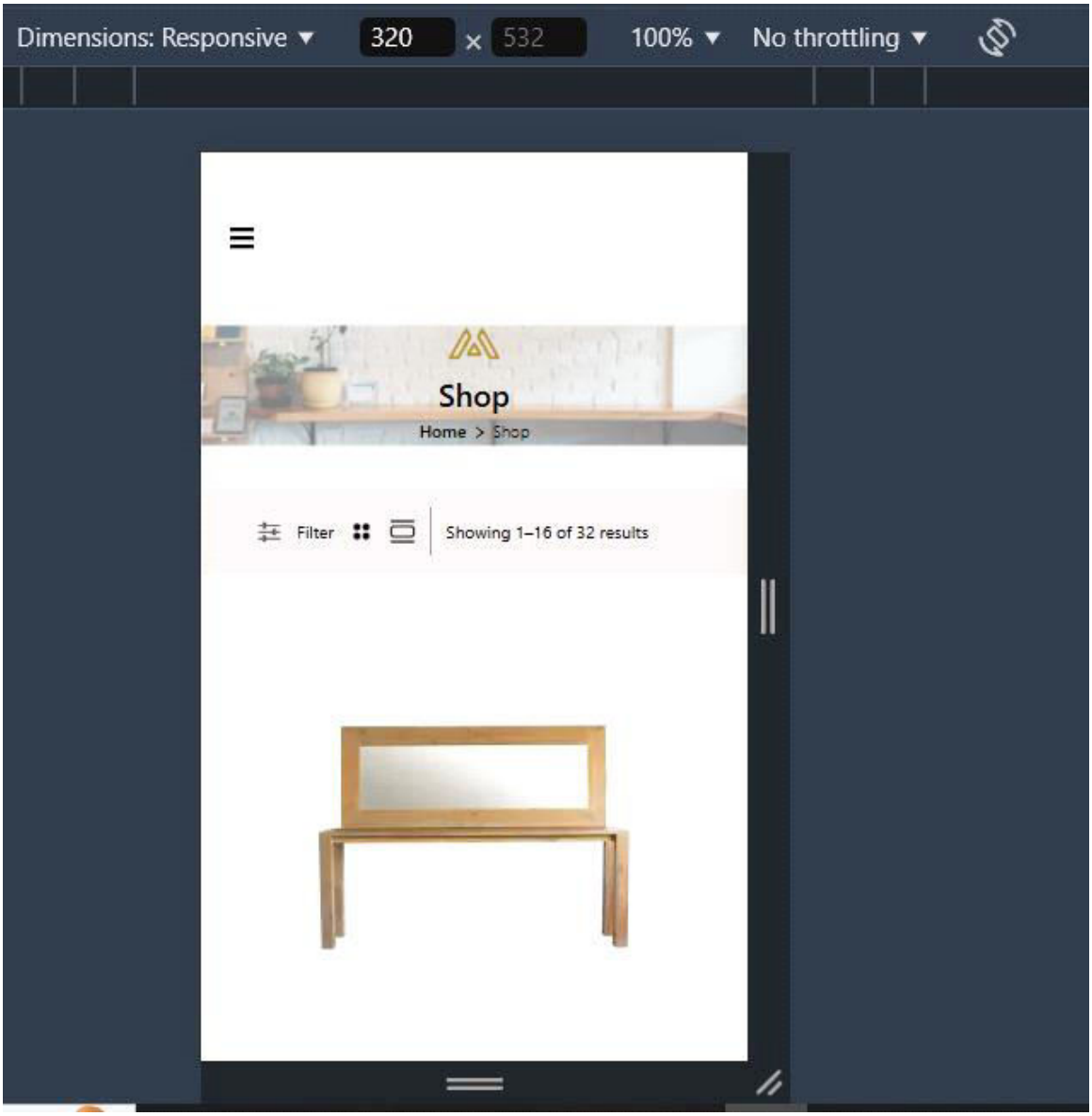
Day 5 - Testing and Backend Refinement - FurniSphere

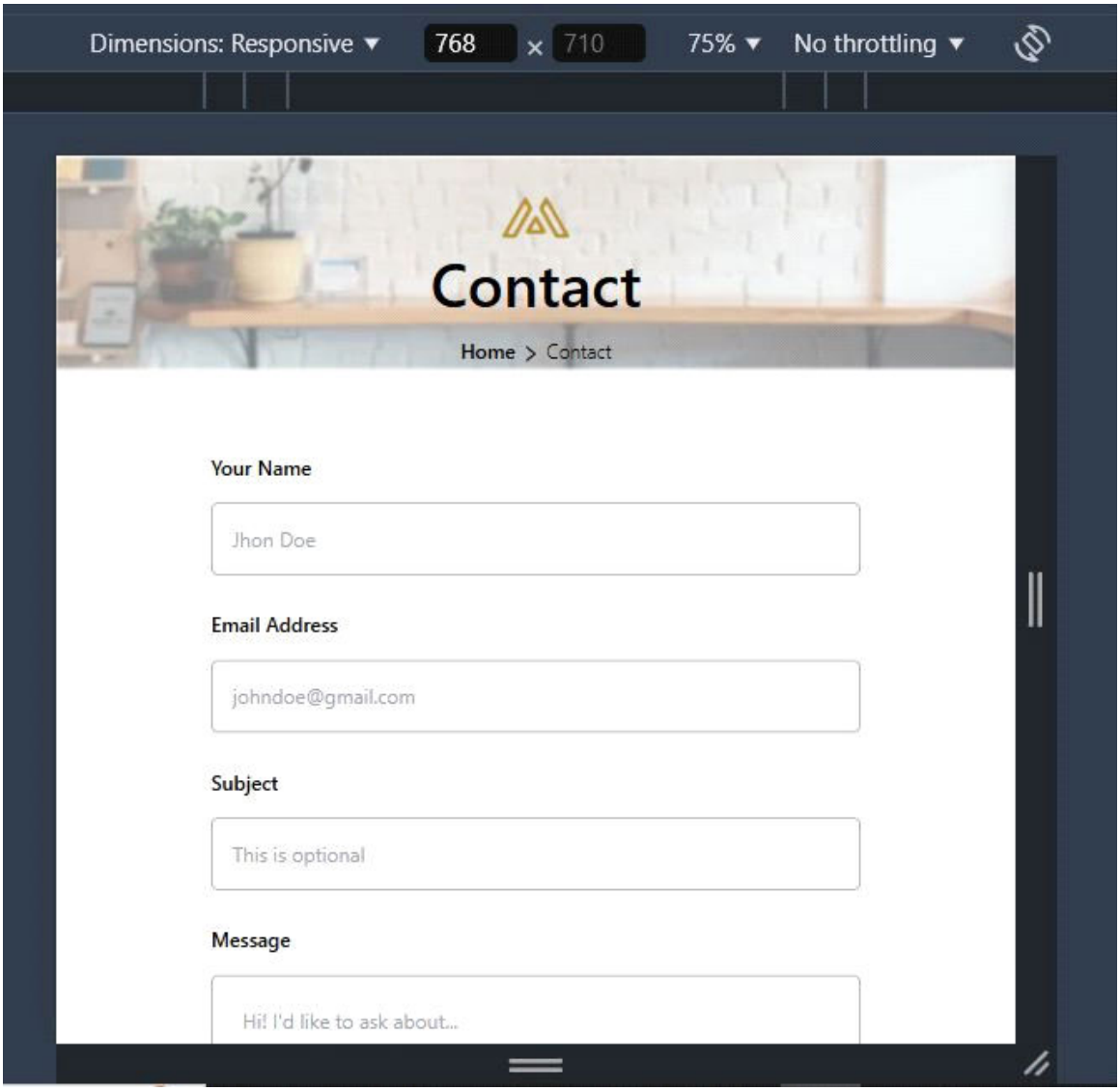
1. Functional Testing:

Test Cases Executed

A	B	C	D	E	F	G
Test Case ID	Description	Steps	Expected Result	Actual Result	Status	Remarks
TC01	Validate product listing	Load homepage, verify displayed products	roducts load and display correctly	Products load as expected	Passed	Displayed perfectly
TC02	Test search functionality	Enter query in search bar, check results	Correct results matching query	query not matches result	Failed	need some changes
TC03	Add product to cart	Click "Add to Cart" on a product	Product is added to cart	Product added	Passed	sucessfully added
TC03	Delete product from cart	Click delete icon	Product is deleted from cart	Product deleted	Passed	sucessfully deleted
TC04	Add product to wishlist	Click wishlist icon	Product is added to wishlist	Product added	Passed	added sucessfully
TC05	Delete product from wishlist	Click delete icon	Product is deleted from wishlist	Product deleted	Passed	deleted sucessfully
TC06	Dynamic routing for product	Click on a product	Individual product page loads and open detailed page	Product detail page opens	Passed	details renderd greatfully
TC07	Api error handling	remove API and loads page	UI fallback display	UI fallback dispalyed	Passed	Handled perfectly
TC08	Input sanitization	Add required inputs in contact form	Invalid input data displays error message	error message displayed	Passed	perfect sanitization
TC09	Filter products	Set filter on product listing page	Products filtered according to category	Products are not filtered based on category	Failed	need improvement
TC10	Fallback UI for offline errors	Disconnect wifi and loads product detail page	Fallback UI message display "You are offline"	message displayed as expected	Passed	message displayed sucessfully
TC11	Responsivness on mobiles/tablets	uses tool browserstack	All components are responsive on both devices	Some components needs improvemnt	Failed	minor changes needed

Responsivness


Dimensions: Responsive
320
532
100%
No throttling
Shop
Home > Shop
Filter
Showing 1–16 of 32 results


Dimensions: Responsive
768
x
710
75%
No throttling
Contact
Home > Contact
Your Name
Jhon Doe
Email Address
johndoe@gmail.com
Subject
This is optional
Message
Hi! I'd like to ask about...

2. Error Handling:

Implemented Mechanisms

- Used try-catch blocks to handle API errors.
- Displayed fallback messages, e.g., "Failed to fetch products."

```
Tabnine | Edit | Test | Explain | Document
async function fetchProductsData(page: number, limit: number) {
  try {
    const Productsdata: IProductsData[] = await client.fetch(`
      *[_type == "product"] {
        product_name,
        product_description,
        price,
        stock,
        sizes_available,
        "ImageUrl": image.asset->url,
        id
      }[${(page - 1) * limit}..${page * limit - 1}]
    `);
    return Productsdata;
  } catch (error) {
    console.error("Error fetching products:", error);
    throw new Error("Failed to fetch products");
  }
}
```

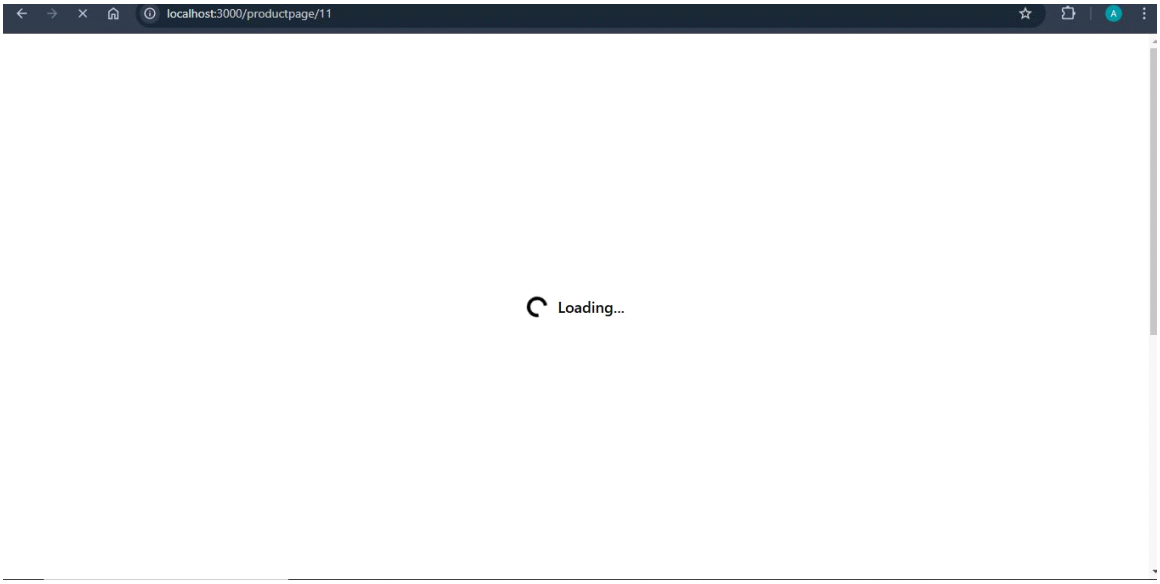
Fallback UI

- Displayed alternative fallback UI if the data is unavailable such as "Product not found".
- Displayed fallback for offline errors like "You are offline".

```
// fallback for offline

if (isOffline) {
  return (
    <div className="flex items-center justify-center h-screen">
      <div className="text-center text-2xl font-semibold text-red-500">
        <p>YOU ARE OFFLINE!</p>
        <p className='text-lg text-black mt-2'>Please check your internet connection to view the product details.</p>
      </div>
    </div>
  );
}

// Fallback UI for product not found

if (productNotFound) {
  return (
    <div className="flex items-center justify-center h-screen">
      <div className="text-center text-lg font-semibold text-red-500">
        <p>Product Not Found</p>
        <p>The product you&#39;re looking for doesn&#39;t exist.</p>
      </div>
    </div>
  );
}
```

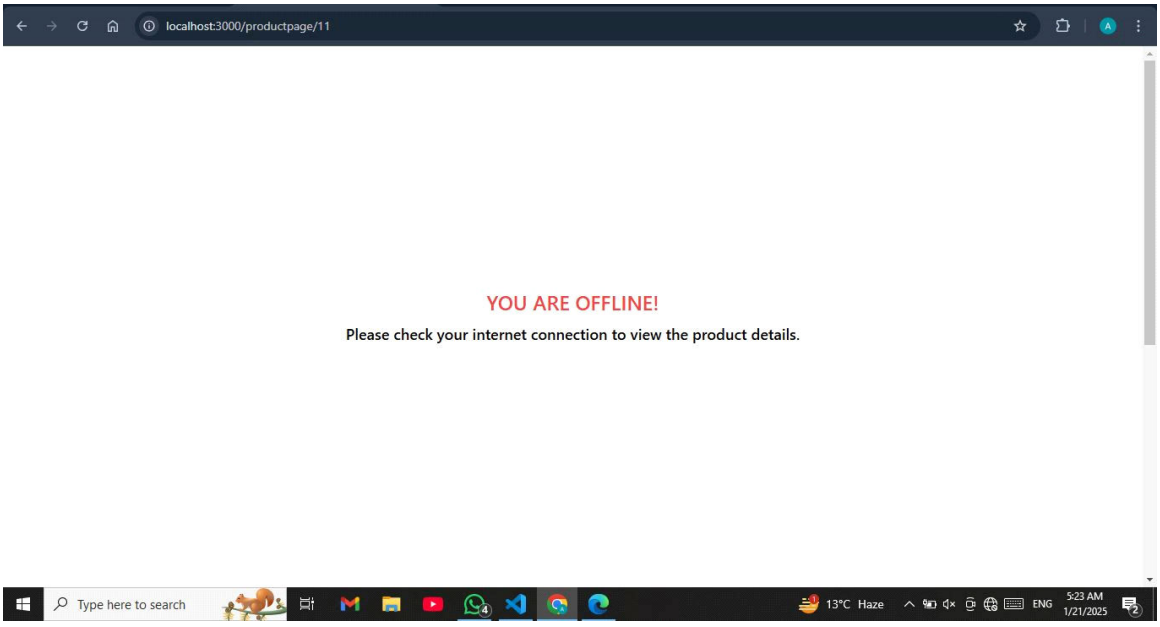
Loader

- Displayed loaders while data is fetching for better user experience.



Offline Fallback

- Displayed offline fallback UI when the network is not available a user-friendly message will dispalyed on screen.

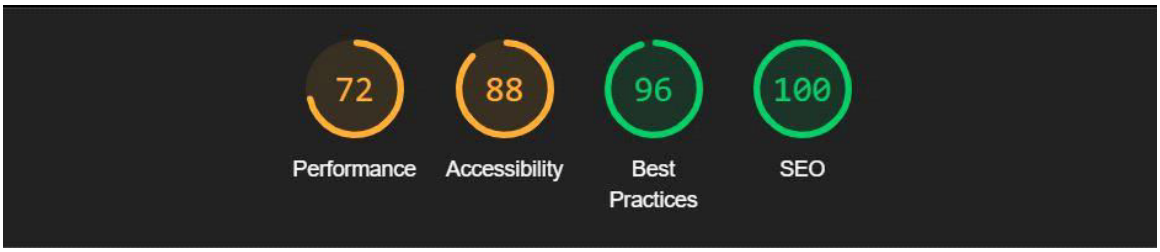


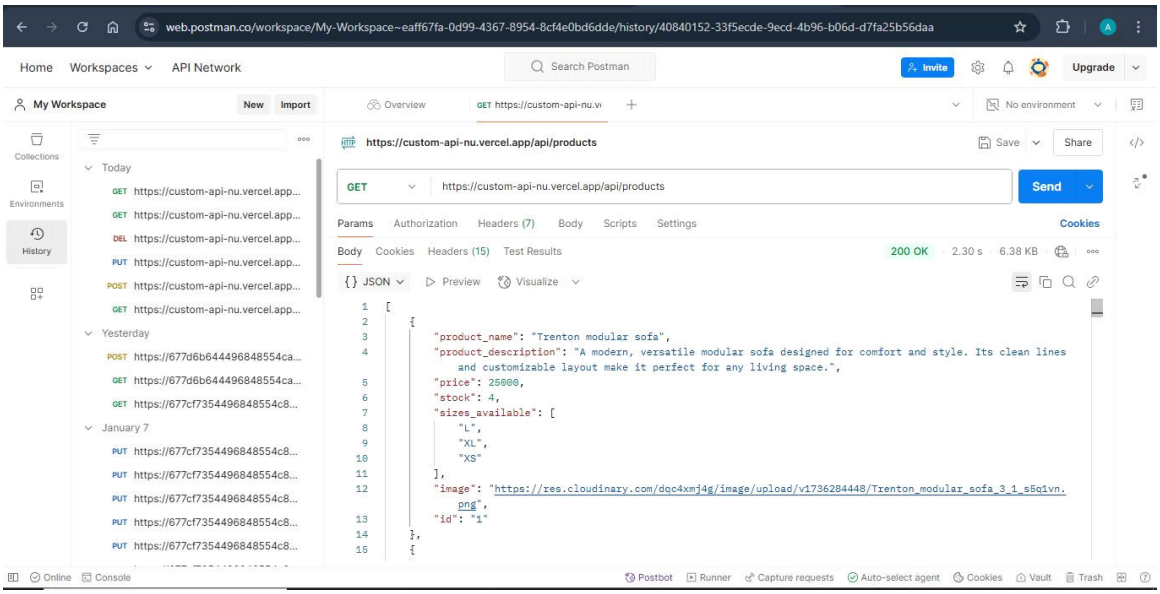
3. Performance Testing:

Optimization Steps

- Compressed images using TinyPNG.
- Implemented lazy loading for large assets.
- Used tools like lighthouse to test the performance of website.

- Uses postman to test API response.





4. Cross-Browser and Device Testing:

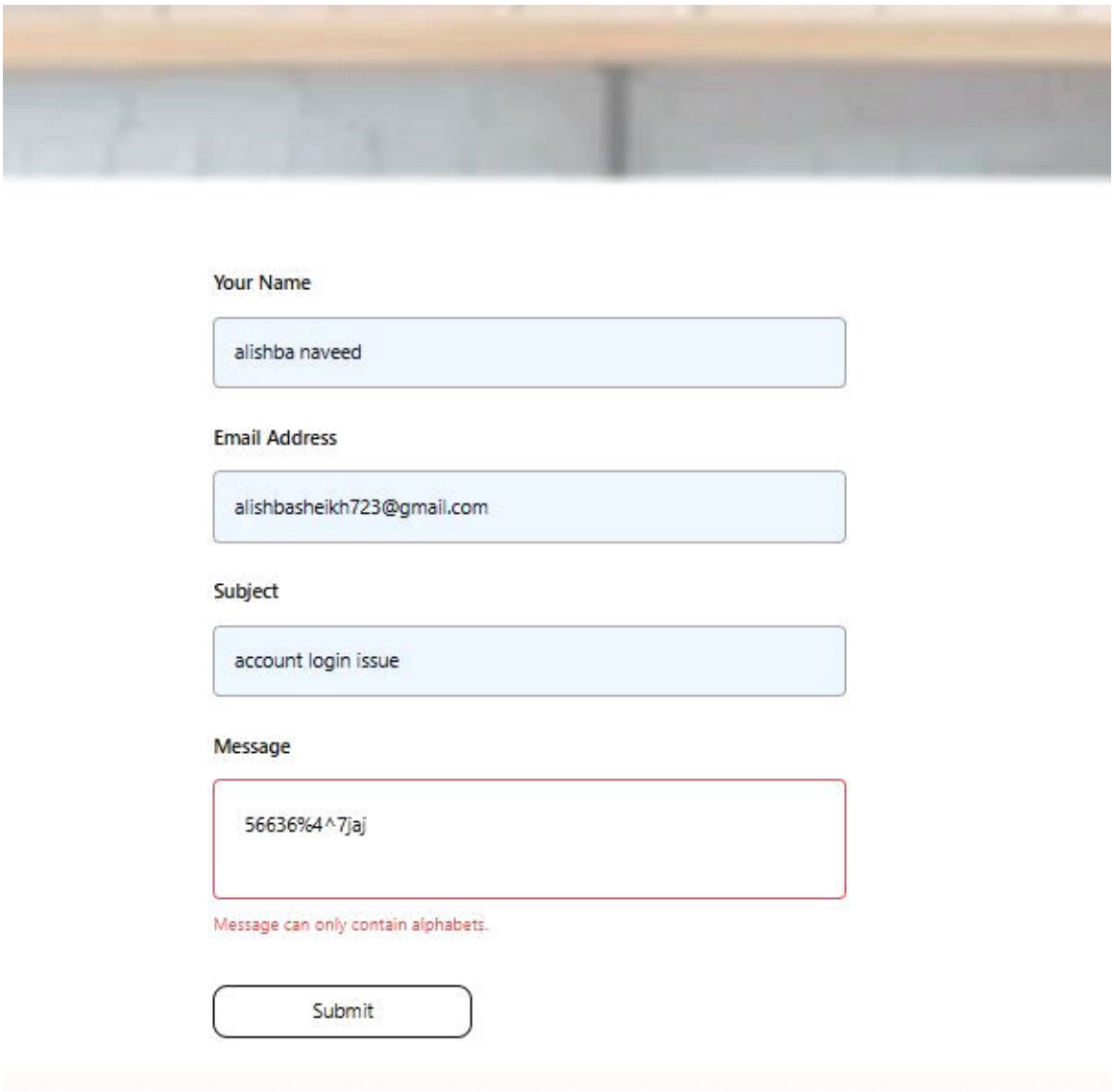
- Tested responsivness on different browsers to ensure compatability.
- Used tool like BrowserStack to verify responsivness on several devices and browsers.
- Ensures that all components rendered and functioned consistently across tested browsers and devices.

5. Security Testing:

Input Validation

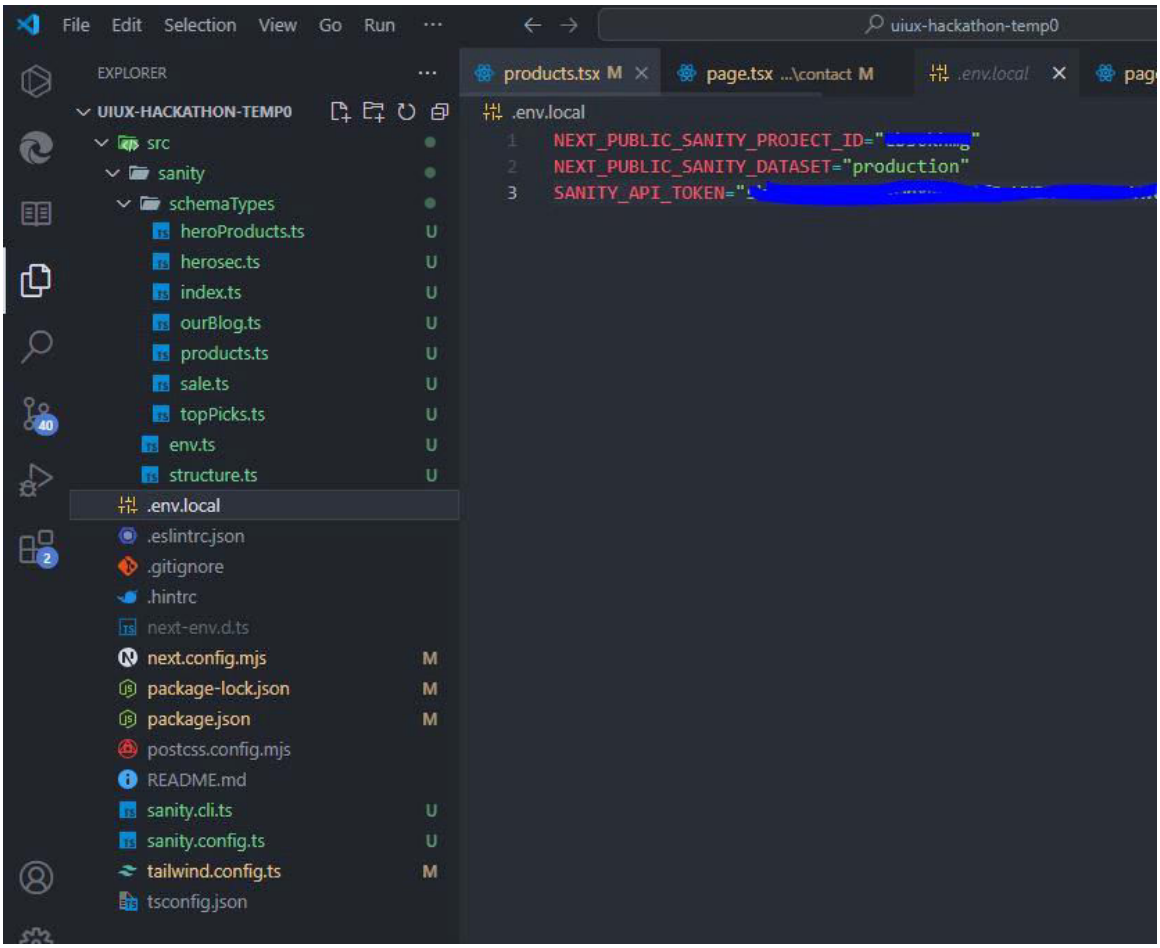
- Sanitized inputs to prevent SQL injection and XSS attacks.
- Added error messages for invalid inputs.

```
const Page = () => {
  const validateInputs = () => {
  //! Validation for name input

    if (!formData.name.trim()) {
      newErrors.name = "Name is required.";
    } else if (!/^[a-zA-Z\s]+$/.test(formData.name)) {
      newErrors.name = "Name can only contain letters and spaces.";
    }

   //! Validation for email input

    if (!formData.email.trim()) {
      newErrors.email = "Email is required.";
    } else if (
      !/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/.test(formData.email)
    ) {
      newErrors.email = "Please enter a valid email address.";
    }

  //! Validation for message

    if (!formData.message.trim()) {
   newErrors.message = "Message is required.";
  } else if (formData.message.length > 500) {
   newErrors.message = "Message cannot exceed 500 characters.";
  } else if (!/^[a-zA-Z\s!@#$%^&*(),.?":{}|<>]+$/.test(formData.message)) {
   newErrors.message = "Message can only contain alphabets.";
  }

   //! Validation for subject

  if (!formData.subject.trim()) {
   newErrors.subject = "Subject is required.";
  } else if (formData.subject.length > 100) {
   newErrors.subject = "Subject cannot exceed 100 characters.";
  } else if (!/^[a-zA-Z\s!@#$%^&*(),.?":{}|<>]+$/.test(formData.subject)) {
   newErrors.subject = "Subject can only contain alphabets.";
  }
```



Environment Variables

- Stored sensitive API secret keys in .env file securely.



6. User Acceptance Testing (UAT):

- Browsing products.
- Searching with specific queries.
- Adding and removing items from the cart and wishlist.
- Completing a mock checkout process.
- Smooth foorm submission flows.
- Paginations check.

Improvements Required

- Overall user experience is smooth .
- Minor improvemnt in category filters required.

Conclusion:

Successfully focused on refining the marketplace application for deployment. All critical aspects, including functional testing, error handling, performance optimization, cross-browser and device compatibility, and security, were thoroughly addressed. The results ensured a user-friendly and robust system ready for real-world usage.With these enhancements, the marketplace application is well-equipped to handle customer interactions seamlessly and efficiently.