Day 4 - Dynamic Frontend Components - Dukandaro

A technical report summarizing:

Steps Taken to Build and Integrate Components

1- Research and Planning:

- Started by researching the key features needed for an e-commerce website. Features like search, filter, add-to-cart, payment processing, wishlist, authentication, and dynamic product pages were identified as essential.

2- Feature Implementation:

- **Search and Filter:** Added functionality to help users easily find products.
- **Add-to-Cart:** Used the `use-shopping-cart` library, which provided a simple way to implement add-to-cart features.
- **Payment Integration:** Integrated Stripe for secure and efficient payment processing, leveraging the `use-shopping-cart` library for seamless implementation.
- **Wishlist:** Created a wishlist feature to allow users to save products they like.
- **Authentication:** Used Clerk to handle user authentication in a straightforward and secure way.
- **Dynamic Pages:** Built product detail pages dynamically, ensuring users can view product-specific information.
- **Search and Filter Logic:** Implemented search and filter functionalities using Next.js `searchParams` to make the process dynamic and user-friendly.

Challenges Faced and Solutions Implemented

Payment Processing:

- **Problem:** Initially struggled to find a secure and user-friendly payment solution.
- **Solution:** Researched and integrated Stripe, which provided a reliable API. The `use-shopping-cart` library simplified the process further.

Dynamic Search and Filtering:

- **Problem:** Managing dynamic search and filter logic was challenging.
- **Solution:** Used Next.js `searchParams` to handle these features efficiently and maintain a smooth user experience.

Authentication Setup:

- **Problem:** Needed a robust authentication system that was easy to integrate.
- **Solution:** Clerk was chosen for its simplicity and comprehensive features, making the setup process quick and effective.

Best Practices Followed

1. **Modular Development:**
    - Created reusable components to simplify development and improve maintainability.
2. **Security First:**
    - Prioritized secure integrations with trusted tools like Stripe and Clerk.
3. **User Experience Focus:**
    - Designed a clean and intuitive interface to ensure the website is easy to use.
4. **Performance Optimization:**
    - Used server-side rendering and dynamic routing in Next.js to improve performance and loading times.
5. **Scalable Design:**
    - Built the system with scalability in mind to handle future growth and new features.

Functional Deliverables

Features Demonstrated:

1. **Search and Filter Functionality:**
    - Implemented a responsive search bar and filter options.
    - Used `searchParams` for seamless integration.
2. **Add to Cart and Wishlist:**
    - Added "Add to Cart" functionality using the `useShoppingCart` library for efficient integration.

    - Wishlist functionality to allow users to save their favorite items.
3. **Payment Integration with Stripe:**
    - Solved the payment gateway issue by integrating Stripe for secure and easy transactions.
    - Enabled checkout functionality with minimal setup.
4. **Dynamic Product Pages:**
    - Fully dynamic product detail pages rendering accurate data.
    - Routing set up for each product dynamically using Next.js.
5. **Authentication**:
    - Integrated Clerk for user authentication.
    - Simplified login and signup processes for users.

Video Demonstration

A detailed video showcasing:

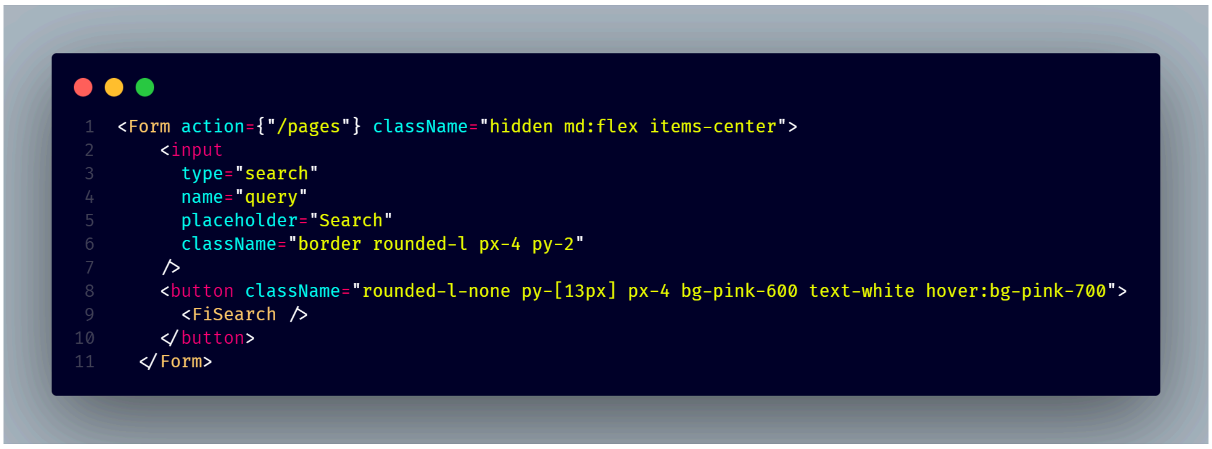
- Search and filter in action.
- Adding items to the cart and wishlist.
- Payment process using Stripe.
- Dynamic product pages and authentication workflows.

Link: Recording 2025-01-21 032518.mp4

Code Deliverables

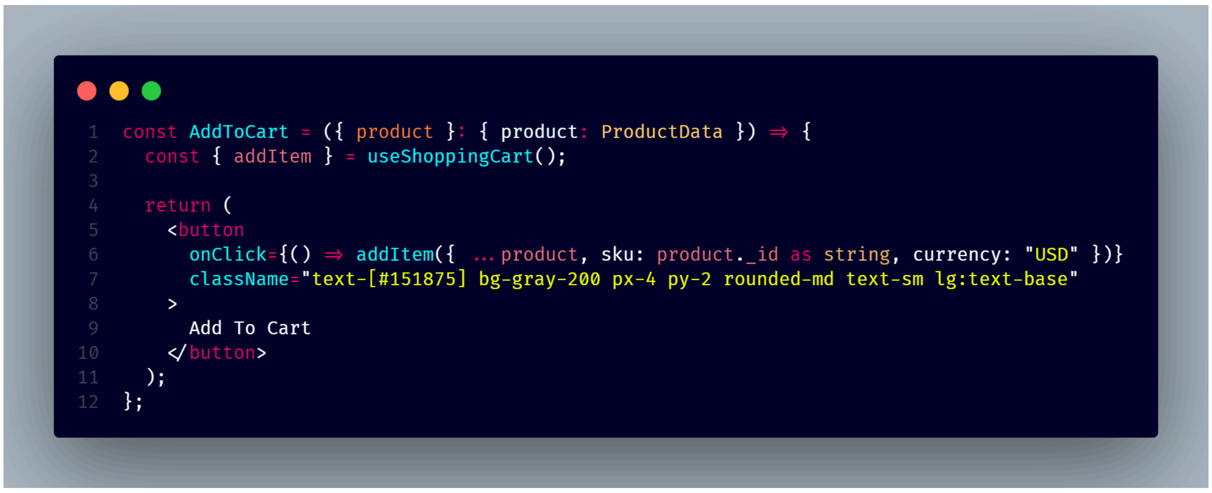
Key Code Snippets

- **Search Functionality using next/form**



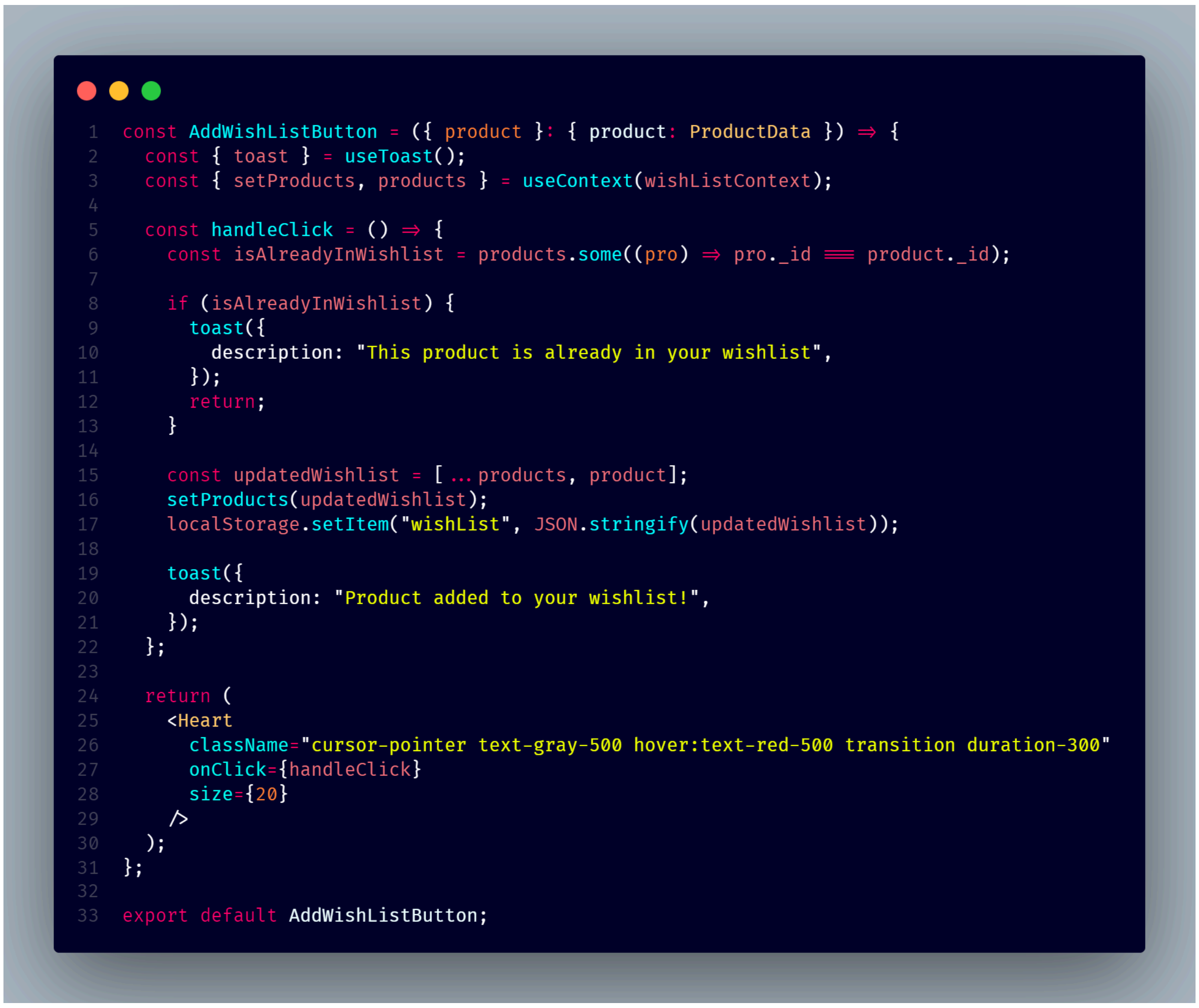
```
<Form action={"/pages"} className="hidden md:flex items-center">
    <input
      type="search"
      name="query"
      placeholder="Search"
      className="border rounded-l px-4 py-2"
    />
    <button className="rounded-l-none py-[13px] px-4 bg-pink-600 text-white hover:bg-pink-700">
      <FiSearch />
    </button>
  </Form>
```

- **Add To Cart functionality using useShoppingCart()**



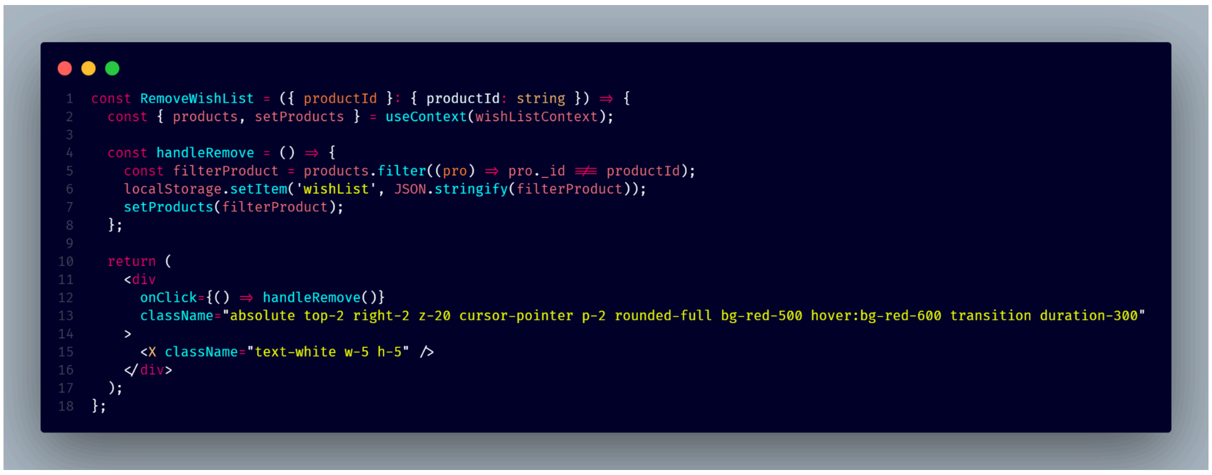
```tsx
const AddToCart = ({ product }: { product: ProductData }) => {
  const { addItem } = useShoppingCart();

  return (
    <button
      onClick={() => addItem({ ...product, sku: product._id as string, currency: "USD" })}
      className="text-[#151875] bg-gray-200 px-4 py-2 rounded-md text-sm lg:text-base"
    >
      Add To Cart
    </button>
  );
};
```

- **Add To WishList functionality using ContextApi**

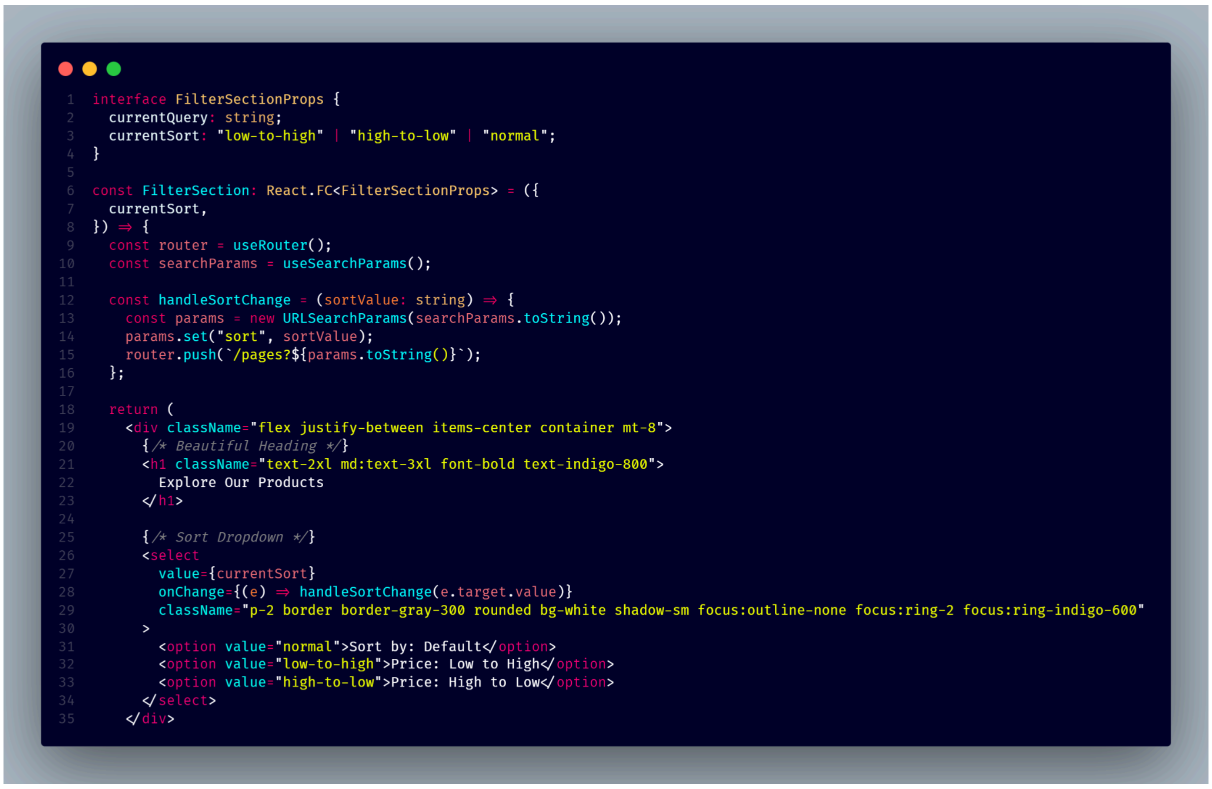


```tsx
const AddWishListButton = ({ product }: { product: ProductData }) => {
  const { toast } = useToast();
  const { setProducts, products } = useContext(wishListContext);

  const handleClick = () => {
    const isAlreadyInWishlist = products.some((pro) => pro._id === product._id);

    if (isAlreadyInWishlist) {
      toast({
        description: "This product is already in your wishlist",
      });
      return;
    }

    const updatedWishlist = [...products, product];
    setProducts(updatedWishlist);
    localStorage.setItem("wishList", JSON.stringify(updatedWishlist));

    toast({
      description: "Product added to your wishlist!",
    });
  };

  return (
    <Heart
      className="cursor-pointer text-gray-500 hover:text-red-500 transition duration-300"
      onClick={handleClick}
      size={20}
    />
  );
};

export default AddWishListButton;
```

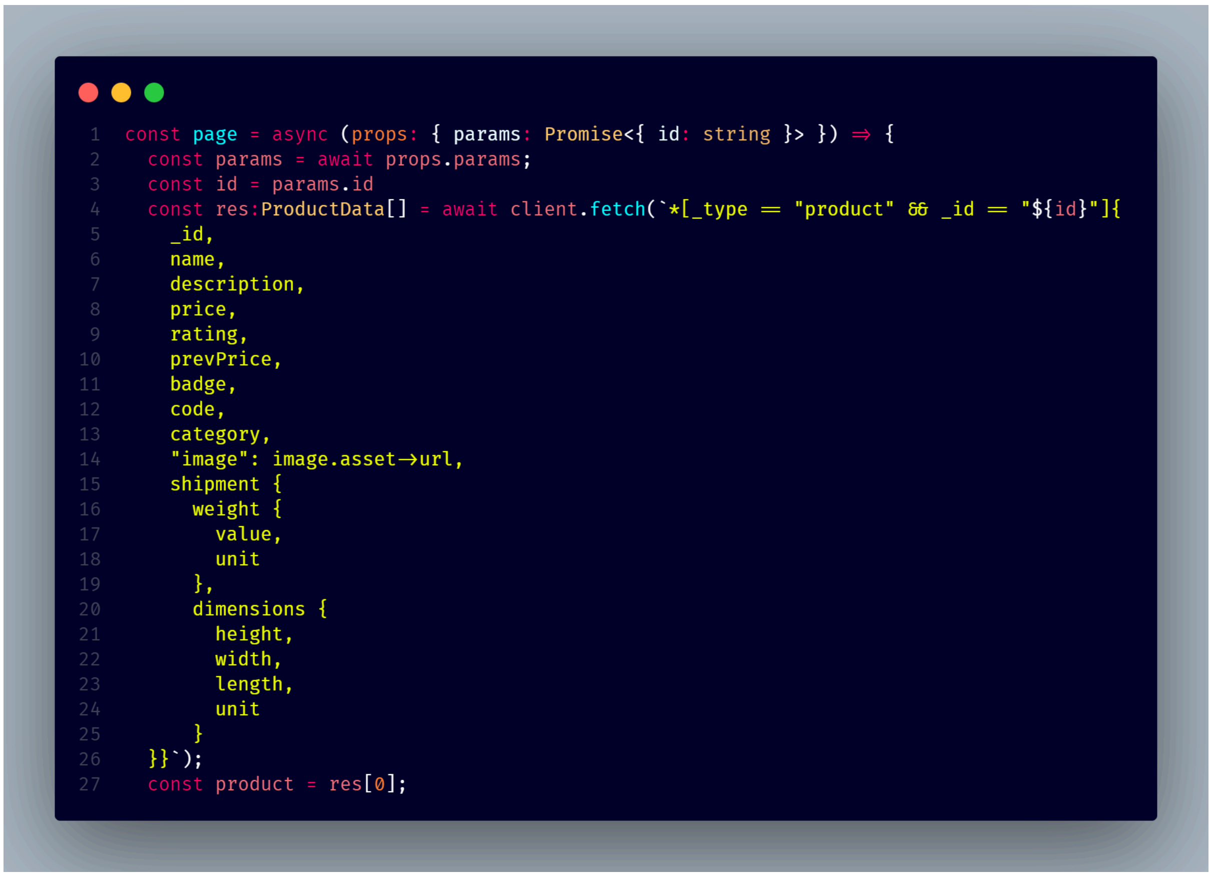
- **Remove Add wishList Functionality**


```tsx
const RemoveWishList = ({ productId }: { productId: string }) => {
  const { products, setProducts } = useContext(wishListContext);

  const handleRemove = () => {
    const filterProduct = products.filter((pro) => pro._id !== productId);
    localStorage.setItem('wishList', JSON.stringify(filterProduct));
    setProducts(filterProduct);
  };

  return (
    <div
      onClick={() => handleRemove()}
      className="absolute top-2 right-2 z-20 cursor-pointer p-2 rounded-full bg-red-500 hover:bg-red-600 transition duration-300"
    >
      <X className="text-white w-5 h-5" />
    </div>
  );
};
```

- **Filter product acc to Price**


```tsx
interface FilterSectionProps {
  currentQuery: string;
  currentSort: "low-to-high" | "high-to-low" | "normal";
}

const FilterSection: React.FC<FilterSectionProps> = ({
  currentSort,
}) => {
  const router = useRouter();
  const searchParams = useSearchParams();

  const handleSortChange = (sortValue: string) => {
    const params = new URLSearchParams(searchParams.toString());
    params.set("sort", sortValue);
    router.push(`/pages?${params.toString()}`);
  };

  return (
    <div className="flex justify-between items-center container mt-8">
      {/* Beautiful Heading */}
      <h1 className="text-2xl md:text-3xl font-bold text-indigo-800">
        Explore Our Products
      </h1>

      {/* Sort Dropdown */}
      <select
        value={currentSort}
        onChange={(e) => handleSortChange(e.target.value)}
        className="p-2 border border-gray-300 rounded bg-white shadow-sm focus:outline-none focus:ring-2 focus:ring-indigo-600"
      >
        <option value="normal">Sort by: Default</option>
        <option value="low-to-high">Price: Low to High</option>
        <option value="high-to-low">Price: High to Low</option>
      </select>
    </div>
```

- **Dynamic Routing and fetching Data**



```
const page = async (props: { params: Promise<{ id: string }> }) => {
  const params = await props.params;
  const id = params.id
  const res:ProductData[] = await client.fetch(`*[_type == "product" && _id == "${id}"]{
    _id,
    name,
    description,
    price,
    rating,
    prevPrice,
    badge,
    code,
    category,
    "image": image.asset->url,
    shipment {
      weight {
        value,
        unit
      },
      dimensions {
        height,
        width,
        length,
        unit
      }
  }}`);
  const product = res[0];
```