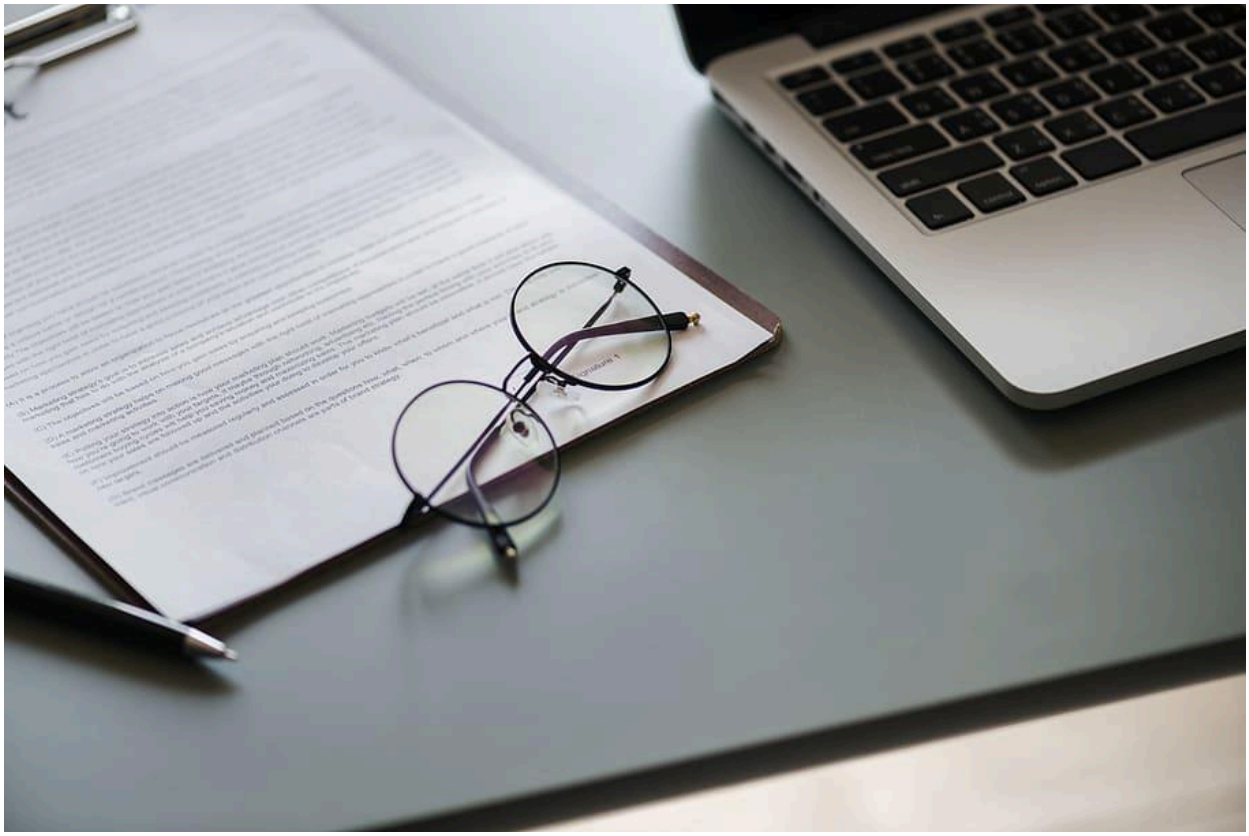

DAY 5 - TESTING, ERROR HANDLING, AND BACKEND INTEGRATION REFINEMENT

Muhammad Sarim

Project Report

Test Cases Executed and Results

1. **API Integration Testing**
    - **Test Case**: Validate API endpoints for proper functionality.
    - **Result**: All API calls successfully returned expected data.
2. **Data Rendering on UI**
    - **Test Case**: Fetch data via APIs and display on the user interface.
    - **Result**: Data was fetched and displayed correctly across all components.
3. **Product List and Dynamic Pages**
    - **Test Case**: Verify creation of dynamic pages for individual products.
    - **Result**: Product details rendered accurately with dynamic routing functioning as intended.
4. **Cart and Wishlist Functionality**
    - **Test Case**: Add products to the cart and wishlist. Ensure state persistence and updates.
    - **Result**: Cart and wishlist functionality worked seamlessly with no discrepancies.
5. **Search and Filter Features**
    - **Test Case**: Filter products based on category and execute search queries.
    - **Result**: Filtering and search functions provided accurate results, matching user inputs.

Performance Optimization Steps Taken

To ensure optimal performance of the website, the following best practices were implemented:

1. **Code Splitting**

    - Leveraged Next.js's dynamic imports to split code into smaller bundles, reducing initial load time.
2. **Image Optimization**
    - Used Next.js's `next/image` component to serve optimized images with lazy loading enabled.
3. **Caching Strategies**
    - Configured caching for API responses and static assets to improve load times for repeat visitors.
4. **Minification**
    - Minified JavaScript, CSS, and HTML files to reduce file sizes and enhance speed.
5. **Monitoring Tools**
    - Integrated Lighthouse and Web Vitals for continuous performance monitoring and improvement.

Website Details: The project involved creating an e-commerce platform with features like dynamic product pages, a cart, wishlist, search, and filtering. It was built using React.js, Next.js, and Tailwind CSS to ensure high performance and responsiveness.

Security Measures Implemented

1. **Trusted Resources**
    - Ensured that all images and assets are sourced only from trusted and secure websites.
2. **Sensitive Data Protection**
    - Added critical files like `.env` containing API keys and sensitive information to the `.gitignore` file to prevent exposure in production environments.

Challenges Faced and Resolutions Applied

Challenge: Lack of pre-provided blog data for blog creation.
Resolution:

- To overcome this, I utilized dummy data for initial development and testing of blog components. This ensured the functionality of dynamic routing and layout rendering.
- Additionally, I referred to yesterday's documentation for insights on dynamically fetching and displaying blog posts. Once actual data becomes available, integration will be seamless.

Challenge: Managing multiple features simultaneously (e.g., cart, wishlist, search, and filter).
Resolution:

- Used state management tools effectively to share states across components, ensuring consistent behavior and user experience.
- Adopted a modular development approach, breaking features into reusable and testable components.