Step
Step 1: Functional Testing
Test Core Features
Testing Tools
How to Perform Functional Testing
Step 2: Error Handling
Error Handling Code Example
Fallback UI
Step 3: Performance Optimization
Image Optimization
Performance Tools
Step 4: Cross-Browser and Device Testing
Browser Testing
Device Testing
Step 5: Security Testing
Input Validation
Secure API Communication
Security Testing Tools

Step 6: User Acceptance Testing (UAT)
Simulate Real-World Usage
Feedback Collection
Step 7: Documentation Updates
Include Testing Results
Submission Format

Description
1. **Product Listing**: Ensure dynamic display with correct details (name, image, price, stock).
2. **Filters and Search**: Verify functionality for filters and search.
3. **Cart Operations**: Ensure users can add, update, and remove items.
4. **Product Detail Pages**: Verify correct dynamic routing.
1. **Postman**: Test backend APIs (e.g., GET /products).
2. **React Testing Library**: Write tests for components (e.g., buttons, inputs).
3. **Cypress**: Perform end-to-end (E2E) testing for user interactions.
1. Write Test Cases: Create tests for each feature.
2. Simulate User Actions: Use Cypress for testing user journeys.
3. Validate Results: Compare actual results with expected behavior.
try {
const response = await fetch('/api/products');
const data = await response.json();
setProducts(data);
} catch (error) {
console.error('Failed to fetch products:', error);
setError('Unable to load products. Please try again later.');
}
1. Display messages like "No products available" if the product list is empty.
2. Add fallback UI for no search results, e.g., "No products match your search criteria."
Compress images using tools like **TinyPNG** or **ImageOptim** for faster page load.
1. **Google Lighthouse**: Identify performance bottlenecks (e.g., unused CSS, unoptimized images).
2. **WebPageTest** or **GTmetrix**: Perform load time tests (aim for under 2 seconds).
Test across browsers: Chrome, Firefox, Safari, Edge for consistent rendering.
Use responsive tools like **BrowserStack** for device simulation. Perform manual testing on a physical mobile device.
1. Sanitize inputs to prevent SQL injection and XSS attacks.
2. Use regex validation for user inputs (e.g., email, phone).
Ensure APIs use HTTPS and store sensitive API keys in environment variables.
1. **OWASP ZAP**: Perform vulnerability scanning.
2. **Burp Suite**: Use for penetration testing.

Perform tasks like browsing products, adding to the cart, and checking out. Ensure workflows are intuitive.
Ask peers/team members to perform UAT and provide feedback on usability.
1. Summarize issues and fixes.
2. Provide before-and-after screenshots.
Document test cases, results, and fixes in PDF or Markdown format. Create a table of contents for easy navigation.