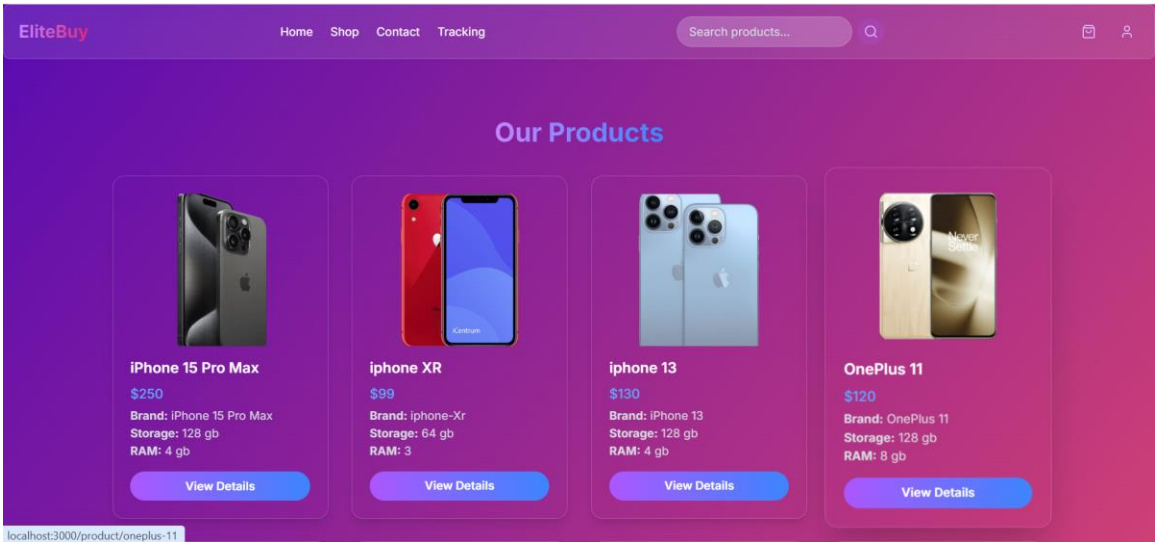
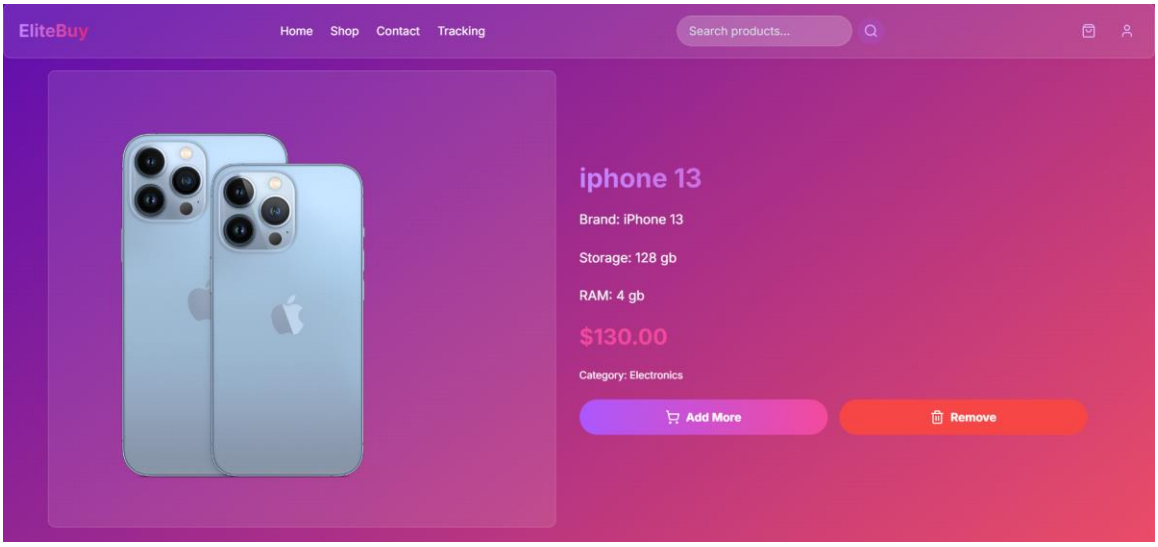
Day 4 - Dynamic Frontend Components - [EliteBuy-Ecommerce]

1. Functional Deliverables:

▪ The product listing page with dynamic data.

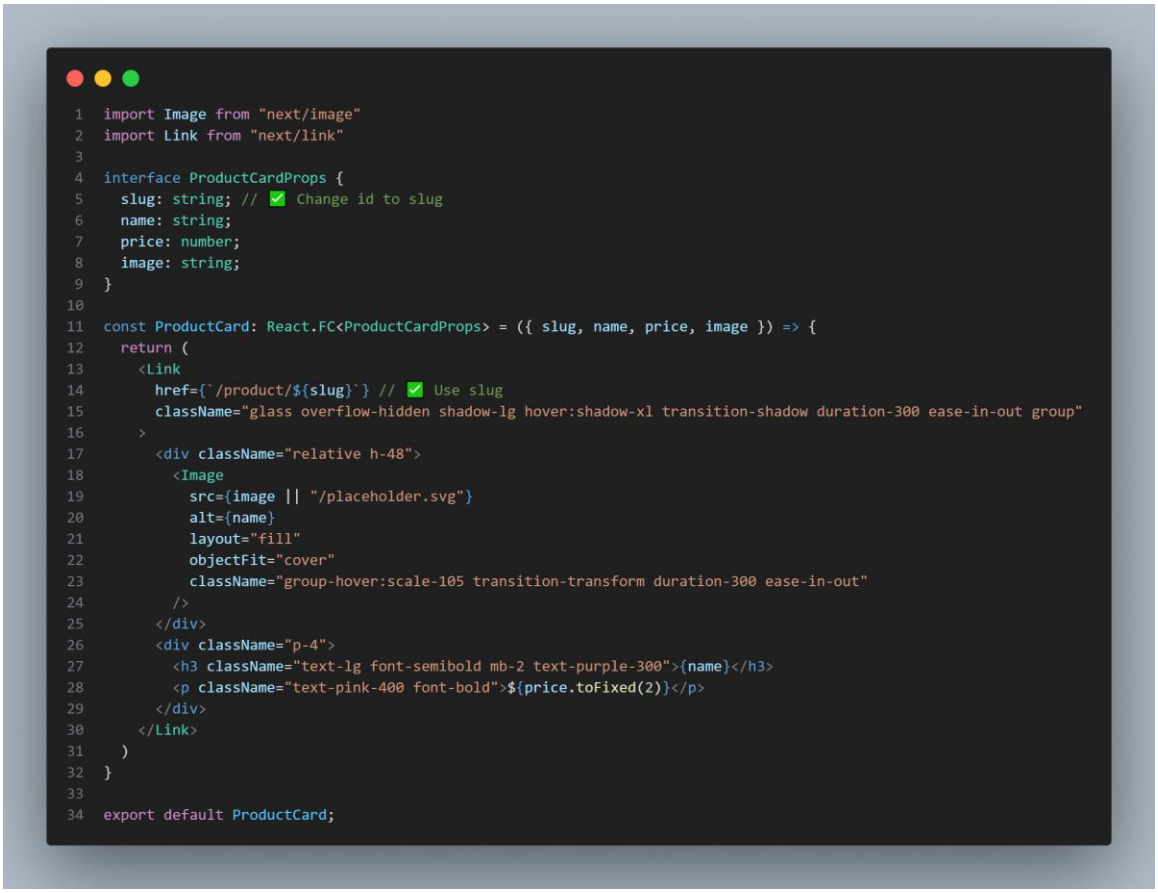


Product Details.



2. Code Deliverables:

- ProductCard,



```tsx
import Image from "next/image"
import Link from "next/link"

interface ProductCardProps {
  slug: string; // ✅ Change id to slug
  name: string;
  price: number;
  image: string;
}

const ProductCard: React.FC<ProductCardProps> = ({ slug, name, price, image }) => {
  return (
    <Link
      href={`/product/${slug}`} // ✅ Use slug
      className="glass overflow-hidden shadow-lg hover:shadow-xl transition-shadow duration-300 ease-in-out group"
    >
      <div className="relative h-48">
        <Image
          src={image || "/placeholder.svg"}
          alt={name}
          layout="fill"
          objectFit="cover"
          className="group-hover:scale-105 transition-transform duration-300 ease-in-out"
        />
      </div>
      <div className="p-4">
        <h3 className="text-lg font-semibold mb-2 text-purple-300">{name}</h3>
        <p className="text-pink-400 font-bold">${price.toFixed(2)}</p>
      </div>
    </Link>
  )
}

export default ProductCard;
```

SearchBar,

```tsx
"use client";

import * as React from "react";
import { Check, ChevronsUpDown } from "lucide-react";
import { cn } from "@/lib/utils";
import { Button } from "@/components/ui/button";
import {
  Command,
  CommandEmpty,
  CommandGroup,
  CommandInput,
  CommandItem,
  CommandList,
} from "@/components/ui/command";
import {
  Popover,
  PopoverContent,
  PopoverTrigger,
} from "@/components/ui/popover";

// Example dataset (Replace with real data if needed)
const products = [
  { value: "iphone 13", label: "iPhone 13" },
  { value: "iphone xr", label: "iPhone XR" },
  { value: "oneplus 11", label: "OnePlus 11" },
  { value: "iphone 15 pro max", label: "iPhone 15 Pro Max" },
  { value: "google pixel 8", label: "Google Pixel 8" },
  { value: "xiaomi redmi note 12", label: "Xiaomi Redmi Note 12" },
  { value: "motorola edge+", label: "Motorola Edge+" },
];

const SearchBar = () => {
  const [open, setOpen] = React.useState(false);
  const [selected, setSelected] = React.useState("");
  const [query, setQuery] = React.useState("");

  // Filter products based on user input
  const filteredProducts = products.filter((product) =>
    product.label.toLowerCase().includes(query.toLowerCase())
  );

  return (
    <Popover open={open} onOpenChange={setOpen}>
      <PopoverTrigger asChild>
        <Button
          variant="outline"
          role="combobox"
          aria-expanded={open}
          className="w-[250px] justify-between"
        >
          {selected
            ? products.find((p) => p.value === selected)?.label
            : "Search product..."}
          <ChevronsUpDown className="opacity-50" />
        </Button>
      </PopoverTrigger>
      <PopoverContent className="w-[250px] p-0">
        <Command>
          <CommandInput
            placeholder="Search product..."
            className="h-9"
            onValueChange={(val) => setQuery(val)}
          />
```

Technical Report

Steps Taken to Build and Integrate Components

The website was developed using Next.js for a seamless and efficient frontend experience, while Sanity was integrated as a headless CMS to manage dynamic content effortlessly. To enable secure and efficient transactions, Strapi was utilized as the payment gateway. The development process involved setting up a structured folder architecture, implementing API routes for fetching and displaying data, and ensuring smooth communication between frontend and backend services.

Challenges Faced and Solutions Implemented

Data Fetching Complexity – Managing real-time content updates from Sanity required optimizing API calls. Implementing ISR (Incremental Static Regeneration) in Next.js resolved performance bottlenecks.

Payment Integration Issues – Strapi's payment setup required configuring authentication and middleware. This was tackled by ensuring proper API permissions and handling secure transactions via Stripe integration.

CMS Content Structure – Structuring Sanity schemas effectively to support flexible content updates posed an initial challenge. Implementing reusable schemas and modular components streamlined content management.

Best Practices Followed During Development

Code Optimization – Leveraged Next.js features like SSR, SSG, and ISR for improved performance.

Security Enhancements – Implemented proper authentication measures in Strapi to secure user transactions.

Scalability & Maintainability – Used modular components and structured API requests to ensure easy future upgrades.

SEO Best Practices – Optimized metadata, lazy-loaded images, and implemented structured data for better search engine visibility.