**CAPSTONE PROJECT**

**Phase – I**

**Overview**

This test automation project was designed for the Royal Brothers website, a bike rental platform. The goal was to create a robust, scalable, and maintainable test suite using Playwright with TypeScript, focusing on end-to-end functionality, UI validation, and regression testing.

**WEB SITE**: https://www.royalbrothers.com/

**Process Followed**

1. Playwright Configuration

Make sure you have the following installed:

* Node.js v16 or later
* npm comes with Node
* TypeScript support
* Required browsers (Chromium, Firefox, WebKit)

1. Folder Structure

├── tests/ # Test files

├── pages/ # Page Object Models

├── fixtures/ # Custom fixtures and test hooks

├── utils/ # Helpers (e.g., data loaders)

├── playwright.config.ts # Global test config

└── package.json

1. Page Object Model (POM)

* Created separate files for each page (e.g., HomePage.ts, BookingPage.ts, LoginPage.ts).
* Each page class encapsulates selectors and actions.
* Promotes reusability and easier maintenance.

1. Test Scenarios

Automated key workflows:

* Home Page Load Test: Verified presence of city list, banners, and navigation.
* Search & Booking Flow:
  + Search for bikes by location and dates.
  + Apply filters (brand, engine capacity).
  + Verify availability and pricing.
* User can change multiple locations.
* Validate the Filter functionality and sort by.

1. Fixtures & Hooks

* Created custom fixtures to manage:
  + Browser context per test
  + Reusable login state
* Used beforeAll, beforeEach, and afterEach hooks for setup/cleanup.

1. Playwright Features

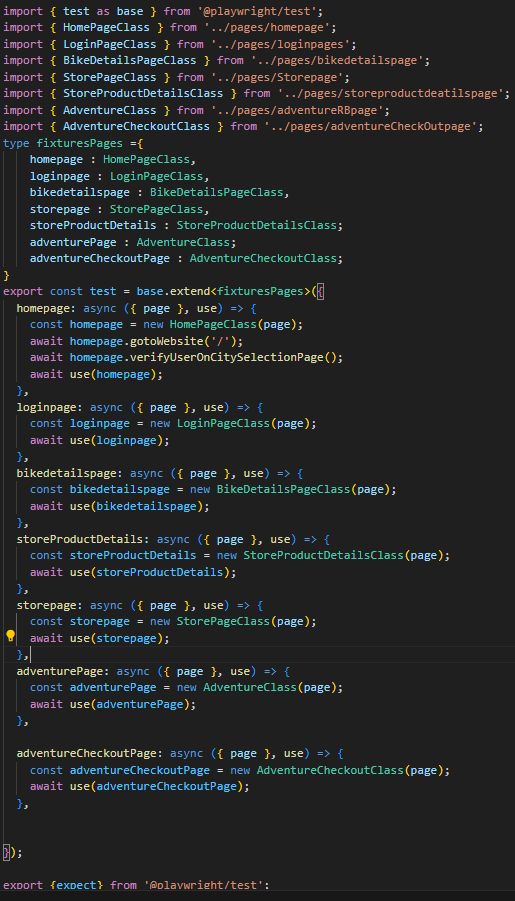
* Isolated browser contexts.
* Auto-waiting for elements.
* Cross-browser support Chromium, Firefox, WebKit.
* Screenshot and video capture.
* Custom test fixtures.
* Parallel execution and test sharding.
* Playwright Inspector for debugging.
* Powerful built-in assertions and locators.

1. Continues Integration
   * Integrated with GitHub Actions for automatic test runs.

* Reports are generated with HTML reporter.

**Fixture.ts**

This fixture setup extends Playwright’s base test by integrating custom page object classes for structured, reusable testing. Each fixture defines a specific page class (e.g., HomePageClass, LoginPageClass), enabling modular and readable test definitions. When a test begins, the homepage fixture navigates to the site and verifies the city selection page, ensuring the test always starts from a consistent state. Other fixtures like loginpage, bikedetailspage, storepage, and adventurePage instantiate their respective page objects, making their methods available in tests. These are injected into test contexts, so they can be accessed directly within each test function. This structure promotes maintainability, reduces duplication, and encapsulates page-specific actions and assertions. It also improves readability by cleanly separating setup logic from test logic. All fixtures are declared as asynchronous, allowing Playwright to await necessary setup steps. Overall, this fixture system streamlines end-to-end testing across multiple site modules like Bike Details, Store, and Adventure Checkout.



**Pages Folder:**

**HomePageClass**

Handles actions on the landing and home screen — city selection, login/signup, navigation menus, chatbot, and other UI elements. It ensures that users land on the correct homepage and can navigate to key sections.

**LoginPageClass**

Manages login-related functionalities — entering credentials, submitting, and validating login success/failure.

**BikeDetailsPageClass**

Handles interaction with individual bike detail views — used to validate specific models, pricing, availability, and booking flows.

**StorePageClass**

Represents the e-commerce “Store by RB” — manages product filtering (e.g., Hoodies, T-Shirts), sorting, navigation, and UI checks like the “View All” button.

**StoreProductDetailsClass**

Works with individual store product pages — handles add-to-cart, product info verification, and transition to the cart.

**AdventureClass**

Represents the Adventures listing page — handles searching, accessing specific adventure activities (e.g., Motorcycle Training), and navigation to detail views.

**AdventureCheckoutClass**

Manages the checkout process for adventure bookings — entering user info, applying discount codes, verifying pricing, quantity, and checkbox behaviors.

**Test Homepage:**

**Verify title of the page**

This test case verifies that the user is able to access the homepage of the application. It ensures that the website loads correctly and the base URL is functional. The method verifyUserNavigateToWebsite() checks the visibility or correctness of UI elements. It serves as a smoke test to confirm the app is up. This is foundational for further testing. If this fails, all subsequent navigation tests could be compromised. It ensures initial access to the application is reliable.

**Validate the user can search for a city**

This test checks whether a user can search and select a city like "Calicut" from the UI. It ensures that after the selection, the application navigates properly to the city-specific homepage. The city selection must be reflected in the UI state or content. This ensures dynamic content rendering based on city is working. It may be tied to geo-based service delivery. It's essential for location-based filtering of results. The test confirms functional integration of city search with homepage routing.

**User can change location to city**

These tests iterate over a list of city names (Cochin, Coorg, Tirupati) and validate the city switch functionality. For each city, the app should navigate to a new context or homepage. It tests dynamic location updates without reloading the app. Ensures the new city reflects in both UI and URL. This simulates real user behavior for planning trips in different locations. It also confirms if each city’s page is correctly linked. Smooth transitions between cities improve UX and SEO.

**Verify the user can navigate to the login page**

This test confirms the login button is functional and visible on the homepage. When clicked, it should redirect the user to the login page. The login page’s unique identifier (like URL or header) is verified. It validates proper routing and page load of a critical entry point. This is a gateway to authentication workflows. If this fails, user sign-in or sign-up cannot proceed. Ensuring reliability of login navigation is vital for user acquisition.

**Validate user can select the date and time**

This test validates that the user can interact with date and time pickers. After choosing values, the user can click the search button. The app then routes to the bike details page. This ensures the booking workflow initiates correctly. It checks proper state handling between pages. The test simulates a real booking interaction. It verifies the UI components for date-time selection are functional and well-integrated.

**User can navigate to hamburger menu and close**

This test ensures that the hamburger menu can be opened and closed properly. It mimics mobile or responsive interface behavior. When opened, the menu options should appear correctly. The test validates UI visibility and toggle behavior. Closing the menu should restore the original layout. This improves usability on smaller screens. It confirms the responsive design elements are working as intended.

**Verify if chatbot is responding to user in web**

This test checks the chatbot feature of the website. It ensures the chatbot icon is clickable and the chat window opens. After typing “hi,” the chatbot should generate a meaningful response. This validates both UI and backend chat integration. It’s essential for user engagement and support. Chatbot responsiveness reflects server connectivity and AI flow. The test confirms user interactions are recognized and handled appropriately.

**Verify user can navigate to offers page**

This test validates that users can click on the "Offers" button and land on the correct page. It ensures the link routes to a city-specific offers section. The page title is checked for correctness. This verifies metadata and page identity. It confirms marketing content is accessible. Ensures routing and linking logic is correctly implemented. This improves visibility of promotional content to users.

**Validate What's New functionality**

This test evaluates hover interaction over the "What’s New" button. It ensures a dropdown is rendered when hovered. The option “Adventures by RB” is selected and clicked. It should route to an external eCommerce site. Title verification ensures successful navigation. The test verifies menu responsiveness and external linking. It validates UI behavior along with outbound redirection

**Validate Sort By RB correctly switching to the tab**

The test hovers over "What’s New" and clicks “Store by RB” from the dropdown. It ensures navigation to a specific product or shopping tab. Title verification confirms the page load. Ensures users can browse merchandise or biker gear. Validates dropdown link consistency and accuracy. This is important for product discovery and sales flow. Confirms seamless transition to Store section via UI interaction.

**Validate navigation to terms & conditions from login page**

This test verifies that clicking on Terms & Conditions from the login page opens a new browser tab. It waits for the new page and validates the title of that tab. Ensures external legal documentation is accessible. Confirms new tab handling in browser context. Critical for compliance and transparency. Tests browser multi-tab handling within the Playwright context. Ensures expected title text matches company legal page.

**Verify user enters an invalid phone number**

The test inputs an incorrect or malformed phone number in the login field. It checks if the validation logic properly flags it as invalid. Ensures frontend form validations are active. The result should show a warning or block progression. Prevents misuse or bad data from being submitted. Essential for accurate customer records. Confirms regex or validation rule enforcement on mobile number inputs.

**Verify user enters a valid phone number**

This test confirms the login page accepts a valid mobile number without validation errors. It uses a known correct number from the constants file. Ensures that the form does not throw an error. Confirms proper recognition of valid data formats. Prepares for OTP or next-step interaction. Confirms no false positives in validation logic. Essential for a smooth login experience.

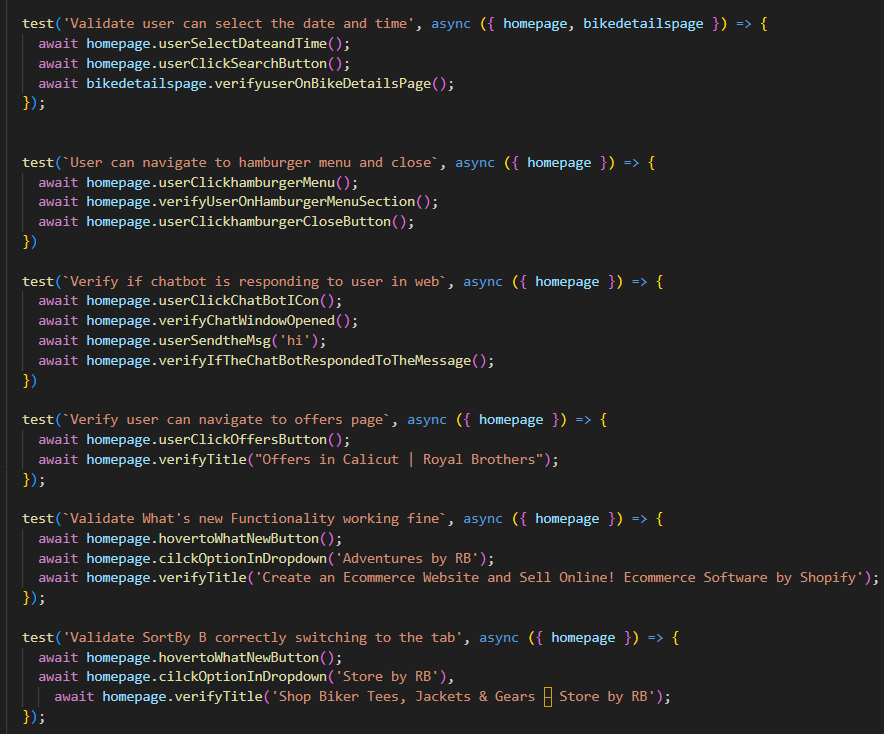
**User selects the country to enter phone number**

This test ensures that users can switch the country selection for the phone number field. Simulates users from different regions. Confirms dropdown renders correct country list. Validates the selection impacts input behavior (like country code). Supports international users. Important for localization and globalization. Ensures backend accepts international formats if needed.

**Validate phone number functionality with alerts**

This test interacts with potential alert messages tied to phone number input. It specifically tests dismissing or cancelling an alert. Ensures alerts are not blocking the user experience. Verifies that the alert cancel button is functional. Tests UI elements that might be dynamically rendered. Confirms user control over browser/system prompts. Adds robustness to form input flow handling.

****

****

****

**Bike Details Page Test Summary**

**Validate user successfully reach bike details page**

This test verifies that after the user selects a city and date, and initiates a search from the homepage, they are successfully redirected to the Bike Details page. It ensures that the navigation flow from home to detailed listings is functioning correctly.

**Validate user reach Blog page**

It checks whether the user can navigate from the Bike Details page to the Blog page via the "About Us" section. The test validates the redirection and confirms the page title to ensure the correct destination is reached.

**Verify user can't book sold out bikes**

This test confirms that bikes marked as "sold out" on the Bike Details page cannot be booked. It simulates the user interaction of selecting an unavailable bike and verifies that no booking action is permitted or triggered.

**Click "Book Now" redirected to booking page**

After selecting a bike and clicking the "Book Now" button, this test ensures the user is taken to the login or booking page. It verifies that the flow from selection to authentication is functioning as expected.

**Verify user can search bike using Filters**

This test checks that users can effectively search for bikes using model filters. It validates the filtering UI, the selection of a checkbox filter, and ensures that the resulting bikes displayed match the filtered model (e.g., "mountain").

**Validate user reach About Us page**

It verifies that clicking the "About Us" option from the Bike Details page redirects the user to the corresponding informational page. The test confirms navigation correctness by checking the page title.

**Validate user apply Filter and clear the Filter**

This test ensures users can apply model filters and also clear them successfully. It first applies a filter to view specific results, then clears it and confirms that the filter state is reset and all bikes are displayed again.

**Check Sign Up functionality**

The test validates that clicking the "Sign Up" button on the homepage redirects users to the login or registration page. It ensures that the sign-up flow is accessible and correctly routed.

**Verify user can navigate to the "tariff" page**

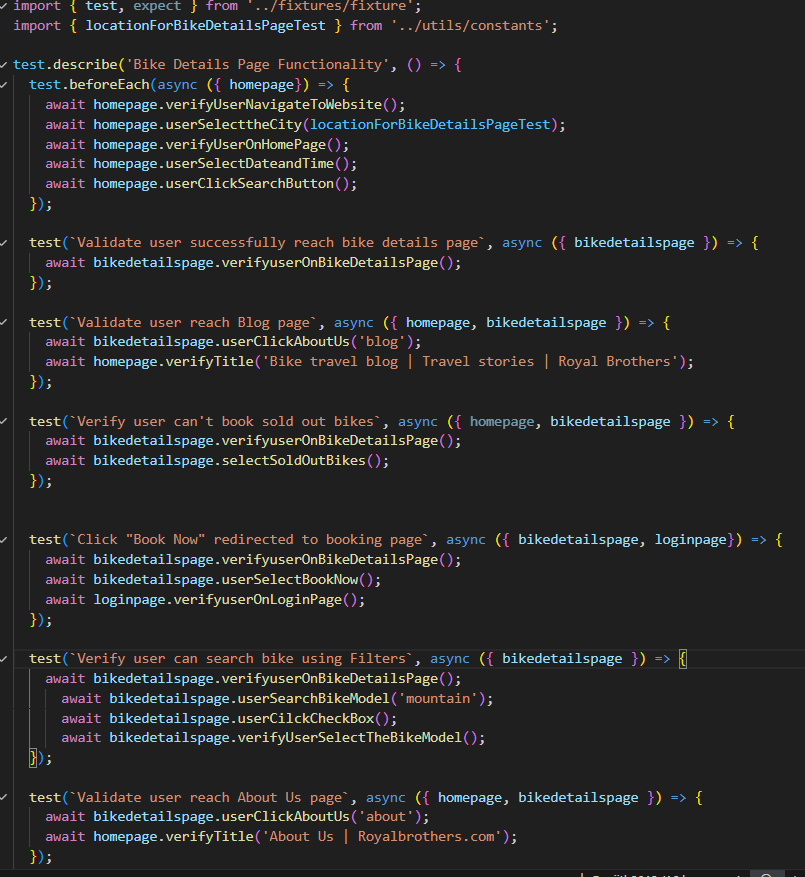
It confirms that users can navigate from the homepage to the "Tariff" page, where rental pricing and plans are listed. The test verifies that the correct page loads based on title checks.

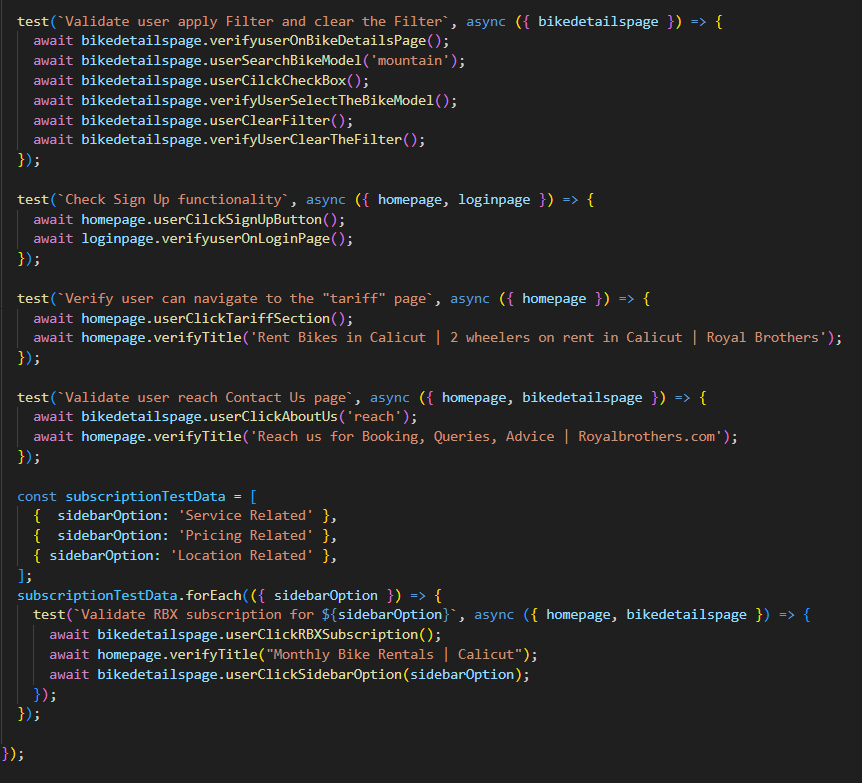
**Validate user reach Contact Us page**

This test checks that the "Contact Us" option from the "About Us" section redirects to the correct page. It ensures that users looking for support or contact details are guided to the right destination.

**Validate RBX subscription for various sidebar options**

For each sidebar category (Service Related, Pricing Related, Location Related), this test validates that users can access the RBX subscription page and navigate within its options. It ensures the sidebar functions correctly and the page title confirms the expected context.





**Test Adventure Page:**

**Validate user successfully reach adventure page**

This test ensures that users can navigate to the "Adventures by RB" page via the homepage dropdown. It confirms that the page loads correctly and that interactive slide options like "adventure" are functional.

**Chatbot is responding to user**

This test verifies that the chatbot on the Adventure page accepts user input such as name, gender, and email, and responds appropriately. It confirms the chatbot’s ability to handle basic user engagement.

**Check cart section navigation working fine**

It checks whether clicking the cart icon on the Adventure page correctly redirects the user to the cart section. This test ensures the shopping cart is accessible and visually presented as expected.

**User navigate to cart section and click "Shop Now" button**

After reaching the cart section, this test simulates a user clicking the "Shop Now" button. It verifies the user is redirected back to the Adventure product listing or homepage, maintaining user flow.

**Validate the search bar functionality**

This test checks that users can search the adventure store using keywords (e.g., "adventures") and are redirected to relevant search results. It confirms the integrity and usability of the search feature.

**Verify user can navigate to bike rental page**

The test ensures that from within the adventure page, selecting the slide option "royal" redirects the user to the bike rental or homepage. It validates cross-navigation between product types.

**Verify "Learn more" functionality**

This test confirms the "Learn more" button on the Adventure page leads to an informative page. It checks for correct redirection and successful loading of the detailed content section.

**Validate the booking functionality**

After searching and selecting a specific adventure card like "Motorcycle Training For Women", this test verifies the user can click "Buy It Now" and proceed to the checkout page. It ensures booking functionality is end-to-end.

**Validate cart page checkout functionality**

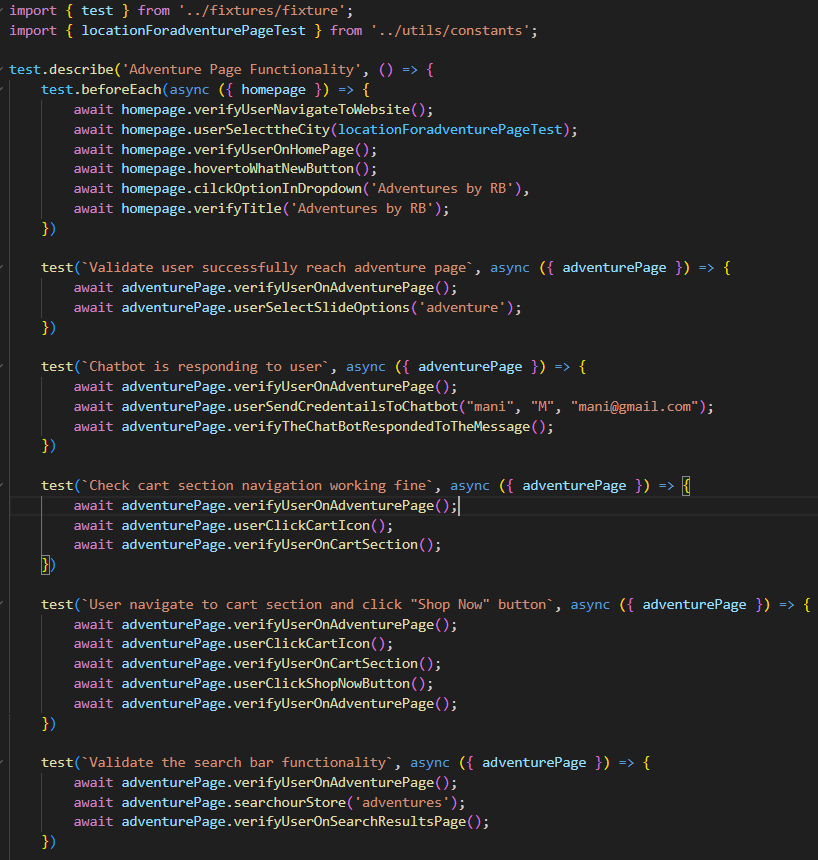
This test checks that users can add an adventure product to the cart and navigate to the cart section successfully. It confirms that add-to-cart and cart page routing works properly.

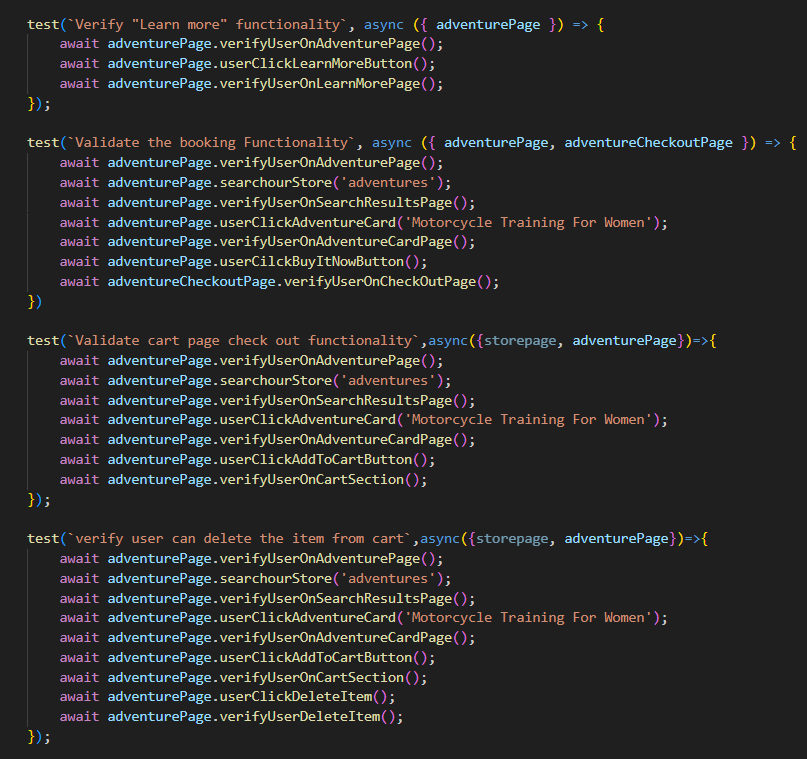
**Verify user can delete the item from cart**

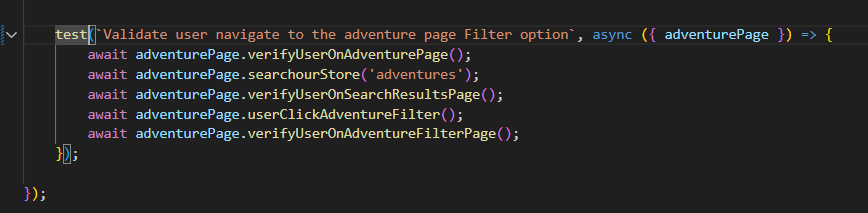
The test simulates adding a product to the cart and then deleting it. It confirms that the cart updates accordingly and the item is successfully removed from the cart view.

**Validate user navigate to the adventure page Filter option**

This test verifies the filtering functionality on the search results page. It checks that clicking the filter button redirects to the appropriate filter page, confirming advanced filtering is accessible.







**Test Store Page:**

**Validate user on Store Page**

This test verifies that the user successfully navigates to the Store page via the homepage dropdown. It confirms that the page loads correctly with the expected title. The test ensures the initial landing on the Store page is smooth and accurate.

**Validate user can navigate to each product category and verify title**

For categories like Hoodies, T-Shirts, Prints, and Essentials, this test ensures users can select each option from the Store page. It verifies the page updates accordingly and the title matches the selected category. This confirms proper category navigation and display.

**Validate sortBy option functionality 'Alphabetically, A-Z'**

This test checks that sorting products alphabetically from A to Z works correctly within a chosen category. It verifies that the product list updates as per the selected sort option. The feature’s accuracy and usability are confirmed.

**Validate add product to the cart**

This test confirms users can select a product, view its details, add it to the cart, and see confirmation of the addition. It ensures the add-to-cart flow is functioning without issues. The cart reflects the user’s selected items properly.

**Verify user can enter into the cart section**

This test verifies that clicking the cart icon from the Store page navigates the user to the cart section. It confirms the cart page loads correctly and is accessible. This ensures users can review their cart contents anytime.

**Validate user can search product**

This test confirms that the search bar on the Store page accepts queries like “unisex hoodie” and returns relevant results. It verifies the search functionality works smoothly. Users can find products easily through search.

**Verify the "View All" button is working fine**

This test ensures that clicking the "View All" button displays all available products or extends the product list. It confirms the button functions as intended. Users get a complete view of all products.

**Verify search "Hoodie" and add to the cart functionality**

This test checks that users can search for “Hoodies,” select a product, view its details, and add it to the cart. It confirms the integrated search and add-to-cart flow works properly. The cart updates correctly after adding items.

**Validate sortBy option functionality 'Price, low to high'**

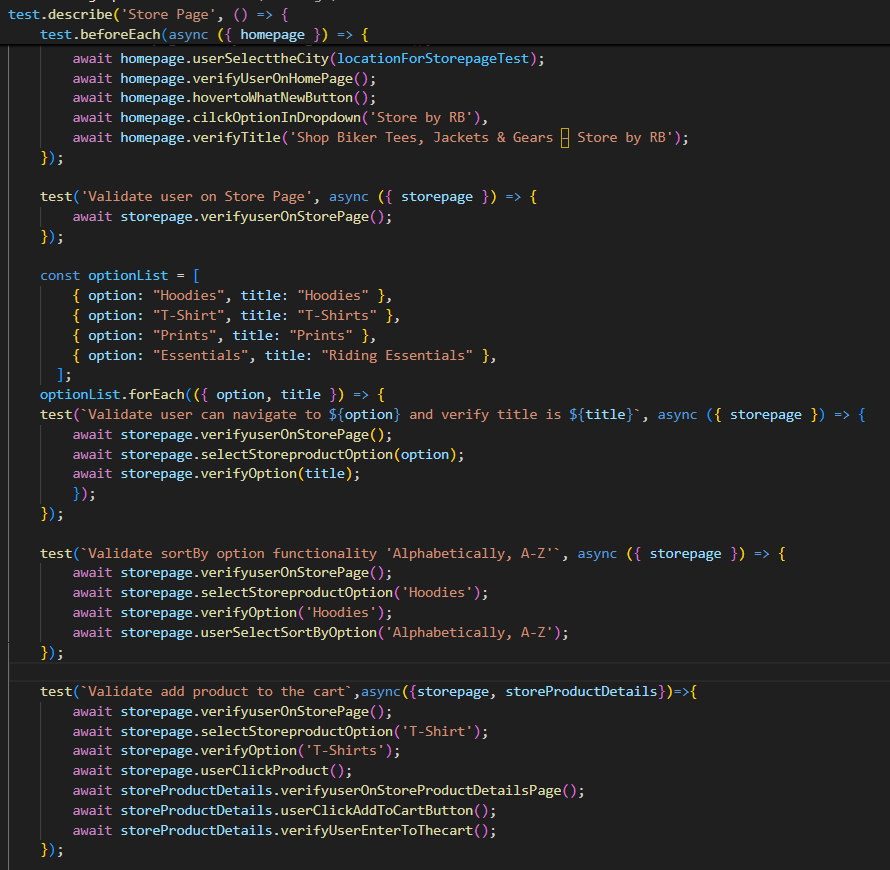
This test validates that sorting products by price from low to high updates the product listing accurately. It confirms the sorting mechanism functions as expected. Users can view products in ascending price order.

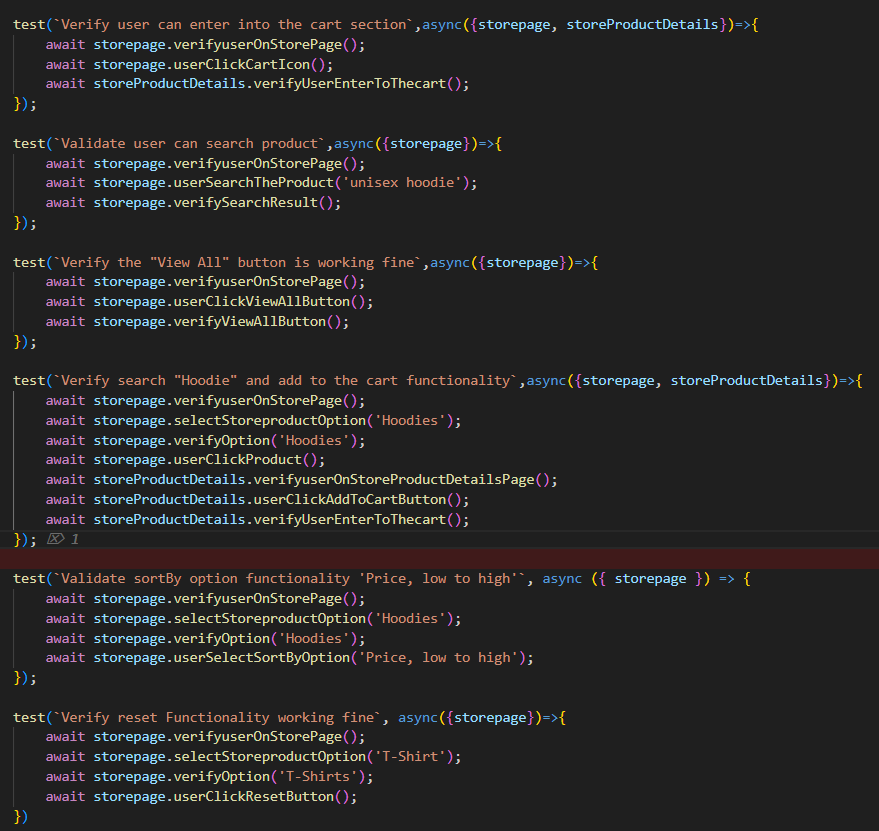
**Verify reset Functionality working fine**

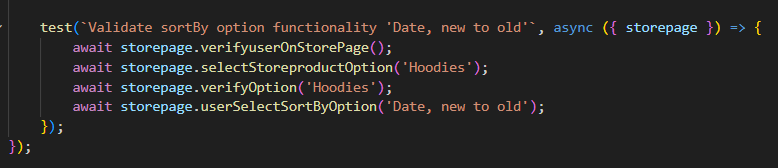
This test verifies that after applying filters or selections, clicking the reset button clears all filters on the Store page. It confirms the product list resets to default. The reset feature allows users to start fresh easily.

**Validate sortBy option functionality 'Date, new to old'**

This test checks that sorting products by newest first updates the listings correctly. It ensures the date-based sorting is applied properly. Users can view the latest products upfront.







**Test AdventureCheckOutPage:**

**Verify "Get Direction" functionality**

This test ensures that users can navigate to the location via the “Get Direction” button on the adventure detail page. It validates redirection to Google Maps. The expected map title is verified to confirm accuracy.

**Verify "Add to cart" functionality**

This test checks whether a user can successfully add an adventure product to the cart from its detail page. It ensures the cart section opens and displays the added item. Basic cart functionality is validated.

**Validate Quantity Increase functionality**

The test simulates a user increasing the product quantity multiple times using the plus icon. It confirms that the quantity reflects the correct count. This ensures the increment logic is working properly.

**Validate Quantity Decrease functionality**

This test confirms that after increasing the product quantity, a user can decrease it using the minus icon. It validates that the quantity updates accurately. Decrease operations are checked for consistency.

**Verify discount code functionality**

This test enters an invalid discount code on the checkout page and applies it. It verifies that the system shows a relevant error for the invalid input. The coupon handling and validation are tested.

**Verify The invalid Email functionality**

This test checks the email field by entering an incorrect email format. It validates whether an appropriate validation message is triggered. Email input checks ensure only valid formats proceed.

**Verify Contact Checkbox functionality**

The test selects the contact preference checkbox during checkout and verifies that it is registered. It confirms checkbox selection is functional. Contact preferences are respected during checkout.

