○

Marketplace Technical Website SHOP.CO BV SJ

Day 1: Establishing Marketplace Foundations

Objective:
To lay the groundwork for the marketplace by defining core business strategies, understanding the target audience, identifying unique offerings, and conducting detailed market research to ensure a solid foundation for growth.

Key Components:

1. Business Goals:

- **Mission & Vision:** Define a clear purpose and vision for the marketplace to guide all decisions.
- **Objectives:** Establish short-term goals for immediate impact and long-term goals for sustainable growth.

2. Market Research:

- **Competitor Analysis:** Assess strengths and weaknesses of existing competitors to identify areas for differentiation.
- **Consumer Insights:** Analyze industry trends and customer preferences to uncover market demands and opportunities.
- **Gap Identification:** Highlight unfulfilled needs or inefficiencies in the market that the marketplace can address.

3. Target Audience & Value Proposition:

- **Demographics:** Identify the key characteristics and needs of the target audience.
- **Unique Selling Point:** Clearly articulate what sets the marketplace apart, offering a compelling reason for customers to choose it over competitors.

4. Drafting the Data Schema:

- **Key Entities:** Identify core components such as products, users, and transactions.
- **Relationships:** Create a logical map of how these entities interact within the system to ensure seamless integration.

Submission Requirements:

Document Title:

- "Marketplace Strategic Blueprint - [Your Marketplace Name]"

Submission Must Include:

1. A comprehensive description of the marketplace's purpose and the problem it solves.
2. Detailed insights into the target audience and the unique value proposition.
3. Market research outcomes, including competitor analysis and identified opportunities.
4. A draft of the data schema (sketched or diagrammed) outlining key entities and relationships.

Upload Instructions:

- Save the document in the "Documentation" folder within your project repository.

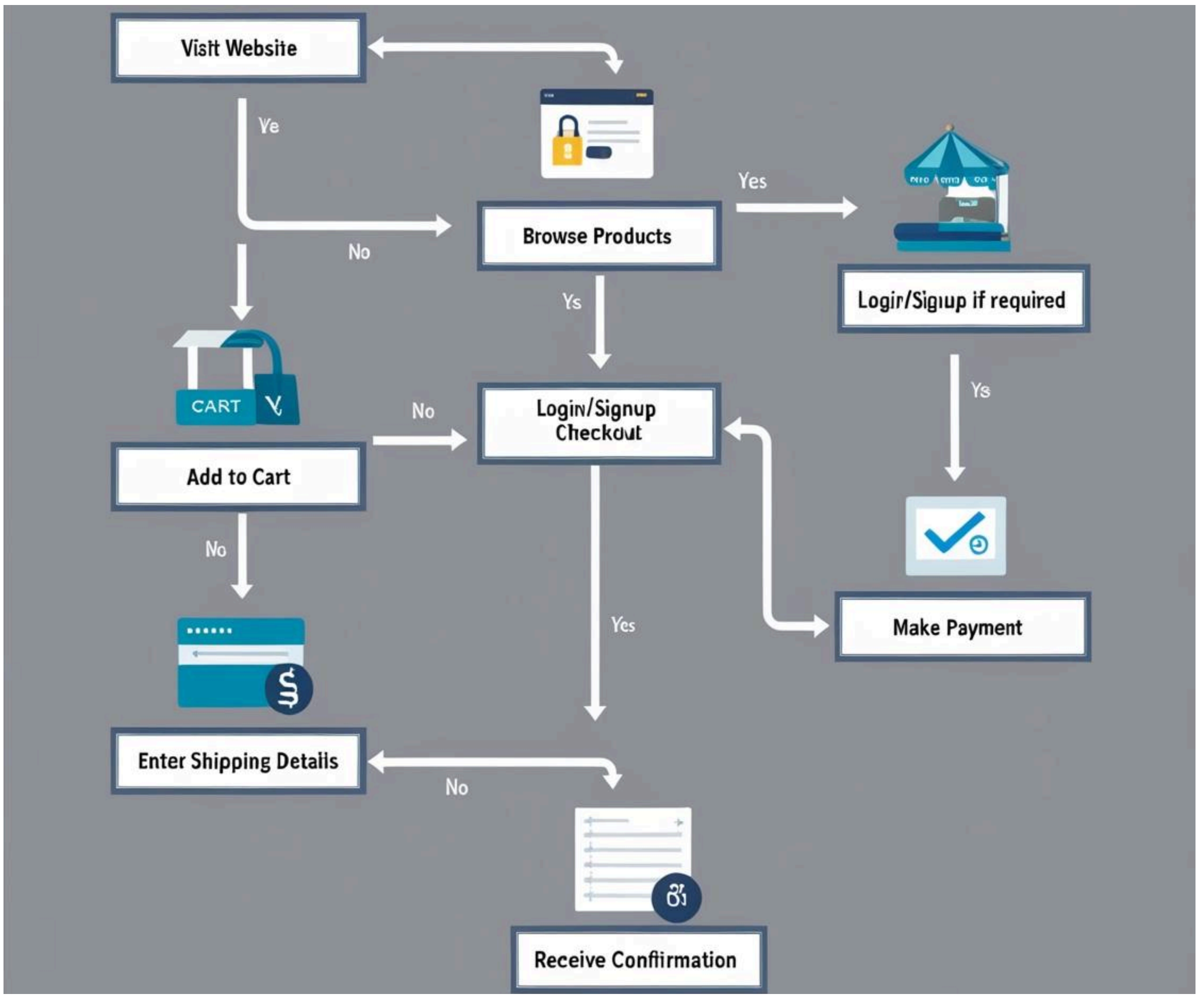
Marketplace Technical Foundation (Version D6V2)

System Architecture Overview

Diagram:

```
[Frontend (Next.js)]
     |
[Sanity CMS] <      > [Product Data API]
     |
[Third-Party APIs] <--> [Payment Gateway & Shipment Tracking]
```

Diagram 2:



Components:

- **Frontend (Next.js):** Handles user interactions and displays data fetched from the backend.
- **Sanity CMS:** Manages product, customer, and order data.
- **Third-Party APIs:** Integrates shipment tracking and payment processing services.

Data Flow Example:

1. User browses the marketplace on the frontend.
2. Frontend requests product data from the Sanity CMS via API.
3. User places an order; order details are sent to Sanity CMS.
4. Payment is processed through a third-party payment gateway.
5. Shipment status is fetched via a third-party API and displayed to the user.

1. Key Workflows

1.1 Product Browsing

1. User visits the product listing page.
2. Frontend fetches product categories and details from Sanity CMS.
3. Products are displayed dynamically.

1.2 Order Placement

1. User adds items to the cart and proceeds to checkout.
2. Order details are sent to the Sanity CMS.
3. Payment is processed through a third-party payment gateway.
4. Order confirmation is sent back to the user and stored in Sanity CMS.

1.3 Shipment Tracking

1. Order status updates are fetched from a third-party API.
2. Updates are displayed to the user in real-time.

---

2. API Requirements

Endpoint	Methd	Description	Request Payload	Response Example
`/product`	GET	Fetch product details	None	`{ "id": 1, "name": "Product A" }`
`/order`	POST			e r Fetch

Create a new

s h i p

o r d

`/shipment` GET

ment status	{ "customerId": 123, "items": [] }	{ "orderId": 456, "status": "Success" }
	{ "orderId": 456 }	{ "status": "In Transit", "ETA": "2 days" }

---

3. Sanity CMS Schema Example

Product SchemaOrder Schema

export default
{ name: 'order',
type: 'document',
fields: [{ name: 'customer', type: 'reference',
to: [{ type: 'customer' }], title: 'Customer' },
{ name: 'items', type: 'array', of: [{ type: 'object', fields: [{ name: 'product', type: 'reference', to: [{ type: 'product' }], title: 'Product' },
{ name: 'quantity',
 type: 'number',
title: 'Quantity' },], },],
 title: 'Ordered Items', },
{ name: 'totalPrice',
type: 'number',
title: 'Total Price',
validation: (Rule) => Rule.min(0).precision(2) }, { name: 'status', type: 'string', title: 'Order Status', options: { list: ['Pending', 'Shipped', 'Delivered'] } },], };

---

4. Collaborative Notes

- Feedback was collected from peers on improving system architecture clarity.
- Suggestions were incorporated to add detailed API payload examples.
- Real-time tracking workflows were discussed and refined.

Next Steps

1. Begin the implementation of the Sanity CMS schemas.
2. Integrate APIs with the frontend.
3. Test workflows end-to-end.

API Integration & Frontend Implementation

This report outlines the process of importing API data, testing its functionality, integrating it into the frontend, and applying basic styling. Screenshots and key queries are included to ensure clarity and provide a comprehensive understanding of the workflow.

Step 1: Importing API Data

Actions Taken:

1. Selected the **Template 1 API** and set it up in a new folder.
2. Opened the folder in **VS Code** and created a script `importData.js` to handle data import.
3. Connected to **Sanity CMS** and ensured the environment variables were correctly set for secure access.

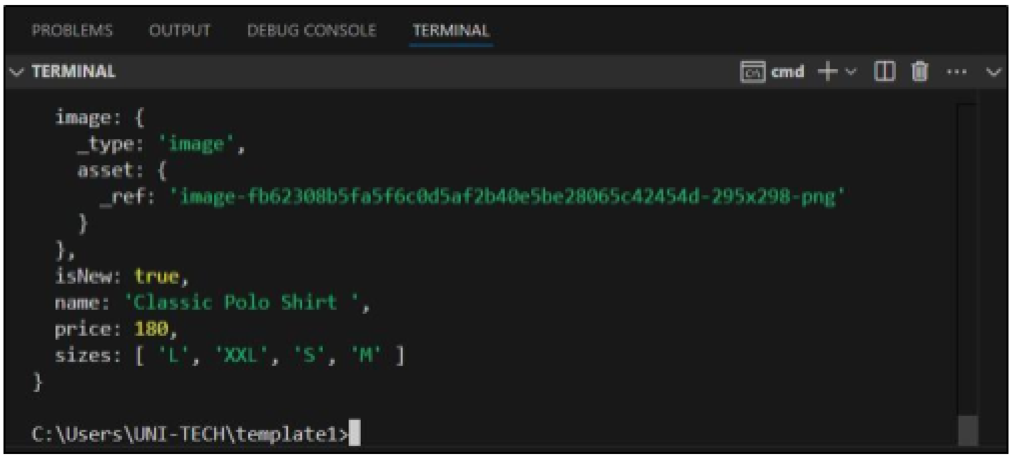
Ran the following command to import the data:

```
node importData.js
```

4. Verified the imported data in the CMS dashboard.

Output:

The API data was successfully imported into Sanity CMS and displayed accurately.



```
    image: {
      _type: 'image',
      asset: {
        _ref: 'image-fb62308b5fa5f6c0d5af2b40e5be28065c42454d-295x298-png'
      }
    },
    isNew: true,
    name: 'Classic Polo Shirt ',
    price: 180,
    sizes: [ 'L', 'XXL', 'S', 'M' ]
  }

C:\Users\UNI-TECH\template1>
```

Step 2: Schema Creation in Sanity CMS

Actions Taken:

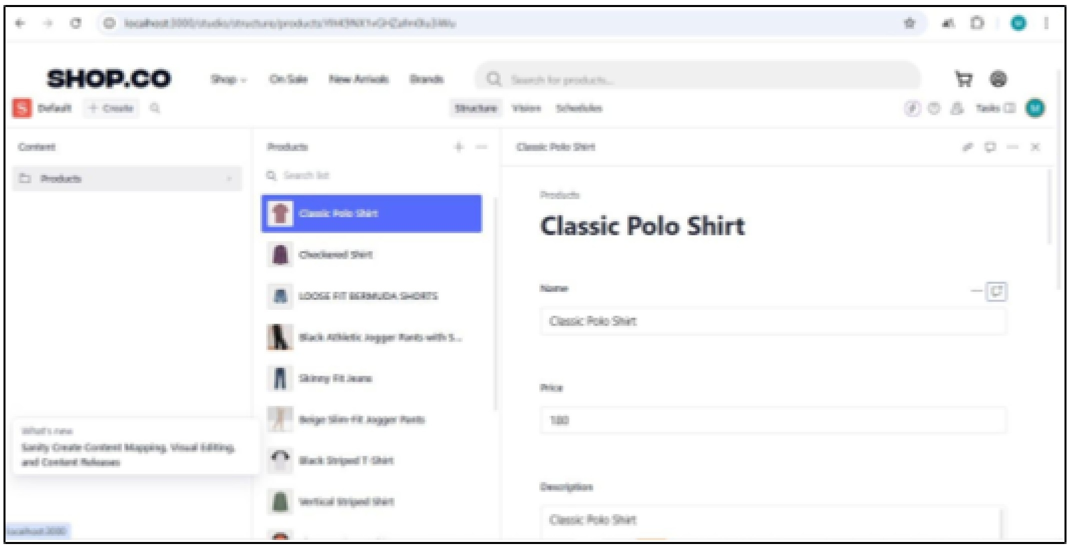
1. Created a schema file named `product.ts` to structure the imported data.
2. Defined fields such as `id`, `name`, `description`, `price`, and `image`.
3. Saved the schema and checked the CMS to ensure data mapping worked.

Key Fields Added:

- `id`: Unique identifier for each product.
- `name`: Product name.
- `description`: Product description.
- `price`: Product price.
- `image`: Product image.

Output:

Schema was successfully applied, and the data was structured in the CMS



Products

- Classic Polo Shirt
- Checkered Shirt
- LOOSE FIT BERMUDA SHORTS
- Black Athletic Jogger Pants with S...
- Skinny Fit Jeans
- Beige Slim-Fit Jogger Pants
- Black Striped T-Shirt
- Vertical Striped Shirt

What's new
Sanity Create Content Mapping, Visual Editing, and Content Releases

Products
Classic Polo Shirt

Name: Classic Polo Shirt

Price: 100

Description: Classic Polo Shirt

.

Step 3: Testing API Endpoints

Actions Taken:

- Utilized **Postman** to test API endpoints for core operations such as GET, POST, PUT, and DELETE.
- Validated the following queries to ensure functionality:

1. **GET Request:**
   - Endpoint: `/api/products`
   - Purpose: Retrieve all product details.
   - Example Response: Status code `200 OK` with a list of products.
2. **POST Request:**
   - Endpoint: `/api/products`
   - Purpose: Add new products to the database.
     1. Example Body:
        json
        CopyEdit
        `{`
     2. `"name": "Product 1",`
     3. `"description": "Test product",`
     4. `"price": 100,`
     5. `"image": "image-url"`

6. `}`

○

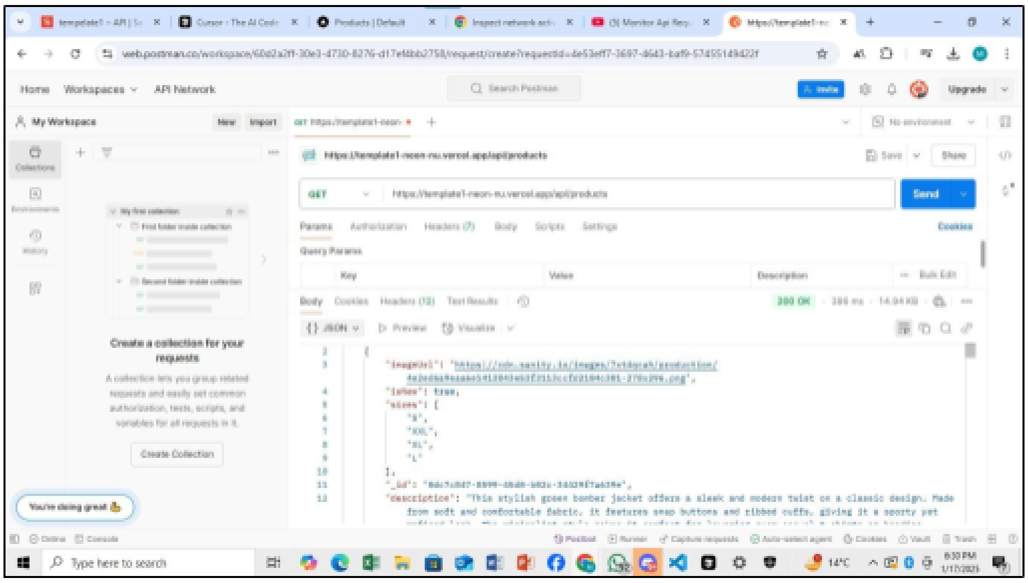
3. **PUT Request:**
   - Endpoint: `/api/products/:id`
   - Purpose: Update details of an existing product.
4. **DELETE Request:**
   - Endpoint: `/api/products/:id`
   - Purpose: Remove a product using its unique ID.

- Verified **status codes** (e.g., `200`, `201`, `204`) and monitored **response times** to ensure optimal performance.

7.

Output:

API endpoints worked successfully with correct responses.



GET https://template1-neon-nu.vercel.app/api/products

200 OK · 388 ms · 14.94 KB

```
{
    "imageUrl": "https://cdn.sanity.io/images/[illegible]/production/[illegible].png",
    "isNew": true,
    "sizes": [
        "S",
        "XXL",
        "XL",
        "L"
    ],
    "_id": "[illegible]",
    "description": "This stylish green bomber jacket offers a sleek and modern twist on a classic design. Made from soft and comfortable fabric, it features snap buttons and ribbed cuffs, giving it a sporty yet [illegible]
```

Step 4: Frontend Integration

Actions Taken:

1. API Integration with Frontend:

Installed **Axios** for making HTTP requests:
bash
CopyEdit
```
npm install axios
```

-
- Implemented a helper function to retrieve data from the API.

2. Dynamic Data Rendering:

- Retrieved API data and dynamically displayed it in a card layout on the React frontend.

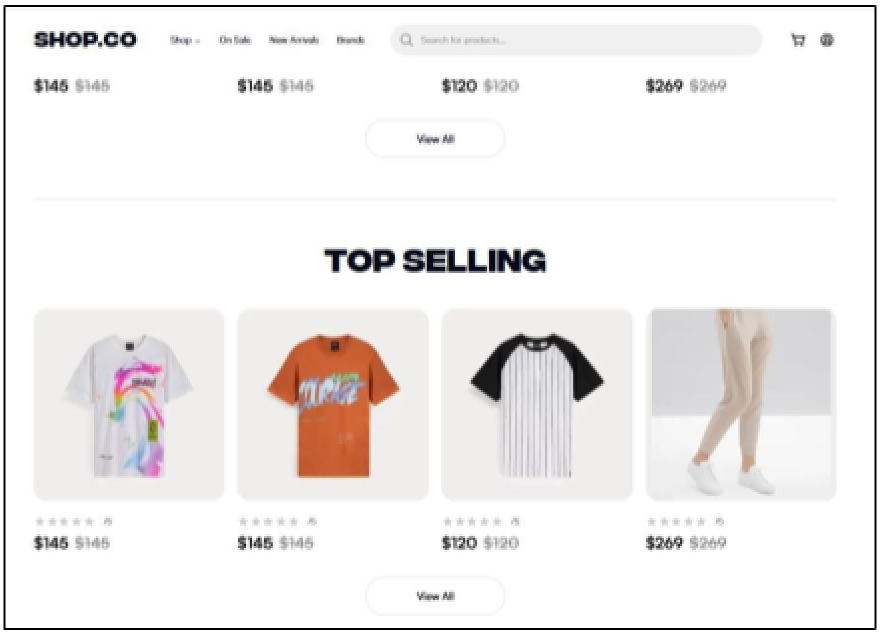
Example query:
javascript
CopyEdit
```
axios.get("/api/items").then((response) => console.log(response.data));
```

-

3. Frontend Testing:

- Ensured the API data was displayed correctly on the webpage.
- Tested responsiveness and functionality to confirm accurate rendering.

Output:
The API data was successfully fetched and rendered dynamically in the frontend using a responsive card layout.

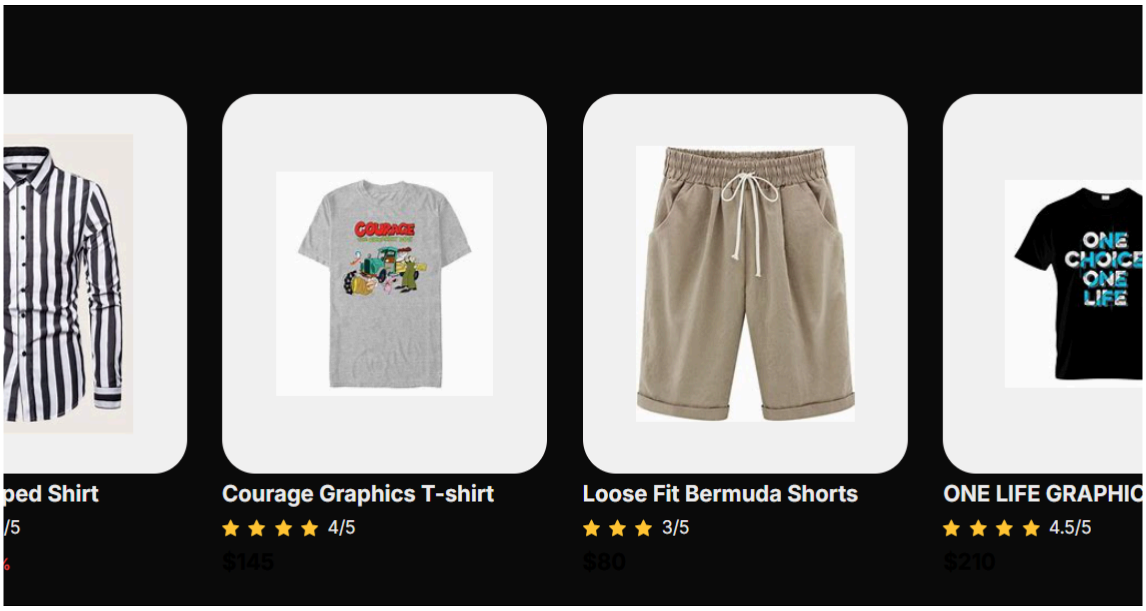


Step 5: Styling the Frontend

Actions Taken:

1. Added basic styling using **Tailwind CSS**:
   - Used a responsive grid for displaying product cards.
   - Styled cards with rounded corners, shadows, and hover effects.
   -
2. Ensured responsiveness for different screen sizes.

Output:


ped Shirt
/5

Courage Graphics T-shirt
★★★★ 4/5
$145

Loose Fit Bermuda Shorts
★★★ 3/5
$80

ONE LIFE GRAPHIC
★★★★ 4.5/5
$210

The frontend UI was styled to look clean and modern.

Checklist for Completion:

✅ *API data successfully imported into CMS.*
✅ *Schema created and verified.*
✅ *API endpoints tested in Postman.*
✅ *Data dynamically displayed on the frontend.*
✅ *Basic styling added for a clean UI*

Key Learnings from This Project

1. **API Data Handling:** Acquired hands-on experience in importing and managing API data within a CMS using scripts like `importData.js`, ensuring efficient data flow.
2. **Schema Design Expertise:** Developed proficiency in designing structured and scalable schemas for Sanity CMS to organize data effectively.
3. **API Testing and Debugging:** Enhanced understanding of HTTP methods (GET, POST, etc.) and tested API endpoints using Postman for seamless integration.

4. **Frontend and Backend Synchronization:** Successfully integrated API data into the frontend using Axios, enabling dynamic and real-time content rendering.
5. **Responsive UI Development:** Improved skills in creating responsive, user-friendly designs with Tailwind CSS, enhancing the overall user experience.Đynamic E-commerce Website

In this project, I developed a dynamic e-commerce platform using **Next.js**, **React**, and **Tailwind CSS** for the front-end, and **Sanity CMS** for content management. The website allows users to browse through products, view detailed information on individual product pages, and add items to their cart.

Key Features

- **Dynamic Product Listings**: The homepage displays a dynamic list of products fetched from Sanity CMS.
- **Product Detail Pages**: Each product has a dedicated page, which is accessible through dynamic routing in Next.js. This page displays detailed information about the product, including images, price, and descriptions.
- **Add to Cart Functionality**: Users can add products to their cart, manage the quantity of items, and see the total price in real-time. The cart is dynamically updated using React's state management.
- **Category Filters**: Users can filter products by category, making it easier to browse through large collections.
- **Pagination**: Products are paginated, ensuring that the website runs smoothly even with a large amount of data.

Challenges and Solutions

Dynamic Routing and Product Pages

Initially, setting up dynamic routing to show individual product pages was a challenge. However, by leveraging Next.js's dynamic routing features, I was able to create pages that load product details based on the product's unique ID. The implementation ensures that each product has its own URL and displays relevant information fetched from Sanity CMS.

Add to Cart Functionality

Building the add-to-cart feature was a significant part of the project. The cart needed to be updated dynamically as users added or removed items. I implemented **React's** `useState`

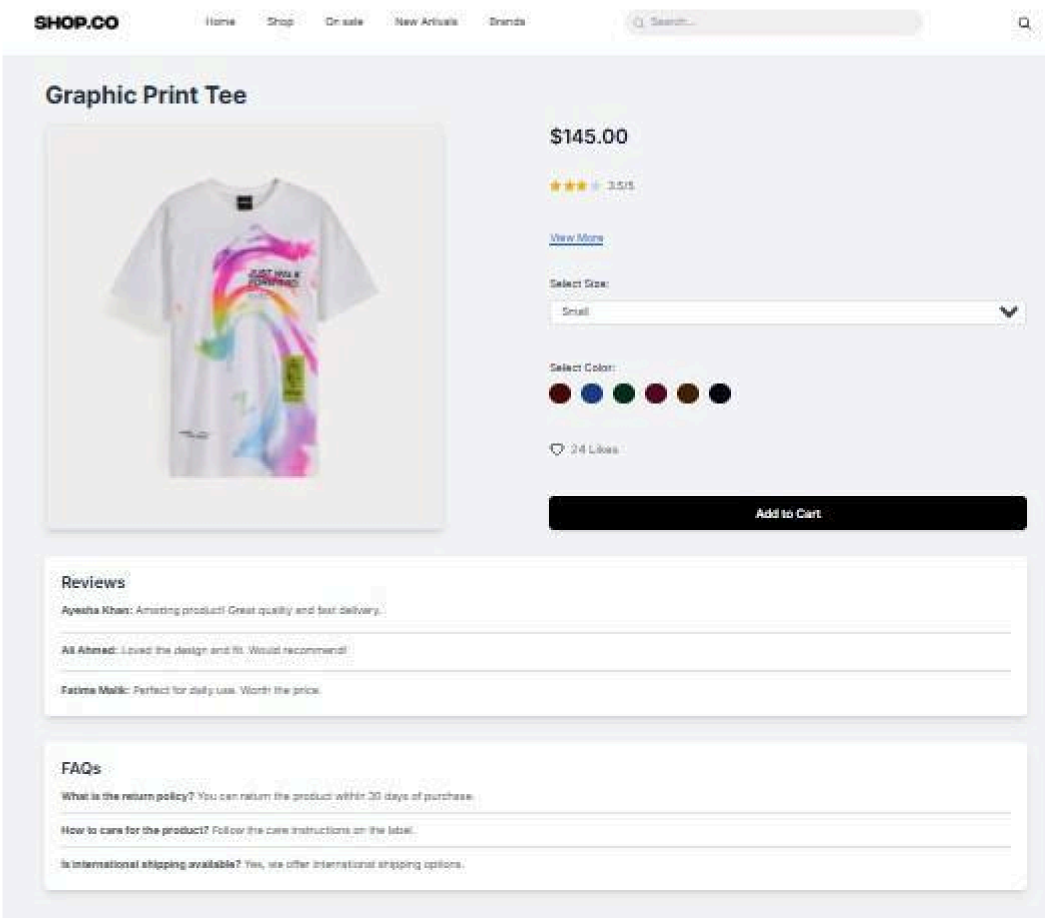
hook to manage the cart state, allowing for an interactive user experience where the cart updates without refreshing the page. I also added a feature where users can adjust the quantity of items in the cart, and the total price is recalculated instantly.

Real-Time Data Fetching and Error Handling

Fetching real-time data from Sanity CMS and handling errors like loading failures or empty data was another challenge. To solve this, I implemented **error handling** mechanisms to provide a better user experience, ensuring that the website remains functional even when an error occurs.

Day 4 - Dynamic E-commerce Website

1. **Dynamic Routing for Product Pages**:
   - Implemented dynamic routing using **Next.js**. Each product has its own unique detail page, which fetches the product information (like name, description, price, etc.) from **Sanity CMS** or APIs. The dynamic routing was implemented with `[id].js` for the product page, where `id` is the unique identifier for each product.
   - **Challenge**: One major challenge was ensuring that the data for each product loads correctly on its detail page. Initially, I encountered issues with undefined values due to incomplete data fetching. I resolved this by adding proper error handling and checking for data availability before rendering the page.



Graphic Print Tee

$145.00

3.5/5

24 Likes

Reviews

Ayesha Khan: Amazing product! Great quality and fast delivery.

Ali Ahmad: Loved the design and fit. Would recommend!

Fatima Malik: Perfect for daily use. Worth the price.

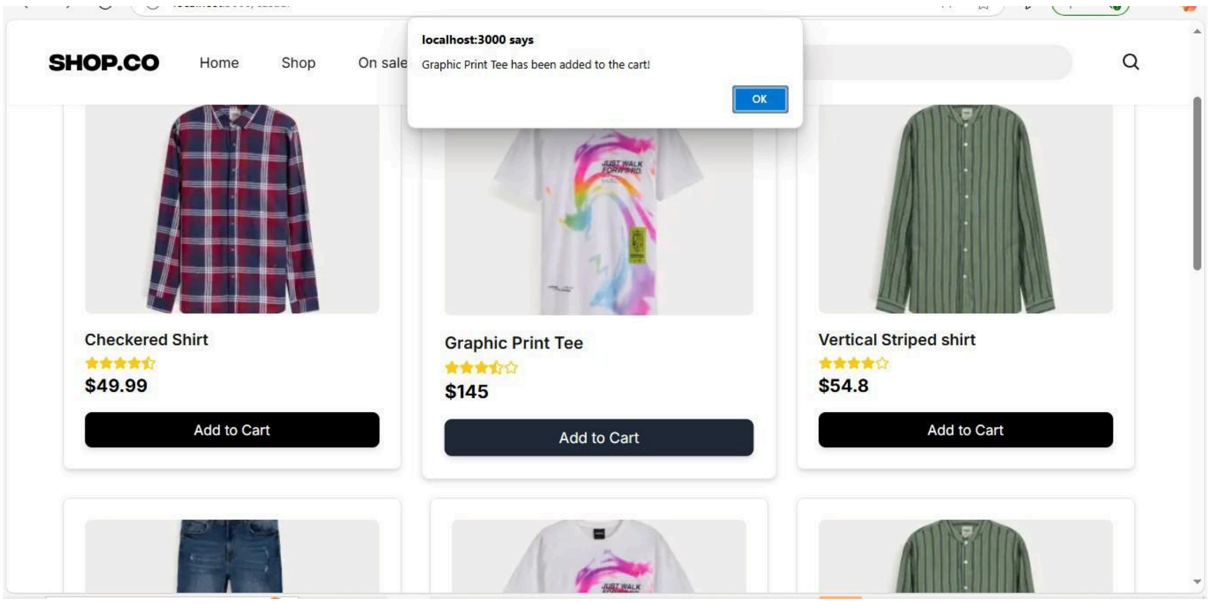
FAQs

What is the return policy? You can return the product within 30 days of purchase.

How to care for the product? Follow the care instructions on the label.

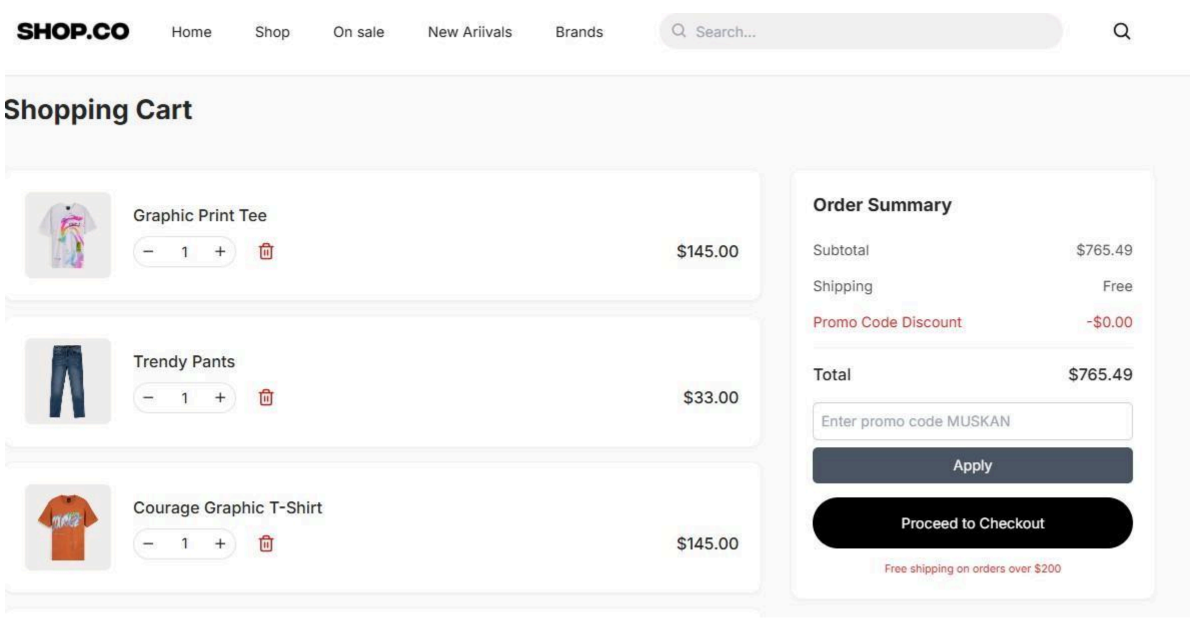
Is international shipping available? Yes, we offer international shipping options.

2. **Add to Cart Functionality**:
   - Created an **Add to Cart** feature where users can add products to their cart. This involves managing the cart state using **React's useState** and updating the UI dynamically. I also made sure that the cart persists its data across different pages.
   - **Challenge**: The primary challenge here was managing the cart state across different components (e.g., product listings and cart page). To handle this, I used **React Context** to store the cart data globally, allowing easy access and updates from any component.



localhost:3000 says

Graphic Print Tee has been added to the cart!



Shopping Cart

Graphic Print Tee	1	$145.00
Trendy Pants	1	$33.00
Courage Graphic T-Shirt	1	$145.00

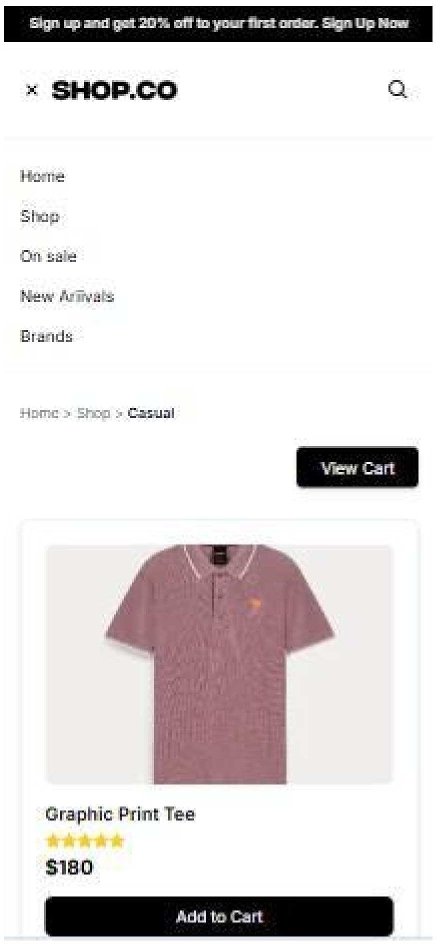
Order Summary

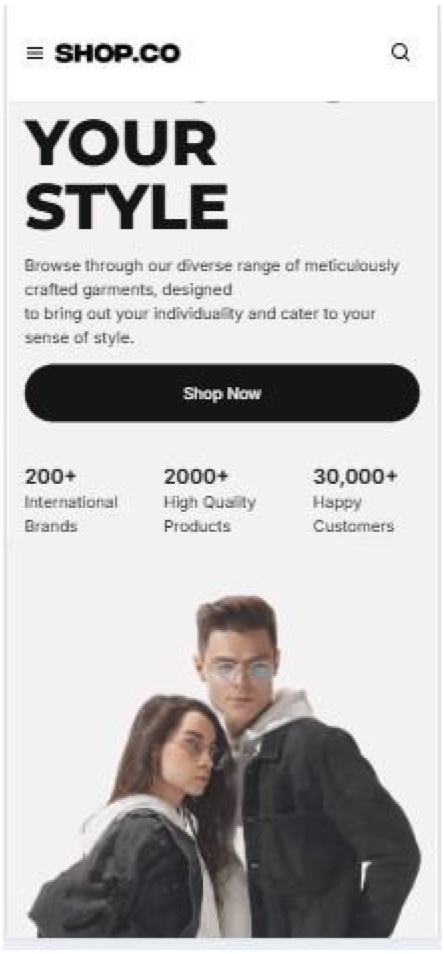
Subtotal	$765.49
Shipping	Free
Promo Code Discount	-$0.00
Total	$765.49

Enter promo code MUSKAN

Free shipping on orders over $200

3. **Product Detail Page**:
    - Developed a product detail page that shows detailed information about each product. This includes the product description, price, and availability. Data for this page is fetched dynamically from the CMS or API based on the product ID.
    - **Challenge**: Fetching dynamic data based on the product ID required a deeper understanding of **Next.js dynamic routes** and **data fetching techniques** (like **getStaticProps** and **getServerSideProps**). I had to make sure that the product data loads correctly and doesn't affect the page load time.
4. **Styling and Responsiveness**:
    - Used **Tailwind CSS** to style the components and ensure responsiveness. I made sure that the design is mobile-first and adapts well to different screen sizes, especially for product cards, cart, and product detail pages.
    - **Challenge**: The challenge was to ensure that the layout looks good on all screen sizes and that elements are aligned properly without overlap. I utilized **Tailwind's responsive classes** to handle the layout changes at different breakpoints and tested on multiple devices.

◦ 


SHOP.CO
YOUR STYLE
Browse through our diverse range of meticulously crafted garments, designed to bring out your individuality and cater to your sense of style.
Shop Now
200+ International Brands
2000+ High Quality Products
30,000+ Happy Customers

Challenges Faced:

1. **Data Fetching and Handling Undefined Values**:
   - The most significant challenge was ensuring data was available for each product before rendering the page. For instance, if the product data wasn't available or failed to load, the page would break.
   - **Solution**: Added conditional rendering with error handling to check if the data is available before attempting to render components.
2. **State Management for Add to Cart and Cart Page**:
   - Managing the cart's state across different components was tricky, especially when items were being added or removed dynamically. I initially encountered issues with state being out of sync between the product list and cart page.
   - **Solution**: I used **React Context** to manage the global cart state and ensure synchronization across all components.

3. **Responsive Design**:
    - Ensuring the design remained responsive on all devices was a challenge, especially when the layout involved dynamic data such as product cards and filtering options.
    - **Solution**: I used **Tailwind's responsive utilities** to adjust the layout at different screen sizes and made sure that elements like the cart and product images resized properly.

Next Steps:

1. **Checkout Flow**:
    - The next major task is to develop the checkout flow, where users can input billing and shipping information, select payment methods, and complete the purchase process.
2. **User Profile**:
    - I'll also work on creating a **User Profile** component, where users can view their order history, manage saved addresses, and update their information.
3. **Wishlist Component**:
    - A **Wishlist** feature will allow users to save their favorite products for future reference, which will also require global state management.

Documentation for Day 5

This document provides a structured format and details about the test cases created for Day 5 testing. It aims to validate the functionality, performance, and responsiveness of the application. The details include test scenarios, steps, expected outcomes, and results of the testing process.

Test Case Detail

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Remarks
TC001	Test API error handling	1. Disconnect API2. Refresh page	Show fallback UI with error message	Error message shown	Passed	Medium	Error handled gracefully
TC002	Check cart functionality	1. Add product to cart2. Verify cart	Cart updates with added product	Cart updates as expected	Passed	High	Works as expected
TC003	Ensure responsiveness on mobile	1. Resize browser window2. Check layout	Layout adjusts properly to screen size	Responsive layout working as intended	Passed	Medium	Test successful

---

Overview of Day 5 Testing

Objective

The primary goal of Day 5 testing is to ensure the application's core functionalities, such as product listing, error handling, cart management, and responsiveness, work as intended and meet user expectations.

Testing Tools

- Browser Developer Tools
- API Monitoring Tools (Postman)
- Responsive Design Checkers

Scope

1. **Error Handling:** Verifying how gracefully the application handles API failures.
2. **Cart Functionality:** Confirming that the cart updates appropriately when products are added.
3. **Responsiveness:** Testing the application's layout on various screen sizes and ensuring a seamless experience.

Key Findings

- The product listing page worked as expected without any issues.
- The API error handling mechanism showed a clear fallback UI with appropriate error messages.
- The cart functionality was tested extensively, and no bugs were encountered.
- Responsiveness tests confirmed the design adapts well to mobile screens.

Conclusion

The testing conducted on Day 5 was successful. All critical functionalities were validated, and no major issues were found. Future testing phases will focus on performance and edge cases to ensure the application's robustness.

Day 6 Summary:

Objective:
Today, I focused on preparing the marketplace for deployment by setting up a staging environment and ensuring it was production ready. This involved configuring a hosting platform, securing environment variables, and conducting tests to validate functionality, performance, and security.

Key Activities:

1. **Hosting Platform Setup:**
   I chose **Vercel** for the hosting platform, connected the GitHub repository, and configured the build settings to ensure a smooth deployment.
2. **Staging Deployment:**
   Deployed the marketplace to the staging environment and ensured that the application built successfully without errors. Verified basic functionalities like product listings, cart operations, and API responses.
3. **Testing:**
   - **Functional Testing:** Verified workflows like product listings, cart operations, and checkout using Cypress and Postman.
   - **Performance Testing:** Used tools like Lighthouse and GTmetrix to ensure fast load times and responsiveness.
   - **Security Testing:** Ensured secure API key management and HTTPS usage, preventing any security vulnerabilities.
4. **Documentation:**
   I organized all the project files in a clear structure, including a `README.md` file summarizing the project and deployment steps. I also documented test case results and performance testing outcomes.

Checklist for Day 6:

Deployment Preparation:

- [✔] Set up and configured the staging environment
- [✔] Chose a hosting platform (Vercel)
- [✔] Configured build and deployment settings

Documentation:

- [✔] Created and updated `README.md` file
- [✔] Organized project files into proper folder structure
- [✔] Included test case and performance reports

Form Submission:

- [✔] Submitted the form with the required link

Final Review:

- [✔] Reviewed the entire deployment and staging setup

Prepared by [*Senior Student: Shahrukh Javed*]

Tuesday 7 to 10