Testing, Error Handling, and Backend Integration Refinement

Author: Mohsin Ali

Project: Furniture Shop

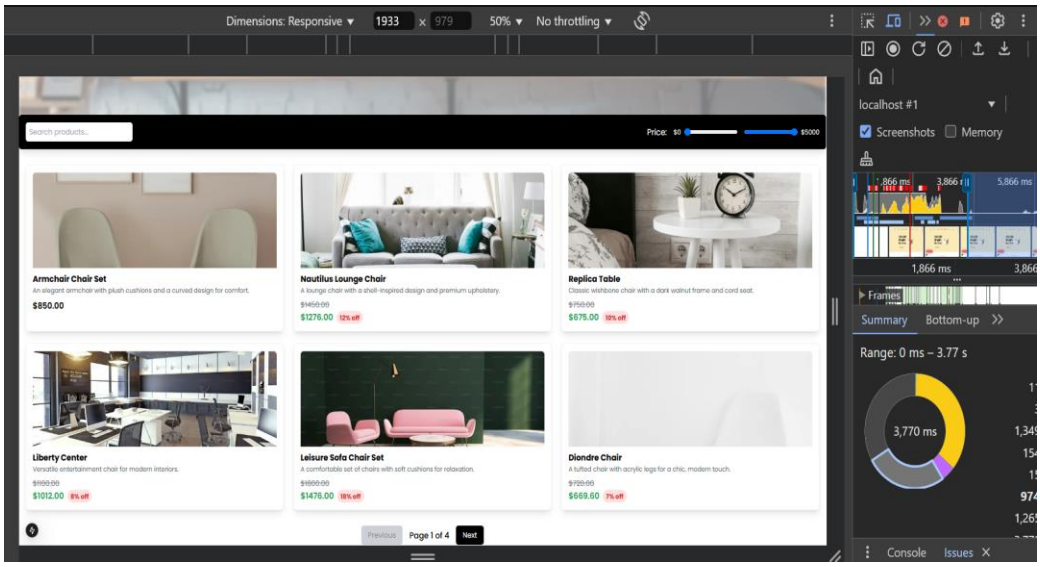
Introduction

This document provides a comprehensive report on the testing and backend refinement processes for our e-commerce store built using **Sanity** and **Next.js**. The document includes details of test cases executed, performance optimizations, security measures implemented, and challenges encountered during testing and backend integration.

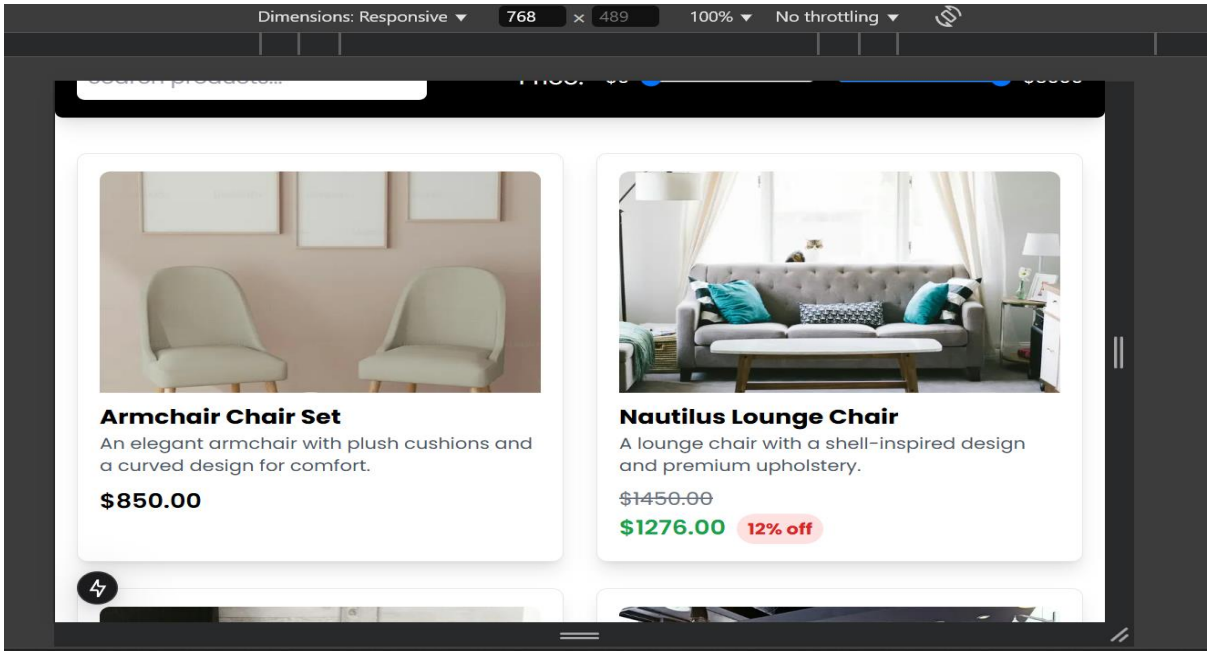
Functional Deliverables

Screenshots of Responsive design

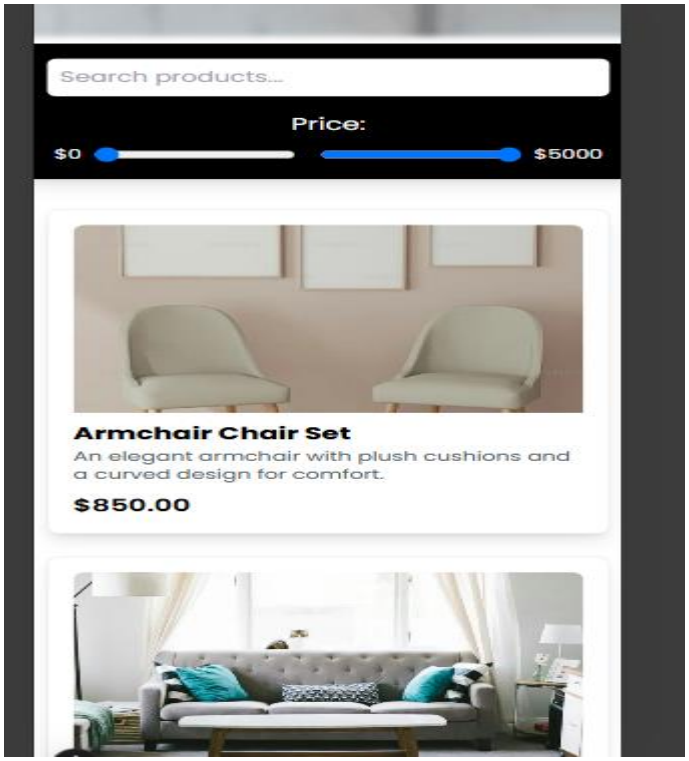
Shop page on desktop screen



Shop page on tablet Screen



Shop page on Mobile Screen



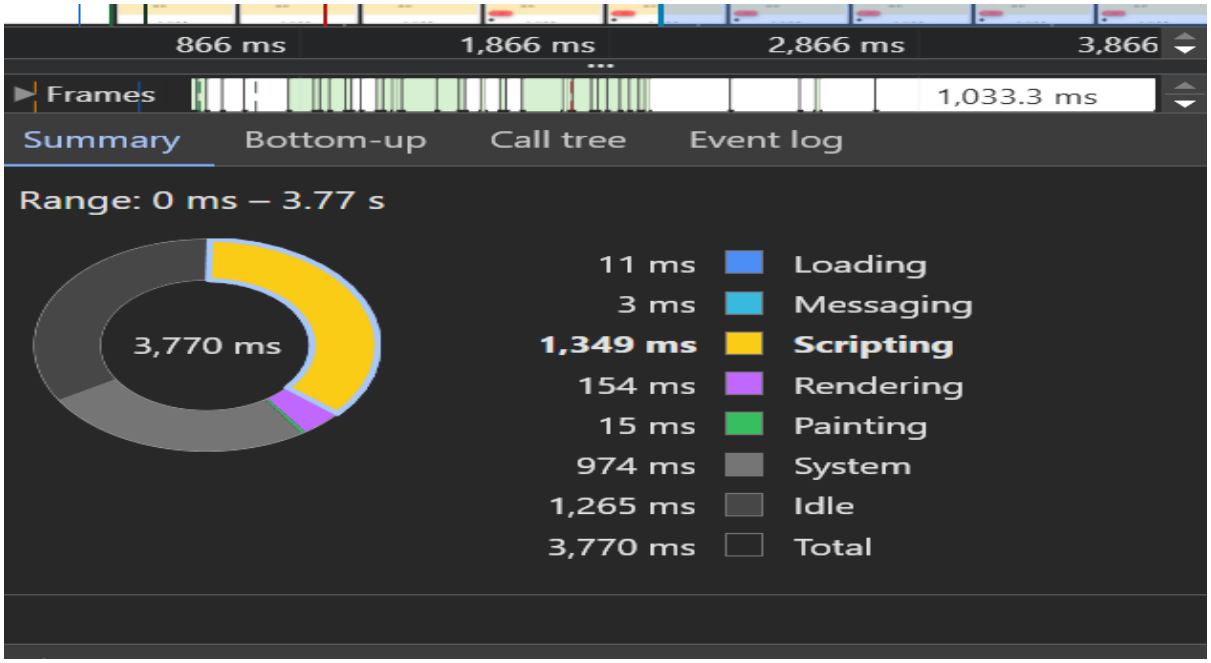
Demonstrations include:

- Home page responsiveness across different devices.
- Checkout and payment flow.
- Product listing and filtering functionality.
- User authentication

Testing Logs & Reports

- **Lighthouse Report:** Attached is the Lighthouse performance report evaluating the application's speed, accessibility, and best practices.
- **Postman API Tests:** Logs from API testing in Postman, including response time, status codes, and data validation.
- **Console Logs & Error Reports:** Logs capturing runtime errors and debugging information.

Performs testing using Browser Performance Testing



Testing Report

A detailed CSV report has been prepared, containing the following columns:

- **Test Case ID**: Unique identifier for each test case.
- **Test Case Description**: Concise explanation of what is being tested.

- **Test Steps**: Step-by-step procedure to execute the test.
- **Expected Result**: Anticipated outcome.
- **Actual Result**: Observed outcome.
- **Status**: Passed, Failed, or Skipped.
- **Severity Level**: Categorized as High, Medium, or Low.

Click to see Testing Sample

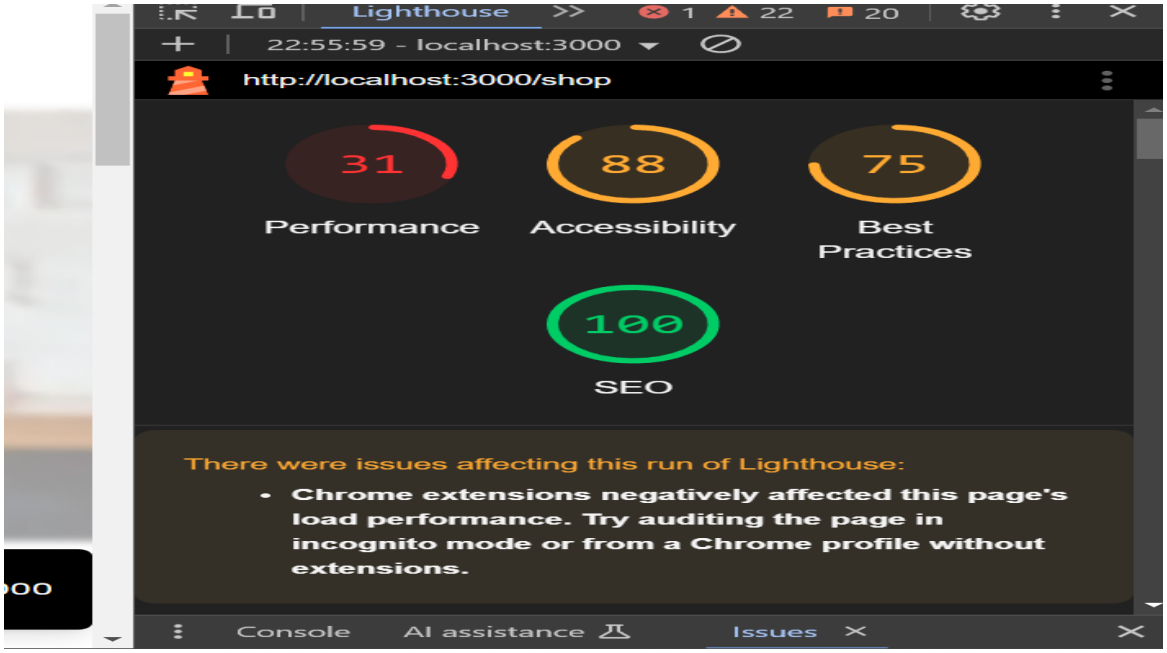
Testing Report CSV format

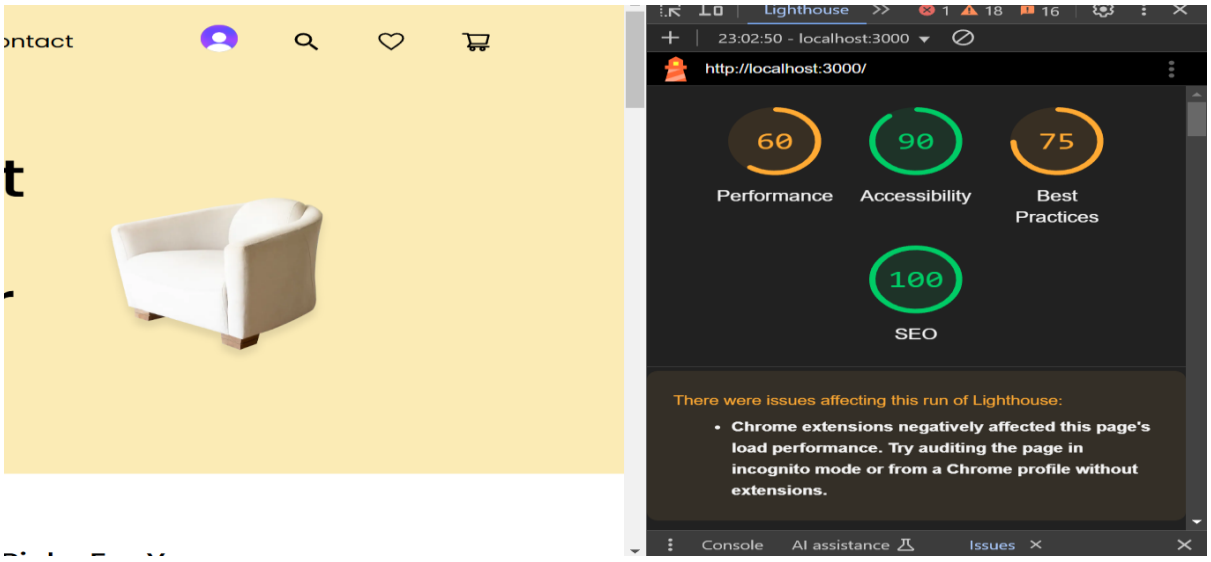
Performance Optimization

To improve the application's performance, the following optimizations were implemented:

- **Image Optimization**: Used Next.js `next/image` for lazy loading and automatic resizing.
- **API Response Enhancement**: Optimized API endpoints to reduce response time by implementing caching and pagination.
- **Code Splitting & Lazy Loading**: Implemented dynamic imports to reduce initial load image
- **Database Query Optimization**: Optimized Sanity queries to reduce redundant calls and improve response efficiency.

Testing Screen Shots





Security Measures

To ensure the security of the application, the following measures were taken:

- **Authentication & Authorization**: Implemented secure user authentication using Clerk Authentication.
- **Input Validation & Sanitization**: Used `express-validator` and server-side validation to prevent XSS and SQL injection attacks.
- **CORS Policy**: Configured appropriate CORS settings to restrict API access.

- **Rate Limiting & Throttling**: Implemented request rate limiting to prevent API abuse.
- **Error Handling & Logging**: Enhanced error handling mechanisms and set up proper logging for tracking security incidents.

Challenges and Resolutions

Challenges Faced

- **Issue with API response delays due to large payloads.**
- **Unexpected authentication failures in certain user scenarios.**
- **Lighthouse performance score was lower due to unoptimized images.**

Resolutions Applied

- Implemented **pagination and filtering** to reduce API payload size.
- Fixed **authentication logic** by refining session handling.
- Used **WebP format** and Next.js image optimization to enhance performance.

Conclusion

The testing and backend refinement phase successfully improved the application's functionality, performance, and security. The testing report highlights all critical issues identified and their resolutions. With these refinements, the e-commerce store is now more robust, scalable, and ready for further enhancements.