

# MarketPlace-Builder-Hackathon

## Day-4: Building Dynamic Frontend Components

1/19/2025

GIAC: Sunday(2pm to 5pm)

Fatima Farooq Khuwaja

00162513

### OVERVIEW

On Day 4, the focus was on **building dynamic frontend components** for an e-commerce marketplace. These components aim to provide a seamless user experience, ensuring that customers can easily interact with the website. The goal was to design and implement reusable, responsive, and modular components that can dynamically display product data fetched from Sanity CMS. By doing so, users can easily interact with the website. The project implements the following functionalities :

1. Product Listing Page
2. Product Detail Component
3. Category Component
4. Search Bar
5. Cart Component
6. Wish list Component
7. Checkout Flow Component
8. User Profile Component
9. Pagination Component
10. Footer and Header Components
11. Reviews and Ratings Component

# **1. Product Listing Component**

**Product Listing** page is primary interface where user can view all available products which are displayed using a grid layout in structured way. The product data is fetched from Sanity CMS and dynamically shown on the UI.

- Product data is fetched in real-time from Sanity CMS which is displayed on the UI is always up-to-date with the latest data from the backend.
- The products are displayed in a grid layout, which helps to structure the page and keep it in organized way.
- The grid layout enhances the user experience by making it easier for users to interact with and navigate through products.
- The component is fully responsive, meaning layout will adjust accordingly, to different screen sizes and devices, such as smart phones, tablets, and desktops.

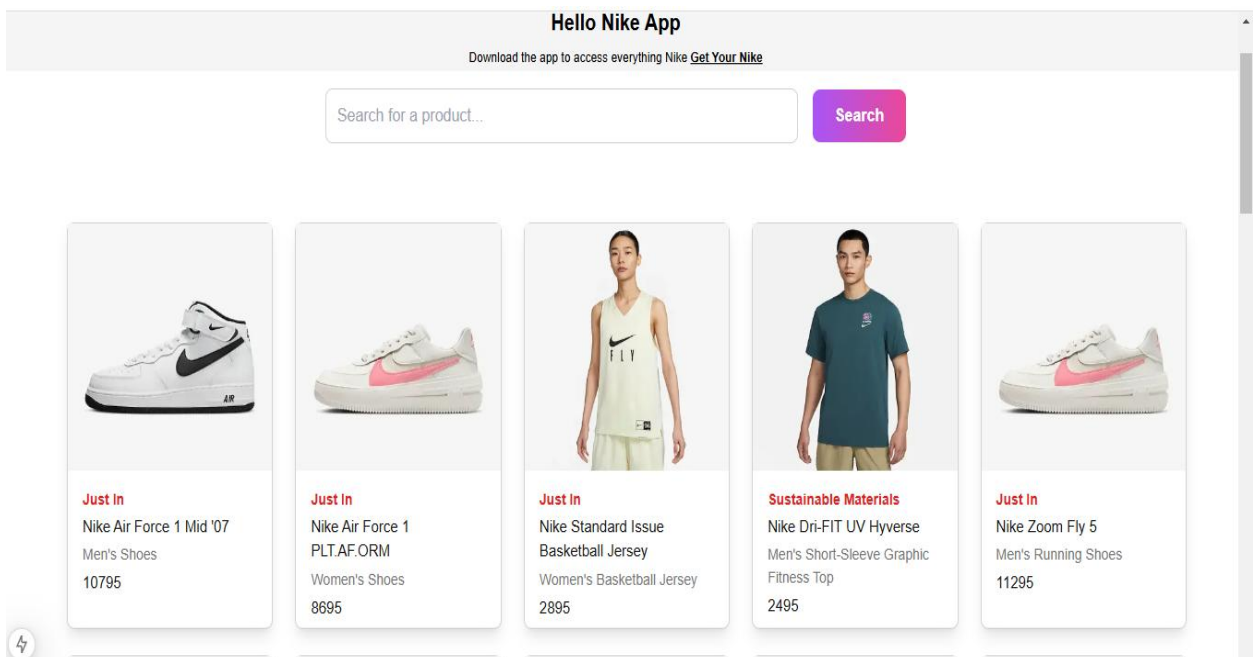
The product card includes the following fields:

- Product Image
- Product Name
- Price
- Category
- Status

```
EXPLORER
HACKTHONNN-3
src
  app
    Component
      Footer.tsx
      FourthSection.tsx
      HeroSection.tsx
      Navbar.tsx
      Navbarthree.tsx
      Navbartwo.tsx
      secondSection.tsx
      SixthSection.tsx
      ThirdSection.tsx
      types.tsx
    fonts
    Help
    Joinus
    Product
    Products
      paget.tsx
    Signin
    studio \ [...tool]
    page.tsx
    favicon.ico
    # globals.css
    layout.tsx
    .js not-found.js
  OUTLINE
  TIMELINE

src > app > Products > paget.tsx > ProductsList

7  productsList = () => {
8    Search
64  </p>
65  </div>
66
67  /* Product List */
68
69
70  <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-5 md:grid-cols-4 gap-6 lg:ml-20 m-14">
71    {filteredProduct.map((p) => {
72      return(
73        <Link href={`/${Product}/${p.slug}`} key={p.id}>
74          <div className="w-full h-[370px] bg-white border border-gray-300 rounded-lg shadow-lg">
75
76            <Image src={p.image} alt="product image" height={300} width={300} className="w-full h-[37vh] object
77            <div className="p-4">
78              <h1 className="text-red-600 text-sm font-semibold">{p.status}</h1>
79              <h1 className="text-[15px] font-[500] mt-1">{p.productName}</h1>
80              <h1 className="text-[#757575] text-[14px] font-[500] mt-1">{p.category}</h1>
81              <h1 className="text-[15px] font-[500] mt-1">{p.price}</h1>
82            </div>
83          </div>
84        </Link>
85      )
86    })
87  }
88  </div>
89  </div>
90
91
92
93
94
```



## **2. Product Detail Component**

On the **Product Detail** page, the data for each product is dynamically displayed, with each product having its own unique page. This is achieved using Next.js dynamic routing. As a result, users can easily view and read all the details of a specific product. The data is dynamically fetched and shown on the page located at `Product/[slug]/page.tsx`. This page includes all the necessary product information.

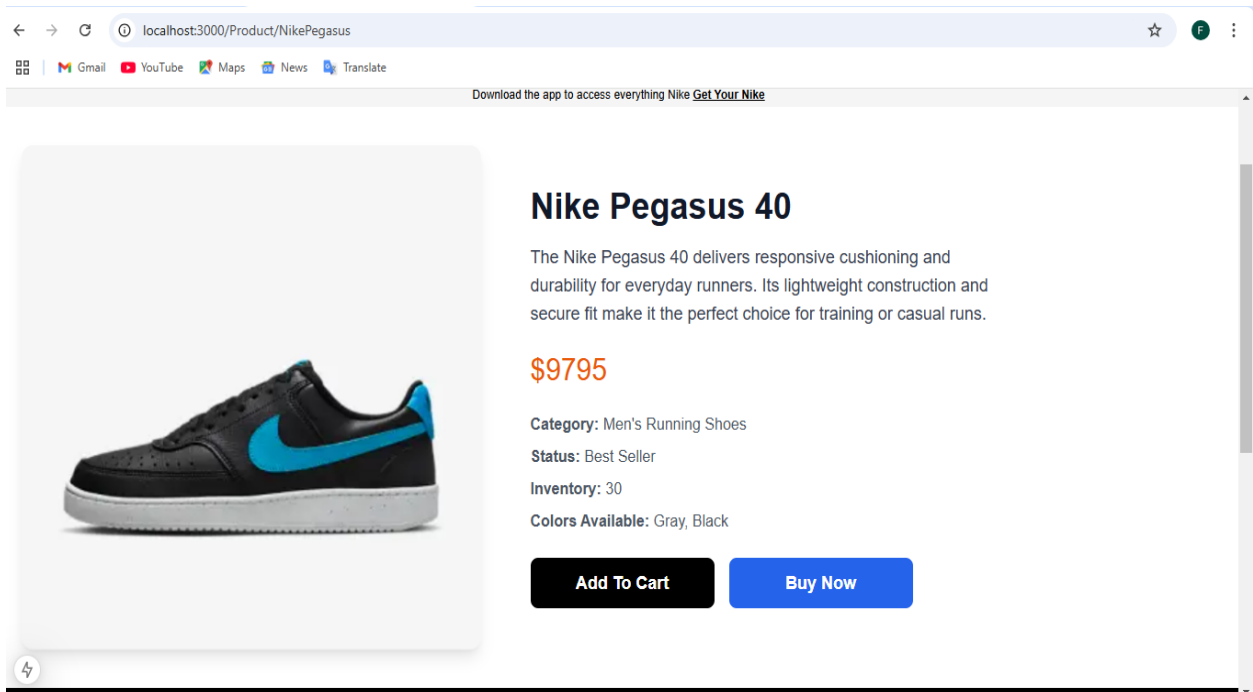
- The UI design is clean and well-organized, ensuring that users can easily read through the product details.
- Using Next.js dynamic routing each product has a unique page. The route is structured as `Product/[slug]/page.tsx`, where the slug serves as a unique identifier for the product.
- This allows the page to dynamically render the specific product data based on the URL (`/product/abc123`).

The product detail page includes all relevant product details:

- Product Name
- Product Images
- Price
- Description
- Colors
- Sizes
- Inventory
- Status
- Category

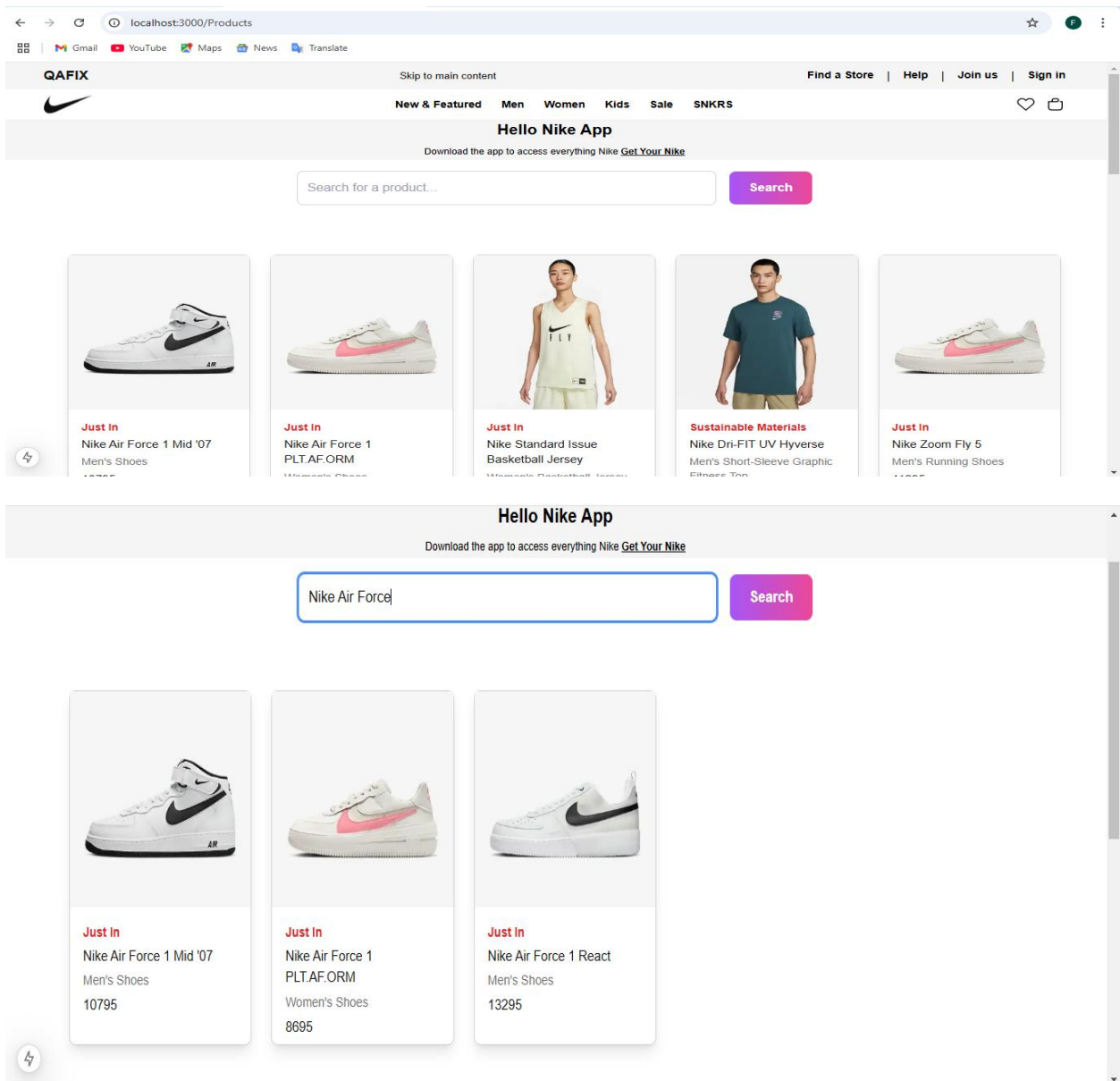
```
EXPLORER
HACKTHONNN-3
src
  app
    Component
      Footer.tsx
      FourthSection.tsx
      HeroSection.tsx
      Navbar.tsx
      Navbartwo.tsx
      Navbartwo.tsx
      secondSection.tsx
      SixthSection.tsx
      ThirdSection.tsx
      types.tsx
    fonts
    Help
    Joinus
    Product.[slug]
      page.tsx 1
    Products
      page.tsx
      Signin
    studio.[[...tool]]
      page.tsx
    faviconico
    globals.css
    layout.tsx
  OUTLINE
  TIMELINE

src > app > Product > [slug] > page.tsx > ProductDetail > res
6 const ProductDetail = async ({ params }: { params: { slug: string } }) => {
7   return (
8     <div className="w-full h-auto bg-white py-12 px-4 sm:px-6 lg:px-8">
9       <div className="flex flex-col lg:flex-row gap-10 lg:gap-16 items-center max-w-screen-xl mx-auto">
10        <div className="w-full max-w-[480px] h-auto">
11          <Image
12            src={data.image}
13            alt={data.productName}
14            width={480}
15            height={480}
16            className="object-contain rounded-xl shadow-lg transition-transform transform hover:scale-105"
17          />
18        </div>
19        <div>
20          <div className="w-full max-w-lg text-gray-900">
21            <h1 className="text-3xl lg:text-4xl font-semibold leading-tight mb-4">{data.productName}</h1>
22
23            <div>
24              <div className="text-lg text-gray-700 mb-6">{data.description}</div>
25
26              <div>
27                <div className="text-3xl font-[500] text-orange-600 mb-6">{`$${data.price}`}</div>
28
29                <div>
30                  <div className="text-gray-600 space-y-2 mb-6">
31                    <p><strong>Category:</strong> {data.category}</p>
32                    <p><strong>Status:</strong> {data.status}</p>
33                    <p><strong>Inventory:</strong> {data.inventory}</p>
34                    <p><strong>Colors Available:</strong> {data.colors.join(', ')}</p>
35                  </div>
36                </div>
37              </div>
38            </div>
39          </div>
40        </div>
41      </div>
42    </div>
43  )
44}
```



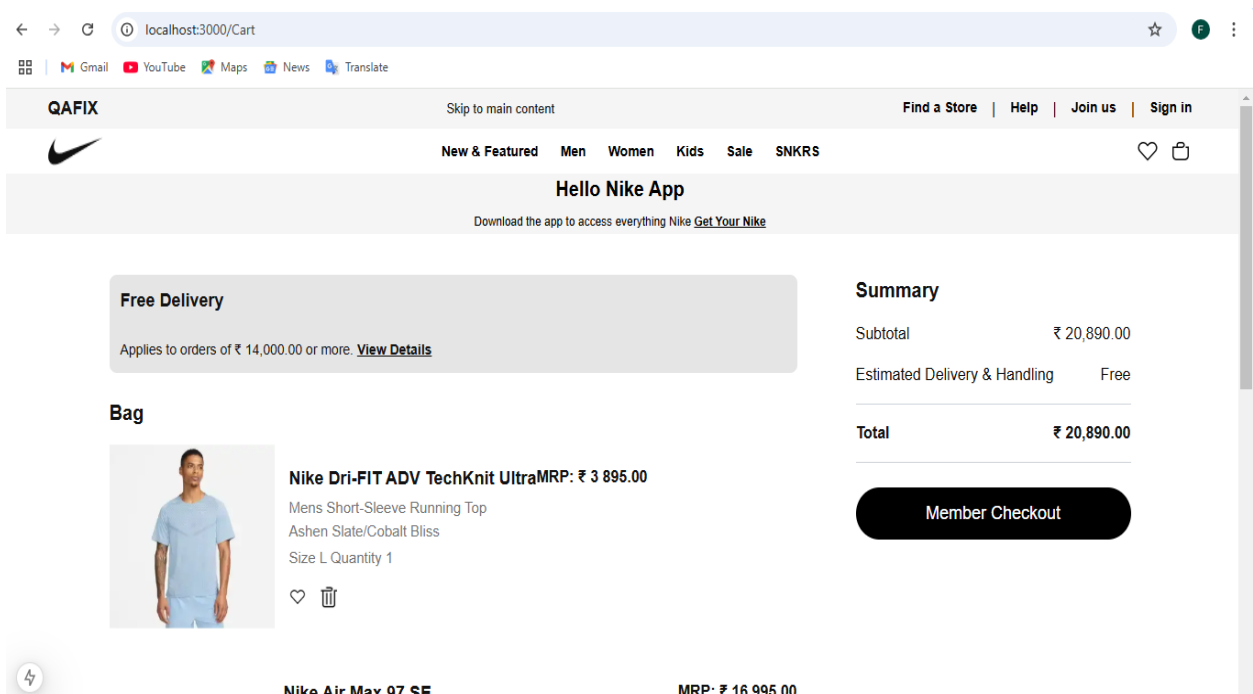
### 3. Search Bar

The **Search Bar Component** allows users to easily search for products by typing the product name. This component provides an efficient way for users to find specific products from a large catalog. As the user types in the search field, it dynamically filters and displays matching results in real-time.



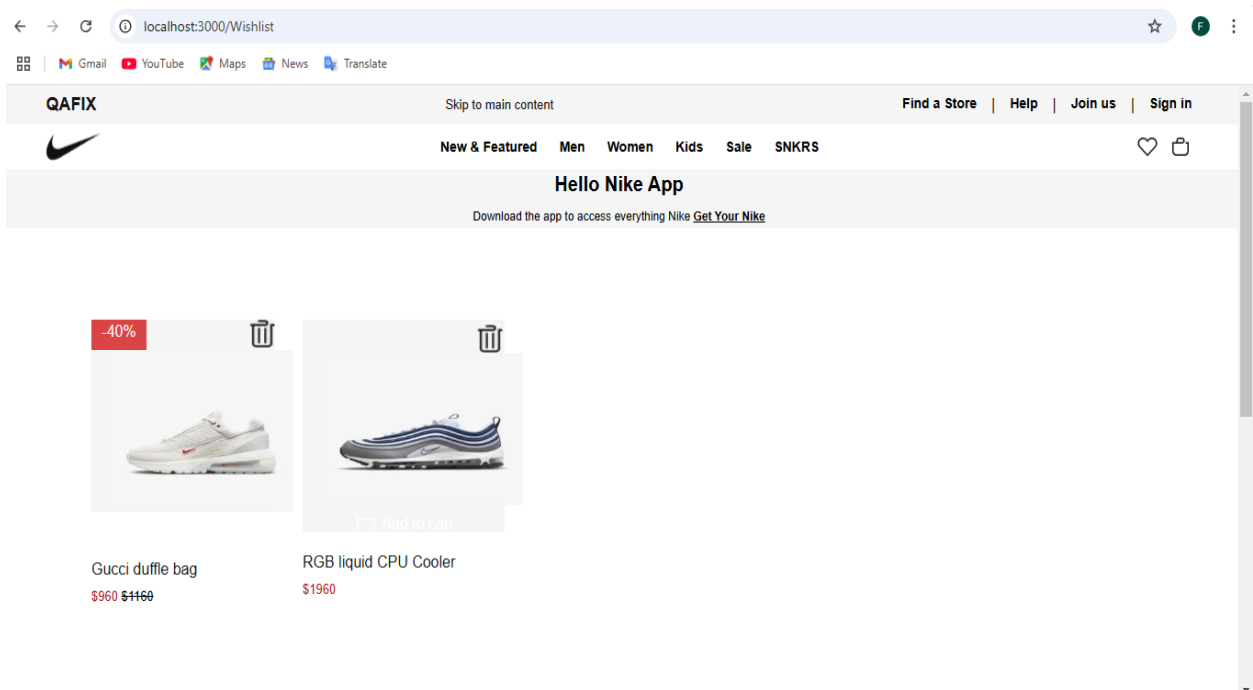
## 4. Cart Component

The **Cart Page** allows users to view and manage the items they have added to their cart. When a user clicks on the **Add to Cart** button for a selected product, that product is stored in the cart, allowing users to review and modify their selections before proceeding to checkout. Users can easily adjust the quantity of each item or remove products they no longer wish to purchase. The cart page also typically includes buttons for proceeding to checkout or continuing shopping, offering clear paths for users to finalize their purchase or keep browsing.



## 5. Wish list Component

The **Wish list Component** allows users to save their favorite products for future reference. When a user comes across a product they like but are not ready to purchase immediately, they can add it to their wish list with a simple click. This feature enables users to personalized list of products they are interested in, making it easier for them to revisit and purchase later. The wish list displays the saved items along with essential product details. Users can also remove items from their if they no longer wish to save them.





## 6. Checkout Flow Component

The **Checkout Flow Component** is a multi-step form designed to guide users through the checkout process, ensuring that all necessary information is collected before completing their purchase.

### Review Cart Items

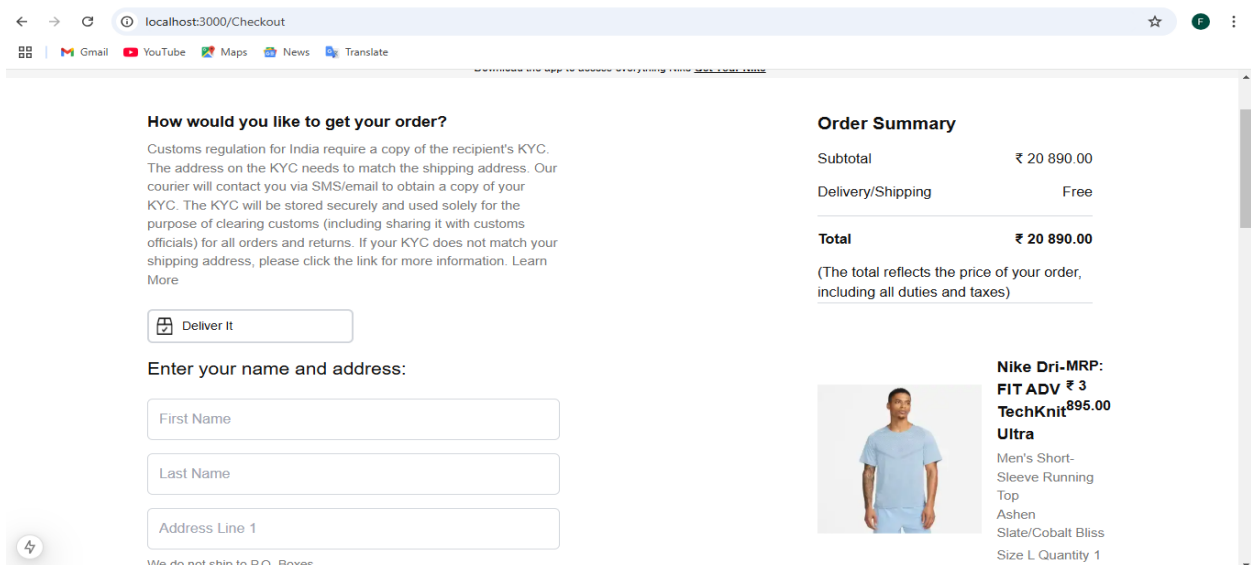
Users see a summary of the items they've selected, including **product name**, **quantity**, and **price**, and can make any changes.

### User and Payment Detail

In this step user select payment method among (Easy paisa, Jazz cash, On cash delivery, Credit or Debit card) users enter their payment information Along with payment details, users also need to provide **personal details** such as their **username**, **email address**, and **billing address**

### Place Order

Users review all details and click **place order** to complete the purchase.



The screenshot shows a web browser window with the URL `localhost:3000/Checkout`. The page layout is as follows:

- Header:** Browser navigation bar with back, forward, and refresh buttons. Address bar shows `localhost:3000/Checkout`. Below the address bar are links for Gmail, YouTube, Maps, News, and Translate.
- Main Content Area:**
  - How would you like to get your order?**
    - Text: "Customs regulation for India require a copy of the recipient's KYC. The address on the KYC needs to match the shipping address. Our courier will contact you via SMS/email to obtain a copy of your KYC. The KYC will be stored securely and used solely for the purpose of clearing customs (including sharing it with customs officials) for all orders and returns. If your KYC does not match your shipping address, please click the link for more information. Learn More"
    - Form: A button labeled "Deliver It" with a small icon of a delivery truck.
  - Enter your name and address:**
    - Form: Three input fields for "First Name", "Last Name", and "Address Line 1".
    - Text: "We do not ship to P.O. Boxes"
- Order Summary:**
  - Subtotal: ₹ 20 890.00
  - Delivery/Shipping: Free
  - Total: ₹ 20 890.00**
  - Text: "(The total reflects the price of your order, including all duties and taxes)"
- Product Image and Details:**
  - Image: A man wearing a light blue t-shirt.
  - Text: "Nike Dri-MRP: FIT ADV ₹ 3 TechKnit<sup>895.00</sup> Ultra Men's Short-Sleeve Running Top Ashen Slate/Cobalt Bliss Size L Quantity 1"

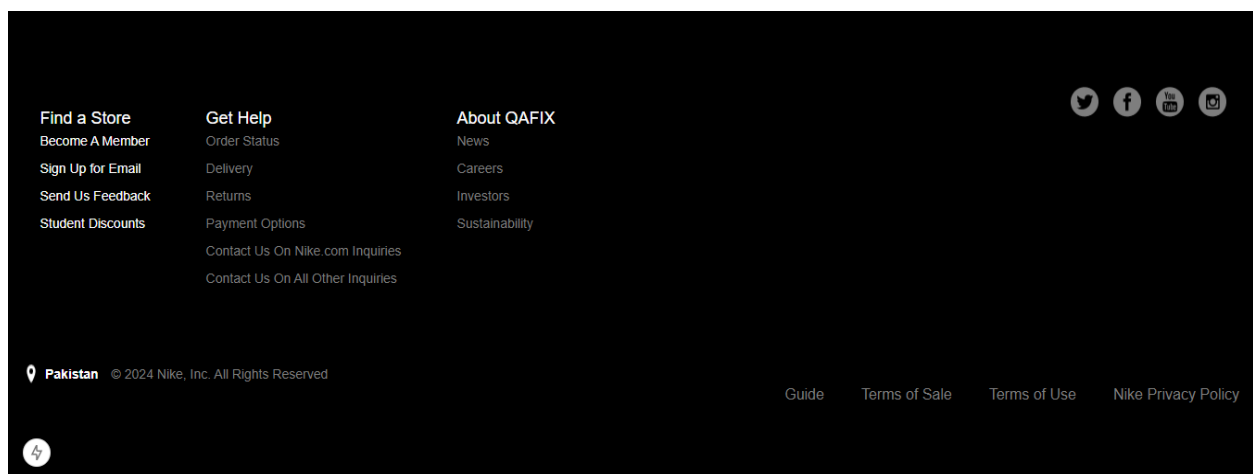
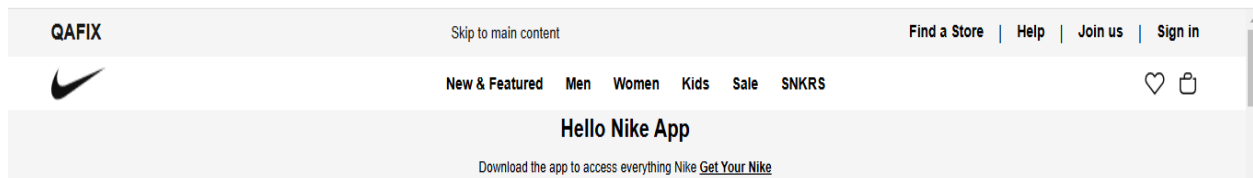
## 7. Footer And Header Component

The **Header** and **Footer** components are essential for the website, providing an easily accessible navigation structure across all pages. These both appear on all pages through the layout system.

The **Header** is sticky at the top of the page which includes links to important pages like Home, Shop, About, Contact, and Cart, along with a search bar and user account/login options. The **company logo** is also displayed in the header.

The **Footer** appears at the bottom of each page, providing links to Privacy Policy, Terms & Conditions, FAQs, and Social Media along with contact details.

Both components are designed to be responsive, adapting to different screen sizes. This allows users to easily navigate and access key information across the site, whether on mobile or desktop.



# **Technical Report**

## **✓ Steps Taken to Build and Integrate Components**

- Created a Next.js project and set up Sanity CMS for managing products.
- Integrated external API to migrate data into Sanity CMS.
- Fetched data from Sanity CMS for product listing and individual product pages.
- Data fetched from Sanity CMS via API calls.
- Used a dynamic grid layout to display products.
- Implemented dynamic routing in Next.js to show individual product details.
- Added functionality for users to search products by product name.
- Implemented state management for adding items to the cart and wish list.
- Created a multi-step checkout form for users to enter billing, shipping details, and payment information.
- Created cart page to show products added in cart by users.
- Created wish list to show products added in wish list by users.
- Used Tailwind CSS for styling, ensuring the application is responsive and works on both mobile and desktop devices.

## **✓ Challenges Faced and Solutions Implemented**

- Initially I faced issues with fetching data from Sanity CMS.  
SOLUTION: I double-checked the API keys, tested queries in Sanity Studio, and also tested the APIs in Postman. I compared the external API product schema with the Sanity schema to ensure they were the same.

- Struggled with dynamically displaying product details.  
SOLUTION: Used Next.js dynamic routing (`/Product/[slug]`) to handle individual product pages and render the product data dynamically based on the URL.
- Managing the cart and wish list states across components was complex.  
Used React's `useState` for local states management.

## ✓ Best Practices Followed During Development

- Built reusable components like `ProductCard` and `ProductDetail` to maintain clean and organized code.
- Used `useState` to efficiently manage application state.
- Followed mobile-first design principles using Tailwind CSS for styling, ensuring the app is fully responsive across all devices.

## ***Self-Validation Checklist***

|                                |   |
|--------------------------------|---|
| Frontend Component Development | ✓ |
| Styling and Responsiveness     | ✓ |
| Code Quality                   | ✓ |
| Documentation and Submission   | ✓ |
| Final Review                   | ✓ |

PREPARED BY : FATIMA FAROOQ KHUWAJA