HACKHATHON

Day 05: Testing, Error Handling, and Performance Optimization

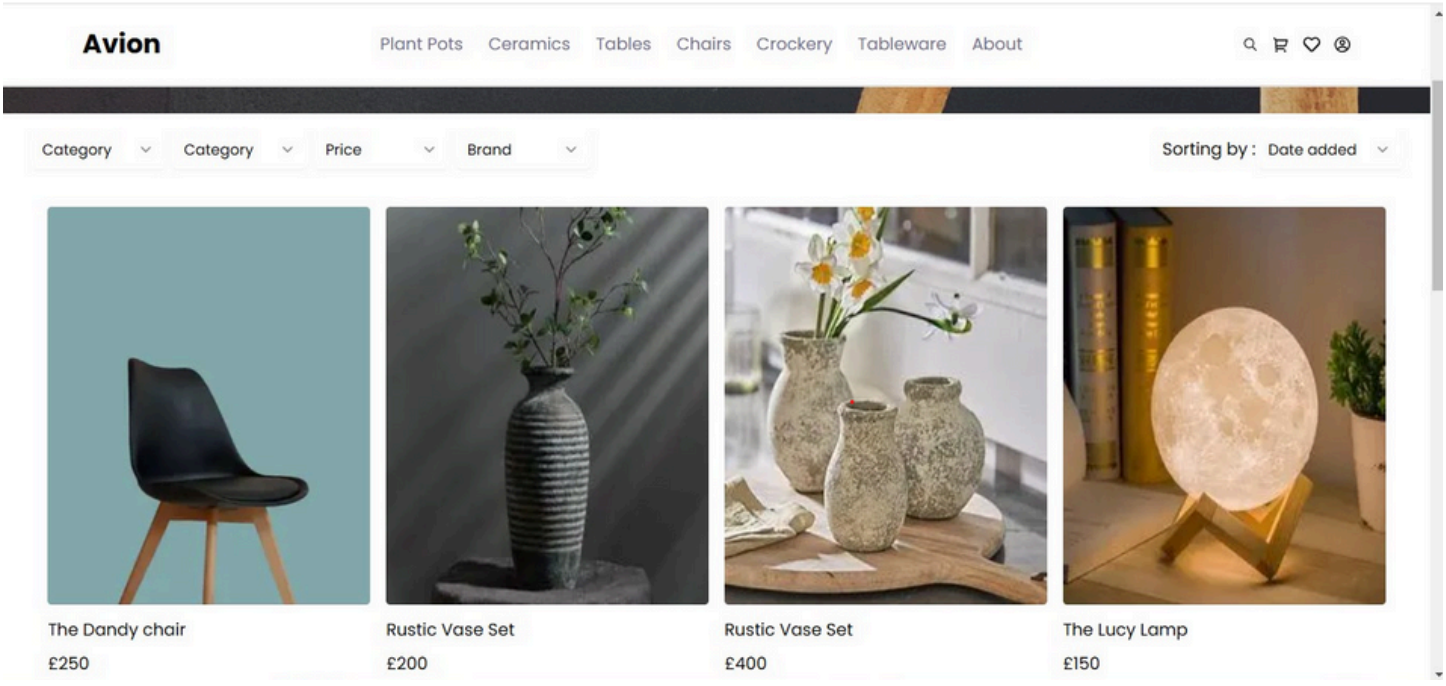
The goal of **Day 05** is to ensure our marketplace is fully functional, error-free, optimized for performance, and ready to handle real-time traffic. The focus is on testing components, handling errors, and improving user experience.

Key Sections to Focus On

1. Functional Testing

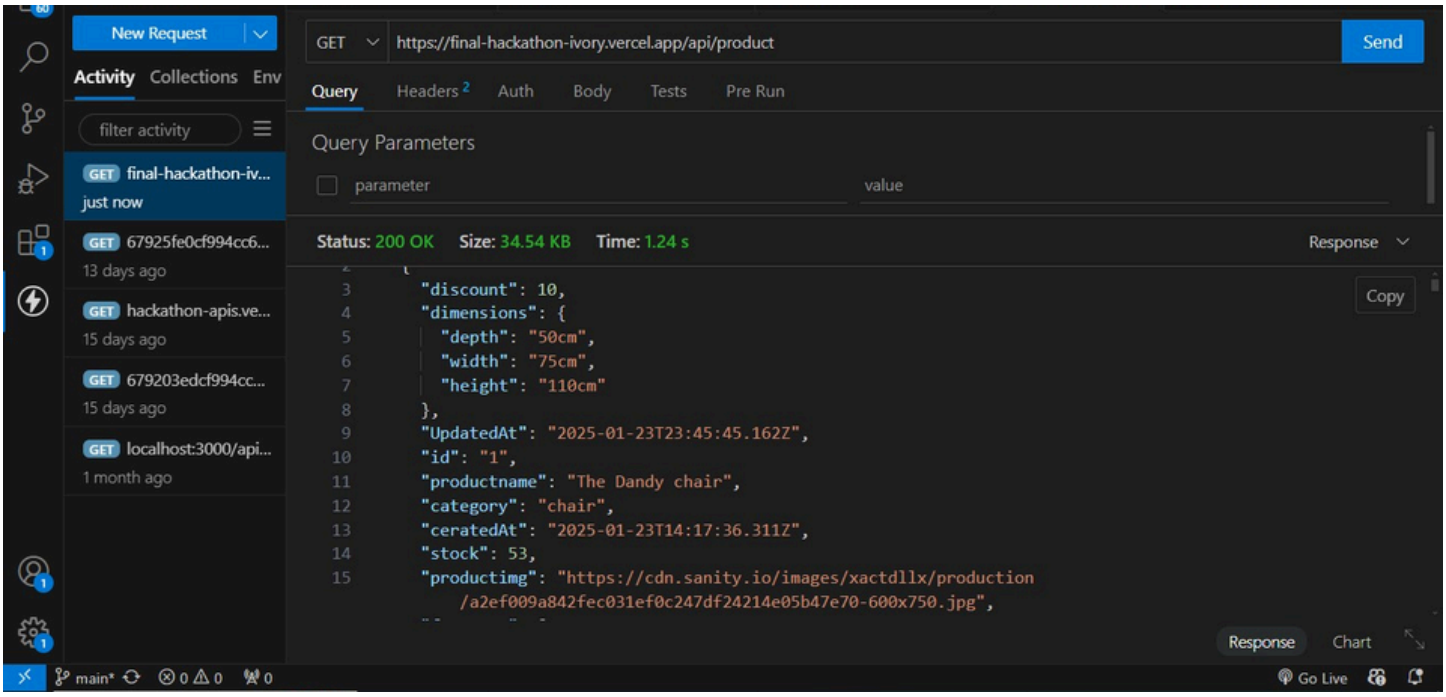
The first and most crucial step in e-commerce development is ensuring that the product page is working perfectly. Since the product page is the backbone of any e-commerce application, it must display products correctly.

Key functionalities tested:
✅ Product listing
✅ Search bar functionality
✅ Filtering by category
✅ Cart operations (add/remove/update items)
✅ Product detail page



2. API Testing

Since all product data comes from an API (Application Programming Interface), we must ensure that the API functions correctly. The API was tested using **Thunder Client** and other testing platforms to verify data retrieval, response times, and error handling.



And that is Pagination , I have also include Pagination.

3. Error Handling

Errors are an inevitable part of development. The key to a great user experience is handling them effectively. Instead of showing raw errors, our system provides user-friendly fallback messages that are easy to understand.

In **Next.js**, error handling is implemented using:

- error.tsx – Displays a custom error page with helpful UI feedback.
- API request error handling – Ensures proper responses when API calls fail

```tsx
'use client'
import ErrorPage from "@/components/ErrorPage/ErrorPage"

function Error({
    error,
    reset,
  }: {
    error: Error & { digest?: string }
    reset: () => void
  }) {
  return (
    <ErrorPage error={error} reset={reset}/>
  )
}

export default Error
```

API Request

```tsx
const {FILTERCATEG,LOADAVION,PRODPAGEONE,PRODPAGETWO,PRODPAGETHREE,PRODPAGEFOUR} = PRODUCTACTION;

const [prodData,dispatch] = useReducer(productReducer,prodInitialData);
function productReducer(state:InitialProdData,action:ProductAction):InitialProdData{···
}
const productApi= async (api:string) =>{
    try {
        const productData =await fetch(api,{cache:'force-cache'})
        const fetchProduct = await productData.json()
    //    setProductList(fetchProduct);
        console.log(fetchProduct)
    return fetchProduct;
    } catch (error) {
        throw new Error(`product not found : ${error}`)
    }

}

//DESTRUCTURE OF PRODUCT DATA
const {productList,limit,page} = prodData;
useEffect(() => {
```

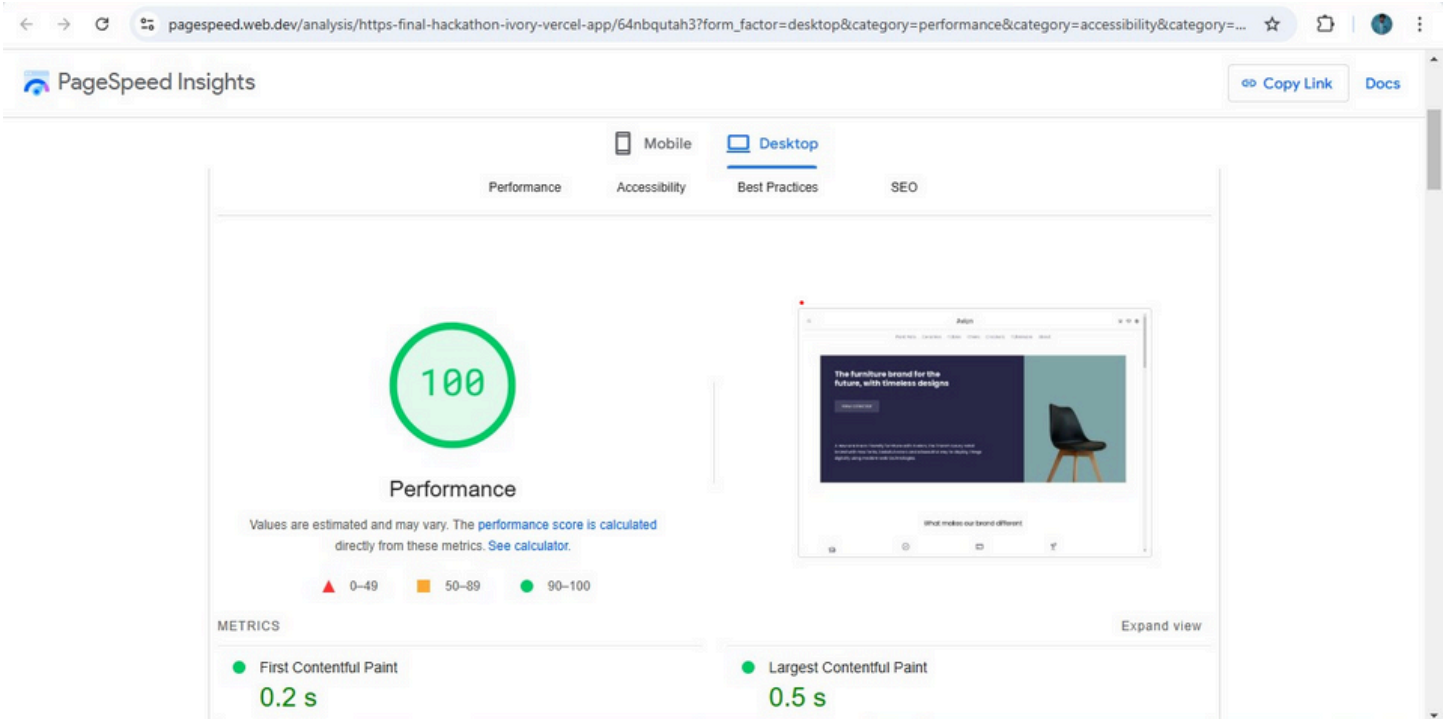
4. Performance Optimization

Performance optimization is essential to ensure a smooth user experience. Think of it as a CEO evaluating an employee's performance—our website must also be analyzed and improved where necessary.

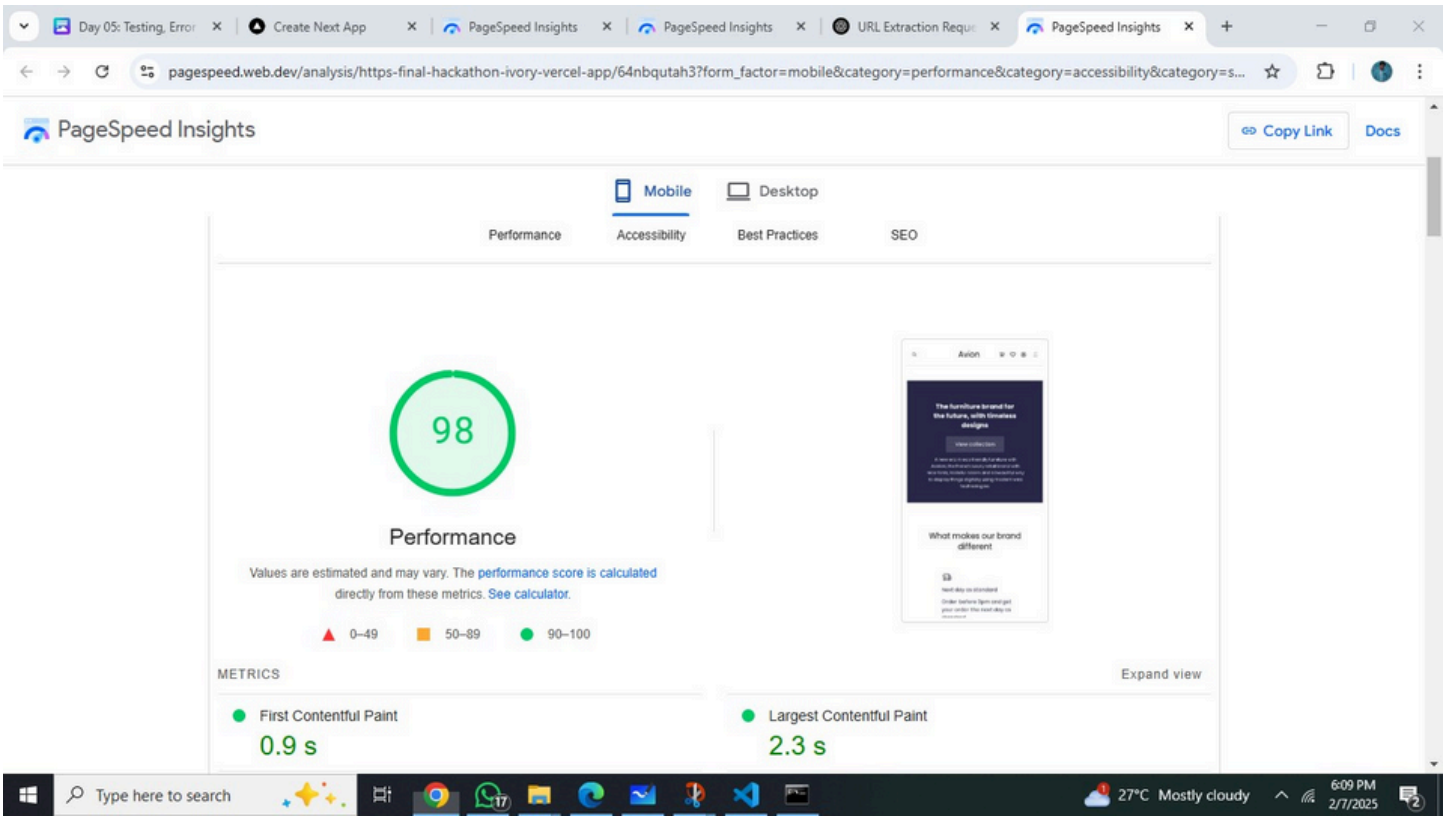
Key performance improvements:
✅ Optimized API calls
✅ Lazy loading for images
✅ Efficient component re-renders
✅ Code splitting and caching

Performance metrics were analyzed for both **desktop and mobile** versions.

Desktop



Mobile



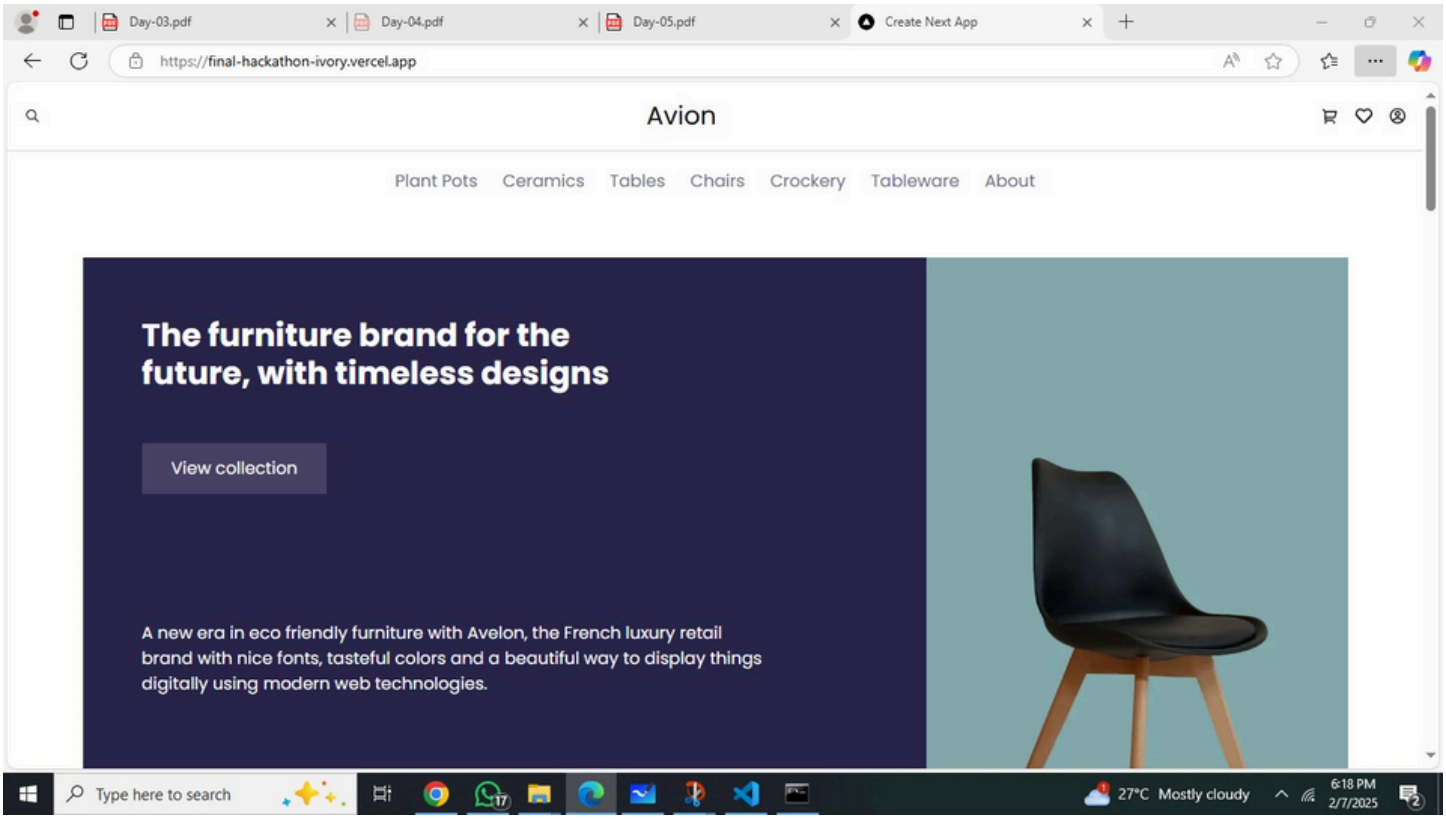
5. Cross-Browser and Device Testing

A good web application must be compatible across different browsers and devices. I tested the marketplace on:

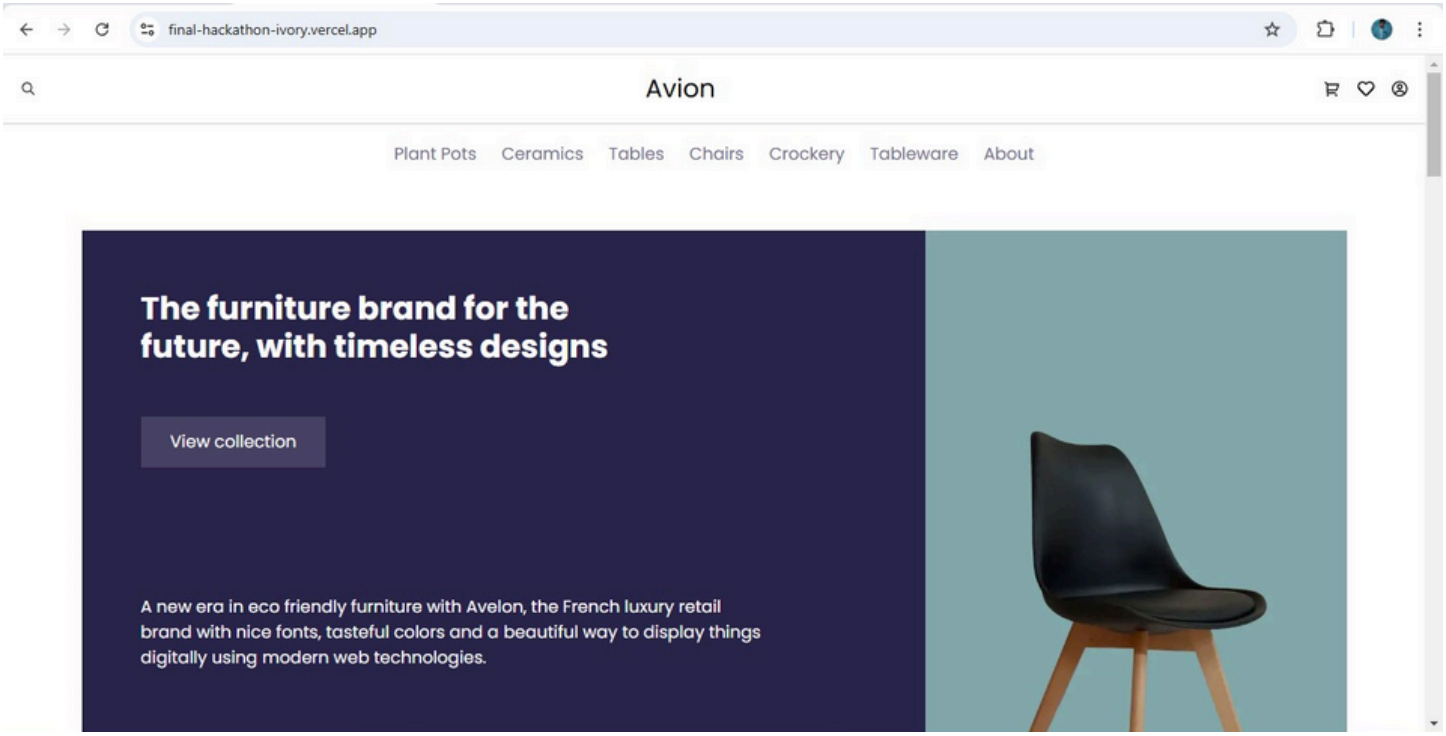
✓ **Browsers** – Chrome, Firefox, Safari, Edge

✓ **Devices** – Desktop, mobile, tablet

MICROSOFT EDGE



GOOGLE CHROME

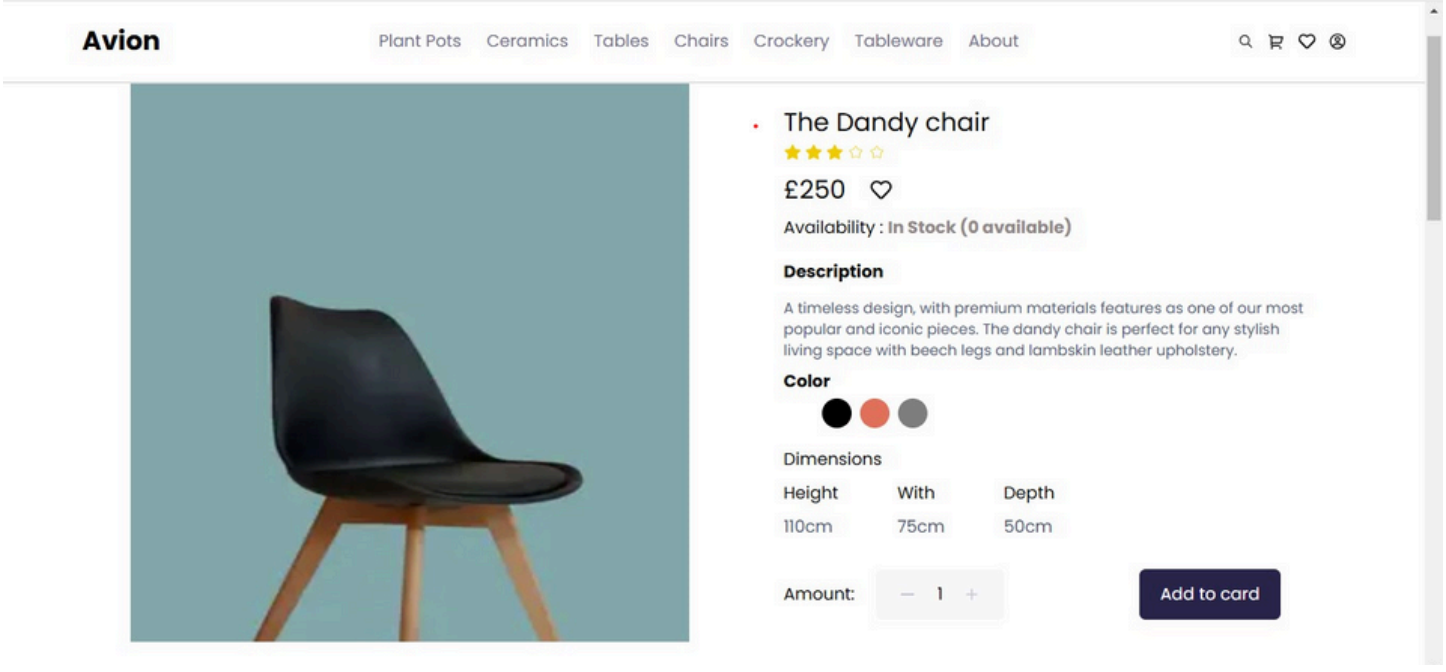


That’s how it’s showing on Microsoft Edge and Chrome , Perfectly.

6. User Acceptance Testing (UAT)

Before launching a product, it's essential to test it thoroughly. The marketplace underwent multiple rounds of testing to ensure all features worked as expected. The goal was to simulate real user interactions and verify that:

✓ The UI is intuitive and responsive
✓ The checkout process works seamlessly
✓ Errors are properly handled
✓ The marketplace functions correctly across devices



7. Testing Report (CSV Documentation)

After testing, I documented the results in a professional **Testing Report (CSV format)**, including details about the tests performed, issues encountered, and their resolutions.

8. Challenges Faced

As a developer, facing challenges is part of the process. Here are some key challenges I encountered:

- 🔹 Implementing search, filtering, and pagination
- 🔹 Handling API errors effectively
- 🔹 Managing responsiveness across devices
- 🔹 Migrating data correctly from the API to Sanity

Each challenge was a learning experience, helping to improve problem-solving skills and debugging strategies.

9. Future Enhancements

A marketplace is always evolving. Some potential future improvements include:

🚀 **Database Integration** – Connecting to a dedicated database for scalability
🌎 **Multi-Language Support** – Enhancing accessibility for global users
📦 **Real-Time Order Tracking** – Providing live updates on order status
💳 **More Payment Options** – Expanding payment gateway support

Conclusion

With rigorous testing, optimization, and error handling, the marketplace is now ready for real-world traffic. This phase was crucial in refining the user experience and ensuring a seamless shopping journey.

Ending this document with the Team Page.

“Ahmed Raza”