NAME: **RUBA HAROON**

ROLL NO: **00441882**

SLOT: **FRIDAY 9 TO 12**

TEACHER: **SIR HAMZAH SYED**

Detailed Documentation for Dynamic Components and Functionalities:

This documentation provides an in-depth analysis of the key functionalities for a dynamic marketplace, emphasizing modularity, reusability, and integration with Sanity CMS. Each feature is described comprehensively, followed by a conclusion summarizing the approach.

Step 1: Functionalities Overview:

The project implements the following core functionalities:

1. Product Listing Page
2. Dynamic Route
3. Cart Functionality
4. Checkout
5. Price Calculation
6. Product Comparison
7. Inventory Management

Each functionality contributes to building a responsive and scalable marketplace.

Step 2: Functionalities in Detail

1. Product Listing Page

The Product Listing Page is the primary interface where users can view all the available products in a structured and visually appealing format. Products are displayed dynamically, fetched from Sanity CMS, and rendered in a grid or list layout.

```
"Use client";
import React, { useEffect, useState } from "react";
import Link from "next/link";
import { createClient } from "next-sanity";
import Image from "next/image";
import { motion, AnimatePresence } from "framer-motion";
import Slider from "rc-slider";
import "rc-slider/assets/index.css";
import Pagination from "@/components/pagination";
import CategoryFilterBar from "@/components/categoryfilterbar";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import ProductComparison from "@/components/productcomparision"; // Import the
ProductComparison component

// Sanity client setup
const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  useCdn: true,
  apiVersion: "2025-01-20",
});

type Product = {
  _id: string;
  name: string;
  slug: { current: string };
  imageUrl: string;
```

```
  price: number;
  features: string[];
  brand: string;
  rating: number;
  tags: string[];
  inStock: boolean;
  category: { _id: string; name: string };
};

type Category = {
  _id: string;
  name: string;
};

const ProductCard = ({
  product,
  onCompare,
}: {
  product: Product;
  onCompare: (product: Product) => void;
}) => (
  <motion.div
    className="border rounded-lg overflow-hidden shadow-sm hover:shadow-lg
transition-shadow duration-300 cursor-pointer bg-white"
    variants={{
      hidden: { opacity: 0, y: 20 },
      visible: { opacity: 1, y: 0 },
    }}
    initial="hidden"
    animate="visible"
    exit="hidden"
    transition={{ duration: 0.3 }}
  >
    <Link href={`/productlisting/${product.slug.current}`} passHref>
      <div className="w-full h-64 overflow-hidden relative group">
        <Image
          src={product.imageUrl || "/placeholder.png"}
          alt={product.name}
          width={1000}
          height={1000}
          className="object-cover w-full h-full transition-transform duration-300
group-hover:scale-110"
```

```
        />
      </div>
    </Link>
    <div className="p-4">
      <h3 className="font-medium text-lg text-gray-800">{product.name}</h3>
      <p className="text-gray-600">£{product.price}</p>
      <button
        onClick={(e) => {
          e.preventDefault(); // Prevent link navigation
          onCompare(product);
        }}
        className="mt-2 px-4 py-2 bg-[#2A254B] text-white rounded hover:bg-gray-
600"
      >
        Add to Compare
      </button>
    </div>
  </motion.div>
);

export default function ProductListing() {
  const [products, setProducts] = useState<Product[]>([]);
  const [filteredProducts, setFilteredProducts] = useState<Product[]>([]);
  const [currentPage, setCurrentPage] = useState(1);
  const [productsPerPage] = useState(8);
  const [categories, setCategories] = useState<Category[]>([]);
  const [selectedCategories, setSelectedCategories] = useState<string[]>([]);
  const [priceRange, setPriceRange] = useState<[number, number]>([0, 1000]);
  const [inStockOnly, setInStockOnly] = useState<boolean>(false);
  const [comparisonProducts, setComparisonProducts] = useState<Product[]>([]);
  const [showComparison, setShowComparison] = useState<boolean>(false);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const productsQuery = `*[_type == "product"]{
          _id,
          name,
          slug,
          "imageUrl": image.asset->url,
          price,
          features,
```

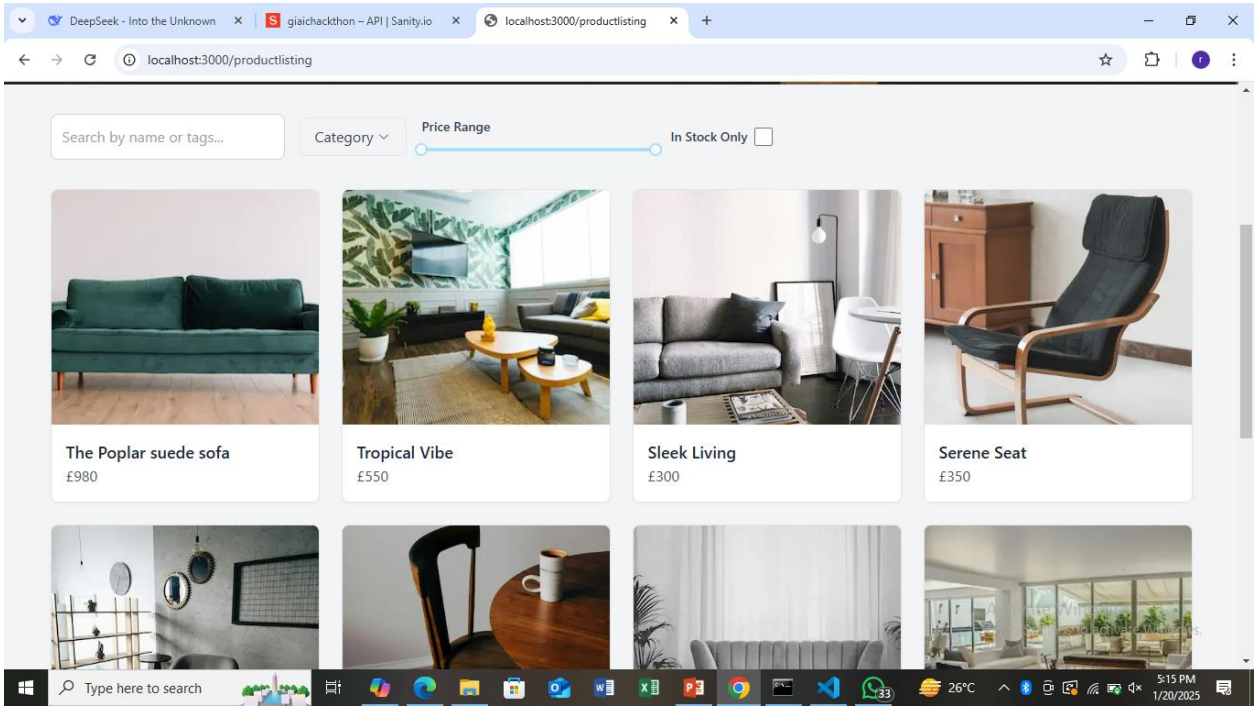
```
        rating,
        tags,
        inStock,
        category->{_id, name}
      }`;
      const productsData = await client.fetch(productsQuery);
      setProducts(productsData);
      setFilteredProducts(productsData);

      const categoriesQuery = `*[_type == "category"]{
        _id,
        name,
      }`;
      const categoriesData = await client.fetch(categoriesQuery);
      setCategories(categoriesData);
    } catch (error) {
      console.error("Error fetching data:", error);
    }
  };
  fetchData();
}, []);

useEffect(() => {
  const filtered = products.filter((product) => {
    const matchesCategory =
      selectedCategories.length === 0 ||
      selectedCategories.includes(product.category._id);
    const matchesPrice =
      product.price >= priceRange[0] && product.price <= priceRange[1];
    const matchesAvailability = !inStockOnly || product.inStock;

    return matchesCategory && matchesPrice && matchesAvailability;
  });

  setFilteredProducts(filtered);
  setCurrentPage(1);
}, [selectedCategories, priceRange, inStockOnly, products]);

const handleCategoryChange = (categoryId: string) => {
  setSelectedCategories((prev) =>
    prev.includes(categoryId)
      ? prev.filter((id) => id !== categoryId)
```

```
      : [...prev, categoryId]
  );
};

const handlePriceChange = (value: number | number[]) =>
  setPriceRange(value as [number, number]);

const handleAvailabilityChange = () => setInStockOnly((prev) => !prev);

const handleProductCompare = (product: Product) => {
  if (comparisonProducts.some((p) => p._id === product._id)) {
    toast.warning("This product is already selected for comparison.");
    return;
  }
  if (comparisonProducts.length >= 4) {
    toast.error("You can compare only 4 products at a time.");
    return;
  }
  setComparisonProducts([...comparisonProducts, product]);
  toast.success(`${product.name} added to comparison.`);
};

const handleCompareButtonClick = () => {
  if (comparisonProducts.length < 2) {
    toast.error("Please select at least 2 products to compare.");
    return;
  }
  setShowComparison(true);
};

const handleCloseComparison = () => {
  setShowComparison(false);
  setComparisonProducts([]);
};

const indexOfLastProduct = currentPage * productsPerPage;
const indexOfFirstProduct = indexOfLastProduct - productsPerPage;
const currentProducts = filteredProducts.slice(
  indexOfFirstProduct,
  indexOfLastProduct
);
const totalPages = Math.ceil(filteredProducts.length / productsPerPage);
```

```
const paginate = (pageNumber: number) => setCurrentPage(pageNumber);
const goToNextPage = () =>
  currentPage < totalPages && setCurrentPage(currentPage + 1);
const goToPreviousPage = () =>
  currentPage > 1 && setCurrentPage(currentPage - 1);

return (
  <div className="min-h-screen bg-gray-100">
    <ToastContainer position="bottom-right" autoClose={3000} />
    {/* Hero Section */}
    <div
      className="relative bg-cover bg-center h-64"
      style={{ backgroundImage: "url('../images/ourproduct.png')" }}
    >
      <div className="absolute inset-0 bg-black bg-opacity-40"></div>
      <div className="absolute inset-0 flex items-center justify-center">
        <h1 className="text-white font-bold text-4xl">All Products</h1>
      </div>
    </div>

    {/* Filter Bar */}
    <div className="container mx-auto mt-8 px-4">
      <div className="flex flex-wrap items-center justify-between gap-4">
        {/* Left Side: Category Filter */}
        <div className="flex-grow sm:flex-grow-0">
          <CategoryFilterBar
            categories={categories}
            selectedCategories={selectedCategories}
            handleCategoryChange={handleCategoryChange}
          />
        </div>

        {/* Center: In Stock Only Checkbox */}
        <div className="flex items-center space-x-2">
          <label className="text-sm font-medium text-gray-700">
            In Stock Only
          </label>
          <input
            type="checkbox"
            checked={inStockOnly}
            onChange={handleAvailabilityChange}
```

```
              className="form-checkbox h-5 w-5 text-blue-600 border-gray-300
rounded focus:ring-blue-500"
            />
          </div>

          {/* Right Side: Price Range Slider */}
          <div className="w-64">
            <label className="block text-sm font-medium text-gray-700 mb-2">
              Price Range
            </label>
            <Slider
              min={0}
              max={1000}
              value={priceRange}
              onChange={handlePriceChange}
              range
            />
          </div>
        </div>
      </div>

      {/* Product Grid */}
      <div className="container mx-auto mt-8 px-4">
        <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-
cols-4 gap-6">
          <AnimatePresence>
            {currentProducts.map((product) => (
              <ProductCard
                key={product._id}
                product={product}
                onCompare={handleProductCompare}
              />
            ))}
          </AnimatePresence>
        </div>

        {/* Pagination */}
        <Pagination
          currentPage={currentPage}
          totalPages={totalPages}
          paginate={paginate}
          goToPreviousPage={goToPreviousPage}
```

```
          goToNextPage={goToNextPage}
        />

        {/* Compare Button */}
        {comparisonProducts.length > 0 && (
          <div className="flex justify-center w-full mt-8">
          <button
            onClick={handleCompareButtonClick}
            className="px-4 py-2 bg-[#2A254B] text-white rounded hover:bg-gray-
600 transition-colors duration-300"
          >
            Compare Products ({comparisonProducts.length}/4)
          </button>
        </div>
        )}
      </div>

      {/* Product Comparison Modal */}
      {showComparison && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center
justify-center p-4">
          <div className="bg-white rounded-lg p-6 w-full max-w-6xl">
            <h2 className="text-2xl font-bold mb-4">Product Comparison</h2>
            <ProductComparison products={comparisonProducts} />
            <button
              onClick={handleCloseComparison}
              className="mt-4 px-4 py-2 bg-red-500 text-white rounded hover:bg-
red-600"
            >
              Close
            </button>
          </div>
        </div>
      )}
    </div>
  );
}
```

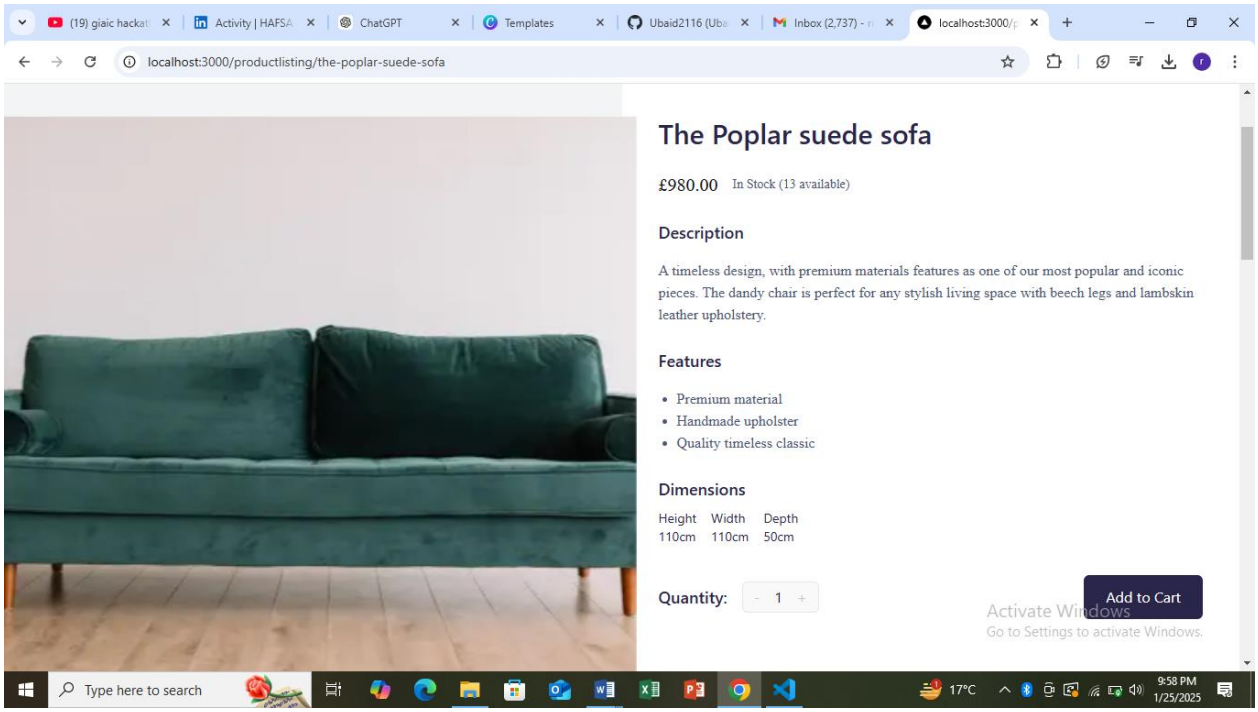


Detailed Description:

- The page offers sorting and filtering options to enhance usability, allowing users to organize products based on price, categories, or popularity.
- Pagination ensures the seamless handling of large datasets, improving performance and user experience.
- Responsive design ensures compatibility across devices, from desktops to mobile phones.
- Integration with Sanity CMS ensures real-time updates, so any product changes in the backend are instantly reflected.

2. Dynamic Route

Dynamic routing allows for the creation of individual product detail pages, enabling users to view detailed information about each product.



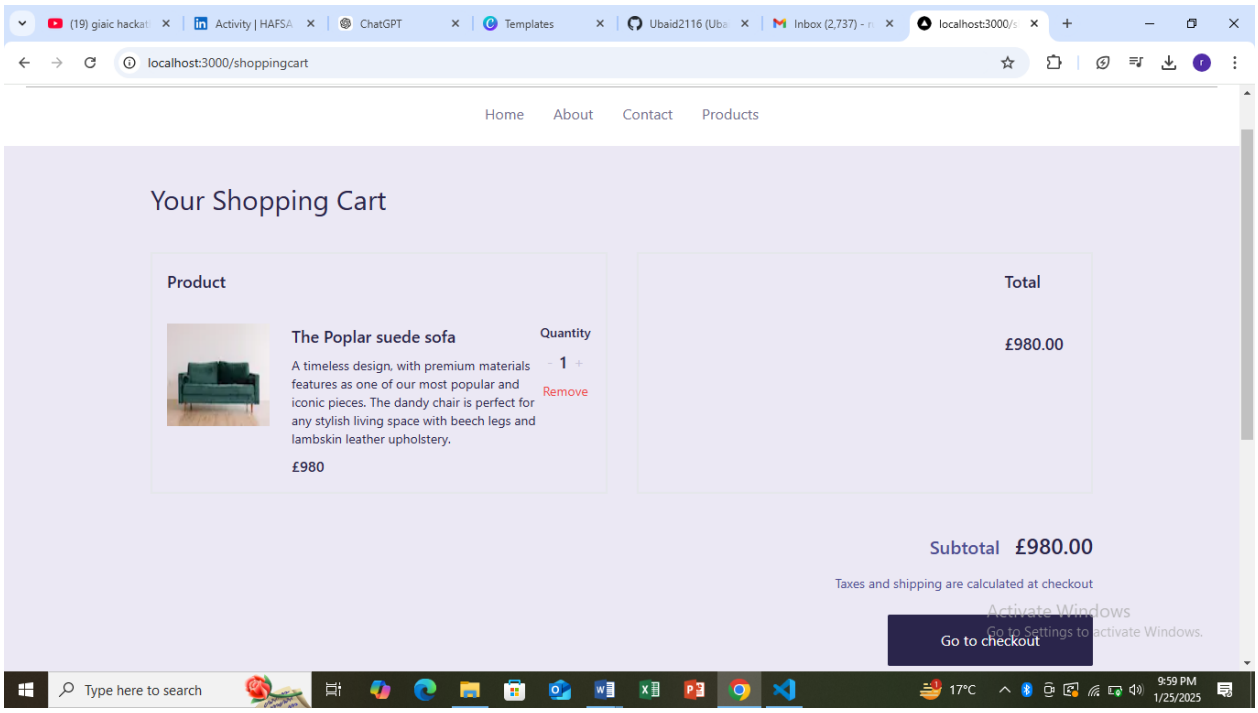
Detailed Description:

- Every product has a unique identifier (ID or slug) used to dynamically generate its URL (e.g., /productlisting/[slug]).
- These pages are server-rendered to improve SEO and provide faster initial load times.
- Dynamic routes display details like product descriptions, images, price, stock status, and reviews.
- This approach ensures scalability, allowing new products to automatically generate corresponding pages without manual intervention.

3. Cart Functionality:

The Cart Functionality manages the user's selected items, providing a seamless

shopping experience by tracking their choices and summarizing costs.

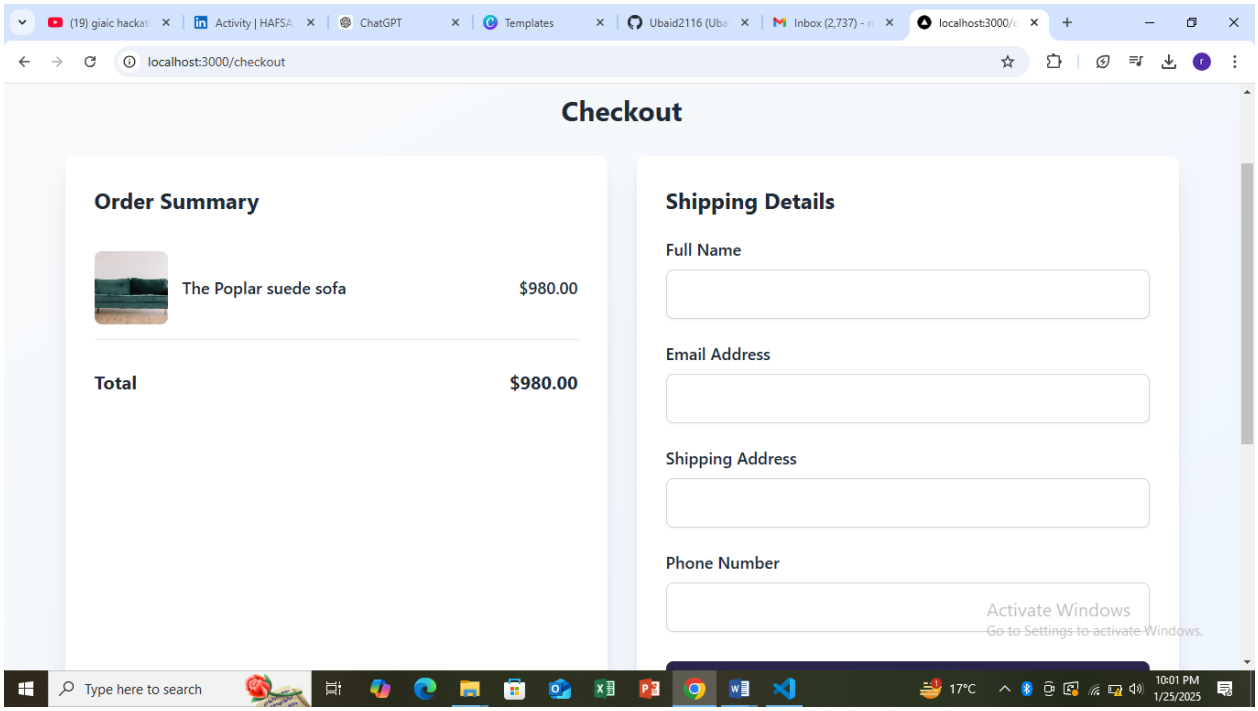


Detailed Description:

- Users can add products to their cart directly from the product listing or detail page.
- The cart dynamically updates quantities and calculates the total cost, ensuring a real-time experience.
- A mini-cart displays a quick summary of selected items, while a detailed cart page offers options to edit or remove items.
- State management tools, such as React Context or Redux, are used to maintain the cart state across the application.
- Cart data persistence is achieved using local storage or session storage, ensuring the cart remains intact even if the page is refreshed.

4. Checkout:

The Checkout functionality streamlines the purchase process, collecting and validating user information to finalize the order.

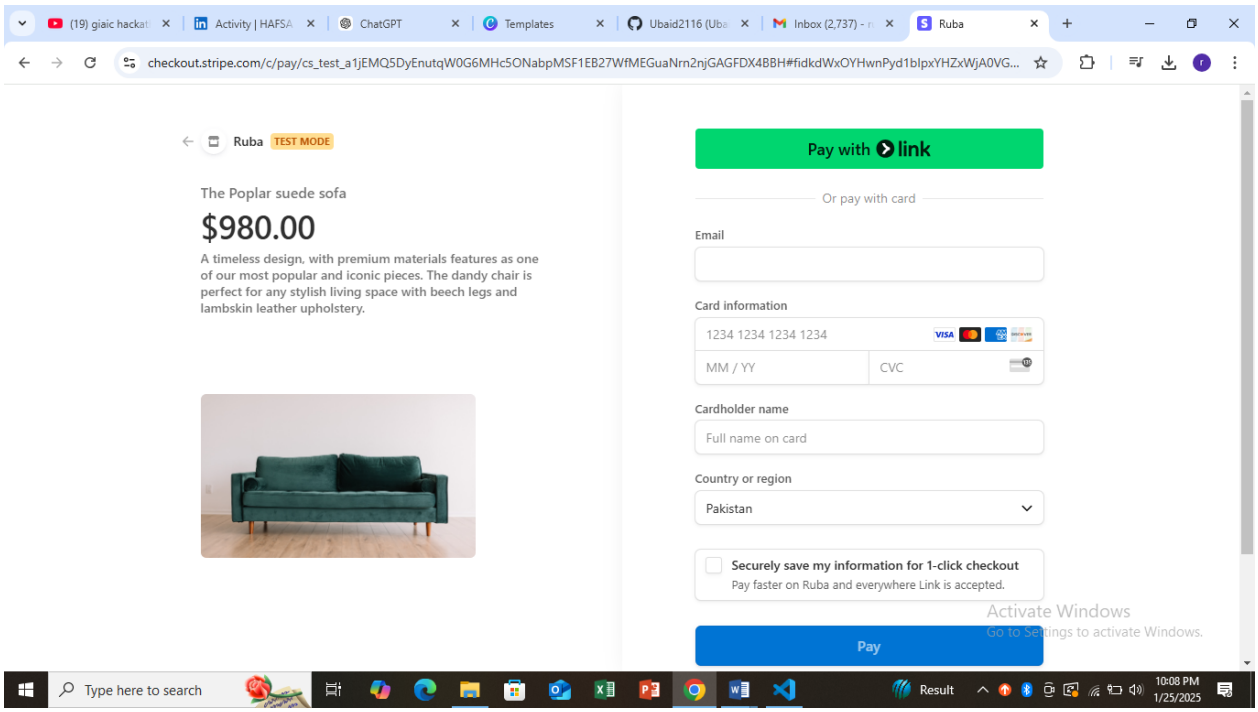


Detailed Description:

- The checkout process is divided into multiple steps: billing details, shipping address, and payment information.
- A dynamic progress tracker indicates the current step, enhancing the user experience.
- Input validation ensures that all required fields are filled correctly, reducing errors during order submission.
- Although payment integration can be mocked initially, it is designed to be extendable with payment gateways like Stripe or PayPal.
- Order summaries are displayed at the end, allowing users to confirm their details before finalizing the purchase.

5. Payment Integration:

Payment method is integrated through Stripe.

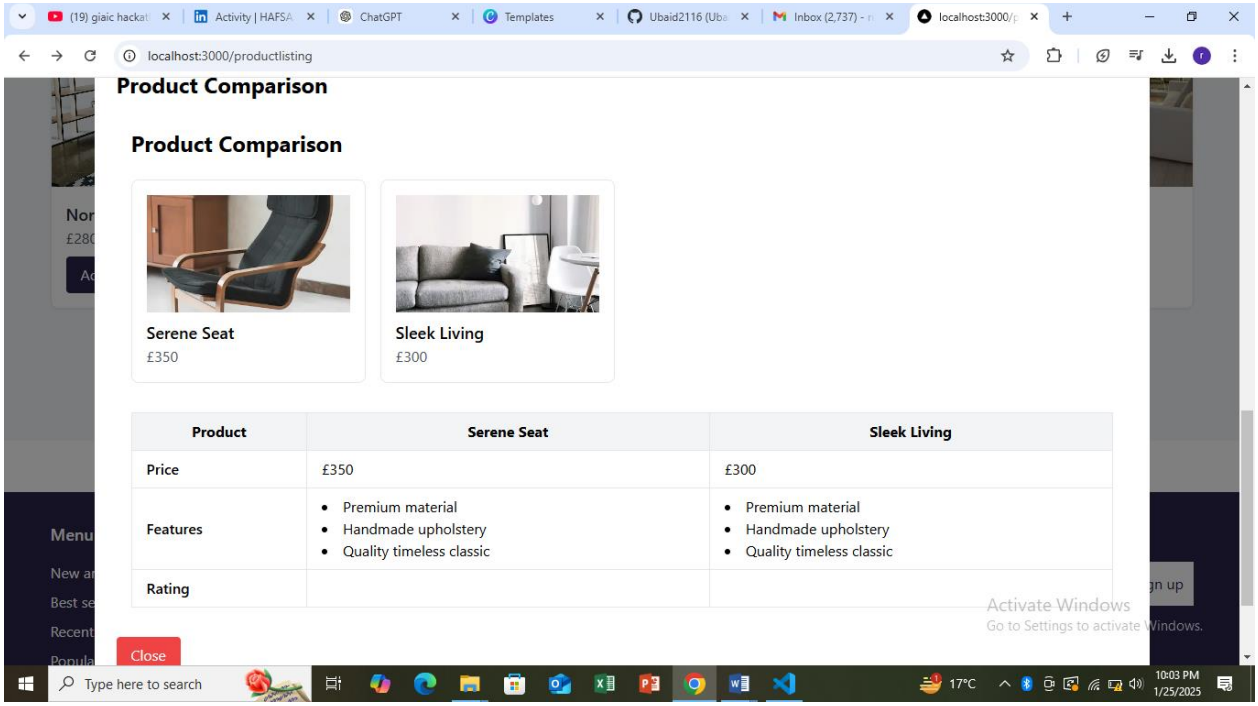


Detailed Description:

- When the user fill the checkout form and click on place order button then a payment form opens in which they have to fill their details and then their payment is processed.
- When their payment is done then a success message pops on screen
- If there is issue and cancel for any reason then they move towards again product page.

6. Product Comparison:

Product Comparison enables users to evaluate multiple products side by side, facilitating informed purchasing decisions.



Detailed Description:

- Users can add products to a comparison list from the product listing or detail page.
- Key attributes, such as price, features, ratings, and availability, are displayed in a table format.
- The interface highlights differences and similarities, simplifying the decision making process.
- This feature is especially useful for marketplaces offering diverse products with varying specifications.
- The comparison functionality is designed to handle multiple products while maintaining readability and user-friendliness.

7. Inventory Management:

Inventory Management tracks the availability of products, ensuring users are

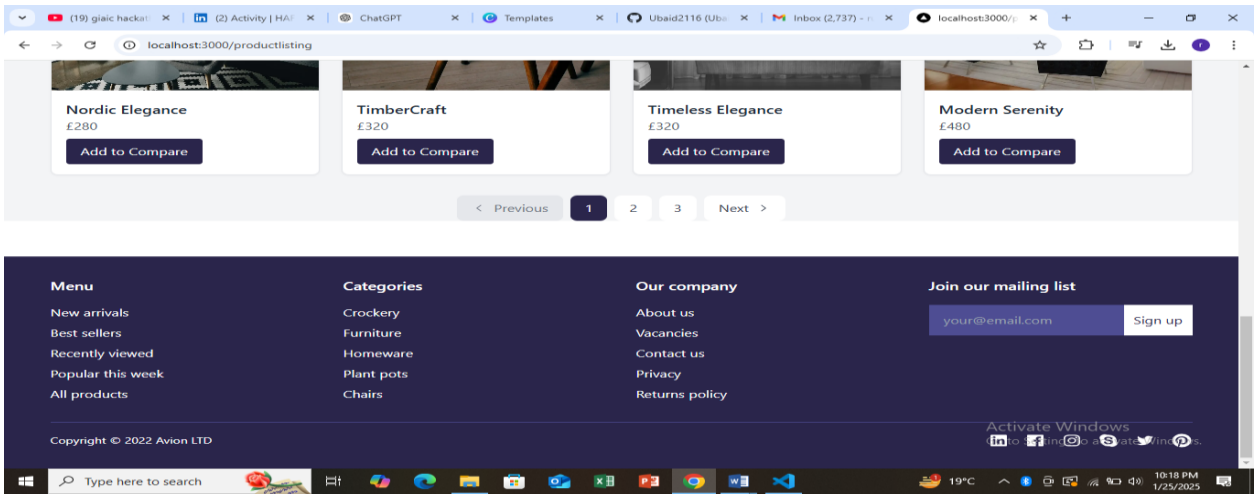
informed about stock level.

Detailed Description:

- Real-time stock tracking helps prevent overselling and notifies users when items are low in stock or unavailable.
- Inventory updates are synchronized with the backend, ensuring accurate data at all times.
- Alerts and indicators, such as "Only 3 left in stock," create a sense of urgency, encouraging purchases.
- Admin interfaces for managing stock levels provide flexibility to update inventory quickly.
- This functionality plays a critical role in maintaining customer satisfaction by avoiding issues related to out-of-stock products.

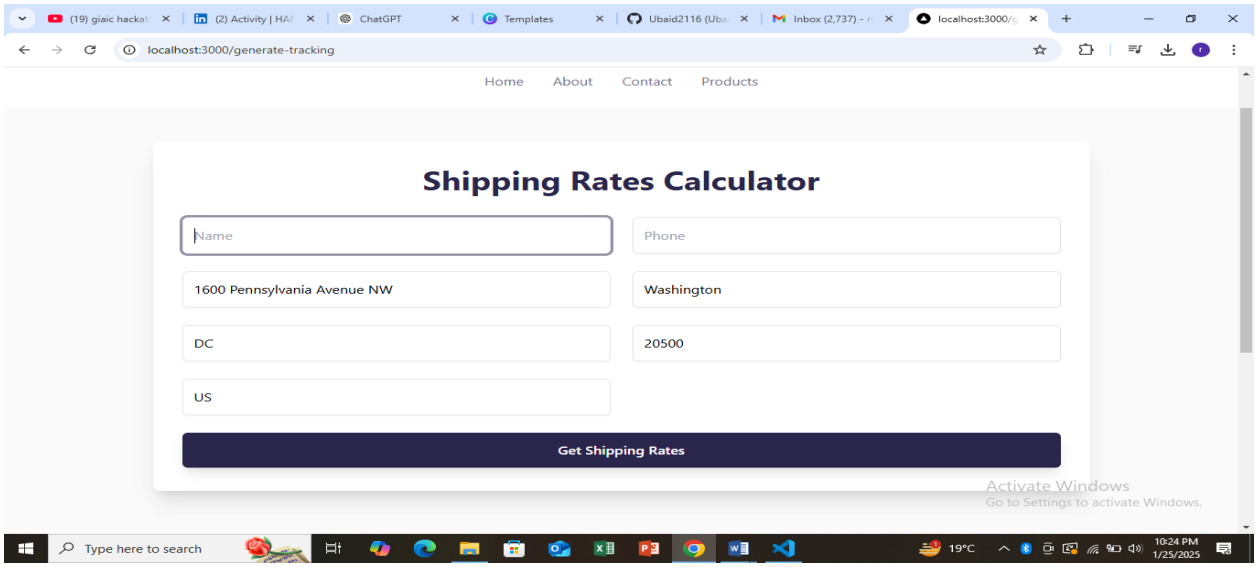
8. Pagination & Search filter with category:

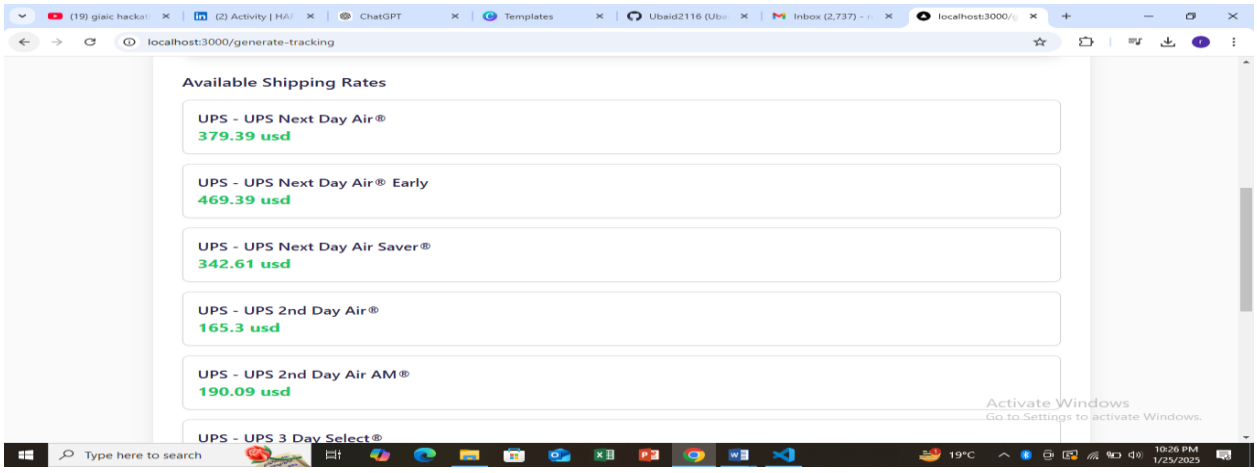
In product listing page there is a pagination effect in which user can go to previous page and next page if they want. And a search bar in which they can select any product with price filter according to their high and low rate. A category dialog in which when you click on any category then their details came.



9. Order Tracking:

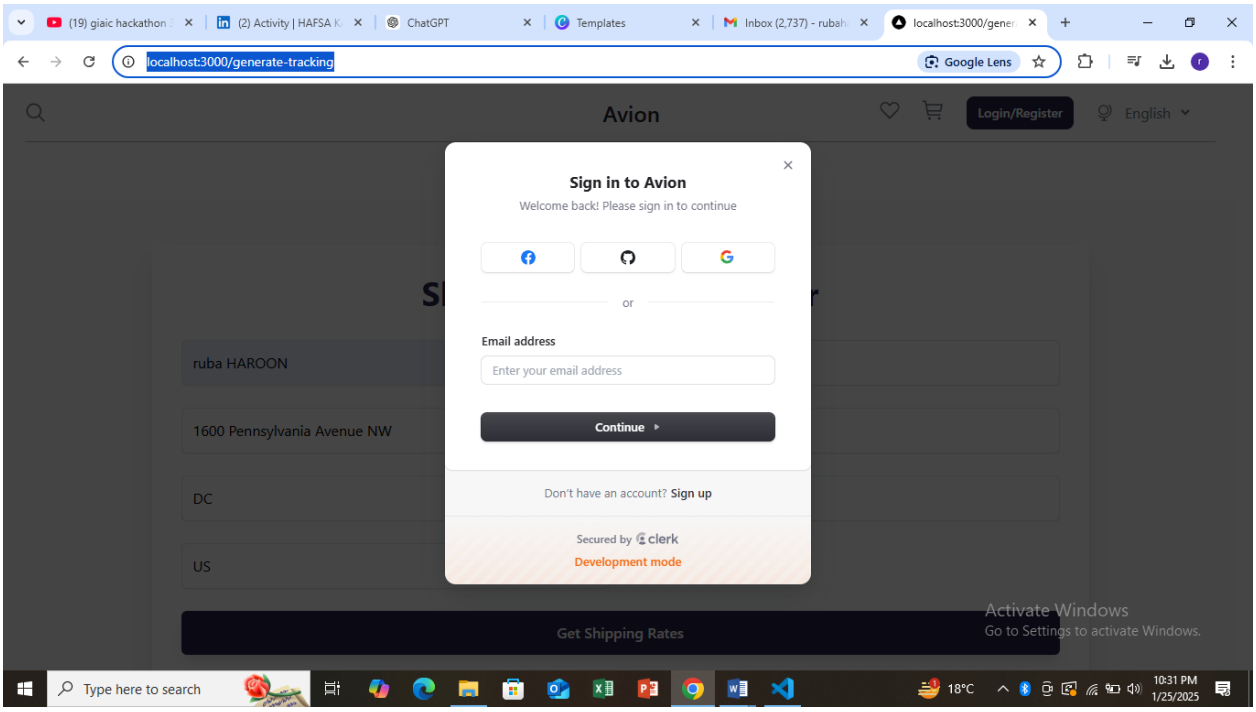
Order tracking through Ship Engine. After payment processing a shipping form came in which when user click on get shipping rates button then rates came user can select any one option then when click on create label button then labels create and through that number user can track their order.





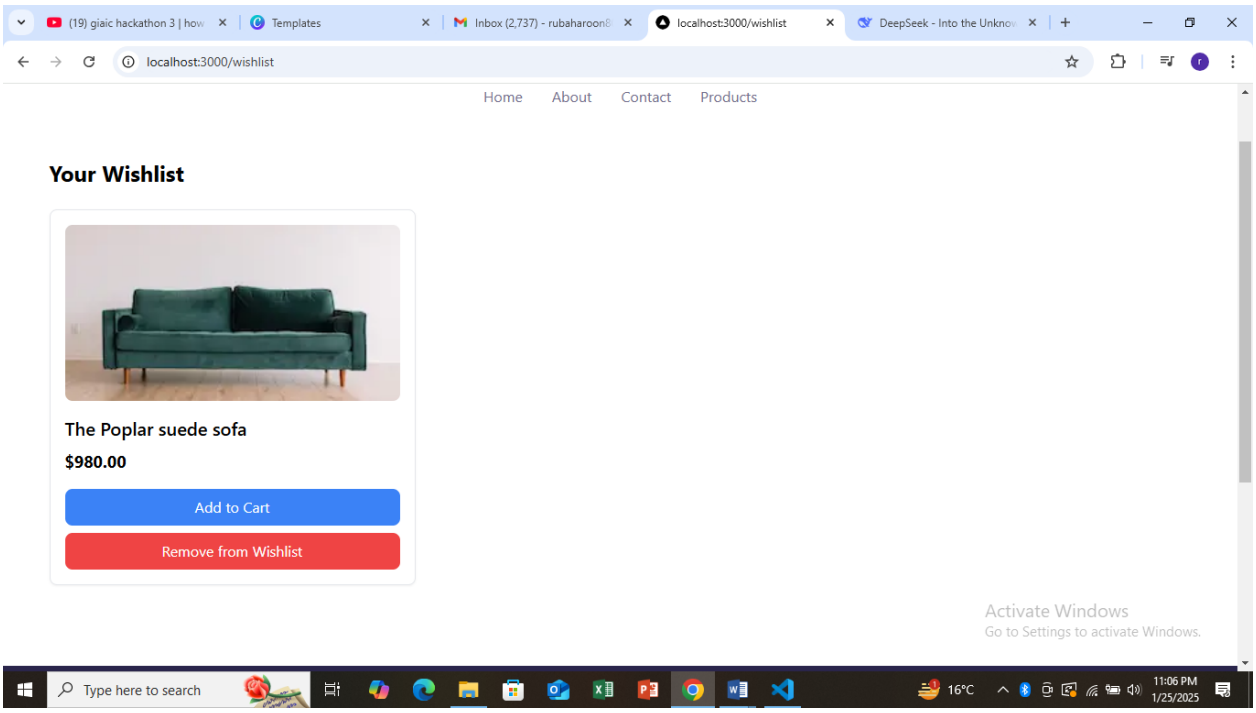
10. Authentication:

Authentication through clerk. When user click on log in/sign up button then a form appears in which they sign up and then sign in website and then they can easily update profile or can remove account.



11. Wish list Functionality:

The Wish list Functionality manages the user's selected items and if they click on heart button then the product get saved in your wish list.

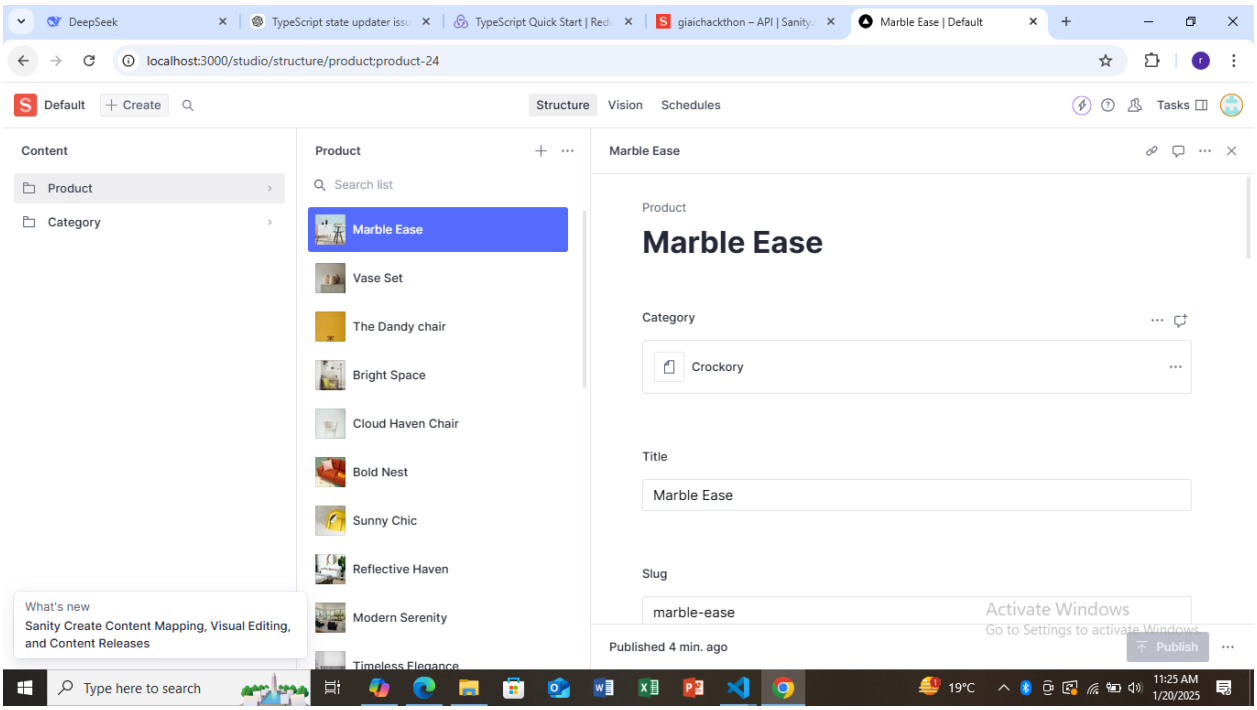


Detailed Description:

- Users can click on heart button directly from the product listing or detail page.
- In wish list page the product get save in favorites.
- A heart icon displays a quick summary of selected items, while a detailed wish list page offers options to edit or remove items.
- State management tools, such as React Context or Redux, are used to maintain the wish list state across the application.

Step 3: Integration with Sanity CMS:

Sanity CMS serves as the backend for managing and retrieving product data dynamically.



Detailed Description:

- Products, categories, and other metadata are stored in Sanity CMS, allowing admins to update content without touching the code base.
- A robust client is used to query Sanity CMS, fetching data dynamically and efficiently.
- Changes made in the CMS are instantly reflected on the frontend, providing a seamless content management experience.
- The integration is designed to be extendable, allowing the addition of new data types or fields as the marketplace grows.

Conclusion:

This documentation outlines a comprehensive approach to building dynamic and responsive marketplace components. By leveraging Sanity CMS for backend management and modular frontend development techniques, the application

achieves scalability, efficiency, and a superior user experience.

Each functionality—from product listing to inventory management—plays a vital role in delivering a professional marketplace that meets real-world needs. Future enhancements, such as integrating advanced analytics or AI-based recommendations, can further elevate the platform.