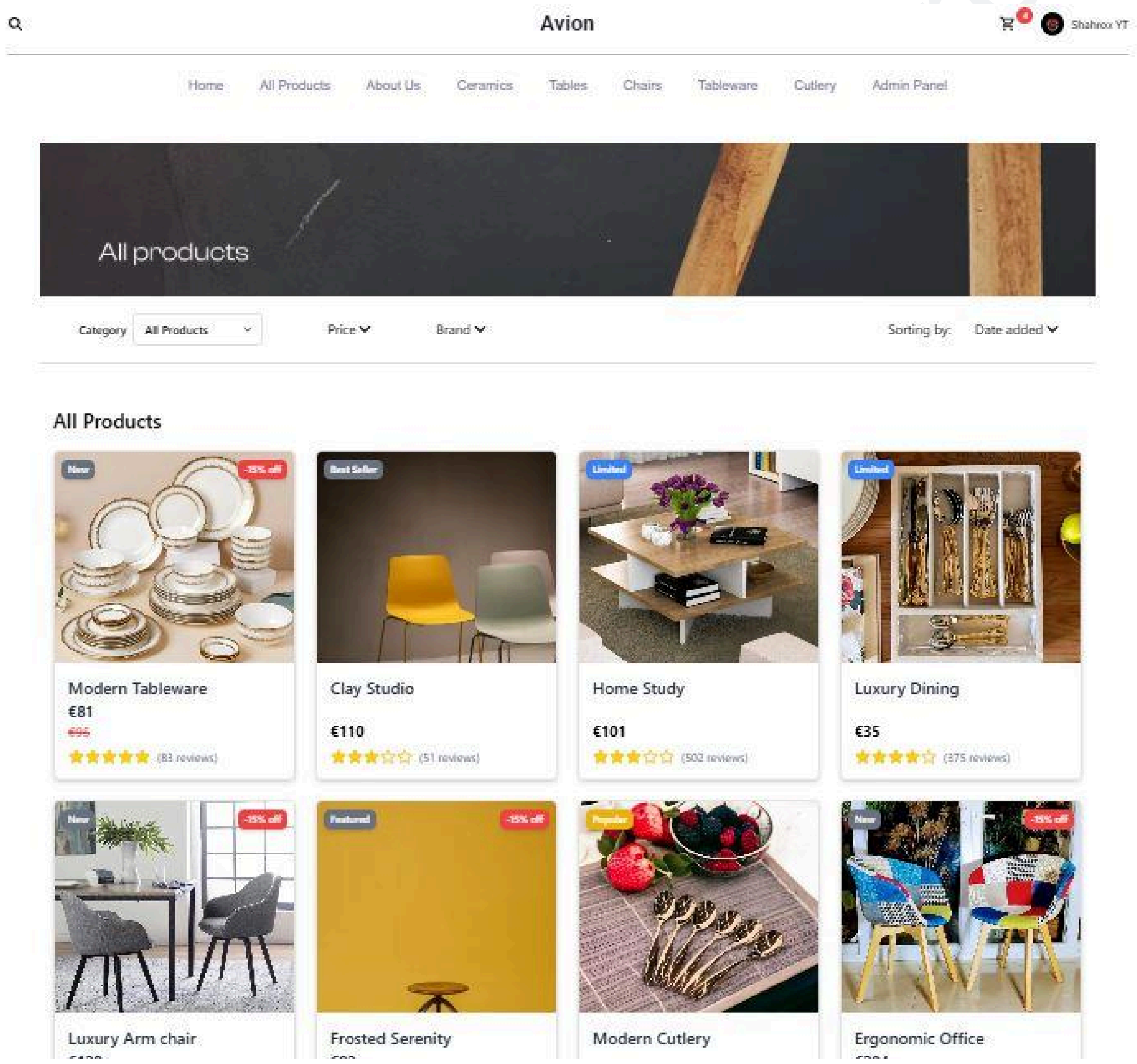
Step-01 Initialization and Testing

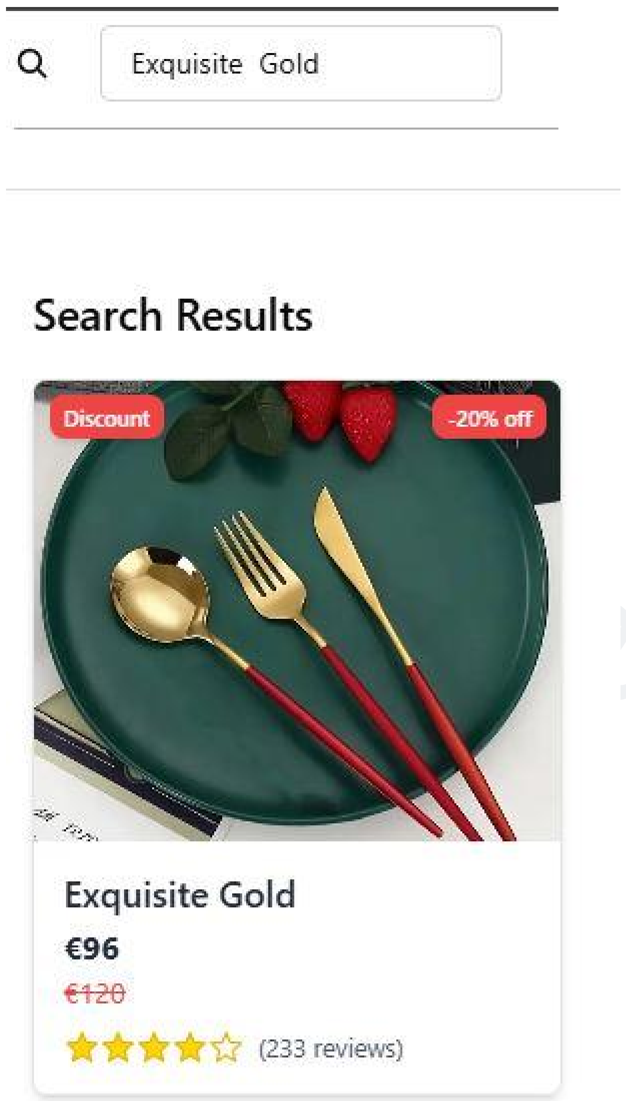
Test Core Features:

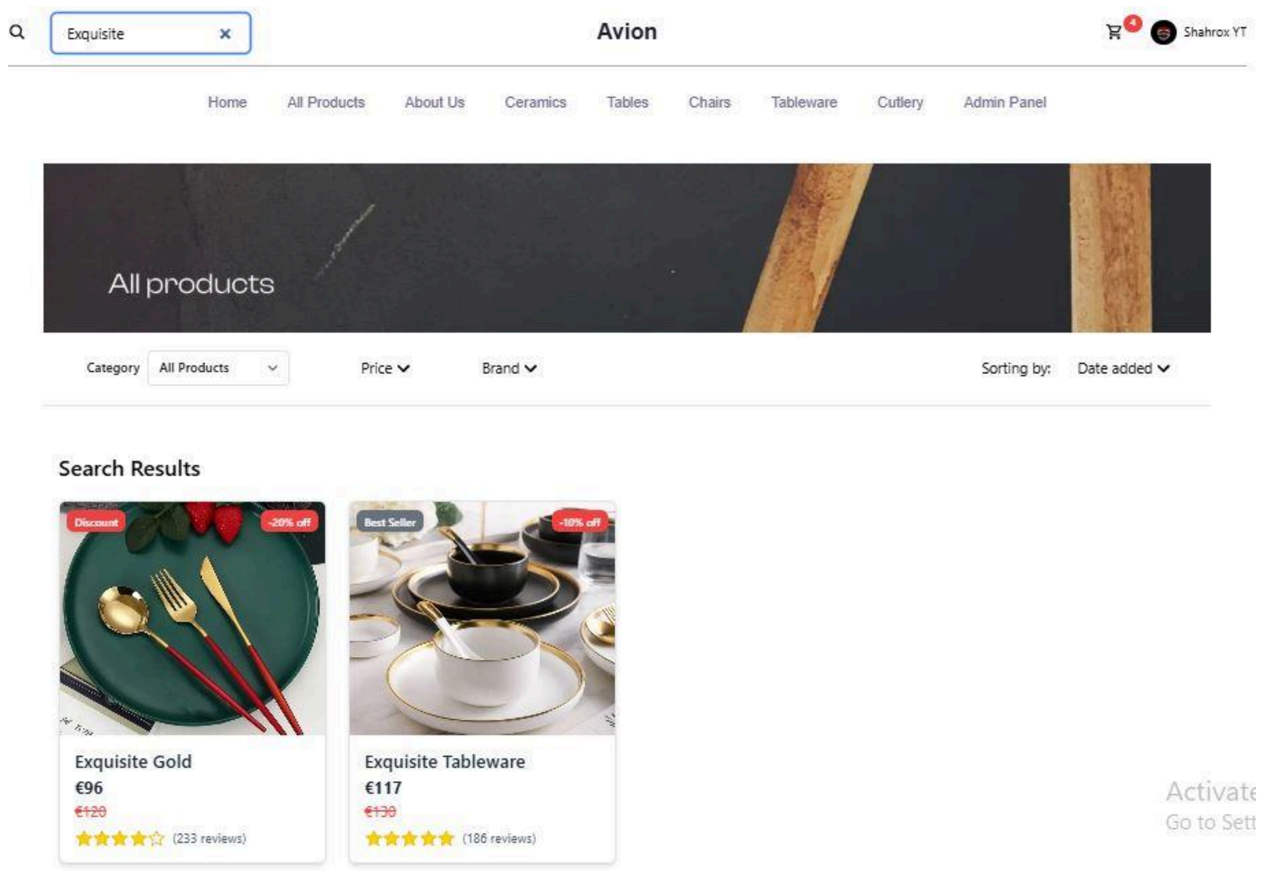
Checked Product Display on Homepage
Opened the homepage in the browser to ensure that the products are being displayed properly.



Filters and Search Functionality Testing

Ensured that the filters and search bar are displaying results accurately based on user input.

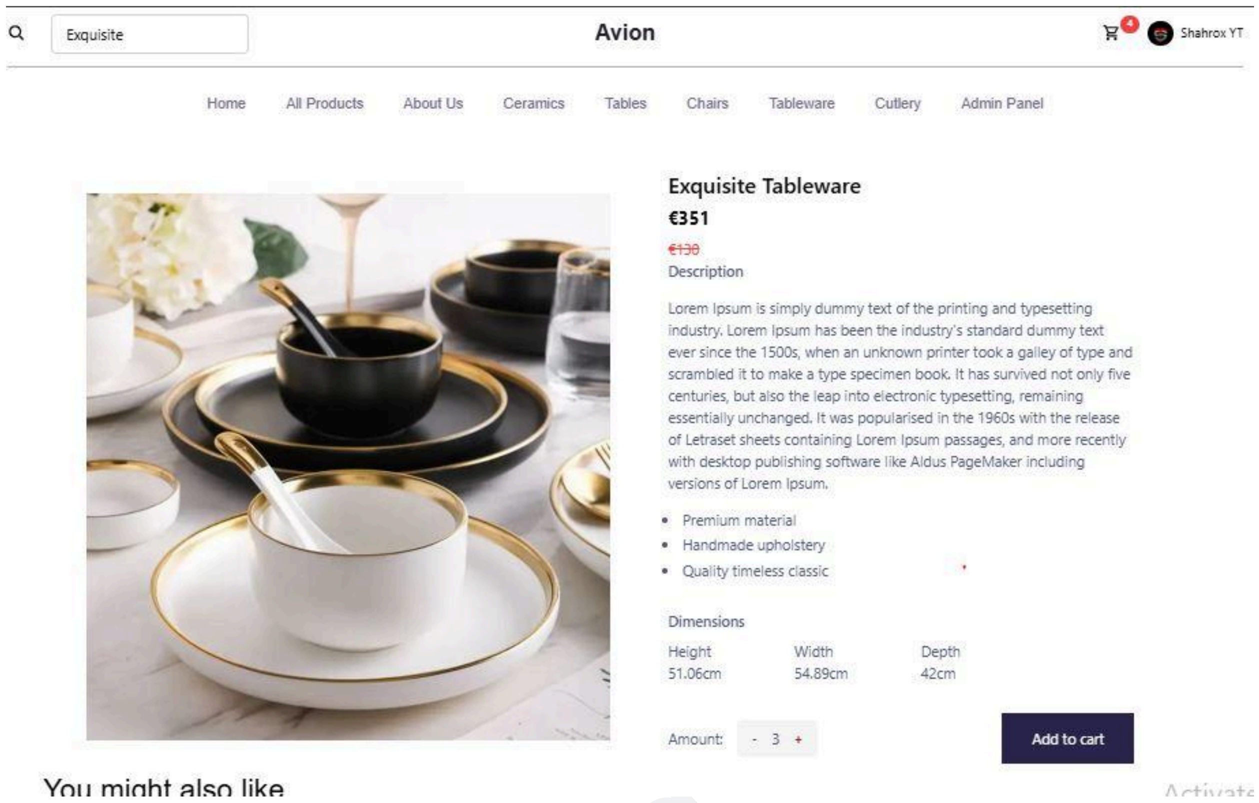




Exquisite Gold
€96
~~€120~~
(233 reviews)

Exquisite Tableware
€117
~~€130~~
(186 reviews)

Add to Cart Function: Verified the addition of products to the cart.



Exquisite Tableware

€351

~~€130~~

Description

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

- Premium material
- Handmade upholstery
- Quality timeless classic

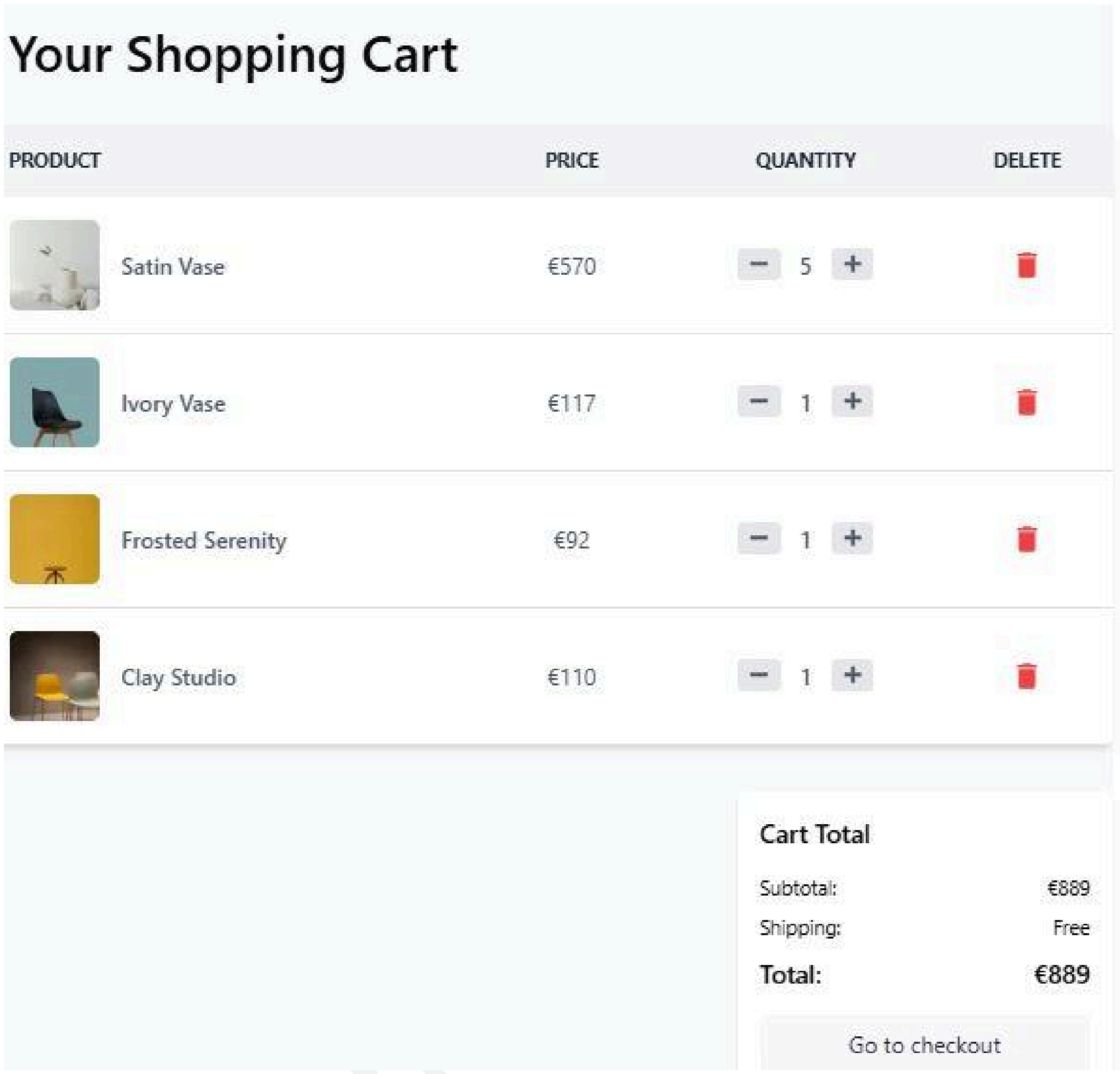
Dimensions

Height	Width	Depth
51.06cm	54.89cm	42cm

You might also like

Add to Cart Function: Verified that clicking the "Add to Cart" button successfully adds items to the cart.

Local Storage Integration: Used the `useAtom` hook from Jotai to store cart items in local storage. Verified that the cart data loaded correctly during page reloads without errors.

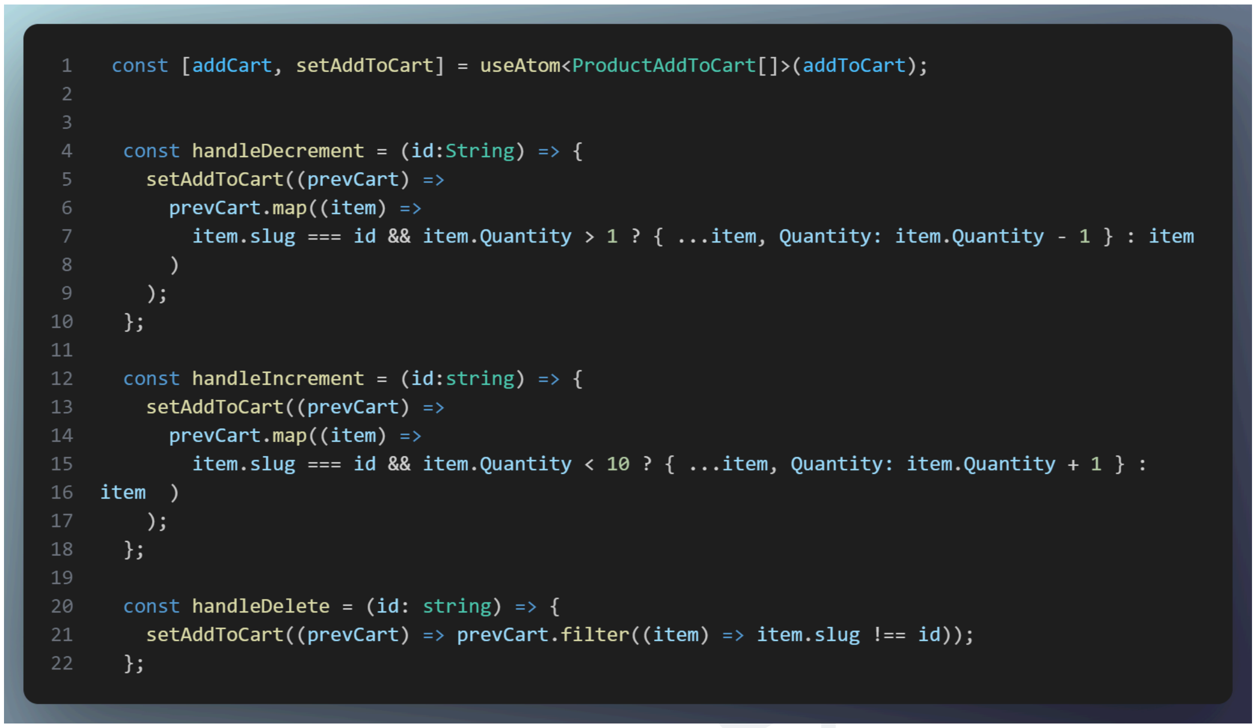

Your Shopping Cart

PRODUCT	PRICE	QUANTITY	DELETE
Satin Vase	€570	− 5 +	
Ivory Vase	€117	− 1 +	
Frosted Serenity	€92	− 1 +	
Clay Studio	€110	− 1 +	

Cart Total

Subtotal:	€889
Shipping:	Free
Total:	**€889**

Handle Delete Function: Ensured items could be removed from the cart without any issues.
Increment and Decrement Functions: Tested the functionality to increase or decrease the product quantity in the cart.
Product Quantity Validation: Confirmed that product quantities update accurately and reflect in real time.**Cart Functionality:** Confirmed that price

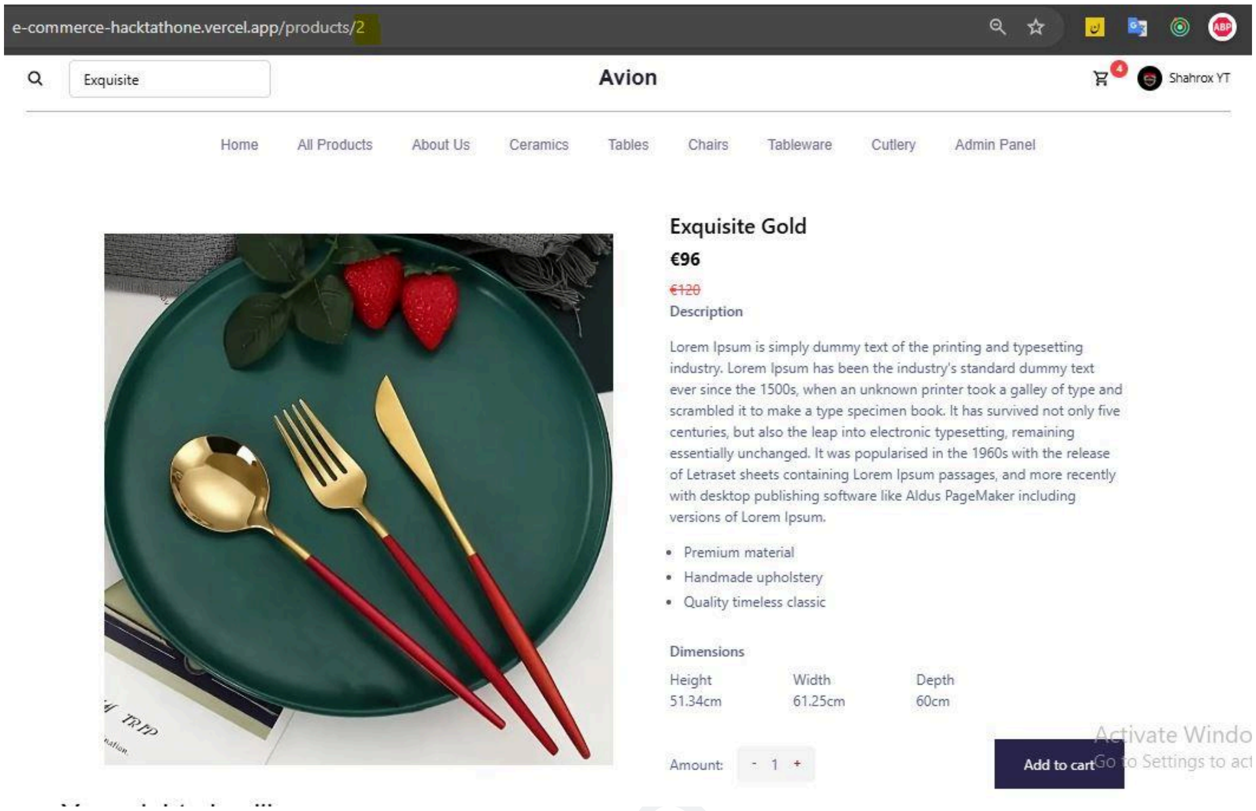


```
const [addCart, setAddToCart] = useAtom<ProductAddToCart[]>(addToCart);


  const handleDecrement = (id:String) => {
    setAddToCart((prevCart) =>
      prevCart.map((item) =>
        item.slug === id && item.Quantity > 1 ? { ...item, Quantity: item.Quantity - 1 } : item
      )
    );
  };

  const handleIncrement = (id:string) => {
    setAddToCart((prevCart) =>
      prevCart.map((item) =>
        item.slug === id && item.Quantity < 10 ? { ...item, Quantity: item.Quantity + 1 } :
item  )
    );
  };

  const handleDelete = (id: string) => {
    setAddToCart((prevCart) => prevCart.filter((item) => item.slug !== id));
  };
```

calculations and automatic discounts worked properly. Discounts were displayed with a label, and the total price reflected the changes correctly.

Reducer Function: Ensured the reducer function managed the cart state accurately.

Dynamic Testing: Verified that no errors occurred during the dynamic testing of 12 projects, with all components loading properly.



Exquisite Gold

€96

~~€120~~

Description

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

- Premium material
- Handmade upholstery
- Quality timeless classic

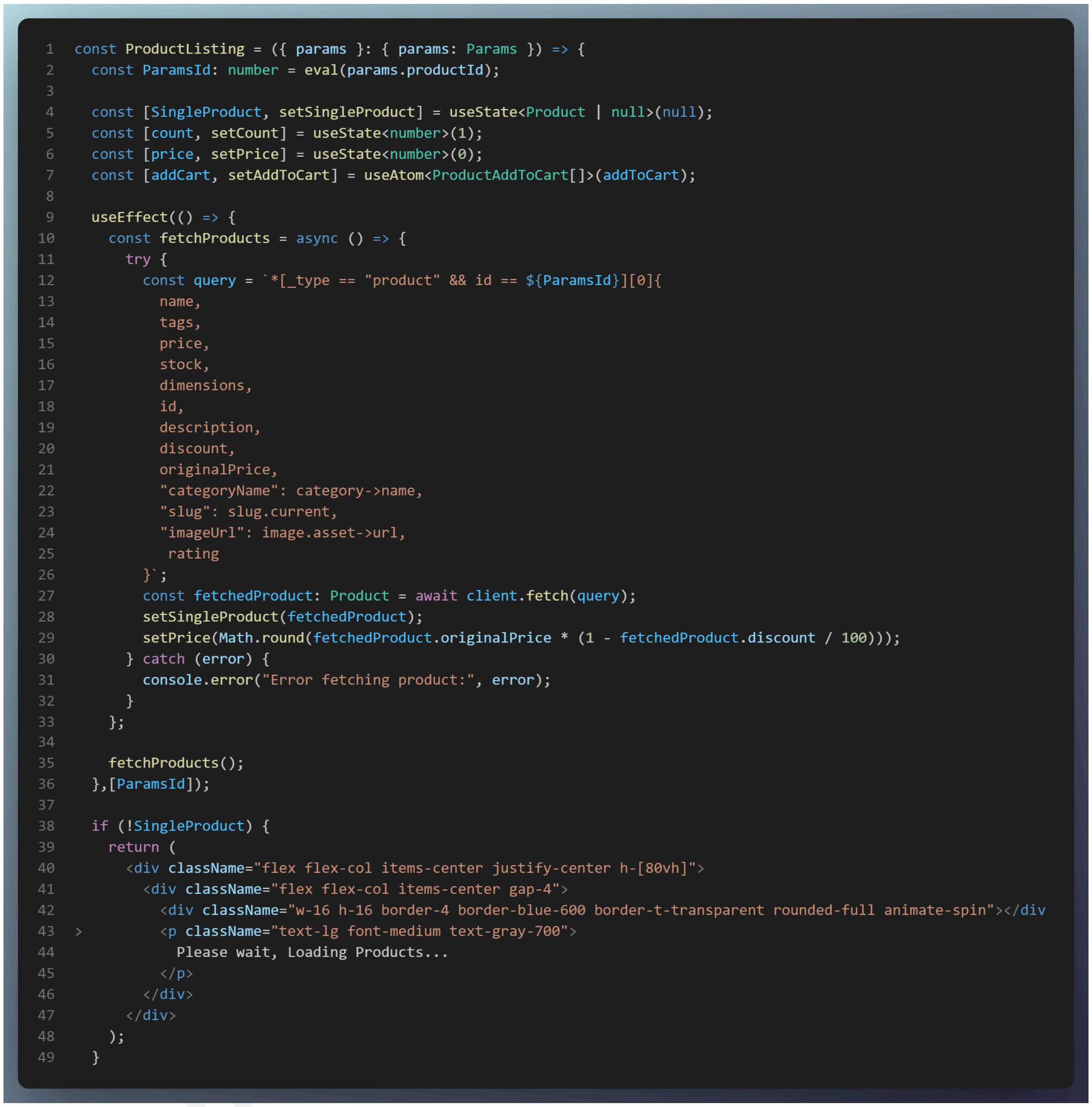
Dimensions

Height	Width	Depth
51.34cm	61.25cm	60cm

Amount: - 1 +

"Dynamic Data Fetch from Sanity"

Detail: This fetches content dynamically from Sanity, using queries to display the relevant information such as titles and descriptions.

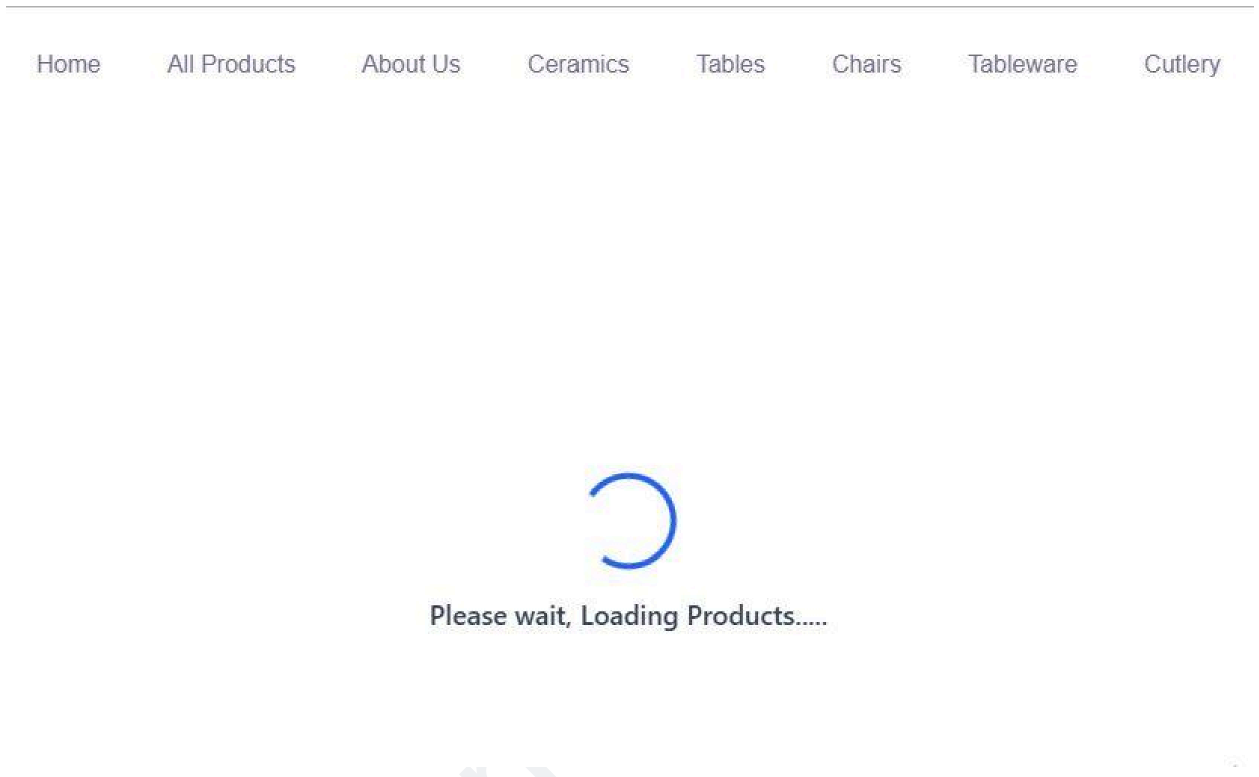


```tsx
const ProductListing = ({ params }: { params: Params }) => {
  const ParamsId: number = eval(params.productId);

  const [SingleProduct, setSingleProduct] = useState<Product | null>(null);
  const [count, setCount] = useState<number>(1);
  const [price, setPrice] = useState<number>(0);
  const [addCart, setAddToCart] = useAtom<ProductAddToCart[]>(addToCart);

  useEffect(() => {
    const fetchProducts = async () => {
      try {
        const query = `*[_type == "product" && id == ${ParamsId}][0]{
          name,
          tags,
          price,
          stock,
          dimensions,
          id,
          description,
          discount,
          originalPrice,
          "categoryName": category->name,
          "slug": slug.current,
          "imageUrl": image.asset->url,
           rating
        }`;
        const fetchedProduct: Product = await client.fetch(query);
        setSingleProduct(fetchedProduct);
        setPrice(Math.round(fetchedProduct.originalPrice * (1 - fetchedProduct.discount / 100)));
      } catch (error) {
        console.error("Error fetching product:", error);
      }
    };

    fetchProducts();
  },[ParamsId]);

  if (!SingleProduct) {
    return (
      <div className="flex flex-col items-center justify-center h-[80vh]">
        <div className="flex flex-col items-center gap-4">
          <div className="w-16 h-16 border-4 border-blue-600 border-t-transparent rounded-full animate-spin"></div
          <p className="text-lg font-medium text-gray-700">
            Please wait, Loading Products...
          </p>
        </div>
      </div>
    );
  }
```

Dynamic Data Fetch

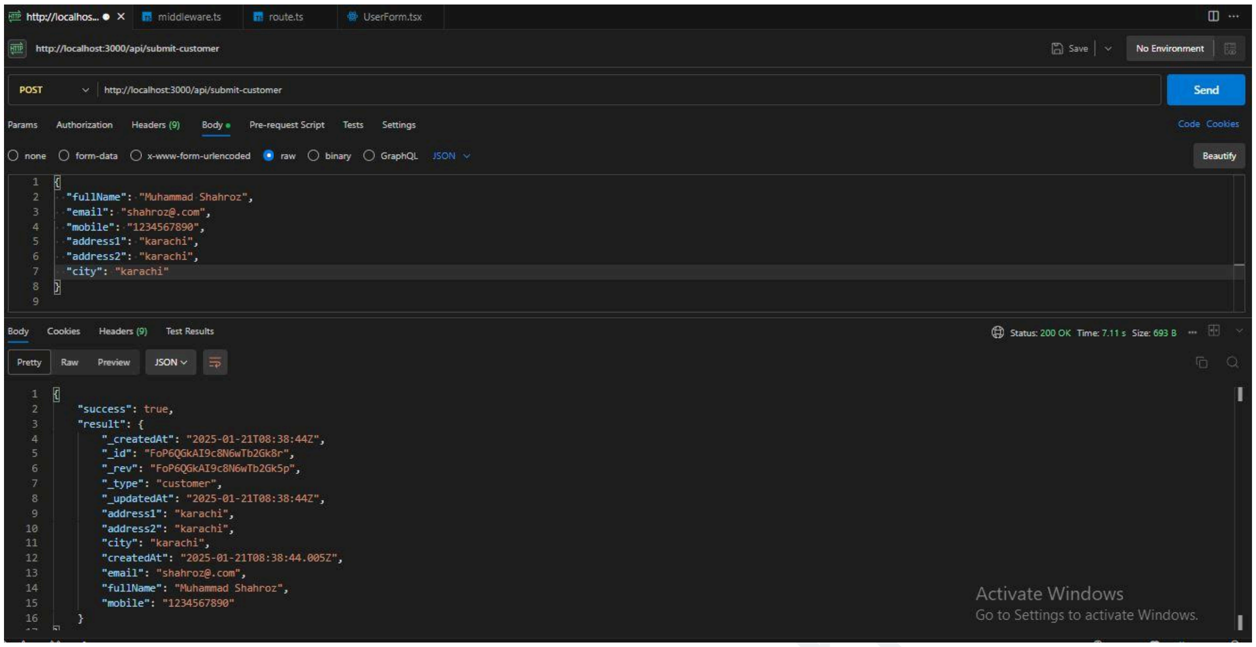
Detail: This process fetches content dynamically from Sanity and displays it after loading. While the data is being fetched, a loading state is shown to avoid errors or incomplete information.

Loading State: Implemented a loader to display while data was being fetched to avoid errors or interruptions during the loading process..



Postman API Testing

Tested APIs using Postman to verify their functionality and response accuracy.



http://localhost:3000/api/submit-customer

POST http://localhost:3000/api/submit-customer

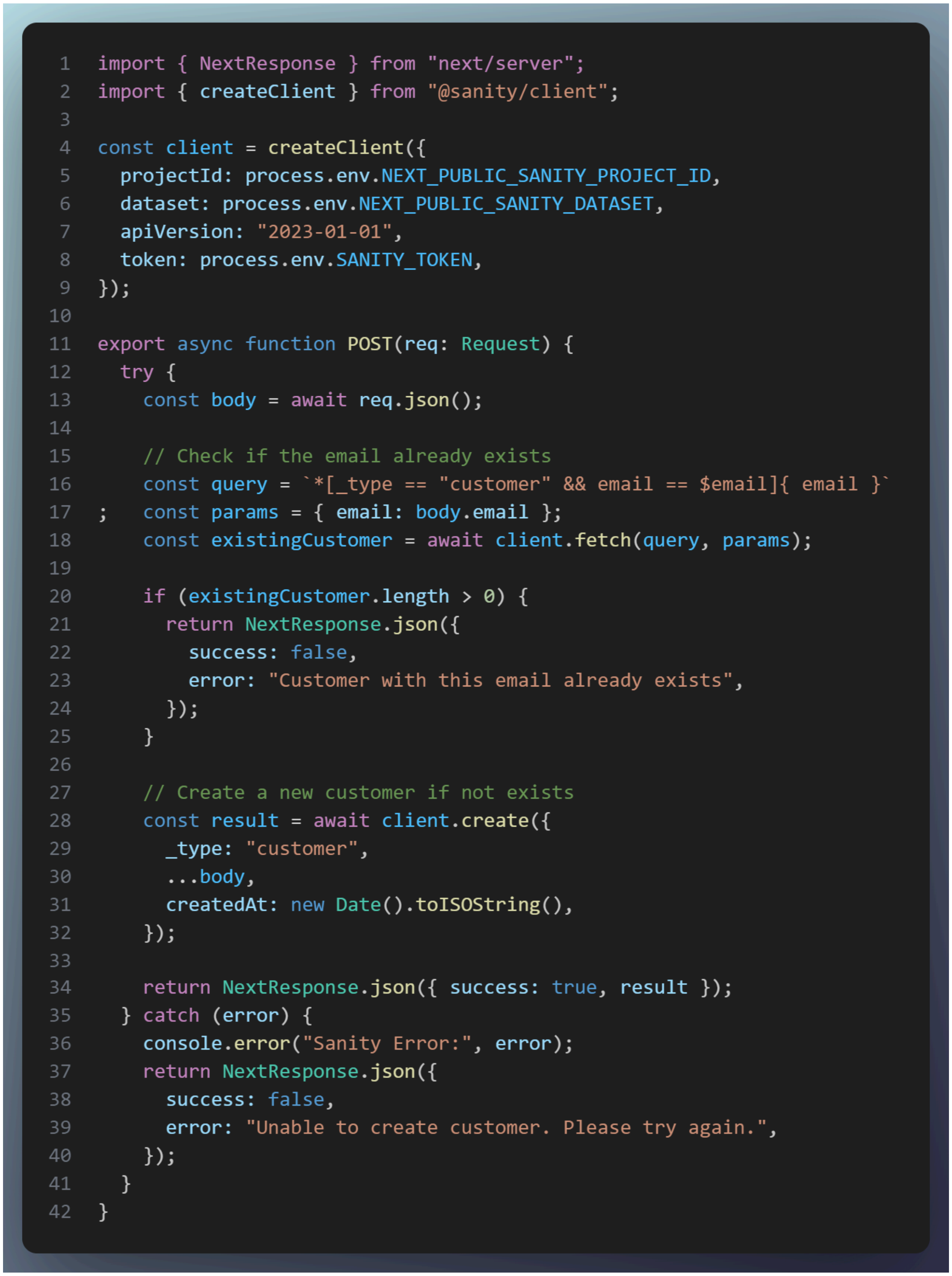
```
{
  "fullName": "Muhammad Shahroz",
  "email": "shahroz@.com",
  "mobile": "1234567890",
  "address1": "karachi",
  "address2": "karachi",
  "city": "karachi"
}
```

Status: 200 OK Time: 7.11 s Size: 693 B

```
{
    "success": true,
    "result": {
        "_createdAt": "2025-01-21T08:38:44Z",
        "_id": "FoP6QGkAI9c8N6wTb2Gk8r",
        "_rev": "FoP6QGkAI9c8N6wTb2Gk5p",
        "_type": "customer",
        "_updatedAt": "2025-01-21T08:38:44Z",
        "address1": "karachi",
        "address2": "karachi",
        "city": "karachi",
        "createdAt": "2025-01-21T08:38:44.005Z",
        "email": "shahroz@.com",
        "fullName": "Muhammad Shahroz",
        "mobile": "1234567890"
    }
```

Collecting Data from User Form:

- collected user-specific data dynamically from the form.



```typescript
import { NextResponse } from "next/server";
import { createClient } from "@sanity/client";

const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  apiVersion: "2023-01-01",
  token: process.env.SANITY_TOKEN,
});

export async function POST(req: Request) {
  try {
    const body = await req.json();

    // Check if the email already exists
    const query = `*[_type == "customer" && email == $email]{ email }`
;   const params = { email: body.email };
    const existingCustomer = await client.fetch(query, params);

    if (existingCustomer.length > 0) {
      return NextResponse.json({
        success: false,
        error: "Customer with this email already exists",
      });
    }

    // Create a new customer if not exists
    const result = await client.create({
      _type: "customer",
      ...body,
      createdAt: new Date().toISOString(),
    });

    return NextResponse.json({ success: true, result });
  } catch (error) {
    console.error("Sanity Error:", error);
    return NextResponse.json({
      success: false,
      error: "Unable to create customer. Please try again.",
    });
  }
}
```

Creating Customer in Sanity Dataset:

- inserted the user data directly into the **Sanity CMS** dataset without defining a schema, used `client.create()` to create customer documents directly.

Email Validation:

- To avoid creating duplicate customers, implemented email validation. This ensures that if a customer with the same email already exists, no new customer is created.

Duplicate Customer Check:

- When creating a customer, queried **Sanity** based on the email field to check if a customer with the same email already exists. If a duplicate customer is found, the new customer creation is prevented.

Implementation Review:

- **Dynamic Data Handling**: efficiently handled dynamic data and stored it in **Sanity CMS** with validation to prevent duplicate entries.

- **Loader Display**: used a loader on the frontend to show a loading state while data is being fetched, ensuring a better user experience.

Error Handling in Data Fetching

1. Try-Catch Block for Error Handling

A try-catch block has been implemented to handle errors during data fetching. If there is an issue with fetching the data, the catch block manages the error.

2. User-Friendly Error Message

A clear and simple error message is shown to the user, indicating that something went wrong and providing them with an option to retry.

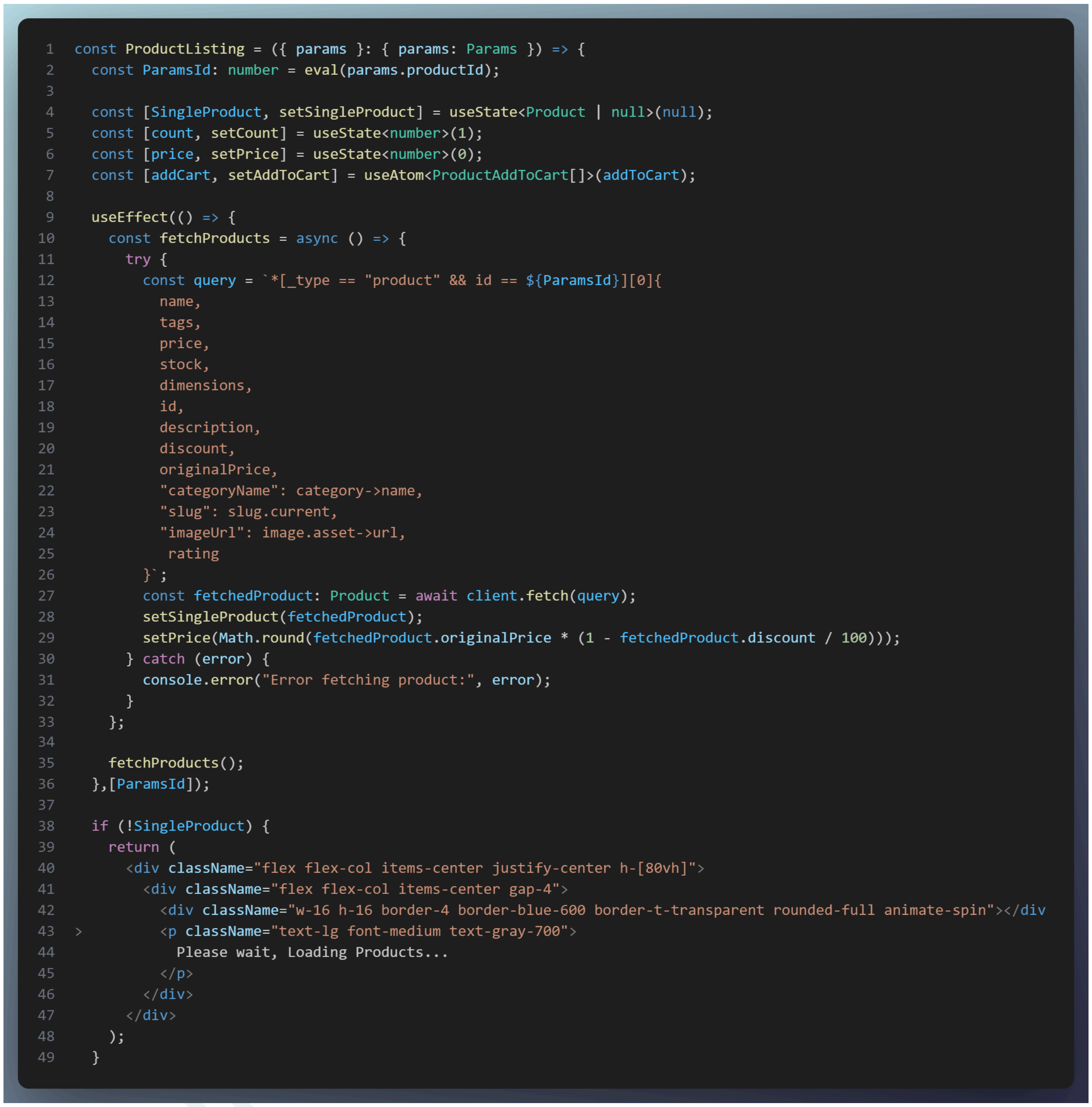
3. Developer Debugging via Console Logging

Console.error is used to log detailed error information for developers, which helps in debugging.

4. Loading State and Error Handling Flow

A loading state is managed, showing a loading message to the user while data is being fetched. Once the data is fetched successfully, the UI updates accordingly.

5. Retry Option (Optional)
An optional retry functionality is provided, allowing the user to attempt fetching the data again if an error occurs



```
const ProductListing = ({ params }: { params: Params }) => {
  const ParamsId: number = eval(params.productId);

  const [SingleProduct, setSingleProduct] = useState<Product | null>(null);
  const [count, setCount] = useState<number>(1);
  const [price, setPrice] = useState<number>(0);
  const [addCart, setAddToCart] = useAtom<ProductAddToCart[]>(addToCart);

  useEffect(() => {
    const fetchProducts = async () => {
      try {
        const query = `*[_type == "product" && id == ${ParamsId}][0]{
          name,
          tags,
          price,
          stock,
          dimensions,
          id,
          description,
          discount,
          originalPrice,
          "categoryName": category->name,
          "slug": slug.current,
          "imageUrl": image.asset->url,
           rating
        }`;
        const fetchedProduct: Product = await client.fetch(query);
        setSingleProduct(fetchedProduct);
        setPrice(Math.round(fetchedProduct.originalPrice * (1 - fetchedProduct.discount / 100)));
      } catch (error) {
        console.error("Error fetching product:", error);
      }
    };

    fetchProducts();
  },[ParamsId]);

  if (!SingleProduct) {
    return (
      <div className="flex flex-col items-center justify-center h-[80vh]">
        <div className="flex flex-col items-center gap-4">
          <div className="w-16 h-16 border-4 border-blue-600 border-t-transparent rounded-full animate-spin"></div
          <p className="text-lg font-medium text-gray-700">
            Please wait, Loading Products...
          </p>
        </div>
      </div>
    );
  }
```

Performance Testing

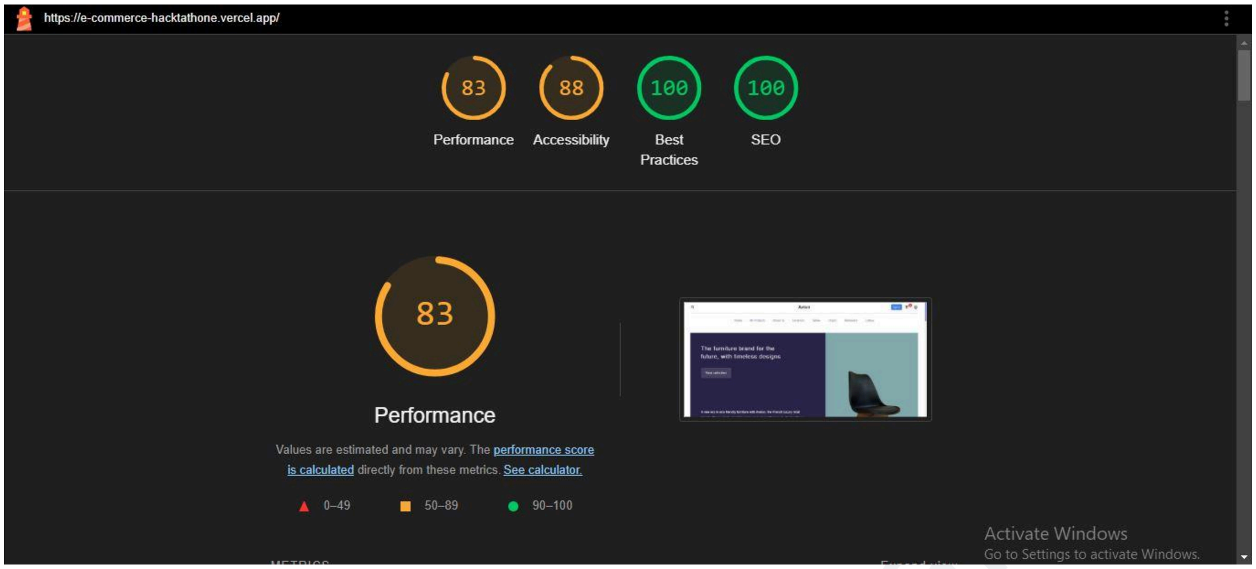
1. Load Testing

The performance of the application was tested by simulating multiple users accessing the data at once to ensure the system can handle high traffic without crashing.

2. Response Time Measurement
The response time for data fetching was measured to ensure that it meets acceptable limits. If response times are high, optimizations are considered.

3. Data Fetching Efficiency
The efficiency of the data fetching process was tested, ensuring minimal delay in loading and smooth transitions between pages, especially when dealing with large datasets.

4. Optimizing Rendering
Performance optimizations were made to ensure that components re-render only when necessary, reducing unnecessary load on the browser.


https://e-commerce-hacktathone.vercel.app/

83	88	100	100
Performance	Accessibility	Best Practices	SEO

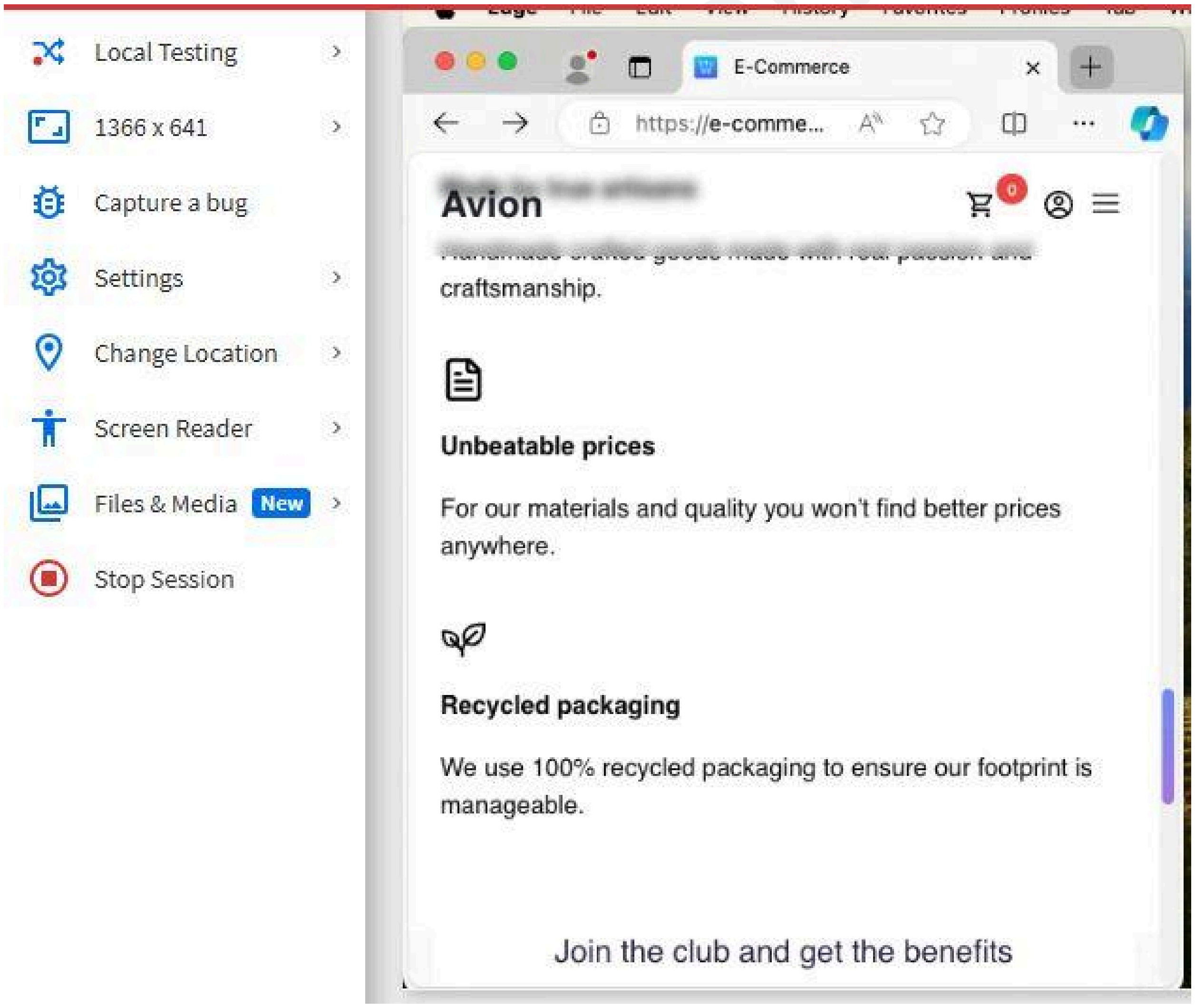
83

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

Microsoft Edge Browser



Avion

craftsmanship.

Unbeatable prices

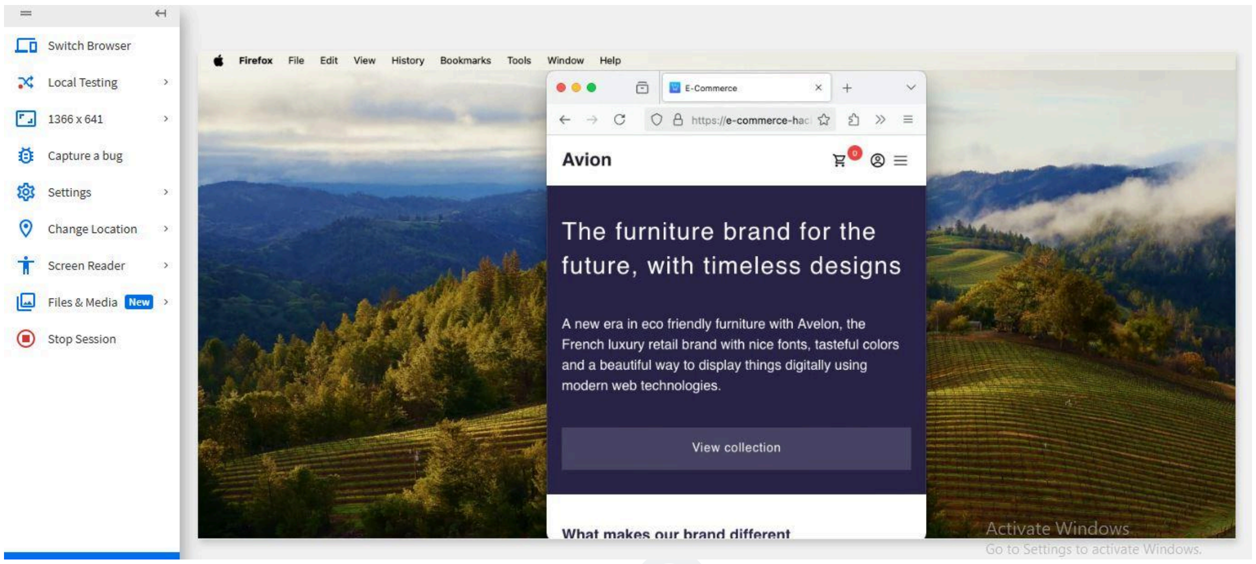
For our materials and quality you won't find better prices anywhere.

Recycled packaging

We use 100% recycled packaging to ensure our footprint is manageable.

Join the club and get the benefits

FireFox Browser
Cross-Browser and Device Testing

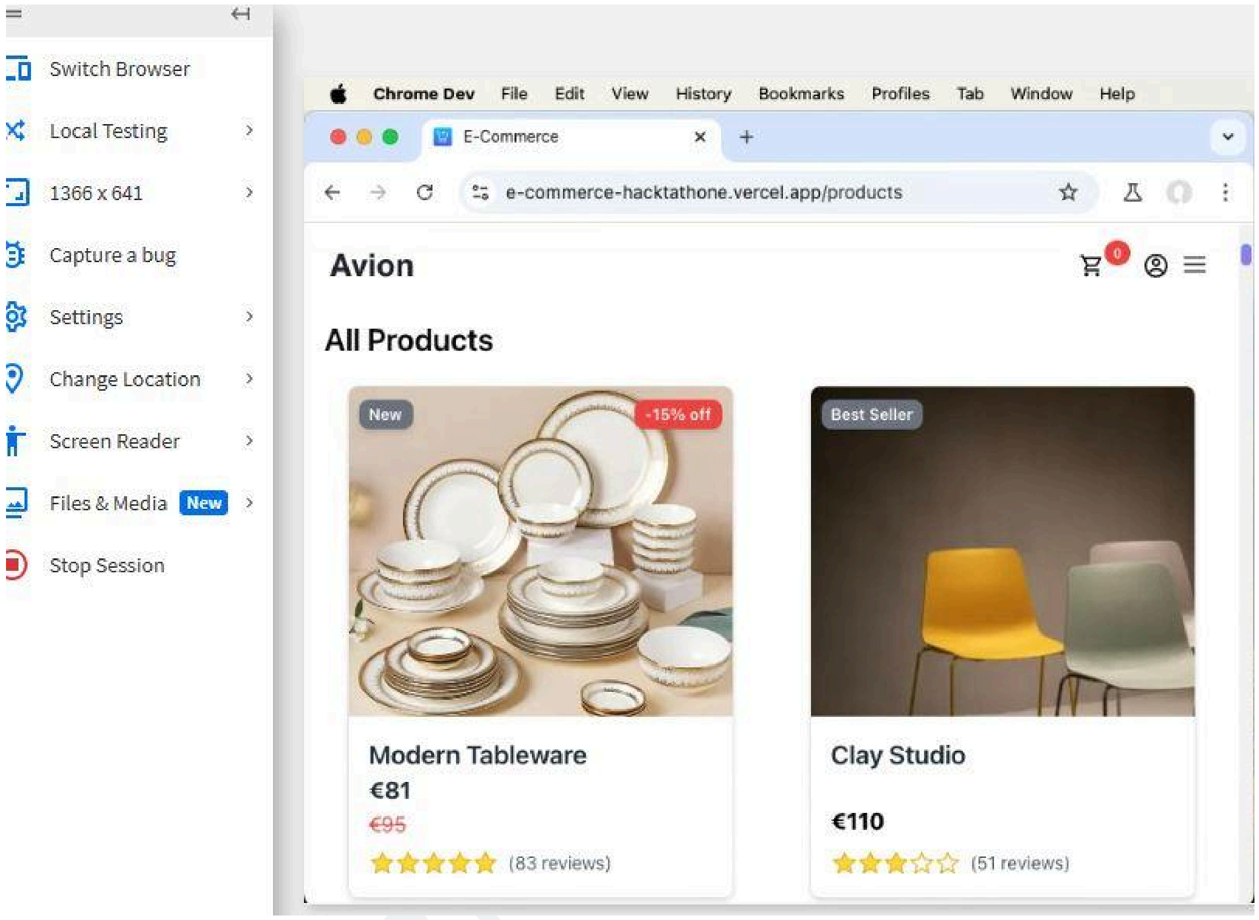


The furniture brand for the future, with timeless designs

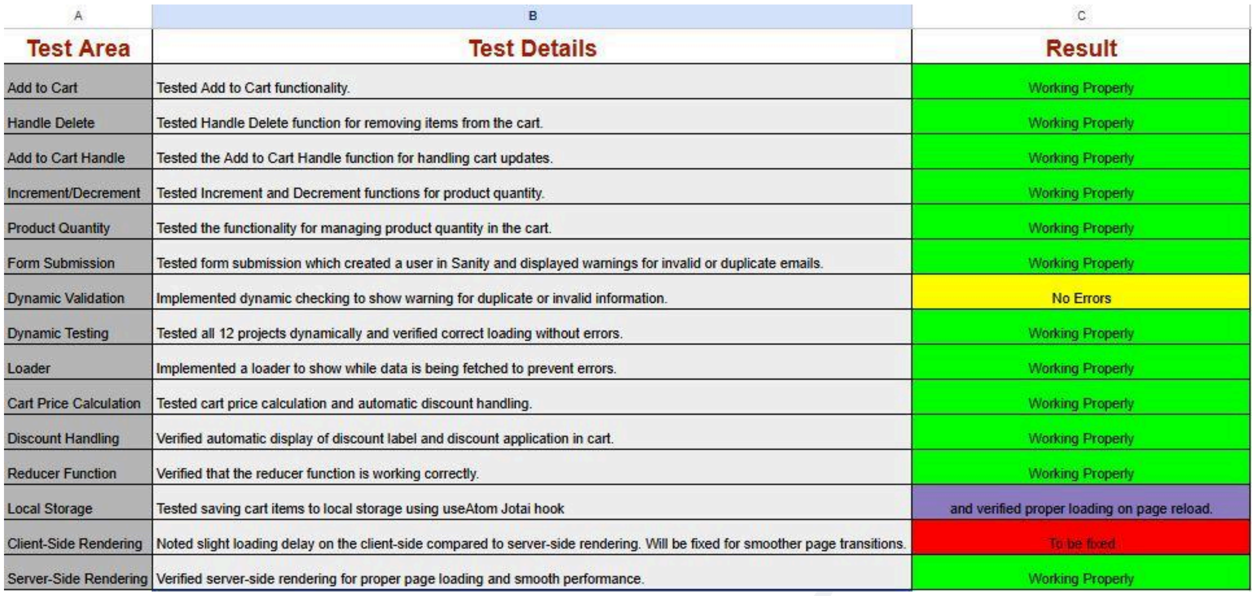
A new era in eco friendly furniture with Avelon, the French luxury retail brand with nice fonts, tasteful colors and a beautiful way to display things digitally using modern web technologies.

What makes our brand different

Google Chrome

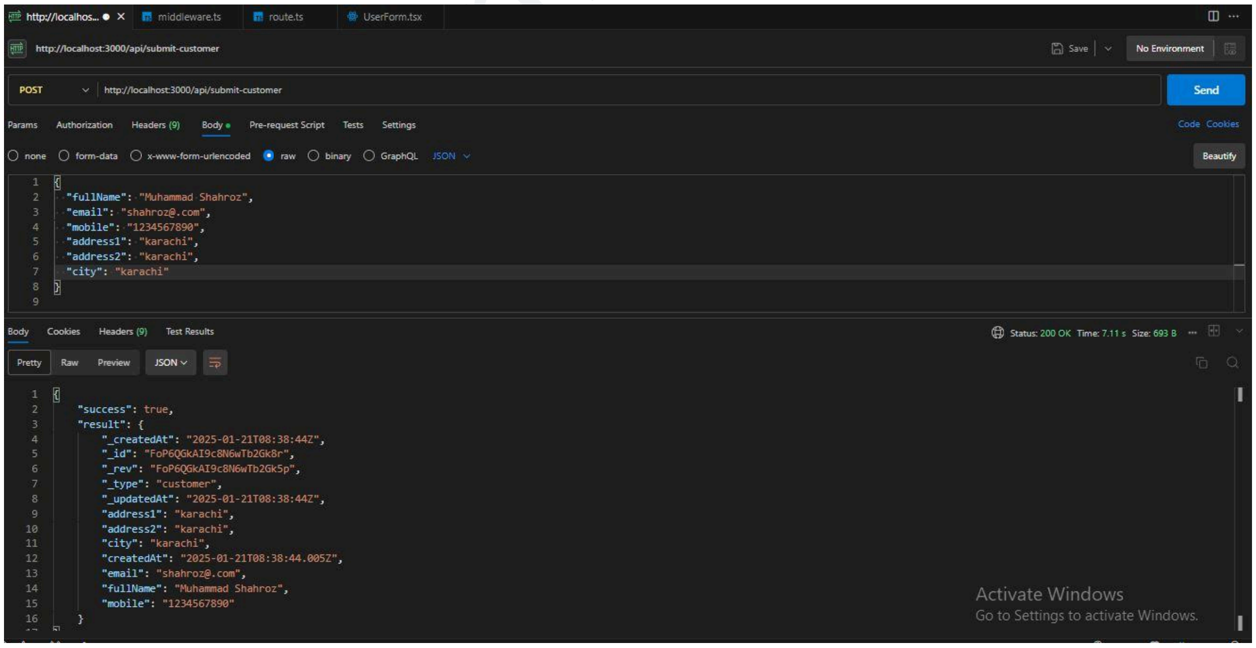


Functions Testing Report.xlsx



Test Area	Test Details	Result
Add to Cart	Tested Add to Cart functionality.	Working Properly
Handle Delete	Tested Handle Delete function for removing items from the cart.	Working Properly
Add to Cart Handle	Tested the Add to Cart Handle function for handling cart updates.	Working Properly
Increment/Decrement	Tested Increment and Decrement functions for product quantity.	Working Properly
Product Quantity	Tested the functionality for managing product quantity in the cart.	Working Properly
Form Submission	Tested form submission which created a user in Sanity and displayed warnings for invalid or duplicate emails.	Working Properly
Dynamic Validation	Implemented dynamic checking to show warning for duplicate or invalid information.	No Errors
Dynamic Testing	Tested all 12 projects dynamically and verified correct loading without errors.	Working Properly
Loader	Implemented a loader to show while data is being fetched to prevent errors.	Working Properly
Cart Price Calculation	Tested cart price calculation and automatic discount handling.	Working Properly
Discount Handling	Verified automatic display of discount label and discount application in cart.	Working Properly
Reducer Function	Verified that the reducer function is working correctly.	Working Properly
Local Storage	Tested saving cart items to local storage using useAtom Jotai hook	and verified proper loading on page reload.
Client-Side Rendering	Noted slight loading delay on the client-side compared to server-side rendering. Will be fixed for smoother page transitions.	To be fixed
Server-Side Rendering	Verified server-side rendering for proper page loading and smooth performance.	Working Properly

Extra

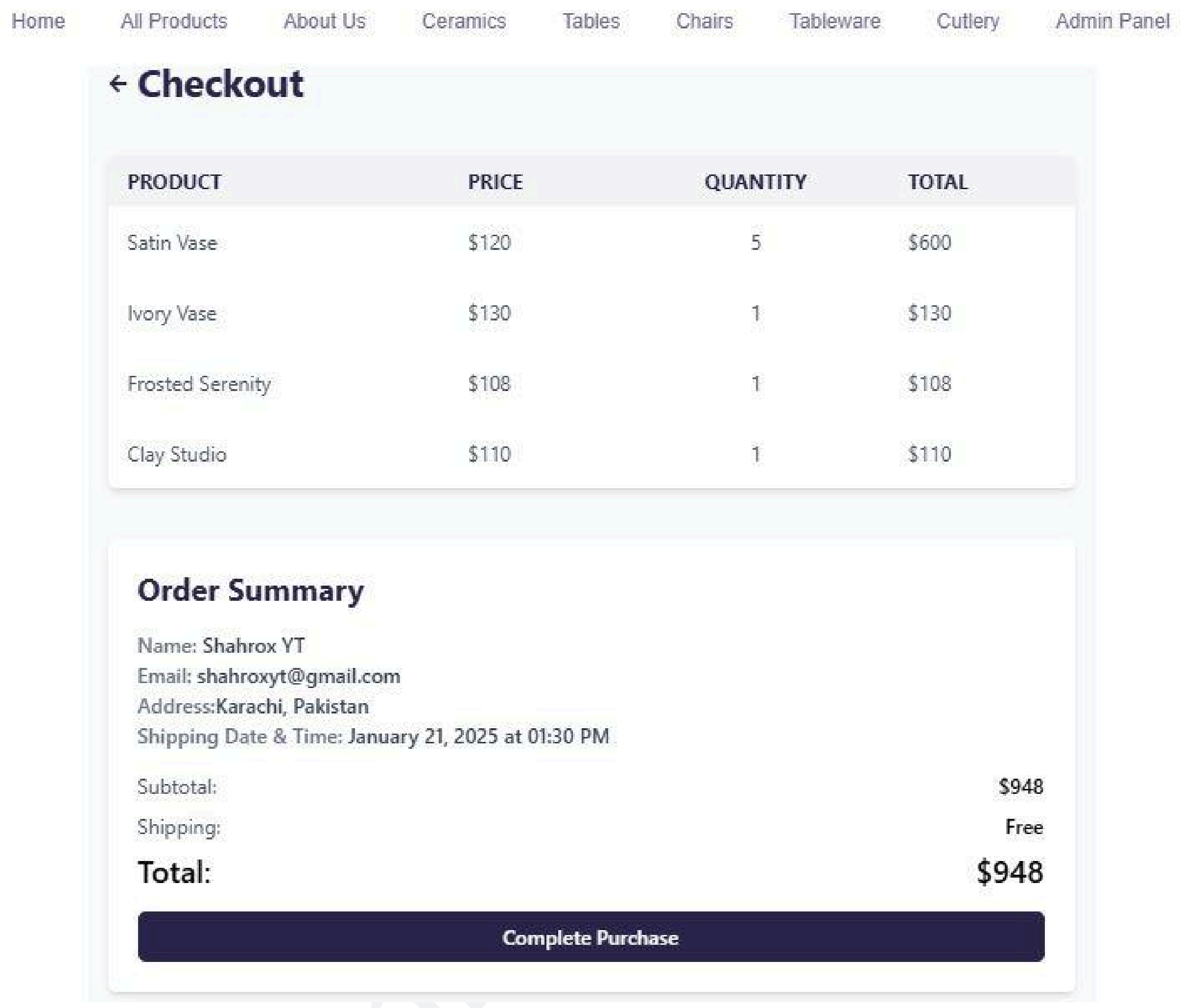


POST http://localhost:3000/api/submit-customer

```
{
  "fullName": "Muhammad Shahroz",
  "email": "shahroz@.com",
  "mobile": "1234567890",
  "address1": "karachi",
  "address2": "karachi",
  "city": "karachi"
}
```

Status: 200 OK Time: 7.11 s Size: 693 B

```
{
    "success": true,
    "result": {
        "_createdAt": "2025-01-21T08:38:44Z",
        "_id": "FoP6QGkAI9c8N6wTb2Gk8r",
        "_rev": "FoP6QGkAI9c8N6wTb2Gk5p",
        "_type": "customer",
        "_updatedAt": "2025-01-21T08:38:44Z",
        "address1": "karachi",
        "address2": "karachi",
        "city": "karachi",
        "createdAt": "2025-01-21T08:38:44.005Z",
        "email": "shahroz@.com",
        "fullName": "Muhammad Shahroz",
        "mobile": "1234567890"
    }
```



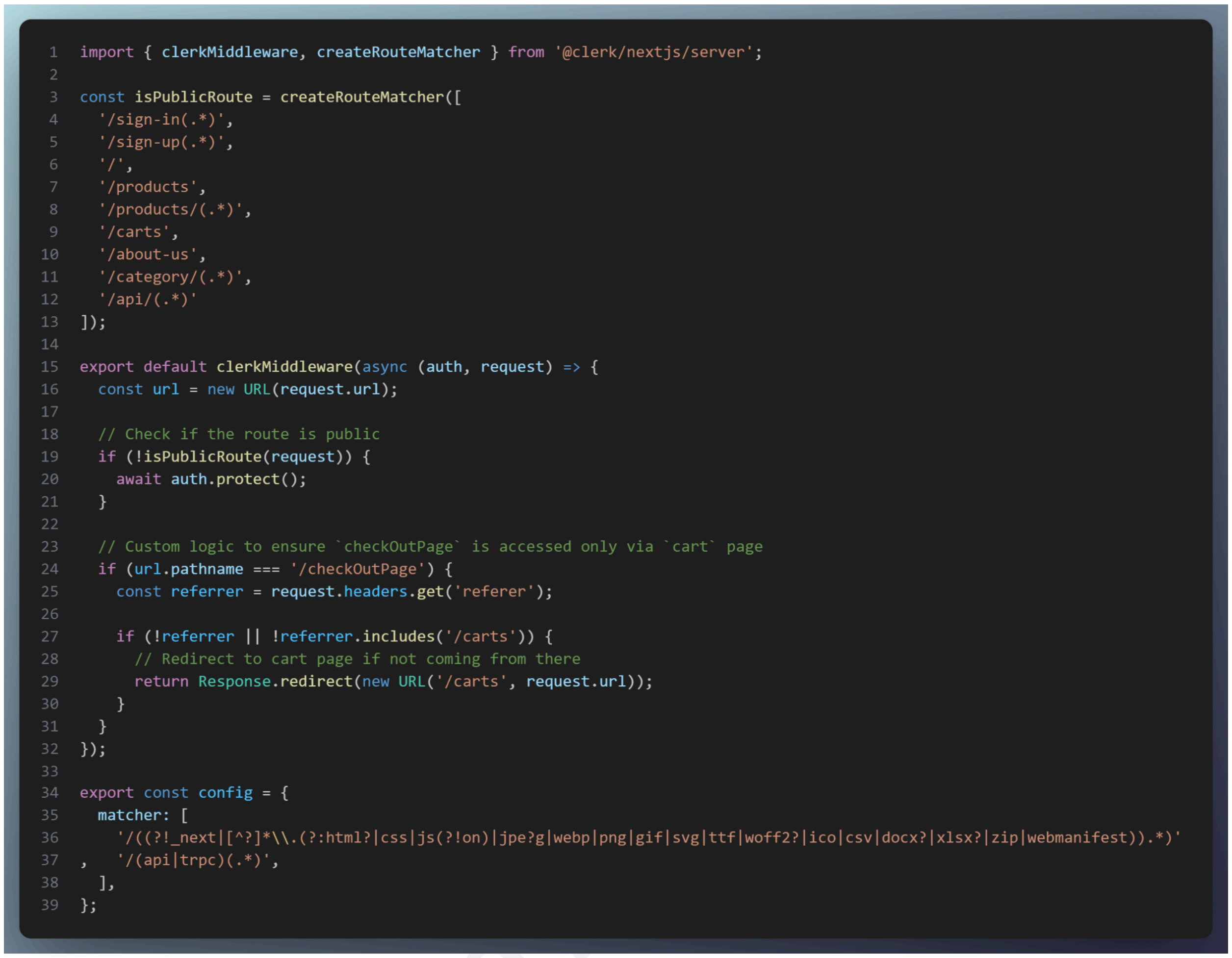
← Checkout

PRODUCT	PRICE	QUANTITY	TOTAL
Satin Vase	$120	5	$600
Ivory Vase	$130	1	$130
Frosted Serenity	$108	1	$108
Clay Studio	$110	1	$110

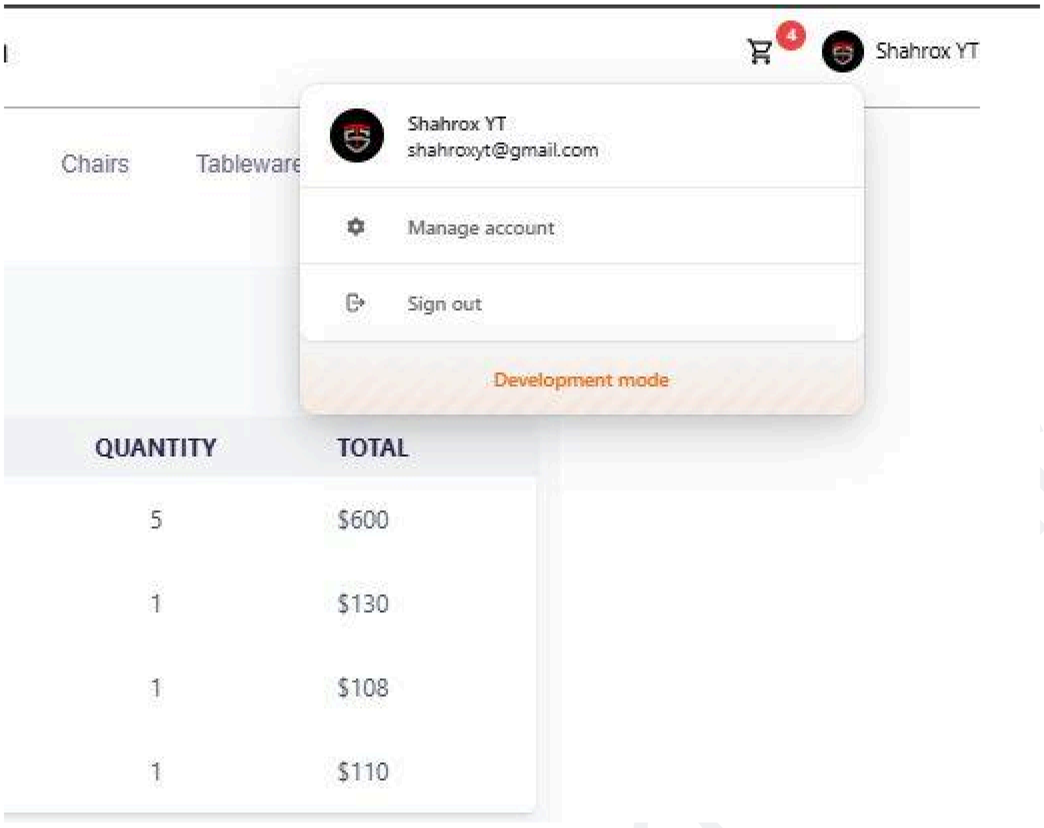
Order Summary

Name: Shahrox YT
Email: shahroxyt@gmail.com
Address:Karachi, Pakistan
Shipping Date & Time: January 21, 2025 at 01:30 PM

Subtotal:	$948
Shipping:	Free
Total:	**$948**



```
import { clerkMiddleware, createRouteMatcher } from '@clerk/nextjs/server';

const isPublicRoute = createRouteMatcher([
  '/sign-in(.*)',
  '/sign-up(.*)',
  '/',
  '/products',
  '/products/(.*)',
  '/carts',
  '/about-us',
  '/category/(.*)',
  '/api/(.*)'
]);

export default clerkMiddleware(async (auth, request) => {
  const url = new URL(request.url);

  // Check if the route is public
  if (!isPublicRoute(request)) {
    await auth.protect();
  }

  // Custom logic to ensure `checkOutPage` is accessed only via `cart` page
  if (url.pathname === '/checkOutPage') {
    const referrer = request.headers.get('referer');

    if (!referrer || !referrer.includes('/carts')) {
      // Redirect to cart page if not coming from there
      return Response.redirect(new URL('/carts', request.url));
    }
  }
});

export const config = {
  matcher: [
    '/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*)'
,   '/(api|trpc)(.*)',
  ],
};
```



QUANTITY	TOTAL
5	$600
1	$130
1	$108
1	$110