Day 6 - Deployment Preparation and Staging Environment Setup

Technical Report: Deployment Preparation and Staging Environment Setup

1. Steps Taken to Build and Integrate Components

Step 1: Deployment Strategy Planning

- Chose **Vercel** as the primary hosting platform due to its seamless GitHub integration and ease of deployment.
- Ensured compatibility with **Sanity CMS** for managing product data.
- Finalized API connections for product listings, authentication, and cart functionalities.

Step 2: Environment Variable Configuration

- Created a `.env` file to store sensitive data securely.
- Configured environment variables in Vercel to avoid exposing credentials in the codebase.

Step 3: Staging Environment Setup

- Deployed the application to a staging environment to replicate a production-like setting.
- Validated successful deployment and identified potential issues.

Step 4: Staging Environment Testing

- Conducted **functional testing** using Cypress and Postman.
- Ran **performance tests** using Lighthouse and GTmetrix.
- Ensured **security compliance** by validating HTTPS implementation and secure API communications.

2. Challenges Faced and Solutions Implemented

Challenge 1: Environment Variable Issues

- **Problem:** `NEXT_PUBLIC_SANITY_DATASET` not being recognized, causing a failure in fetching data.
- **Solution:** Verified `.env` configuration, ensured proper loading of environment variables, and used Vercel's environment settings.

Challenge 2: Dynamic Routing Not Fetching Data from Sanity

- **Problem:** Dynamic routing for product pages (`slug`) was returning `null`.
- **Solution:** Ensured proper schema structure in Sanity, revalidated GROQ queries, and updated client fetch logic.

Challenge 3: Topbar Not Displaying Logged-in User Information

- **Problem:** The top bar did not update with user details after login.
- **Solution:** Implemented `useEffect` to retrieve `localStorage` data and trigger a UI update after authentication.

Challenge 4: Structure Tool Not Working in Sanity Studio

- **Problem:** Encountered `StructureResolver` type mismatch errors.
- **Solution:** Updated `sanity.config.ts` to use the correct `structureTool` import and verified package versions.

Challenge 5: Cart Functionality Not Updating Correctly

- **Problem:** Items were not persisting in the cart.
- **Solution:** Implemented `localStorage` for cart persistence and ensured state updates triggered UI re-renders.

3. Best Practices Followed During Development

Security Best Practices

- Used `.env` files for sensitive credentials and stored them securely in hosting configurations.
- Ensured API calls were **protected** with authentication and HTTPS.
- Validated input fields to prevent **SQL injection** and **XSS attacks**.

Performance Optimization

- Optimized images using **Next.js Image component**.
- Minimized API calls by implementing **server-side caching** where possible.
- Used **lazy loading** for components and assets to enhance page speed.

Code Maintainability and Readability

- Followed **component-based architecture** for reusability.
- Used **TypeScript** for type safety and better code clarity.
- Organized the repository with a structured folder hierarchy (`components/`, `pages/`, `public/`, `sanity/`, etc.).

Testing and Deployment Standards

- Conducted rigorous **staging environment testing** before final deployment.
- Used **GitHub repository management** to track changes and document progress.
- Maintained a **detailed README.md** file summarizing the project structure and deployment process.

Conclusion

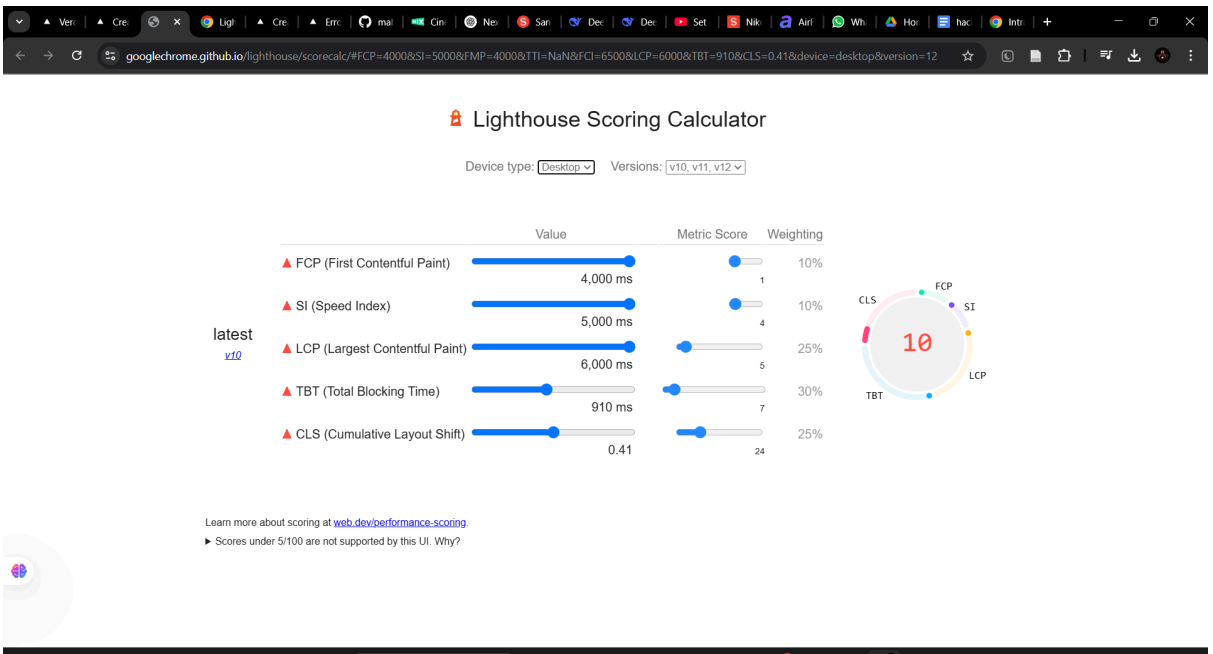
This project successfully implemented a **scalable** and **secure** deployment workflow, integrating **Sanity CMS**, **Next.js**, and **Vercel**. Despite challenges in **environment variables**, **routing**, and **state management**, proper debugging and best practices ensured a smooth deployment.

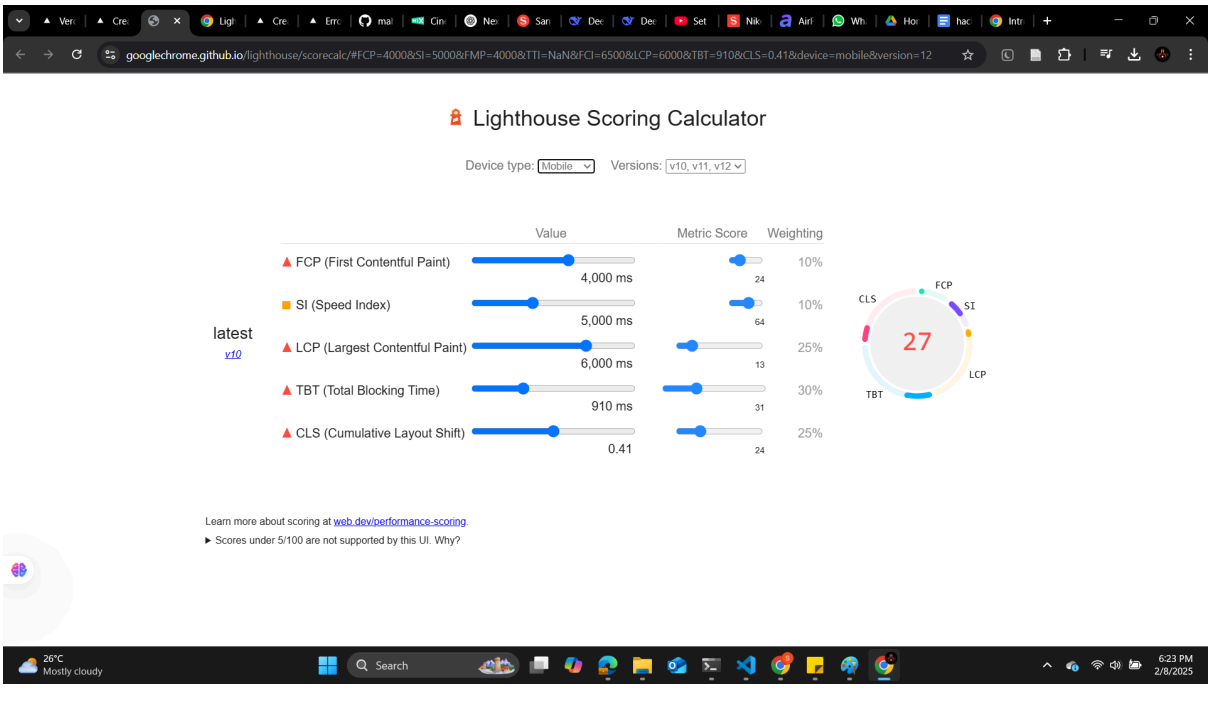
Moving forward, further enhancements can include **automated testing**, **CI/CD pipelines**, and **real-time database updates** to enhance scalability and performance.

2. Test case reporting:

A	B	C	D	E	F	G
Test Case ID	Description	Steps	Expected Result	Actual Result	Status	Remarks
TC001	Validate product listing	Open product page > Verify products	Products displayed	Products displayed	Passed	No issues found
TC002	Test API error handling	Disconnect API > Refresh page	Show fallback message	Fallback message shown	Passed	Handled gracefully
TC003	Check cart functionality	Add item to cart > Verify cart	Cart updates correctly	Cart updates correctly	Passed	Works as expected
TC004	Test responsiveness of layout	Resize browser window > Check layout	Layout adjusts properly	Layout adjusts properly	Passed	Responsive

3. Performance Report


googlechrome.github.io/lighthouse/scorecalc/#FCP=4000&SI=5000&FMP=4000&TTI=NaN&FCI=6500&LCP=6000&TBT=910&CLS=0.41&device=desktop&version=12
Lighthouse Scoring Calculator
Device type: Desktop
Versions: v10, v11, v12
Value
Metric Score
Weighting
FCP (First Contentful Paint)
4,000 ms
1
10%
SI (Speed Index)
5,000 ms
4
10%
LCP (Largest Contentful Paint)
6,000 ms
5
25%
TBT (Total Blocking Time)
910 ms
7
30%
CLS (Cumulative Layout Shift)
0.41
24
25%
latest
v10
FCP
SI
LCP
TBT
CLS
10
Learn more about scoring at web.dev/performance-scoring.
▶ Scores under 5/100 are not supported by this UI. Why?
25°C
Haze
Search
6:24 PM
2/8/2025


googlechrome.github.io/lighthouse/scorecalc/#FCP=4000&SI=5000&FMP=4000&TTI=NaN&FCI=6500&LCP=6000&TBT=910&CLS=0.41&device=mobile&version=12
Lighthouse Scoring Calculator
Device type: Mobile
Versions: v10, v11, v12
Value
Metric Score
Weighting
FCP (First Contentful Paint)
4,000 ms
24
10%
SI (Speed Index)
5,000 ms
64
10%
LCP (Largest Contentful Paint)
6,000 ms
13
25%
TBT (Total Blocking Time)
910 ms
31
30%
CLS (Cumulative Layout Shift)
0.41
24
25%
latest
v10
FCP
SI
LCP
TBT
CLS
27
Learn more about scoring at web.dev/performance-scoring.
▶ Scores under 5/100 are not supported by this UI. Why?
26°C
Mostly cloudy
Search
6:23 PM
2/8/2025