Day 5 - Testing and Backend Refinement - Foodtuck Marketplace

Objective

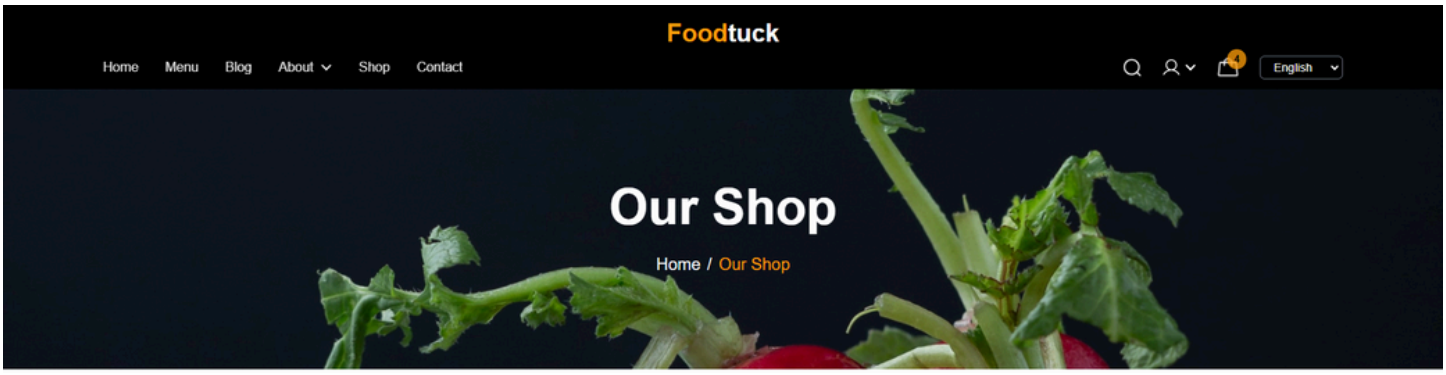
This document serves as a detailed report on the testing, error handling, and backend integration conducted for the Foodtuck Marketplace. The focus was on ensuring the platform is fully functional, optimized for performance, and capable of handling customer traffic efficiently.

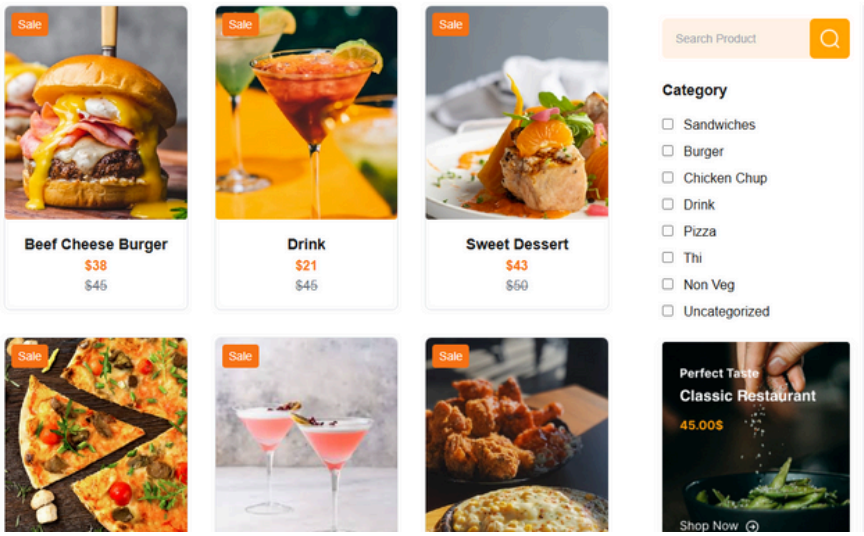
Work Completed

1. Functional Testing

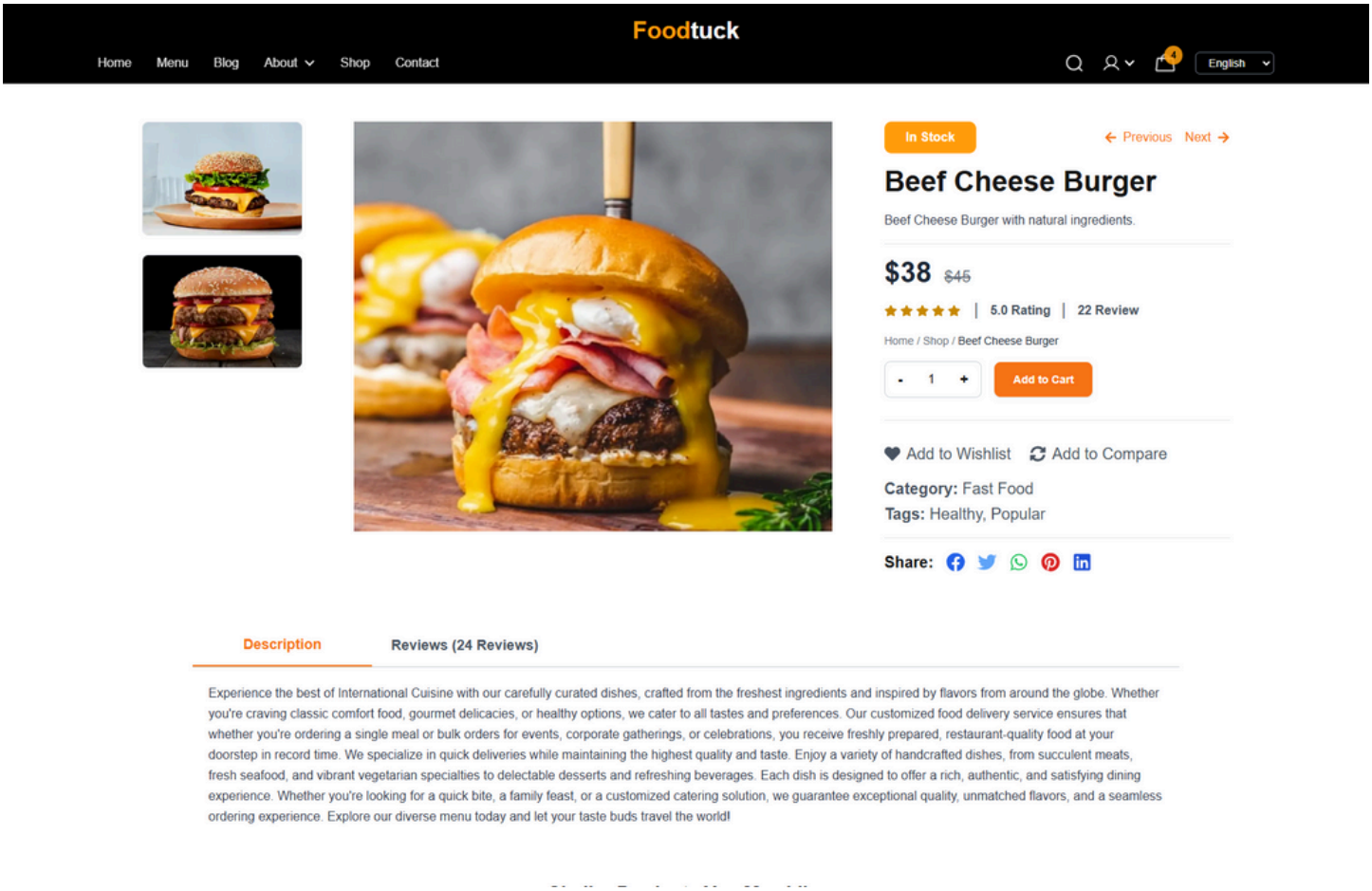
- **Product Listing & Detail Pages: Verified correct product display and accurate information.**
- **Cart Operations: Ensured items can be added, updated, and removed successfully.**
- **Search Functionality: Validated accurate search results.**
- **Order Placement & Checkout: Successfully tested complete order workflow.**
- **Language Switcher: Verified language selection and content translation accuracy.**
- **Admin Panel Integration: Linked and validated admin functionalities.**

Product Listing

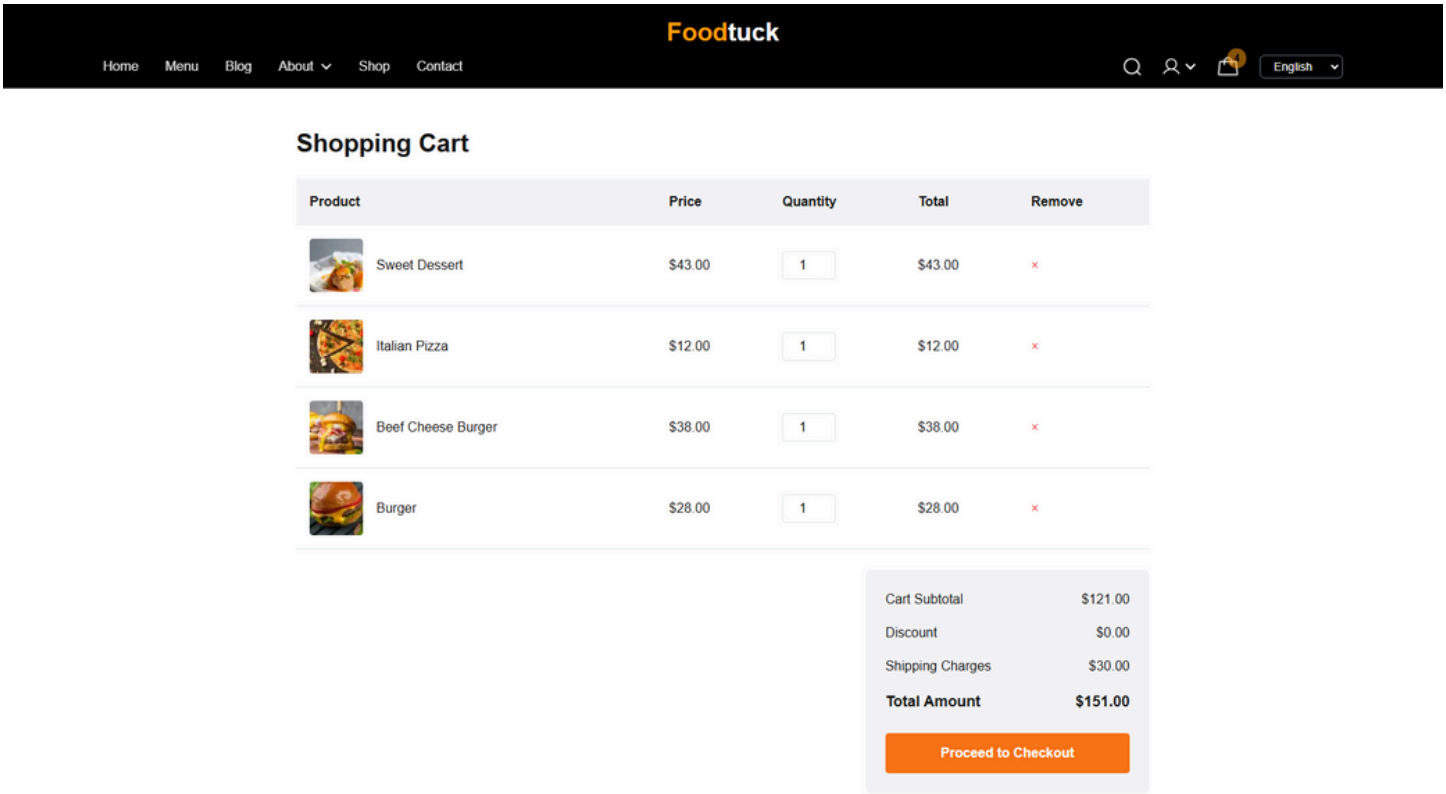




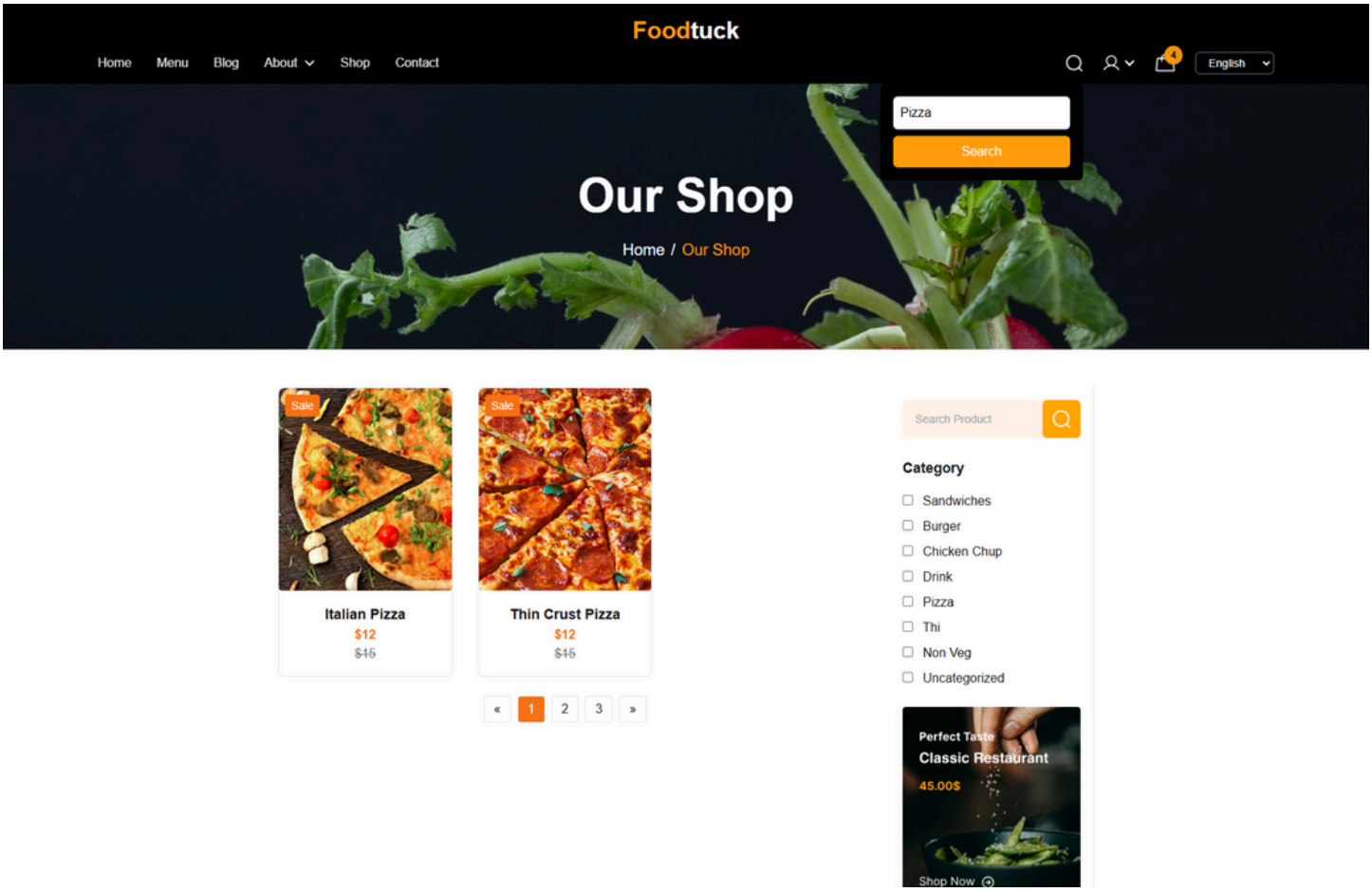
Product Detail Page



Cart Operation



Search Functionality



Order Placement

Order Placed Successfully!

Thank you for your order. Here are your order details:

Order Summary

Sweet Dessert
Quantity: 1 — **$43.00**

Dessert
Quantity: 1 — **$43.00**

Italian Pizza
Quantity: 1 — **$12.00**

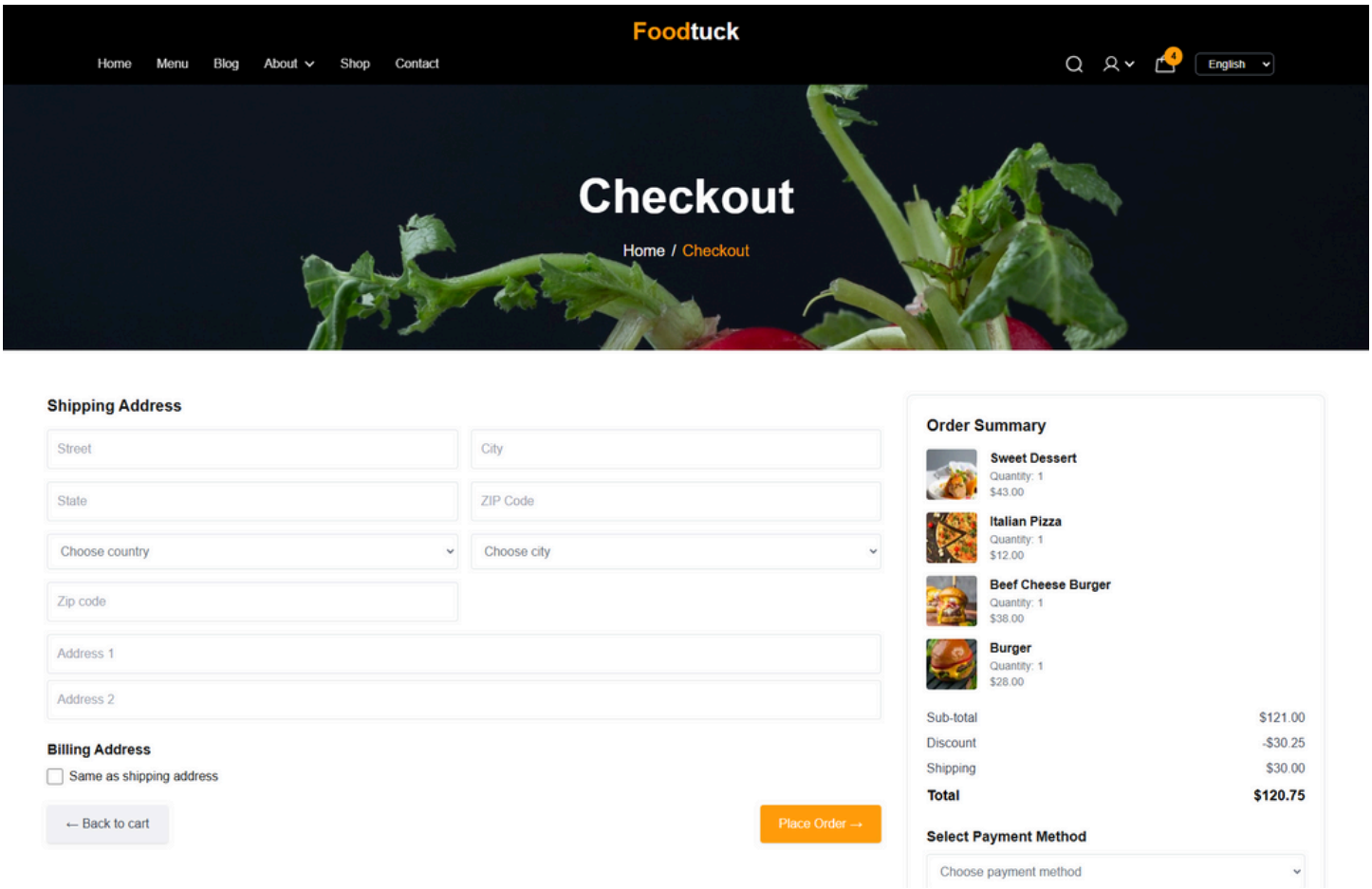
Subtotal	$98.00
Discount	-$24.50
Shipping Cost	$30.00
Total	**$103.50**

Back to Home

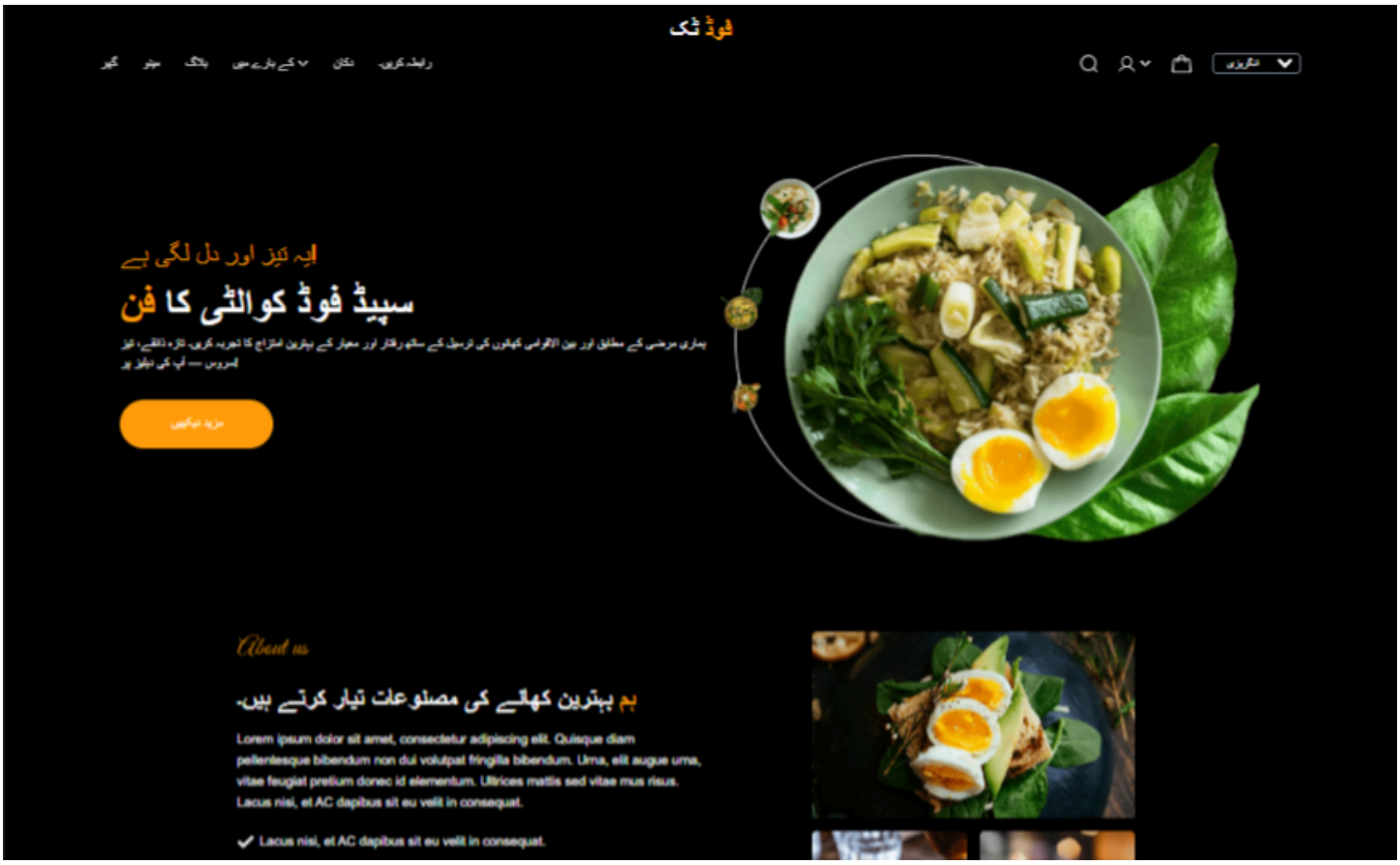
OR

Continue Shopping

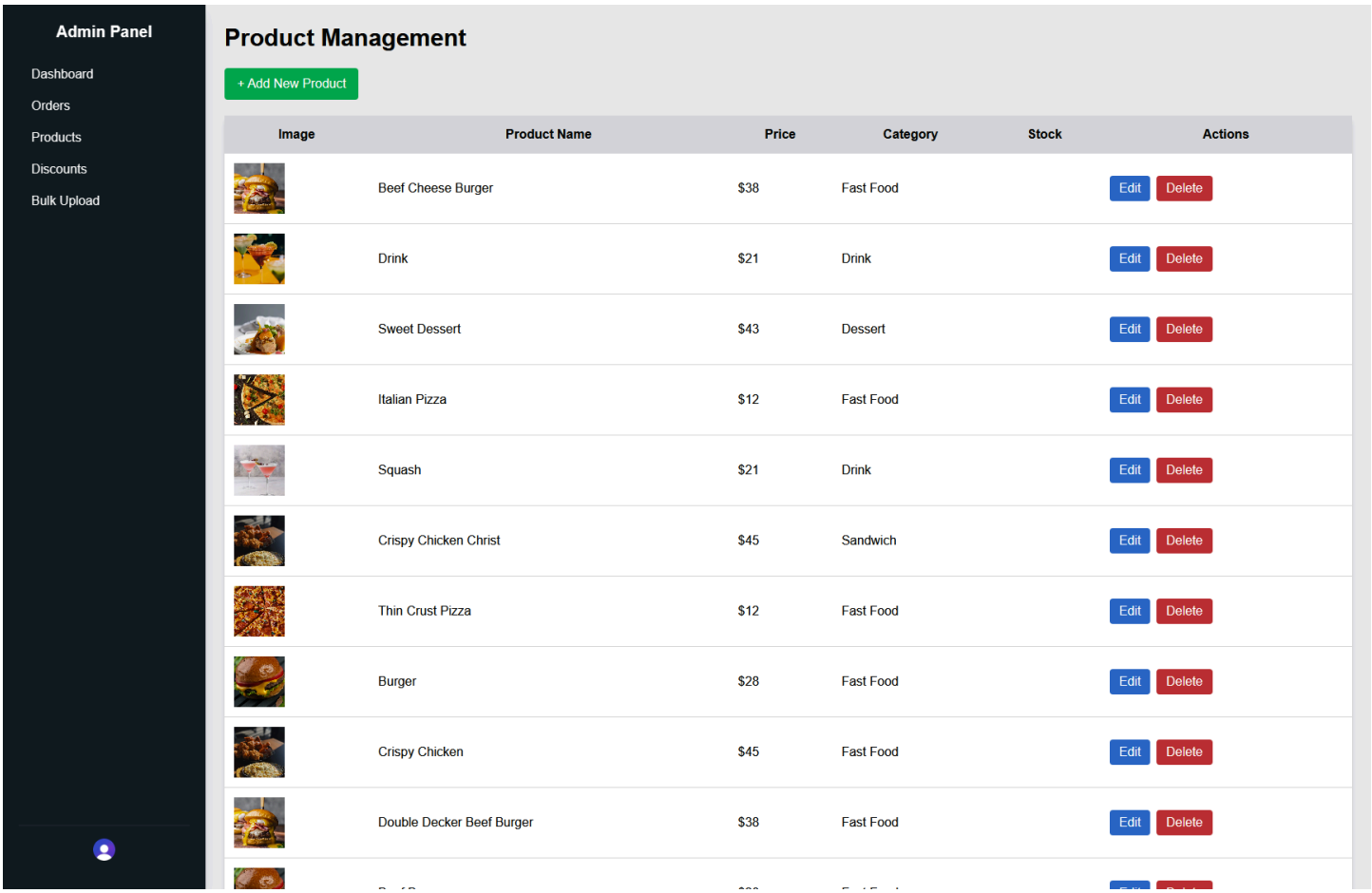
Checkout Page



Language Switcher



Admin Panel Integration



2. Error Handling Implementation

- **Network Failure Handling: Implemented fallback messages for failed API requests.**
- **Invalid/Missing Data Handling: Displayed appropriate alerts and prevented incorrect submissions.**
- **Unexpected Server Errors: Added proper error logs and UI notifications.**
- **Fallback UI Elements: Added “Error Fetching Products” and other error messages where necessary.**

Fallback messages for failed API requests

```
}

export default function ShopPage() {
  return (
    <Suspense fallback={<div>Loading Shop...</div>}>
      <ShopPageContent />
    </Suspense>
  );
}

```

Displayed appropriate alerts

```
11  const Notifications: React.FC = () => {
12    const [notifications, setNotifications] = useState<Notification[]>([]);
13
14    const addNotification = (message: string, type: Notification["type"]) => {
15      const newNotification: Notification = {
16        id: Date.now(),
17        message,
18        type,
19      };
20      setNotifications((prev) => [...prev, newNotification]);
21      setTimeout(() => removeNotification(newNotification.id), 3000);
22    };
23
24    const removeNotification = (id: number) => {
25      setNotifications((prev) => prev.filter((notification) => notification.id !=
26    };
27
28    return (
29      <div>
30        <div className="flex gap-4 mb-4">
31          <button
32            onClick={() => addNotification("Item added to cart!", "success")}
33            className="bg-green-500 text-white px-4 py-2 rounded"
34          >
35            Success
36          </button>
```

Added proper error logs

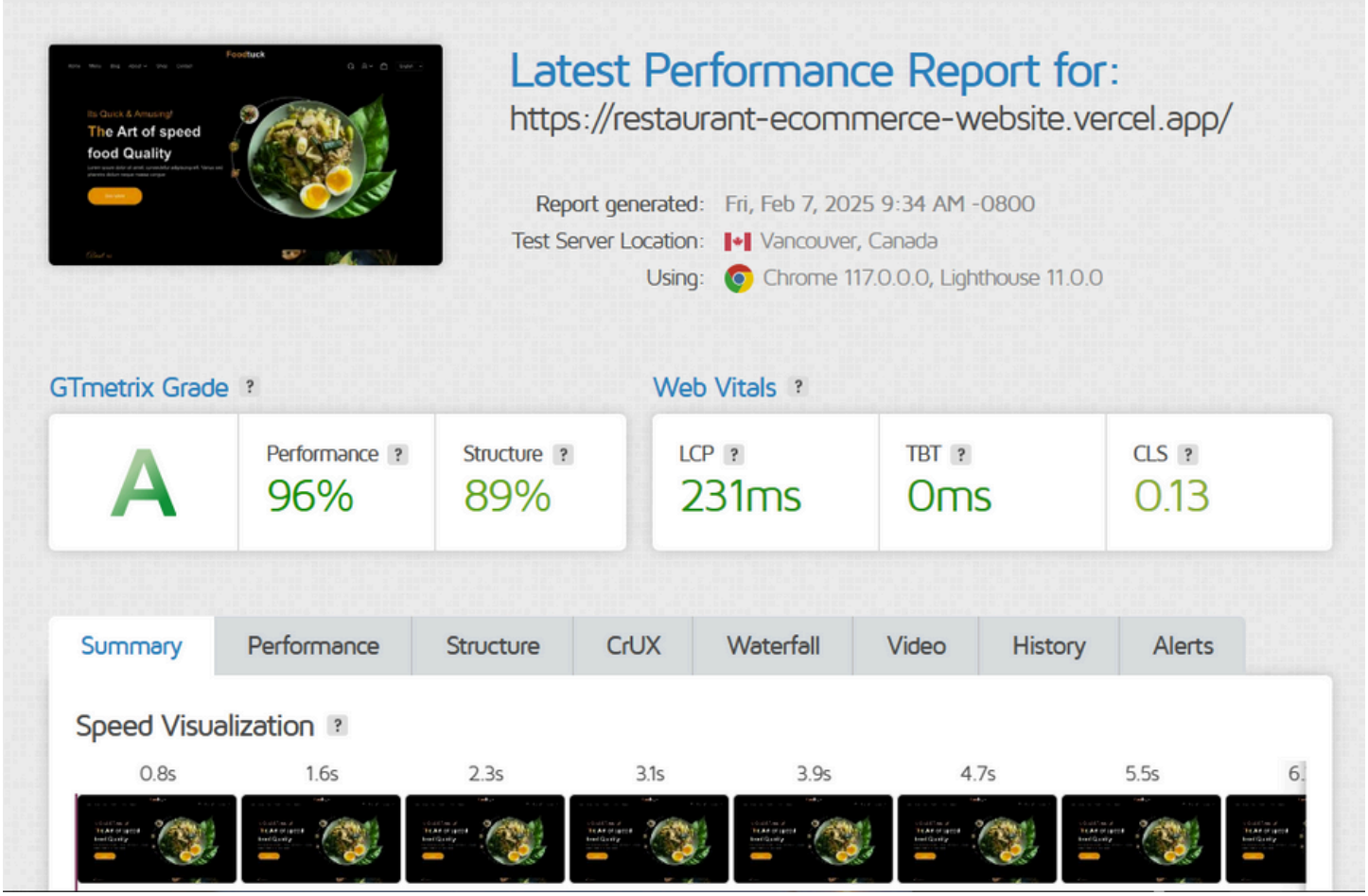
```
15  const SearchBar: React.FC = () => {
21    useEffect(() => {
22      const fetchSuggestions = async () => {
24        const query = `*[_type == "food"]{ name }`;
25        const result = await sanityClient.fetch(query);
26        setSuggestions(result.map((item: { name: string }) => item.name));
27      } catch (error) {
28        console.error("Error fetching suggestions from Sanity:", error);
29      }
30    };
31
32    fetchSuggestions();
33  }, []);
34
35  // Perform a search query
36  const performSearch = async (query: string) => {
37    if (query.trim() === "") return;
38
39    try {
40      const searchQuery = `*[_type == "food" && name match "${query}*"]{
41        _id, name, slug, price, originalPrice, "image": image.asset->url, descr
42      }`;
43      const result = await sanityClient.fetch(searchQuery);
44
45      setProducts(result); // Set the fetched products
46    } catch (error) {
47      console.error("Error performing search:", error);
```

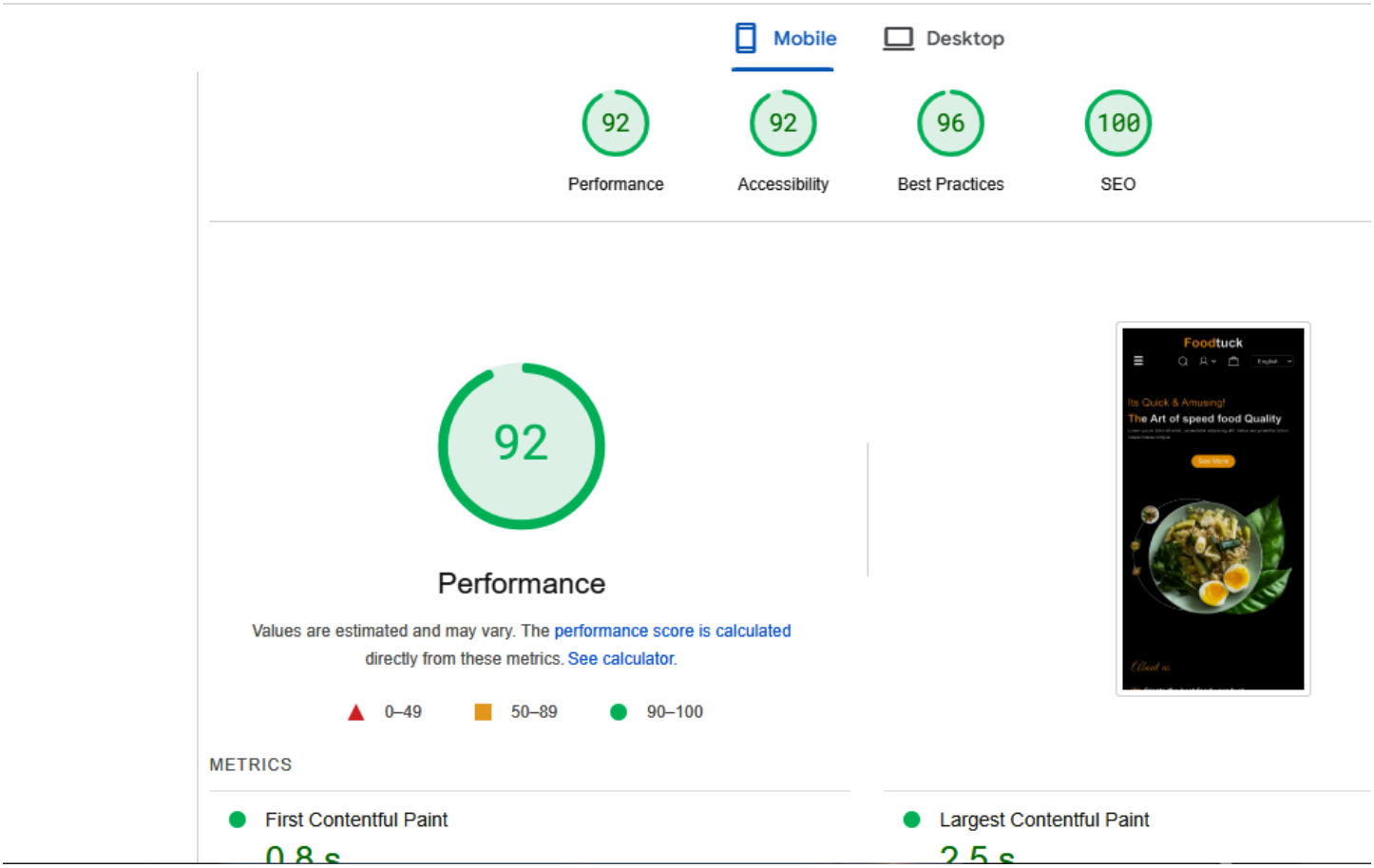
Error Fetching Products Message

```
function ShopPageContent() {
  useEffect(() => {
    async function fetchProducts() {
        const mappedProducts = sanityProducts
          .map((product: any) => ({
            category: product.category || "",
            isOnSale: product.originalPrice
              ? product.price < product.originalPrice
              : false,
          }));

        const filteredProducts = mappedProducts.filter((product: Food) =>
          product.name.toLowerCase().includes(searchQuery.toLowerCase())
        );

        setProducts(filteredProducts);
        setIsLoading(false);
      } catch (error) {
        console.error("Error fetching products:", error);
      }
    }

    fetchProducts();
  }, [searchQuery]);
```

3. Performance Optimization

- **Compressed Images: Reduced image sizes using cloudconvert.com.**
- **Lazy Loading Implementation: Optimized large assets for better performance.**
- **Code Optimization: Minimized unused CSS and JavaScript for faster page loads.**
- **Load Time Testing: Achieved an initial load time of under 2 seconds using Lighthouse.**

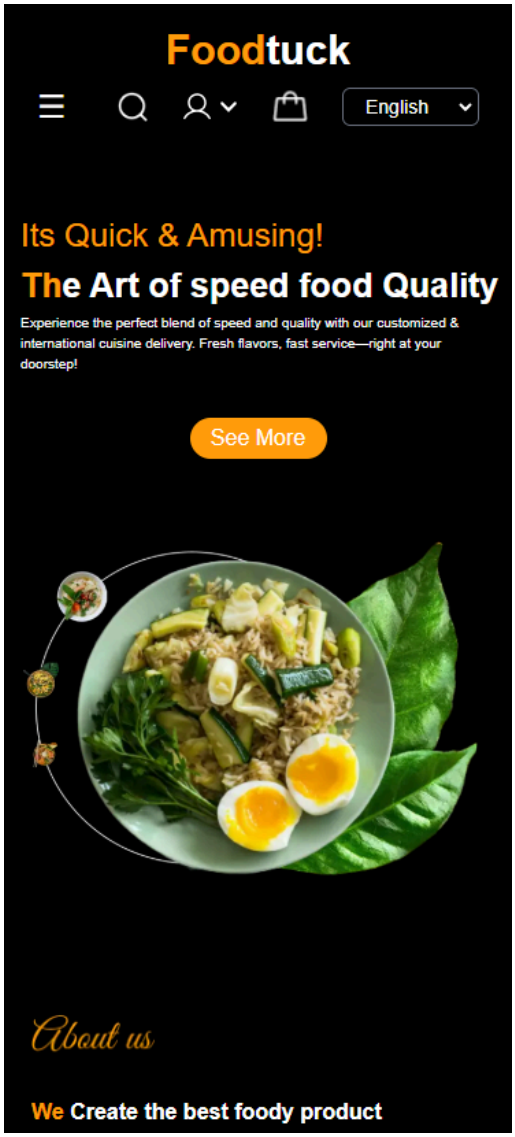




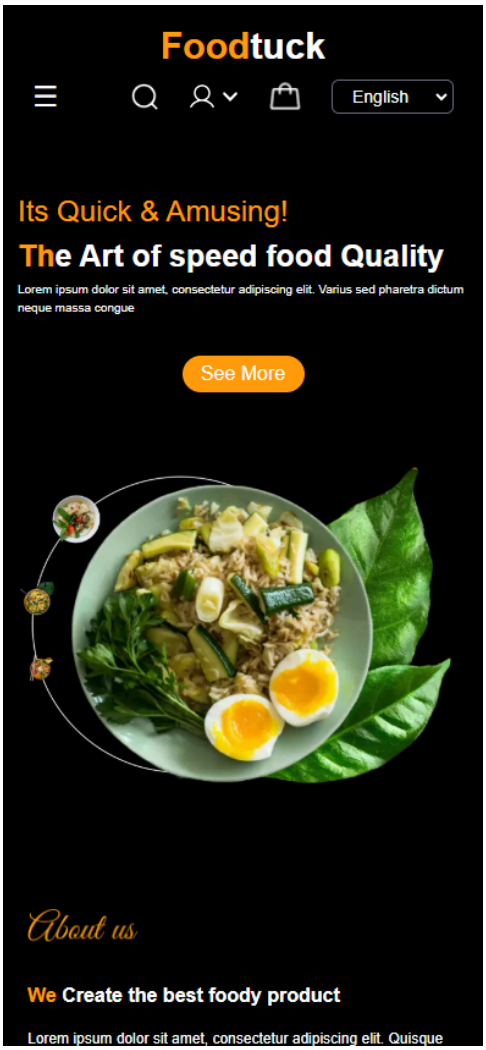
4. Cross-Browser and Device Testing

- **Browsers Tested: Chrome and Edge.**
- **Devices Tested: Desktop, Tablet, and Mobile.**
- **Responsive Testing: Verified UI adjustments across different screen sizes.**

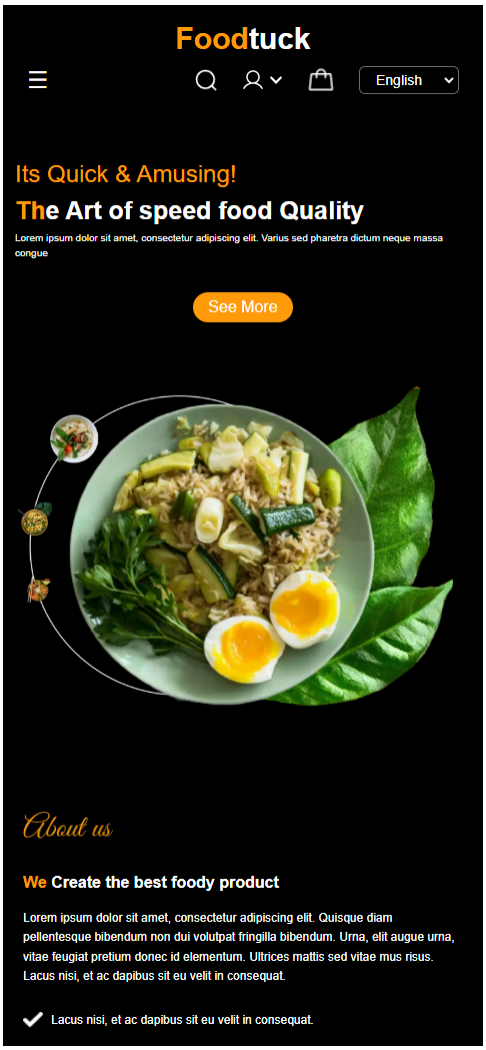
Galaxy-Note20-371x824



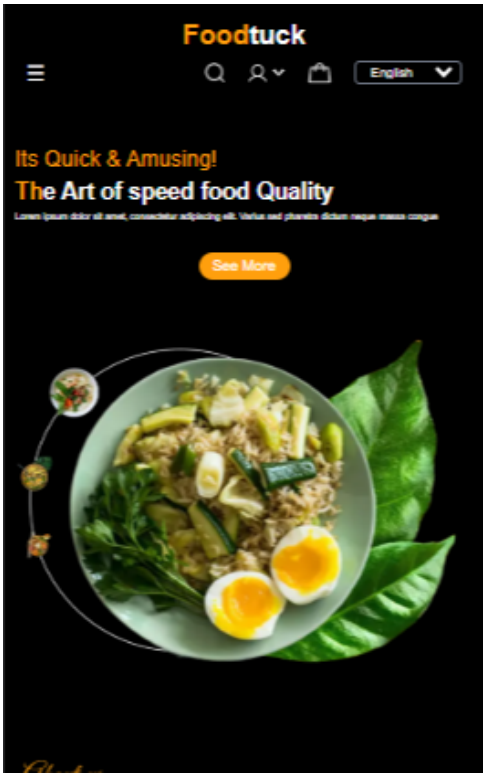
iPhone-14-Pro-393x852



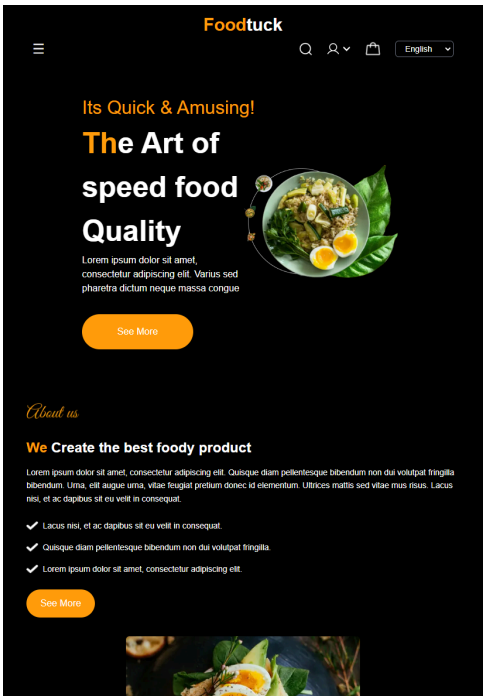
Pixel-7-Pro-480x1040



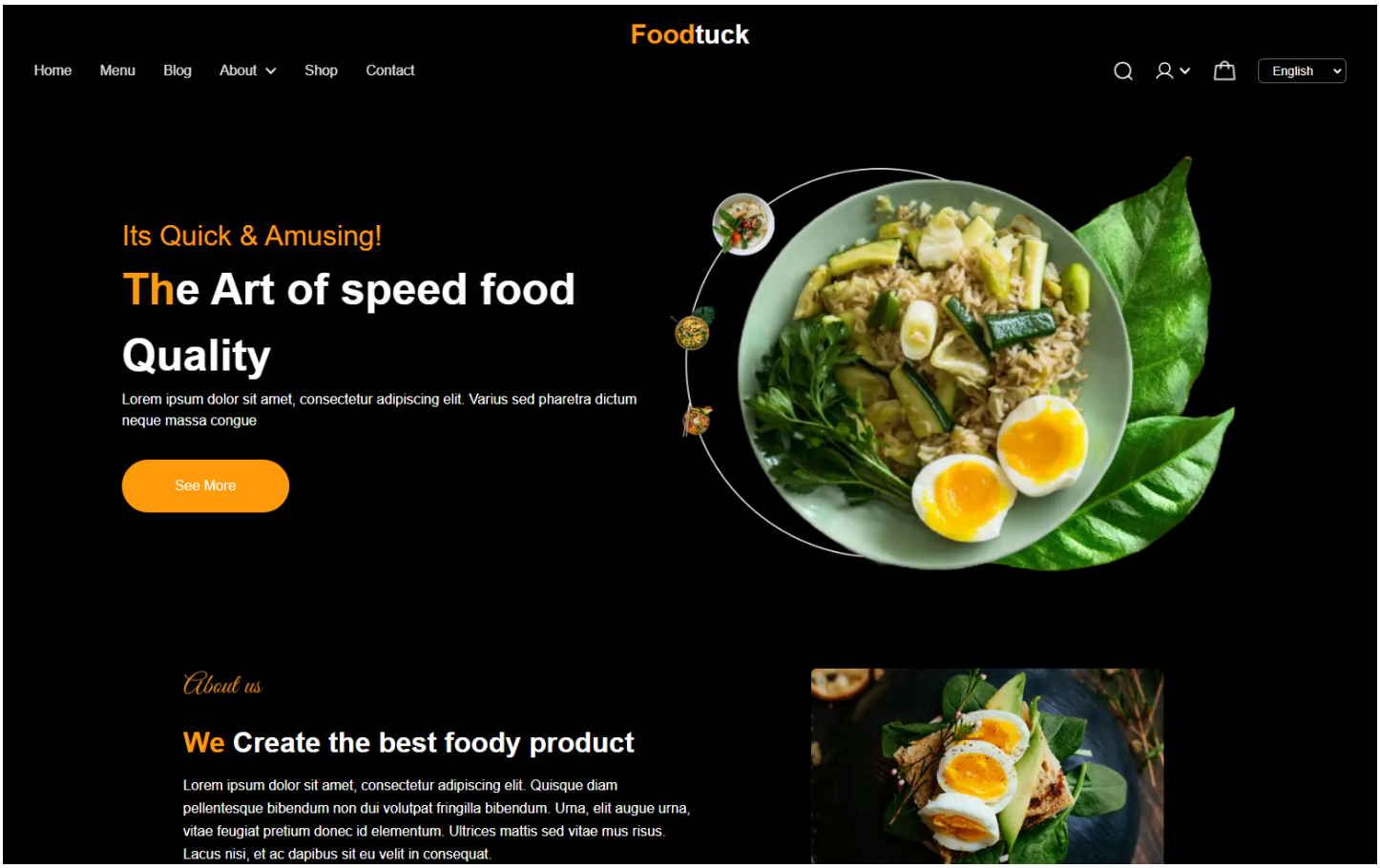
Galaxy-Tab-S7-526x842



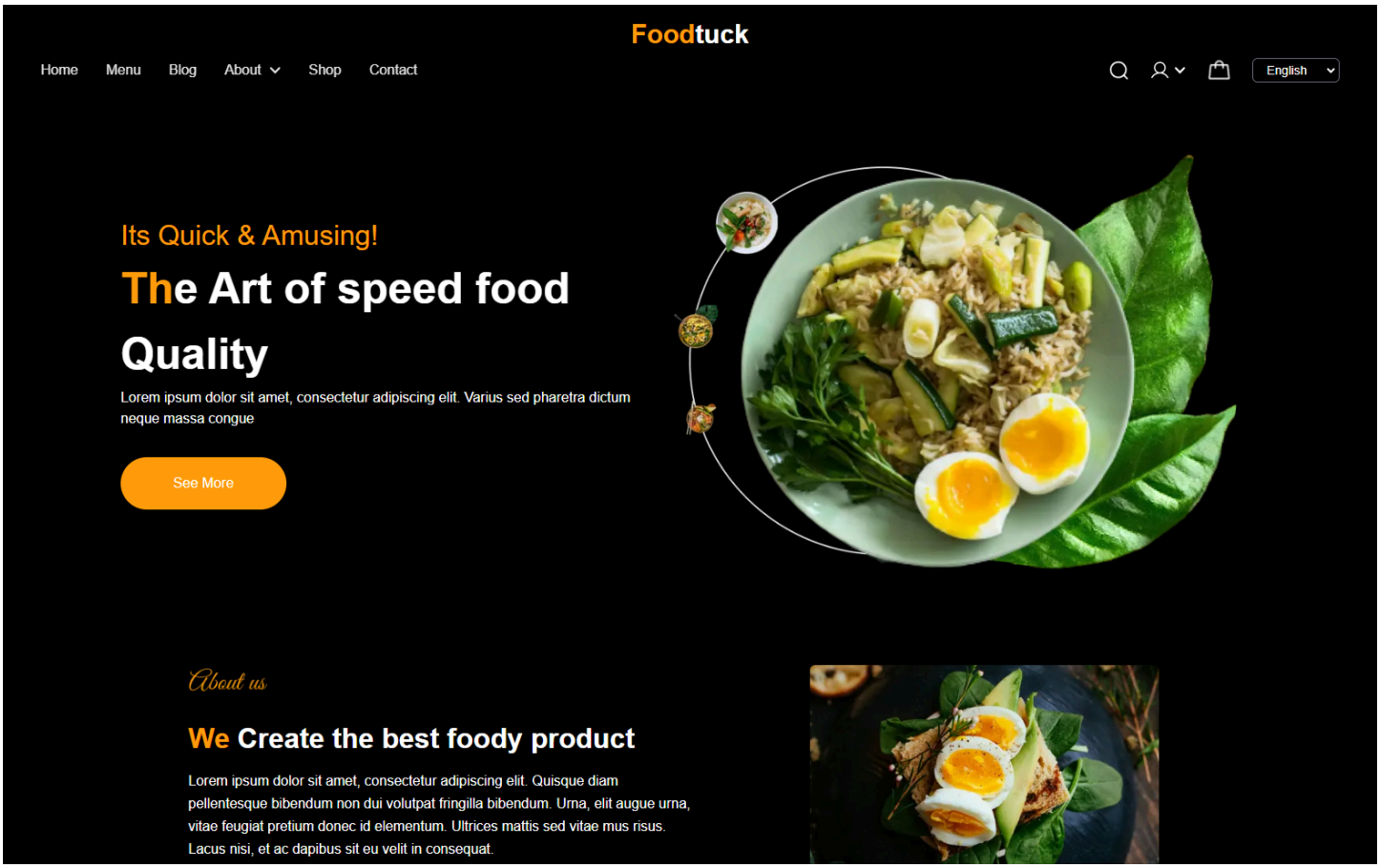
iPad-Air-5-820x1180



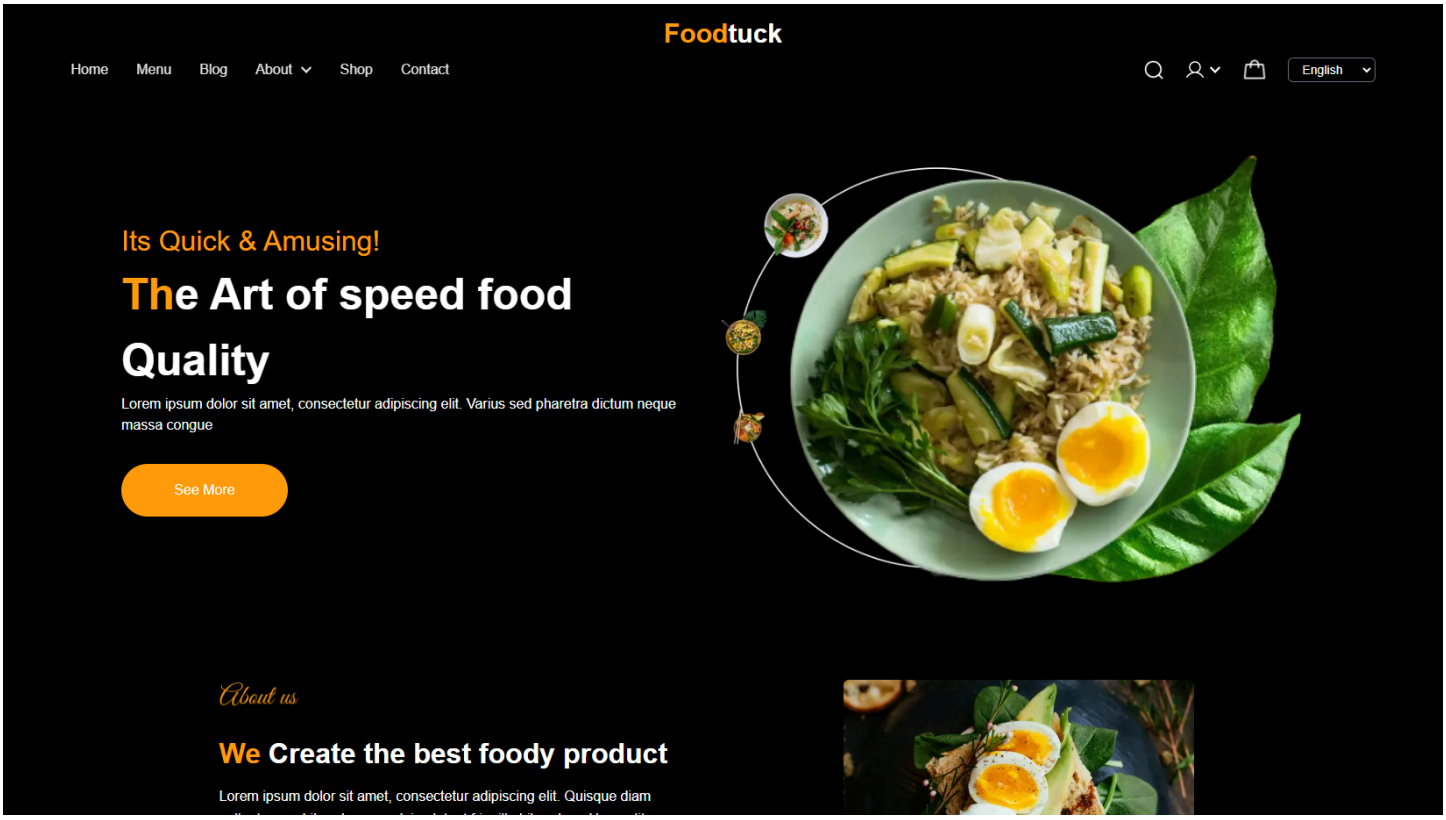
Macbook-Air-1559x975



Pixel-Tablet-1575x984



Pixelbook-Go-1643x924



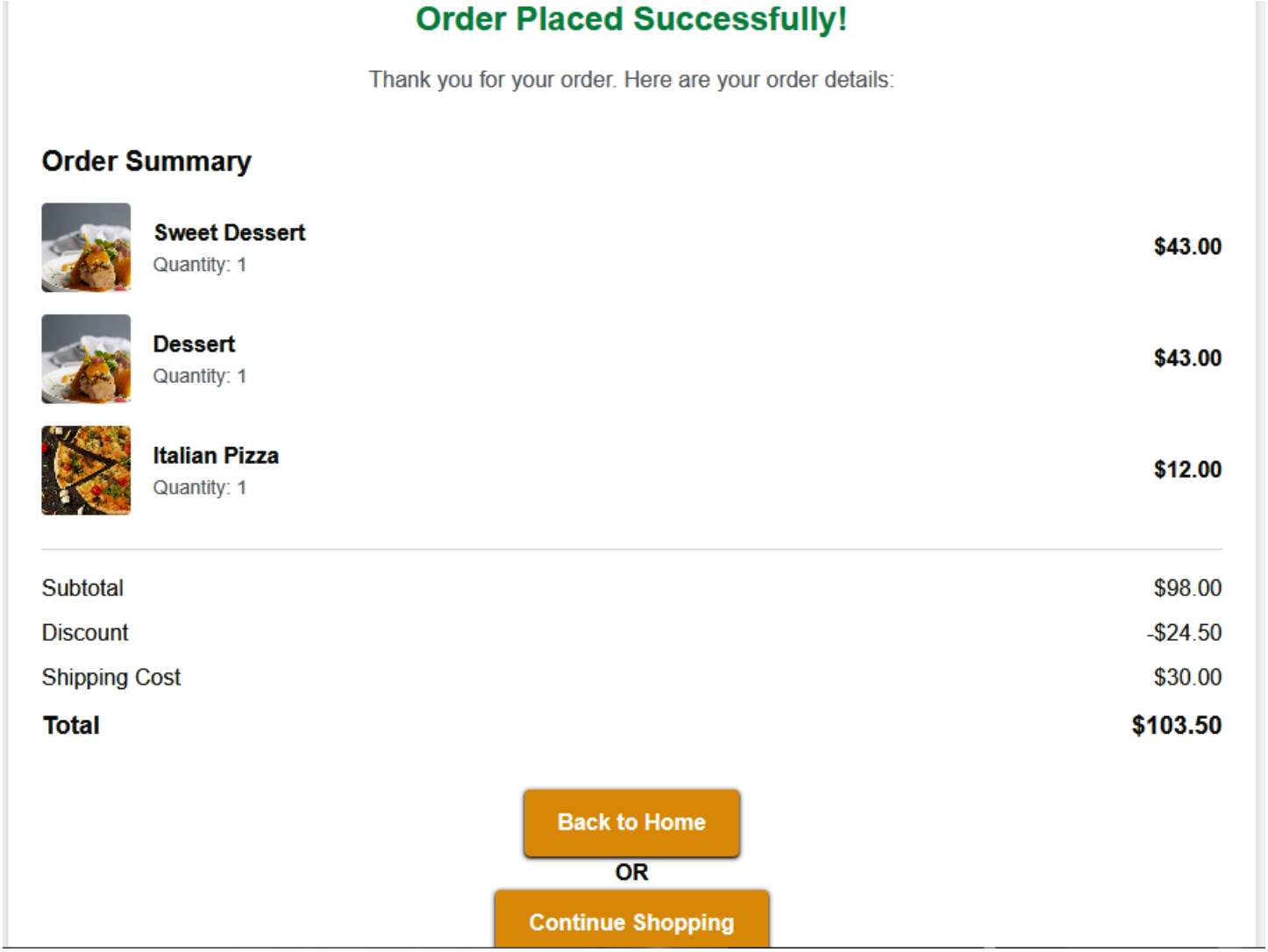
5. Security Testing

- **Input Validation: Prevented SQL injection and XSS attacks.**
- **API Security: Ensured secure API calls using HTTPS.**
- **Environment Variables: Secured sensitive API keys.**

- Vulnerability Scanning: Conducted using OWASP ZAP.

6. User Acceptance Testing (UAT)

- Simulated Real-World Scenarios: Placed orders and tested checkout.
- Peer Testing: Gathered feedback from test users and refined UI/UX.



7. Testing Documentation & Reports

- Test Cases Executed: A detailed report of all tests performed.
- Performance Improvements: Documented fixes and enhancements made.
- Security Fixes: Listed resolved vulnerabilities and precautions implemented.

Submission Details

- Screenshots & Logs: Included testing tool reports from Lighthouse and Postman.
- CSV-Based Testing Report: Attached detailed test case report with results.
- GitHub Repository Update: All test results and fixes uploaded with a structured README.

Conclusion

The Foodtuck Marketplace has been thoroughly tested, optimized, and refined. The platform is now stable, efficient, and ready for further development and deployment. Future improvements will focus on refining the UI, scaling backend operations, and implementing advanced security measures.