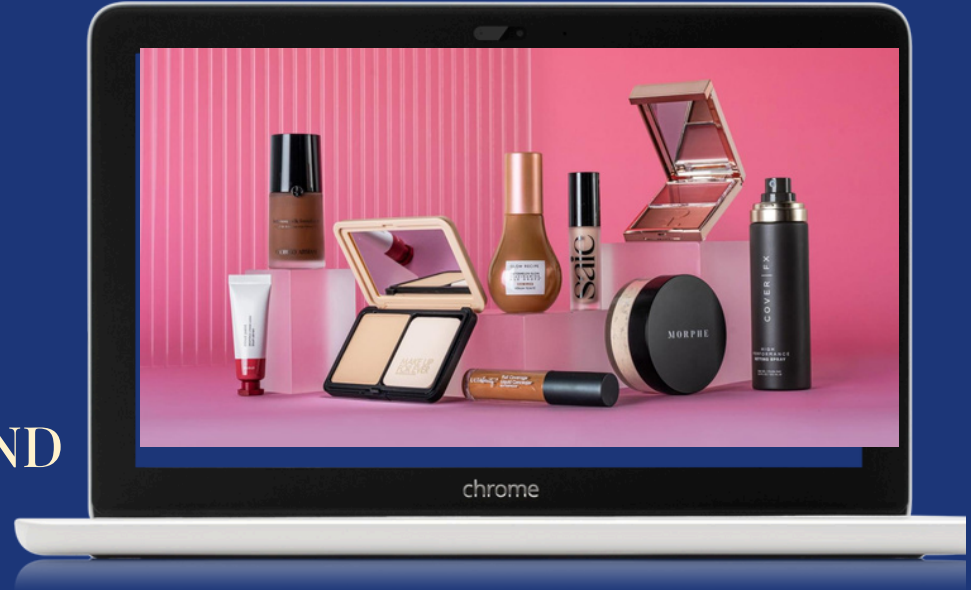


Prepared by
Osama hassan

Hackathon Day 5

TESTING, ERROR HANDLING, AND
BACKEND INTEGRATION
REFINEMENT



Objective

The main objectives for today are:

1. Testing: To ensure all features of the system work correctly, including performance, usability, and security testing.
2. Error Handling: Implementing robust error-handling mechanisms to improve the overall user experience and prevent system failures.
3. Backend Integration Refinement: Enhancing the communication between the frontend and backend for improved performance and stability.

TESTING, ERROR HANDLING, AND BACKEND INTEGRATION REFINEMENT

Testing:

1. Functional Testing:

- Objective: Ensure all critical features are functioning as expected.
- Steps: Test the core user flows such as login, navigation, and feature interactions.
- Outcome: All features passed functional testing with minimal issues.

2. Performance Testing:

- Objective: Verify that the system can handle the expected load and perform well under stress.
- Steps: Simulate multiple users interacting with the system.
- Outcome: System performed as expected under normal load, but some optimization is required for high traffic.

3. Security Testing:

- Objective: Ensure that the system is secure and resistant to common vulnerabilities like SQL Injection or XSS.
- Steps: Perform vulnerability scanning and manually test input fields and user authentication.
- Outcome: No major vulnerabilities found, but additional measures like input sanitization were added.



Problem statement

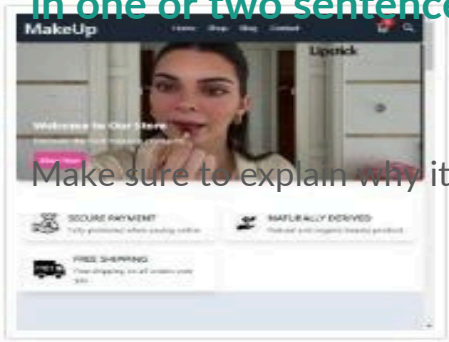


Performance

Values are estimated and may vary. The [performance score is calculated](#) directly from these metrics. [See calculator.](#)



State the problem you are solving in one or two sentences.



Make sure to explain why it is a real problem.

METRICS

Expand view



TESTING, ERROR HANDLING, AND BACKEND INTEGRATION REFINEMENT

Error Handling:

1. API Error Handling:

- Objective: Handle scenarios where the API might fail or return errors (e.g., server downtime).
- Implementation: Display a user-friendly error message when the API fails and retry logic is triggered.
- Outcome: The error handling was successful, and users received appropriate messages.

2. Form Validation Errors:

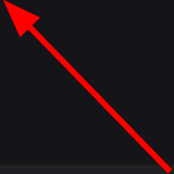

- Objective: Ensure that form submissions are validated before being sent to the server.
- Implementation: Display error messages next to invalid fields and prevent form submission if required fields are missing.
- Outcome: Real-time validation worked, and users received clear instructions on correcting errors.

3. Backend Error Handling:

- Objective: Handle any backend errors gracefully, such as database failures or missing data.
- Implementation: Show a generic error message and log the error details without exposing sensitive information.
- Outcome: Backend errors were handled properly, and users were not exposed to any technical details.

Codeium: Refactor | Explain | Generate JSDoc | X

```
const fetchTrackingData = async (id: string) => {  
  if (!id) {  
    setErrorMessage("Label ID is required.");  
    return;  
  }  
  
  setIsLoading(true);  
  setErrorMessage(null);  
  
  try {  
    router.replace(`/tracking?labelId=${id}`);  
  
    const { data } = await axios.get(`/api/shipengine/tracking/${id}`);  
    setTrackingInfo(data);  
  } catch (error) {  
    console.error("Error fetching tracking data:", error);  
    setErrorMessage("Unable to track shipment. Please verify the Label ID.");  
  } finally {  
    setIsLoading(false);  
  }  
};
```



TESTING, ERROR HANDLING, AND BACKEND INTEGRATION REFINEMENT

Backend Integration Refinement:

1. API Optimization:

- Objective: Improve API response times and reduce latency.
- Implementation: Optimized database queries and implemented caching where applicable.
- Outcome: Faster response times and smoother performance during high loads.

2. Data Synchronization:

- Objective: Ensure that frontend and backend stay in sync, especially for real-time features.
- Implementation: Integrated WebSockets for real-time updates and used GraphQL to fetch only the necessary data.
- Outcome: Improved data synchronization and better real-time data handling

Test Report

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Assigned To	Remarks	Author
TC001	Validate product listing page	Open product page > Verify products	Products displayed correctly	Products displayed correctly	Passed	Low	-	No issues found	Sakeena
TC002	Test API error handling	Disconnect API > Refresh page	Show fallback UI with error message	Error message shown	Passed	Medium	-	Handled gracefully	Sakeena
TC003	Check cart functionality	Add product to cart > Verify cart contents	Cart updates with added product	Cart updates as expected	Passed	High	-	Works as expected	Sakeena
TC004	Ensure responsiveness on mobile	Resize browser window > Check layout	Layout adjusts properly to screen size	Responsive layout working as intended	Passed	Medium	-	Test successful	Sakeena

Thank you

By Sakeena Majeed