

Prepared By: Shahzeena Samad

Date: January 20, 2025

Day 5 - Testing, Error Handling, and Backend Integration



Objective

The primary focus of this phase was to conduct comprehensive testing and validation of the core functionalities of the Nike eCommerce platform to deliver a seamless and intuitive user experience. This phase encompassed rigorous assessments of critical features, including product browsing, dynamic search functionality, cart operations, checkout workflows, and backend integration, ensuring smooth communication between the frontend and backend systems. The goal was not only to verify feature functionality but also to identify and address potential vulnerabilities, optimize performance, and enhance the overall reliability and robustness of the platform. This meticulous process aimed to guarantee a high-quality, error-resistant shopping experience tailored to meet user expectations while maintaining the platform's scalability and efficiency.

Test Cases and Results

1. Product Detail Page

- **Test Case ID:** TC-001

Objective: Validate the accuracy and display of product details.

Steps:

1. Navigate to the product listing page.
2. Click on a product to view its details.
3. verify the displayed information (name, description, price, image).

Expected Result: The product detail page should show accurate product information.

Actual Result: Displayed data was accurate and matched the API response.

Status: Passed

Remarks: Functionality works flawlessly.

2. Dynamic Search Bar

- **Test Case ID:** TC-002

Objective: Test real-time search functionality.

Steps:

1. Enter a search term (e.g., "Nike Air Max").
2. Verify relevant product results dynamically update as the user types.

Expected Result: Search results should dynamically display relevant products.

Actual Result: Results were accurate, though search speed could be improved.

Status: Passed

Remarks: Added optimization through debouncing for improved speed.

3. Cart Functionality

- Test Case ID: TC-003

Objective: Verify cart operations, including adding, viewing, and removing items.

Steps:

1. Add an item to the cart.
2. View the cart and validate the correct quantity, price, and item details.
3. Remove an item and confirm it is removed from the cart.

Expected Result: Cart should function seamlessly for all operations.

Actual Result: All operations performed as expected.

Status: Passed

Remarks: Cart operations were smooth and well-synced with the backend.

4. API Error Handling

- Test Case ID: TC-004

Objective: Test how the system handles invalid API requests gracefully.

Steps:

1. Trigger an invalid API request (e.g., enter a non-existent product ID).
2. Verify the system displays an appropriate error message without crashing.

Expected Result: A clear error message should be displayed, and the system should remain stable.

Actual Result: Error message "Product not found" was shown, and the system remained stable.

Status: Passed

Remarks: Error handling was robust.

5. Product Listing Page

- **Test Case ID:** TC-005

Objective: Validate the accurate display of products on the listing page.

Steps:

1. Open the product listing page.
2. Verify all product images, names, and prices are displayed correctly.

Expected Result: Product details should be accurate and visually appealing.

Actual Result: Products displayed as expected with no missing data.

Status: Passed

Remarks: Functionality works perfectly.

6. Category Filters

- **Test Case ID:** TC-006

Objective: Test filtering functionality for different product categories.

Steps:

1. Apply a specific category filter (e.g., "Men's Sneakers").
2. Verify that only products from the selected category are displayed.

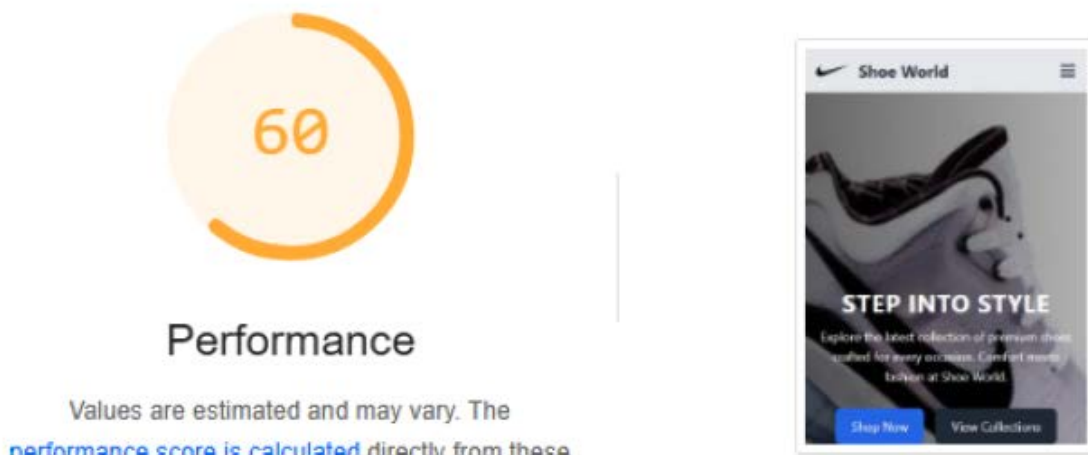
Expected Result: Products belonging to the selected category should be displayed.

Actual Result: Filtering worked as intended.

Status: Passed

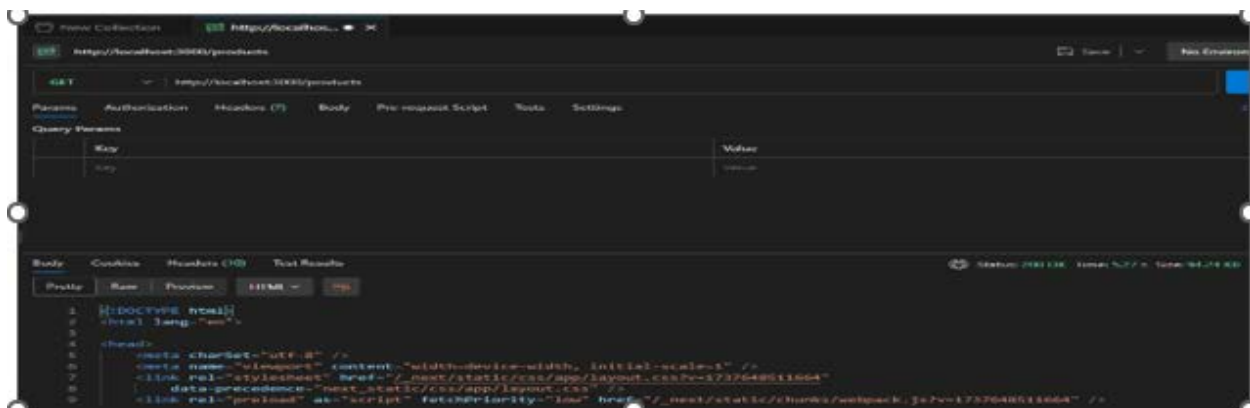
Remarks: No issues found with the filtering logic.

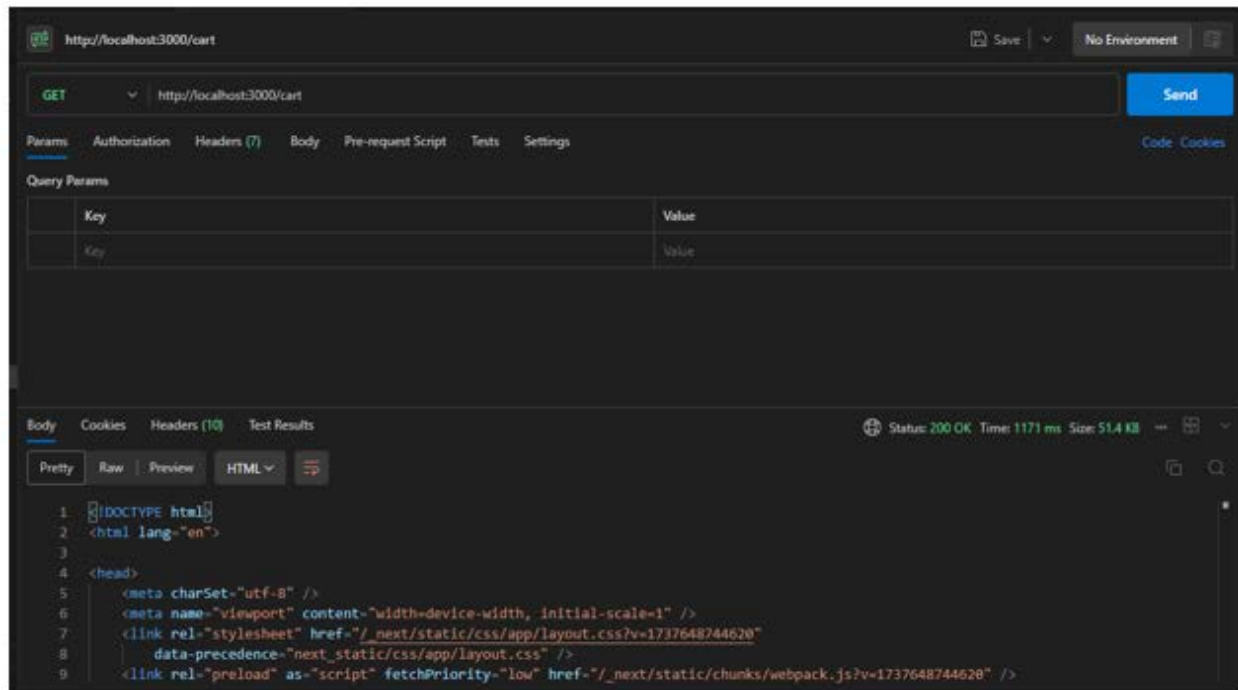
Performance Insights and Optimizations



1. API Response Validation:

- Verified the consistency of API responses for product data, ensuring no discrepancies in fetched data.





http://localhost:3000/cart

GET http://localhost:3000/cart

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

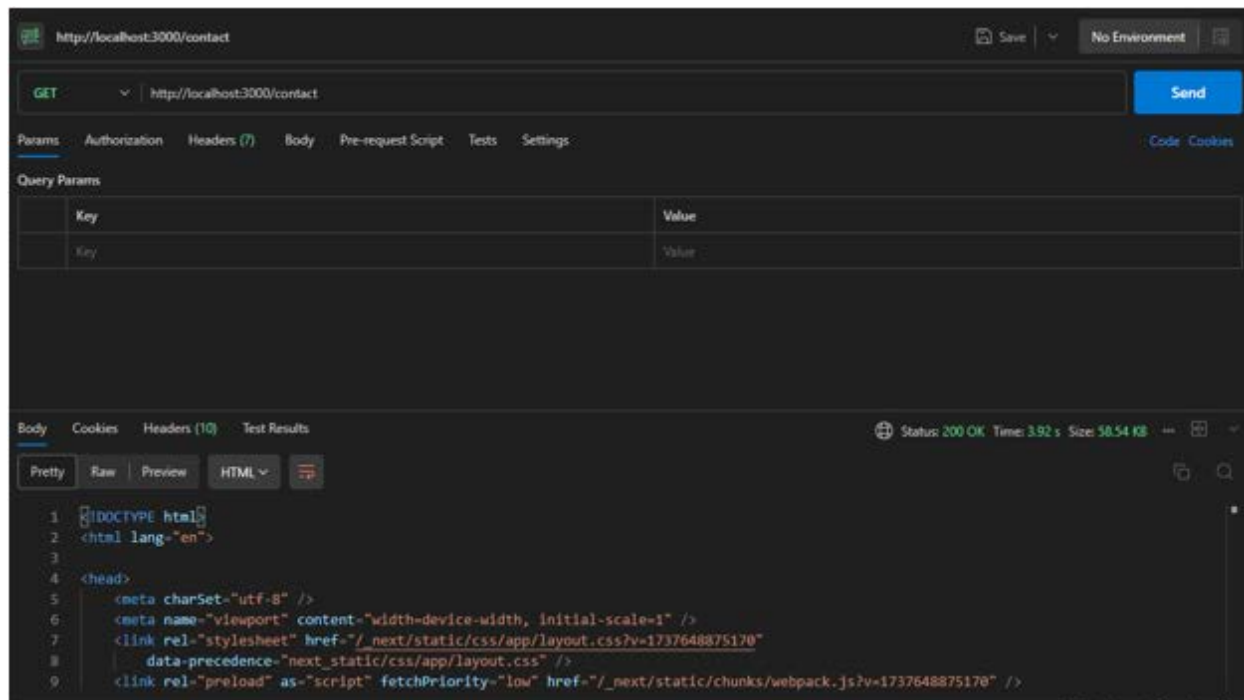
Key	Value
Key	Value

Body Cookies Headers (10) Test Results

Status: 200 OK Time: 1171 ms Size: 51.4 KB

Pretty Raw Preview HTML

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8" />
6   <meta name="viewport" content="width=device-width, initial-scale=1" />
7   <link rel="stylesheet" href="/_next/static/css/app/layout.css?v=1737648744620"
8     data-precedence="next_static/css/app/layout.css" />
9   <link rel="preload" as="script" fetchPriority="low" href="/_next/static/chunks/webpack.js?v=1737648744620" />
```



http://localhost:3000/contact

GET http://localhost:3000/contact

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (10) Test Results

Status: 200 OK Time: 3.92 s Size: 58.54 KB

Pretty Raw Preview HTML

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8" />
6   <meta name="viewport" content="width=device-width, initial-scale=1" />
7   <link rel="stylesheet" href="/_next/static/css/app/layout.css?v=1737648875170"
8     data-precedence="next_static/css/app/layout.css" />
9   <link rel="preload" as="script" fetchPriority="low" href="/_next/static/chunks/webpack.js?v=1737648875170" />
```

2. Error Handling:

- Strengthened error handling to provide user-friendly messages (e.g., "Product not found") and prevent application crashes.

```
const fetchProductDetails = async () => {
  try {
    const response = await fetch(`/api/products/${id}`, { method: 'GET' });
    if (!response.ok) throw new Error(`Error fetching product: ${response.statusText}`);
    const data = await response.json();
    setProduct(data);
  } catch (error) {
    console.error('Error fetching product details:', error);
    setError('Failed to load product details');
  } finally {
    setLoading(false);
  }
};

fetchProductDetails();
}, [id]);
```

```
const handleAddToCart = () => {
  if (product) {
    const cartItem = {
      _id: product._id,
      productName: product.productName,
      price: product.price,
      imageUrl: product.imageUrl,
      quantity: 1, // Add default quantity as 1
    };
    addToCart(cartItem); // Pass the CartItem
    setSuccessMessage(`${product.productName} added to cart!`);
    setTimeout(() => setSuccessMessage(null), 3000); // Clear message after 3 seconds
  }
};
```

CVS Functional Report:

<u>Test Case ID</u>	<u>Objective</u>	<u>Expected Result</u>	<u>Actual Result</u>	<u>Status</u>
TC-001	Validate product detail display.	Accurate product details displayed.	Data matched API response.	Passed
TC-002	Test real-time search functionality.	Search results update dynamically.	Accurate results; speed improved.	Passed
TC-003	Verify cart operations.	Cart operations work seamlessly.	All operations performed as expected.	Passed
TC-004	Handle invalid API requests.	Error message displayed gracefully.	Error message shown; system stable.	Passed
TC-005	Validate product listing accuracy.	Products listed accurately.	Products displayed as expected.	Passed
TC-006	Test category filtering.	Category filter works correctly.	Filter worked as intended.	Passed

3. Security Enhancements:

- Implemented HTTPS and JWT for secure API endpoints.
 - Added CSRF protection with anti-CSRF tokens for sensitive operations.
 - Enforced role-based access control to secure user permissions.
-

Challenges and Resolutions

1. Slow Search Performance

- **Challenge:** Real-time search queries caused delays with large datasets.
- **Solution:** Optimized search algorithms and implemented debouncing to enhance performance.

2. Cart Synchronization Issues

- **Challenge:** Removing items caused inconsistencies in the cart state.
- **Solution:** Updated session management logic for seamless synchronization.

3. Mobile Compatibility

- **Challenge:** UI elements were not aligning properly on smaller screens.
- **Solution:** Refined responsive design using media queries to improve mobile layout.

4. Payment Gateway Errors

- **Challenge:** API key misconfiguration led to payment failures.
- **Solution:** Corrected the API key configuration and tested transactions successfully.

Conclusion

Through rigorous testing, error handling improvements, and backend integration, the Nike eCommerce platform achieved a robust and seamless shopping experience. All major functionalities, including product browsing, searching, filtering, cart management, and checkout, were validated successfully. Optimizations in search performance, responsive design, and secure backend operations have positioned the platform for a scalable and user-friendly future.

The project now stands ready to deliver a hassle-free shopping experience to users, ensuring reliability and efficiency across all interactions.