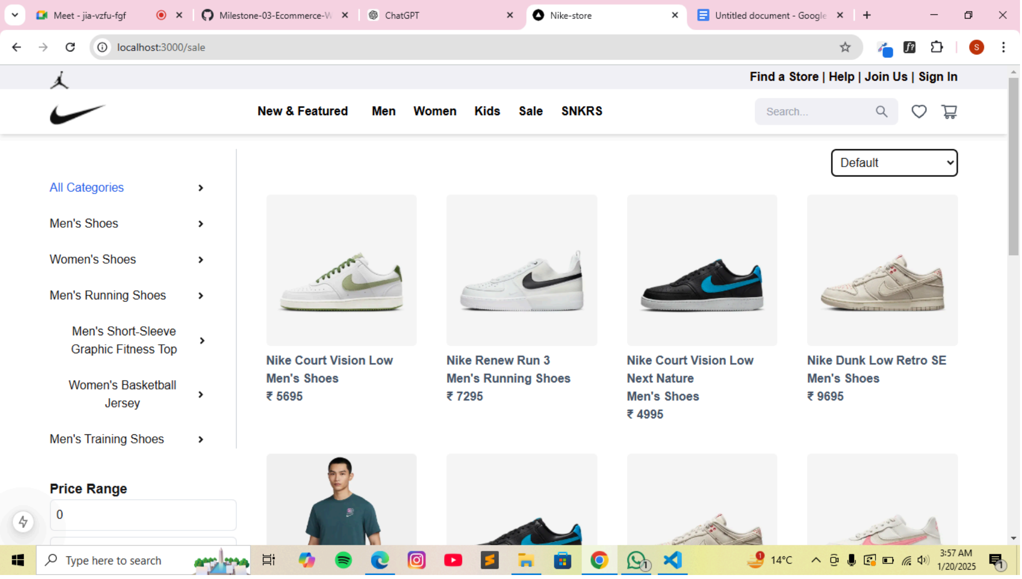
DAY 5 HACKATHON

SCREEN SHOTS:

The product listing page with dynamic data.



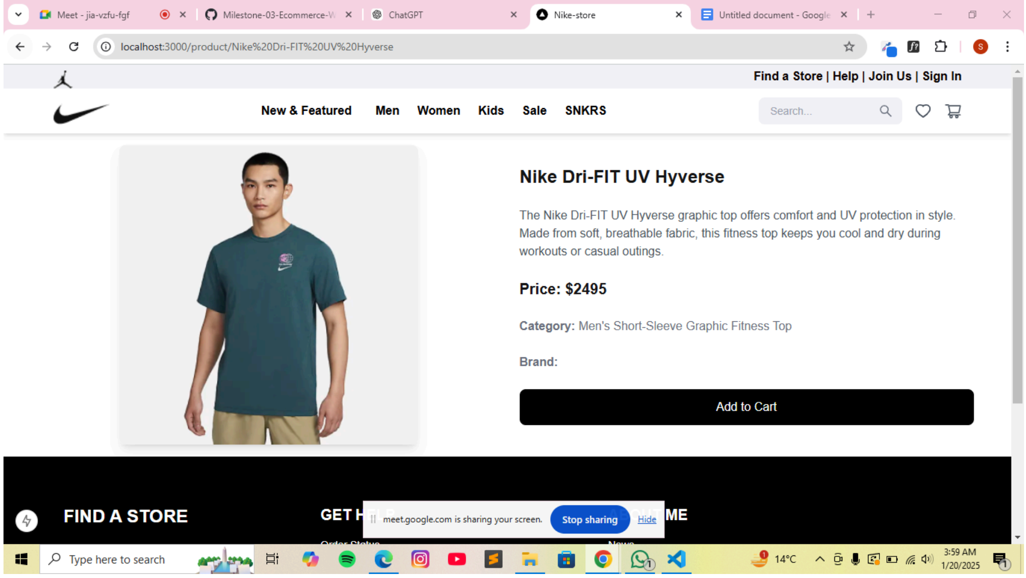
Nike Court Vision Low
Men's Shoes
₹ 5695

Nike Renew Run 3
Men's Running Shoes
₹ 7295

Nike Court Vision Low Next Nature
Men's Shoes
₹ 4995

Nike Dunk Low Retro SE
Men's Shoes
₹ 9695

Individual product detail pages with accurate routing and data rendering.



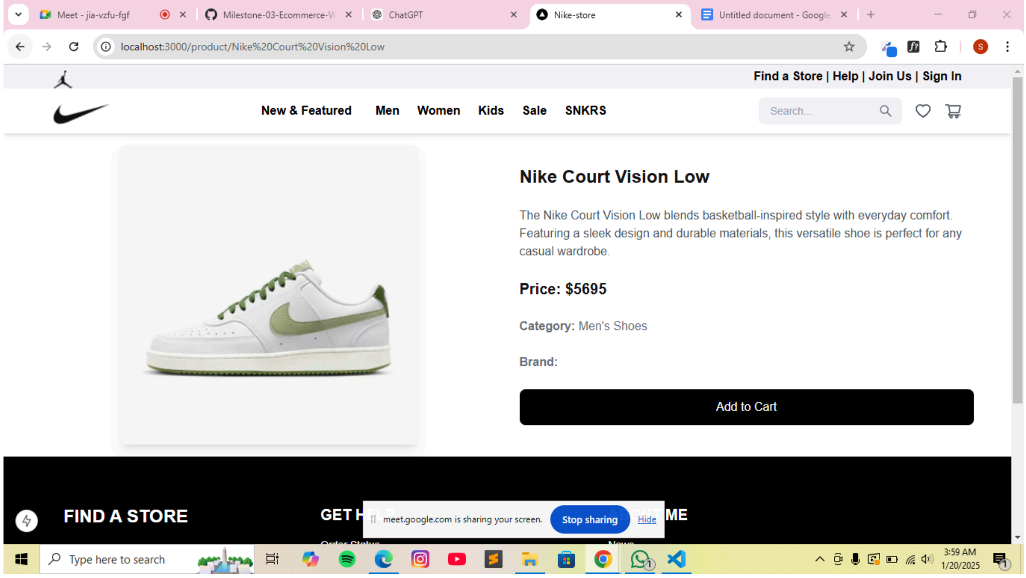
Nike Dri-FIT UV Hyverse

The Nike Dri-FIT UV Hyverse graphic top offers comfort and UV protection in style. Made from soft, breathable fabric, this fitness top keeps you cool and dry during workouts or casual outings.

Price: $2495

Category: Men's Short-Sleeve Graphic Fitness Top

Brand:



Nike Court Vision Low

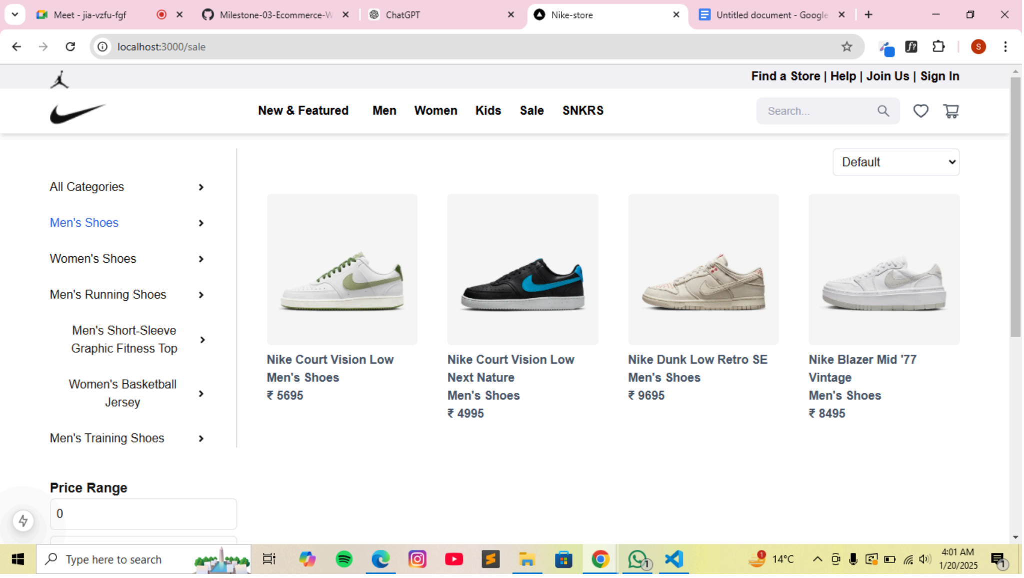
The Nike Court Vision Low blends basketball-inspired style with everyday comfort. Featuring a sleek design and durable materials, this versatile shoe is perfect for any casual wardrobe.

Price: $5695

Category: Men's Shoes

Brand:

Working category filters

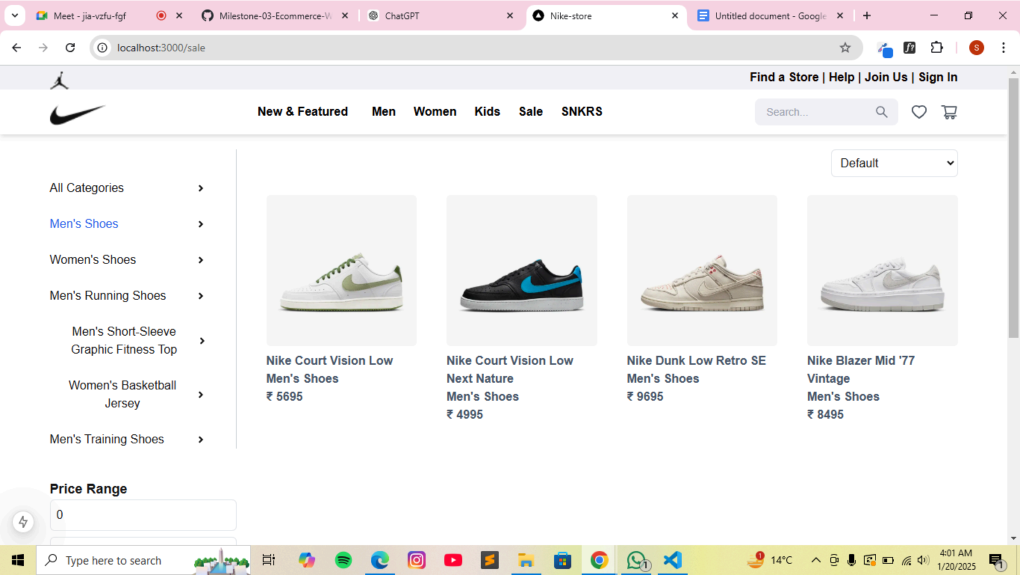


Nike Court Vision Low
Men's Shoes
₹ 5695

Nike Court Vision Low Next Nature
Men's Shoes
₹ 4995

Nike Dunk Low Retro SE
Men's Shoes
₹ 9695

Nike Blazer Mid '77 Vintage
Men's Shoes
₹ 8495


Nike Court Vision Low
Men's Shoes
₹ 5695

Nike Court Vision Low Next Nature
Men's Shoes
₹ 4995

Nike Dunk Low Retro SE
Men's Shoes
₹ 9695

Nike Blazer Mid '77 Vintage
Men's Shoes
₹ 8495

Technical Report: Building and Integrating Components for Sale Page

Overview

This technical report summarizes the steps taken to build and integrate the components for the Sale Page, the challenges encountered during development, the solutions implemented to address these challenges, and the best practices followed throughout the process.

Steps Taken to Build and Integrate Components

1. **Project Setup**:
   - Initialized the project using Next.js for server-side rendering and component-based development.
   - Configured the Sanity CMS to fetch and manage product data efficiently.
2. **Data Fetching and Integration**:
   - Defined a query in Sanity CMS to retrieve product data, including product ID, name, category, price, and image.
   - Implemented the `fetchData` function using Sanity's client library to fetch data asynchronously and store it in the `products` state.
   - Extracted unique product categories from the data and added an "All Categories" option to ensure all products could be displayed by default.
3. **UI Components Development**:
   - Designed a responsive layout using Tailwind CSS with separate sections for the sidebar (categories and price filter) and product display grid.
   - Added dynamic category filtering, allowing users to filter products based on their selected category.
   - Implemented a price range filter with two input fields for minimum and maximum price values.
4. **Sorting Feature**:
   - Created a dropdown menu to allow users to sort products by price, with options for "Default" and "Price: High to Low".
   - Integrated the sorting logic into the filtered product pipeline to dynamically update the displayed products.
5. **Responsive Sidebar**:
   - Developed a toggleable sidebar for mobile screens, ensuring accessibility and a seamless user experience.

Challenges Faced and Solutions Implemented

1. Category Filtering Issue:

- **Challenge**: Extracting unique categories from fetched product data resulted in a TypeScript error due to type mismatch (`unknown[]` vs. `string[]`).

- **Solution**: Used `Array.from(new Set(...))` to ensure the extracted categories were correctly typed as `string[]`.

2. Price Range Validation:

- **Challenge**: Users entering invalid or extreme values in the price range input fields caused unexpected filtering results.
- **Solution**: Added input validation by clamping the minimum and maximum values to predefined limits and ensuring numeric input.

3. Image URL Handling:

- **Challenge**: Displaying product images required proper URL transformation from Sanity's asset structure.
- **Solution**: Used Sanity's `urlFor` helper function to generate correct and optimized URLs for images.

4. Sorting Conflicts:

- **Challenge**: Sorting and filtering logic occasionally conflicted, leading to incorrect product order.
- **Solution**: Ensured the sorting operation was performed after applying all filters by maintaining a strict pipeline for data manipulation.

Best Practices Followed During Development

1. **Component Reusability**:
   - Broke down the UI into smaller, reusable components to ensure modularity and maintainability.
2. **Responsive Design**:
   - Used Tailwind CSS to create a mobile-first design that adapts seamlessly to various screen sizes.
3. **TypeScript for Type Safety**:
   - Leveraged TypeScript to enforce type safety, reducing runtime errors and improving code reliability.
4. **Error Handling**:
   - Wrapped asynchronous functions in `try-catch` blocks to handle errors gracefully and log issues for debugging.
5. **State Management**:
   - Used React's `useState` and `useEffect` hooks to manage application state effectively and implement side effects in a structured manner.
6. **Code Readability and Comments**:
   - Ensured the code was well-documented with comments explaining key logic and decisions.

7. **User Experience (UX):**
    - Focused on creating an intuitive interface with clear visual cues and easy navigation.
