HACKATHON DAY 5 - TESTING, ERROR HANDLING, AND BACKEND INTEGRATION REFINEMENT

Objective:
Prepare "My Marketplace" for direct launch by having done thorough tests, optimizing for performance, and strengthening backend integration. It is supposed to be all set so it could handle front-facing traffic smoothly with no error for users.

Key Objective:
Thorough Testing: 1. Verify all backend integrations and functionality. 2. Verify that user acceptability, non-functional, and functional testing are finished. Error Handling: 1. Put in place fallback procedures and easily understandable error messages. 2. Use fallback UI components to gracefully resolve API issues. 1. Optimize performance by increasing responsiveness, speed, and dependability. 2. Test performance to find and address bottlenecks. Compatibility across devices and browsers: Verify that it works flawlessly on a variety of devices and browsers. Testing for security: 1. Find weak points and protect private information. Records: 1. Produce expert testing reports that summarize findings and solutions using CSV. 2. Create best practices-based documentation that is ready for deployment.

Key Learning Outcomes:
Verify all backend integrations and functionality.
2. Verify that user acceptability, non-functional, and functional testing are finished.
Error Handling: 1. Put in place fallback procedures and error messages that are easy to understand. 2. Use fallback UI components to gracefully resolve API issues.
Enhancement of Performance:
1. Increase dependability, responsiveness, and quickness.
2. To find and address bottlenecks, test performance.
Compatibility across devices and browsers:
Verify that it works flawlessly on various devices and browsers. Testing for security:
2. 1. Find weak points and protect private information
Records:
1. Produce expert testing reports that summarize findings and solutions using CSV.
2. Create documentation that is suitable for deployment using best practices.

Step 1: Functional Testing

Ensure the core features of the marketplace work as expected, providing a seamless and error-free user experience.

Key Features to Test:

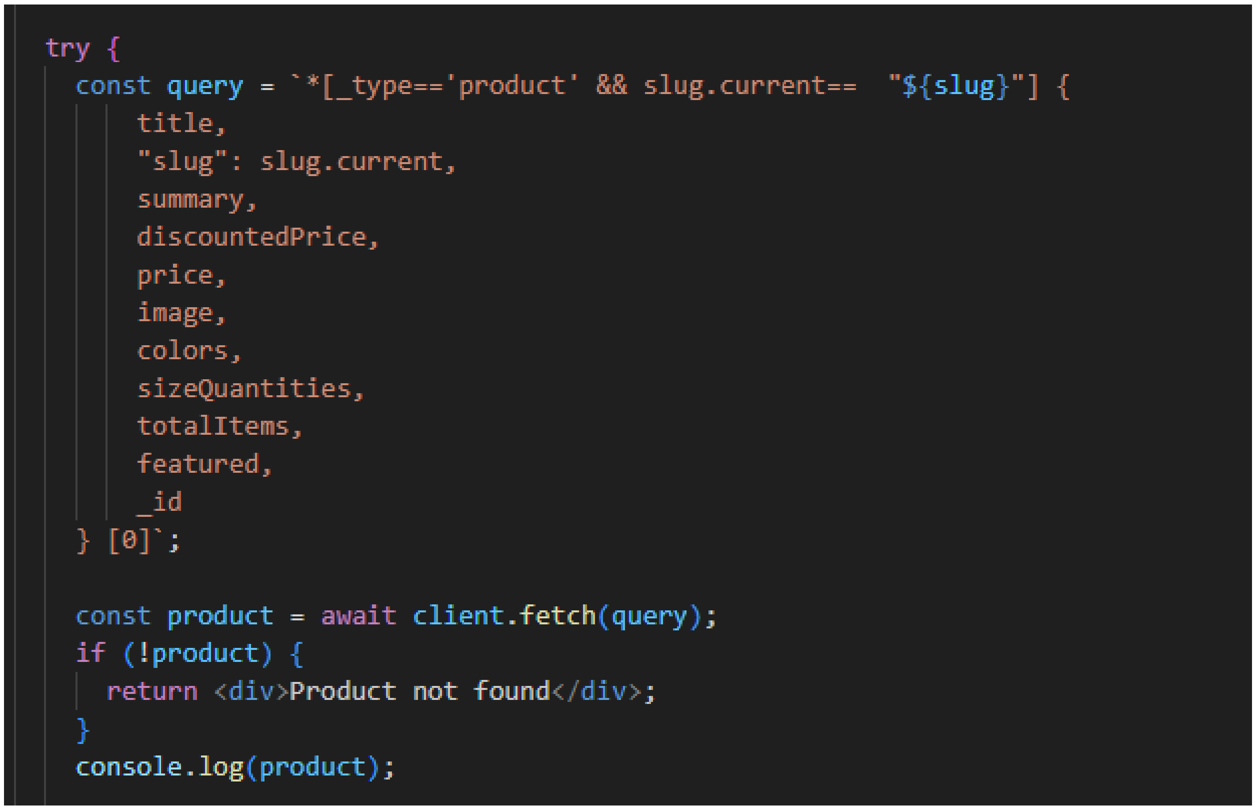
1. **Product Listing**: Confirm that products are displayed correctly with accurate details.
2. **Product Details**: Verify that product detail pages show the correct information, including price, description, images, and availability.
3. **Category Filters**: Test that the filters return the correct results based on user selections.
4. **Dynamic Routing**: Ensure smooth navigation to individual product pages.
5. **Add to Cart**: Check that items can be added, updated, and removed from the cart properly.
6. **Add to Wishlist**: Ensure that users can add products to and manage their wishlist.
7. **Responsive Design**: Test that all features and pages adjust seamlessly to different screen sizes (mobile, tablet, and desktop).
8. **User Profile Management**: Verify that users can update personal information, view order history, and manage saved addresses.

Step 2: Error Handling

Establish strong error handling processes to maintain a smooth user experience, even in the event of issues, by showing clear, user-friendly fallback messages.

Key Areas to Address in Error Handling:

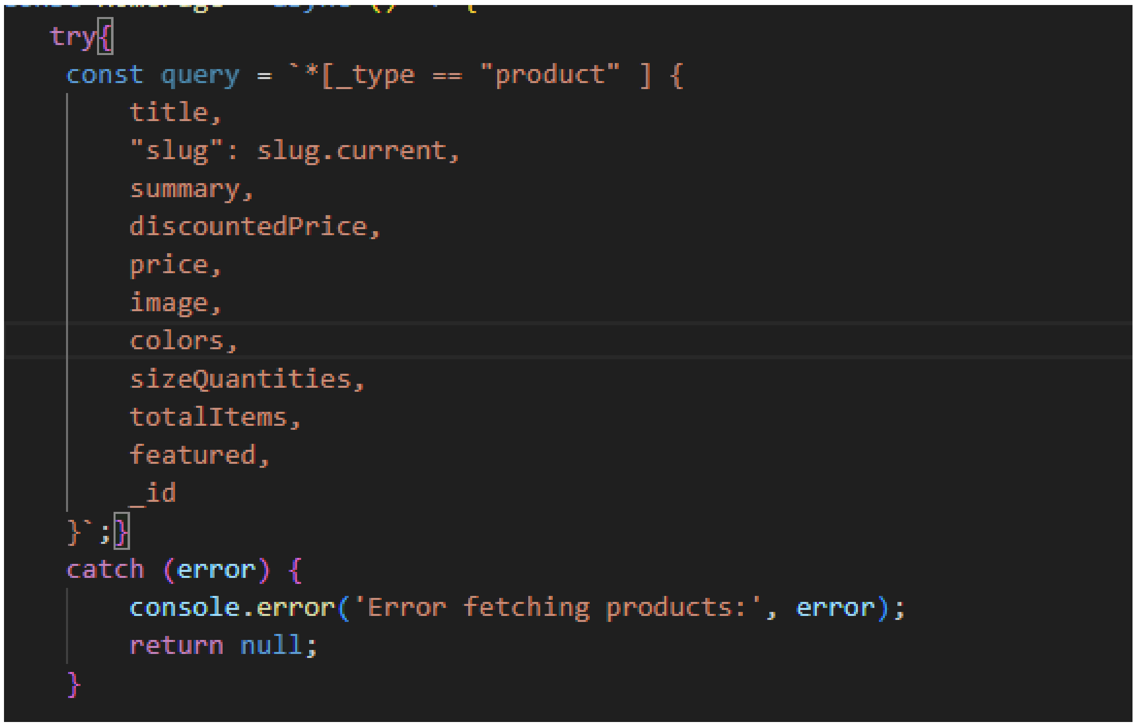
1. **Network Failures**: Show a meaningful error message when there are connectivity issues.
   Example: "Unable to connect to the server. Please check your internet connection."
2. **Invalid or Missing Data**: Handle cases where API responses contain incomplete or invalid data.
   Example: "Some information is missing. Please try again later."
3. **Unexpected Server Errors**: Display a generic fallback message for unhandled server-side errors.
   Example: "Something went wrong on our end. Please try again later."
4. **API Error Handling**: Use try-catch blocks to properly manage API errors.



```
try {
  const query = `*[_type=='product' && slug.current==  "${slug}"] {
      title,
      "slug": slug.current,
      summary,
      discountedPrice,
      price,
      image,
      colors,
      sizeQuantities,
      totalItems,
      featured,
      _id
  } [0]`;

  const product = await client.fetch(query);
  if (!product) {
    return <div>Product not found</div>;
  }
  console.log(product);
```

5. **Fallback UI Elements**: Provide alternative content when data is unavailable.



```
   try{
    const query = `*[_type == "product" ] {
        title,
        "slug": slug.current,
        summary,
        discountedPrice,
        price,
        image,
        colors,
        sizeQuantities,
        totalItems,
        featured,
        _id
    }`;}
    catch (error) {
        console.error('Error fetching products:', error);
        return null;
    }
```

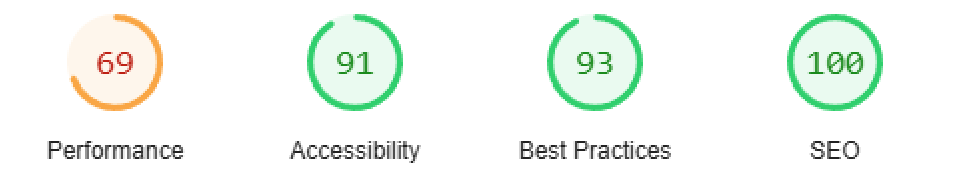
Example: Show a "Error fetching product" message or a placeholder image when the product list is empty.

6. **Form Validation Errors**: Ensure user inputs are validated both on the frontend and backend to prevent invalid data submissions.

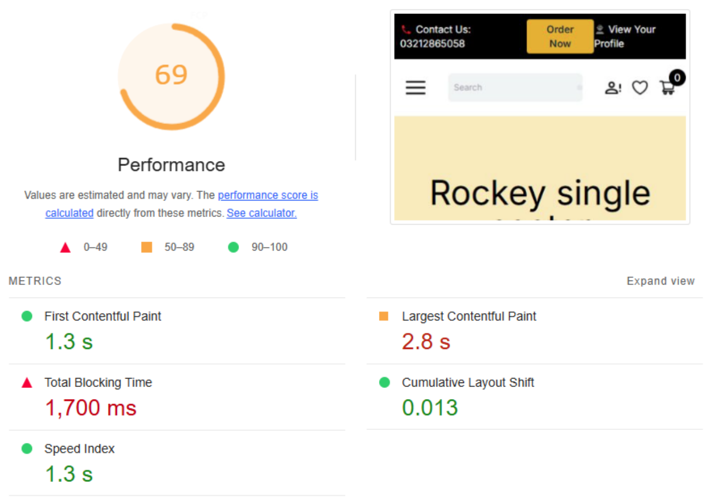
Key Areas to Address:

1. **Optimize Assets**:
   - **Image Optimization**: Compress images using tools like TinyPNG or ImageOptim to reduce file size without compromising quality.
   - **Lazy Loading**: Implement lazy loading for images and assets so they are only loaded when required, improving initial page load times.
2. **Minimize JavaScript and CSS**:
   - **Minification**: Minify JavaScript (using Terser) and CSS (using CSSNano) to reduce file sizes, speeding up page loading.
   - **Remove Unused Code**: Eliminate unnecessary CSS and JavaScript to further decrease file sizes.
3. **Implement Caching Strategies**:
   - **Browser Caching**: Cache static assets in the user's browser to prevent redundant network requests on subsequent visits.
   - **Service Workers**: Use service workers to cache resources and enhance offline functionality.
4. **Analyze Performance**:
   - **Google Lighthouse**: Utilize Lighthouse to audit the website's performance, identifying areas for improvement like image optimization and resource loading.
   - **WebPageTest & GTmetrix**: Leverage these tools to pinpoint performance bottlenecks and enhance page load speed.

Analyze Performance Tetsing



Analyze Performance Tetsing



69

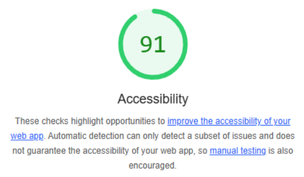
Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

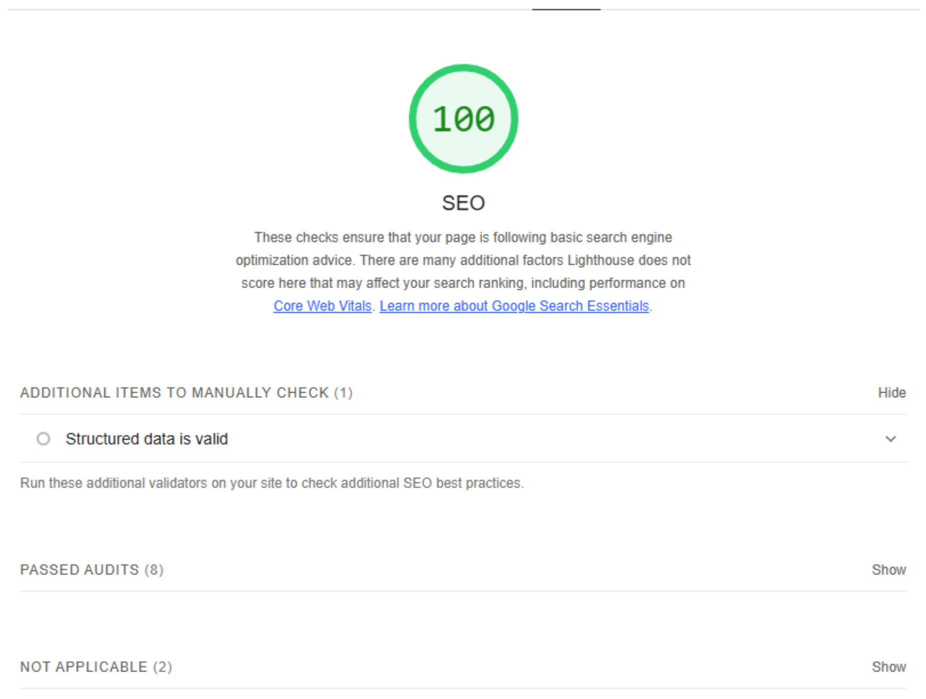
● First Contentful Paint 1.3 s	■ Largest Contentful Paint 2.8 s
▲ Total Blocking Time 1,700 ms	● Cumulative Layout Shift 0.013
● Speed Index 1.3 s	



91

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.



100

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more about Google Search Essentials.

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Hide

○ Structured data is valid

Run these additional validators on your site to check additional SEO best practices.

PASSED AUDITS (8) Show

NOT APPLICABLE (2) Show

Step 5: Security Testing

Ensure the marketplace is secure by addressing key areas:

1. **Input Validation**: Sanitize inputs to prevent attacks like SQL injection and XSS.
2. **Secure API**: Use HTTPS and store API keys in environment variables.
3. **Authentication**: Implement secure login (JWT, OAuth) and access control.
4. **CSRF Protection**: Use anti-CSRF tokens.
5. **Security Headers**: Set CSP, X-Content-Type-Options, HSTS.
6. **Vulnerability Scanning**: Use OWASP ZAP or Burp Suite to scan for issues.

```
User Request --> API Server --> Secure ENV Variables (API Keys)
                  (HTTPS)          (Not Exposed)
```

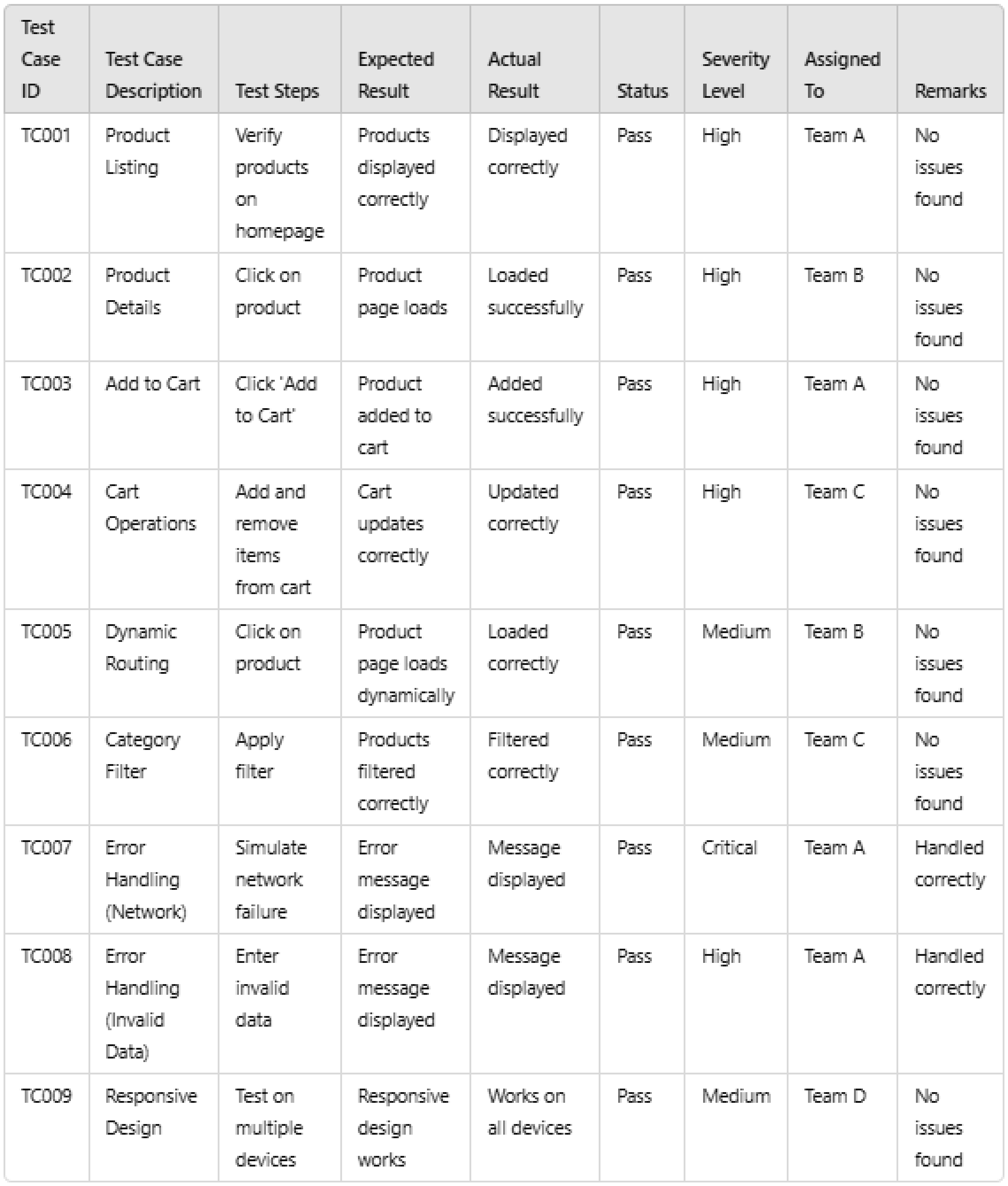
Step 6: User Acceptance Testing (UAT)

1. **Simulate Real-World Usage**:
    - **Task 1: Browsing Products**: I would ensure that product listings are accessible and filterable, allowing easy navigation and filtering by category, price, or brand.
    - **Task 2: Adding Items to Cart**: I would test adding items to the cart and verify that the cart updates correctly with the correct quantity and price.
    - **Task 3: Checkout Process**: I would ensure a smooth checkout experience, testing payment methods and the order confirmation flow.

    Issues Found:

    - **Usability Issue**: The "Add to Cart" button is not immediately visible on mobile screens.
    - **Fix**: I would adjust the button's position and size for mobile responsiveness.
2. **Feedback Collection**:
    - I would ask 3-5 peers to test the marketplace by performing various tasks like product search, adding items to the cart, and completing the checkout process. Feedback would be collected on:
        - Ease of navigation.
        - Clarity of product descriptions.
        - Checkout experience.

CVS TABLE



Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Assigned To	Remarks
TC001	Product Listing	Verify products on homepage	Products displayed correctly	Displayed correctly	Pass	High	Team A	No issues found
TC002	Product Details	Click on product	Product page loads	Loaded successfully	Pass	High	Team B	No issues found
TC003	Add to Cart	Click 'Add to Cart'	Product added to cart	Added successfully	Pass	High	Team A	No issues found
TC004	Cart Operations	Add and remove items from cart	Cart updates correctly	Updated correctly	Pass	High	Team C	No issues found
TC005	Dynamic Routing	Click on product	Product page loads dynamically	Loaded correctly	Pass	Medium	Team B	No issues found
TC006	Category Filter	Apply filter	Products filtered correctly	Filtered correctly	Pass	Medium	Team C	No issues found
TC007	Error Handling (Network)	Simulate network failure	Error message displayed	Message displayed	Pass	Critical	Team A	Handled correctly
TC008	Error Handling (Invalid Data)	Enter invalid data	Error message displayed	Message displayed	Pass	High	Team A	Handled correctly
TC009	Responsive Design	Test on multiple devices	Responsive design works	Works on all devices	Pass	Medium	Team D	No issues found

Conclusion :

Day 5 focused on preparing the marketplace for deployment by ensuring all components were thoroughly tested, optimized, and refined. Comprehensive functional, error handling, and performance tests were conducted, validating key features like product listing, cart operations, and dynamic routing. Robust error handling mechanisms and fallback UI elements were implemented to enhance reliability. Performance optimization efforts resulted in faster load times and improved responsiveness across devices and browsers. The team also documented all findings and resolutions in a professional format, ensuring readiness for real-world deployment.