver:

```
driver = webdriver.Chrome(service=service, options=chrome_options)
driver.get("https://vercel-commerce.oramasearch.com/search/shirts?sort=pri
ce-asc")
```

Load and click the cart icon:

```
try:
    cart_icon = WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.CSS_SELECTOR,
"button[aria-label='Open cart']"))
    )
    cart_icon.click()
```

Verify that the cart is empty:

```
empty_cart_message = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR,
"p.mt-6.text-center.text-2xl.font-bold"))
    )

    # Verify the cart is empty
    assert "Your cart is empty." in empty_cart_message.text, "Cart is not
empty!"
```

Check for Remove button:

```
  # Check if the remove button is present or clickable
    remove_button_present = False
    try:
        remove_button = driver.find_element(By.CSS_SELECTOR,
"button[aria-label='Reduce item quantity']")
        if remove_button:
            remove_button_present = True
    except:
        remove_button_present = False

    #Check remove button not present
    assert not remove_button_present, "Remove button should not be present
when the cart is empty!"
```

Print the Result:

```
    print("Test Passed: Cannot remove an item when the cart is empty.")
except Exception as e:
    print(f"Test Failed: {e}")
```

## Test Result:

This test results indicates the logical functionality of the webpage which is that once an item is loaded into the cart it can be removed however the fundamental test scenario arises when the cart is empty and the item is being attempted to being removed from the cart.Since the cart is empty and no item can be removed it corresponds to the error and prints successfully if the case establishes that the cart disables itme removing if the cart is empty.