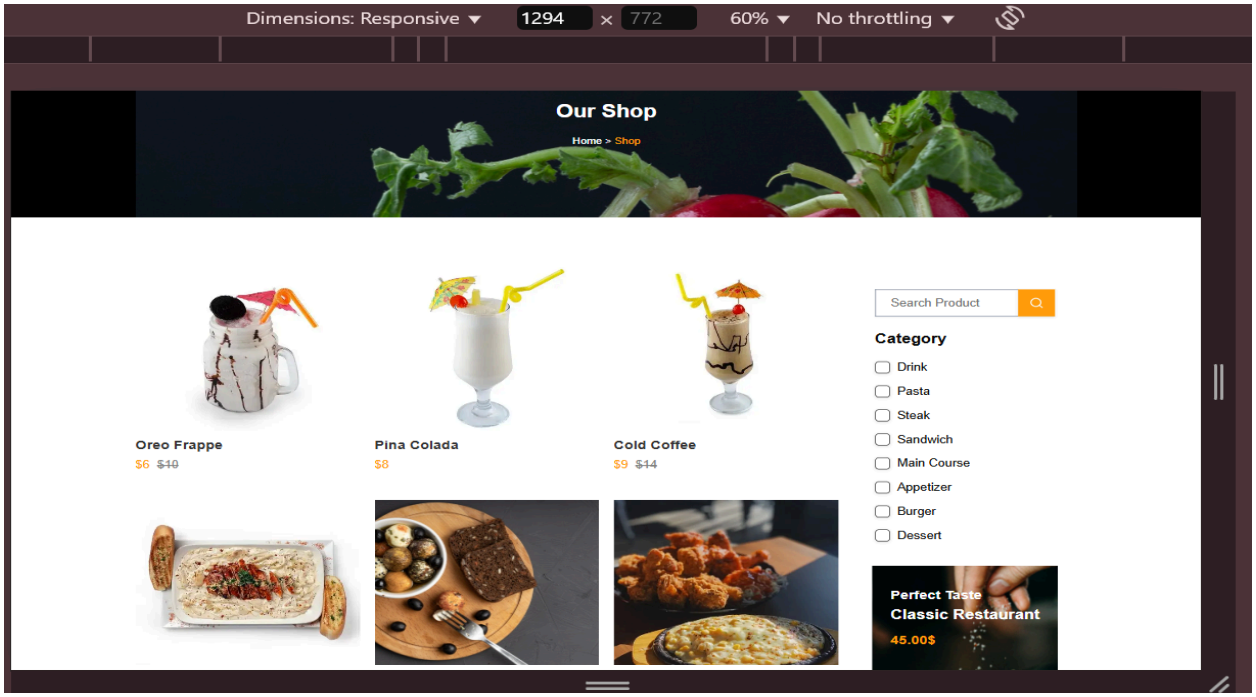
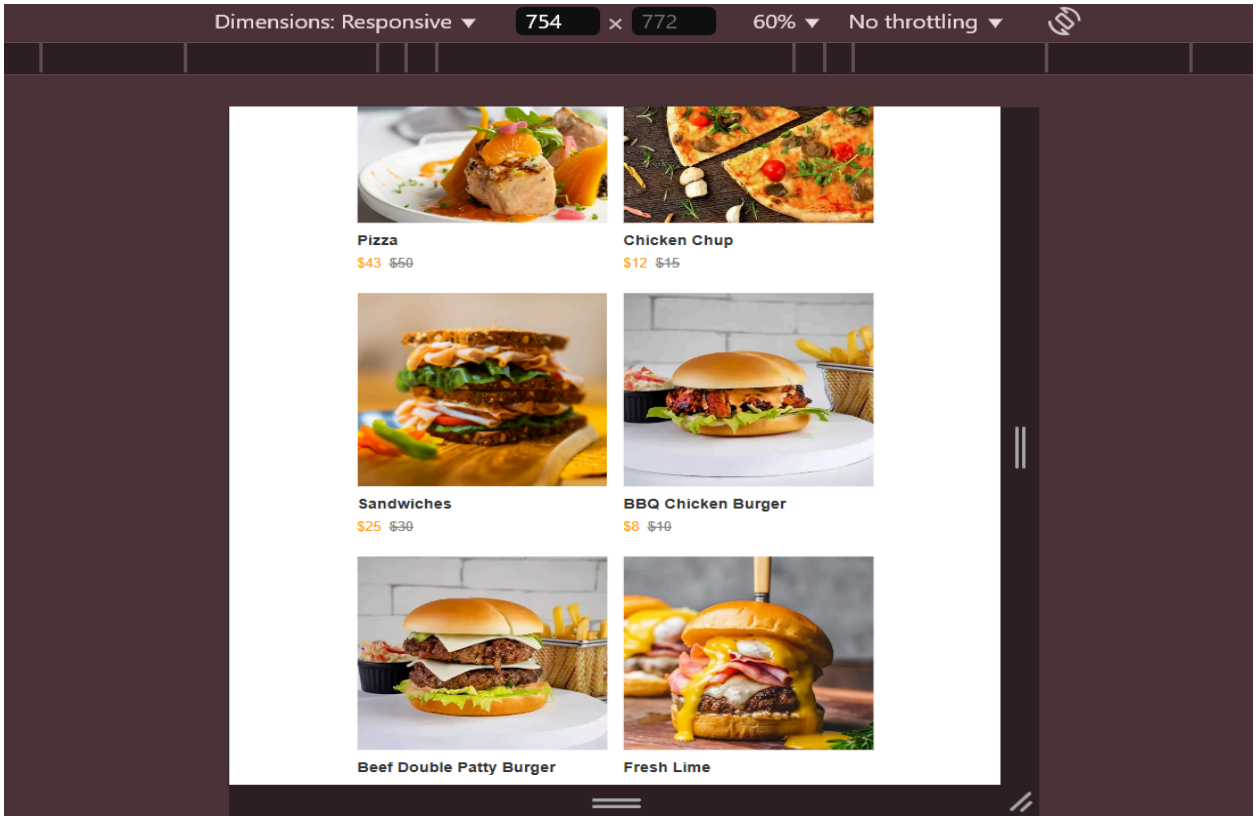
Table of Contents

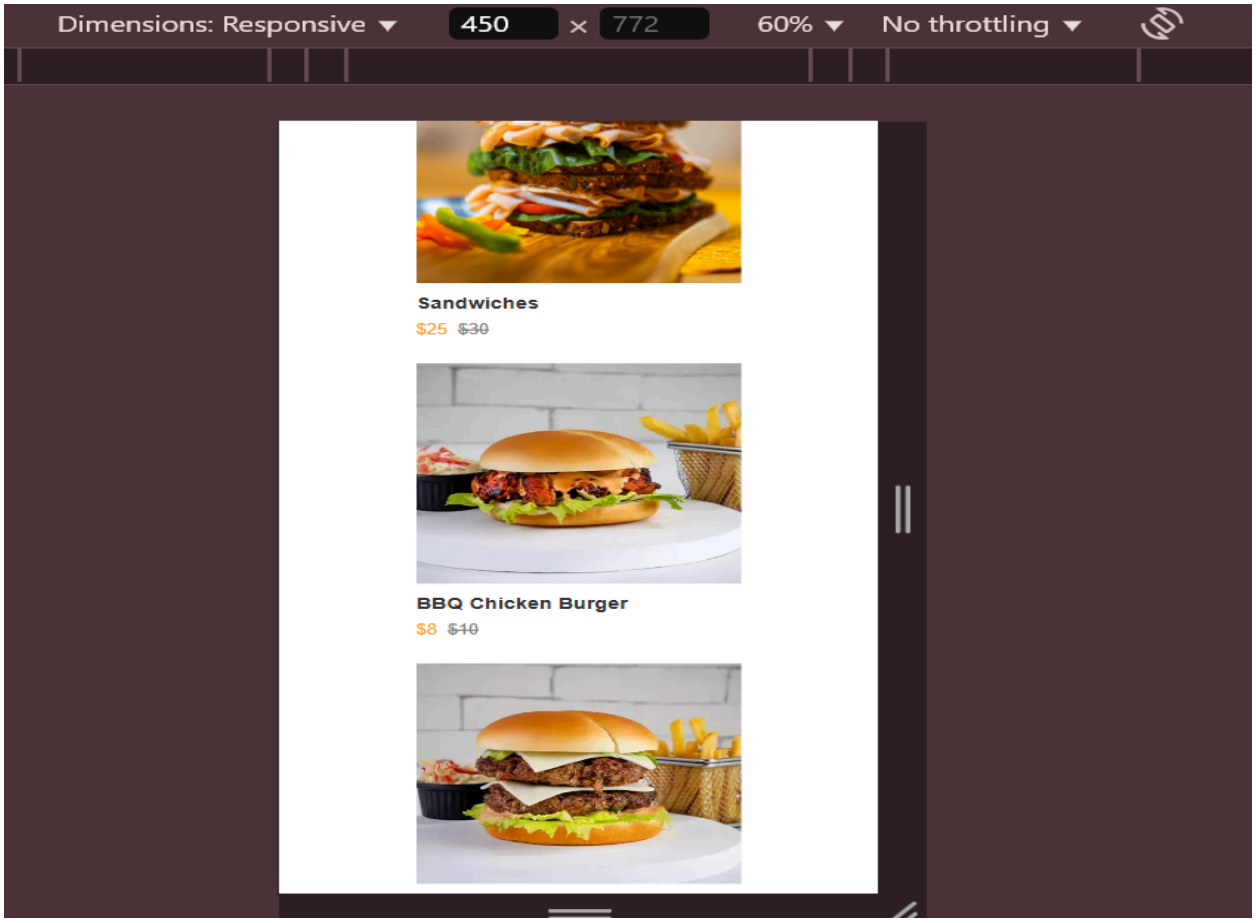
1.0	Without cart functionality screenshots
1.1	With cart functionality screenshots

Functional Testing

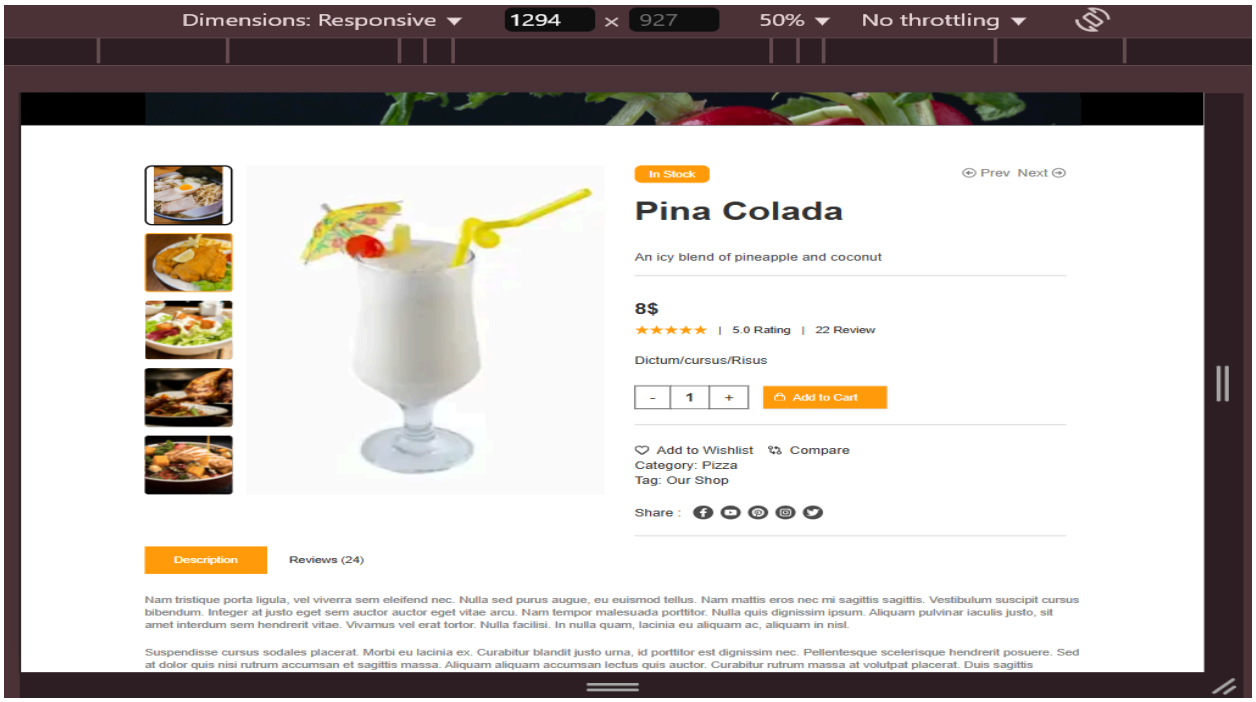
Product Listing Page with Functionality & Responsiveness

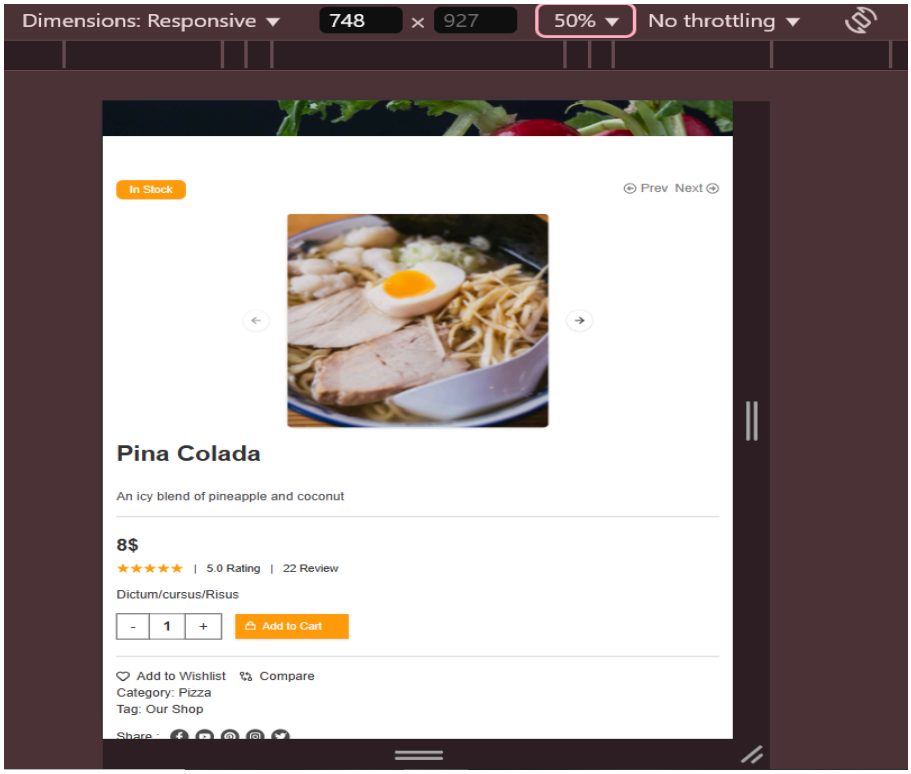


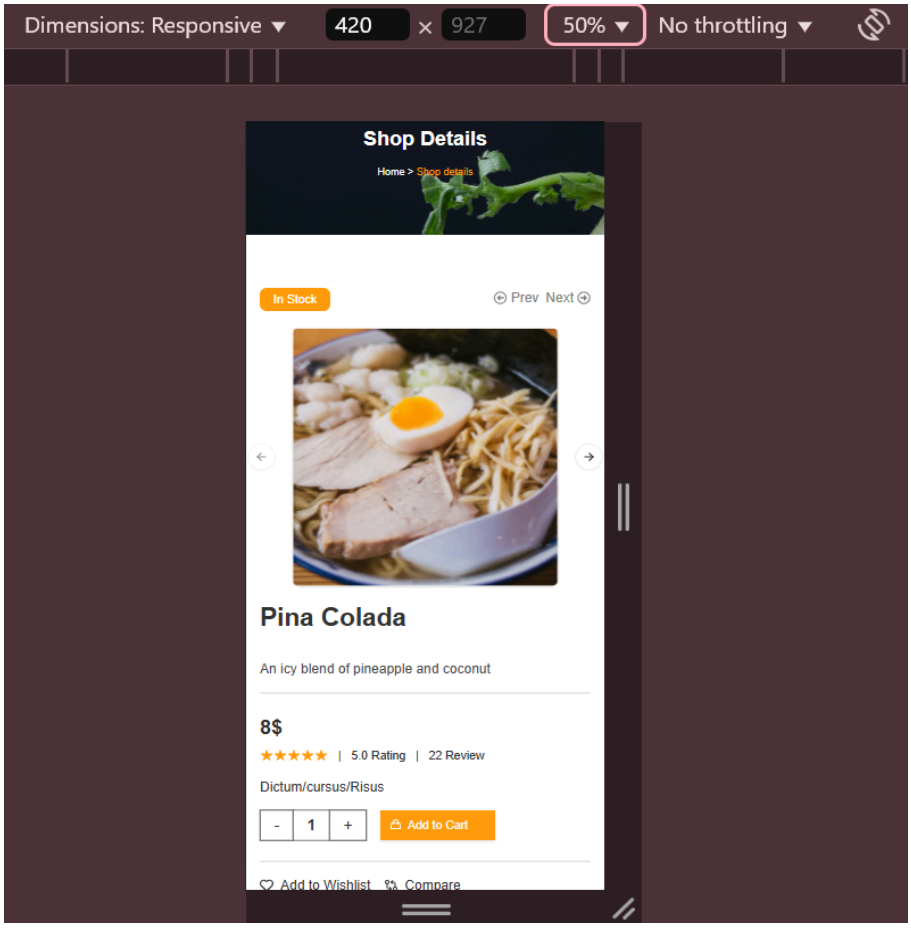

Dimensions: Responsive
754
772
60%
No throttling
Pizza
$43 $50
Chicken Chup
$12 $15
Sandwiches
$25 $30
BBQ Chicken Burger
$8 $10
Beef Double Patty Burger
Fresh Lime



Product Detail Page with Functionality and Responsiveness

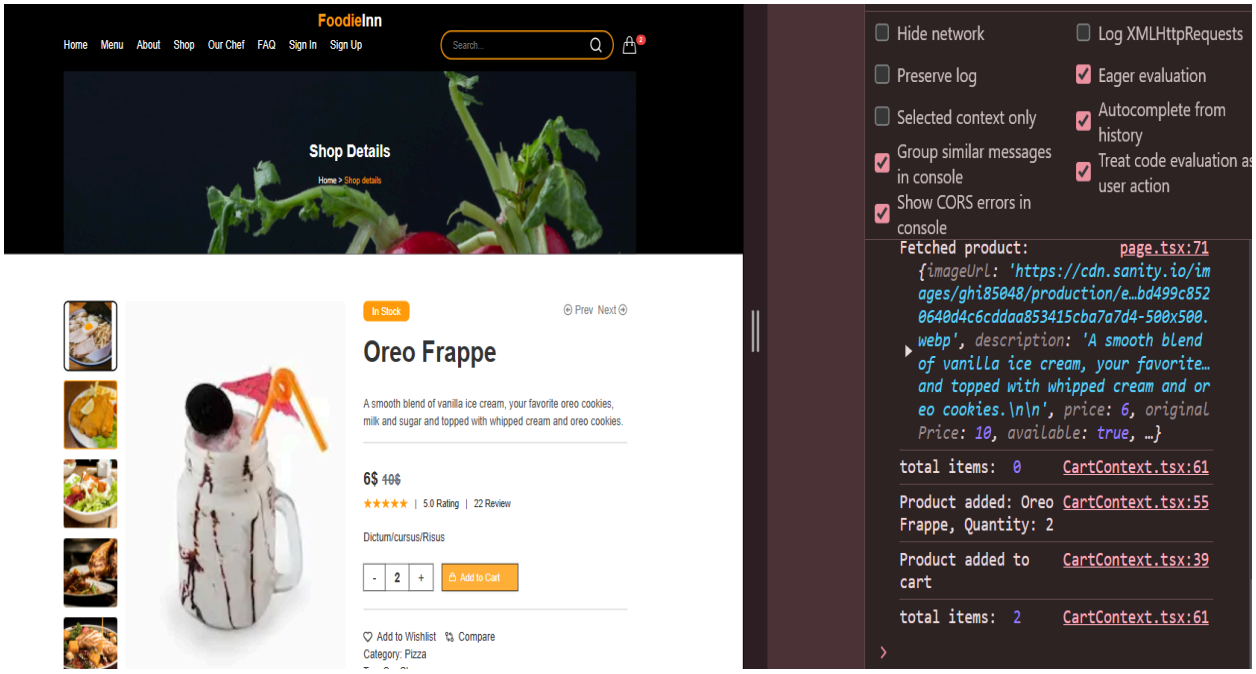



Dimensions: Responsive
748
927
50%
No throttling
In Stock
Prev Next
Pina Colada
An icy blend of pineapple and coconut
8$
5.0 Rating | 22 Review
Dictum/cursus/Risus
1
Add to Cart
Add to Wishlist
Compare
Category: Pizza
Tag: Our Shop
Share

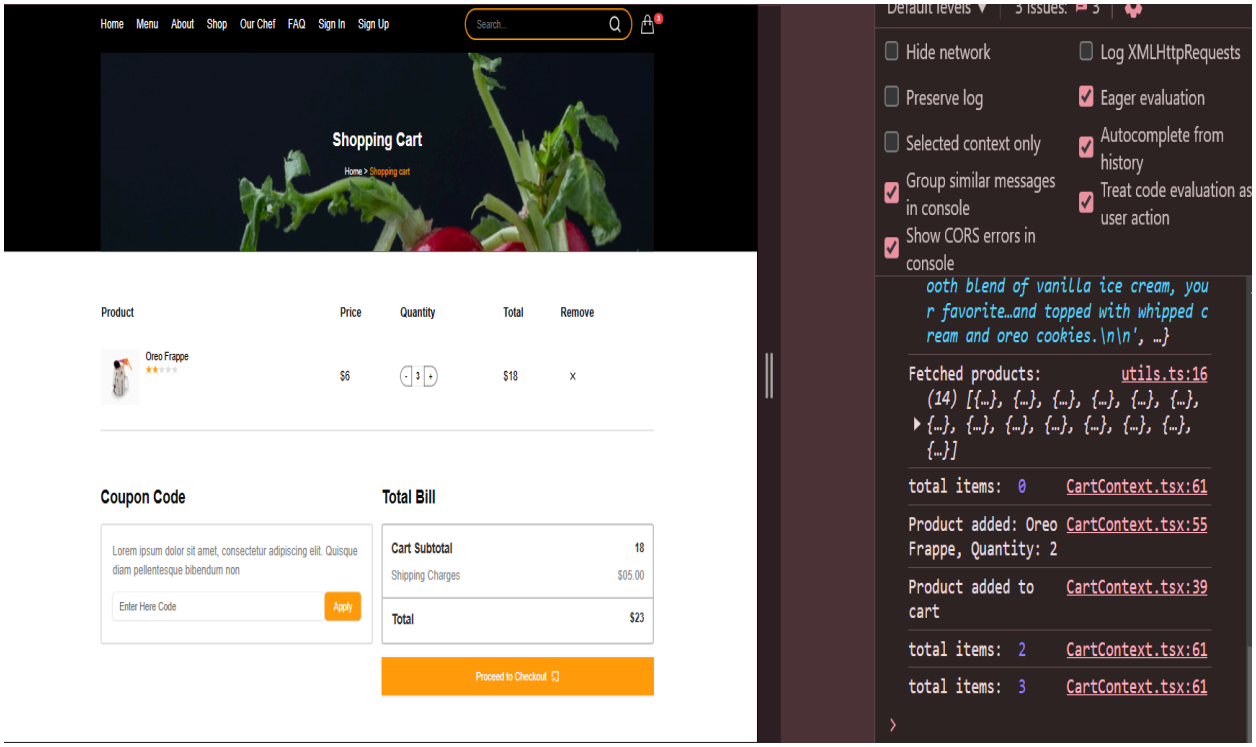

Dimensions: Responsive
420
927
50%
No throttling
Shop Details
Home > Shop details
In Stock
Prev Next
Pina Colada
An icy blend of pineapple and coconut
8$
5.0 Rating | 22 Review
Dictum/cursus/Risus
1
Add to Cart
Add to Wishlist
Compare

Cart Operations

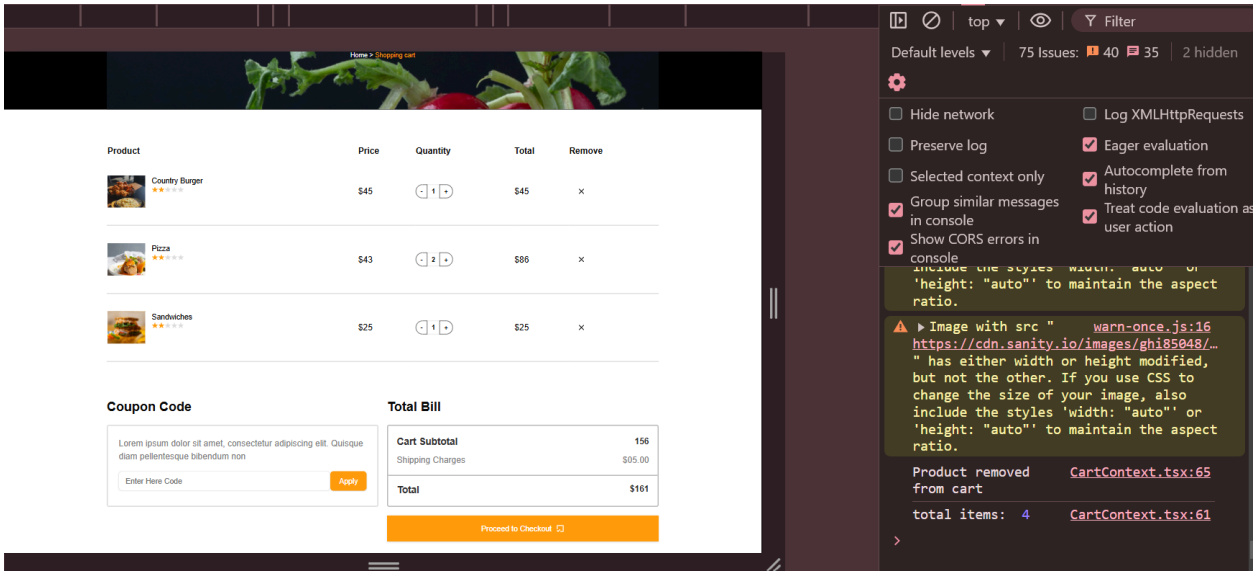
Add



Update Quanity

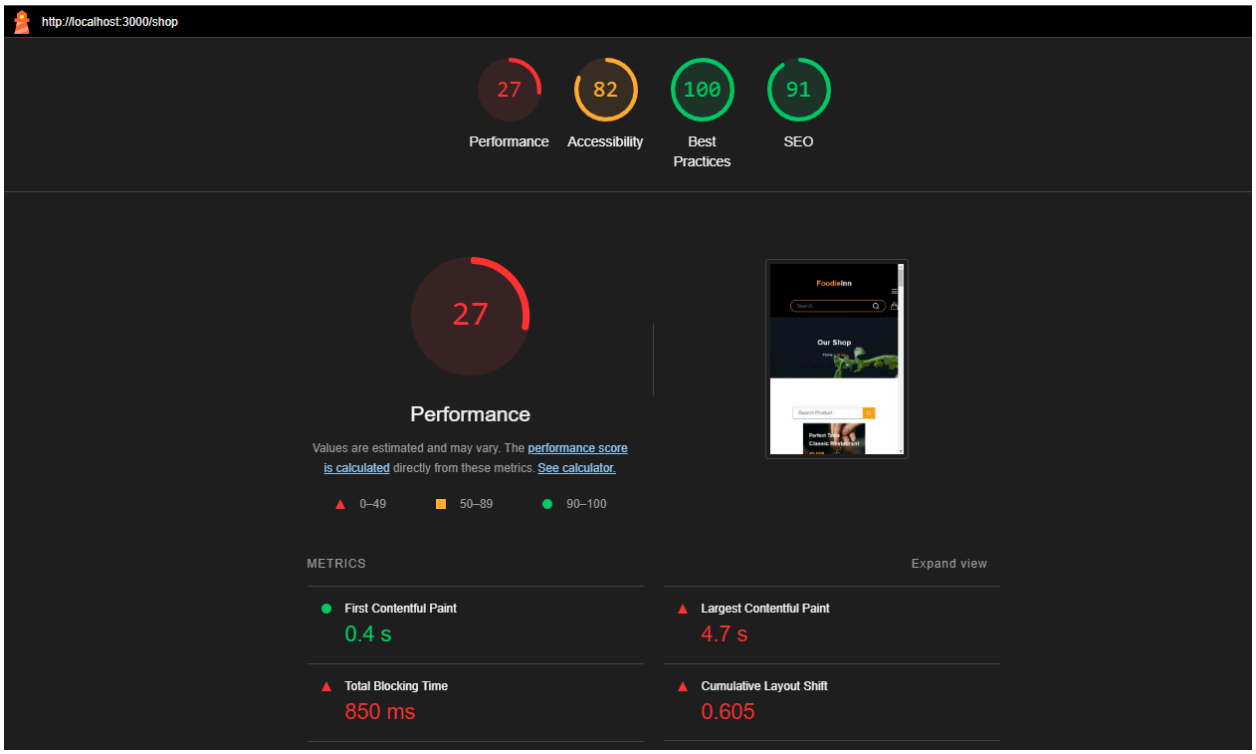


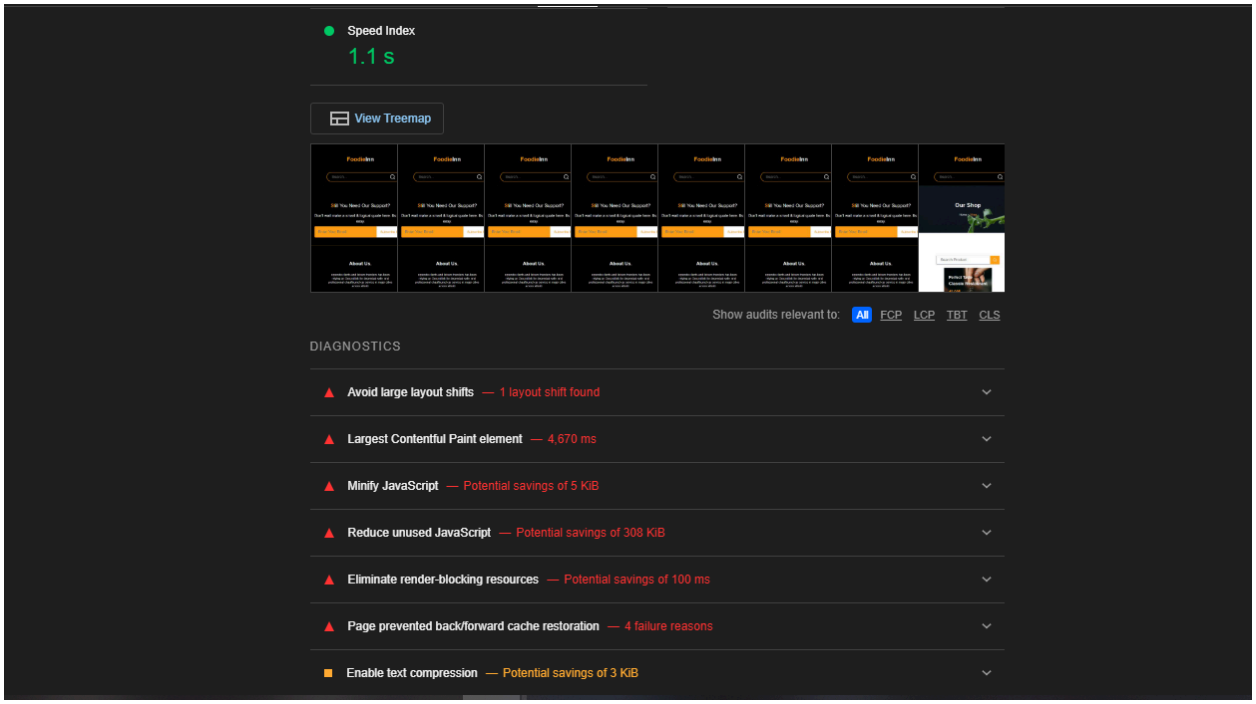
Remove

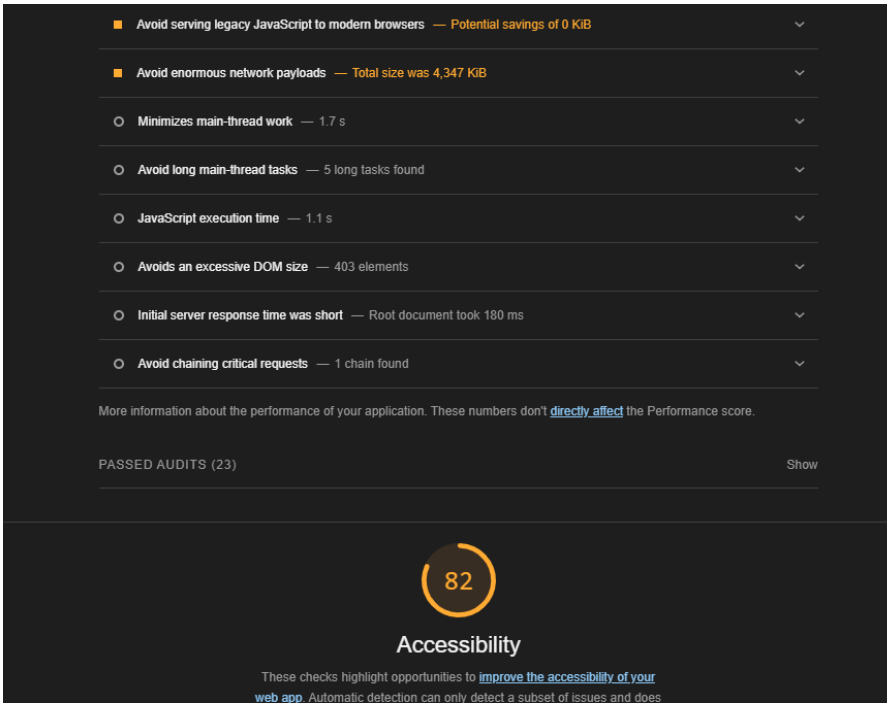


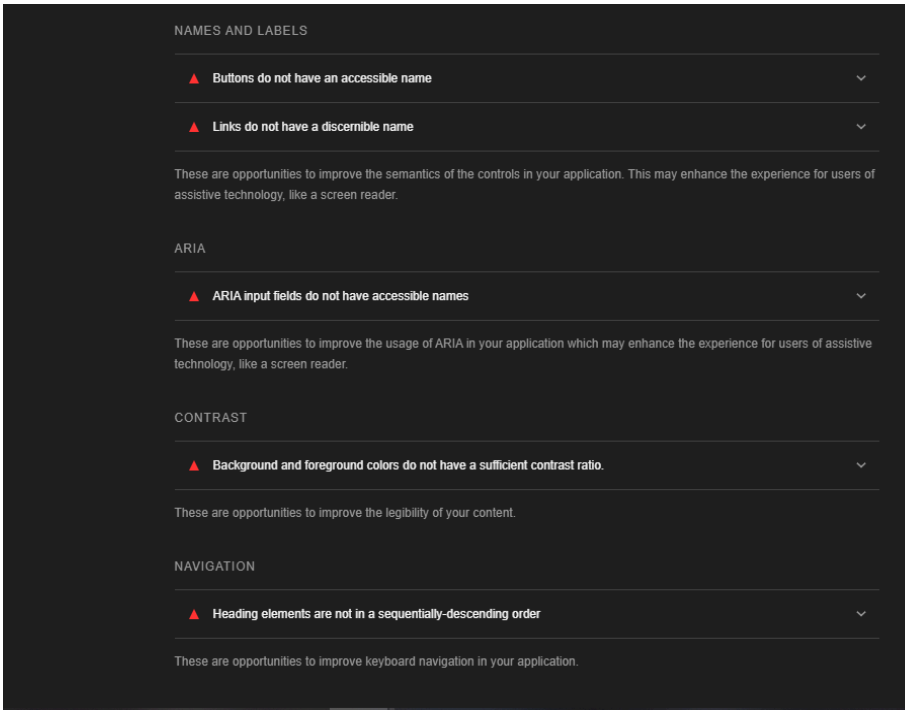
Logs from Testing Tools

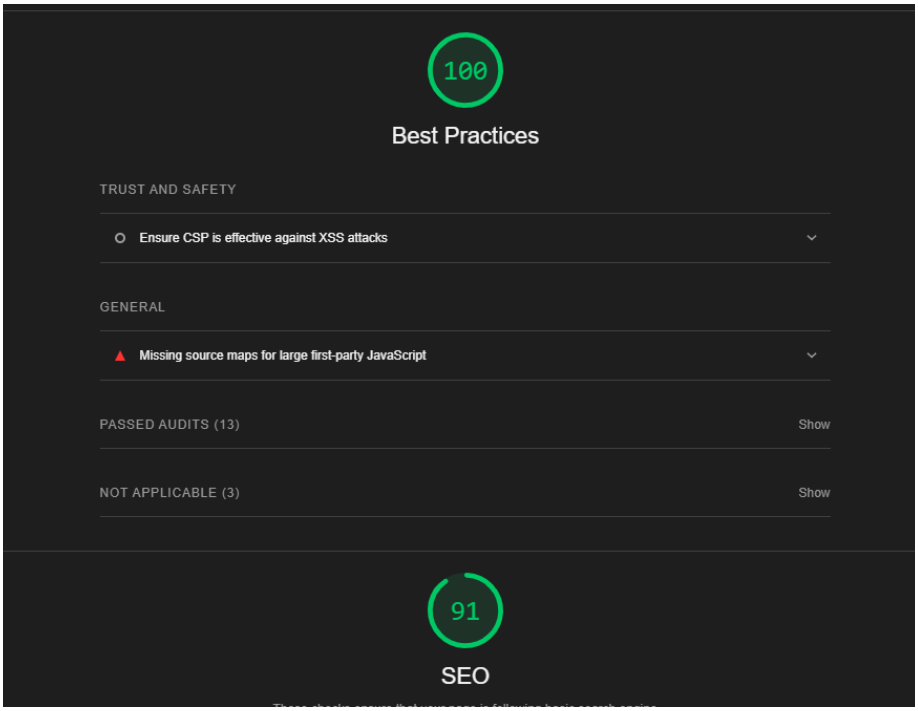
Lighthouse




Speed Index
1.1 s
View Treemap
Show audits relevant to: All FCP LCP TBT CLS
DIAGNOSTICS
Avoid large layout shifts — 1 layout shift found
Largest Contentful Paint element — 4,670 ms
Minify JavaScript — Potential savings of 5 KiB
Reduce unused JavaScript — Potential savings of 308 KiB
Eliminate render-blocking resources — Potential savings of 100 ms
Page prevented back/forward cache restoration — 4 failure reasons
Enable text compression — Potential savings of 3 KiB

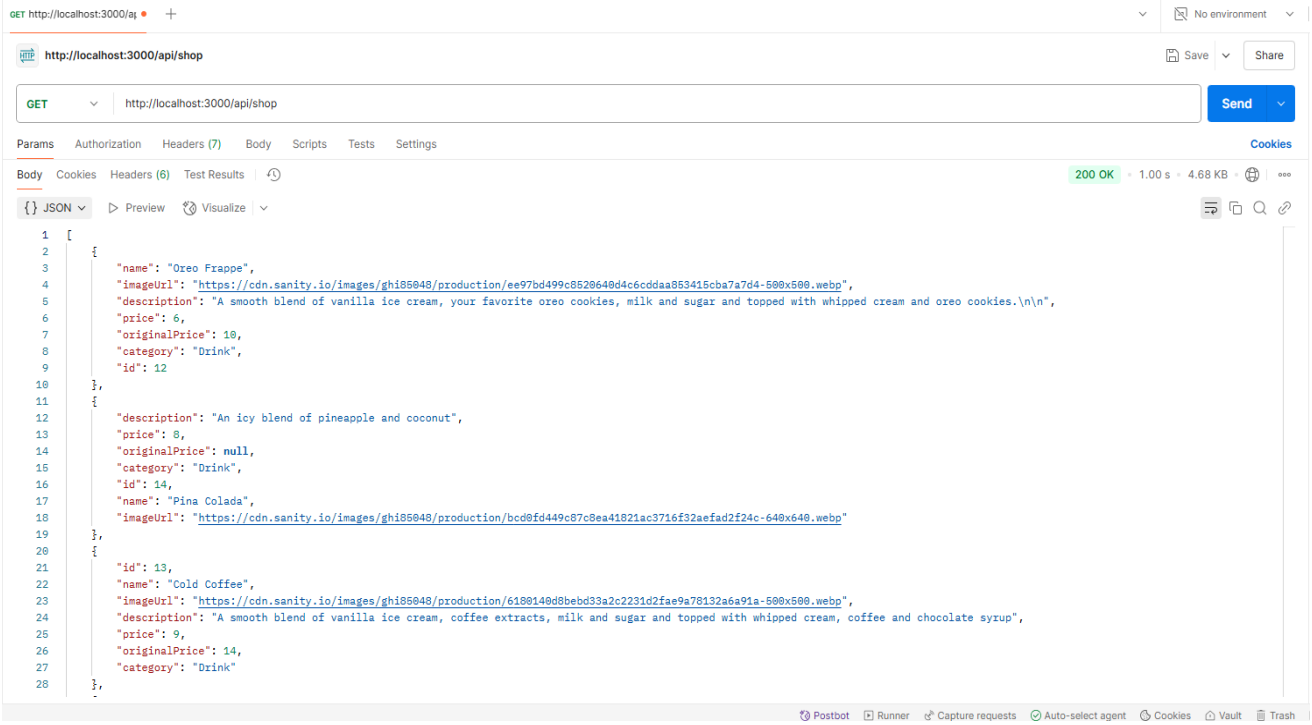

Avoid serving legacy JavaScript to modern browsers — Potential savings of 0 KiB
Avoid enormous network payloads — Total size was 4,347 KiB
Minimizes main-thread work — 1.7 s
Avoid long main-thread tasks — 5 long tasks found
JavaScript execution time — 1.1 s
Avoids an excessive DOM size — 403 elements
Initial server response time was short — Root document took 180 ms
Avoid chaining critical requests — 1 chain found
More information about the performance of your application. These numbers don't directly affect the Performance score.
PASSED AUDITS (23)
Show
82
Accessibility
These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does


NAMES AND LABELS
Buttons do not have an accessible name
Links do not have a discernible name
These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.
ARIA
ARIA input fields do not have accessible names
These are opportunities to improve the usage of ARIA in your application which may enhance the experience for users of assistive technology, like a screen reader.
CONTRAST
Background and foreground colors do not have a sufficient contrast ratio.
These are opportunities to improve the legibility of your content.
NAVIGATION
Heading elements are not in a sequentially-descending order
These are opportunities to improve keyboard navigation in your application.

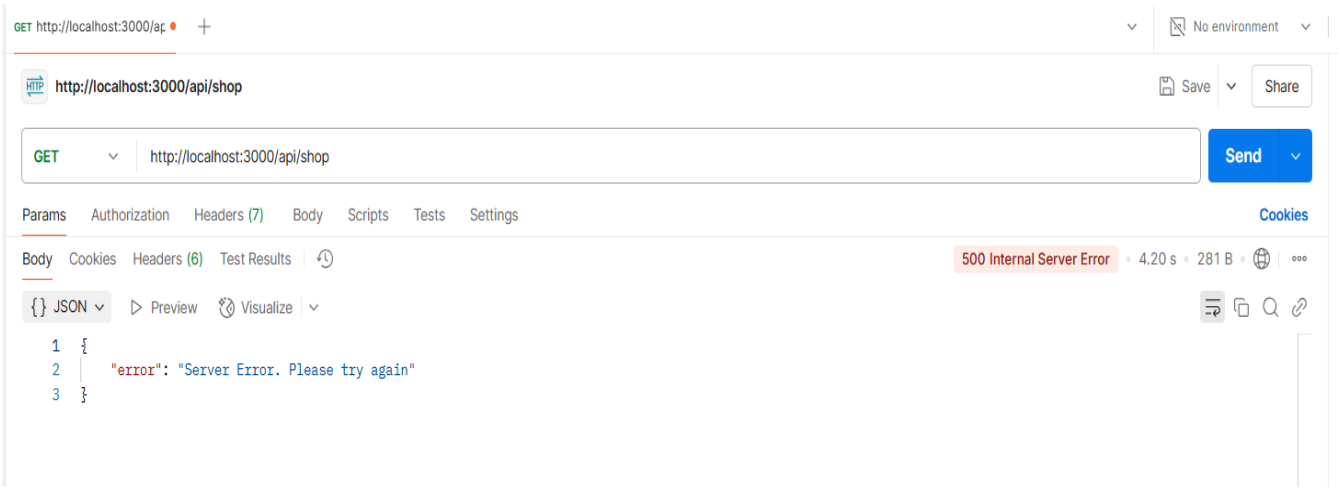

100
Best Practices
TRUST AND SAFETY
Ensure CSP is effective against XSS attacks
GENERAL
Missing source maps for large first-party JavaScript
PASSED AUDITS (13)
Show
NOT APPLICABLE (3)
Show
91
SEO

Postman

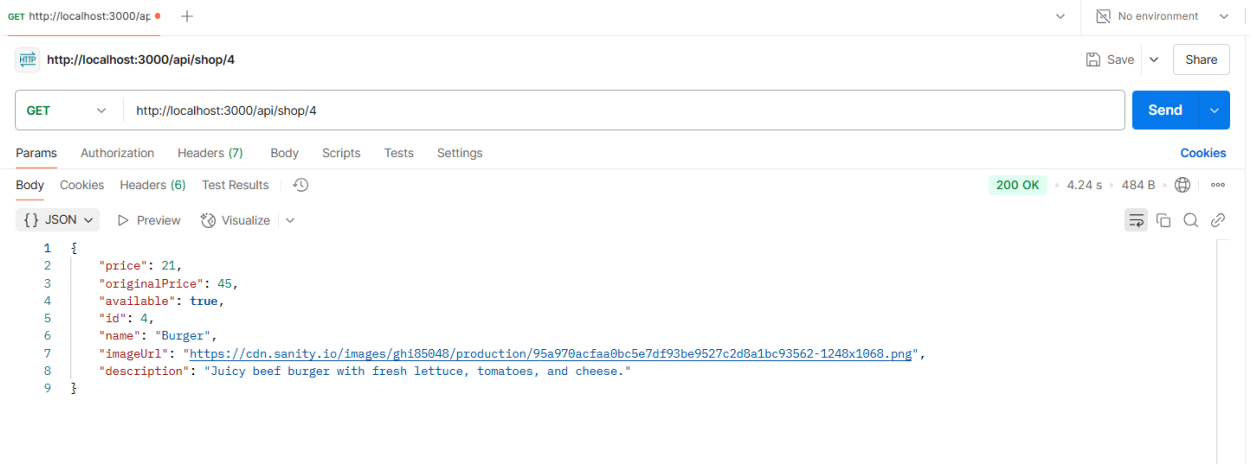
Successful fetching of all products



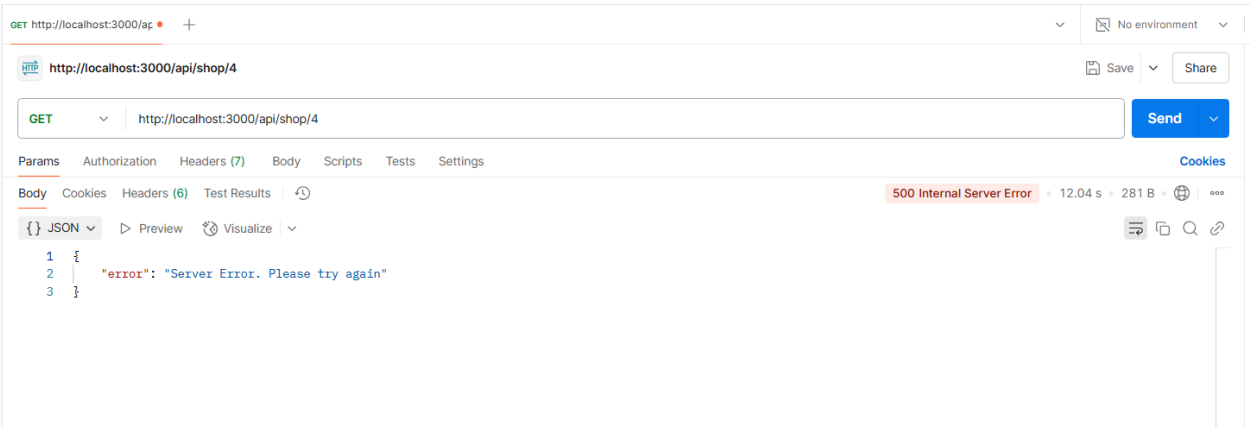
Error Handling for status 500



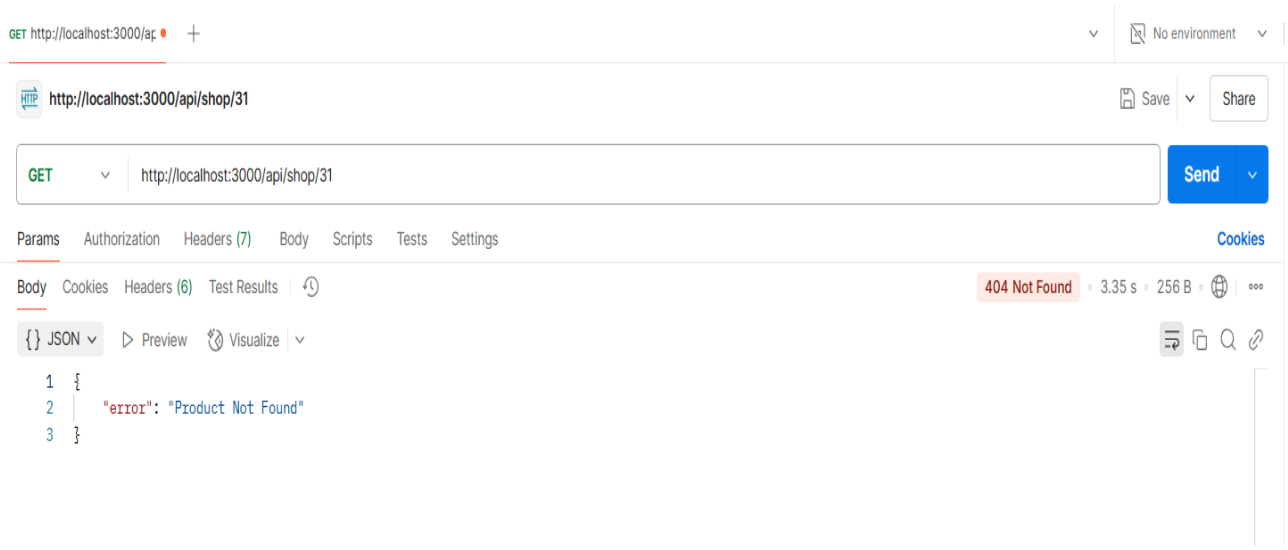
Successful fetching of individual product



Error Handling for status 500



Error Handling for status 404 - Not Found



Error Handling

In my project, I implemented robust error handling to ensure smooth functionality and a better user experience. I used **try-catch** blocks in API route handlers to catch and handle errors gracefully. For frontend error management, I incorporated an **error fallback UI**, allowing users to see meaningful error messages instead of application crashes. Additionally, I structured my error handling in a **modular and scalable** way, making it easier to maintain and extend. Logging mechanisms were also added to track unexpected errors and improve debugging.

src > lib > TS utils.ts > ...

```ts
export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs))
}

export const fetchProducts = async (api: string) => {
  try {
      const res = await fetch(api);
      const data = await res.json();

      if(!res.ok) {
        throw new Error(data.error || "Failed to load products");
      }
      console.log("Fetched products: ", data);
      return {data};
  } catch (error: any) {
      console.error("Error fetching products", error);
      return {error: error.message};
  }
}

```

src > app > shop > page.tsx > ShopPage

```tsx
export default function ShopPage() {

    useEffect(() => {
        fetchProducts('/api/shop').then((res) => {
            if (res.error) {
                setError(res.error);
                return;
            }

            if (res.data) {
                setMenu(res.data);
            }

            setError('');
            setLoading(false)
        })
    }, []);
```

Fallback UI Examples

```
95                {/* Food Cards */}
96                <div className="col-span-9 row-auto flex flex-wrap md:justify-normal justify-center mt
97                {error && (
98                        <p className="mx-auto mt-8">{error}</p>
99                )}
100                   {paginatedMenu && (
101                       paginatedMenu.map((food, idx) => (
102                       <Link key={idx} href={`/shop/${food.id}`} >
103                           <ShopCard
104                               ImagePath={food.imageUrl}
105                               AltText={food.name}
106                               ImageHeight={220}
107                               ImageWidth={244}
108                               DishName={food.name}
109                               CurrentPrice={food.price}
110                               OldPrice={food.originalPrice}
111                               />
112                           </Link>
113               )))
114               }
115           </div>
```

src > app > shop > [food_id] > page.tsx > ShopDetails > fetchProduct

```
36  export default function ShopDetails() {
79      useEffect(() => {
109     }, [food_id]);
110
111     if (loading) return <div>Loading...</div>;
112     // if (error) return <div>{error}</div>;
113
114     const socialIcons = [···
145     ]
146
147     const handleNext = () => {
148         if (currentIndex + itemsPerPage < allFood.length){
149             setCurrentIndex((prev) => prev + 1);
150         }
151     }
152
153     const handlePrev = () => {
154         if (currentIndex > 0) {
155             setCurrentIndex((prev) => prev - 1);
156         }
157     }
158
159
160     return (
161         <>
162             <Banner Title="Shop Details" Page="Shop details" />
163             <div className=" bg-white">
164             {foodData ? (
165                 <div className="container max-w-screen-lg mx-auto grid grid-cols-1 md:grid-cols-12 gap-y-4 md:gap-4 py-16 text-[#333333]">···
397                 </div>
398             ) : (<div className="flex justify-center items-center h-screen text-4xl"><p className=" ">{error}</p></div>)
399             }
400             </div>
401         </>
402     )
403 }
404
```

Cross-Browser & Device Testing

Description:
I tested the marketplace across various browsers and devices to ensure consistent performance and responsiveness.

Key Points:

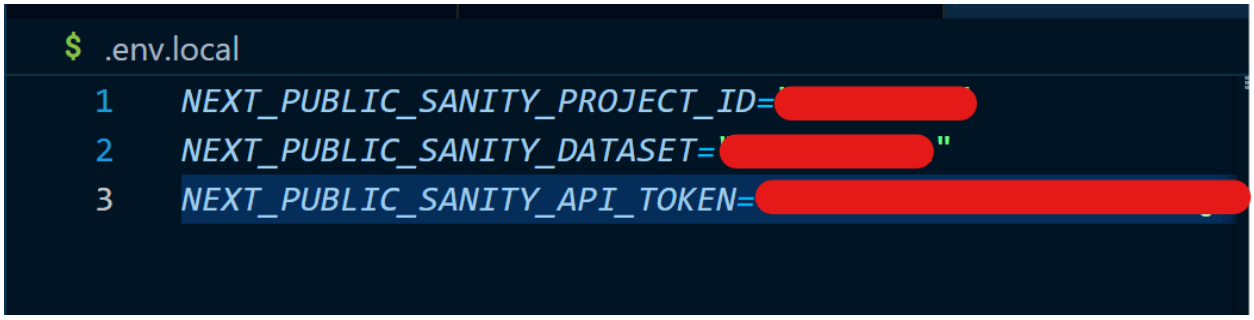
- Tested on popular browsers: Chrome, FireFox, Safari, and Microsoft Edge.
- Verified responsiveness on desktop, tablet, and mobile devices.
- Ensured no layout issues or broken features across different screen sizes.

Security Testing

Description:
I prioritized securing the marketplace by implementing measures to ensure safe communication and protect sensitive data.

Secure API Communication:

- Store sensitive data like API keys in environment variables to prevent exposure.



User Acceptance Testing (UAT)

Description:
I tested the marketplace to ensure it meets real-world usage expectations.

Key Points:

- Simulated tasks like browsing, navigating between different pages, search, and categorization.
- Collected feedback from peers to improve usability.

Expected Result:
A seamless and user-friendly experience.

ActualResult:
UAT completed successfully with no major issues.

Final Checklist

Task	Status
Functional Testing	☑
Performance Testing	☑
Error Handling	☑
Device Testing	☑
Security Testing	☑
Documentation	☑