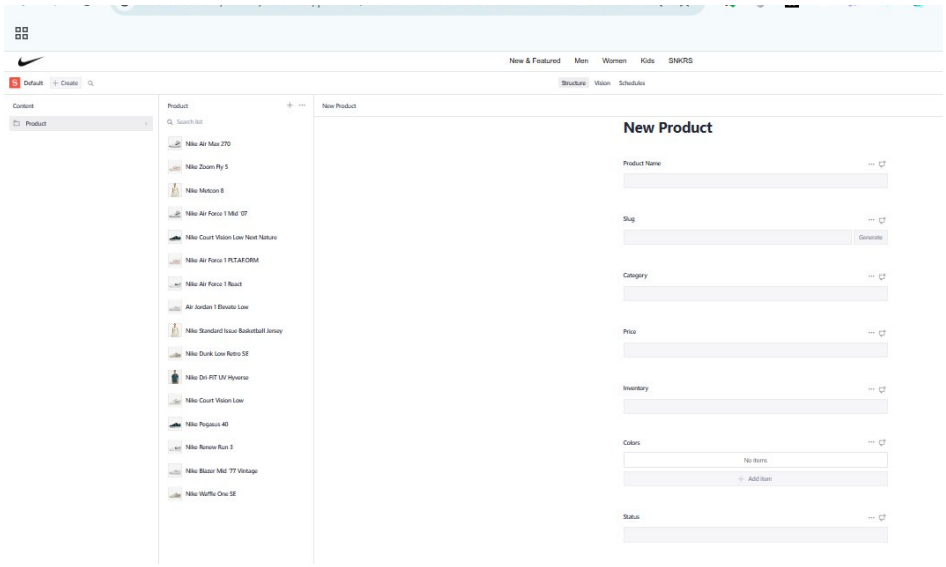
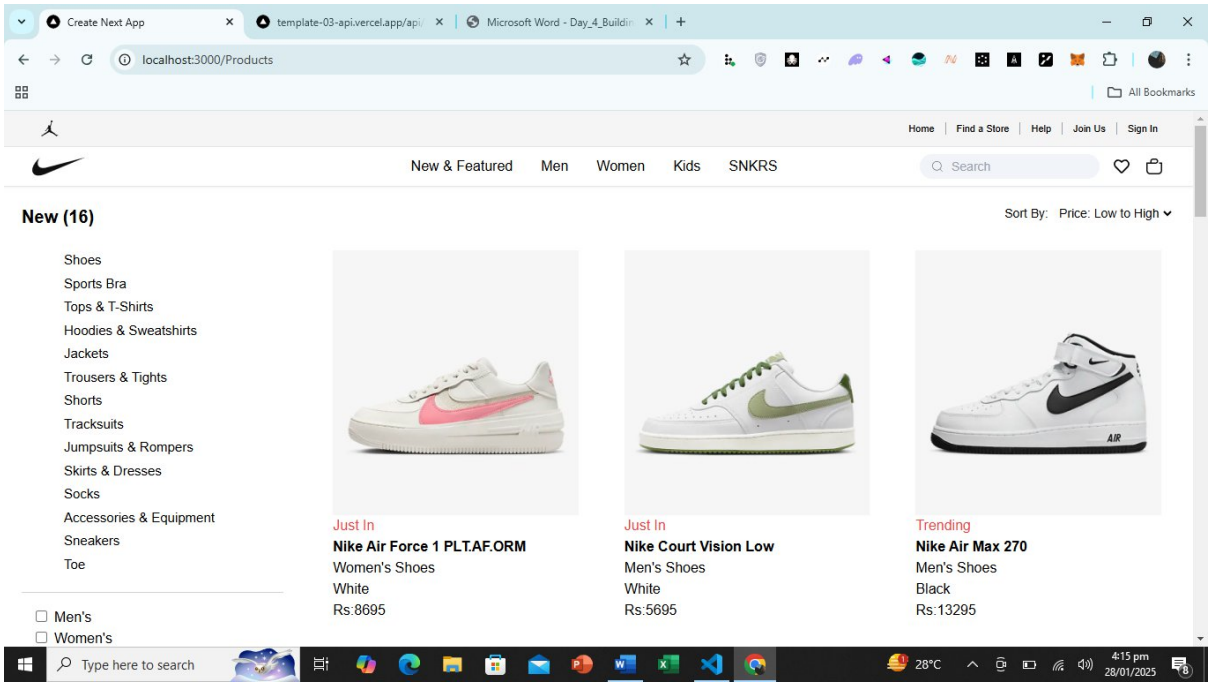
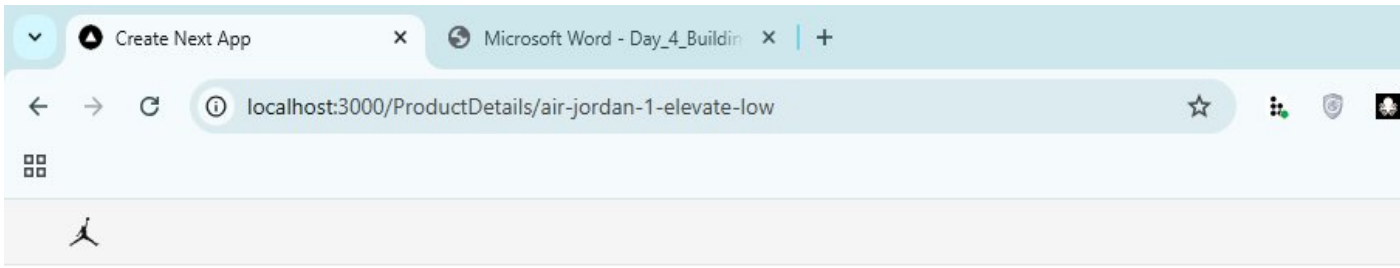
"Day 4 - Dynamic Frontend Components - [Nike]"

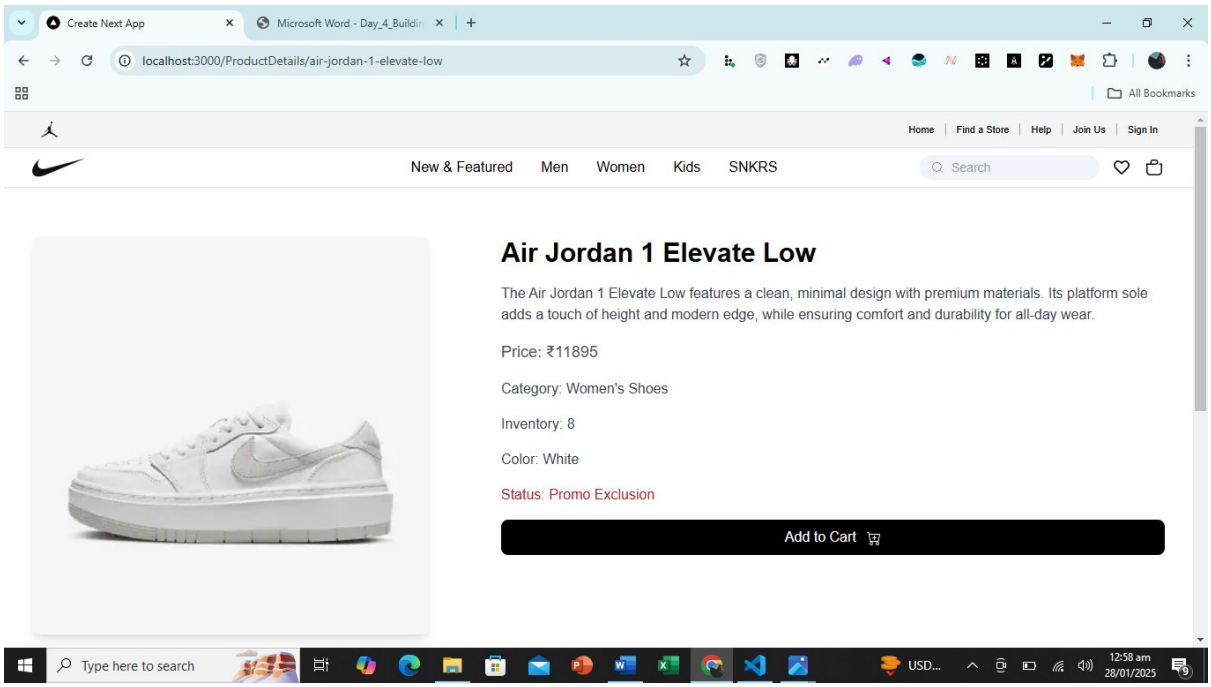
.>>>>>>>>>>>Product Listing Page Fetching Data from Sanity CMS:
On this page, the products are shown dynamically. The data for each product (like name, image, price) is fetched from Sanity CMS and displayed. Every time the page loads, it gets updated data directly from the backend.

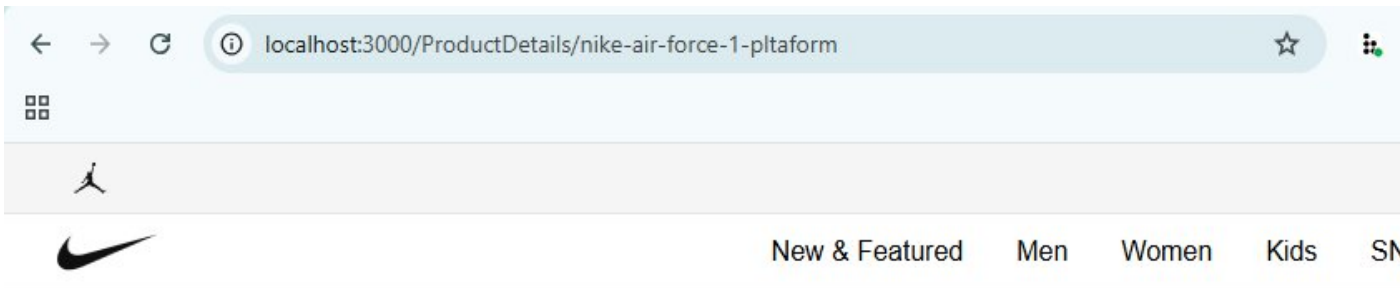


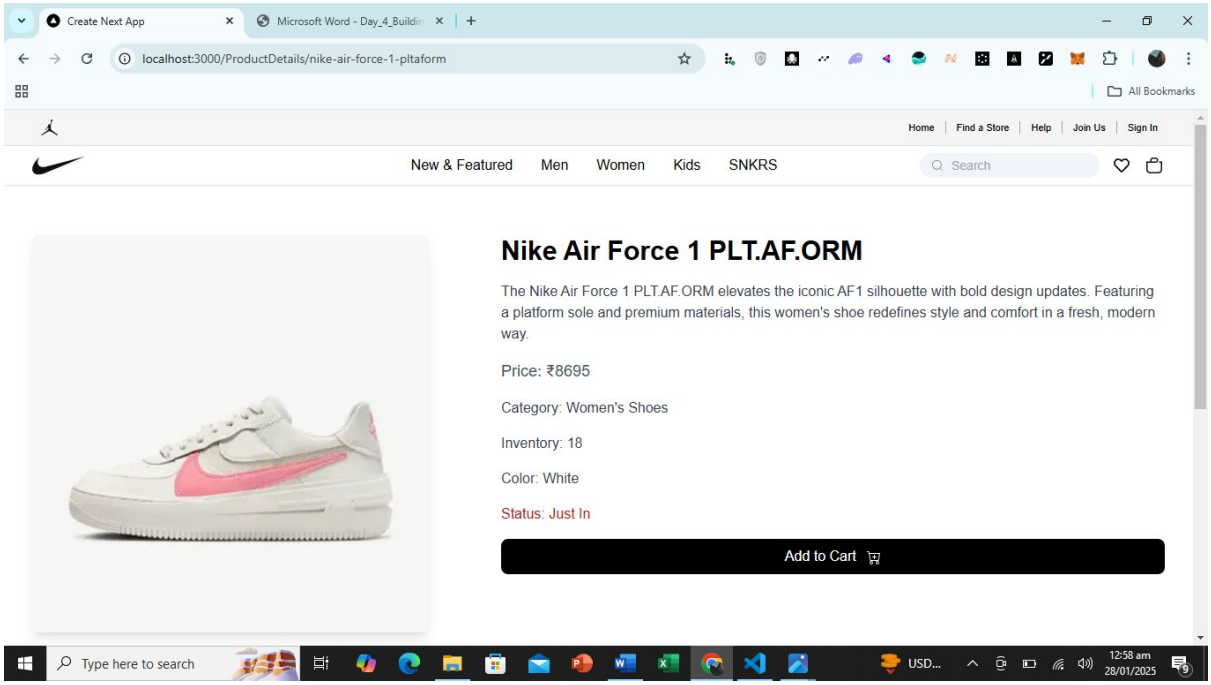


Product Detail Page with Accurate Routing:
This page displays detailed information about each product. When you click on a product, it takes you to its own page where you can see more information. The routing is set up correctly so that each product has a unique URL, and the correct data shows up.



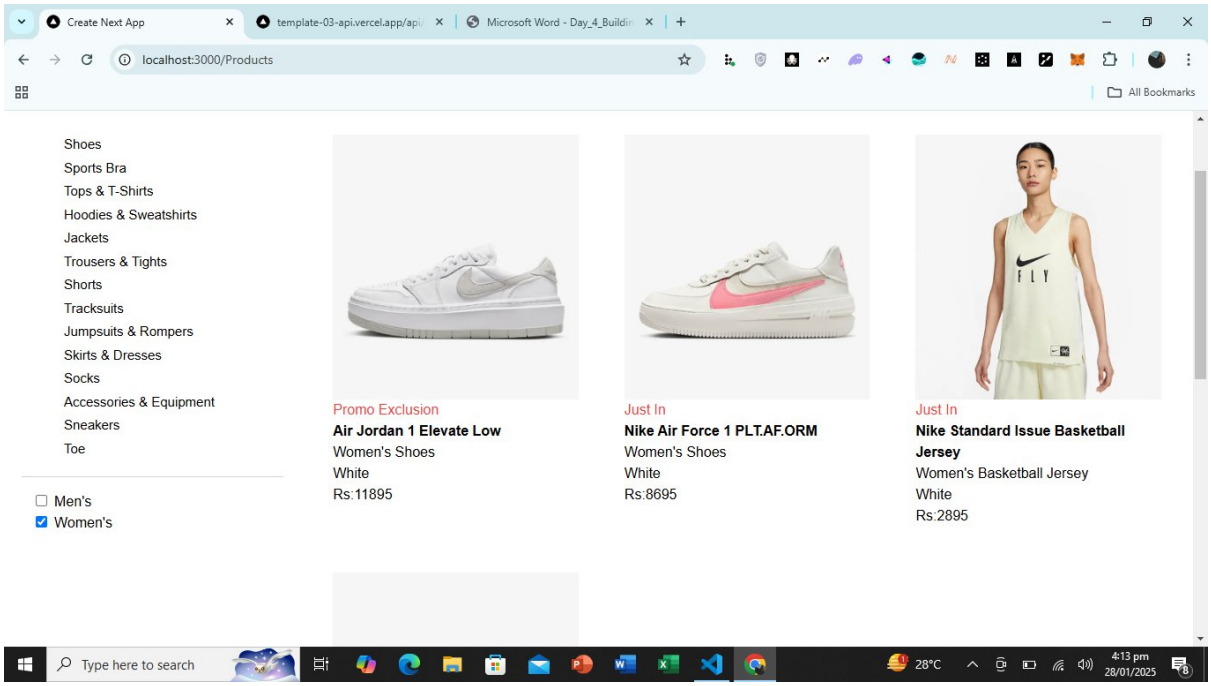


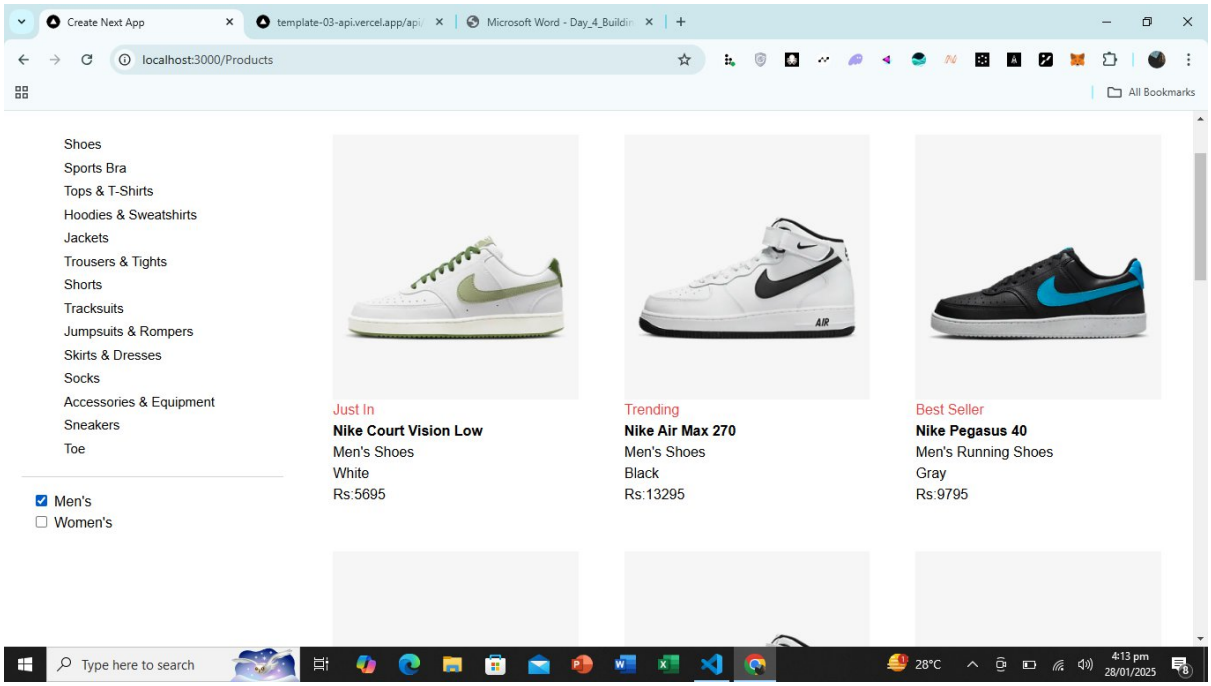




<<<<<<<<<<<<<<<<<<<<<<<<Categories Filter >>>>>>>>>>>>>>>>>

This feature allows users to filter products by category. When you select a category, only products that belong to that category will appear on the page. It makes browsing easier for users who are looking for specific types of products.

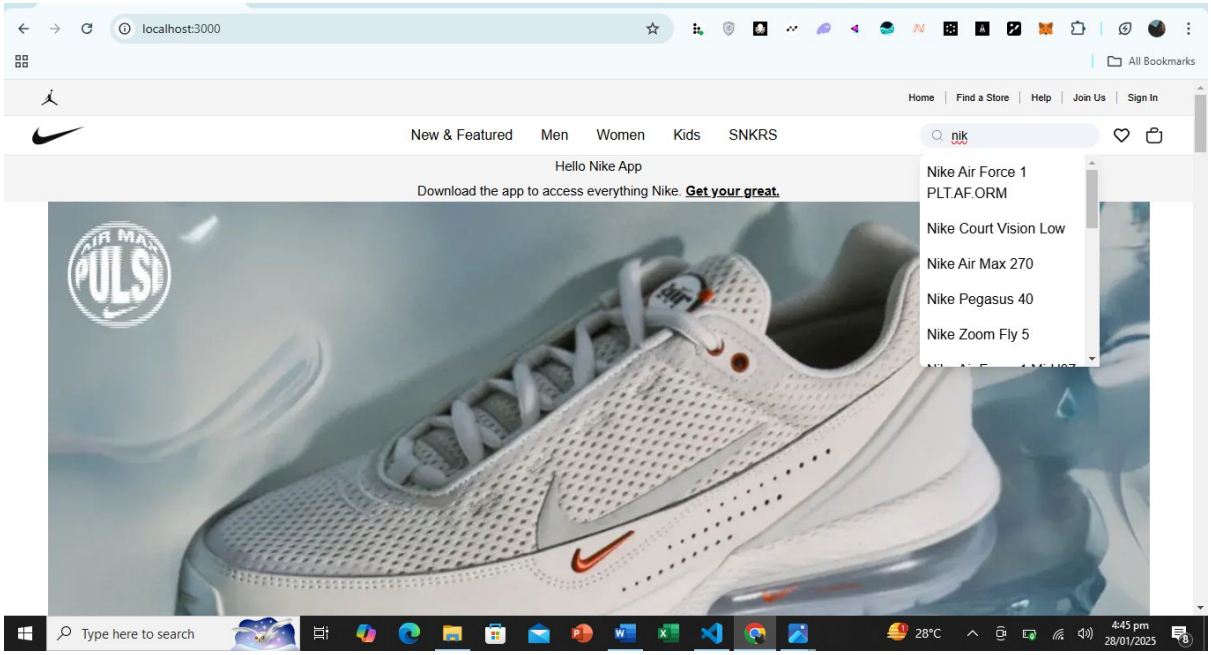




<<<<<<<<<<<<<<<<<<< Fully functional Search bar >>>>>>>>>>>>>>>>>>>

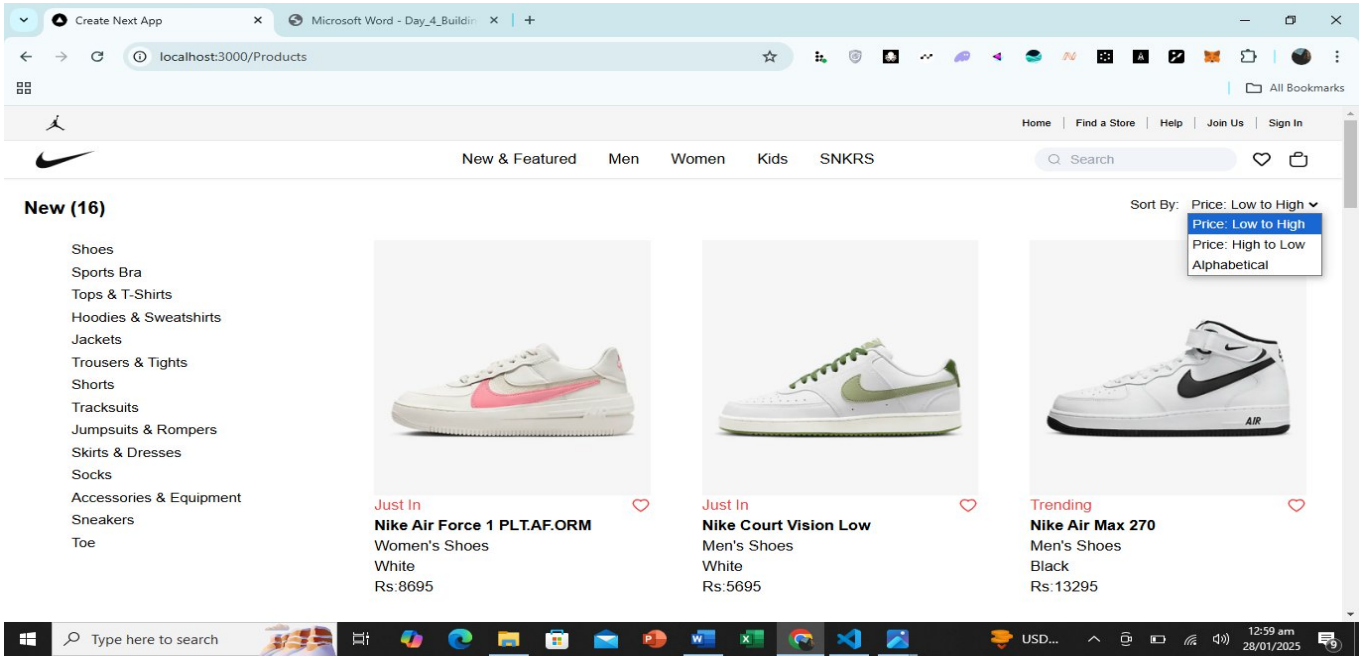
The search bar allows users to type and search for specific products. When users enter a keyword,

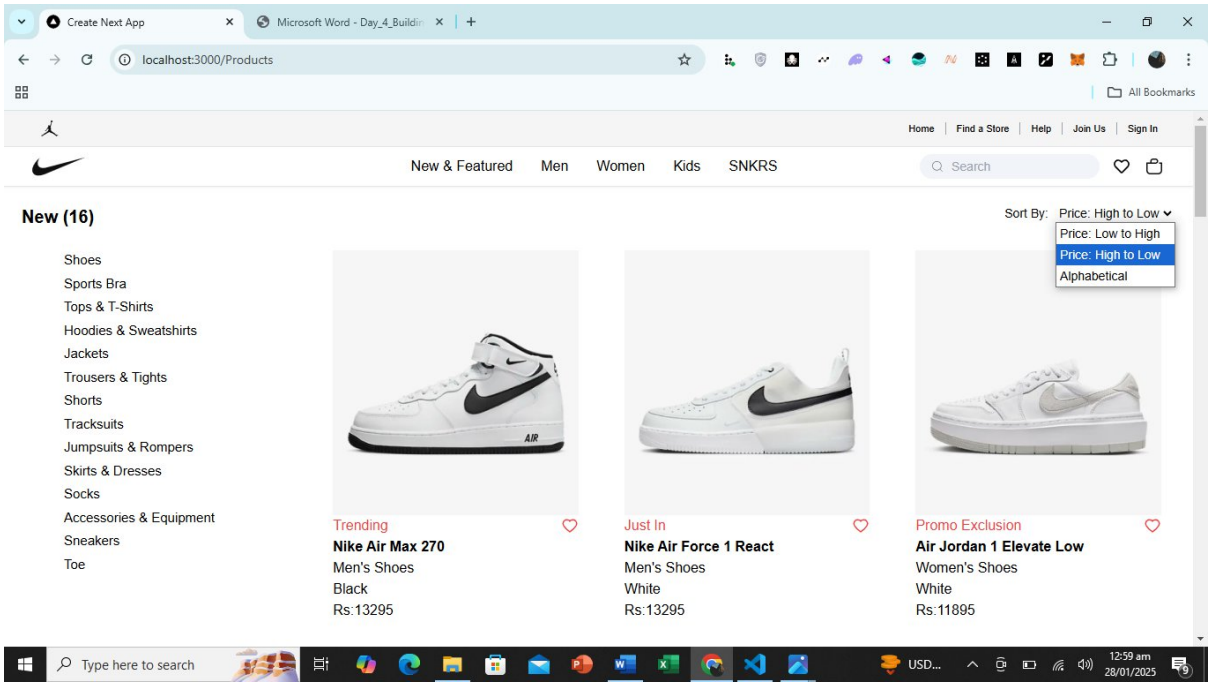
the products related to that keyword will appear. This feature helps users find products quickly without scrolling through the entire list.



<<<<<<<<<<<<<<<<<<<<<<Sort by >>>>>>>>>>>>>>>>>>>>

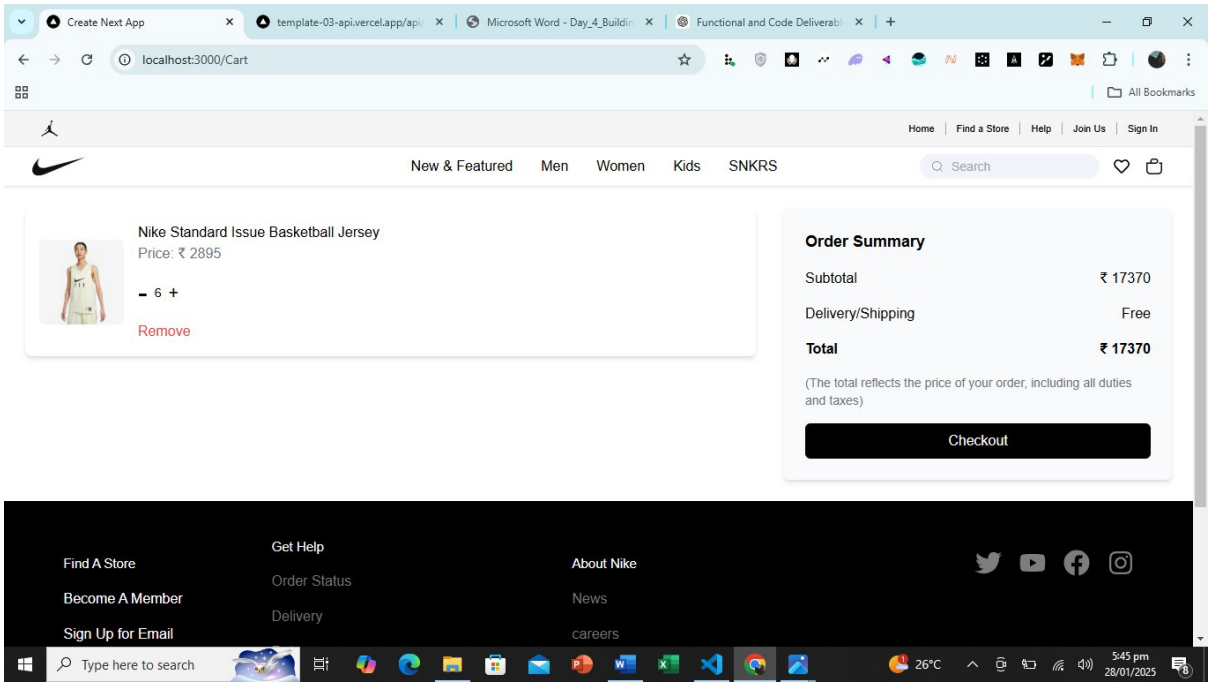
This allows users to sort the products in different ways, such as by price or by newest arrival. It makes it easier for users to find products based on their preferences.





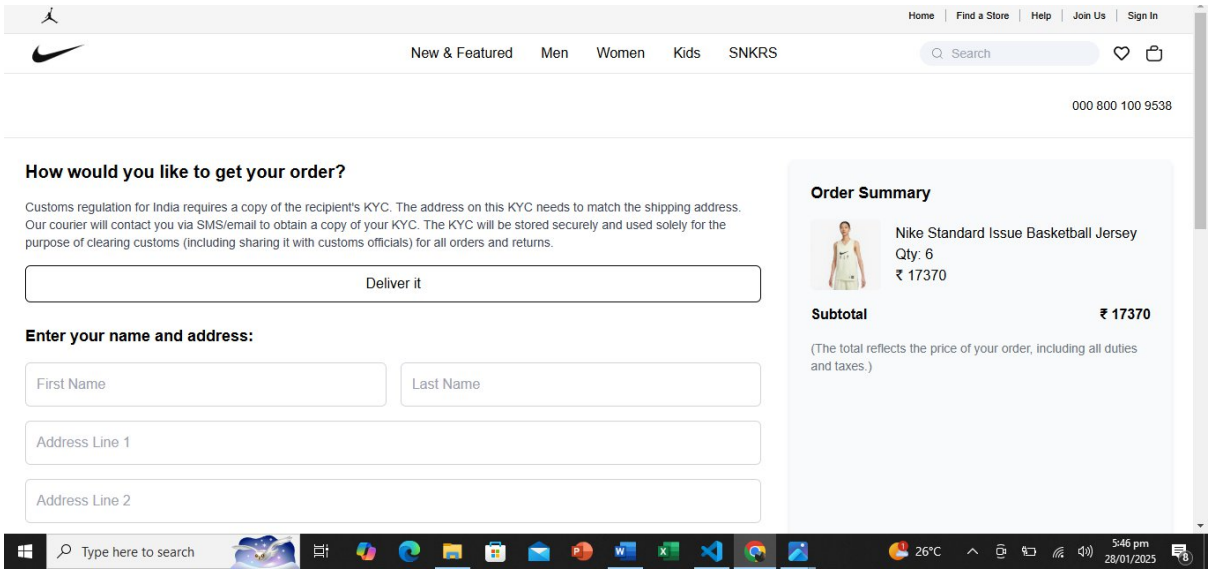
<<<<<<<<<<<<<<<Add to Cart Functionailty>>>>>>>>>>

When users find a product they like, they can add it to their shopping cart. The cart saves the product, so users can proceed to checkout later. The functionality is set up to update the cart whenever a new item is added.



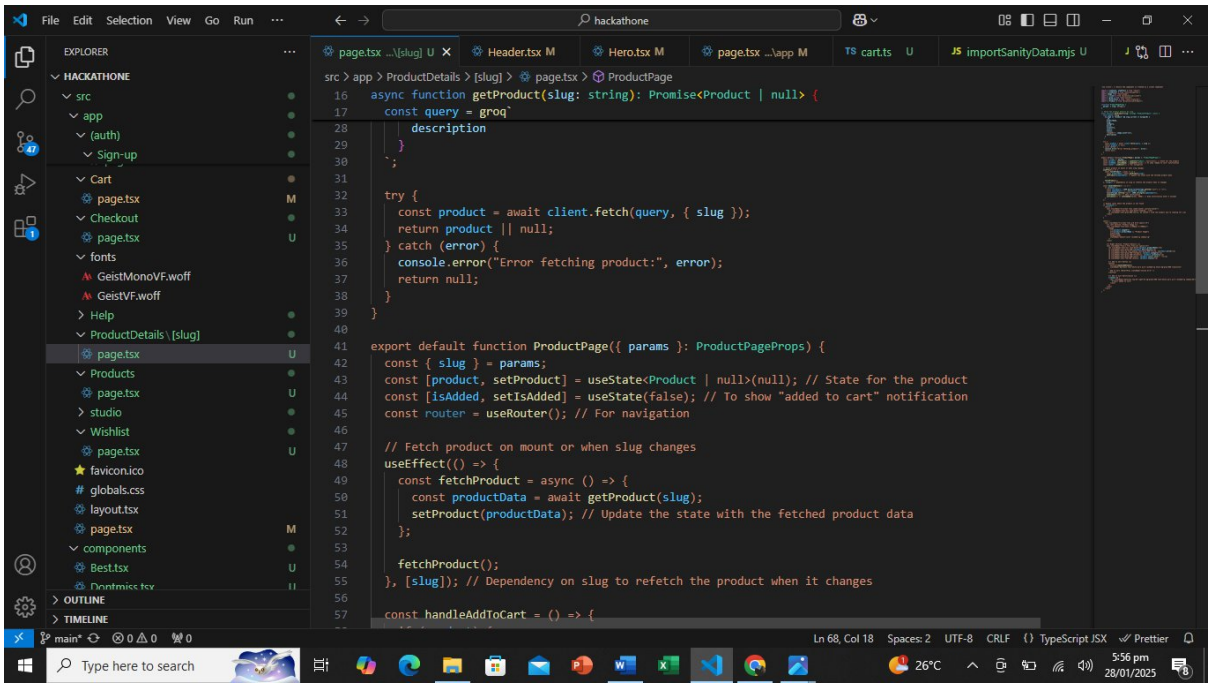
<<<<<<<Checkout Page >>>>>>>>>>>

This page is where users finalize their orders. They can provide their details (like shipping address) and make payments. After completing the checkout, the order is placed, and users will receive a confirmation.



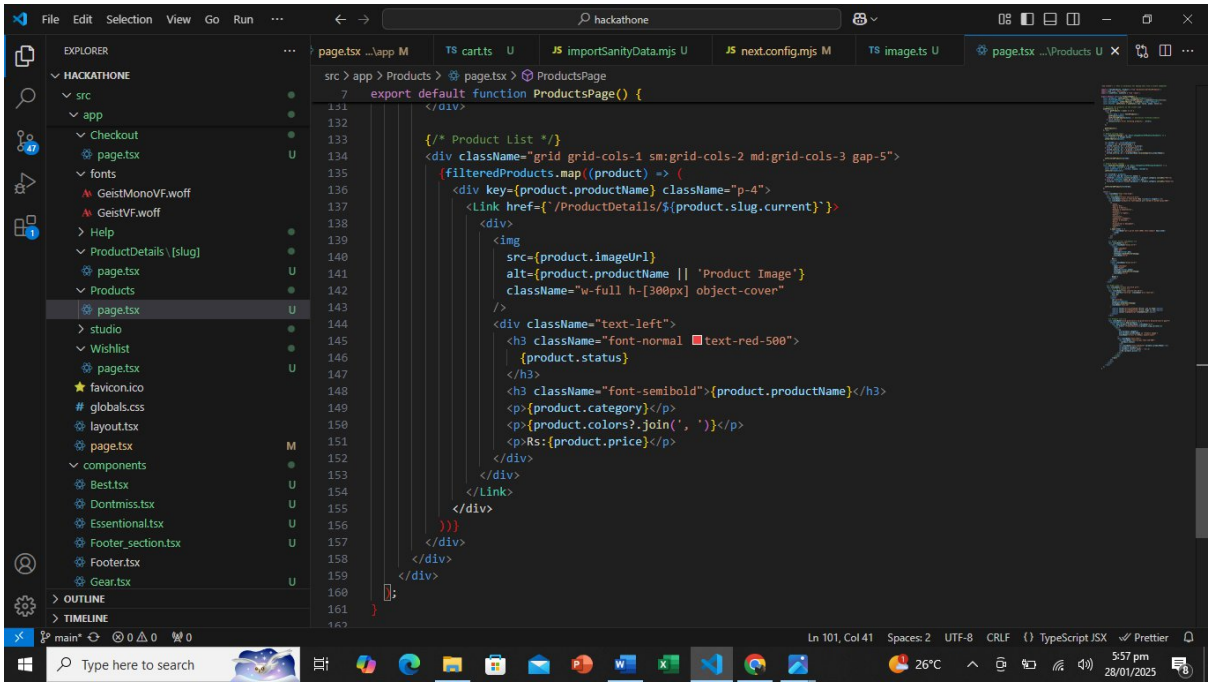
<<<<<<<<<<code snipts>>>>>>>>>>>>

Product detail page dynamically working

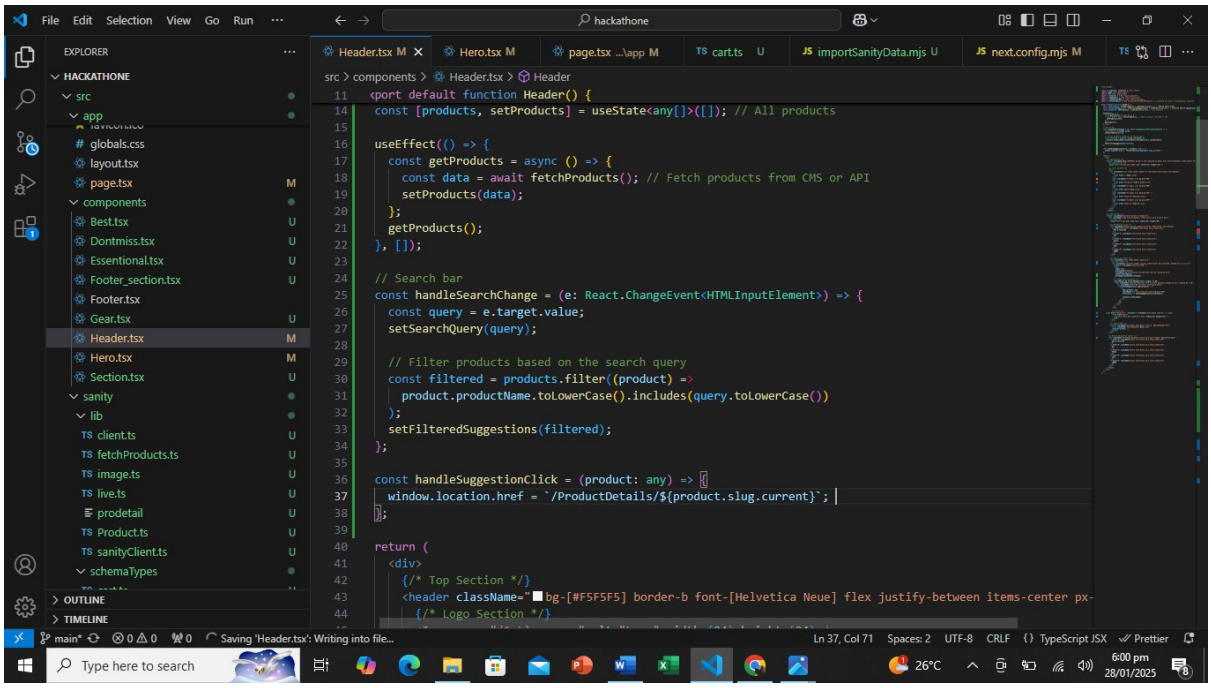


<<<<<<<<<<<<< Product Listing page >>>>>>>>>>>

Product listing page code



<<<<<<<<<<<<<Search bar Functionalty>>>>>>>>>>



Steps Taken to Build and Integrate Components:

1. **Dynamic Product Listing:**
    - **Using Hooks and Map Function:** I started by making the product page dynamic. I used React hooks (useState and useEffect) to fetch product data from the backend (likely from a CMS or local API). The map function was then used to loop through the fetched products and display them on the UI.
    - **UI Improvement:** I focused on making the UI more user-friendly and visually appealing by styling the product cards with images, names, and prices.
2. **“Add to Cart” Functionality:**
    - I added the "Add to Cart" functionality where users can select products and add them to their cart. This was implemented dynamically, ensuring that when a user clicks on the "Add to Cart" button, the selected item is stored and displayed in the cart. The cart updates accordingly whenever a new product is added.
3. **Search Bar Integration:**
    - I implemented a fully functional search bar that allows users to type in keywords and filter products in real-time. As the user types, the products are filtered based on the search query, which helps users find products more efficiently.
4. **Sort By Price Functionality:**
    - I added a “Sort By” feature that allows users to sort the products by price, either from low to high or high to low. This was useful for improving the user experience, especially for e-commerce websites where users want to compare prices quickly.

Challenges Faced and Solutions Implemented:

1. **Product Images Not Showing:**
    - Initially, my product images were not displaying correctly on the product listing page. After troubleshooting, I realized that the image paths were incorrect or missing. To fix this, I ensured that the image URLs were correctly linked to the backend and that each product had a valid image URL. Once corrected, the images were displayed as expected.
2. **Detail Page URL Not Working:**
    - I faced an issue where the product detail page URL was not working correctly, leading to a 404 error when trying to access a product's details. I had mistakenly put an incorrect URL structure. After investigating the routing logic, I corrected the URL pattern in the dynamic route, ensuring that each product had a unique URL based on its ID or slug.
3. **Dynamic Data Handling:**

- Another challenge was handling dynamic data with state management. Initially, I struggled with making the data load correctly when the page re-rendered. I addressed this by making sure I was using the right hook dependencies and handling API responses properly to avoid stale or incorrect data being displayed.

4. **Handling Search Functionality:**
    - I faced some difficulty in making the search bar work dynamically with the mapped product list. At first, the filtering wasn't responsive. To resolve this, I adjusted the logic to ensure that the filter applied only after the user had typed something, and I optimized the state updates to prevent unnecessary re-renders.

Best Practices Followed During Development:

1. **Component Reusability:**
    - I made sure to break down the product page into reusable components, such as product cards and the search bar. This made the codebase cleaner and more maintainable, as I could reuse these components across different parts of the website.
2. **State Management:**
    - I used React's useState and useEffect hooks effectively for managing and updating state based on user interactions (e.g., adding to cart, filtering products). This allowed me to maintain a responsive and dynamic interface.
3. **Error Handling:**
    - When facing issues like missing images or incorrect URLs, I implemented proper error handling and debugging techniques. I used console.log() and inspected the network requests to ensure the data was being fetched and rendered correctly.
4. **Optimized Rendering:**
    - To improve performance, I made sure that I didn't unnecessarily re-render components. I only re-rendered components when their state changed, preventing any performance issues related to too many unnecessary re-renders.
5. **Responsive Design:**
    - I made the UI more user-friendly by ensuring that it was responsive. This meant using CSS media queries or Tailwind CSS utilities to adjust the layout based on the screen size.