**Marketplace Builder Hackathon 2025**

**Day 5: Testing, Error Handling, and Backend Integration Refinement in Next.js**

**Table of Contents**

1. [Overview](#overview)
2. [Objectives](#objectives)
3. [Prerequisites](#prerequisites)
4. [Environment Setup](#environment-setup)
5. [Testing in Next.js](#testing-in-nextjs)
   * [Unit Testing](#unit-testing)
   * [Integration Testing](#integration-testing)
   * [End-to-End Testing](#end-to-end-testing)
6. [Error Handling in Next.js](#error-handling-in-nextjs)
   * [Best Practices](#best-practices)
   * [Error Logging](#error-logging)
   * [Retry Mechanisms](#retry-mechanisms)
7. [Backend Integration Refinement](#backend-integration-refinement)
   * [API Endpoint Enhancements](#api-endpoint-enhancements)
   * [Authentication and Security](#authentication-and-security)
   * [Performance Optimizations](#performance-optimizations)
8. [Implementation Steps](#implementation-steps)
9. [Troubleshooting](#troubleshooting)
10. [Conclusion](#conclusion)
11. [References](#references)

**Overview**

Day 5 of the Marketplace Builder Hackathon 2025 focuses on improving your Next.js application by implementing robust **testing**, effective **error handling**, and refining the **backend integration**. This will help ensure your application is production-ready with reliable APIs, efficient backend communication, and a resilient user experience.

**Objectives**

* **Testing in Next.js:**
  + Implement unit, integration, and end-to-end tests to ensure that your app behaves as expected.
  + Make sure your components and services are well-tested to prevent errors during production.
* **Error Handling:**
  + Use best practices for error handling in both frontend and backend in a Next.js app.
  + Ensure proper error logging and retry mechanisms.
* **Backend Integration Refinement:**
  + Optimize your backend APIs and enhance security.
  + Implement performance optimizations for smooth communication between the frontend (Next.js) and the backend.

**Prerequisites**

* Familiarity with **Next.js** framework.
* Basic understanding of **testing** in JavaScript using tools like **Jest**, **React Testing Library**, and **Cypress**.
* Knowledge of **API** integration and authentication.
* A **Node.js** environment setup with Next.js installed.

**Environment Setup**

1. **Clone the Repository:** If you are starting from scratch, clone the relevant repository:

bash

CopyEdit

git clone https://github.com/panaverse/learn-nextjs.git

cd learn-nextjs/HACKATHONS/Marketplace\_Builder\_Hackathon\_2025

1. **Install Dependencies:** Install necessary dependencies for the Next.js app and testing:

bash

CopyEdit

npm install

1. **Setting Up .env File:** Set up environment variables for your app in a .env.local file. For example:

env

CopyEdit

NEXT\_PUBLIC\_API\_URL=http://localhost:3000/api

NEXT\_PUBLIC\_GOOGLE\_API\_KEY=your\_google\_api\_key\_here

**Testing in Next.js**

Testing is crucial to ensure your app functions as expected. Let’s explore the types of testing that should be implemented in your Next.js app.

**Unit Testing**

* **Purpose:** Verify that individual components or functions behave as expected.
* **Tools:** Jest for testing, React Testing Library for React components.
* **Example:** Create a simple test for a component:

jsx

CopyEdit

// components/Button.js

export default function Button({ label }) {

return <button>{label}</button>;

}

// tests/Button.test.js

import { render, screen } from '@testing-library/react';

import Button from '../components/Button';

test('renders button with label', () => {

render(<Button label="Click me" />);

expect(screen.getByText('Click me')).toBeInTheDocument();

});

**Integration Testing**

* **Purpose:** Ensure that multiple components or services work together as expected.
* **Tools:** Jest, React Testing Library, or a mock API.
* **Example:** Test API integration with components:

jsx

CopyEdit

// pages/api/user.js

export default async (req, res) => {

const user = await getUserFromDatabase();

res.status(200).json(user);

};

// tests/api/user.test.js

import { fetch } from 'node-fetch';

test('fetches user data', async () => {

const res = await fetch('/api/user');

const data = await res.json();

expect(data).toHaveProperty('name');

});

**End-to-End Testing**

* **Purpose:** Simulate user actions to ensure the app functions as a whole.
* **Tools:** Cypress for E2E testing.
* **Example:** Test a user login flow:

js

CopyEdit

// tests/e2e/login.spec.js

describe('Login Flow', () => {

it('should log in successfully', () => {

cy.visit('/login');

cy.get('input[name="username"]').type('testUser');

cy.get('input[name="password"]').type('password');

cy.get('button[type="submit"]').click();

cy.url().should('include', '/dashboard');

});

});

**Error Handling in Next.js**

Error handling is a critical aspect of building resilient apps.

**Best Practices**

* **Graceful Error Handling:** Handle errors gracefully in the UI and backend, showing user-friendly messages.
* **Global Error Handler:** Use the built-in ErrorBoundary for handling frontend errors.

jsx

CopyEdit

// components/ErrorBoundary.js

class ErrorBoundary extends React.Component {

componentDidCatch(error, info) {

logErrorToMyService(error, info);

}

render() {

if (this.state.hasError) {

return <h1>Something went wrong.</h1>;

}

return this.props.children;

}

}

**Error Logging**

* Use libraries like **Sentry** to log errors from both frontend and backend.
  + Install Sentry:

bash

CopyEdit

npm install @sentry/react

* + Configure Sentry:

jsx

CopyEdit

import \* as Sentry from '@sentry/react';

Sentry.init({ dsn: 'your\_sentry\_dsn' });

**Retry Mechanisms**

* Implement retry mechanisms for API calls and critical operations.
  + Use **Axios** with retry capabilities:

bash

CopyEdit

npm install axios-retry

* + Example:

javascript

CopyEdit

import axios from 'axios';

import axiosRetry from 'axios-retry';

axiosRetry(axios, { retries: 3 });

**Backend Integration Refinement**

**API Endpoint Enhancements**

* Ensure that your **API routes** are optimized and secure.
* Use Next.js API routes for server-side processing:

javascript

CopyEdit

// pages/api/products.js

export default async function handler(req, res) {

const products = await getProductsFromDatabase();

res.status(200).json(products);

}

**Authentication and Security**

* Implement **JWT authentication** for securing endpoints.

bash

CopyEdit

npm install jsonwebtoken

* Secure API routes using authentication middleware.

**Performance Optimizations**

* Use **API caching** strategies (e.g., with Redis).
* Optimize database queries and use pagination for large data sets.

**Implementation Steps**

1. **Set Up Testing Libraries:** Install Jest, React Testing Library, and Cypress.
2. **Write Unit and Integration Tests:** Add tests for components and API endpoints.
3. **Implement Error Handling:** Set up global error boundaries and integrate error logging.
4. **Refine API Integration:** Optimize API endpoints and implement JWT authentication.
5. **Test Performance:** Test API performance using load testing tools and optimize queries.

**Troubleshooting**

* **Test Failures:** Check logs for failed tests and address issues like missing dependencies or incorrect mocks.
* **API Issues:** Ensure endpoints are correct and use proper error handling.
* **Authentication Problems:** Verify JWT tokens and ensure they are being sent with requests.