Full Stack React Ecommerce with NextJs NextAuth NextAPI

Build a massive ecommerce applications using Next.js, NextAuth, NextAPI and deploy to Vercel. Stay in touch with me Ryan Dhungel on twitter.com/@kaloraat for more..

Full Authentication & Authorization using NextAuth

Create nextjs project

```
mkdir nextcom
cd nextcom
npx create-next-app@latest
// use . to create project inside nextcom
// run project using
npm run dev
```

Using bootstrap material css

```
// remove all css from globals.css and page.module.css

// app/layout.js
import "./globals.css";

export const metadata = {
  title: "Create Next App",
  description: "Generated by create next app",
};

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}


// app/page.js
export default function Home() {
  return (
    <main>
      <div>
```

```
        <h1>Home</h1>
      </div>
    </main>
  );
}

// install bootstrap-material
// npm i bootstrap-material-design

// import in layout
import "bootstrap-material-design/dist/css/bootstrap-material-
design.min.css";

// try some bootstrap-material class names
// app/page.js
export default function Home() {
  return (
    <main>
      <div>
        <h1 className="d-flex justify-content-center align-items-center
vh-100 text-uppercase">
          Home
        </h1>
      </div>
    </main>
  );
}
```

Create navigation

```
// components/nav/TopNav.js
import Link from "next/link";

export default function TopNav() {
  return (
    <nav className="nav shadow p-2 justify-content-between mb-3">
      <Link className="nav-link" href="/">
        🛒 NEXTCOM
      </Link>

      <div className="d-flex">
        <Link className="nav-link" href="/login">
          Login
        </Link>
        <Link className="nav-link" href="/register">
          Register
        </Link>
      </div>
    </nav>
  );
}
```

```
// import in layout
import TopNav from "@/components/nav/TopNav";
// ...

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <TopNav />
        {children}
      </body>
    </html>
  );
}
```

Nextjs API routes

```
// app/api/route.js
import { NextResponse } from "next/server";

export async function GET(req) {
  return NextResponse.json({ time: new Date().toLocaleString() });
}

// try visiting
// http://localhost:3000/api
```

Signup to mongodb

Signup to mongo atlas to get a connection string A tutorial link

Using ENV variables

Use custom `config` file along with `next.config.js` to use `env` variables so that it works perfectly once deployed to **vercel**

```
// config.js
const DB_URI =
  process.env.NODE_ENV === "production"
    ? "mongodb+srv://ryan:xxx@nextecom.xxx.mongodb.net/?
retryWrites=true&w=majority"
    : "mongodb://localhost:27017/nextecom";

module.exports = {
  DB_URI,
};
```

```
// next.config.js
const config = require("./config");

/** @type {import('next').NextConfig} */
const nextConfig = {
  env: {
    DB_URI: config.DB_URI,
  },
};

module.exports = nextConfig;
```

Connect to mongondb

```
// npm i mongoose mongoose-unique-validator

// utils/dbConnect.js
import mongoose from "mongoose";

const dbConnect = async () => {
  if (mongoose.connection.readyState >= 1) {
    return;
  }
  mongoose.connect(process.env.DB_URI);
};

export default dbConnect;
```

Create user model

```
// models/user
import mongoose from "mongoose";
import uniqueValidator from "mongoose-unique-validator";

const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Please enter your name"],
      trim: true,
      minLength: 1,
      maxLength: 20,
    },
    email: {
      type: String,
      required: true,
      index: true,
      lowercase: true,
      unique: true,
      trim: true,
      minLength: 5,
      maxLength: 20,
    },
    password: String,
    role: {
      type: String,
      default: "user",
    },
    image: String,
    resetCode: {
      data: String,
      expiresAt: {
        type: Date,
        default: () => new Date(Date.now() + 10 * 60 * 1000), // 10 
minutes in milliseconds
      },
    },
  },
  { timestamps: true }
);

userSchema.plugin(uniqueValidator);

export default mongoose.models.User || mongoose.model("User", userSchema);
```

Register API route

```
// npm i bcrypt

// app/api/register/route.js
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import User from "@/models/user";
import bcrypt from "bcrypt";

export async function POST(req) {
  const body = await req.json();
  // console.log("body in register api => ", body);

  await dbConnect();

  try {
    const { name, email, password } = body;

    await new User({
      name,
      email,
      password: await bcrypt.hash(password, 10),
    }).save();
```

```
      return NextResponse.json({ success: "Registered Successfully" });
    } catch (err) {
      console.log(err);
      return NextResponse.json({ err: err.message }, { status: 500 });
    }
  }

  // try using postman
  // send name, email, password as json format in re.body
```

Register form

```
// app/register/page.js
"use client";
import { useState } from "react";

export default function Register() {
  const [name, setName] = useState("Ryan");
  const [email, setEmail] = useState("ryan@gmail.com");
  const [password, setPassword] = useState("rrrrrr");
  const [loading, setLoading] = useState(false);

  const handleSubmit = async (e) => {
    try {
      e.preventDefault();
      setLoading(true);
      console.table({ name, email, password });
    } catch (err) {
      console.log(err);
      setLoading(false);
    }
  };

  return (
    <main>
      <div className="container">
        <div className="row d-flex justify-content-center align-items-
center vh-100">
          <div className="col-lg-5 bg-light p-5 shadow">
            <h2 className="mb-4">Register</h2>

            <form onSubmit={handleSubmit}>
              <input
                type="text"
                value={name}
                onChange={(e) => setName(e.target.value)}
                className="form-control mb-4"
                placeholder="Your name"
              />
              <input
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                className="form-control mb-4"
                placeholder="Your email"
              />
              <input
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                className="form-control mb-4"
                placeholder="Your password"
              />
              <button
                className="btn btn-primary btn-raised"
                disabled={loading || !name || !email || !password}
              >
                {loading ? "Please wait.." : "Submit"}
              </button>
            </form>
          </div>
        </div>
      </div>
    </main>
  );
}
```

Regisger API request

```
// config.js
const API =
  process.env.NODE_ENV === "production"
    ? "https://xxx.vercel.com/api"
    : "http://localhost:3000/api";

module.exports = {
  // ...
  API,
};

// next.config.js
const nextConfig = {
  env: {
    DB_URI: config.DB_URI,
    API: config.API,
  },
};

// npm i react-hot-toast

// layout.tsx
import { Toaster } from "react-hot-toast";
```

```
// ...
<body>
  <TopNav />
  <Toaster />
  {children}
</body>;
// ...

// app/register/page
import toast from "react-hot-toast";
import { useRouter } from "next/navigation";

// ...
const router = useRouter();

const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
  try {
    e.preventDefault();
    setLoading(true);

    const response = await fetch(`${process.env.API}/register`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        name,
        email,
        password,
      }),
    });

    const data = await response.json();

    if (!response.ok) {
      toast.error(data.err);
      setLoading(false);
    } else {
      toast.success(data.success);
      router.push("/login");
    }
  } catch (err) {
    console.log(err);
    setLoading(false);
  }
};
```

Login page

```
// app/login/page
"use client";
```

```
import { set } from "mongoose";
import { useState } from "react";
import { FormEvent } from "react";
import toast from "react-hot-toast";
import { useRouter } from "next/navigation";

export default function Register() {
  const [email, setEmail] = useState("ryan@gmail.com");
  const [password, setPassword] = useState("rrrrrr");
  const [loading, setLoading] = useState(false);

  const router = useRouter();

  const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    //
  };

  return (
    <main>
      <div className="container">
        <div className="row d-flex justify-content-center align-items-
center vh-100">
          <div className="col-lg-5 bg-light p-5 shadow">
            <h2 className="mb-4">Login</h2>

            <form onSubmit={handleSubmit}>
              <input
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                className="form-control mb-4"
                placeholder="Your email"
              />
              <input
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                className="form-control mb-4"
                placeholder="Your password"
              />
              <button
                className="btn btn-primary btn-raised"
                disabled={loading || !email || !password}
              >
                {loading ? "Please wait.." : "Submit"}
              </button>
            </form>
          </div>
        </div>
      </div>
    </main>
  );
}
```

Email and password login with next auth

```
// npm i net-auth

// app/login/page
import { signIn } from "next-auth/react";

// ...
const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
  e.preventDefault();
  setLoading(true);

  const result = await signIn("credentials", {
    redirect: false,
    email,
    password,
  });

  setLoading(false);

  if (result?.error) {
    toast.error(result?.error);
  } else {
    toast.success("Login success");
    router.push("/");
  }
};
// without next-auth config, you get redirected to '/api/auth/error'
```

NextAuth configuration

```
// config.js
// name "NEXTAUTH_SECRET" is important, dont rename it
const NEXTAUTH_SECRET = "YOUR_SECRET";

// utils/authOptions.js
import CredentialsProvider from "next-auth/providers/credentials";
import User from "@/models/user";
import bcrypt from "bcrypt";
import dbConnect from "@/utils/dbConnect";

export const authOptions = {
  session: {
    strategy: "jwt",
  },
  providers: [
    CredentialsProvider({
      async authorize(credentials, req) {
        dbConnect();

        const { email, password } = credentials;

        const user = await User.findOne({ email });

        if (!user) {
          throw new Error("Invalid email or password");
        }

        // If the user has no password (i.e., they signed up via a social
network), throw an error
        if (!user?.password) {
          throw new Error("Please login via the method you used to sign
up");
        }

        const isPasswordMatched = await bcrypt.compare(
          password,
          user?.password
        );

        if (!isPasswordMatched) {
          throw new Error("Invalid email or password");
        }

        return user;
      },
    }),
  ],
  secret: process.env.NEXT_AUTH_SECRET,
  pages: {
    signIn: "/login",
  },
};

// use authOptions in [...nextauth]/route
// app/api/auth/[...nextauth]/route
import NextAuth from "next-auth";

import { authOptions } from "@/utils/authOptions";

const handler = NextAuth(authOptions);

export { handler as GET, handler as POST };
```

Provide user session from next auth

```
// SessionProvider in Layout
"use client";
import "./globals.css";
import "bootstrap-material-design/dist/css/bootstrap-material-
design.min.css";
import TopNav from "@/components/nav/TopNav";
import { Toaster } from "react-hot-toast";
import { SessionProvider } from "next-auth/react";

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <SessionProvider>
        <body>
          <TopNav />
          <Toaster />
          {/* children props/components can be server rendered */}
          {children}
        </body>
      </SessionProvider>
    </html>
  );
}
```

Access logged in user info

```
// components/nav/TopNav
import Link from "next/link";
import { useSession, signOut } from "next-auth/react";

export default function TopNav() {
  const { data, status, loading } = useSession();

  // console.log(data, status);

  return (
    <nav className="nav shadow p-2 justify-content-between mb-3">
      <Link className="nav-link" href="/">
        🛒 NEXTCOM
      </Link>

      {status === "authenticated" ? (
        <div className="d-flex">
          <Link className="nav-link" href="/dashboard/user">
            {data?.user?.name}
          </Link>
          <a
            className="nav-link pointer"
            onClick={() => signOut({ callbackUrl: "/login" })}
          >
            Logout
          </a>
        </div>
      ) : (
        <div className="d-flex">
          <Link className="nav-link" href="/login">
            Login
          </Link>
          <Link className="nav-link" href="/register">
            Register
          </Link>
        </div>
      )}
    </nav>
  );
}
```

```
// globals.css
.pointer {
  cursor: pointer;
}
```

Loading page

This is default loading page in nextjs

```
// app/loading.js
export default function Loading() {
  return (
    <div className="d-flex justify-content-center align-items-center vh-
100 text-danger">
      LOADING
    </div>
  );
}

// using session 'loading' status
// TopNav
return (
  <nav className="nav shadow p-2 justify-content-between mb-3">
    <Link className="nav-link" href="/">
      🛒 NEXTCOM
    </Link>

    {status === "authenticated" ? (
      <div className="d-flex">{/*  */}</div>
    ) : status === "loading" ? (
      <div className="d-flex">
        <a className="nav-link text-danger">Loading</a>
      </div>
    ) : (
      <div className="d-flex">{/*  */}</div>
    )}
  </nav>
);
```

404 Page not found

```
// app/not-found.js
export default function NotFound() {
  return (
    <div className="d-flex justify-content-center align-items-center vh-
100 text-danger">
      404
    </div>
  );
}
```

User dashboard

```
// app/dashboard/user/page
export default function UserDashboard() {
  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p className="lead">Dashboard</p>
          <hr />
          ...
        </div>
      </div>
    </div>
  );
}
// this page is accessible to anyone
```

Protecting pages

Protect dashboard pages

```
// middleware.js
export { default } from "next-auth/middleware";
export const config = { matcher: ["/dashboard/:path*"] };
```

Redirect back to intended page

```
// login
import { useRouter, useSearchParams } from "next/navigation";

const router = useRouter();
const searchParams = useSearchParams();
const callbackUrl = searchParams.get("callbackUrl") || "/";

// handleSubmit()
router.push(callbackUrl);
```

Login with google

```
// config.js
GOOGLE_CLIENT_ID=xxx
GOOGLE_CLIENT_SECRET=xxx
// import in next.config.js

// utils/authOptions

// ...
import GoogleProvider from "next-auth/providers/google";

// providers: [
GoogleProvider({
    clientId: process.env.GOOGLE_CLIENT_ID,
    clientSecret: process.env.GOOGLE_CLIENT_SECRET,
  }),
```

Login page

```
<button
  className="btn btn-danger btn-raised mb-4"
  onClick={() => signIn("google", { callbackUrl: "/" })}
>
  Sign in with Google
</button>
```

Save social login user in database

Currently user who login with google, is not saved in database

```
// utils/authOptions
// after providers array
callbacks: {
    async signIn({ user }) {
      dbConnect();

      const { email } = user;

      // Try to find a user with the provided email
      let dbUser = await User.findOne({ email });

      // If the user doesn't exist, create a new one
      if (!dbUser) {
        dbUser = await User.create({
          email,
          name: user?.name,
          image: user?.image,
        });
      }

      return true;
    },
  },
```

Additional user info in session

Currently only user name and email is in the session. Let's add role and other user info. Try consonle.log(data) in TopNav

```
// get user roles

// authOptions
callbacks: {
  // ...
  // add user profile/role to token and session
  jwt: async ({ token, user }) => {
    const userByEmail = await User.findOne({ email: token.email });
    userByEmail.password = undefined;
    token.user = userByEmail;
    return token;
  },
  session: async ({ session, token }) => {
    session.user = token.user;
    return session;
  },
},
```

Show user role

```
// TopNav
<Link
  className="nav-link"
```

```
    href={`/dashboard/${data?.user?.role === "admin" ? "admin" : "user"}`}
  >
    {data.user.name} ({data?.user?.role})
  </Link>
  // manually update user role to "admin" and try
```

Admin layout and page

```
// components/nav/AdminNav

// used in admin layout
import Link from "next/link";

export default function AdminNav() {
  return (
    <>
      <nav className="nav justify-content-center mb-3">
        <Link className="nav-link" href="/dashboard/admin">
          Admin
        </Link>
        <Link className="nav-link" href="/dashboard/admin/product/create">
          Create Product
        </Link>
      </nav>
    </>
  );
}


// app/dashboard/admin/layout
import Link from "next/link";
import AdminNav from '@/components/nav/AdminNav';

export default function AdminLayout({ children }) {
  return (
    <>
      <AdminNav />
      {children}
    </>
  );
}

// app/dashboard/admin/page
export default function AdminDashboard() {
  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p>Admin Dashboard</p>
          <hr />
          ...
        </div>
      </div>
    </div>
  );
}
```

Role based page protection for admin

Currently any logged in user can access '/dashboard/admin' routes

```
// middleware.js

// export { default } from "next-auth/middleware";
// export const config = { matcher: ["/dashboard/:path*"] };

import { withAuth } from "next-auth/middleware";
import { NextResponse } from "next/server";

// client and server side protection
export const config = {
  matcher: [
    "/dashboard/user/:path*",
    "/dashboard/admin/:path*",
    "/api/user/:path*",
    "/api/admin/:path*",
  ],
};

export default withAuth(
  async function middleware(req) {
    // authorize roles
    const url = req.nextUrl.pathname;
    const userRole = req?.nextauth?.token?.user?.role;

    // client side protection
    if (url?.includes("/admin") && userRole !== "admin") {
      return NextResponse.redirect(new URL("/", req.url));
    }
  },
  {
    callbacks: {
      authorized: ({ token }) => {
        if (!token) {
          return false;
        }
      },
    },
  }
);
```

Deploy to vercel

```
npm i -g vercel@latest
vercel
// for future updates
vercel --prod
// update the env variables with production url
// https://nextecom-kaloraat.vercel.app/

const API =
  process.env.NODE_ENV === "production"
    ? "https://nextecom-kaloraat.vercel.app/api"
    : "http://localhost:3000/api";
```

This is all you need to build full authentication and authorization system in nextjs using nextauth. You can save your project code now and use it as a base project for your other future projects.

Complete Ecommerce Project

Category model

Now we start building ecommerce app. Start off with categories

```
// models/category
import mongoose from "mongoose";
import uniqueValidator from "mongoose-unique-validator";

const categorySchema = new mongoose.Schema(
  {
    name: {
      type: String,
      trim: true,
      required: true,
      minLength: 1,
      maxLength: 20,
    },
    slug: {
      type: String,
      unique: true,
      lowercase: true,
      index: true,
    },
  },
  { timestamps: true }
);

categorySchema.plugin(uniqueValidator);

export default mongoose.models.Category ||
  mongoose.model("Category", categorySchema);
```

Category create API

```
// api/admin/category/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Category from "@/models/category";
import slugify from "slugify";

export async function POST(req) {
  const body = await req.json();
  await dbConnect();

  try {
    const { name } = body;

    const category = await Category.create({
      name,
      slug: slugify(name),
    });

    return NextResponse.json(category);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Categories list API

This will be publicly accessible route

```
// api/category/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Category from "@/models/category";

export async function GET(req) {
  await dbConnect();

  try {
    const categories = await Category.find({}).sort({ createdAt: "-1" });
    return NextResponse.json(categories);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
```

Categories update API

```
// api/admin/category/[id]/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Category from "@/models/category";
import slugify from "slugify";

export async function PUT(req, context) {
  await dbConnect();
  const body = await req.json();

  try {
    const updatingCategory = await Category.findByIdAndUpdate(
      context.params.id,
      { ...body, slug: slugify(body.name) },
      { new: true }
    );

    return NextResponse.json(updatingCategory);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Categories delete API

```
// api/admin/category/[id]/route
export async function DELETE(req, context) {
  await dbConnect();

  try {
    const deletedCategory = await
Category.findByIdAndDelete(context.params.id);
```

```
    return NextResponse.json(deletedCategory);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Category context

```
// context/category
"use client";
import { createContext, useState, useContext } from "react";
import toast from "react-hot-toast";

export const CategoryContext = createContext();

export const CategoryProvider = ({ children }) => {
  const [name, setName] = useState("");
  // for fetching all categories
  const [categories, setCategories] = useState([]);
  // for update and delete
  const [updatingCategory, setUpdatingCategory] = useState(null);

  const createCategory = async () => {
    try {
      //
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while creating the category");
    }
  };

  const fetchCategories = async () => {
    try {
      //
    } catch (error) {
      console.error("Error fetching search results:", error);
    }
  };

  const updateCategory = async () => {
    try {
      //
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while updating the category");
    }
  };

  const deleteCategory = async () => {
    try {
      //
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while deleting the category");
    }
  };

  return (
    <CategoryContext.Provider
      value={{
        name,
        setName,
        createCategory,
        categories,
        setCategories,
        fetchCategories,
        updatingCategory,
        setUpdatingCategory,
        updateCategory,
        deleteCategory,
      }}
    >
      {children}
    </CategoryContext.Provider>
  );
};

export const useCategory = () => useContext(CategoryContext);
```

Category create function

```
// context/category
// ...
const createCategory = async () => {
  try {
    const response = await fetch(`${process.env.API}/admin/category`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        name,
      }),
    });
```

```
      if (response.ok) {
        toast.success("Category created successfully");
        const newlyCreatedCategory = await response.json();
        setName("");
        setCategories([newlyCreatedCategory, ...categories]);
      } else {
        const errorData = await response.json();
        toast.error(errorData.err);
      }
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while creating the category");
    }
  };
```

Category list function

```
// context/category
// ...
const fetchCategories = async () => {
  try {
    // '/category' not '/categories'
    const response = await fetch(`${process.env.API}/category`);

    if (!response.ok) {
      throw new Error("Network response was not ok");
    }

    const data = await response.json();
    setCategories(data);
  } catch (error) {
    console.error("Error fetching search results:", error);
  }
};
```

Category update function

```
// context/category
// ...
const updateCategory = async () => {
  try {
    const response = await fetch(
      `${process.env.API}/admin/category/${updatingCategory._id}`,
      {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(updatingCategory),
      }
    );

    if (!response.ok) {
      throw new Error("Network response was not ok");
    }

    const updatedCategory = await response.json();

    // Update the categories state with the updated category
    setCategories((prevCategories) =>
      prevCategories.map((c) =>
        c._id === updatedCategory._id ? updatedCategory : c
      )
    );

    // Clear the categoryUpdate state
    setUpdatingCategory(null);

    toast.success("Category updated successfully");
  } catch (err) {
    console.log("err => ", err);
    toast.error("An error occurred while updating the category");
  }
};
```

Category delete function

```
// context/category
// ...
const deleteCategory = async () => {
  try {
    const response = await fetch(
      `${process.env.API}/admin/category/${updatingCategory._id}`,
      {
        method: "DELETE",
      }
    );

    if (!response.ok) {
      throw new Error("Network response was not ok");
    }

    const deletedCategory = await response.json();

    // Category deleted successfully, now update the categories state
    setCategories((prevCategories) =>
      prevCategories.filter((c) => c._id !== deletedCategory._id)
    );

    // Clear the categoryUpdate state
```

```
      setUpdatingCategory(null);

      toast.success("Category deleted successfully");
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while deleting the category");
    }
  };
```

Category provider

```
// app/layout
// ...
import { CategoryProvider } from "@/context/category";

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <SessionProvider>
        <CategoryProvider>
          <body>
            <TopNav />
            <Toaster />
            {/* children props/components can be server rendered */}
            {children}
          </body>
        </CategoryProvider>
      </SessionProvider>
    </html>
  );
}
```

Category page for admin

```
// components/nav/AdminNav
<Link className="nav-link" href="/dashboard/admin/category">
  Categories
</Link>;

// app/dashboard/admin/category/page
("use client");
import CategoryCreate from "@/components/category/CategoryCreate";
import CategoryList from "@/components/category/CategoryList";

export default function Categories() {
  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
```

```
          <p className="lead">Create Category</p>
          <CategoryCreate />
        </div>
      </div>

      <div className="row mt-5">
        <div className="col">
          <p className="lead mb-4">List of Categories</p>
          <CategoryList />
        </div>
      </div>
    </div>
  );
}
```

ChatGPT for category and tags ideas

```
Category: Women's Fashion
Tags: Dresses, Tops, Bottoms, Outerwear, Activewear, Swimwear, Lingerie,
Accessories, Shoes, Handbags.

Category: Men's Fashion
Tags: Shirts, T-Shirts, Pants, Jeans, Suits, Blazers, Activewear,
Underwear, Accessories, Shoes.

Category: Kids' Fashion
Tags: Boys' Clothing, Girls' Clothing, Baby Clothing, Toddler Clothing,
Kids' Shoes, Kids' Accessories.

Category: Activewear
Tags: Yoga Wear, Running Gear, Gym Clothing, Sportswear, Workout
Accessories.

Category: Formal Wear
Tags: Evening Dresses, Suits, Tuxedos, Cocktail Dresses, Formal Shoes,
Formal Accessories.

Category: Casual Wear
Tags: Casual Dresses, Casual Tops, Jeans, T-Shirts, Casual Shoes, Hats.

Category: Shoes
Tags: High Heels, Boots, Sneakers, Flats, Sandals, Loafers, Running Shoes.

Category: Accessories
Tags: Jewelry, Watches, Scarves, Belts, Sunglasses, Hats, Handbags.
```

Category create, update and delete component

When a category is clicked, it will be put in the state as updatingCategory. Then you can update or delete that category using same form that was used to create.

```
// components/category/CategoryCreate
"use client";
import { useCategory } from "@/context/category";

export default function AdminCreateCategory() {
  // context
  const {
    name,
    setName,
    updatingCategory,
    setUpdatingCategory,
    createCategory,
    updateCategory,
    deleteCategory,
  } = useCategory();

  return (
    <>
      <p>Create Category</p>
      <input
        type="text"
        value={updatingCategory ? updatingCategory.name : name}
        onChange={(e) =>
          updatingCategory
            ? setUpdatingCategory({ ...updatingCategory, name:
e.target.value })
            : setName(e.target.value)
        }
        className="form-control p-2 my-2"
      />

      {/* <pre>{JSON.stringify(categoryUpdate, null, 4)}</pre> */}

      <div className="d-flex justify-content-between">
        <button
          className={`btn bg-${
            updatingCategory ? "info" : "primary"
          } text-light`}
          onClick={(e) => {
            e.preventDefault();
            updatingCategory ? updateCategory() : createCategory();
          }}
        >
          {updatingCategory ? "Update" : "Create"}
        </button>

        {updatingCategory && (
          <>
            <button
              className={`btn bg-danger text-light`}
              onClick={(e) => {
                e.preventDefault();
                deleteCategory();
              }}
            >
              Delete
            </button>

            <button
              className="btn bg-success text-light"
              onClick={() => setUpdatingCategory(null)}
            >
              Clear
            </button>
          </>
        )}
      </div>
    </>
  );
}
// see created categories list
// http://localhost:3000/api/category
```

Category list component

```
// components/category/CategoryList
"use client";
import { useState, useEffect } from "react";
import toast from "react-hot-toast";
import { useCategory } from "@/context/category";

export default function Categories() {
  // context
  const { categories, fetchCategories, setUpdatingCategory } =
useCategory();

  useEffect(() => {
    fetchCategories();
  }, []);

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          {categories.map((c) => (
            <button
              className="btn"
              onClick={() => {
                setUpdatingCategory(c);
              }}
            >
              {c.name}
            </button>
          ))}
        </div>
      </div>
    </div>
  );
}
```

Tags

Follow similar steps as Category

Tag Model

```
// models/tag
const mongoose = require("mongoose");
import uniqueValidator from "mongoose-unique-validator";

const tagSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      trim: true,
      required: true,
      min: 2,
      max: 32,
    },
    slug: {
      type: String,
      unique: true,
      lowercase: true,
      index: true,
    },
    parent: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Category",
      required: true,
    },
  },
  { timestamps: true }
);

tagSchema.plugin(uniqueValidator);

export default mongoose.models.Tag || mongoose.model("Tag", tagSchema);
```

Tag create API

```
// api/admin/tag/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Tag from "@/models/tag";
import slugify from "slugify";

export async function POST(req) {
  const _req = await req.json();
  await dbConnect();

  try {
    const { name, parent } = _req;

    const tag = await Tag.create({
      name,
      parent,
      slug: slugify(name),
    });

    return NextResponse.json(tag);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Tag list API

```
// api/tag/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Tag from "@/models/tag";

export async function GET(req) {
  await dbConnect();

  try {
    const tags = await Tag.find({}).sort({ createdAt: "-1" });

    return NextResponse.json(tags);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Tag update API

```
// api/admin/tag/[id]/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Tag from "@/models/tag";

export async function PUT(req, context) {
  await dbConnect();

  const _req = await req.json();

  try {
    const updatingTag = await Tag.findByIdAndUpdate(
      context.params.id,
      { ..._req },
      { new: true }
    );

    return NextResponse.json(updatingTag);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Tag delete API

```
// api/admin/tag/[id]/route
export async function DELETE(req, context) {
  await dbConnect();

  try {
    const deletedTag = await Tag.findByIdAndDelete(context.params.id);

    return NextResponse.json(deletedTag);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Tag context

```
"use client";
import { createContext, useState, useContext } from "react";
import toast from "react-hot-toast";

export const TagContext = createContext();

export const TagProvider = ({ children }) => {
  const [name, setName] = useState("");
  const [parentCategory, setParentCategory] = useState("");
  const [tags, setTags] = useState([]);
  const [updatingTag, setUpdatingTag] = useState(null);

  const createTag = async () => {
    try {
      //
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while creating a tag");
    }
  };

  const fetchTags = async () => {
    try {
      //
    } catch (error) {
      console.error("Error fetching search results:", error);
    }
  };

  const updateTag = async () => {
    try {
      //
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while updating a tag");
    }
  };

  const deleteTag = async () => {
    try {
      //
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while deleting the sub category");
    }
  };

  return (
    <TagContext.Provider
      value={{
        name,
        setName,
        parentCategory,
        setParentCategory,
        createTag,
        tags,
        setTags,
        fetchTags,
        updatingTag,
        setUpdatingTag,
        updateTag,
        deleteTag,
      }}
    >
      {children}
    </TagContext.Provider>
  );
};

export const useTag = () => useContext(TagContext);
```

Tag create function

```
// context/tag
// ...
const createTag = async () => {
  try {
    const response = await fetch(`${process.env.API}/admin/tag`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        name,
        parent: parentCategory,
      }),
    });

    if (response.ok) {
      toast.success("Tag created successfully");
      const newlyCreatedTag = await response.json();
```

```
        setName("");
        setParentCategory("");
        setTags([newlyCreatedTag, ...tags]);
      } else {
        const errorData = await response.json();
        toast.error(errorData.err);
      }
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while creating a tag");
    }
  };
```

Tag list function

```
// context/tag
// ...
const fetchTags = async () => {
  try {
    const response = await fetch(`${process.env.API}/tags`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    });

    if (!response.ok) {
      throw new Error("Network response was not ok");
    }

    const data = await response.json();
    setTags(data);
  } catch (error) {
    console.error("Error fetching search results:", error);
  }
};
```

Tag update function

```
// context/tag
// ...
const updateTag = async () => {
  try {
    const response = await fetch(
      `${process.env.API}/admin/tag/${updatingTag._id}`,
      {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(updatingTag),
      }
    );

    if (!response.ok) {
      throw new Error("Network response was not ok");
    }

    const updatedTag = await response.json();

    // Update the categories state with the updated category
    setTags((prevTags) =>
      prevTags.map((t) => (t._id === updatedTag._id ? updatedTag : t))
    );

    // Clear the categoryUpdate state
    setUpdatingTag(null);
    setParentCategory("");

    toast.success("Tag updated successfully");
  } catch (err) {
    console.log("err => ", err);
    toast.error("An error occurred while updating a tag");
  }
};
```

Tag delete function

```
// context/tag
// ...
const deleteTag = async () => {
  try {
    const response = await fetch(
      `${process.env.API}/admin/tag/${updatingTag._id}`,
      {
        method: "DELETE",
      }
    );

    if (!response.ok) {
      throw new Error("Network response was not ok");
    }

    const deletedTag = await response.json();

    // Category deleted successfully, now update the categories state
    setTags((prevTags) => prevTags.filter((t) => t._id !==
deletedTag._id));

    // Clear the categoryUpdate state
```

```
      setUpdatingTag(null);
      setParentCategory("");

      toast.success("Tag deleted successfully");
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while deleting the sub category");
    }
  };
```

Tag provider

```
// app/layout
// ...
import { TagProvider } from "@/context/tag";

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <SessionProvider>
        <CategoryProvider>
          <TagProvider>
            <body>
              <TopNav />
              <Toaster />
              {/* children props/components can be server rendered */}
              {children}
            </body>
          </TagProvider>
        </CategoryProvider>
      </SessionProvider>
    </html>
  );
}
```

Tags page for admin

```
// components/nav/AdminNav
<Link className="nav-link" href="/dashboard/admin/tag">
  Tags
</Link>
```

```
// app/dashboard/admin/tag/page
"use client";
import { useEffect } from "react";
import { useTag } from "@/context/tag";
import TagCreate from "@/components/tag/TagCreate";
import TagList from "@/components/tag/TagList";

export default function Tags() {
  // context
  const { fetchTags } = useTag();

  useEffect(() => {
    fetchTags();
  }, []);

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          <p className="lead">Create Tags</p>
          <TagCreate />
        </div>
      </div>

      <div className="row mt-5">
        <div className="col">
          <p className="lead mb-4">List of Tags</p>
          <TagList />
        </div>
      </div>
    </div>
  );
}
```

Tag create update and delete component

```
// components/tag/TagCreate
"use client";
import { useTag } from "@/context/tag";
import { useCategory } from "@/context/category";
import { useEffect } from "react";

export default function AdminTagCreate() {
  // context
  const {
    name,
    setName,
    parentCategory,
    setParentCategory,
    updatingTag,
    setUpdatingTag,
    createTag,
    updateTag,
    deleteTag,
  } = useTag();
  const { fetchCategories, categories } = useCategory();

  useEffect(() => {
    fetchCategories();
  }, []);

  return (
    <>
      <p>Create tag</p>
      <input
        type="text"
        value={updatingTag ? updatingTag.name : name}
        placeholder="Tag Name"
        onChange={(e) =>
          updatingTag
            ? setUpdatingTag({
                ...updatingTag,
                name: e.target.value,
              })
            : setName(e.target.value)
        }
        className="form-control p-2 my-2"
      />

      <div className="form-group mt-4">
        <label>Parent category</label>
        <select
          name="category"
          className="form-control"
          onChange={(e) =>
            updatingTag
              ? setUpdatingTag({
                  ...updatingTag,
                  parentCategory: e.target.value,
                })
              : setParentCategory(e.target.value)
          }
        >
          <option value="">Select one</option>
          {categories.length > 0 &&
            categories.map((c) => (
              <option
                key={c._id}
                value={c._id}
                selected={
                  c._id === updatingTag?.parent || c._id ===
parentCategory
                }
              >
                {c.name}
              </option>
            ))}
        </select>
      </div>
      {/* <pre>{JSON.stringify(updatingTag, null, 4)}</pre> */}

      <div className="d-flex justify-content-between">
        <button
          className={`btn bg-${updatingTag ? "info" : "primary"} text-
light`}
          onClick={(e) => {
            e.preventDefault();
            updatingTag ? updateTag() : createTag();
          }}
        >
          {updatingTag ? "Update" : "Create"}
        </button>

        {updatingTag && (
          <>
            <button
              className={`btn bg-danger text-light`}
              onClick={(e) => {
                e.preventDefault();
                deleteTag();
              }}
            >
              Delete
            </button>

            <button
              className="btn bg-success text-light"
              onClick={() => setUpdatingTag(null)}
            >
              Clear
            </button>
          </>
        )}
      </div>
    </>
  );
}
```

Tag list component

```
// components/tag/TagList
"use client";
import { useEffect } from "react";
import { useTag } from "@/context/tag";

export default function TagsList() {
  // context
  const { tags, fetchTags, setUpdatingTag } = useTag();
  useEffect(() => {
    fetchTags();
  }, []);

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          {tags.map((t) => (
            <button
              className="btn"
              onClick={() => {
                setUpdatingTag(t);
              }}
            >
              {t.name}
            </button>
          ))}
        </div>
      </div>
    </div>
  );
}
```

Trying API post routes using Postman (optional)

```
POST        http://localhost:3000/api/admin/category
Headers     next-auth.session-token=eyxxx...
```

Product model

```
// models/product

// npm i mongoose-unique-validator

import mongoose from "mongoose";
import uniqueValidator from "mongoose-unique-validator";
import Category from "@/models/category";
import Tag from "@/models/tag";

const ratingSchema = new mongoose.Schema(
  {
    rating: {
      type: Number,
      min: 1,
      max: 5,
    },
    comment: {
      type: String,
      maxlength: 200,
    },
    postedBy: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
    },
  },
  { timestamps: true } // Add timestamps
);

const likeSchema = new mongoose.Schema(
  {
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
    },
  },
  { timestamps: true } // Add timestamps
);

const productSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      trim: true,
      required: true,
      unique: true,
      maxlength: 160,
      text: true, // for text search
    },
    slug: {
      type: String,
      lowercase: true,
      index: true,
    },
    description: {
      type: String,
      trim: true,
      required: true,
      maxlength: 2000,
      text: true,
    },
    price: {
      type: Number,
      required: true,
      trim: true,
      maxlength: 32,
      validate: {
        validator: function (value) {
          return value !== 0;
        },
        message: "Price must be greater than 0.",
      },
    },
    previousPrice: Number,
    color: String,
    brand: String,
    stock: Number,
    shipping: {
      type: Boolean,
      default: true,
    },
    category: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Category",
    },
    tags: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Tag",
      },
    ],
    images: [
      {
        public_id: {
          type: String,
          default: "",
        },
        secure_url: {
          type: String,
          default: "",
        },
      },
    ],
    sold: {
      type: Number,
      default: 0,
    },
    likes: [likeSchema],
    // likes: [
    //   {
    //     type: mongoose.Schema.Types.ObjectId,
    //     ref: "User",
    //   },
    // ],
    ratings: [ratingSchema],
    // ratings: [
    //   {
    //     rating: {
    //       type: Number,
    //       min: 1,
    //       max: 5,
    //     },
    //     comment: {
    //       type: String,
    //       maxlength: 200,
    //     },
    //     postedBy: {
```

```
    //       type: mongoose.Schema.Types.ObjectId,
    //       ref: "User",
    //     },
    //   },
    // ],
  },
  { timestamps: true }
);

productSchema.plugin(uniqueValidator);

export default mongoose.models.Product ||
  mongoose.model("Product", productSchema);
```

Product create API

```
// api/admin/product/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import slugify from "slugify";

export async function POST(req) {
  const _req = await req.json();
  await dbConnect();

  try {
    // console.log(_req);
    const product = await Product.create({
      ..._req,
      slug: slugify(_req.title),
    });

    return NextResponse.json(product);
  } catch (err) {
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

```
// model route context provider page component
// ask chatgpt to give you product json data to paste in postman
```

Get products list with pagination

```
// api/product/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import queryString from "query-string";

/**
 * route query alternatives
 * req.nextUrl.searchParams.get('xyz')
 */

export async function GET(req) {
  await dbConnect();

  const searchParams = queryString.parseUrl(req.url).query;

  const { page } = searchParams || {};
  const pageSize = 6;

  try {
    const currentPage = Number(page) || 1;
    const skip = (currentPage - 1) * pageSize;
    const totalProducts = await Product.countDocuments({});

    const products = await Product.find({})
      .skip(skip)
      .limit(pageSize)
      .sort({ createdAt: "-1" });

    return NextResponse.json(
      {
        products,
        currentPage,
        totalPages: Math.ceil(totalProducts / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Get single product

```
// api/product/[slug]/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";

export async function GET(req, context) {
  await dbConnect();

  try {
    const product = await Product.findOne({ slug: context.params.slug });
    return NextResponse.json(product);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Product update and delete by admin

```
// api/admin/product/[id]/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";

export async function PUT(req, context) {
  await dbConnect();

  const _req = await req.json();
  // console.log("context ===================> ", context.params);

  try {
    const updatedProduct = await Product.findByIdAndUpdate(
      context.params.id,
      { ..._req },
      { new: true }
    );

    return NextResponse.json(updatedProduct);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err,
      },
      { status: 500 }
    );
  }
}

export async function DELETE(req, context) {
  // console.log("context in DELETE ===================> ",
context.params.id);

  await dbConnect();

  try {
    const deletedProduct = await
Product.findByIdAndDelete(context.params.id);

    return NextResponse.json(deletedProduct);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Product context

```
// context/product.js
"use client";
import { createContext, useState, useEffect, useContext } from "react";
import toast from "react-hot-toast";
import { useRouter } from "next/navigation";

export const ProductContext = createContext();

export const ProductProvider = ({ children }) => {
  // State
  const [product, setProduct] = useState(null);
  const [products, setProducts] = useState([]);
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);
  const [updatingProduct, setUpdatingProduct] = useState(null);
  const [uploading, setUploading] = useState(false);

  const router = useRouter();

  const uploadImages = (e) => {
    console.log(e.target.files);
  };
  const deleteImage = (public_id) => {
    //
  };

  const createProduct = async () => {
    try {
      const response = await fetch(`${process.env.API}/admin/product`, {
        method: "POST",
        body: JSON.stringify(product),
      });

      const data = await response.json();

      if (!response.ok) {
        toast.error(data.err);
      } else {
        toast.success(`Product "${data?.title}" created`);
        // router.push("/dashboard/admin/products");
        window.location.reload();
      }
    } catch (err) {
      console.log(err);
    }
  };

  const fetchProducts = async (page = 1) => {
    try {
      const response = await fetch(`${process.env.API}/product?
page=${page}`, {
        method: "GET",
      });

      const data = await response.json();

      if (!response.ok) {
        toast.error(data?.err);
      } else {
        setProducts(data?.products);
        setCurrentPage(data?.currentPage);
        setTotalPages(data?.totalPages);
      }
    } catch (err) {
      console.log(err);
    }
  };

  const updateProduct = async () => {
    try {
      const response = await fetch(
        `${process.env.API}/admin/product/${updatingProduct?._id}`,
        {
          method: "PUT",
          body: JSON.stringify(updatingProduct),
        }
      );

      const data = await response.json();

      if (!response.ok) {
        toast.error(data?.err);
      } else {
        toast.success(`Product "${data?.title}" updated`);
        router.back();
      }
    } catch (err) {
      console.log(err);
    }
  };

  const deleteProduct = async () => {
    try {
      const response = await fetch(
        `${process.env.API}/admin/product/${updatingProduct?._id}`,
        {
          method: "DELETE",
        }
      );

      const data = await response.json();

      if (!response.ok) {
        toast.error(data?.err);
      } else {
        toast.success(`Product "${data?.title}" deleted`);
        router.back();
      }
    } catch (err) {
      console.log(err);
    }
  };

  return (
    <ProductContext.Provider
      value={{
        product,
        setProduct,
        products,
        setProducts,
        currentPage,
        setCurrentPage,
        totalPages,
        setTotalPages,
        updatingProduct,
        setUpdatingProduct,
        uploading,
        setUploading,
        uploadImages,
        deleteImage,
        createProduct,
        fetchProducts,
        updateProduct,
        deleteProduct,
      }}
    >
      {children}
    </ProductContext.Provider>
  );
};

export const useProduct = () => useContext(ProductContext);
```

Admin create and update product

```
// app/dashboard/admin/layout
<Link
  className="nav-link"
  href="/dashboard/admin/product"
>
  Add Product
</Link>
<Link className="nav-link" href="/dashboard/admin/products">
  Products List
</Link>

// app/dashboard/admin/product/page
"use client";
import ProductCreate from "@/components/product/admin/ProductCreate";

export default function AddProduct() {
  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          <ProductCreate />
        </div>
      </div>
    </div>
  );
}
```

```
// components/product/admin/ProductCreate.js
"use client";
import { useEffect } from "react";
import { useProduct } from "@/context/product";
import { useCategory } from "@/context/category";
import { useTag } from "@/context/tag";
```

```
export default function ProductCreate() {
  const {
    product,
    setProduct,
    updatingProduct,
    setUpdatingProduct,
    createProduct,
    updateProduct,
    deleteProduct,
    uploading,
    setUploading,
    uploadImages,
    deleteImage,
  } = useProduct();

  const { categories, fetchCategories } = useCategory();
  const { tags, fetchTags } = useTag();

  useEffect(() => {
    fetchCategories();
    fetchTags();
  }, []);

  return (
    <div>
      <p className="lead">{updatingProduct ? "Update" : "Create"}
Product</p>

      <input
        type="text"
        placeholder="Title"
        value={updatingProduct ? updatingProduct?.title : product?.title}
        onChange={(e) =>
          updatingProduct
            ? setUpdatingProduct({ ...updatingProduct, title:
e.target.value })
            : setProduct({ ...product, title: e.target.value })
        }
        className="form-control p-2 my-2"
      />

      <textarea
        rows="5"
        className="form-control p-2 mb-2"
        placeholder="Description"
        value={
          updatingProduct ? updatingProduct?.description :
product?.description
        }
        onChange={(e) =>
          updatingProduct
            ? setUpdatingProduct({
                ...updatingProduct,
                description: e.target.value,
              })
            : setProduct({ ...product, description: e.target.value })
        }
      ></textarea>

      <input
        type="number"
        placeholder="Price"
        min="1"
        class="form-control p-2 mb-2"
        value={updatingProduct ? updatingProduct?.price : product?.price}
        onChange={(e) =>
          updatingProduct
            ? setUpdatingProduct({
                ...updatingProduct,
                price: e.target.value,
              })
            : setProduct({ ...product, price: e.target.value })
        }
      />

      <input
        type="text"
        placeholder="Color"
        value={updatingProduct ? updatingProduct?.color : product?.color}
        onChange={(e) =>
          updatingProduct
            ? setUpdatingProduct({ ...updatingProduct, color:
e.target.value })
            : setProduct({ ...product, color: e.target.value })
        }
        className="form-control p-2 my-2"
      />

      <input
        type="text"
        placeholder="Brand"
        value={updatingProduct ? updatingProduct?.brand : product?.brand}
        onChange={(e) =>
          updatingProduct
            ? setUpdatingProduct({ ...updatingProduct, brand:
e.target.value })
            : setProduct({ ...product, brand: e.target.value })
        }
        className="form-control p-2 my-2"
      />

      <input
        type="number"
        placeholder="Stock"
        min="1"
        class="form-control p-2 mb-2"
        value={updatingProduct ? updatingProduct?.stock : product?.stock}
        onChange={(e) =>
```

```
            updatingProduct
              ? setUpdatingProduct({
                  ...updatingProduct,
                  stock: e.target.value,
                })
              : setProduct({ ...product, stock: e.target.value })
          }
        />

        <div className="form-group">
          <select
            name="category"
            className="form-control p-2 mb-2"
            onChange={(e) => {
              const categoryId = e.target.value;
              const categoryName =

e.target.options[e.target.selectedIndex].getAttribute("name");

              const category = categoryId
                ? { _id: categoryId, name: categoryName }
                : null;

              if (updatingProduct) {
                setUpdatingProduct({
                  ...updatingProduct,
                  category,
                });
              } else {
                setProduct({ ...product, category });
              }
            }}
            value={
              updatingProduct
                ? updatingProduct?.category?._id
                : product?.category?._id
            }
          >
            <option value="">Select Category</option>
            {categories?.map((c) => (
              <option key={c._id} value={c._id} name={c?.name}>
                {c.name}
              </option>
            ))}
          </select>
        </div>

        <div className="d-flex flex-wrap justify-content-evenly align-items-
center">
          {tags
            ?.filter(
              (ft) =>
                ft?.parentCategory ===
                (updatingProduct?.category?._id || product?.category?._id)
            )
            ?.map((tag) => (
              <div key={tag?._id} className="form-check">
                <input
                  type="checkbox"
                  value={tag?._id}
                  onChange={(e) => {
                    const tagId = e.target.value;
                    const tagName = tag?.name;

                    let selectedTags = updatingProduct
                      ? [...(updatingProduct?.tags ?? [])]
                      : [...(product?.tags ?? [])];

                    if (e.target.checked) {
                      selectedTags.push({ _id: tagId, name: tagName });
                    } else {
                      selectedTags = selectedTags.filter((t) => t._id !==
  tagId);
                    }

                    if (updatingProduct) {
                      setUpdatingProduct({
                        ...updatingProduct,
                        tags: selectedTags,
                      });
                    } else {
                      setProduct({ ...product, tags: selectedTags });
                    }
                  }}
                />{" "}
                <label>{tag?.name}</label>
              </div>
            ))}
        </div>

        <div className="form-group mb-3">
          <label
            className={`btn btn-primary col-12 ${uploading ? "disabled" :
  ""}`}
          >
            {uploading ? "Processing" : "Upload Images"}
            <input
              type="file"
              multiple
              hidden
              accept="images/*"
              onChange={uploadImages}
              disabled={uploading}
            />
          </label>
        </div>

        <pre>{JSON.stringify(product, null, 4)}</pre>
      </div>
    );
  }
```

Uploading images with client side resize

```
// context/product
// npm i react-image-file-resizer

const uploadImages = (e) => {
  let files = e.target.files;

  let allUploadedFiles = updatingProduct
    ? updatingProduct.images || []
    : product
    ? product.images || []
    : [];

  if (files) {
    // Check if the total combined images exceed 10
    const totalImages = allUploadedFiles.length + files.length;
    if (totalImages > 4) {
      alert("You can't upload more than 4 images.");
      return;
    }

    setUploading(true);
    const uploadPromises = [];

    for (let i = 0; i < files.length; i++) {
      const file = files[i];
      const promise = new Promise((resolve) => {
        Resizer.imageFileResizer(
          file,
          1280,
          720,
          "JPEG",
          100,
          0,
          (uri) => {
            fetch(`${process.env.API}/admin/upload/image`, {
              method: "POST",
              headers: {
                "Content-Type": "application/json",
              },
              body: JSON.stringify({ image: uri }),
            })
              .then((response) => response.json())
              .then((data) => {
                // Insert the new image at the beginning of the array
                allUploadedFiles.unshift(data);
                resolve();
              })
              .catch((err) => {
                console.log("CLOUDINARY UPLOAD ERR", err);
                resolve();
              });
          },
          "base64"
        );
      });

      uploadPromises.push(promise);
    }

    Promise.all(uploadPromises)
      .then(() => {
        // Update the state after all images are uploaded
        updatingProduct
          ? setUpdatingProduct({
              ...updatingProduct,
              images: allUploadedFiles,
            })
          : setProduct({ ...product, images: allUploadedFiles });

        setUploading(false);
      })
      .catch((error) => {
        console.log("Error uploading images: ", error);
        setUploading(false);
      });
  }
};

const deleteImage = (public_id) => {
  setUploading(true);
  fetch(`${process.env.API}/admin/upload/image`, {
    method: "PUT",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ public_id }),
  })
    .then((response) => response.json())
    .then((data) => {
      // console.log("IMAGE DELETE RES DATA", data);
      const filteredImages = updatingProduct
        ? updatingProduct.images.filter(
            (image) => image.public_id !== public_id
          )
        : product.images.filter((image) => image.public_id !== public_id);

      updatingProduct
        ? setUpdatingProduct({
            ...updatingProduct,
            images: filteredImages,
          })
        : setProduct({ ...product, images: filteredImages });
    })
    .catch((err) => {
      toast.error("Image delete failed");
      console.log("CLOUDINARY DELETE ERR", err);
    })
    .finally(() => {
      setUploading(false);
    });
};
```

Signup to cloudinary to get the credentials

Image upload API

```
// api/admin/upload/image
import { NextResponse } from "next/server";
import cloudinary from "cloudinary";

// config
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINRAY_API_SECRET,
});

export async function POST(req) {
  const { image } = await req.json();

  try {
    const result = await cloudinary.uploader.upload(image);
    return NextResponse.json({
      public_id: result.public_id,
      secure_url: result.secure_url,
    });
  } catch (err) {
    console.log(err);
  }
}

export async function PUT(req) {
  const { public_id } = await req.json();

  try {
    const result = await cloudinary.uploader.destroy(public_id);
    return NextResponse.json({ success: true });
  } catch (err) {
    console.log(err);
    }
  }
```

Image uploads in product create

```
// ProductCreate.js

// preview images
const imagePreviews = updatingProduct
  ? updatingProduct?.images ?? []
  : product?.images ?? [];

<div className="d-flex justify-content-center">
  {imagePreviews?.map((img) => (
    <div key={img?.public_id}>
      <img
        src={img?.secure_url}
        className="img-thumbnail mx-1 shadow"
        style={{ width: "100px", height: "100px", objectFit: "cover" }}
      />
      <br />
      <div
        className="text-center pointer"
        onClick={() => deleteImage(img?.public_id)}
      >
        ❌
      </div>
    </div>
  ))}
</div>;
```

Product Create, update, delete or clear buttons

```
// ProductCreate.js
<div className="d-flex justify-content-between mt-3">
  <button
    className={`btn btn-raised btn-${updatingProduct ? "info" :
"primary"}`}
    onClick={() => (updatingProduct ? updateProduct() : createProduct())}
  >
    {updatingProduct ? "Update" : "Create"}
  </button>

  {updatingProduct && (
    <>
      <button onClick={() => deleteProduct()} className="btn btn-danger">
        Delete
      </button>
      <button
```

```
          onClick={() => window.location.reload()}
          className="btn btn-danger"
        >
          Clear
        </button>
      </>
    )}
  </div>
```

Admin products list

```
// dashboard/admin/products/page
"use client";
import ProductList from "@/components/product/admin/ProductList";

export default function AddProduct() {
  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          <p className="lead mb-4">List of Products</p>
          <ProductList />
        </div>
      </div>
    </div>
  );
}
```

Products list component for admin

```
// component/product/admin/ProductList
"use client";
import { useEffect } from "react";
import { useProduct } from "@/context/product";
import { useRouter, usePathname, useSearchParams } from "next/navigation";
import Link from "next/navigation";

export default function ProductList() {
  const {
    products,
    currentPage,
    totalPages,
    fetchProducts,
    setUpdatingProduct,
  } = useProduct();

  const router = useRouter();
  const pathname = usePathname();
  const searchParams = useSearchParams();
  const page = searchParams.get("page");

  useEffect(() => {
    fetchProducts(page);
  }, [page]);

  return (
    <div className="container my-5">
      <div className="row">
        <pre>{JSON.stringify(products, null, 4)}</pre>
      </div>
    </div>
  );
}
```

Display product info in ProductList

```
"use client";
import { useEffect } from "react";
import { useProduct } from "@/context/product";
import { useRouter, usePathname, useSearchParams } from "next/navigation";
import Link from "next/link";
import Image from "next/image";

export default function ProductList() {
  const {
    products,
    currentPage,
    totalPages,
    fetchProducts,
    setUpdatingProduct,
  } = useProduct();

  const router = useRouter();
  const pathname = usePathname();
  const searchParams = useSearchParams();
  const page = searchParams.get("page");

  useEffect(() => {
    fetchProducts(page);
  }, [page]);

  return (
    <div className="container my-5">
      <div className="row gx-3">
        {/* <pre>{JSON.stringify(products, null, 4)}</pre> */}
        {products?.map((product) => (
          <div key={product?._id} className="col-lg-6 my-3">
            <div style={{ height: "200px", overflow: "hidden" }}>
              <Image
                src={product?.images[0]?.secure_url ||
"/images/default.jpeg"}
                alt={product?.title}
                width={500}
                height={300}
                style={{
                  objectFit: "cover",
                  height: "100%",
                  width: "100%",
                }}
              />
            </div>
            <div className="card-body">
              <h5 className="card-title">
                <Link href={`/product/${product?.slug}`}>
                  ${product?.price?.toFixed(2)} {product?.title}
                </Link>
              </h5>

              <p className="card-text">
                <div
                  dangerouslySetInnerHTML={{
                    __html:
                      product?.description?.length > 160
                        ? `${product?.description?.substring(0, 160)}..`
                        : product?.description,
                  }}
                />
              </p>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
}
```

```
// components/product/admin/ProductList
"use client";
import { useEffect } from "react";
import { useProduct } from "@/context/product";
import { useRouter, usePathname, useSearchParams } from "next/navigation";
import Link from "next/link";
import ProductCard from "@/components/product/ProductCard";

export default function AdminProducts() {
  // context
  const {
    products,
    currentPage,
    totalPages,
    fetchProducts,
    setUpdatingProduct,
  } = useProduct();

  const router = useRouter();
  const pathname = usePathname();
  const searchParams = useSearchParams();
  const page = searchParams.get("page");

  useEffect(() => {
    fetchProducts(page);
  }, [page]);

  return (
     { /* rest of code */ }

      {/* <pre>{JSON.stringify(currentPage, null, 4)}</pre> */}

      <div className="row">
        <div className="col">
          <nav className="d-flex justify-content-center">
            <ul className="pagination">
              {Array.from({ length: totalPages }, (_, index) => {
                const page = index + 1;
                return (
                  <li
                    key={page}
                    className={`page-item ${
                      currentPage === page ? " active" : ""
                    }`}
                  >
                    <Link
                      className="page-link"
                      href={`${pathname}?page=${page}`}
                      as={`${pathname}?page=${page}`}
                    >
                      {page}
                    </Link>
                  </li>
                );
              })}
            </ul>
          </nav>
        </div>
      </div>
    </div>
  );
}
```

Products display on home page

```
// app/page
import Image from "next/image";

async function getProducts(searchParams) {
  const searchQuery = new URLSearchParams({
    page: searchParams?.page || 1,
  }).toString();

  const response = await fetch(`${process.env.API}/product?
${searchQuery}`, {
    method: "GET",
    next: { revalidate: 1 },
  });

  if (!response.ok) {
    throw new Error("Failed to fetch products");
  }

  const data = await response.json();
  return data;
}

export default async function Home({ searchParams }) {
  // console.log("searchParams => ", searchParams);
  const data = await getProducts(searchParams);
  console.log(data);

  return (
    <div>
      <h1 className="d-flex justify-content-center align-items-center vh-
100 text-uppercase">
        Home
      </h1>
      <pre>{JSON.stringify(data, null, 4)}</pre>
    </div>
  );
}
```

Pagination component

```
// components/product/Pagination
import Link from "next/link";

export default function Pagination({ currentPage, totalPages, pathname })
{
  return (
    <div className="row">
      <div className="col">
        <nav className="d-flex justify-content-center">
          <ul className="pagination">
```

```
              {Array.from({ length: totalPages }, (_, index) => {
                const page = index + 1;
                return (
                  <li
                    key={page}
                    className={`page-item ${
                      currentPage === page ? " active" : ""
                    }`}
                  >
                    <Link
                      className="page-link"
                      href={`${pathname}?page=${page}}`}
                      as={`${pathname}?page=${page}`}
                    >
                      {page}
                    </Link>
                  </li>
                );
              })}
            </ul>
          </nav>
        </div>
      </div>
    );
  }

  // now use in '/shop'
  <Pagination
    currentPage={data.currentPage}
    totalPages={data.totalPages}
    pathname="/shop"
  />;
  // use it in admin products list component
  <Pagination
    currentPage={currentPage}
    totalPages={totalPages}
    pathname={pathname}
  />;
```

Product card component

Import and use in home page on products?.map()

```
// components/product/ProductCard
import Image from "next/image";
import Link from "next/link";
import dayjs from "dayjs";
import relativeTime from "dayjs/plugin/relativeTime";

dayjs.extend(relativeTime);
```

```
export default function ({ product }) {
  return (
    <div key={product?._id} className="card my-3">
      <div style={{ height: "200px", overflow: "hidden" }}>
        <Image
          src={product?.images?.[0]?.secure_url || "/images/default.jpeg"}
          width={500}
          height={300}
          style={{ objectFit: "cover", width: "100%", height: "100%" }}
          alt={product?.title}
        />
      </div>

      <div className="card-body">
        <h5 className="card-title">{product?.title}</h5>
        <div
          dangerouslySetInnerHTML={{
            __html:
              product?.description?.length > 160
                ? `${product?.description?.substring(0, 160)}..`
                : product?.description,
          }}
        />
      </div>
      {/* before accessing category and tags, make sure .populate() is
used in api routes and ref: 'Category' models are imported in `Product`
model */}
      <div className="card-footer d-flex justify-content-between">
        <small>Category: {product?.category?.name}</small>
        <small>Tags: {product?.tags?.map((t) => t?.name).join(" ")}
</small>
      </div>

      <div className="card-footer d-flex justify-content-between">
        <small>❤ Likes</small>
        <small>Posted {dayjs(product?.createdAt).fromNow()}</small>
      </div>

      <div className="card-footer d-flex justify-content-between">
        <small>Brand: {product?.brand}</small>
        <small>🌟 Stars</small>
      </div>
    </div>
  );
}
```

Product Single View Page

```
// app/product/[slug]/page
import dayjs from "dayjs";
import relativeTime from "dayjs/plugin/relativeTime";
import Image from "next/image";
import ProductImage from "@/components/product/ProductImage";

dayjs.extend(relativeTime);

async function getProducts(slug) {
  try {
    const response = await fetch(`${process.env.API}/product/${slug}`, {
      method: "GET",
      next: { revalidate: 1 },
    });

    if (!response.ok) {
      throw new Error(`Failed to fetch products`);
    }

    const data = await response.json();
    return data;
  } catch (error) {
    console.error(error);
    return null;
  }
}

export default async function ProductViewPage({ params }) {
  const product = await getProducts(params.slug);

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col-lg-8 mb-4">
          <div className="card">
            {/* images and preview modal */}
            <ProductImage product={product} />

            {/* card body */}
            <div className="card-body">
              <h5 className="card-title">{product.title}</h5>
              <div className="card-text">
                <div
                  dangerouslySetInnerHTML={{
                    __html: product.description,
                  }}
                ></div>
              </div>
            </div>

            <div className="card-footer d-flex justify-content-between">
              <small className="text-muted">
                Category: {product.category.name}
              </small>
              <small className="text-muted">
                Tags: {product.tags.map((tag) => tag.name).join(" ")}
              </small>
            </div>
            <div className="card-footer d-flex justify-content-between">
              <small className="text-muted">Ratings</small>
              <small className="text-muted">
                Added {dayjs(product.createdAt).fromNow()}
              </small>
            </div>
          </div>
        </div>

        <div className="col-lg-4">Add to cart / wishlist</div>
      </div>

      <div className="row">
        <div className="col my-5">
          <p className="lead">Related products</p>
        </div>
      </div>
    </div>
  );
}
```

Product Images and Preview Modal

```
// components/product/ProductImage
"use client";
import Image from "next/image";
import { useState, useEffect } from "react";

export default function ProductImage({ product }) {
  const [showImagePreviewModal, setShowImagePreviewModal] =
useState(false);
  const [currentImagePreviewUrl, setCurrentImagePreviewUrl] =
useState("");

  const openModal = (url) => {
    setCurrentImagePreviewUrl(url);
    setShowImagePreviewModal(true);
  };

  const closeModal = () => {
    setShowImagePreviewModal(false);
    setCurrentImagePreviewUrl("");
  };

  const showImage = (src, title) => (
    <Image
      src={src}
      className="card-img-top"
      width={500}
      height={300}
```

```
        style={{ objectFit: "contain", height: "100%", width: "100%" }}
        alt={title}
      />
    );

    return (
      <>
        {showImagePreviewModal && (
          <div className="modal fade show" style={{ display: "block" }}>
            <div
              className="modal-dialog modal-dialog-centered modal-lg"
              style={{ height: "calc(100% - 60px)" }}
            >
              <div
                className="modal-content"
                style={{ height: "calc(100% - 60px)" }}
              >
                <div className="modal-body overflow-auto">
                  {showImage(currentImagePreviewUrl, product?.title)}
                </div>
                <div className="modal-footer">
                  <button
                    type="button"
                    className="btn btn-secondary"
                    data-bs-dismiss="modal"
                    onClick={closeModal}
                  >
                    Close
                  </button>
                </div>
              </div>
            </div>
          </div>
        )}
        <div className="d-flex justify-content-center align-items-center">
          {product?.images?.length > 0 ? (
            <>
              {product?.images?.map((image) => (
                <div
                  key={image.public_id}
                  style={{ height: "350px", overflow: "hidden" }}
                  className="pointer"
                  onClick={() => openModal(image?.secure_url)}
                >
                  {showImage(image?.secure_url, product?.title)}
                </div>
              ))}
            </>
          ) : (
            <div
              style={{ height: "350px", overflow: "hidden" }}
              className="pointer"
              onClick={() => openModal("/images/default.jpeg")}
            >
              {showImage("/images/default.jpeg", product?.title)}
            </div>
          )}
        </div>
      </>
    );
  }
```

Modal close on page click

```
// components/product/ProductImage
useEffect(() => {
  // close modal on clicks on the page
  window.addEventListener("click", handleClickOutside);
  return () => {
    window.removeEventListener("click", handleClickOutside);
  };

  function handleClickOutside(event) {
    if (event.target.classList.contains("modal")) {
      closeModal();
    }
  }
}, []);
```

Current user from server session

```
// utils/currentUser.js
import { getServerSession } from "next-auth/next";
import { authOptions } from "@/utils/authOptions";

export const currentUser = async () => {
  const session = await getServerSession(authOptions);
  return session.user;
};
```

Product Like API

```
// api/user/product/like/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import { currentUser } from "@/utils/currentUser";

export async function PUT(req) {
  await dbConnect();
  const user = await currentUser();
  const { productId } = await req.json();

  try {
    const updated = await Product.findByIdAndUpdate(
      productId,
      {
        $addToSet: { likes: user._id },
      },
      { new: true }
    );
    return NextResponse.json(updated);
  } catch (err) {
    return NextResponse.json({ err: err.message }, { status: 500 });
  }
}
```

Product Unlike API

```
// api/user/product/unlike/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import { getToken } from "next-auth/jwt";

export async function PUT(req) {
  await dbConnect();

  const _req = await req.json();

  const { productId } = _req;
  const token = await getToken({
    req,
    secret: process.env.NEXTAUTH_SECRET,
  });

  try {
    const updated = await Product.findByIdAndUpdate(
      productId,
      { $pull: { likes: token.user._id } },
      { new: true }
    );

    return NextResponse.json(updated);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

User Liked Products API

```
// api/user/product/liked/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import { getToken } from "next-auth/jwt";

export async function GET(req) {
  await dbConnect();

  const token = await getToken({
    req,
    secret: process.env.NEXTAUTH_SECRET,
  });
  // console.log("token in user liked-products => ", token);

  try {
    const likedProducts = await Product.find({ likes: token.user._id });

    return NextResponse.json(likedProducts);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

User Product Like Unlike Component

```
// components/product/ProductLike
"use client";
import { useState } from "react";
import { useSession } from "next-auth/react";
import toast from "react-hot-toast";
import { useRouter, usePathname } from "next/navigation";

export default function ProductLike({ product }) {
  const { data, status } = useSession();
  // console.log("product", product);
  const [likes, setLikes] = useState(product?.likes);
  const router = useRouter();
  const pathname = usePathname();

  const isLiked = likes?.includes(data?.user?._id);

  const handleLike = async () => {
    if (status !== "authenticated") {
      toast.error("Please login to like");
      router.push(
        `/login?callbackUrl=${process.env.API.replace("/api",
"")}${pathname}`
      );

      return;
    }
    try {
      if (isLiked) {
        const answer = window.confirm("You liked it. Want to unlike?");
        if (answer) {
          handleUnlike();
        }
      } else {
        const options = {
          method: "PUT",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            productId: product._id,
          }),
        };

        const response = await fetch(
          `${process.env.API}/user/product/like`,
          options
        );
        if (!response.ok) {
          throw new Error(
            `Failed to like: ${response.status} ${response.statusText}`
          );
        }

        const data = await response.json();
        // console.log("product liked response => ", data);
        setLikes(data.likes);
        toast.success("Product liked");
        router.refresh(); // only works in server components
      }
    } catch (err) {
      console.log(err);
      toast.error("Error liking product");
    }
  };
  const handleUnlike = async () => {
    try {
      const options = {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          productId: product._id,
        }),
      };

      const response = await fetch(
        `${process.env.API}/user/product/unlike`,
        options
      );
      if (!response.ok) {
        throw new Error(`Failed to unlike`);
      }

      const data = await response.json();
      // console.log("product unliked response => ", data);
      setLikes(data.likes);
      toast.success("Product unliked");
      router.refresh();
    } catch (err) {
      console.log(err);
      toast.error("Error unliking product");
    }
  };

  // 🖤

  return (
    <small className="text-muted pointer">
      {!likes?.length ? (
        <span onClick={handleLike}>❤ Be the first person to like</span>
      ) : (
        <span onClick={handleLike}>❤ {likes?.length} people liked</span>
      )}
    </small>
  );
}
```

API update to fix likes issues:

```
// models/product
likes: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "User",
      },
    ],

// like route
{ $addToSet: { likes: user._id } },

// unlike route
 $pull: { likes: user._id },
```

Reusable Modal component

```
// components/Modal
"use client";
import { useProduct } from "@/context/product";

export default function ProductImage({ children }) {
  const { closeModal } = useProduct();

  return (
    <>
      <div className="modal fade show" style={{ display: "block" }}>
        <div
          className="modal-dialog modal-dialog-centered modal-lg"
          style={{ height: "calc(100% - 60px)" }}
        >
          <div
            className="modal-content"
            style={{ height: "calc(100% - 60px)" }}
          >
            <div className="modal-body overflow-auto">{children}</div>
            <div className="modal-footer">
              <button
                type="button"
                className="btn btn-secondary"
                data-bs-dismiss="modal"
                onClick={closeModal}
              >
                Close
              </button>
            </div>
          </div>
        </div>
      </div>
    </>
  );
}
```

Using Modal component

```
// ProductImage
{
  showImagePreviewModal && (
    <Modal>{showImage(currentImagePreviewUrl, product?.title)}</Modal>
  );
}
```

5 star rating system

```
// context/product
  const [showRatingModal, setShowRatingModal] = useState(false);
  const [currentRating, setCurrentRating] = useState(0);
  const [comment, setComment] = useState("");

  // modal for image preview and ratings
  const openImagePreviewModal = (url) => {
    setCurrentImagePreviewUrl(url);
    setShowImagePreviewModal(true);
  };

  const closeModal = () => {
    setShowImagePreviewModal(false);
    setShowRatingModal(false);
  };

  const handleClickOutside = (event) => {
    if (event.target.classList.contains("modal")) {
      closeModal();
    }
  };

  // components/Modal
  "use client";
import { useProduct } from "@/context/product";

export default function Modal({ children }) {
  // context
  const { closeModal } = useProduct();

  return (
    <>
      <div
        className="modal fade show"
        style={{ display: "block", maxHeight: "100vh", overflow: "auto" }}
      >
        <div className="modal-dialog modal-dialog-centered modal-lg">
          <div className="modal-content">
            <div className="modal-body">{children}</div>
            <div className="modal-footer">
              <button
```

```
                type="button"
                className="btn btn-secondary"
                data-bs-dismiss="modal"
                onClick={closeModal}
              >
                Close
              </button>
            </div>
          </div>
        </div>
      </div>
    </>
  );
}


// component/product/Stars
import { FaStar, FaStarHalfAlt, FaRegStar } from "react-icons/fa";

export default function Stars({ rating }) {
  const stars = [];

  for (let i = 1; i <= 5; i++) {
    if (i <= rating) {
      // push all gold stars
      stars.push(<FaStar key={i} className="text-danger" />);
    } else if (i === Math.ceil(rating) && rating % 1 >= 0.5) {
      // push half gold star if any
      stars.push(<FaStarHalfAlt key={i} className="text-danger" />);
    } else {
      // push empty star
      stars.push(<FaRegStar key={i} />);
    }
  }

  return <>{stars}</>;
}




// utils/helpers
export function calculateAverageRating(ratings) {
  let totalRating = 0;
  for (const ratingObj of ratings) {
    totalRating += ratingObj.rating;
  }
  const averageRating = totalRating / ratings.length;
  return averageRating;
}

// components/product/ProductRating
"use client";
import { useState, useEffect } from "react";
import { toast } from "react-hot-toast";
import { useRouter, usePathname } from "next/navigation";
import { useProduct } from "@/context/product";
import Stars from "@/components/product/Stars";
import { calculateAverageRating } from "@/utils/helpers";
import Modal from "@/components/Modal";
import { useSession } from "next-auth/react";
import { FaStar, FaRegStar } from "react-icons/fa";

export default function ProductRating({ product }) {
  const {
    showRatingModal,
    setShowRatingModal,
    currentRating,
    setCurrentRating,
    comment,
    setComment,
  } = useProduct();

  const [productRatings, setProductRatings] = useState(product?.ratings ||
[]);
  const [averageRating, setAverageRating] = useState(0);

  // current user
  const { data, status } = useSession();

  const router = useRouter();
  const pathname = usePathname();

  const alreadyRated = productRatings?.find(
    (rate) => rate?.postedBy?._id === data?.user?._id
  );

  useEffect(() => {
    if (alreadyRated) {
      setCurrentRating(alreadyRated?.rating);
      setComment(alreadyRated?.comment);
    } else {
      setCurrentRating(0);
      setComment("");
    }
  }, [alreadyRated]);

  useEffect(() => {
    if (productRatings) {
      const average = calculateAverageRating(productRatings);
      setAverageRating(average);
    }
  }, [product?.ratings]);

  const submitRating = async () => {
    if (status !== "authenticated") {
      toast.error("You must be logged in to leave a rating");
      router.push(`/login?callbackUrl=${pathname}`);
      return;
    }

    try {
      const response = await
fetch(`${process.env.API}/user/product/rating`, {
        method: "POST",
        body: JSON.stringify({
          productId: product?._id,
          rating: currentRating,
          comment,
        }),
      });

      if (!response.ok) {
        throw new Error("Failed to leave a rating");
      }

      const data = await response.json();
      setProductRatings(data?.ratings);
      setShowRatingModal(false);
      toast.success("Thanks for leaving a rating");
      router.refresh();
    } catch (err) {
      console.log(err);
      toast.error("Error leaving a rating");
    }
  };

  return (
    <div className="d-flex justify-content-between card-footer">
      <div>
        <Stars rating={averageRating} />
        <small className="text-muted"> ({productRatings?.length})</small>
      </div>

      <small onClick={() => setShowRatingModal(true)} className="pointer">
        {alreadyRated ? "Update your rating" : "Leave a rating"}
      </small>

      {showRatingModal && (
        <Modal>
          <input
            type="text"
            className="form-control mb-3"
            placeholder="Write a review"
            value={comment}
            onChange={(e) => setComment(e.target.value)}
          />
          <div className="pointer">
            {[...Array(5)].map((_, index) => {
              const ratingValue = index + 1;
              return (
                <span
                  key={ratingValue}
                  className={
                    ratingValue <= currentRating ? "star-active lead" :
"lead"
                  }
                  onClick={() => setCurrentRating(ratingValue)}
                >
                  {ratingValue <= currentRating ? (
                    <FaStar className="text-danger" />
                  ) : (
                    <FaRegStar />
                  )}
                </span>
              );
            })}
          </div>

          <button
            onClick={submitRating}
            className="btn btn-primary btn-raised my-3"
          >
            Submit
          </button>
        </Modal>
      )}
    </div>
  );
}

// import <ProductRating /> in product/[slug]/page
  <div className="card-footer text-center">
    <ProductRating product={product} />
  </div>

// show rating in <ProductCard />
 <div className="card-footer">
    {/* <pre>{JSON.stringify(product?.ratings, null, 4)}</pre> */}
    <div className="d-flex justify-content-between align-items-center">
      <small className="text-muted">Brand: {product?.brand}</small>
      <div>
        <small>
          <Stars rating={calculateAverageRating(product?.ratings)} />
        </small>
        <small className="text-muted ml-1">
          {`(${product?.ratings?.length})`}
        </small>
      </div>
    </div>
  </div>
```

Rating API

```
// api/user/product/ratingimport { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import Order from "@/models/order";
import { getToken } from "next-auth/jwt";

export async function POST(req) {
  await dbConnect();

  const _req = await req.json();
  console.log("_req in rating route", _req);

  const { productId, rating, comment } = _req;
  const token = await getToken({
    req,
    secret: process.env.NEXTAUTH_SECRET,
  });

  try {
    const product = await Product.findById(productId);

    // Check if the user has already rated the product
    const existingRating = product.ratings.find(
      (rate) => rate.postedBy.toString() === token.user._id.toString()
    );

    // Check if the user has purchased the product
    const userPurchased = await Order.findOne({
      userId: token.user._id,
      "cartItems._id": productId,
    });

    if (!userPurchased) {
      return NextResponse.json(
        {
          err: "You can only leave a review for products you've
purchased.",
        },
        { status: 400 }
      );
    }

    if (existingRating) {
      // Update the existing rating
      existingRating.rating = rating;
      existingRating.comment = comment;
      await product.save();

      return NextResponse.json(product, { status: 200 });
    }

    // If the user has not already rated, add a new rating
    product.ratings.push({
      rating: rating,
      postedBy: token.user._id,
      comment: comment,
    });
    const updated = await product.save();

    return NextResponse.json(updated, { status: 200 });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Shop page for advance product filtering

```
// app/shop/page
import ProductFilter from "@/components/product/ProductFilter";

async function getProducts(searchParams) {
  const searchQuery = new URLSearchParams({
    page: searchParams.page || 1,
    minPrice: searchParams.minPrice || "",
    maxPrice: searchParams.maxPrice || "",
    ratings: searchParams.ratings || "",
    category: searchParams.category || "",
    tag: searchParams.tag || "",
    brand: searchParams.brand || "",
  }).toString();
  //
}

export default async function Shop({ searchParams }) {
  console.log("searchParams in shop page => ", searchParams);
  const data = await getProducts(searchParams);

  return (
    <div className="container-fluid">
      <div className="row">
        <div className="col-lg-3">
          <ProductFilter searchParams={searchParams} />
        </div>
        <div className="col-lg-9">Products list</div>
      </div>
    </div>
  );
}
```

Product filter component

Price range display and remove filtering

```
// utils/filterData
export const priceRanges = [
  { min: 0, max: 20, label: "Under $20" },
  { min: 20, max: 50, label: "$20 - $50" },
  { min: 50, max: 100, label: "$50 - $100" },
  { min: 100, max: 200, label: "$100 - $200" },
  { min: 200, max: 500, label: "$200 - $500" },
  { min: 500, max: 900, label: "$500 - $900" },
];
```

```
// components/product/ProductFilter
"use client";
import { priceRanges } from "@/utils/filterData";
import Link from "next/link";
import { useRouter } from "next/navigation";

export default function ProductFilter({ searchParams }) {
  const pathname = "/shop";
  const { minPrice, maxPrice, ratings, category, tag, brand } =
searchParams;

  const router = useRouter();

  const activeButton = "btn btn-primary btn-raised mx-1 rounded-pill";
  const button = "btn btn-secondary btn-raised mx-1 rounded-pill";

  const handleRemoveFilter = (filterName) => {
    const updatedSearchParams = { ...searchParams };
    // delete updatedSearchParams[filterName];

    // if filterName is string
    if (typeof filterName === "string") {
      delete updatedSearchParams[filterName];
    }
    // if filterName is array
    if (Array.isArray(filterName)) {
      filterName?.forEach((name) => {
        delete updatedSearchParams[name];
      });
    }

    // reset page to 1 when applying new filtering options
    updatedSearchParams.page = 1;

    const queryString = new
URLSearchParams(updatedSearchParams).toString();
    const newUrl = `${pathname}?${queryString}`;
    router.push(newUrl);
  };

  return (
    <div>
      <p className="lead">Filter Products</p>

      <Link className="text-danger" href="/shop">
        Clear Filters
      </Link>

      <p className="mt-4 alert alert-primary">Price</p>
      <div className="row d-flex align-items-center mx-1">
        {priceRanges?.map((range) => {
          const url = {
            pathname,
            query: {
              ...searchParams,
              minPrice: range?.min,
              maxPrice: range?.max,
              page: 1,
            },
          };
          const isActive =
            minPrice === String(range?.min) && maxPrice ===
String(range?.max);
          return (
            <div key={range?.label}>
              <Link href={url} className={isActive ? activeButton :
button}>
                {range?.label}
              </Link>
              {isActive && (
                <span
                  onClick={() => handleRemoveFilter(["minPrice",
"maxPrice"])}
                  className="pointer"
                >
                  X
                </span>
              )}
            </div>
          );
        })}
      </div>

      <pre>{JSON.stringify(searchParams, null, 4)}</pre>
    </div>
  );
}
```

Categories, Tags and Brands API

```
// api/categories/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Category from "@/models/category";

export async function GET() {
  await dbConnect();
  try {
    const categories = await Category.find({}).sort({ createdAt: -1 });
    return NextResponse.json(categories);
  } catch (err) {
    return NextResponse.json(err.message, { status: 500 });
  }
}

// api/tags/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Tag from "@/models/tag";

export async function GET() {
  await dbConnect();
  try {
    const tags = await Tag.find({}).sort({ createdAt: -1 });
    return NextResponse.json(tags);
  } catch (err) {
    return NextResponse.json(err.message, { status: 500 });
  }
}

// api/product/brands/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";

export async function GET(req) {
  await dbConnect();

  try {
    const brands = await Product.distinct("brand");
    return NextResponse.json(brands);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      { err: "An error occurred. Try again" },
      { status: 500 }
    );
  }
}
```

API request from context

```
// context/category
const fetchCategoriesPublic = async () => {
  try {
    const response = await fetch(`${process.env.API}/categories`);
    const data = await response.json();

    if (!response.ok) {
      toast.error(data);
    } else {
      setCategories(data);
    }
  } catch (err) {
    console.log(err);
    toast.error("An error occurred. Try again");
  }
};

// context/tag
const fetchTagsPublic = async () => {
  try {
    const response = await fetch(`${process.env.API}/tags`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    });

    const data = await response.json();

    if (!response.ok) {
      toast.error(data);
    } else {
      setTags(data);
    }
  } catch (err) {
    console.log(err);
    toast.error("Error creating tag");
  }
};

// context/product
const fetchBrands = async () => {
  try {
    const response = await fetch(`${process.env.API}/product/brands`, {
      method: "GET",
    });

    const data = await response.json();

    if (!response.ok) {
      toast.error(data?.err);
    } else {
      setBrands(data);
    }
  } catch (err) {
    console.log(err);
  }
};
```

Filtering products

```
// components/product/ProductFilter.js

"use client";
import { useEffect } from "react";
import { priceRanges } from "@/utils/filterData";
import Link from "next/link";
import { useRouter } from "next/navigation";
import Stars from "@/components/product/Stars";
import { useCategory } from "@/context/category";
import { useTag } from "@/context/tag";
import { useProduct } from "@/context/product";

export default function ProductFilter({ searchParams }) {
  const pathname = "/shop";
  const { minPrice, maxPrice, ratings, category, tag, brand } =
searchParams;
  // context
  const { fetchCategoriesPublic, categories } = useCategory();
  const { fetchTagsPublic, tags } = useTag();
  const { fetchBrands, brands } = useProduct();

  useEffect(() => {
    fetchCategoriesPublic();
    fetchTagsPublic();
    fetchBrands();
  }, []);

  const router = useRouter();

  const activeButton = "btn btn-primary btn-raised mx-1 rounded-pill";
  const button = "btn btn-raised mx-1 rounded-pill";

  const handleRemoveFilter = (filterName) => {
    const updatedSearchParams = { ...searchParams };
    // delete updatedSearchParams[filterName];

    // if filterName is string
    if (typeof filterName === "string") {
      delete updatedSearchParams[filterName];
    }
    // if filterName is array
    if (Array.isArray(filterName)) {
      filterName?.forEach((name) => {
        delete updatedSearchParams[name];
      });
    }

    // reset page to 1 when applying new filtering options
    updatedSearchParams.page = 1;

    const queryString = new
URLSearchParams(updatedSearchParams).toString();
    const newUrl = `${pathname}?${queryString}`;
    router.push(newUrl);
  };

  return (
    <div className="mb-5 overflow-scroll">
      <p className="lead">Filter Products</p>

      <Link className="text-danger" href="/shop">
        Clear Filters
      </Link>

      <p className="mt-4 alert alert-primary">Price</p>
      <div className="row d-flex align-items-center mx-1">
        {priceRanges?.map((range) => {
          const url = {
            pathname,
            query: {
              ...searchParams,
              minPrice: range?.min,
              maxPrice: range?.max,
              page: 1,
            },
          };
          const isActive =
            minPrice === String(range?.min) && maxPrice ===
String(range?.max);
          return (
            <div key={range?.label}>
              <Link href={url} className={isActive ? activeButton :
button}>
                {range?.label}
              </Link>
              {isActive && (
                <span
                  onClick={() => handleRemoveFilter(["minPrice",
"maxPrice"])}
                  className="pointer"
                >
                  X
                </span>
              )}
            </div>
          );
        })}
      </div>

      <p className="mt-4 alert alert-primary">Ratings</p>
      <div className="row d-flex align-items-center mx-1">
        {[5, 4, 3, 2, 1].map((ratingValue) => {
          const isActive = String(ratings) === String(ratingValue);

          const url = {
            pathname,
            query: {
              ...searchParams,
              ratings: ratingValue,
              page: 1,
            },
          };
          return (
            <div key={ratingValue}>
              <Link
                href={url}
                className={
                  isActive
                    ? "btn btn-primary btn-raised mx-1 rounded-pill"
                    : "btn btn-raised mx-1 rounded-pill"
                }
              >
                <Stars rating={ratingValue} />
              </Link>
              {isActive && (
                <span
                  onClick={() => handleRemoveFilter("ratings")}
                  className="pointer"
                >
                  X
                </span>
              )}
            </div>
          );
        })}
      </div>

      <p className="mt-4 alert alert-primary">Categories</p>
      <div className="row d-flex align-items-center mx-1 filter-scroll">
        {categories?.map((c) => {
          const isActive = category === c._id;

          const url = {
            pathname,
            query: {
              ...searchParams,
              category: c?._id,
              page: 1,
            },
          };
          return (
            <div key={c._id}>
              <Link href={url} className={isActive ? activeButton :
button}>
                {c?.name}
              </Link>
              {isActive && (
                <span
                  onClick={() => handleRemoveFilter("category")}
                  className="pointer"
                >
                  X
                </span>
              )}
            </div>
          );
        })}
      </div>

      {category && (
        <>
          <p className="mt-4 alert alert-primary">Tags</p>
          <div className="row d-flex align-items-center mx-1 filter-
scroll">
            {tags
              ?.filter((t) => t?.parentCategory === category)
              ?.map((t) => {
                const isActive = tag === t._id;

                const url = {
                  pathname,
                  query: {
                    ...searchParams,
                    tag: t?._id,
                    page: 1,
                  },
                };
                return (
                  <div key={t._id}>
                    <Link
                      href={url}
                      className={isActive ? activeButton : button}
                    >
                      {t?.name}
                    </Link>
                    {isActive && (
                      <span
                        onClick={() => handleRemoveFilter("tag")}
                        className="pointer"
                      >
                        X
                      </span>
                    )}
                  </div>
                );
              })}
          </div>
        </>
      )}

      <p className="mt-4 alert alert-primary">Brands</p>
      <div className="row d-flex align-items-center mx-1 filter-scroll">
        {brands?.map((b) => {
          const isActive = brand === b;

          const url = {
            pathname,
            query: {
              ...searchParams,
              brand: b,
              page: 1,
            },
          };
          return (
            <div key={b}>
              <Link href={url} className={isActive ? activeButton :
button}>
                {b}
              </Link>
              {isActive && (
                <span
                  onClick={() => handleRemoveFilter("brand")}
                  className="pointer"
                >
                  X
                </span>
              )}
            </div>
          );
        })}
      </div>

      {/* <pre>{JSON.stringify(tags, null, 4)}</pre> */}
    </div>
  );
}
```

Shop page layout with scrolling sidebar for filters

```
// app/shop/page
import ProductFilter from "@/components/product/ProductFilter";

async function getProducts(searchParams) {
  const searchQuery = new URLSearchParams({
    page: searchParams.page || 1,
    minPrice: searchParams.minPrice || "",
    maxPrice: searchParams.maxPrice || "",
    ratings: searchParams.ratings || "",
    category: searchParams.category || "",
    tag: searchParams.tag || "",
    brand: searchParams.brand || "",
  }).toString();
  //
}

export default async function Shop({ searchParams }) {
  console.log("searchParams in shop page => ", searchParams);
  const data = await getProducts(searchParams);

  return (
    <div className="container-fluid">
      <div className="row">
        <div className="col-lg-3 overflow-auto" style={{ maxHeight: "90vh"
}}>
          <ProductFilter searchParams={searchParams} />
        </div>
        <div className="col-lg-9">Products list</div>
      </div>
    </div>
  );
```

Filtering products API request

```
// shop page
async function getProducts(searchParams) {
  const searchQuery = new URLSearchParams({
    page: searchParams.page || 1,
    minPrice: searchParams.minPrice || "",
    maxPrice: searchParams.maxPrice || "",
    ratings: searchParams.ratings || "",
    category: searchParams.category || "",
    tag: searchParams.tag || "",
    brand: searchParams.brand || "",
  }).toString();

  try {
    const response = await fetch(
      `${process.env.API}/product/filters?${searchQuery}`,
      {
        method: "GET",
      }
    );
    if (!response.ok) {
      throw new Error("Failed to fetch products");
    }
    const data = await response.json();
    if (!data || !Array.isArray(data.products)) {
      throw new Error("No products returned");
    }

    return data;
  } catch (err) {
    console.log(err);
    return { products: [], currentPage: 1, totalPages: 1 };
  }
}
```

Filtered products API

```
// app/product/filters/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import queryString from "query-string";

export async function GET(req) {
  await dbConnect();
  // parse query params from the req url
  const searchParams = queryString.parseUrl(req.url).query;
  // destructure query searchParams
  const { page, category, brand, tag, ratings, minPrice, maxPrice } =
    searchParams || {};
  const pageSize = 6;
  // initialize an empty filter object
  const filter = {};

  // apply filters based on query params
  if (category) {
    filter.category = category;
  }
  if (brand) {
    filter.brand = brand;
  }
  if (tag) {
    filter.tags = tag;
  }
  if (minPrice && maxPrice) {
    filter.price = {
      $gte: minPrice,
      $lte: maxPrice,
    };
  }

  try {
    // determine the current page and calculate the skip value for
```

```
pagination
    const currentPage = Number(page) || 1;
    const skip = (currentPage - 1) * pageSize;
    // retrieve all products based on the applied filters
    const allProducts = await Product.find(filter)
      .populate("category", "name")
      .populate("tags", "name")
      .sort({ createdAt: -1 });

    // function to calculate the average rating for each product
    const calculateAverageRating = (ratings) => {
      if (ratings.length === 0) return 0;
      let totalRating = 0;
      ratings.forEach((rating) => {
        totalRating += rating.rating;
      });
      return totalRating / ratings.length;
    };

    // calculate the average rating for each product
    const productsWithAverageRating = allProducts.map((product) => ({
      ...product.toObject(),
      averageRating: calculateAverageRating(product.ratings),
    }));

    // filter products based on the ratings query param
    const filteredProducts = productsWithAverageRating.filter((product) =>
{
      if (!ratings) {
        return true; // no rating filter applied
      }

      const targetRating = Number(ratings);
      const difference = product.averageRating - targetRating;
      return difference >= -0.5 && difference <= 0.5; // (4) [3.5 to 4.5]
    });

    const totalFilteredProducts = filteredProducts.length;
    // apply pagination to filtered products
    const paginatedProducts = filteredProducts.slice(skip, skip +
pageSize);
    // return the paginated product data as json
    return NextResponse.json(
      {
        products: paginatedProducts,
        currentPage,
        totalPages: Math.ceil(totalFilteredProducts / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    console.log("filter products err => ", err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Product search (text based)

```
// context/product

// text search
const [productSearchQuery, setProductSearchQuery] = useState("");
const [productSearchResults, setProductSearchResults] = useState([]);

const fetchProductSearchResults = async (e) => {
  e.preventDefault();
  try {
    const response = await fetch(
      `${process.env.API}/search/products?
productSearchQuery=${productSearchQuery}`
    );
    if (!response.ok) {
      throw new Error("Network response was not ok");
    }
    const data = await response.json();
    setProductSearchResults(data);
    // console.log("search results => ", data);
    router.push(`/search/products?
productSearchQuery=${productSearchQuery}`);
  } catch (error) {
    console.error("Error fetching search results:", error);
  }
};

// TopNav
const { productSearchQuery, setProductSearchQuery,
fetchProductSearchResults } =
  useProduct();

<form
  className="d-flex mx-2"
  role="search"
  onSubmit={fetchProductSearchResults}
>
  <input
    className="form-control"
    type="search"
    placeholder="Search products"
    aria-label="Search"
    onChange={(e) => setProductSearchQuery(e.target.value)}
```

```
      value={productSearchQuery}
    />
    <button className="btn" type="submit" style={{ borderRadius: "20px" }}>
      &#128270;
    </button>
  </form>;

// api/search/products/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import Category from "@/models/category"; // Import the Category model
import Tag from "@/models/tag"; // Import the Tag model
import queryString from "query-string";

export async function GET(req) {
  await dbConnect();

  const { productSearchQuery } = queryString.parseUrl(req.url).query;

  try {
    // Search for categories and tags based on the productSearchQuery
    const [categories, tags] = await Promise.all([
      Category.find({ name: { $regex: productSearchQuery, $options: "i" }
}),
      Tag.find({ name: { $regex: productSearchQuery, $options: "i" } }),
    ]);

    const categoryIds = categories.map((category) => category._id);
    const tagIds = tags.map((tag) => tag._id);

    // Main product search query
    const products = await Product.find({
      $or: [
        { title: { $regex: productSearchQuery, $options: "i" } },
        { description: { $regex: productSearchQuery, $options: "i" } },
        { brand: { $regex: productSearchQuery, $options: "i" } },
        { category: { $in: categoryIds } }, // Search for products with
matching category IDs
        { tags: { $in: tagIds } }, // Search for products with matching
tag IDs
      ],
    })
      .populate("category", "name")
      .populate("tags", "name")
      .sort({ createdAt: -1 });

    return NextResponse.json(products);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}

// app/search/products/page
("use client");
import { useEffect } from "react";
import ProductList from "@/components/product/ProductList";
import { useSearchParams } from "next/navigation";
import { useProduct } from "@/context/product";

export default function ProductsSearchPage() {
  // context
  const {
    setProductSearchQuery,
    productSearchResults,
    setProductSearchResults,
  } = useProduct();
  // console.log("searchQuery in search page =====> ", searchQuery);

  const productSearchParams = useSearchParams();
  const query = productSearchParams.get("productSearchQuery");

  // to fetch results on page load based on query
  useEffect(() => {
    if (query) {
      console.log(
        "Got search params in search page => ",
        productSearchParams.get("productSearchQuery")
      );
      setProductSearchQuery(query);
      fetchProductResultsOnLoad(query);
    }
  }, [query]);

  const fetchProductResultsOnLoad = async () => {
    try {
      const response = await fetch(
        `${process.env.NEXT_PUBLIC_API}/search/products?
productSearchQuery=${query}`
      );

      if (!response.ok) {
        throw new Error("Network response was not ok");
      }

      const data = await response.json();
      setProductSearchResults(data);
    } catch (error) {
      console.error("Error fetching search results:", error);
    }
  };
  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p>Search result {productSearchResults?.length}</p>
          {/* <pre>{JSON.stringify(searchResults, null, 4)}</pre> */}
          {productSearchResults ? (
            <ProductList products={productSearchResults} />
          ) : (
            ""
          )}
        </div>
      </div>
    </div>
  );
}
```

Show Reviews Comments

```
// product/[slug]/page
<div className="row">
  <div className="col my-5">
    <UserReviews reviews={product?.ratings} />
  </div>
</div>

// components/product/UserReviews
import RatingDistribution from "@/components/product/RatingDistribution";
import Stars from "@/components/product/Stars";

export default function UserReviews({ reviews }) {
  return (
    <>
      {reviews?.length > 0 ? (
        <div>
          <RatingDistribution reviews={reviews} />

          {/* List of user reviews */}
          <ul className="list-group mt-4 bg-white">
            {reviews.map((review) => (
              <li
                className="list-group-item mb-1"
                key={review._id}
              >
                <div>
                  <p>
                    <strong>{review.postedBy.name}</strong>
                  </p>
                  <Stars rating={review.rating} />
                  {review?.comment && <p className="mt-3">{review.comment}
</p>}
```

```
                </div>
              </li>
            ))}
          </ul>
        </div>
      ) : (
        <p>No reviews yet.</p>
      )}
    </>
  );
}

// components/product/RatingDistribution
import { FaStar, FaRegStar } from "react-icons/fa";
import { calculateAverageRating } from "@/utils/helpers";
import Stars from "@/components/product/Stars";

export default function RatingDistribution({ reviews }) {
  // Calculate rating distribution and total number of reviews
  const distribution = {
    5: 0,
    4: 0,
    3: 0,
    2: 0,
    1: 0,
  };
  let totalReviews = 0;

  reviews.forEach((review) => {
    // distribution[index]: ++
    distribution[review.rating]++;
    totalReviews++;
  });

  // Calculate percentage and generate rating icons
  const ratingIcons = Object.keys(distribution).map((rating) => {
    const count = distribution[rating]; // how maany reviews
    let percentage = ((count / totalReviews) * 100).toFixed(2);
    percentage =
      parseFloat(percentage) === parseInt(percentage)
        ? parseInt(percentage)
        : percentage;

    const starIcons = Array.from({ length: parseInt(rating) }, (_, index)
=> (
      <FaStar key={index} className="text-danger" />
    ));
    const emptyStarIcons = Array.from(
      { length: 5 - parseInt(rating) },
      (_, index) => <FaRegStar key={index} />
    );

    return (
      <div
```

```
          key={rating}
          className="d-flex justify-content-between align-items-center"
        >
          <div className="progress col-6 p-0 m-0 mt-1" style={{ height:
"10px" }}>
            <div
              className="progress-bar bg-secondary"
              role="progressbar"
              style={{ width: `${percentage}%` }}
              aria-valuenow={percentage}
              aria-valuemin="0"
              aria-valuemax="100"
            ></div>
          </div>

          <div className="col-6">
            {starIcons} {emptyStarIcons} {percentage}%
          </div>
        </div>
      );
    });

    return (
      <div className="row">
        <div className="col-3 d-flex align-items-center">
          <div className="text-center">
            <p className="display-2 mb-0">
              <strong>{calculateAverageRating(reviews)?.toFixed(1)}</strong>
            </p>
            <Stars rating={calculateAverageRating(reviews)} />
            <p>Product Rating</p>
          </div>
        </div>
        <div className="col-9">{ratingIcons.reverse()}</div>
      </div>
    );
}
```

Products metadata

```
// products page
export const metadata = {
  title: "Next Ecommerce",
  description: "Find the latest in fashion, electronics and more",
};

// products single view page
export async function generateMetadata({ params }) {
  const product = await getProduct(params.slug);
  return {
    title: product?.title,
    description: product?.description?.substr(0, 160),
  };
}
```

Add to cart

```
// context/cart
import { createContext, useState, useContext, useEffect } from "react";

export const CartContext = createContext();

export const CartProvider = ({ children }) => {
  const [cartItems, setCartItems] = useState([]);

  // Load cart items from local storage on component mount
  useEffect(() => {
    const storedCartItems = JSON.parse(localStorage.getItem("cartItems"))
|| [];
    setCartItems(storedCartItems);
  }, []);

  // Save cart items to local storage whenever cartItems state changes
  useEffect(() => {
    localStorage.setItem("cartItems", JSON.stringify(cartItems));
  }, [cartItems]);

  const addToCart = (product, quantity) => {
    const existingProduct = cartItems.find((item) => item._id ===
product._id);

    if (existingProduct) {
      const updatedCartItems = cartItems.map((item) =>
        item._id === product._id
          ? { ...item, quantity: item.quantity + quantity }
          : item
      );
      setCartItems(updatedCartItems);
    } else {
      setCartItems([...cartItems, { ...product, quantity }]);
    }
  };

  const updateQuantity = (product, quantity) => {
    const updatedItems = cartItems.map((item) =>
      item._id === product._id ? { ...item, quantity } : item
    );
    setCartItems(updatedItems);
    localStorage.setItem("cartItems", JSON.stringify(updatedItems));
  };

  const removeFromCart = (productId) => {
    const updatedCartItems = cartItems.filter((item) => item._id !==
productId);
    setCartItems(updatedCartItems);

    // Update local storage
    if (typeof window !== "undefined") {
      localStorage.setItem("cart", JSON.stringify(updatedCartItems));
    }
  };

  // Provide cart items and functions to the rest of the app
  return (
    <CartContext.Provider
      value={{
        cartItems,
        addToCart,
        updateQuantity,
        removeFromCart,
      }}
    >
      {children}
    </CartContext.Provider>
  );
};

export const useCart = () => useContext(CartContext);


// components/products/AddToCart
"use client";
import { useState, useEffect } from "react";
import { useCart } from "@/context/cart";
import Link from "next/link";

export default function AddToCart({ product }) {
  const { addToCart, updateQuantity, cartItems, removeFromCart } =
useCart();

  // Find the product in cartItems, if it exists
  const existingProduct = cartItems.find((item) => item._id ===
product._id);
  const initialQuantity = existingProduct ? existingProduct.quantity : 1;

  const [quantity, setQuantity] = useState(initialQuantity);

  useEffect(() => {
    // Update quantity state if the product's quantity changes in
cartItems
    setQuantity(existingProduct ? existingProduct.quantity : 1);
  }, [existingProduct]);

  const handleIncrement = () => {
    const newQuantity = quantity + 1;
    setQuantity(newQuantity);
```

```
      updateQuantity(product, newQuantity);
    };

    const handleDecrement = () => {
      if (quantity > 1) {
        const newQuantity = quantity - 1;
        setQuantity(newQuantity);
        updateQuantity(product, newQuantity);
      } else {
        // If quantity becomes 0, remove the item from the cart
        removeFromCart(product._id);
        setQuantity(1); // Reset quantity to 1 after removing from cart
      }
    };

    const handleAddToCart = () => {
      addToCart(product, quantity);
    };

    return (
      <div>
        {cartItems.some((item) => item._id === product._id) ? (
          <>
            <div className="input-group quantity-input">
              <div className="input-group-prepend">
                <button
                  className="btn btn-outline-secondary"
                  type="button"
                  onClick={handleDecrement}
                >
                  -
                </button>
              </div>
              <input
                type="number"
                className="form-control no-spin-arrows mx-5 text-center"
                value={quantity}
                onChange={(e) => setQuantity(parseInt(e.target.value, 10))}
              />
              <div className="input-group-append">
                <button
                  className="btn btn-outline-secondary"
                  type="button"
                  onClick={handleIncrement}
                >
                  +
                </button>
              </div>
            </div>

            <Link
              className="btn btn-outline-danger btn-raised btn-block mt-2"
              href="/cart"
            >
```

```
              Review & Checkout
            </Link>
          </>
        ) : (
          <button
            className="btn btn-danger btn-raised btn-block"
            onClick={handleAddToCart}
          >
            Add to Cart
          </button>
        )}
      </div>
    );
  }


  // TopNav
  <Link className="nav-link text-danger" href="/cart">
    <BsFillCartCheckFill size={25} /> {cartItems?.length}
  </Link>

  ## 3 step checkout process
  // cart page
  "use client";
  import { useState } from "react";

  export default function Cart() {
    const [step, setStep] = useState(1);

    const handleNextStep = () => {
      setStep(step + 1);
    };

    const handlePrevStep = () => {
      setStep(step - 1);
    };

    return (
      <div>
        {step === 1 && <Step1 onNextStep={handleNextStep} />}
        {step === 2 && (
          <Step2 onNextStep={handleNextStep} onPrevStep={handlePrevStep} />
        )}
        {step === 3 && <Step3 onPrevStep={handlePrevStep} />}
      </div>
    );
  }

  function Step1({ onNextStep }) {
    return (
      <div>
        Review cart
        <button onClick={onNextStep}>Next</button>
      </div>
    );
  }

  function Step2({ onNextStep, onPrevStep }) {
    return (
      <div>
        Contact details
        <button onClick={onPrevStep}>Previous</button>
        <button onClick={onNextStep}>Next</button>
      </div>
    );
  }

  function Step3({ onPrevStep }) {
    return (
      <div>
        Payment
        <button onClick={onPrevStep}>Previous</button>
        <button>Place Order</button>
      </div>
    );
  }
```

Cart page

```
// app/cart/page
"use client";
import { useState } from "react";
import Link from "next/link";
import { GoCheckCircleFill } from "react-icons/go";
import Step1 from "@/components/product/cart/Step1";
import Step2 from "@/components/product/cart/Step2";
import Step3 from "@/components/product/cart/Step3";
import { useCart } from "@/context/cart";

export default function Cart() {
  // context
  const { cartItems } = useCart();

  // state
  const [step, setStep] = useState(1);

  const handleNextStep = () => {
    setStep(step + 1);
  };

  const handlePrevStep = () => {
    setStep(step - 1);
  };

  const tickIcon = (stepNumber) => {
    return step === stepNumber ? (
      <GoCheckCircleFill className="mb-1 text-danger" />
    ) : null;
  };

  if (!cartItems?.length)
    return (
      <div className="container d-flex justify-content-center align-items-
center vh-100">
        <div className="text-center">
          <p className="lead">Your cart is empty!</p>
          <Link className="btn btn-lg btn-primary btn-raised"
href="/products">
            Continue Shopping
          </Link>
        </div>
      </div>
    );

  return (
    <div>
      <div className="col-lg-6 offset-lg-3 my-5">
        <div className="d-flex justify-content-between lead">
          <div>{tickIcon(1)} Review Cart</div>
          <div>{tickIcon(2)} Contact Details</div>
          <div>{tickIcon(3)} Payment</div>
        </div>
      </div>

      {step === 1 && <Step1 onNextStep={handleNextStep} />}
      {step === 2 && (
        <Step2 onNextStep={handleNextStep} onPrevStep={handlePrevStep} />
      )}
      {step === 3 && <Step3 onPrevStep={handlePrevStep} />}
    </div>
  );
}

## Cart checkout step 1 orders review

// components/product/cart/Step1
import { useCart } from "@/context/cart";
import Image from "next/image";
import Link from "next/link";
import AddToCart from "@/components/product/AddToCart";
import OrderSummary from "@/components/product/cart/OrderSummary";

export default function Step1({ onNextStep }) {
  const { cartItems } = useCart();

  return (
    <div className="container">
      <div className="row">
        <div className="col-lg-8">
```

```
          <p className="alert alert-primary">Review Cart / Adjust
Quantity</p>

          {cartItems?.map((product) => (
            <div className="card mb-3" key={product._id}>
              <div className="row g-0">
                <div className="col-md-4">
                  <div style={{ height: "200px", overflow: "hidden" }}>
                    <Image
                      src={
                        product?.images?.[0]?.secure_url ||
                        "/images/new-wave.jpeg"
                      }
                      className="card-img-top"
                      width={500}
                      height={300}
                      style={{
                        objectFit: "cover",
                        height: "100%",
                        width: "100%",
                      }}
                      alt={product?.title}
                    />
                  </div>
                </div>
                <div className="col-md-8">
                  <div className="card-body">
                    <h5 className="card-title">
                      <Link
                        href={`/product/${product?.slug}`}
                        as={`/product/${product?.slug}`}
                      >
                        {product.title} [{product?.images?.length} 📸 ]
                      </Link>
                    </h5>
                    <h4>${product?.price.toFixed(2)}</h4>
                    <div className="card-text">
                      <div
                        dangerouslySetInnerHTML={{
                          __html:
                            product?.description?.length > 160
                              ? `${product?.description.substring(0,
160)}...`
                              : product?.description,
                        }}
                      />
                    </div>

                    <div className="mt-3">
                      <AddToCart product={product} reviewAndCheckout=
{false} />
                    </div>
                  </div>
                </div>
              </div>
            </div>
          ))}

          <div className="d-flex justify-content-end my-4">
            <button
              className="btn btn-danger btn-raised col-6"
              onClick={onNextStep}
            >
              Next
            </button>
          </div>
        </div>

        <div className="col-lg-4">
          <OrderSummary />
        </div>
      </div>
    </div>
  );
}

## Cart checkout step 2 user info

// components/product/cart/Step2
import { useSession } from "next-auth/react";
import Link from "next/link";
import { useState } from "react";
import toast from "react-hot-toast";
import OrderSummary from "@/components/product/cart/OrderSummary";

// SKIP DELIVERY ADDRESS PART
// USE STRIPE CHECKOUT TO GRAB USER DELIVERY ADDRESS

export default function Step2({ onNextStep, onPrevStep }) {
  const { data, status, update } = useSession();
  // state
  const [deliveryAddress, setDeliveryAddress] = useState(
    data?.user?.deliveryAddress || ""
  );

  // update or confirm delivery address on next click
  const handleAddressThenNext = async () => {
    // update delivery address
    try {
      const response = await fetch(`${process.env.API}/user/profile`, {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ deliveryAddress }),
      });

      if (!response.ok) {
        const data = await response.json();
        toast.error(data.err);
        return;
      } else {
        const data = await response.json();
        // console.log("address updated, update user session", data);
        update({ user: { ...data.user, deliveryAddress: data } });
        // take to next step
        onNextStep();
      }
    } catch (err) {
      console.log(err);
      setLoading(false);
      toast.error("An error occurred. Please try again.");
    }
  };

  if (status !== "authenticated") {
    return (
      <div className="container">
        <div className="row">
          <div className="col-lg-8 offset-lg-2">
            <div className="d-flex justify-content-end my-4">
              <button
                className="btn btn-outline-danger btn-raised col-6"
                onClick={onPrevStep}
              >
                Previous
              </button>

              <Link
                className="btn btn-primary btn-raised col-6"
                href={`/login?callbackUrl=${window.location.href}`}
              >
                Login to Continue
              </Link>
            </div>
          </div>
        </div>
      </div>
    );
  }

  return (
    <div className="container">
      <div className="row">
        <div className="col-lg-8">
          <p className="alert alert-primary">Contact Details / Login</p>

          <div>
            <input
              type="text"
              value={data?.user?.name}
              className="form-control mb-2 px-2"
              placeholder="Your name"
              disabled
            />
            <input
              type="email"
              value={data?.user?.email}
              className="form-control mb-2 px-2"
              placeholder="Your email"
              disabled
            />

            {/* delivery address */}
            <textarea
              maxLength="320"
              value={deliveryAddress}
              onChange={(e) => setDeliveryAddress(e.target.value)}
              className="form-control mb-2 px-2 mt-4"
              placeholder="Enter your delivery address"
              rows="5"
            />

            {/* <pre>{JSON.stringify(data, null, 4)}</pre> */}
          </div>

          <div className="d-flex justify-content-end my-4">
            <button
              className="btn btn-outline-danger btn-raised col-6"
              onClick={onPrevStep}
            >
              Previous
            </button>

            <button
              className="btn btn-danger btn-raised col-6"
              onClick={handleAddressThenNext}
              disabled={!deliveryAddress.trim()}
            >
             Next
            </button>
          </div>
        </div>

        <div className="col-lg-4">
          <OrderSummary />
        </div>
      </div>
    </div>
  );
}

## Cart checkout step 3 stripe payment system

// components/product/cart/Step3
import { useState } from "react";
import { useCart } from "@/context/cart";
import OrderSummary from "@/components/product/cart/OrderSummary";
import toast from "react-hot-toast";

export default function Step3({ onPrevStep }) {
  const { cartItems } = useCart();
  // state
  const [loading, setLoading] = useState(false);

  const handleClick = async () => {
    try {
      setLoading(true);

      const cartData = cartItems.map((item) => ({
        _id: item._id,
        quantity: item.quantity,
      }));

      const response = await
fetch(`${process.env.API}/user/stripe/session`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          cartItems: cartData,
        }),
      });

      if (response.ok) {
        const data = await response.json();
        // console.log("checkout session response data", data);
        window.location.href = data.url;
      } else {
        const errorData = await response.json();
        toast.error(errorData.err);
        setLoading(false);
      }
    } catch (err) {
      console.log(err);
      toast.error("An error occurred. Please try again.");
      setLoading(false);
    }
  };

  return (
    <div className="container">
      <div className="row">
        <div className="col-lg-8">
          <p className="alert alert-primary">Payment Method</p>

          <h2 className="text-center">🔒 💳</h2>

          <p className="alert alert-danger">
```

```
            Flat rate $5 shipping fee will apply for all orders Australia
wide!
          </p>

          <p className="lead card p-5 bg-secondary text-light">
            Clicking 'Place Order' will securely redirect you to our
trusted
            payment partner, Stripe to complete your checkout. Your
payment
            information is fully protected and encrypted for your
security.
          </p>

          <div className="d-flex justify-content-end my-4">
            <button
              className="btn btn-outline-danger btn-raised col-6"
              onClick={onPrevStep}
            >
              Previous
            </button>

            {/* trigger stripe payment on this button click */}
            <button
              className="btn btn-success btn-raised col-6"
              onClick={handleClick}
              disabled={loading}
            >
              {loading ? "Processing ..." : "Place Order"}
            </button>
          </div>
        </div>

        <div className="col-lg-4">
          <OrderSummary />
        </div>
      </div>
    </div>
  );
}

## Order summary component

// components/product/cart/OrderSummary
import React from "react";
import { useCart } from "@/context/cart";
import Image from "next/image";

export default function OrderSummary() {
  const { cartItems } = useCart();

  const calculateTotal = () => {
    return cartItems.reduce(
      (total, item) => total + item.price * item.quantity,
      0
    );
  };
  const totalItems = cartItems.reduce(
    (total, item) => total + item.quantity,
    0
  );
  const itemOrItems = totalItems === 1 ? "item" : "items";

  return (
    <div>
      <p className="alert alert-primary">Order Summary</p>
      <ul className="list-unstyled">
        {cartItems?.map((product) => (
          <div className="card mb-3" key={product._id}>
            <div className="row g-0 d-flex align-items-center p-1">
              <div className="col-md-3">
                <div style={{ height: "66px", overflow: "hidden" }}>
                  <Image
                    src={
                      product?.images?.[0]?.secure_url ||
                      "/images/new-wave.jpeg"
                    }
                    className="card-img-top"
                    width={500}
                    height={300}
                    style={{
                      objectFit: "cover",
                      height: "100%",
                      width: "100%",
                    }}
                    alt={product?.title}
                  />
                </div>
              </div>
              <div className="col-md-6">
                <p className="card-title text-secondary">{product.title}
</p>
              </div>
              <div className="col-md-3">
                <p className="h6">${product?.price.toFixed(2)}</p>
                <p className="text-secondary">Qty: {product?.quantity}</p>
              </div>
            </div>
          </div>
        ))}
      </ul>
      <div className="d-flex justify-content-between p-1">
        <p>
          Total {totalItems} {itemOrItems}:
        </p>
        <p className="h4">${calculateTotal().toFixed(2)}</p>
      </div>
    </div>
  );
```

```
}

## User profile udpate with address (optional)

// Optional!
// update user profile/delivery address
// api/user/profile/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import User from "@/models/user";
import { getToken } from "next-auth/jwt";

export async function PUT(req) {
  await dbConnect();

  const _req = await req.json();

  const { deliveryAddress } = _req;
  const token = await getToken({
    req,
    secret: process.env.NEXTAUTH_SECRET,
  });

  try {
    const updated = await User.findByIdAndUpdate(
      token.user._id,
      { deliveryAddress },
      { new: true }
    );

    return NextResponse.json(updated);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}

## Create stripe checkout session with tax rates, shipping cost

// several updates need to adjust shipping, tax and coupons
// change based on product _id quantity and auto tax
// api/user/stripe/session/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import { getToken } from "next-auth/jwt";
import Product from "@/models/product";

const stripe = require("stripe")(process.env.STRIPE_SECRET_KEY);
```

```
// create checkout session
// https://stripe.com/docs/api/checkout/sessions/create?lang=node

export async function POST(req) {
  await dbConnect();
  const _req = await req.json();
  console.log("_req in stripe checkout session api", _req);

  const token = await getToken({
    req,
    secret: process.env.NEXTAUTH_SECRET,
  });

  try {
    const lineItems = await Promise.all(
      _req.cartItems.map(async (item) => {
        const product = await Product.findById(item._id); // Fetch product
details from the database
        const unitAmount = product.price * 100; // Stripe expects the
amount in cents
        return {
          price_data: {
            currency: "aud",
            product_data: {
              name: product.title,
              images: [product.images[0].secure_url],
            },
            unit_amount: unitAmount,
          },
          tax_rates: [process.env.STRIPE_TAX_RATE],
          quantity: item.quantity,
        };
      })
    );

    const session = await stripe.checkout.sessions.create({
      success_url: `${process.env.DOMAIN}/dashboard/user/stripe/success`,
      client_reference_id: token?.user?._id,
      line_items: lineItems,
      mode: "payment",
      // https://stripe.com/docs/api/payment_methods/create
      payment_method_types: ["card"],
      // search tax in dashboard under "Pricing catalog"
      // https://dashboard.stripe.com/test/settings/tax
      payment_intent_data: {
        metadata: {
          cartItems: JSON.stringify(_req.cartItems), // Store cart items
as metadata
          userId: token?.user?._id,
        },
      },
      shipping_options: [
        {
          shipping_rate: process.env.STRIPE_SHIPPING_RATE,
        },
      ],
      shipping_address_collection: {
        allowed_countries: ["AU"], // Only allow shipping to Australia
      },
      // fR6Qwywx
      discounts: [
        {
          coupon: _req.couponCode, // Replace with your coupon code
        },
      ],
      customer_email: token.user.email, // pre-populate customer email in
checkout page
    });

    return NextResponse.json(session);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Create Order with Stripe Webhook

Log in to your Stripe account. Go to "Developers" > "Webhooks" in the left sidebar.

```
stripe login
stripe listen --forward-to localhost:3000/api/webhook
use the webhook secret in your code
```

Now try checkout, keep an eye on terminal

https://stripe.com/docs/payments/checkout/fulfill-orders

```
// api/webhook/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Order from "@/models/order";
import Product from "@/models/product";

// https://github.com/shadcn-ui/taxonomy
```

```
const stripe = require("stripe")(process.env.STRIPE_SECRET_KEY);

export async function POST(req) {
  await dbConnect();

  const _raw = await req.text();
  const sig = req.headers.get("stripe-signature");

  try {
    // Construct the event using the Stripe SDK
    const event = stripe.webhooks.constructEvent(
      _raw,
      sig,
      process.env.STRIPE_WEBHOOK_SECRET
    );
    // console.log("event => ", event);

    // Handle the event
    switch (event.type) {
      case "charge.succeeded":
        const chargeSucceeded = event.data.object;
        // console.log("chargeSucceeded => ", chargeSucceeded);

        const { id, ...rest } = chargeSucceeded;

        // decrement stock and gather product IDs
        const cartItems = JSON.parse(chargeSucceeded.metadata.cartItems);
        const productIds = cartItems.map((cartItem) => cartItem._id);

        // Fetch all products in one query
        const products = await Product.find({ _id: { $in: productIds } });

        // Create an object to quickly map product details by ID
        const productMap = {};
        products.forEach((product) => {
          productMap[product._id.toString()] = {
            _id: product._id,
            title: product.title,
            slug: product.slug,
            price: product.price,
            image: product.images[0]?.secure_url || "",
          };
        });

        // Create cartItems with product details
        const cartItemsWithProductDetails = cartItems.map((cartItem) => ({
          ...productMap[cartItem._id],
          quantity: cartItem.quantity,
        }));

        // Create order
        const orderData = {
          ...rest,
          chargeId: id,
          userId: chargeSucceeded.metadata.userId,
          cartItems: cartItemsWithProductDetails,
        };
        await Order.create(orderData);

        // Decrement product stock
        for (const cartItem of cartItems) {
          const product = await Product.findById(cartItem._id);
          if (product) {
            product.stock -= cartItem.quantity;
            await product.save();
          }
        }

        return NextResponse.json({ ok: true });
    }
  } catch (err) {
    console.log("====================================> ", err);
    return NextResponse.json(`Webhook Error: ${err.message}`, { status:
400 });
  }
}

//   const {
//     id, // chargeId
//     payment_intent,
//     receipt_url,
//     refunded,
//     status,
//     amount_captured,
//     currency,
//     shipping,
//   } = event.data.object;
```

Webhooks issues fix

```
// how I fixed?
// import order model
// add type string to order model status
// change 'event.type' listening to 'charge.succeeded'
// on webhook create in stripe, choose 'charges'
```

Order model

```
// models/order
import mongoose from "mongoose";

const cartItemSchema = new mongoose.Schema({
  product: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Product", // Reference to the Product model
  },
  title: String, // Add fields you need from the product
  slug: String,
  price: Number,
  image: String,
  quantity: Number,
});

const orderSchema = new mongoose.Schema({
  chargeId: String,
  payment_intent: String,
  receipt_url: String,
  refunded: Boolean,
  status: String,
  amount_captured: Number,
  currency: String,
  shipping: {
    address: {
      city: String,
      country: String,
      line1: String,
      line2: String,
      postal_code: String,
      state: String,
    },
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
  },
  cartItems: [cartItemSchema],
  delivery_status: {
    type: String,
    default: "Not Processed",
    enum: [
      "Not Processed",
      "processing",
      "Dispatched",
      "Refunded",
      "Cancelled",
      "Delivered",
    ],
  },
});

export default mongoose.models.Order || mongoose.model("Order",
orderSchema);
```

Stripe Coupon Discounts on checkout

```
// context/cart
const [couponCode, setCouponCode] = useState("");
const [percentOff, setPercentOff] = useState(0);
const [validCoupon, setValidCoupon] = useState(false);

const handleCoupon = async (coupon) => {
  // apply coupon
  try {
    const response = await fetch(`${process.env.API}/stripe/coupon`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ couponCode: coupon }),
    });

    if (!response.ok) {
      // const data = await response.json();
      // toast.error("Invalid coupon code");
      setPercentOff(0);
      setValidCoupon(false);
      return;
    } else {
      const data = await response.json();
      setPercentOff(data.percent_off);
      setValidCoupon(true);
      console.log("coupon code applied => ", data);
      toast.success(`${data?.name} applied successfully`); //
data.percent_off
      // if (cartItems?.length > 0) {
      //   toast.success(`${data?.name} applied successfully`); //
data.percent_off
      // }
    }
  } catch (err) {
    console.log(err);
    setPercentOff(0);
    setValidCoupon(false);
    toast.error("An error occurred. Please try again.");
  }
};

// components/product/cart/Step2
const { couponCode, setCouponCode, handleCoupon } = useCart();

<input
    type="text"
    value={couponCode}
    onChange={(e) => setCouponCode(e.target.value)}
    className="form-control mb-2 px-2 mt-4"
    placeholder="Enter your coupon code here"
  />
```

```
<button
  className="btn btn-success btn-raised"
  onClick={() => handleCoupon(couponCode)}
  disabled={!couponCode.trim()}
>
  Apply Coupon
</button>;
```

Stripe coupon API

```
// api/stripe/coupon/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";

const stripe = require("stripe")(process.env.STRIPE_SECRET_KEY);

export async function POST(req) {
  await dbConnect();
  const _req = await req.json();
  // console.log("_req in stripe checkout session api", _req);

  try {
    const coupon = await stripe.coupons.retrieve(_req.couponCode);
    console.log("coupon", coupon);
    return NextResponse.json(coupon, { status: 200 });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Discount coupon code embeded links for products

```
// app/product/[slug]/page
// move price display to <CouponCode />
<CouponCode product={product} />;

// components/product/CouponCode
("use client");
import { useEffect } from "react";
import { useCart } from "@/context/cart";
import { useSearchParams } from "next/navigation";

export default function CouponCode({ product }) {
  const { handleCoupon, setCouponCode, percentOff, validCoupon } =
useCart();

  const searchParams = useSearchParams();

  const code = searchParams.get("couponCode");

  // console.log("search params coupon => ", searchParams);
  useEffect(() => {
    if (code) {
      setCouponCode(code);
      handleCoupon(code);
    }
  }, [code]);

  return (
    <div className="d-flex justify-content-between align-items-center">
      {validCoupon ? (
        <del>
          <h4 className="text-danger">${product?.price?.toFixed(2)}</h4>
        </del>
      ) : (
        <h4>${product?.price?.toFixed(2)}</h4>
      )}
      {percentOff > 0 && (
        <h4 className="alert alert-danger">
          🔥 ${((product.price * (100 - percentOff)) / 100).toFixed(2)} (
          {percentOff}% discount coupon applied)
        </h4>
      )}

      {product?.previousPrice > product?.price && (
        <h4 className="text-danger">
          <del>${product?.previousPrice?.toFixed(2)}</del>
        </h4>
      )}
    </div>
  );
}

// components/product/cart/Step3
// send couponCode only if it's valid
const handleClick = async () => {
  // console.log("couponCode => ", couponCode, "validCoupon => ",
validCoupon);
  // return;
  try {
    setLoading(true);

    let payload = {};

    const cartData = cartItems.map((item) => ({
      _id: item._id,
      quantity: item.quantity,
    }));

    payload.cartItems = cartData;
    if (validCoupon) {
      payload.couponCode = couponCode;
    }

    const response = await fetch(`${process.env.API}/user/stripe/session`,
{
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },

      body: JSON.stringify(payload),
    });

    if (response.ok) {
      const data = await response.json();
      // console.log("checkout session response data", data);
      window.location.href = data.url;
    } else {
      const errorData = await response.json();
      toast.error(errorData.err);
      setLoading(false);
    }
  } catch (err) {
    console.log(err);
    toast.error("An error occurred. Please try again.");
    setLoading(false);
  }
};
```

On Sale price (previous price)

```
// models/product
previousPrice: Number,

// components/product/admin/ProductCreate
{updatingProduct && (
  <div className="form-group">
    <input
      type="number"
      placeholder="Previous Price"
      min="1"
      name="previousPrice"
      className="form-control p-2 my-2"
      value={updatingProduct?.previousPrice}
      onChange={(e) => {
        setUpdatingProduct({
          ...updatingProduct,
          previousPrice: e.target.value,
        });
      }}
    />
  </div>
)}

// ProductCard
// app/product/[slug]/page
<div className="d-flex justify-content-between">
  <h4>${product?.price?.toFixed(2)}</h4>
  {product?.previousPrice > product?.price && (
    <h4 className="text-danger">
      <del>${product?.previousPrice?.toFixed(2)}</del>
    </h4>
  )}
</div>
```

Orders for user

```
// api/user/orders/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Order from "@/models/order";
import { currentUser } from "@/utils/currentUser";

export async function GET(req) {
  await dbConnect();

  try {
    const user = await currentUser(); // Get the current user
asynchronously

    const orders = await Order.find({ userId: user._id }).sort({
      createdAt: -1,
    });

    return NextResponse.json(orders);
  } catch (err) {
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}
```

Stripe success page

```
// app/dashboard/user/stripe/success/page
import Link from "next/link";

export default function UserStripeSuccess() {
  return (
    <div className="container">
      <div className="row">
        <div className="col text-center">
          <p>
            Thank your for your purchase. You can now check your order
status in
            the dashboard
          </p>
          <hr />
          <Link
            className="btn btn-primary btn-raised"
            href="/dashboard/user/orders"
          >
            View Order Status
          </Link>
        </div>
      </div>
    </div>
  );
}
```

```
// app/dashboard/user/orders/page
// order info, total paid, receipt, cancle the order
"use client";
import { useEffect, useState } from "react";
import toast from "react-hot-toast";
import { useRouter } from "next/navigation";

export default function UserOrders() {
const [orders, setOrders] = useState([]);

const router = useRouter();

useEffect(() => {
fetchOrders();
}, []);

const fetchOrders = async () => {
try {
const response = await fetch(`${process.env.API}/user/orders`, {
method: "GET",
});
const data = await response.json();
setOrders(data);
} catch (error) {
console.log(error);
```

```
toast.error(error);
}
};

const handleCancelOrder = async (orderId) => {
try {
const response = await fetch(
`/api/user/orders/refund?orderId=${orderId}`,
{
method: "POST",
}
);
const data = await response.json();

      fetchOrders();
      // router.refresh();
    } catch (error) {
      console.log(error);
    }

};

return (
<div className="container mb-5">
<div className="row">
<div className="col">
<h4 className="text-center">Recent Orders</h4>

          {orders?.map((order) => (
            <div key={order?._id} className="mb-4 p-4 alert alert-
secondary">
              <table className="table table-striped">
                <tbody>
                  {/* order info */}
                  <tr>
                    <th scope="row">Charge ID:</th>
                    <td>{order?.chargeId}</td>
                  </tr>
                  <tr>
                    <th scope="row">Created:</th>
                    <td>{new Date(order?.createdAt).toLocaleDateString()}
</td>
                  </tr>
                  <tr>
                    <th scope="row">Payment Intent:</th>
                    <td>{order?.payment_intent}</td>
                  </tr>
                  <tr>
                    <th scope="row">Receipt:</th>
                    <td>
                      <a href={order?.receipt_url} target="_blank">
                        View
                      </a>
                    </td>
                  </tr>
                  <tr>
                    <th scope="row">Refunded:</th>
                    <td>{order?.refunded ? "Yes" : "No"}</td>
                  </tr>
                  <tr>
                    <th scope="row">Status:</th>
                    <td>{order?.status}</td>
                  </tr>
                  <tr>
                    <th scope="row">Total Charged:</th>
                    <td>
                      ${(order?.amount_captured / 100)?.toFixed(2)}{" "}
                      {order?.currency}
                    </td>
                  </tr>
                  <tr>
                    <th scope="row">Shipping Address:</th>
                    <td>
                      {order?.shipping?.address?.line1}
                      <br />
                      {order?.shipping?.address?.line2 &&
                        `${order?.shipping?.address?.line2}, `}
                      {order?.shipping?.address?.city},{" "}
                      {order?.shipping?.address?.state},{" "}
                      {order?.shipping?.address?.postal_code}
                      <br />
                      {order?.shipping?.address?.country}
                    </td>
                  </tr>
                  {/* products info */}
                  <tr>
                    <th scope="row" className="w-25">
                      Ordered Products:
                    </th>
                    <td className="w-75">
                      {order?.cartItems?.map((product) => (
                        <div
                          className="pointer text-primary"
                          key={product._id}
                          onClick={() =>
                            router.push(`/product/${product?.slug}`)
                          }
                        >
                          {product?.quantity} x {product?.title} $
                          {product?.price?.toFixed(2)} {order?.currency}
                        </div>
                      ))}
                    </td>
                  </tr>
                  <tr>
                    <th scope="row">Delivery Status:</th>
                    <td>
                      {order?.delivery_status}
                      {order?.delivery_status === "Not Processed" &&
                        !order.refunded && (
                          <>
                            <br />
                            <span
                              className="text-danger pointer"
                              onClick={() =>
handleCancelOrder(order?._id)}
                            >
                              Cancel the order
                            </span>
                          </>
                        )}
                    </td>
                  </tr>
                </tbody>
              </table>
            </div>
          ))}
        </div>
      </div>
    </div>

);
}
```

## Cart item component (optional)

// optional code refactoring
// components/cart/CartItem
// use this component in <Step1 /> component
import Image from "next/image";
import Link from "next/link";
import AddToCart from "@/components/product/AddToCart";

export default function CartItem({ product, addToCart = true, quantity })
{
return (
<>

<div className="card mb-3" key={product._id}>
<div className="row g-0">
<div className="col-md-4">
<div style={{ height: "200px", overflow: "hidden" }}>
<Image
src={
product?.images?.[0]?.secure_url || "/images/new-wave.jpeg"
}
className="card-img-top"
width={500}
height={300}
style={{
                  objectFit: "cover",
```
```

```
                  height: "100%",
                  width: "100%",
                }}
alt={product?.title}
/>
</div>
</div>
<div className="col-md-8">
<div className="card-body">
<h5 className="card-title">
<Link
href={`/product/${product?.slug}`}
as={`/product/${product?.slug}`} >
{product.title} {!addToCart && quantity && `x ${quantity}`}
</Link>
</h5>
<h4>${product?.price.toFixed(2)}</h4>
              <div className="card-text">
                <div
                  dangerouslySetInnerHTML={{
                    __html:
                      product?.description?.length > 160
                        ? `${product?.description.substring(0, 160)}...`
: product?.description,
}}
/>
</div>

              {addToCart && (
                <div className="mt-3">
                  <AddToCart product={product} reviewAndCheckout={false}
/>
                </div>
              )}
            </div>
          </div>
        </div>
      </div>
    </>

);
}
```

Stripe success and removal of products from cart

```
// cart context
const clearCart = () => {
  localStorage.removeItem("cartItems");
  setCartItems([]);
};
```

```
// stripe success page
// dashboard/user/stripe/success/page
("use client");
import { useEffect } from "react";
import Link from "next/link";
import { useCart } from "@/context/cart";

export default function UserStripeSuccess() {
  const { clearCart } = useCart();

  useEffect(() => {
    clearCart();
  }, []);

  return (
    <div className="container">
      <div className="row">
        <div className="col text-center">
          <p>
            Thank your for your purchase. You can now check your order
status in
            the dashboard
          </p>
          <hr />
          <Link
            className="btn btn-primary btn-raised"
            href="/dashboard/user/orders"
          >
            View Order Status
          </Link>
        </div>
      </div>
    </div>
  );
}
```

When order is created decrement stock

```
// webhook/route

// show low stock or out of stock
// utils/helpers
export const stockStatus = (stock) => {
  if (stock === 0) {
    return "Out of Stock";
  } else if (stock <= 10) {
    return "Low Stock";
  }
  return null;
};
```

```
// use in ProductCard and single product view
import { stockStatus } from "@/utils/helpers";

<div className="bg-warning text-center">{stockStatus(product?.stock)}
</div>;
```

User order refund/cancle API

```
// if the order is still "Not Processed"
// also increment refunded products stock
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Order from "@/models/order";
import Product from "@/models/product";
import { currentUser } from "@/utils/currentUser";
import queryString from "query-string";

const stripe = require("stripe")(process.env.STRIPE_SECRET_KEY);

export async function POST(req, res) {
  await dbConnect();

  try {
    const user = await currentUser(); // Get the current user
asynchronously

    // Get the order to refund
    const { orderId } = queryString.parseUrl(req.url).query;
    const order = await Order.findById(orderId);

    // Check if the order exists and belongs to the current user
    if (!order || order.userId.toString() !== user._id.toString()) {
      return NextResponse.json(
        { error: "Order not found or unauthorized" },
        { status: 404 }
      );
    }

    // Check if the order is still "Not Processed"
    if (order.delivery_status !== "Not Processed") {
      return NextResponse.json(
        { error: "Order cannot be refunded" },
        { status: 400 }
      );
    }

    // Make the refund request to Stripe
    const refund = await stripe.refunds.create({
      payment_intent: order.payment_intent, // Use the payment intent ID
from your order
```

```
        reason: "requested_by_customer",
      });

      // Update the product quantities based on the refunded items
      for (const cartItem of order.cartItems) {
        const product = await Product.findById(cartItem._id);

        if (product) {
          product.stock += cartItem.quantity;
          await product.save();
        }
      }

      // Update the order in the database with refund details
      order.status = "Refunded";
      order.refunded = true;
      order.delivery_status = "Cancelled";
      order.refundId = refund.id; // Store the refund ID for reference
      await order.save();

      return NextResponse.json(
        { message: "Order refunded successfully" },
        { status: 200 }
      );
    } catch (err) {
      return NextResponse.json(
        {
          err: err.message,
        },
        { status: 500 }
      );
    }
  }
```

Orders for admin

```
// api/admin/orders/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Order from "@/models/order";
import queryString from "query-string";

export async function GET(req) {
  await dbConnect();

  // req.nextUrl.searchParams.get('page')
  const searchParams = queryString.parseUrl(req.url).query;
  console.log("searchParams in admin orders => ", searchParams.page);

  const { page } = searchParams || {};
  const pageSize = 3;
  try {
    const currentPage = Number(page) || 1;
    const skip = (currentPage - 1) * pageSize;
    const totalOrders = await Order.countDocuments({});

    const orders = await Order.find({})
      .populate("userId", "name")
      .skip(skip)
      .limit(pageSize)
      .sort({
        createdAt: -1,
      });

    return NextResponse.json(
      {
        orders,
        currentPage,
        totalPages: Math.ceil(totalOrders / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}

// api/admin/orders/[orderId]/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Order from "@/models/order";

export async function PUT(req, context) {
  await dbConnect();
  const body = await req.json();

  try {
    const order = await Order.findByIdAndUpdate(
      context.params.orderId,
      {
        delivery_status: body.delivery_status,
      },
      { new: true }
    );
    return NextResponse.json(order);
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}

## Admin orders with pagination

// dashboard/admin/orders/page
// with pagination
("use client");
import { useEffect, useState } from "react";
import toast from "react-hot-toast";
import { usePathname, useSearchParams } from "next/navigation";
import Pagination from "@/components/Pagination";

export default function AdminOrders() {
  const [orders, setOrders] = useState([]);
  // pagination
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);

  const pathname = usePathname();
  const searchParams = useSearchParams();
  const page = searchParams.get("page");
  console.log("current page => ", page);

  useEffect(() => {
    fetchOrders(page);
  }, [page]);

  const fetchOrders = async (page) => {
    try {
      const response = await fetch(
        `${process.env.API}/admin/orders?page=${page}`,
        {
          method: "GET",
        }
      );
      const data = await response.json();
      // console.log("DATA in admin orders with pagination => ", data);
      setOrders(data.orders);
      setCurrentPage(data.currentPage);
      setTotalPages(data.totalPages);
    } catch (error) {
      console.log(error);
      toast.error(error);
    }
  };

  const handleStatusChange = async (newStatus, orderId) => {
    try {
      const response = await fetch(
        `${process.env.API}/admin/orders/${orderId}`,
        {
          method: "PUT",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({ delivery_status: newStatus }),
        }
      );

      if (response.ok) {
        // Update the order's status locally if the request was successful
        setOrders((prevOrders) =>
          prevOrders.map((o) =>
            o._id === orderId ? { ...o, delivery_status: newStatus } : o
          )
        );
        toast.success("Order status updated successfully");
      } else {
        toast.error("Failed to update order status");
      }
    } catch (error) {
      console.error("Error updating order status:", error);
      toast.error("An error occurred while updating order status");
    }
  };

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          <h4 className="text-center">Recent Orders</h4>

          {orders?.map((order) => (
            <div key={order?._id} className="mb-4 p-4 alert alert-
secondary">
              <table className="table table-striped">
                <tbody>
                  {/* order info */}
                  <tr>
                    <th scope="row">Customer Name:</th>
                    <td>{order?.userId?.name}</td>
                  </tr>
                  <tr>
                    <th scope="row">Charge ID:</th>
                    <td>{order?.chargeId}</td>
                  </tr>
                  <tr>
                    <th scope="row">Created:</th>
                    <td>{new Date(order?.createdAt).toLocaleDateString()}
</td>
                  </tr>
                  <tr>
                    <th scope="row">Payment Intent:</th>
                    <td>{order?.payment_intent}</td>
                  </tr>
                  <tr>
                    <th scope="row">Receipt:</th>
                    <td>
                      <a href={order?.receipt_url} target="_blank">
                        View
                      </a>
                    </td>
                  </tr>
                  <tr>
                    <th scope="row">Refunded:</th>
                    <td>{order?.refunded ? "Yes" : "No"}</td>
                  </tr>
                  <tr>
                    <th scope="row">Status:</th>
                    <td>{order?.status}</td>
                  </tr>
                  <tr>
                    <th scope="row">Total Charged:</th>
                    <td>
                      ${(order?.amount_captured / 100)?.toFixed(2)}{" "}
                      {order?.currency}
                    </td>
                  </tr>
                  <tr>
                    <th scope="row">Shipping Address:</th>
                    <td>
                      {order?.shipping?.address?.line1}
                      <br />
                      {order?.shipping?.address?.line2 &&
                        `${order?.shipping?.address?.line2}, `}
                      {order?.shipping?.address?.city}, {
                        order?.shipping?.address?.state
                      }, {order?.shipping?.address?.postal_code}
                      <br />
                      {order?.shipping?.address?.country}
                    </td>
                  </tr>
                  {/* products info */}
                  <tr>
                    <th scope="row" className="w-25">
                      Ordered Products:
                    </th>
                    <td className="w-75">
                      {order?.cartItems?.map((product) => (
                        <div
                          className="pointer text-primary"
                          key={product._id}
                          onClick={() =>
                            router.push(`/product/${product?.slug}`)
                          }
                        >
                          {product?.quantity} x {product?.title} $
```

```
                              {product?.price?.toFixed(2)} {order?.currency}
                            </div>
                          ))}
                        </td>
                      </tr>
                      <tr>
                        <th scope="row">Delivery Status:</th>
                        <td>
                          <select
                            className="form-control"
                            onChange={(e) =>
                              handleStatusChange(e.target.value, order._id)
                            }
                            value={order?.delivery_status}
                            disabled={order?.refunded}
                          >
                            <option value="Not Processed">Not
Processed</option>
                            <option value="processing">Processing</option>
                            <option value="Dispatched">Dispatched</option>
                            {order?.refunded && (
                              <option value="Cancelled">Cancelled</option>
                            )}
                            <option value="Delivered">Delivered</option>
                          </select>
                        </td>
                      </tr>
                    </tbody>
                  </table>
                </div>
              ))}
            </div>
          </div>

          <Pagination
            currentPage={currentPage}
            totalPages={totalPages}
            pathname={pathname}
          />
        </div>
      );
    }
```

Admin can manually issue refund or view receipt in stripe dashboard using payment intent [pi_xxx] id.

Graphical Chart on Admin Dashboard using recharts

```
// api/admin/chart/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import Category from "@/models/category";
import Tag from "@/models/tag";
import Order from "@/models/order";
import Blog from "@/models/blog";

export async function GET(req, context) {
  await dbConnect();

  try {
    const totalProducts = await Product.countDocuments();
    const totalOrders = await Order.countDocuments();
    const totalCategories = await Category.countDocuments();
    const totalTags = await Tag.countDocuments();
    const totalBlogs = await Blog.countDocuments();

    const data = [
      { label: "Products", count: totalProducts },
      { label: "Orders", count: totalOrders },
      { label: "Categories", count: totalCategories },
      { label: "Tags", count: totalTags },
      { label: "Blogs", count: totalBlogs },
    ];

    return NextResponse.json({ data });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}

// app/dashboard/admin/page
"use client";
import { useEffect, useState } from "react";
import AdminChart from "@/components/admin/AdminChart";

export default function AdminDashboard() {
  const [chartData, setChartData] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchChartData();
  }, []);

  const fetchChartData = async () => {
    try {
      const response = await fetch(`${process.env.API}/admin/chart`);
      const data = await response.json();

      setChartData(data.data);
      setLoading(false);
    } catch (error) {
      console.error("Error fetching chart data:", error);
      setLoading(false);
    }
  };

  if (loading) {
    return (
      <div className="d-flex justify-content-center align-items-center
text-danger vh-100 h1">
        LOADING...
      </div>
    );
  }

  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p className="lead text-center">Admin Dashboard</p>

          <AdminChart chartData={chartData} />
        </div>
      </div>
    </div>
  );
}

// components/admin/AdminChart.js
import React from "react";
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
} from "recharts";

export default function AdminChart({ chartData }) {
  return (
    <div className="container-fluid">
      <div className="row">
        <div className="col">
          <ResponsiveContainer width="95%" height={400}>
            <BarChart width={1000} height={300} data={chartData}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis dataKey="label" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Bar dataKey="count" fill="rgba(75, 192, 192, 0.6)" />
            </BarChart>
          </ResponsiveContainer>
        </div>
      </div>
    </div>
  );
}
```

Only purchaser can leave rating

```
// api/user/product/rating/route
// ...

// Check if the user has already rated the product
const existingRating = product.ratings.find(
  (rate) => rate.postedBy.toString() === token.user._id.toString()
);

// Check if the user has purchased the product
const userPurchased = await Order.findOne({
  userId: token.user._id,
  "cartItems._id": productId,
});

if (!userPurchased) {
  return NextResponse.json(
    {
      err: "You can only leave a review for products you've purchased.",
    },
    { status: 400 }
  );
}

if (existingRating) {
}
// ...

// components/product/ProductRating
const submitRating = async () => {
  if (status !== "authenticated") {
    toast.error("Please login to leave a rating");
    router.push(`/login?callbackUrl=${process.env.DOMAIN}${pathname}`);

    return;
  }
  try {
    const response = await fetch(`${process.env.API}/user/product/rating`,
{
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        productId: product?._id,
        rating: currentRating,
        comment,
      }),
    });

    if (response.status === 200) {
      const data = await response.json();
      setProductRatings(data?.ratings);
      setShowRatingModal(false);
      console.log("product rating response => ", data);
      toast.success("You left a rating");
      router.refresh(); // only works in server components
    } else if (response.status === 400) {
      const errorData = await response.json();
      toast.error(errorData.err);
    } else {
      // Handle other error scenarios
      toast.error("An error occurred. Please try again later.");
    }
  } catch (err) {
    console.log(err);
    toast.error("Error leaving a rating");
  }
};
```

Related products

```
// api/product/[slug]/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";

export async function GET(req, context) {
  await dbConnect();

  try {
    const product = await Product.findOne({
      slug: context.params.slug,
    })
      .populate("category", "name")
      .populate("tags", "name")
      .populate({
        path: "ratings.postedBy",
        model: "User", // The User model name
        select: "name", // Select the fields you want to populate
      });
    // Fetch related products based on category or tags
    const relatedProducts = await Product.find({
      $or: [
        { category: product.category }, // Fetch products in the same
category
        { tags: { $in: product.tags } }, // Fetch products with similar
tags
      ],
      _id: { $ne: product._id }, // Exclude the current product
    }).limit(3); // Limit the number of related products

    return NextResponse.json({ product, relatedProducts });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}

// app/product/[slug]/page
// ...
const { product, relatedProducts } = await getProduct(params?.slug);

<div className="row">
  <div className="col-lg-10 offset-lg-1">
    <p className="lead text-center my-5">Other products you may like</p>
    <div className="row">
      {relatedProducts?.map((product) => (
        <div className="col-lg-4" key={product._id}>
          <ProductCard product={product} />
        </div>
      ))}
    </div>
  </div>
</div>;
```

Shop page for products (without filters)

```
// api/shop/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import queryString from "query-string";

export async function GET(req) {
  await dbConnect();
  const searchParams = queryString.parseUrl(req.url).query;

  const { page } = searchParams || {};
  const pageSize = 6;

  try {
    const currentPage = Number(page) || 1;
    const skip = (currentPage - 1) * pageSize;
    const totalProducts = await Product.countDocuments({});

    const products = await Product.find({})
      .skip(skip)
      .limit(pageSize)
      .sort({ createdAt: "-1" });

    return NextResponse.json(
      {
        products,
        currentPage,
        totalPages: Math.ceil(totalProducts / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}

// app/shop/page
import ProductList from "@/components/product/ProductList";
import Pagination from "@/components/Pagination";

export const dynamic = "force-dynamic";

export const metadata = {
  title: "Next Ecommerce",
  description: "Find the latest in fashion, electronics and more",
};

async function getProducts(searchParams) {
  const searchQuery = new URLSearchParams({
    page: searchParams?.page || 1,
  }).toString();

  try {
    const response = await fetch(`${process.env.API}/product?
${searchQuery}`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
      next: { revalidate: 1 },
      // next: { cache: "no-store" },
    });

    if (!response.ok) {
      throw new Error(`Failed to fetch products: ${response.statusText}`);
    }

    const data = await response.json();

    // Check if the response has products or is empty
    if (!data || !Array.isArray(data.products)) {
      throw new Error("No products returned.");
    }

    return data;
  } catch (error) {
    console.error("Error fetching search results:", error);
    // Handle the error here, such as showing an error message to the user
    // or returning a default value
    return { products: [], currentPage: 1, totalPages: 1 };
  }
}

export default async function Prducts({ searchParams }) {
  // console.log("searchParams in products page => ", searchParams);
  const data = await getProducts(searchParams);

  return (
    <main>
      <div className="container-fluid">
        <div className="row">
          <div className="col">
            <p className="text-center lead fw-bold">Latest Products</p>
            <ProductList products={data?.products} />
          </div>
        </div>

        <Pagination
          currentPage={data?.currentPage}
          totalPages={data?.totalPages}
          pathname="/shop"
          searchParams={searchParams}
        />
      </div>
    </main>
  );
}
```

Post deployment issues (fixed)

- Replace all NEXTAUTH_URL with DOMAIN
- Use production url webhook in stripe
- Get webhook signing secret from stripe for production
- Update .env

```
// config
const DOMAIN =
  process.env.NODE_ENV === "production"
    ? "https://blog2-six-gilt.vercel.app"
    : "http://localhost:3000";

// [next-auth][warn][NEXTAUTH_URL]
const NEXTAUTH_URL =
  process.env.NODE_ENV === "production"
    ? "https://blog2-six-gilt.vercel.app"
    : "http://localhost:3000";

const STRIPE_WEBHOOK_SECRET =
  process.env.NODE_ENV === "production"
    ? "whsec_3VUXiLWiqKz3UqdSDo36oOTedT0PKScL"
    :
"whsec_0c4638ee2cb64fdb508f5a42bf58b4391d19d6c1d23dfd4fc726b7c430ad5963";

// update stripe secret if using 'live' mode for real payments
// vercel --prod
```

Post deployment updates

```
// admin and user orders page
const [loading, setLoading] = useState(true);

const fetchOrders = async (page) => {
  try {
    // ...
    setLoading(false);
  } catch (error) {
    // ...
    setLoading(false);
  }
};

if (loading) {
  return (
    <div className="d-flex justify-content-center align-items-center text-
danger vh-100 h1">
      LOADING...
    </div>
  );
}

if (!orders?.length) {
  return (
    <div className="d-flex justify-content-center align-items-center text-
danger vh-100 h1">
      No Orders
    </div>
  );
}
```

Show graphical chart in user dashboard

```
// api/user/chart/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import Order from "@/models/order";
import Blog from "@/models/blog";

import { currentUser } from "@/utils/currentUser";

export async function GET(req) {
  await dbConnect();

  const user = await currentUser();
  const userId = user._id;

  try {
    const totalLikedBlogs = await Blog.countDocuments({ likes: userId });
    const totalOrders = await Order.countDocuments({ userId });
    const totalReviews = await Product.countDocuments({
      "ratings.postedBy": userId,
    });
    const totalLikes = await Product.countDocuments({ likes: userId });

    const data = [
      {
        label: "Total Orders",
        url: "/dashboard/user/orders",
        count: totalOrders,
      },
      {
        label: "Liked Blogs",
        url: "/dashboard/user/liked/blogs",
        count: totalLikedBlogs,
      },
      {
        label: "Product Reviews",
        url: "/dashboard/user/product/reviews",
        count: totalReviews,
      },
      {
        label: "Product Likes",
        url: "/dashboard/user/liked/product",
        count: totalLikes,
      },
    ];

    return NextResponse.json({ data });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: err.message,
      },
      { status: 500 }
    );
  }
}

// check api response
// http://localhost:3000/api/user/chart

{
  "data": [
    {
      "label": "Total Orders",
      "url": "/dashboard/user/orders",
      "count": 1
    },
    {
      "label": "Liked Blogs",
      "url": "/dashboard/user/liked/blogs",
      "count": 2
    },
    {
      "label": "Product Reviews",
      "url": "/dashboard/user/product/reviews",
      "count": 1
    },
    {
      "label": "Product Likes",
      "url": "/dashboard/user/liked/products",
      "count": 1
    }
  ]
}

// app/dashboard/user/page
"use client";
import { useEffect, useState } from "react";
import UserChart from "@/components/user/UserChart";
```

```
export default function UserDashboard() {
  const [chartData, setChartData] = useState([]);

  useEffect(() => {
    fetchChartData();
  }, []);

  const fetchChartData = async () => {
    try {
      const response = await fetch(`${process.env.API}/user/chart`);
      const data = await response.json();

      setChartData(data.data);
      setLoading(false);
    } catch (error) {
      console.error("Error fetching chart data:", error);
    }
  };

  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p className="lead text-center">User Dashboard</p>

          <UserChart chartData={chartData} />
        </div>
      </div>
    </div>
  );
}

// components/user/UserChart
// non clickable
import React from "react";
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
} from "recharts";

export default function UserChart({ chartData }) {
  return (
    <div className="container-fluid">
      <div className="row">
        <div className="col">
          <ResponsiveContainer width="95%" height={400}>
            <BarChart width={1000} height={300} data={chartData}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis dataKey="label" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Bar dataKey="count" fill="rgba(75, 192, 192, 0.6)" />
            </BarChart>
          </ResponsiveContainer>
        </div>
      </div>
    </div>
  );
}

## Clickable charts label

import React from "react";
import Link from "next/link";
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
} from "recharts";

export default function UserChart({ chartData }) {
  const CustomTick = ({ payload, x, y, dataPoint }) => (
    <Link href={dataPoint.url}>
      <g transform={`translate(${x},${y})`}>
        <text
          x={0}
          y={0}
          dy={16}
          textAnchor="end"
          fill="#666"
          transform="rotate(-35)"
        >
          {payload.value}
        </text>
      </g>
    </Link>
  );

  return (
    <div className="container-fluid">
      <div className="row">
        <div className="col">
          <ResponsiveContainer width="95%" height={400}>
            <BarChart width={1000} height={300} data={chartData}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis
```

```
                dataKey="label"
                height={60}
                tick={({ payload, x, y }) => (
                  <CustomTick
                    payload={payload}
                    x={x}
                    y={y}
                    dataPoint={chartData.find(
                      (item) => item.label === payload.value
                    )}
                  />
                )}
              />
              <YAxis />
              <Tooltip />
              <Legend />
              <Bar dataKey="count" fill="rgba(75, 192, 192, 0.6)" />
            </BarChart>
          </ResponsiveContainer>
        </div>
      </div>
    </div>
  );
}
```

User reviewed products list

```
// api/user/product/reviews/route
// without pagination
// with pagination
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import { currentUser } from "@/utils/currentUser";
import queryString from "query-string";

export async function GET(req) {
  await dbConnect();

  const user = await currentUser();
  const searchParams = queryString.parseUrl(req.url).query;
  const { page } = searchParams || {};
  const pageSize = 6; // Number of ratings per page

  try {
    const currentPage = Number(page) || 1;
    const skip = (currentPage - 1) * pageSize;

    const reviews = await Product.aggregate([
      {
        $match: {
          "ratings.postedBy": user._id,
        },
      },
      {
        $lookup: {
          from: "products", // The collection name
          localField: "_id",
          foreignField: "_id",
          as: "product",
        },
      },
      {
        $unwind: "$product", // Unwind the product array
      },
      {
        $project: {
          _id: 0,
          product: {
            title: 1,
            slug: 1,
            price: 1,
           image: { $arrayElemAt: ["$product.images.secure_url", 0] },the
first image from the array
          },
          ratings: {
            $arrayElemAt: ["$ratings", 0], // Extract the first rating
from the array
          },
        },
      },
      {
        $skip: skip,
      },
      {
        $limit: pageSize,
      },
    ]);

    const totalRatings = await Product.aggregate([
      {
        $match: {
          "ratings.postedBy": user._id,
        },
      },
      {
        $group: {
          _id: null,
          totalRatings: { $sum: { $size: "$ratings" } },
        },
      },
    ]);

    const totalUserRatings =
      totalRatings.length > 0 ? totalRatings[0].totalRatings : 0;
    return NextResponse.json(
      {
        reviews,
        currentPage,
        totalPages: Math.ceil(totalUserRatings / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

User reviews API

```
// api/user/product/reviews/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import { currentUser } from "@/utils/currentUser";
import queryString from "query-string";

export async function GET(req) {
  await dbConnect();

  const user = await currentUser();
  const searchParams = queryString.parseUrl(req.url).query;
  const { page } = searchParams || {};
  const pageSize = 6; // Number of ratings per page

  try {
    const currentPage = Number(page) || 1;
    const skip = (currentPage - 1) * pageSize;

    // for each user review, lookup products
    const reviews = await Product.aggregate([
      {
        $match: {
          "ratings.postedBy": user._id,
        },
      },
      {
        $lookup: {
          from: "products", // The collection name
          localField: "_id",
          foreignField: "_id",
          as: "product",
        },
      },
      {
        $unwind: "$product", // Unwind the product array
      },
      {
        $project: {
          _id: 0,
          product: {
            title: 1,
            slug: 1,
            price: 1,
            image: { $arrayElemAt: ["$product.images.secure_url", 0] },
          },
          // ratings: {
          //   $arrayElemAt: ["$ratings", 0], // Extract the first rating
from the array
          // },
          ratings: {
            // this is to send rating of the current user only for given
product
            $arrayElemAt: [
              {
                $filter: {
                  input: "$ratings",
                  as: "rating",
                  cond: { $eq: ["$$rating.postedBy", user._id] },
                },
              },
              0,
            ],
          },
        },
      },
      {
        $sort: { createdAt: -1 }, // Sort by createdAt field in descending
order
      },
      {
        $skip: skip,
      },
      {
        $limit: pageSize,
      },
    ]);

    const totalRatings = await Product.aggregate([
      {
        $match: {
          "ratings.postedBy": user._id,
        },
      },
      {
        $group: {
          _id: null,
          totalRatings: { $sum: { $size: "$ratings" } },
        },
      },
    ]);

    const totalUserRatings =
      totalRatings.length > 0 ? totalRatings[0].totalRatings : 0;

    console.log("totalUserRatings => ", totalUserRatings);

    return NextResponse.json(
      {
        reviews,
        totalRatings: totalUserRatings,
        currentPage,
        totalPages: Math.ceil(totalUserRatings / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

User reviews

```
// app/dashboard/user/product/reviews/page
"use client";
import { useState, useEffect } from "react";
import { useRouter, usePathname, useSearchParams } from "next/navigation";
import ProductReviews from "@/components/product/ProductReviews";
import Pagination from "@/components/Pagination";

export default function UserProductReviewsPage() {
  const [reviews, setReviews] = useState([]);
  // pagination
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);
  const [loading, setLoading] = useState(true);
  const pathname = usePathname();
  const searchParams = useSearchParams();
  const page = searchParams.get("page");
  console.log("current page => ", page);

  const router = useRouter();

  useEffect(() => {
    fetchReviews(page);
  }, [page]);

  const fetchReviews = async (page) => {
    try {
      const response = await fetch(
        `${process.env.API}/user/product/reviews?page=${page}`,
        {
          method: "GET",
        }
      );
      const data = await response.json();
      // console.log("DATA in admin orders with pagination => ", data);
      setReviews(data.reviews);
      setCurrentPage(data.currentPage);
      setTotalPages(data.totalPages);
      setLoading(false);
    } catch (error) {
      console.log(error);
      toast.error(error);
      setLoading(false);
    }
  };

  if (loading) {
    return (
      <div className="d-flex justify-content-center align-items-center
text-danger vh-100 h1">
        LOADING...
      </div>
    );
  }

  if (!reviews?.length) {
    return (
      <div className="d-flex justify-content-center align-items-center
text-danger vh-100 h1">
        No Orders
      </div>
    );
  }

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
```

```
          <p className="lead mb-4 text-center">Product Reviews</p>
          <ProductReviews reviews={reviews} />
        </div>
      </div>

      <Pagination
        currentPage={currentPage}
        totalPages={totalPages}
        pathname={pathname}
      />
    </div>
  );
}
```

All product reviews API

```
// api/admin/product/reviews/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
import queryString from "query-string";

export async function GET(req) {
  await dbConnect();

  const searchParams = queryString.parseUrl(req.url).query;
  const { page } = searchParams || {};
  const pageSize = 6; // Number of reviews per page

  try {
    const currentPage = Number(page) || 1;
    const skip = (currentPage - 1) * pageSize;

    // Count all ratings, not just documents
    const allRatings = await Product.aggregate([
      {
        $unwind: "$ratings",
      },
    ]);
    const totalReviews = allRatings.length;

    const reviews = await Product.aggregate([
      {
        $lookup: {
          from: "products",
          localField: "_id",
          foreignField: "_id",
          as: "product",
        },
      },
      {
        $unwind: "$ratings",
      },
      {
        $project: {
          _id: 0,
          product: {
            title: 1,
            slug: 1,
            images: { $arrayElemAt: ["$images", 0] },
          },
          rating: "$ratings.rating",
          comment: "$ratings.comment",
          postedBy: "$ratings.postedBy",
        },
      },
    ])
      .skip(skip)
      .limit(pageSize);

    return NextResponse.json(
      {
        reviews,
        currentPage,
        totalPages: Math.ceil(totalReviews / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Product reviews and delete for admin

Admin can see all reviews and delete

```
// dashboard/admin/product/reviews/page
"use client";
import { useState, useEffect } from "react";
import { useRouter, usePathname, useSearchParams } from "next/navigation";
import ProductReviews from "@/components/product/ProductReviews";
import Pagination from "@/components/Pagination";
import toast from "react-hot-toast";

export default function AdminProductReviewsPage() {
  const [reviews, setReviews] = useState([]);
  const [totalRatings, setTotalRatings] = useState(0);
  // pagination
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);
  const [loading, setLoading] = useState(true);

  const pathname = usePathname();
  const searchParams = useSearchParams();
  const page = searchParams.get("page");
  console.log("current page => ", page);

  const router = useRouter();

  useEffect(() => {
    fetchReviews(page);
  }, [page]);

  const fetchReviews = async (page) => {
    try {
      const response = await fetch(
        `${process.env.API}/admin/product/reviews?page=${page}`,
        {
          method: "GET",
        }
      );
      const data = await response.json();
      console.log("DATA in admin reviews with pagination => ", data);
      setReviews(data.reviews);
      setCurrentPage(data.currentPage);
      setTotalPages(data.totalPages);
      setTotalRatings(data.totalRatings);
      setLoading(false);
    } catch (error) {
      console.log(error);
      toast.error(error);
      setLoading(false);
    }
  };

  const handleDelete = async (ratingId) => {
    try {
      const response = await fetch(
        `${process.env.API}/admin/product/reviews/remove`,
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({ ratingId }),
        }
      );

      const data = await response.json();
      toast.success(data.message);
      fetchReviews(page);
    } catch (error) {
      console.error("Error deleting rating:", error);
    }
  };

  if (loading) {
    return (
      <div className="d-flex justify-content-center align-items-center
text-danger vh-100 h1">
        LOADING...
      </div>
    );
  }

  if (!reviews?.length) {
    return (
      <div className="d-flex justify-content-center align-items-center
text-danger vh-100 h1">
        No Orders
      </div>
    );
  }

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          <p className="lead mb-4 text-center">
            Product Reviews ({totalRatings})
          </p>
          <ProductReviews reviews={reviews} handleDelete={handleDelete} />
        </div>
      </div>

      <Pagination
        currentPage={currentPage}
        totalPages={totalPages}
        pathname={pathname}
      />
    </div>
  );
}
```

Admin product reviews delete API

```
// api/admin/product/reviews/remove/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Product from "@/models/product";
```

```
export async function POST(req) {
  await dbConnect();

  const body = await req.json();

  const { ratingId } = body;
  // console.log("ratingId => ", ratingId);

  try {
    const product = await Product.findOneAndUpdate(
      { "ratings._id": ratingId },
      { $pull: { ratings: { _id: ratingId } } },
      { new: true }
    );

    if (!product) {
      return NextResponse.json(
        { message: "Rating not found", success: false },
        { status: 404 }
      );
    }

    return NextResponse.json({ message: "Rating removed", success: true
});
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Product reviews component

```
import Image from "next/image";
import Link from "next/link";
import Stars from "@/components/product/Stars";

export default function ProductReviews({ reviews, handleDelete }) {
  return (
    <div className="row">
      {reviews.map((review, index) => (
        <div className="col-lg-8 offset-lg-2 card mb-3" key={index}>
          {/* <pre className="bg-warning">{JSON.stringify(review, null,
4)}</pre> */}
          <div className="row g-0">
            <div style={{ width: "100px", overflow: "hidden" }}>
```

```
                <Image
                  src={review?.product?.image || "/images/new-wave.jpeg"}
                  className="card-img-top"
                  width={200}
                  height={200}
                  style={{
                    objectFit: "cover",
                    height: "100%",
                    width: "100%",
                  }}
                />
              </div>

              <div className="col-lg-8">
                <div className="card-body">
                  <h5 className="card-title">
                    <Link
                      href={`/product/${review?.product?.slug}`}
                      as={`/product/${review?.product?.slug}`}
                    >
                      {review?.product?.title}
                    </Link>
                  </h5>

                  <div className="d-flex justify-content-between">
                    <div>
                      <Stars rating={review?.ratings?.rating} />
                    </div>

                    {handleDelete && (
                      <button
                        className="btn btn-danger btn-raised border-20"
                        onClick={() => handleDelete(review?.ratings?._id)}
                      >
                        X
                      </button>
                    )}
                  </div>

                  {review?.ratings?.comment && (
                    <p className="card-text mb-0">{review?.ratings?.comment}
</p>
                  )}
                  {review?.ratings?.postedBy?.name && (
                    <p className="text-secondary">
                      {review?.ratings?.postedBy?.name}
                    </p>
                  )}
                </div>
              </div>
            </div>
          </div>
        ))}
      </div>
```

```
  );
}
```

Forgot password

```
// login page
<Link className="btn mb-4" href="/forgot-password">
  <small>Forgot Password</small>
</Link>;

// app/forgot-password/page
("use client");
import { useState } from "react";
import toast from "react-hot-toast";
import { useRouter } from "next/navigation";

export default function ForgotPassword() {
  // to find user in db and send resetcode via email
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  // to reset password (enter emailed resetcode and new password)
  const [resetCode, setResetCode] = useState("");
  const [loading, setLoading] = useState(false);

  const router = useRouter();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      setLoading(true);
      const response = await fetch(`${process.env.API}/password/forgot`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          email,
          password,
        }),
      });

      const data = await response.json();

      if (!response.ok) {
        toast.error(data.err);
        setLoading(false);
      } else {
        setResetCode(" "); // Set reset code to trigger the resetCode form
with white space
        toast.success(data.message);
        setLoading(false); // Clear loading state after successful reset
      }
    } catch (err) {
      console.log(err);
      setLoading(false);
      toast.error("An error occurred. Please try again.");
    }
  };

  const handleReset = async (e) => {
    e.preventDefault();
    try {
      setLoading(true);
      const response = await fetch(`${process.env.API}/password/reset`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          email,
          password,
          resetCode,
        }),
      });

      const data = await response.json();

      if (!response.ok) {
        toast.error(data.err);
        setLoading(false);
        return;
      } else {
        toast.success(data.message);
        setLoading(false); // Clear loading state after successful reset
        router.push("/login");
      }
    } catch (err) {
      console.log(err);
      setLoading(false);
      toast.error("An error occurred. Please try again.");
    }
  };

  if (resetCode) {
    return (
      <div className="container">
        <div className="row d-flex justify-content-center align-items-
center vh-100">
          <div className="col-lg-5 bg-light p-5 shadow">
            <h2 className="mb-3">Reset Password</h2>

            <form onSubmit={handleReset}>
              <input
                type="text"
                value={resetCode}
                onChange={(e) => setResetCode(e.target.value.trim())}
                className="form-control mb-3"
                placeholder="Your reset code"
              />
              <button
                className="btn btn-primary btn-raised"
                disabled={loading || !resetCode}
              >
                {loading ? "Please wait.." : "Reset Password"}
              </button>
            </form>
          </div>
        </div>
      </div>
    );
  }

  return (
    <main>
      <div className="container">
        <div className="row d-flex justify-content-center align-items-
center vh-100">
          <div className="col-lg-5 bg-light p-5 shadow">
            <h2 className="mb-3">Forgot Password</h2>

            <form onSubmit={handleSubmit}>
              <input
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                className="form-control mb-3"
                placeholder="Your email"
              />
              <input
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                className="form-control mb-3"
                placeholder="Your new password"
              />
              <button
                className="btn btn-primary btn-raised"
                disabled={loading || !email || !password}
              >
                {loading ? "Please wait.." : "Submit"}
              </button>
            </form>
          </div>
        </div>
      </div>
    </main>
  );
}
```

Forgot password API with sending emails using nodemailer

```
// api/password/forgot/route
/**
 * https://support.google.com/accounts/answer/185833?
visit_id=638278421558889969-1671626849&p=InvalidSecondFactor&rd=1
 * first turn on 2 factor authentication > https://myaccount.google.com/
 * sidebar > security > 2-step verification
 * once you add 2-step verification, again click on that section that says
"2-Step Verification", on newly opened page/modal scroll down until you
see "App passwords"
 * https://myaccount.google.com/u/6/apppasswords?utm_source=google-
account&utm_medium=myaccountsecurity&utm_campaign=tsv-
settings&rapt=AEjHL4P5CPsNWmPWuuUS-
H4oWsBgxGz4qfnj4NVejZUlP8cdQwUgHtaqJXd9y6QtTpVnra2ca8UxK-
tUb3n2wZ4lsoWmosCLtw
 * check spam folders if not received in inbox
 */

import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import User from "@/models/user";
import randomInteger from "random-int";
import nodemailer from "nodemailer";

const transporter = nodemailer.createTransport({
  service: "Gmail",
  auth: {
    user: process.env.GMAIL_AUTH_USER,
    pass: process.env.GMAIL_AUTH_PASS,
  },
});

export async function POST(req) {
  const body = await req.json();

  await dbConnect();

  const { email } = body;

  // Check if user with email exists
  const user = await User.findOne({ email });

  if (!user) {
    return NextResponse.json(
      {
        err: "User not found",
      },
      { status: 400 }
    );
  }
```

```
    // const resetCode = nanoid(6); // Generate a 6-character code
    const resetCode = randomInteger(100000, 999999);

    // Save reset code in the user document
    user.resetCode = {
      data: resetCode,
      expiresAt: new Date(Date.now() + 10 * 60 * 1000), // 10 minutes in
  milliseconds
    };
    await user.save();

    // Send email
    const mailOptions = {
      to: email,
      from: process.env.GMAIL_AUTH_USER,
      subject: "Password Reset Code",
      html: `
            Hi ${user.name},<br />
            <br />
            You have requested a password reset. Please use the following
  code to reset your password:<br />
            <br />
            <strong>${resetCode}</strong><br />
            <br />
            If you did not request a password reset, please ignore this
  email.<br />
            <br />
            Thanks,<br />
            The Nextecom Team
        `,
    };

    //   return NextResponse.json({
    //     message: "Check your email for password reset code",
    //   });

    //   transporter.sendMail(mailOptions, (error, info) => {
    //     if (error) {
    //       console.error("Error sending email:", error);
    //       return NextResponse.json(
    //         {
    //           err: "Error sending email",
    //         },
    //         { status: 500 }
    //       );
    //     } else {
    //       console.log("Email sent:", info.response);
    //       return NextResponse.json({
    //         message: "Check your email for password reset code",
    //       });
    //     }
    //   });
    try {
      // Send the email
      await transporter.sendMail(mailOptions);

      // Assuming that the email is sent successfully, send the response to
  the client.
      return NextResponse.json({
        message: "Check your email for password reset code",
      });
    } catch (error) {
      console.error("Error sending email:", error);

      // If there's an error while sending the email, return an appropriate
  error response.
      return NextResponse.json(
        {
          err: "Error sending email",
        },
        { status: 500 }
      );
    }
  }
```

Password reset API

```
// api/password/reset/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import User from "@/models/user";
import bcrypt from "bcrypt";

export async function POST(req) {
  const body = await req.json();

  await dbConnect();

  try {
    const { email, password, resetCode } = body;

    // Check if user with email exists
    const user = await User.findOne({
      email: email,
      "resetCode.data": resetCode,
      "resetCode.expiresAt": { $gt: new Date() },
    });

    if (!user) {
      return NextResponse.json(
        {
          err: "Invalid or expired reset code",
        },
        { status: 400 }
      );
    }

    // Reset the user's password and save the updated user
    user.password = await bcrypt.hash(password, 10);
    user.resetCode = null; // Clear the reset code
    await user.save();

    // Send success response
    return NextResponse.json({
      message: "Password reset successful. Login with your new password.",
    });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Single category with products API

```
// api/category/slug/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Category from "@/models/category";
import Product from "@/models/product";

export async function GET(req, context) {
  await dbConnect();
  const slug = context.params.slug;

  try {
    const category = await Category.findOne({ slug });

    const products = await Product.find({ category }).limit(12).sort({
      createdAt: "-1",
    });

    return NextResponse.json({ category, products });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
      );
    }
  }
```

Category products page

Single category view with products

```
// app/category/[slug]/page
import ProductList from "@/components/product/ProductList";
import TagsList from "@/components/tag/TagList";

export const dynamic = "force-dynamic";

export async function generateMetadata({ slug }) {
  const category = await getCategory(slug);
  return {
    title: category?.name,
    description: `Best selling products on category ${category?.name}`,
  };
}

async function getCategory(slug) {
  try {
    const response = await fetch(`${process.env.API}/category/${slug}`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    });

    const data = await response.json();
    console.log("category page response => ", data);
    return data;
  } catch (error) {
    console.error("Error fetching search results:", error);
  }
}

export default async function CategoryViewPage({ params }) {
  const { category, products } = await getCategory(params?.slug);

  return (
    <main>
      <div className="container-fluid">
        <div className="row">
          <div className="col-lg-3 mt-5">
            <div className="btn btn-danger btn-raised border-20 col p-4
mb-3">
              {category?.name}
              <div className="mt-4">
                <TagsList category={category} />
              </div>
            </div>
          </div>

          <div className="col-lg-9">
            <p className="text-center lead fw-bold">
              Products in category "{category?.name}"
            </p>
            <ProductList products={products} />
          </div>
        </div>
      </div>
    </main>
  );
}
```

Update TagList component

```
// components/tag/TagList
"use client";
import { useEffect } from "react";
import { useTag } from "@/context/tag";
import Link from "next/link"; // Use Link from next/link

export default function TagsList({ category }) {
  // context
  const { tags, fetchTags, setUpdatingTag } = useTag();

  useEffect(() => {
    fetchTags();
  }, []);

  if (category) {
    // Display only filtered tags within Link
    const filteredTags = tags.filter((t) => t.parent?._id ===
category._id);

    return (
      <div className="container mb-5">
        <div className="row">
          <div className="col">
            {filteredTags.map((t) => (
              <div key={t._id}>
                <Link href={`/tag/${t.slug}`} className="btn text-dark">
                  {t?.name}
                </Link>
              </div>
            ))}
          </div>
        </div>
      </div>
    );
  } else {
    // Display all tags as buttons
    return (
      <div className="container mb-5">
        <div className="row">
          <div className="col">
            {tags.map((t) => (
              <div key={t._id}>
                <button
                  className="btn"
                  onClick={() => {
                    setUpdatingTag(t);
                  }}
                >
                  {t?.name}
                </button>
              </div>
            ))}
          </div>
        </div>
      </div>
    );
  }
}
```

Tag with products API

```
// api/tag/[slug]/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Tag from "@/models/tag";
import Product from "@/models/product";

export async function GET(req, context) {
  await dbConnect();
  const slug = context.params.slug;

  try {
    const tag = await Tag.findOne({ slug }).populate("parent", "name
slug");

    const products = await Product.find({ tags: tag })
      .populate("tags", "name")
      .populate("category", "name")
      .limit(12)
      .sort({
        createdAt: "-1",
      });
    return NextResponse.json({ tag, products });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Tag view page with products

```
// app/tag/[slug]/page
import ProductList from "@/components/product/ProductList";
import TagsList from "@/components/tag/TagList";
import Link from "next/link";

export const dynamic = "force-dynamic";

export async function generateMetadata({ slug }) {
  const tag = await getTag(slug);
  return {
    title: tag?.name,
    description: `Best selling products with the tag of "${tag?.name}" in
category "${tag?.parent?.name}"`,
  };
}

async function getTag(slug) {
  try {
    const response = await fetch(`${process.env.API}/tag/${slug}`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    });

    const data = await response.json();
    console.log("tags page response => ", data);
    return data;
  } catch (error) {
    console.error("Error fetching search results:", error);
  }
}

export default async function TagViewPage({ params }) {
  const { tag, products } = await getTag(params?.slug);
  return (
    <main>
      <div className="container-fluid">
        <div className="row">
          <div className="col-lg-3 mt-5">
            <div className="btn btn-danger btn-raised border-20 col p-4
mb-3">
              {tag?.name} /{" "}
              <Link
                href={`/category/${tag?.parent?.slug}`}
                className="text-dark"
              >
                {tag?.parent?.name}
              </Link>
            </div>
          </div>

          <div className="col-lg-9">
            <p className="text-center lead fw-bold">
              Products with tag "{tag?.name}" from category "
{tag?.parent?.name}"
            </p>
            <ProductList products={products} />
          </div>
        </div>
      </div>
    </main>
  );
}
```