Comprehensive End-to-End Testing with Cypress

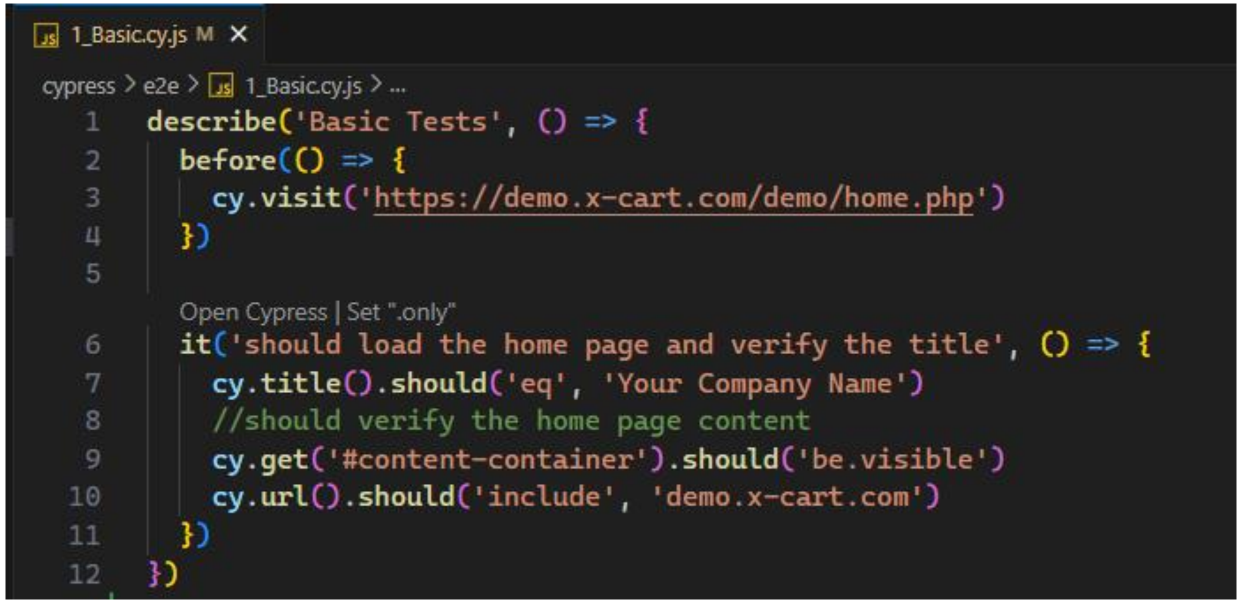
By Akhil Janapatla

Basic Tests

Objective: Verify that the home page loads correctly with the appropriate title, URL, and content visibility.

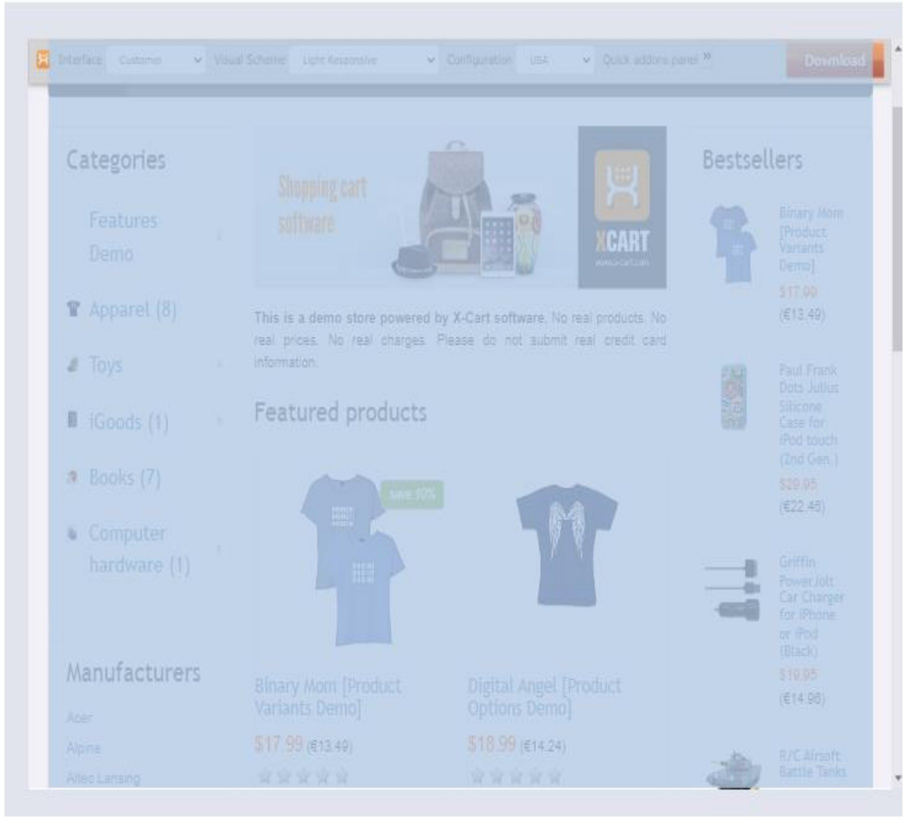
Steps:

1. Visit the home page using cy.visit() to load the X-Cart demo.
2. Use cy.title() to confirm that the page title matches the expected value.
3. Check the URL with cy.url() to ensure the correct domain and path are loaded.
4. Ensure the main content container (the page's primary section) is visible using cy.get().


1_Basic.cy.js M ×

cypress > e2e > 1_Basic.cy.js > ...

```javascript
describe('Basic Tests', () => {
  before(() => {
    cy.visit('https://demo.x-cart.com/demo/home.php')
  })

  Open Cypress | Set ".only"
  it('should load the home page and verify the title', () => {
    cy.title().should('eq', 'Your Company Name')
    //should verify the home page content
    cy.get('#content-container').should('be.visible')
    cy.url().should('include', 'demo.x-cart.com')
  })
})
```



This is a demo store powered by X-Cart software. No real products. No real prices. No real charges. Please do not submit real credit card information.

Output Obtained:

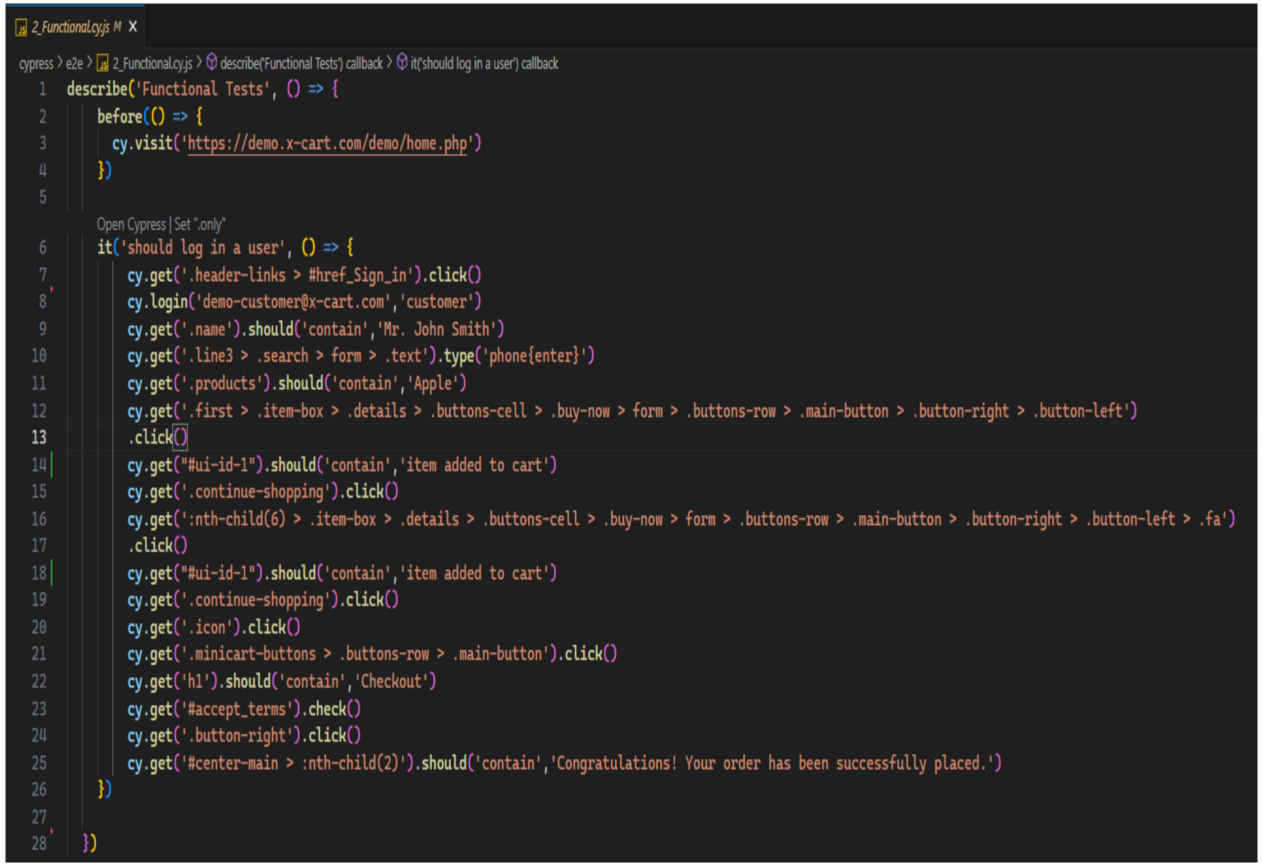
- The page loaded as expected, with the title, URL, and main content all correctly displayed. This test verified that the initial page load is functioning correctly, setting the stage for further testing of functionality.

Functional Tests

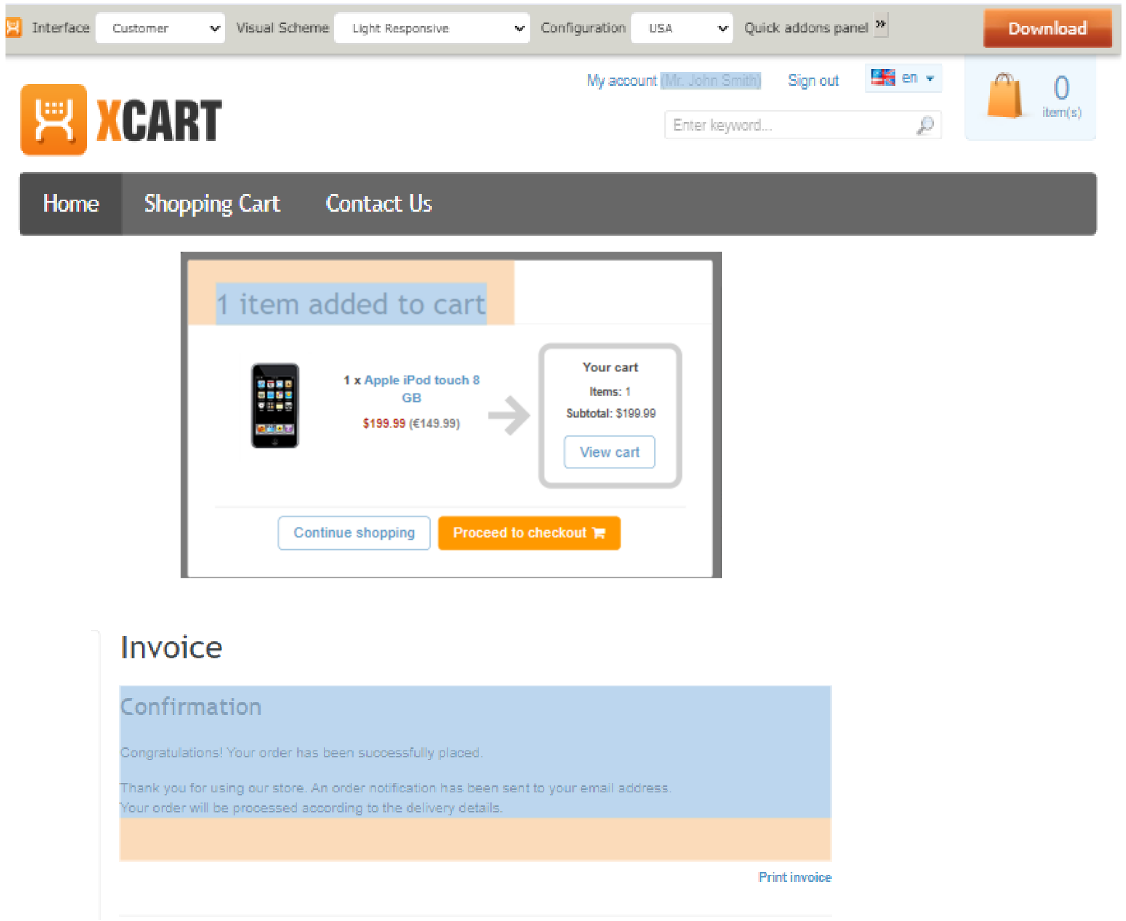
Objective: Test the entire user journey, including logging in, searching for products, adding items to the cart, and completing the checkout process.

Steps:

1. Navigate to the home page and click the Sign In button to trigger the login flow.
2. Use the cy.login() custom command to log in with valid credentials.
3. Search for a product (e.g., "phone") using the search bar.
4. Verify that relevant product results (e.g., "Apple" phones) are displayed.
5. Add one of the displayed products to the cart.
6. Confirm that the product has been successfully added to the cart by verifying the cart status.
7. Proceed to the checkout page and confirm that the user can finalize the purchase by clicking the appropriate buttons.
8. Validate that a success message appears after completing the order.



```
describe('Functional Tests', () => {
  before(() => {
    cy.visit('https://demo.x-cart.com/demo/home.php')
  })

  it('should log in a user', () => {
    cy.get('.header-links > #href_Sign_in').click()
    cy.login('demo-customer@x-cart.com','customer')
    cy.get('.name').should('contain','Mr. John Smith')
    cy.get('.line3 > .search > form > .text').type('phone{enter}')
    cy.get('.products').should('contain','Apple')
    cy.get('.first > .item-box > .details > .buttons-cell > .buy-now > form > .buttons-row > .main-button > .button-right > .button-left')
    .click()
    cy.get("#ui-id-1").should('contain','item added to cart')
    cy.get('.continue-shopping').click()
    cy.get(':nth-child(6) > .item-box > .details > .buttons-cell > .buy-now > form > .buttons-row > .main-button > .button-right > .button-left > .fa')
    .click()
    cy.get("#ui-id-1").should('contain','item added to cart')
    cy.get('.continue-shopping').click()
    cy.get('.icon').click()
    cy.get('.minicart-buttons > .buttons-row > .main-button').click()
    cy.get('h1').should('contain','Checkout')
    cy.get('#accept_terms').check()
    cy.get('.button-right').click()
    cy.get('#center-main > :nth-child(2)').should('contain','Congratulations! Your order has been successfully placed.')
  })

})
```



1 item added to cart

1 x Apple iPod touch 8 GB
$199.99 (€149.99)

Your cart
Items: 1
Subtotal: $199.99

Invoice

Confirmation

Congratulations! Your order has been successfully placed.

Thank you for using our store. An order notification has been sent to your email address.
Your order will be processed according to the delivery details.

Output Obtained:

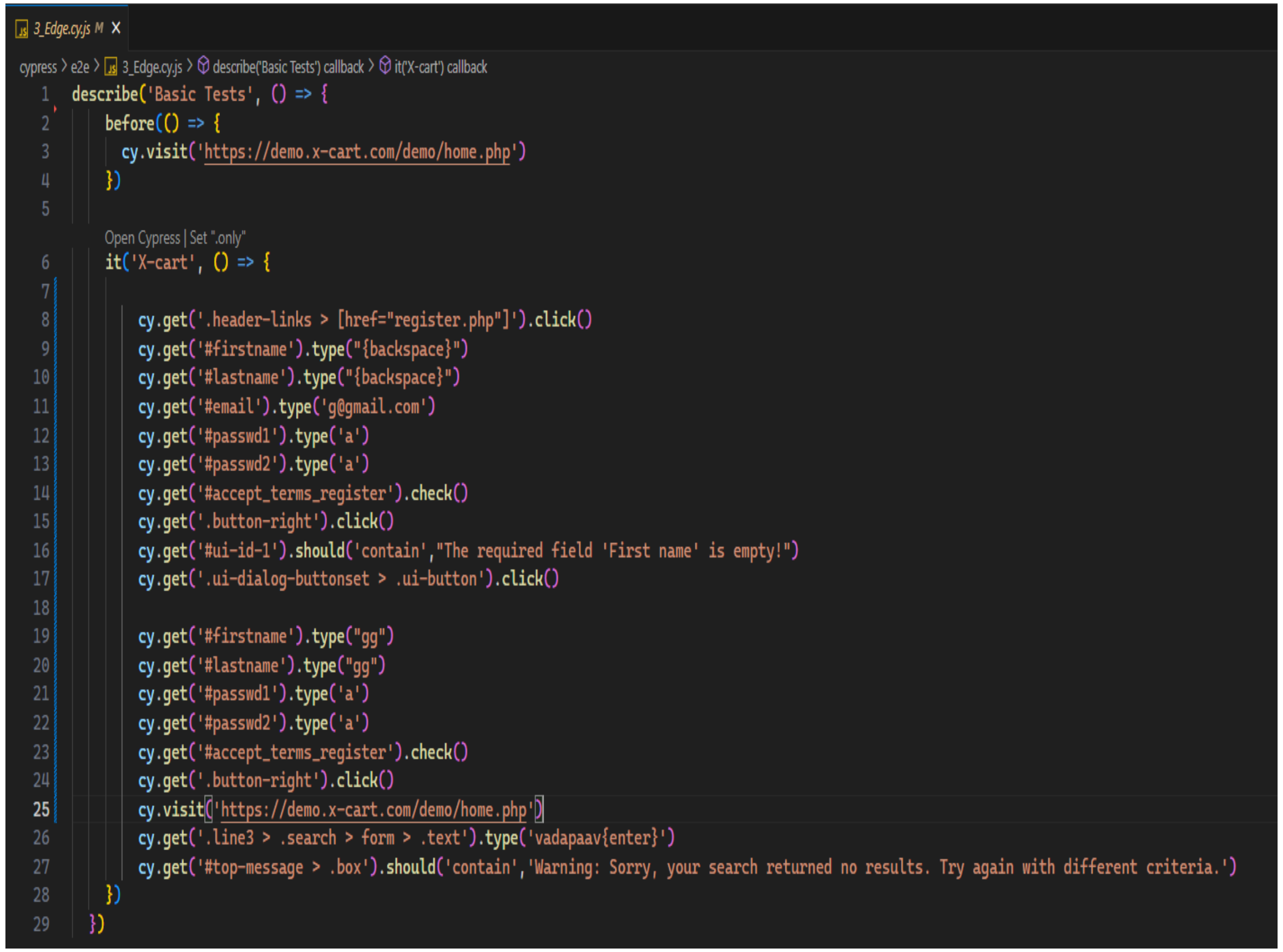
- The test passed successfully. The user was able to log in, search for products, add them to the cart, and proceed through checkout to place an order. The confirmation message was displayed, confirming that the workflow functions as intended.

Edge Case Tests

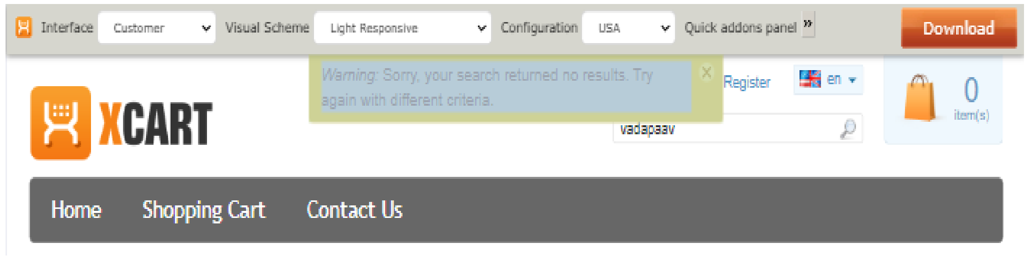
Objective: Test how the application handles invalid form inputs and searches for non-existent products.

Steps:

1. Attempt to register for a new account with empty first name and last name fields.
2. Submit the form with an invalid email address and check for the error message.
3. Perform a product search for a non-existent item **(e.g., "vadapaav")** and observe the system's response.


3_Edge.cy.js M X
cypress > e2e > 3_Edge.cy.js > describe('Basic Tests') callback > it('X-cart') callback
Open Cypress | Set ".only"

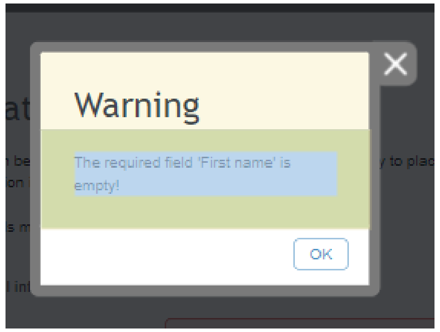
```javascript
describe('Basic Tests', () => {
  before(() => {
    cy.visit('https://demo.x-cart.com/demo/home.php')
  })

  it('X-cart', () => {

    cy.get('.header-links > [href="register.php"]').click()
    cy.get('#firstname').type("{backspace}")
    cy.get('#lastname').type("{backspace}")
    cy.get('#email').type('g@gmail.com')
    cy.get('#passwd1').type('a')
    cy.get('#passwd2').type('a')
    cy.get('#accept_terms_register').check()
    cy.get('.button-right').click()
    cy.get('#ui-id-1').should('contain',"The required field 'First name' is empty!")
    cy.get('.ui-dialog-buttonset > .ui-button').click()

    cy.get('#firstname').type("gg")
    cy.get('#lastname').type("gg")
    cy.get('#passwd1').type('a')
    cy.get('#passwd2').type('a')
    cy.get('#accept_terms_register').check()
    cy.get('.button-right').click()
    cy.visit('https://demo.x-cart.com/demo/home.php')
    cy.get('.line3 > .search > form > .text').type('vadapaav{enter}')
    cy.get('#top-message > .box').should('contain','Warning: Sorry, your search returned no results. Try again with different criteria.')
  })
})
```



Interface: Customer   Visual Scheme: Light Responsive   Configuration: USA   Quick addons panel

Warning: Sorry, your search returned no results. Try again with different criteria.

vadapaav



Warning

The required field 'First name' is empty!

Output Obtained:

- The test successfully validated the error messages for missing fields and invalid email addresses. The search for a non-existent product returned a clear warning, "Warning: Sorry, your search returned no results." This confirms that the system is robust in handling invalid inputs.

Custom Commands

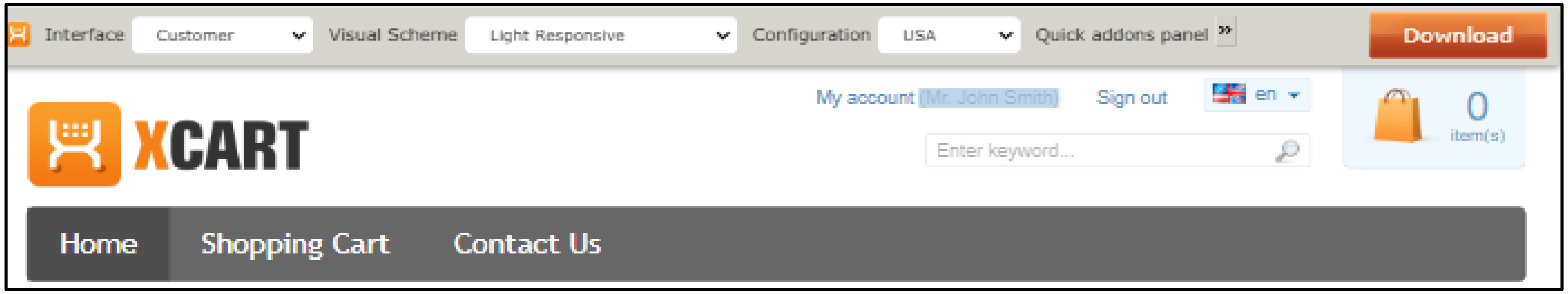
Login Command

In this test suite, a custom command **cy.login()** is used to streamline the login process and simplify test cases that require user authentication. This custom command enhances test efficiency by reducing redundancy and making the login steps reusable across multiple tests.

Objective: Simplify the login process across various test cases.

Steps:

1. The custom command accepts email and password as arguments.
2. It automatically fills in the login form fields and submits the form, ensuring that the user is authenticated.



Output Obtained:

- The custom login command worked as expected, and all test cases that relied on this command passed successfully.

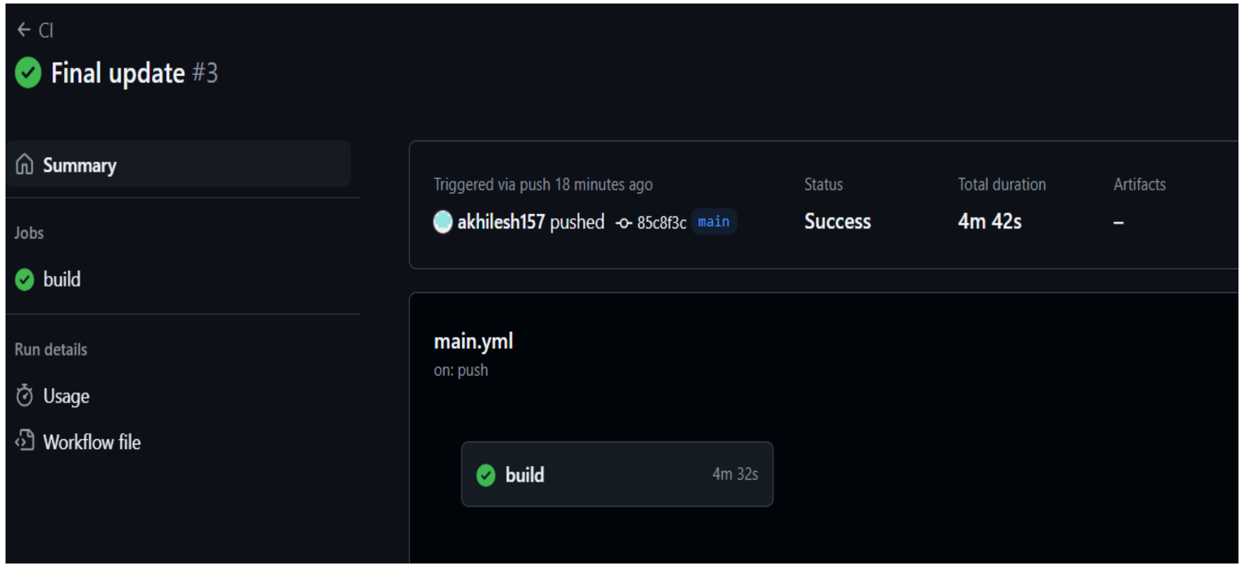
CI/CD Integration with Cypress Testing using GitHub Actions

Objective

To establish a continuous integration and continuous deployment (CI/CD) pipeline using GitHub Actions, automating Cypress tests on each commit or pull request. The goal is to ensure the early detection of bugs and maintain code quality by running tests automatically.

Steps

1. **Create a YAML file** in the .github/workflows directory of your repository.
2. **Define the workflow** by specifying the event triggers such as push or pull_request. The pipeline is triggered whenever there is a new commit or pull request.
3. **Install dependencies** using Node.js and checkout the repository code.
4. **Run Cypress tests** using the cypress-io/github-action command. This runs all tests in headless mode by default, which is suitable for CI environments.
5. **Report test results** using GitHub Actions' built-in reporting tools. GitHub Actions displays test results directly in the workflow interface, allowing developers to view the pass/fail status and error logs.



Final update #3

build

Triggered via push 18 minutes ago
akhilesh157 pushed 85c8f3c main

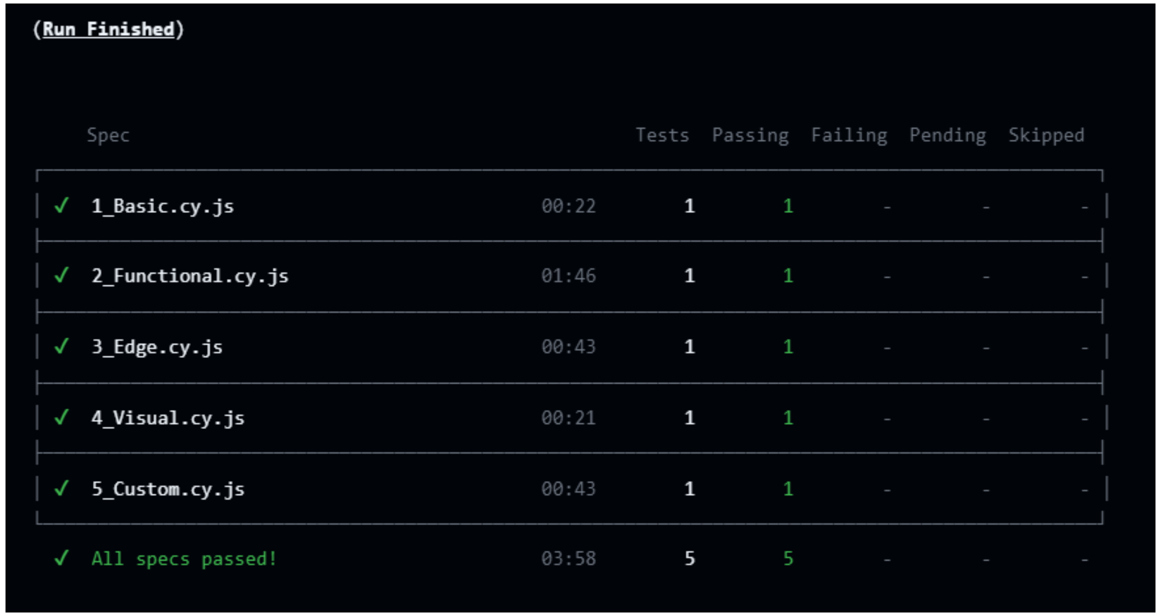
Status
Success

Total duration
4m 42s

Artifacts
–

main.yml
on: push

build 4m 32s



```
(Run Finished)


       Spec                                              Tests  Passing  Failing  Pending  Skipped
  ┌────────────────────────────────────────────────────────────────────────────────────────────┐
  │ ✓  1_Basic.cy.js                           00:22        1        1        -        -        - │
  ├────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  2_Functional.cy.js                      01:46        1        1        -        -        - │
  ├────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  3_Edge.cy.js                            00:43        1        1        -        -        - │
  ├────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  4_Visual.cy.js                          00:21        1        1        -        -        - │
  ├────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  5_Custom.cy.js                          00:43        1        1        -        -        - │
  └────────────────────────────────────────────────────────────────────────────────────────────┘
    ✓  All specs passed!                       03:58        5        5        -        -        -
```

Output Obtained

- The CI/CD pipeline runs automatically on each commit or pull request.
- Cypress tests are executed without manual intervention, and the results are reported in real-time.
- If tests pass, the workflow succeeds; if any tests fail, the workflow is marked as failed, providing immediate feedback to developers.

Generating Reports

Objective

To generate detailed and insightful test reports using Cypress and Mochawesome, enabling developers and testers to analyze test results, debug issues, and review the overall health of the test suite.

Steps

1. Install Mochawesome reporter packages

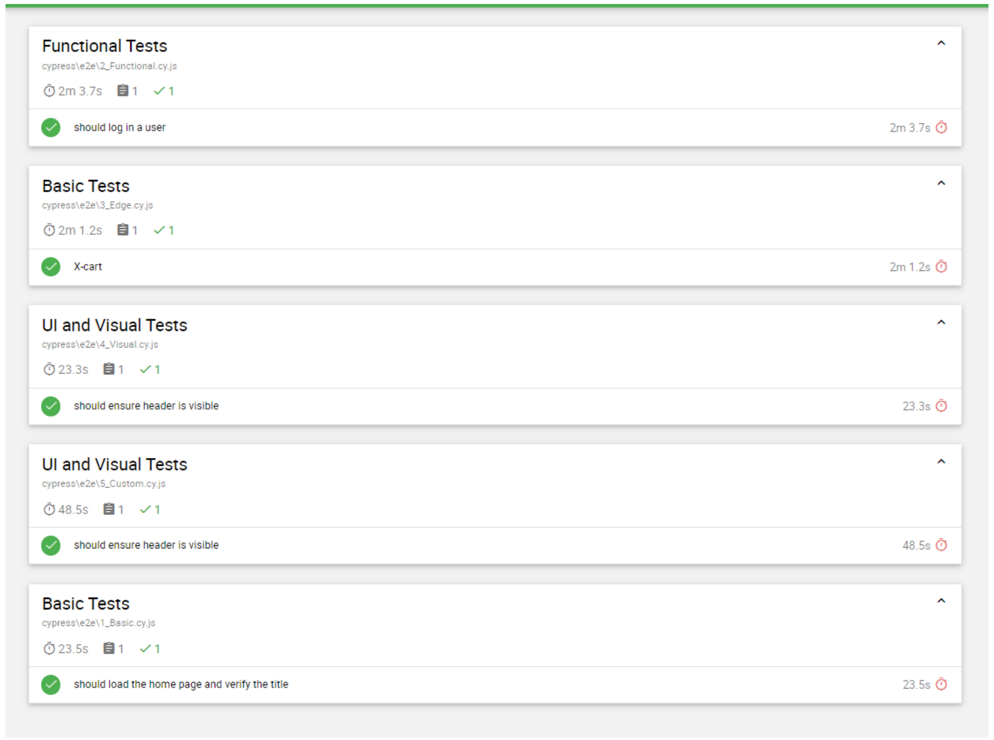
```
npm install --save-dev mochawesome mochawesome-report-generator cypress-mochawesome-reporter
```

2. Configure Cypress to use Mochawesome reporter in the cypress configuration file

Update the Cypress configuration file (cypress.json or cypress.config.js) to specify Mochawesome as the reporter.

3. Run the tests to generate the reports:

When Cypress tests are executed, Mochawesome will generate detailed HTML and JSON reports summarizing the test execution results.



```
Functional Tests
cypress\e2e\2_Functional.cy.js
2m 3.7s   1   ✓ 1
✓ should log in a user                                2m 3.7s

Basic Tests
cypress\e2e\3_Edge.cy.js
2m 1.2s   1   ✓ 1
✓ X-cart                                              2m 1.2s

UI and Visual Tests
cypress\e2e\4_Visual.cy.js
23.3s   1   ✓ 1
✓ should ensure header is visible                     23.3s

UI and Visual Tests
cypress\e2e\5_Custom.cy.js
48.5s   1   ✓ 1
✓ should ensure header is visible                     48.5s

Basic Tests
cypress\e2e\1_Basic.cy.js
23.5s   1   ✓ 1
✓ should load the home page and verify the title      23.5s
```

Output Obtained

- Detailed Mochawesome reports were generated after running the Cypress tests, providing an easy-to-read HTML summary of test results. The reports included the pass/fail status of each test, error messages for failed tests, and attached screenshots where necessary.
- The JSON report can be used for further automation or integration with other tools.

Challenges Faced

- **Environment Setup**: Configuring GitHub Actions to run Cypress tests with all dependencies, especially setting up the correct Node version and caching.
- **Test Flakiness**: Some tests were unreliable in the CI/CD pipeline due to network instability or race conditions in asynchronous UI operations.
- **Report Integration**: Ensuring that the Cypress-Mochawesome reporter generated reports correctly in the CI/CD environment, as path issues sometimes led to missing reports.

Overcome methods

- **Environment Setup**: Used pre-built GitHub Actions (like actions/setup-node and cypress-io/github-action) for smoother dependency management.
- **Test Stability**: Added retries and better wait commands for asynchronous operations to reduce test flakiness.
- **Report Configuration**: Adjusted the Cypress configuration to ensure correct report paths, and verified report generation using local tests before CI integration.

Future Aspects

Parallelization: Running tests in parallel to reduce the time taken for test execution in CI.

Cross-browser Testing: Expanding test coverage to include different browsers like Firefox and Edge within the CI pipeline.

Advanced Reporting: Integrating dashboards like Cypress Dashboard for real-time test monitoring and in-depth analysis.

Conclusion

The Cypress test suite developed for the X-Cart demo application offers a comprehensive evaluation of the system's functionality, user interactions, and visual elements. By covering both functional testing (such as user workflows) and non-functional aspects (such as UI consistency and error handling), this test suite ensures that the application is both reliable and user-friendly.

- **Basic Tests** confirm the application's core functionality, ensuring that pages load correctly with the appropriate content and structure.
- **Functional Tests** simulate key user journeys, such as logging in, searching for products, and checking out, ensuring that the system performs these critical tasks without issues.
- **Edge Case Tests** validate how the application handles unexpected inputs, such as invalid form data or failed product searches, ensuring that the system remains robust and user-friendly even under non-ideal conditions.
- **UI and Visual Tests** ensure that the application maintains a consistent and polished user interface, with all essential elements (headers, footers, images) visible and correctly displayed.