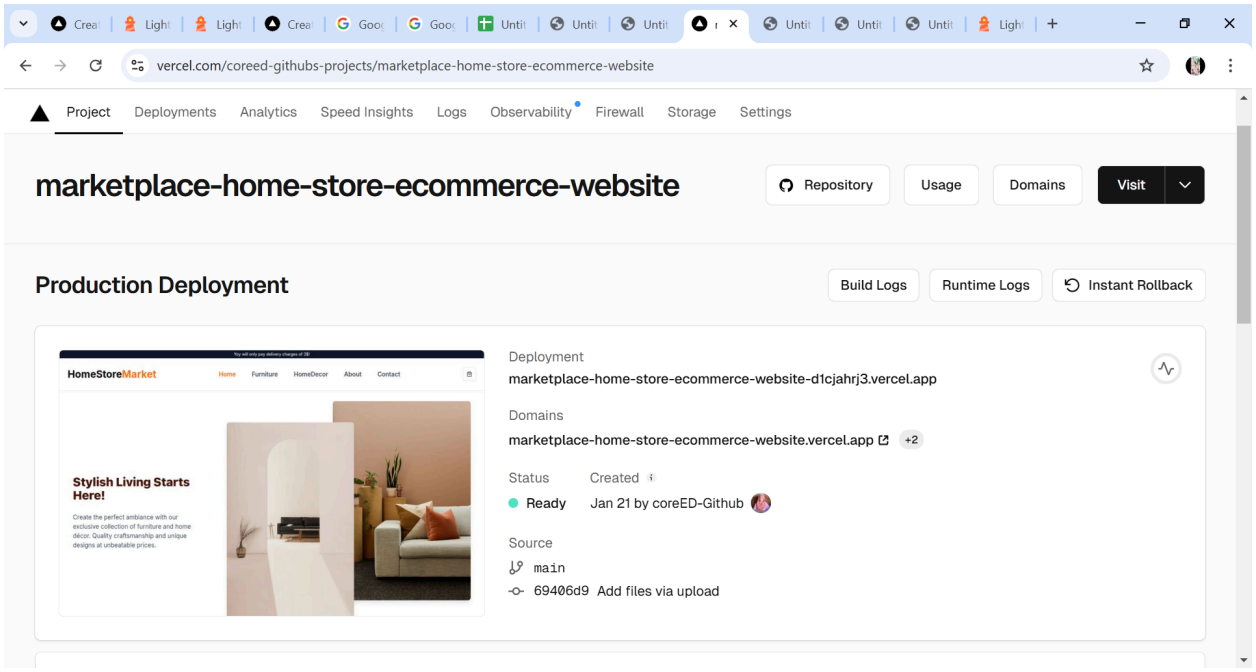
Day 6 - Deployment Preparation and Staging
Environment Setup

For Day 6, I will focus on preparing my marketplace for deployment. I'll be setting up a staging environment and configuring hosting platforms to ensure that everything is ready for a customer-facing application. After building on the testing and optimization work from Day 5, I will emphasize ensuring my marketplace works seamlessly in a production-like environment. Additionally, I will be learning about industry-standard practices for managing different environments like non-production (TRN, DEV, SIT) and production (UAT, PROD, DR).

Step 1: Hosting Platform Setup

1. Choose a Hosting Platform:

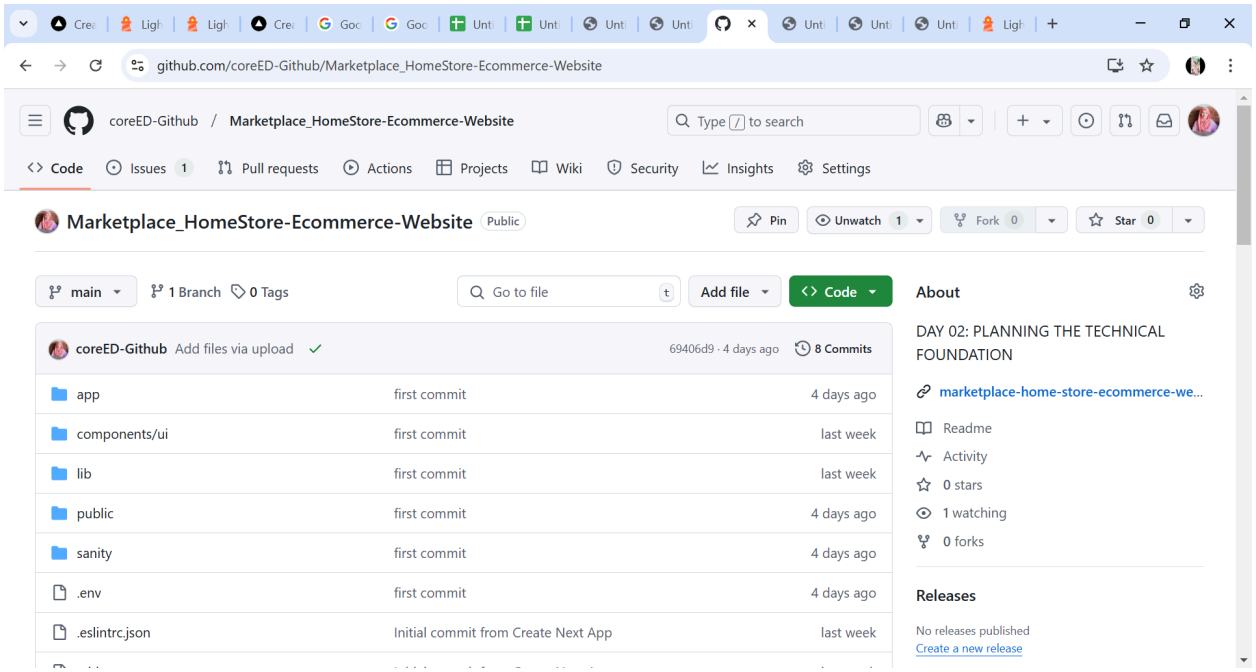
I selected Vercel for quick deployment, as it integrates seamlessly with Next.js projects. For more advanced configurations, platforms like AWS or Azure could be considered, but Vercel met my needs for this project.



2. Connect Repository:

I linked my GitHub repository to the chosen hosting platform (Vercel).

Configured the build settings to ensure proper deployment. This included specifying the necessary deployment scripts to build and deploy the project automatically whenever changes are pushed to the repository.



Step 2: Configure Environment Variables

1. Create a .env File:

I have created a .env file in the root directory of my project to securely store sensitive variables such as API keys and tokens.

The following variables have been added to the .env file:

NEXT_PUBLIC_SANITY_PROJECT_ID=your_project_id

NEXT_PUBLIC_SANITY_DATASET=production

API_KEY=your_api_key

2. Upload Variables to Hosting Platform:

I have uploaded the environment variables to the hosting platform's dashboard, ensuring that the variables are securely stored and accessible during deployment.

Step 03: Deploy to Staging

1. Deploy Application:

First, I deployed my marketplace application to a staging environment using my chosen hosting platform. This allowed me to test the app in a production-like setup, ensuring it was ready for real-world scenarios before going live.

2. Validate Deployment:

After deploying, I ensured the build process completed without any errors. I reviewed the build logs and confirmed everything was processed smoothly.

Then, I verified the basic functionality in the staging environment by checking key features like product listings, navigation, cart functionality, and user interactions to ensure they worked as expected

Step 4: Staging Environment Testing

I have completed the following tasks for testing my marketplace:

1. Testing Types:

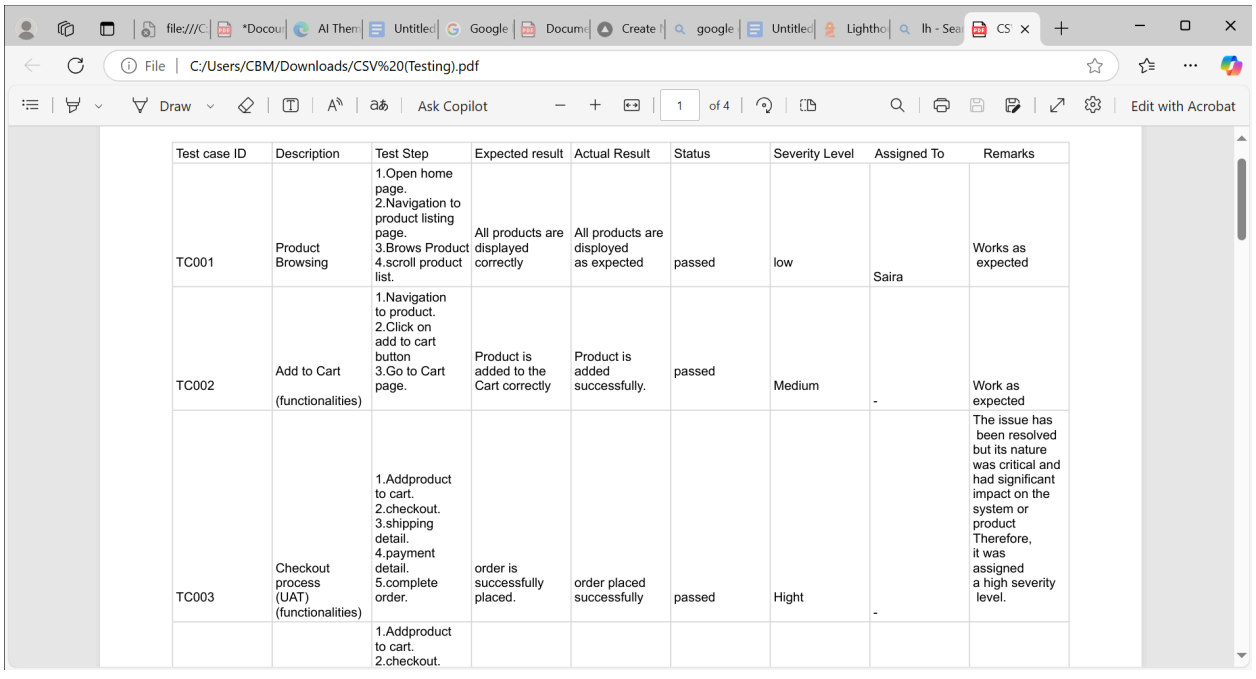
Functional Testing: I have verified all core features of the marketplace, including product listing, search functionality, and cart operations to ensure they work as expected.

Performance Testing: I used Lighthouse and GTmetrix to analyze the speed and responsiveness of the marketplace. The performance was assessed based on various metrics like FCP, LCP, TBT, and CLS.

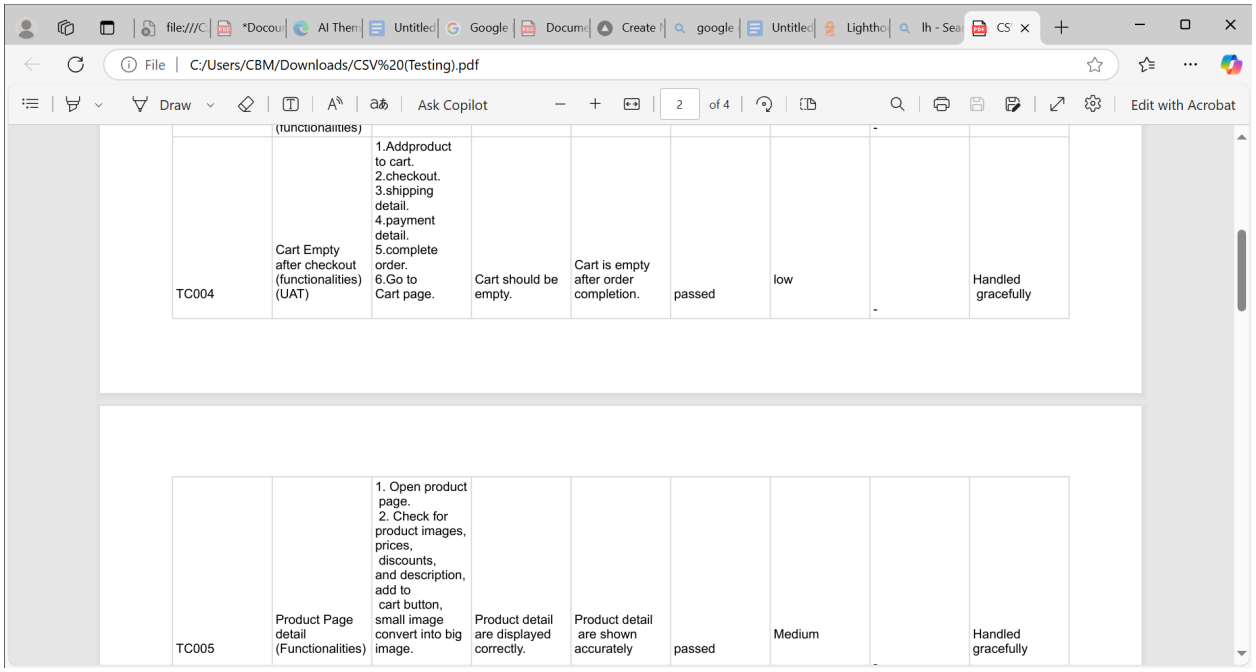
Security Testing: I validated input fields, ensured the usage of HTTPS for secure connections, and checked the security of API communications to safeguard user data and interactions.

2. Test Case Reporting:

I documented all test cases in a CSV file with essential details like Test Case ID, Description, Steps, Expected Result, Actual Result, Status, and Remarks to track the testing process effectively.

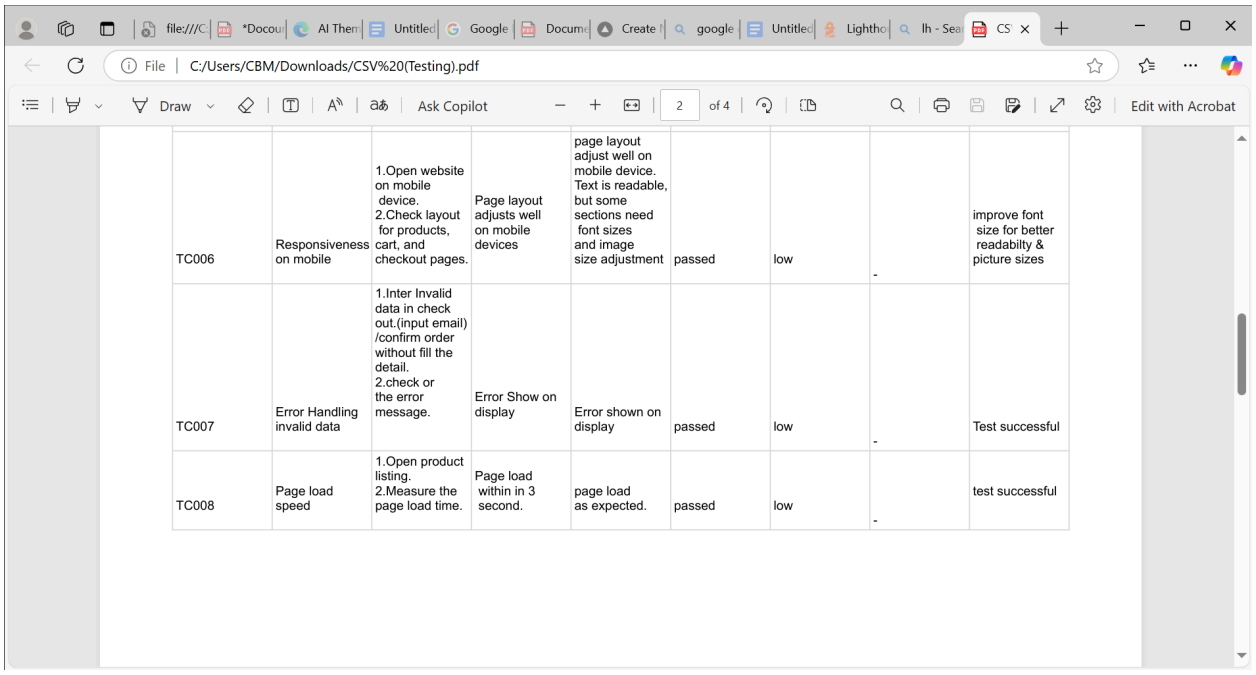


Test case ID	Description	Test Step	Expected result	Actual Result	Status	Severity Level	Assigned To	Remarks
TC001	Product Browsing	1.Open home page. 2.Navigation to product listing page. 3.Brows Product 4.scroll product list.	All products are displayed correctly	All products are displayed as expected	passed	low	Saira	Works as expected
TC002	Add to Cart (functionalities)	1.Navigation to product. 2.Click on add to cart button 3.Go to Cart page.	Product is added to the Cart correctly	Product is added successfully.	passed	Medium	-	Work as expected
TC003	Checkout process (UAT) (functionalities)	1.Addproduct to cart. 2.checkout. 3.shipping detail. 4.payment detail. 5.complete order.	order is successfully placed.	order placed successfully	passed	Hight	-	The issue has been resolved but its nature was critical and had significant impact on the system or product Therefore, it was assigned a high severity level.
		1.Addproduct to cart. 2.checkout.						

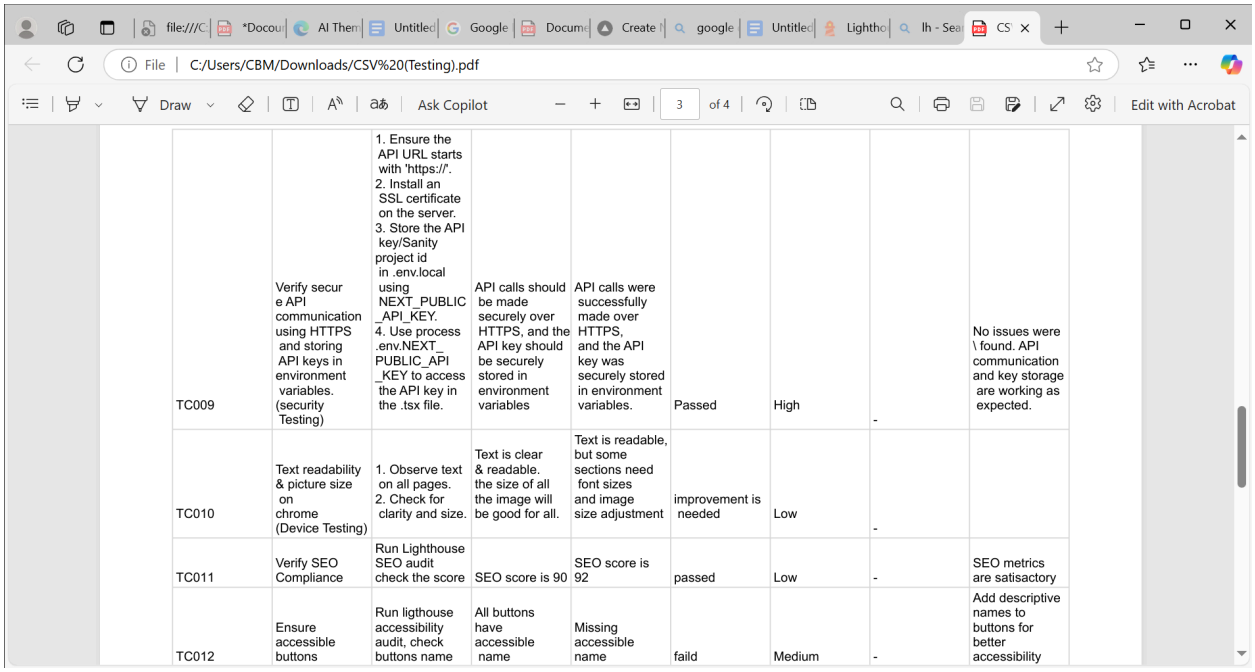


	(functionalities)							
TC004	Cart Empty after checkout (functionalities) (UAT)	1.Addproduct to cart. 2.checkout. 3.shipping detail. 4.payment detail. 5.complete order. 6.Go to Cart page.	Cart should be empty.	Cart is empty after order completion.	passed	low	-	Handled gracefully

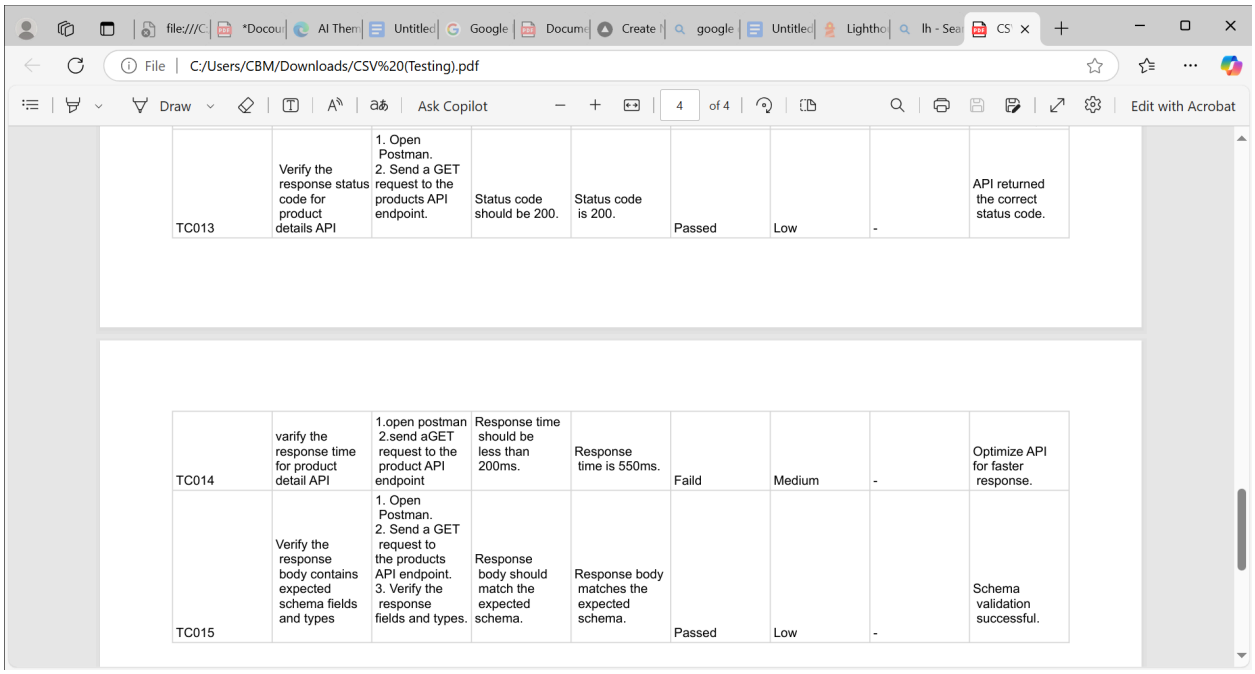
TC005	Product Page detail (Functionalities)	1. Open product page. 2. Check for product images, prices, discounts, and description, add to cart button, small image convert into big image.	Product detail are displayed correctly.	Product detail are shown accurately	passed	Medium	-	Handled gracefully



TC006	Responsiveness on mobile	1.Open website on mobile device. 2.Check layout for products, cart, and checkout pages.	Page layout adjusts well on mobile devices	page layout adjust well on mobile device. Text is readable, but some sections need font sizes and image size adjustment	passed	low	-	improve font size for better readabilty & picture sizes
TC007	Error Handling invalid data	1.Inter Invalid data in check out.(input email) /confirm order without fill the detail. 2.check or the error message.	Error Show on display	Error shown on display	passed	low	-	Test successful
TC008	Page load speed	1.Open product listing. 2.Measure the page load time.	Page load within in 3 second.	page load as expected.	passed	low	-	test successful



TC009	Verify secur e API communication using HTTPS and storing API keys in environment variables. (security Testing)	1. Ensure the API URL starts with 'https://'. 2. Install an SSL certificate on the server. 3. Store the API key/Sanity project id in .env.local using NEXT_PUBLIC _API_KEY. 4. Use process .env.NEXT_ PUBLIC_API _KEY to access the API key in the .tsx file.	API calls should be made securely over HTTPS, and the API key should be securely stored in environment variables	API calls were successfully made over HTTPS, and the API key was securely stored in environment variables.	Passed	High	-	No issues were \ found. API communication and key storage are working as expected.
TC010	Text readability & picture size on chrome (Device Testing)	1. Observe text on all pages. 2. Check for clarity and size.	Text is clear & readable. the size of all the image will be good for all.	Text is readable, but some sections need font sizes and image size adjustment	improvement is needed	Low	-	
TC011	Verify SEO Compliance	Run Lighthouse SEO audit check the score	SEO score is 90	SEO score is 92	passed	Low	-	SEO metrics are satisactory
TC012	Ensure accessible buttons	Run ligthouse accessibility audit, check buttons name	All buttons have accessible name	Missing accessible name	faild	Medium	-	Add descriptive names to buttons for better accessibility



TC013	Verify the response status code for product details API	1. Open Postman. 2. Send a GET request to the products API endpoint.	Status code should be 200.	Status code is 200.	Passed	Low	-	API returned the correct status code.
TC014	varify the response time for product detail API	1.open postman 2.send aGET request to the product API endpoint	Response time should be less than 200ms.	Response time is 550ms.	Faild	Medium	-	Optimize API for faster response.
TC015	Verify the response body contains expected schema fields and types	1. Open Postman. 2. Send a GET request to the products API endpoint. 3. Verify the response fields and types.	Response body should match the expected schema.	Response body matches the expected schema.	Passed	Low	-	Schema validation successful.

3. Performance Testing:

I submitted the performance report generated by tools like Lighthouse in my GitHub repository for transparency and to share the performance analysis results.

On day 6, I successfully completed the following tasks to prepare their marketplace for deployment:

1. Deployed the Staging Environment: They set up a fully functional staging environment for the marketplace.

2. Configured Environment Variables: Environment variables were securely configured to ensure the app runs smoothly in a production-like environment.

3. Test Case and Performance Reports: Saira documented staging tests through comprehensive test case and performance reports.

4. Organized GitHub Repository: All project files and documentation were neatly organized in a GitHub repository for easy access and version control.

5. Professional README.md: They created a professional README.md file that summarizes the project activities and results, ensuring clarity for future reference.