DAY 6 - DEPLOYMENT PREPARATION AND STAGING ENVIRONMENT SETUP

Through this way I deployed my project on Vercel, securely managed environment variables, integrated Sanity as the backend, and configured third-party APIs for payment and shipment. These are the steps through which I get smooth deployment process.

---

1. Deployment Strategy Planning

Choosing a Hosting Platform

- Vercel for quick deployment.
- **Backend and API Integration**
- Finalize the interaction with backend services such as Sanity CMS and third-party APIs for payment and shipment.

---

2. Environment Variable Configuration

Secure Sensitive Data

1. Create a .env file in your project root.
2. NEXT_PUBLIC_SANITY_PROJECT_ID=your_project_id

3. NEXT_PUBLIC_SANITY_DATASET=production
4. PAYMENT_API_KEY=your_api_key
5. SHIPMENT_API_KEY=your_shipment_api_key
6. **Configure in Hosting Platform:**
   - Navigate to the Vercel dashboard.
   - Go to **Settings** > **Environment Variables**.
   - Add your environment variables securely.

Access Variables in Code

- Use process.env to reference variables.
- const sanityProjectId = process.env.NEXT_PUBLIC_SANITY_PROJECT_ID;
- const paymentApiKey = process.env.PAYMENT_API_KEY;

---

3. Staging Environment Setup

Deploy to Staging

1. Connect your GitHub repository to Vercel.
2. Deploy the application to a staging environment through the Vercel dashboard.

Validate Deployment

- Ensure the build process completes without errors.
- Verify the site loads correctly.

---

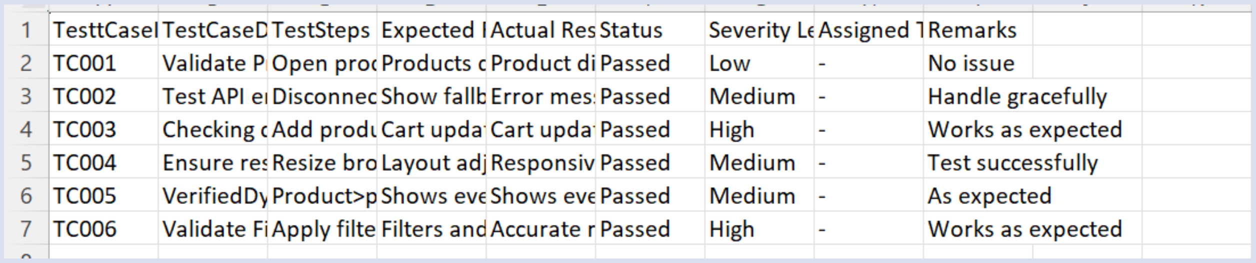
4. Staging Environment Testing

Types of Testing

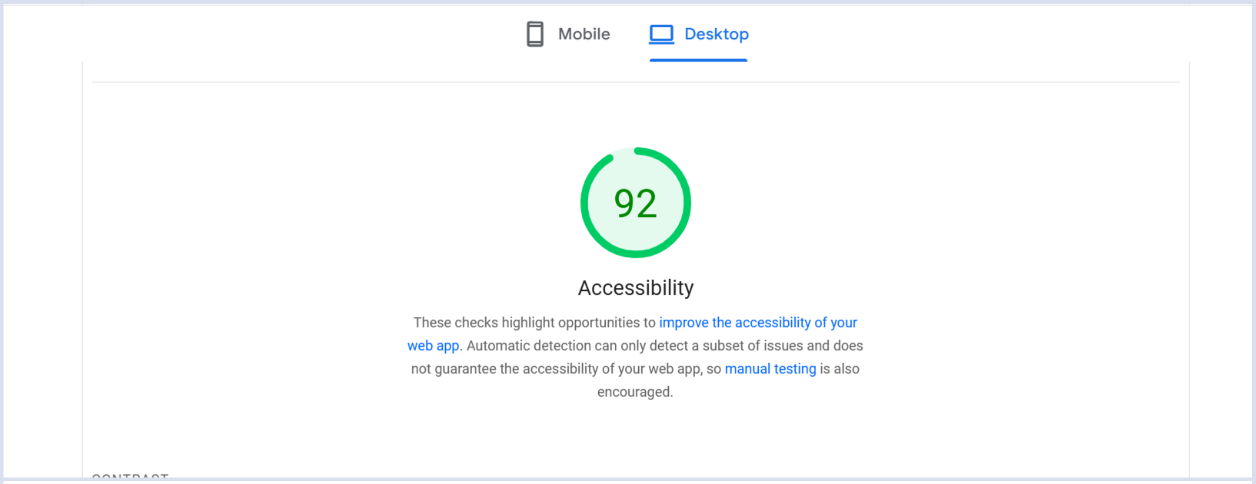
1. **Functional Testing**
   - Test workflows and interactions using tools like Cypress.
   - Validate API responses with Postman.
2. **Performance Testing**
   - Analyze load times and responsiveness using Lighthouse or GTmetrix.
3. **Security Testing**
   - Validate input fields to prevent SQL injection.
   - Ensure HTTPS is enabled.
   - Verify proper handling of sensitive data like API keys.

Document Test Cases

Performance Report

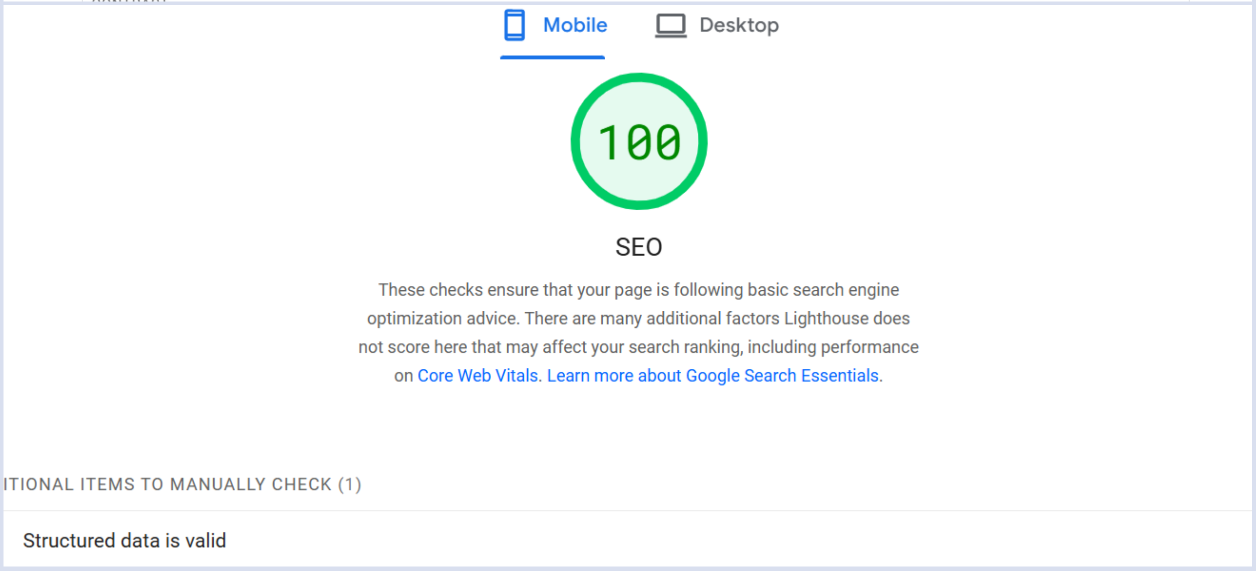


TesttCaseI	TestCaseD	TestSteps	Expected I	Actual Res	Status	Severity Le	Assigned T	Remarks
TC001	Validate P	Open proc	Products c	Product di	Passed	Low	-	No issue
TC002	Test API e	Disconnec	Show fallb	Error mes	Passed	Medium	-	Handle gracefully
TC003	Checking c	Add produ	Cart upda	Cart upda	Passed	High	-	Works as expected
TC004	Ensure res	Resize bro	Layout adj	Responsiv	Passed	Medium	-	Test successfully
TC005	VerifiedDy	Product>p	Shows eve	Shows eve	Passed	Medium	-	As expected
TC006	Validate Fi	Apply filte	Filters and	Accurate r	Passed	High	-	Works as expected

- 

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.



SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more about Google Search Essentials.

ITIONAL ITEMS TO MANUALLY CHECK (1)

Structured data is valid

-

---

5. Documentation Updates

Create README.md

- Summarize project activities, including:
    - Deployment steps.
    - Test case results.

Organize Files

- Structure project files systematically in your GitHub repository.

---

6. Integrating Sanity Backend

Set Up Sanity Studio

1. Install Sanity CLI:
2. npm install -g @sanity/cli
3. Initialize a Sanity project:
4. sanity init
    - Configure the project name, dataset (e.g., production), and template.
5. Deploy the studio:
6. sanity deploy

Integrate Sanity with Your Project

1. Install the Sanity client:
2. npm install @sanity/client
3. Configure the client:
4. import { createClient } from '@sanity/client';
5.
6. const client = createClient({
7.   projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
8.   dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,

9. useCdn: true,
10. });
11. Fetch data using queries:
12. export async function fetchPosts() {
13.   const query = '*[_type == "post"]';
14.   return await client.fetch(query);
15. }

---

7. Integrating Third-Party APIs

Payment API Integration

1. Choose a payment gateway (e.g., Stripe).
2. Install the SDK:
3. npm install @stripe/stripe-js
4. Implement payment processing:
5. import { loadStripe } from '@stripe/stripe-js';
6.
7. const stripePromise = loadStripe(process.env.PAYMENT_API_KEY);
8.
9. export async function handlePayment() {
10.   const stripe = await stripePromise;
11.   const { error } = await stripe.redirectToCheckout({

```
      sessionId: 'your-session-id',
    });
    if (error) console.error(error);
  }
```

Shipment API Integration

1. Select a shipment provider (e.g., Shippo, EasyPost).
2. Install the SDK:

```
npm install shippo
```

4. Configure and create shipments:

```
import Shippo from 'shippo';

const shippo = Shippo(process.env.SHIPMENT_API_KEY);

export async function createShipment() {
    const shipment = await shippo.shipment.create({
      address_from: {/* sender details */},
      address_to: {/* recipient details */},
      parcels: [{/* package details */}],
      async: false,
    });
    return shipment;
  }
```

---

8. Final Testing and Deployment

Verify Integration

1. Test all Sanity data queries and API endpoints locally.
2. Confirm payment and shipment processes work seamlessly.

Redeploy

1. Push changes to your repository.
2. Vercel will automatically trigger a redeployment.

---

9. Post-Deployment Checklist

- Environment variables are securely configured.
- Sanity backend is integrated and operational.
- Payment gateway integration is functioning.
- Shipment API integration is tested.
- Live site is verified and bug-free.
- Now I have a live deployed project to overcome in market.
