NextJS_13_NextAuth_Serverless_API

- Search page
- Deploy to vercel

Project setup

```
npx create-next-app@latest
```

Delete `page.module.css` and all css from `globals.css`

```
// app/page
export default function Home() {
  return (
    <div className="container">
      <h1>Home</h1>
    </div>
  );
}

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

export const metadata = {
  title: "Blogs App",
  description: "Latest blogs on Web Development, React, Next.js, and
more.",
};

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

Use bootstrap material

```
npm i bootstrap-material-design
```

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

Navigation

```
// create nav
// components/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="/">
        BLOG
      </Link>

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

// import <TopNav /> in layout
```

Create NextJs 13 API (Serverless functions) inside app directory

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

export async function GET(request) {
  return NextResponse.json({ message: "Hello world" });
}
```

Connect to MongoDB

```
// MongoDB connection
// npm i mongoose next-auth

// .env.local
DB_URI=mongodb://127.0.0.1:27017/next-auth

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

const dbConnect = async () => {
  if (mongoose.connection.readyState >= 1) {
```

```
    return;
  }
  mongoose.connect(process.env.DB_URI);
};

export default dbConnect;
```

Create mongoose model

```
// User model
import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
  {
    name: String,
    email: {
      type: String,
      required: true,
      index: true,
      lowercase: true,
    },
    password: String,
    role: {
      type: String,
      default: "user",
    },
    image: String,
  },
  { timestamps: true }
);

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

Register API

```
// create register api
// npm i bcrypt

// app/api/register/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 _req = await req.json();
  // console.log("_req => ", _req);
  await dbConnect();
  try {
```

```
    const { name, email, password } = _req;
    // Check if user with email already exists
    const existingUser = await User.findOne({ email });

    if (existingUser) {
      return NextResponse.json(
        {
          err: "User with that email already exists",
        },
        { status: 409 }
      );
    } else {
      await new User({
        name,
        email,
        password: await bcrypt.hash(password, 10),
      }).save();
      return NextResponse.json(
        {
          success: "Registered successfully",
        },
        { status: 200 }
      );
    }
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
// you can test with postman
```

Toast notification

```
// npm i react-hot-toast

// layout
<Toaster position="top-right" />
```

Environment variables

```
// config.js
const DB_URI = "mongodb+srv://xxx";
const API =
  process.env.NODE_ENV === "production"
```

```
    ? "https://xxx.vercel.com/api"
    : "http://localhost:3000/api";
const NEXTAUTH_SECRET = "YOUR_dsafdsa";
const GOOGLE_CLIENT_ID = "xxx-xxx.apps.googleusercontent.com";
const GOOGLE_CLIENT_SECRET = "XWOixxxhM-xxx-cv_S";
const CLOUDINARY_UPLOAD_PRESET = "xxx";
const CLOUDINARY_URL = "https://api.cloudinary.com/v1_1/xxx/image/upload";

module.exports = {
  DB_URI,
  API,
  NEXTAUTH_SECRET,
  GOOGLE_CLIENT_ID,
  GOOGLE_CLIENT_SECRET,
  CLOUDINARY_UPLOAD_PRESET,
  CLOUDINARY_URL,
};

// next.config.js
/** @type {import('next').NextConfig} */
const config = require("./config");

const nextConfig = {
  env: {
    DB_URI: config.DB_URI,
    API: config.API,
    NEXTAUTH_SECRET: config.NEXTAUTH_SECRET,
    GOOGLE_CLIENT_ID: config.GOOGLE_CLIENT_ID,
    GOOGLE_CLIENT_SECRET: config.GOOGLE_CLIENT_SECRET,
    CLOUDINARY_UPLOAD_PRESET: config.CLOUDINARY_UPLOAD_PRESET,
    CLOUDINARY_URL: config.CLOUDINARY_URL,
  },
};

module.exports = nextConfig;
```

Register page

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

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 router = useRouter();
```

```
  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      setLoading(true);
      const response = await
fetch(`${process.env.NEXT_PUBLIC_API}/register`, {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          name,
          email,
          password,
        }),
      });

      if (!response.ok) {
        const data = await response.json();
        toast.error(data.err);
        setLoading(false);
        return;
      }

      const data = await response.json();
      toast.success(data.success);
      router.push("/login");
    } catch (err) {
      console.log(err);
      setLoading(false);
      toast.error("An error occurred. Please try again.");
    }
  };

  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>Register</h2>

            <form onSubmit={handleSubmit}>
              <input
                type="text"
                value={name}
                onChange={(e) => setName(e.target.value)}
                className="form-control mb-2"
                placeholder="Your name"
              />
              <input
                type="email"
                value={email}
```

```
                onChange={(e) => setEmail(e.target.value)}
                className="form-control mb-2"
                placeholder="Your email"
              />
              <input
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                className="form-control mb-2"
                placeholder="Your password"
              />
              <button
                className="btn btn-primary"
                disabled={loading || !name || !email || !password}
              >
                {loading ? "Please wait.." : "Submit"}
              </button>
            </form>
          </div>
        </div>
      </div>
    </main>
  );
}
```

Login Page

```
"use client";
import { useState } from "react";
import toast from "react-hot-toast";
import { useRouter } from "next/navigation";
import { signIn } from "next-auth/react";

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

  const router = useRouter();

  const handleSubmit = async (e) => {
    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("/");
    }
  };

  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>Login</h2>

            <form onSubmit={handleSubmit}>
              <input
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                className="form-control mb-2"
                placeholder="Your email"
              />
              <input
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                className="form-control mb-2"
                placeholder="Your password"
              />
              <button
                className="btn btn-primary"
                disabled={loading || !email || !password}
              >
                {loading ? "Please wait.." : "Submit"}
              </button>
            </form>
          </div>
        </div>
      </div>
    </main>
  );
}
```

Authentication using Next Auth

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

```
// utils/authOptions
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, handler as PUT, handler as
DELETE };
```

Provide user session

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

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <SessionProvider>
          <Toaster position="top-right" />
          <TopNav />
          {children}
        </SessionProvider>
      </body>
    </html>
  );
}
```

Access logged in user info

```
// Access session info in TopNav
import { useSession, signOut } from "next-auth/react";

import Link from "next/link";

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

  // console.log(data, status);

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

      {status === "authenticated" ? (
```

```
          <div className="d-flex">
            <Link className="nav-link" href="/dashboard">
              Dashboard
            </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;
}
```

User dashboard

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

Protecting pages

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

Redirect back to intended page

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

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

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

// save social login user in db

// 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;
  },
},

// Current user info
// 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 }) => {
      const userByEmail = await User.findOne({ email: token.email });
      userByEmail.password = undefined;
      session.user = userByEmail;
      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>
```

Admin layout and page

```
// app/dashboard/admin/layout
import Link from "next/link";

export default function AdminLayout({ children }) {
  return (
    <>
      <nav className="nav justify-content-center">
        <Link className="nav-link" href="/dashboard/admin">
          Admin
        </Link>
        <Link className="nav-link" href="/dashboard/admin/blog/create">
          Create Blog
        </Link>
      </nav>
      {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

```
// middleware.js
// export { default } from "next-auth/middleware";
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;
        }
      },
    },
  }
);
```

Blog Model

```
// Blog model

// models/blog
import mongoose from "mongoose";

const blogSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
    },
    slug: {
      type: String,
      required: true,
      unique: true,
      lowercase: true,
    },
    content: {
      type: String,
      required: true,
    },
    category: {
      type: String,
      required: true,
    },
```

```
    image: String,
    postedBy: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
    },
    likes: {
      type: [mongoose.Schema.Types.ObjectId],
      default: [],
    },
  },
  { timestamps: true }
);

export default mongoose.models.Blog || mongoose.model("Blog", blogSchema);
```

Use cloudinary for image uploads

```
// config.js
NEXT_PUBLIC_CLOUDINARY_UPLOAD_PRESET = xxx;
NEXT_PUBLIC_CLOUDINARY_URL = xxx;
```

Blog create page

```
// Blog create page for admin

// npm i react-quill

// app/dashboard/admin/blog/create/page
"use client";
import { useState } from "react";
import { useRouter } from "next/navigation";
import toast from "react-hot-toast";
import dynamic from "next/dynamic";
const ReactQuill = dynamic(() => import("react-quill"), { ssr: false });
import "react-quill/dist/quill.snow.css";

export default function AdminBlogCreate() {
  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");
  const [category, setCategory] = useState("");
  const [image, setImage] = useState("");
  const [loading, setLoading] = useState(false);

  const router = useRouter();

  // cloudinary - click on settings icon > preset > unsigned
  const uploadImage = async (e) => {
    const file = e.target.files[0];
    if (file) {
```

```
      setLoading(true);
      const formData = new FormData();
      formData.append("file", file);
      formData.append(
        "upload_preset",
        process.env.NEXT_PUBLIC_CLOUDINARY_UPLOAD_PRESET
      ); // replace with your upload_preset

      // upload to cloudinary
      try {
        const response = await
fetch(process.env.NEXT_PUBLIC_CLOUDINARY_URL, {
          method: "POST",
          body: formData,
        });

        if (response.ok) {
          const data = await response.json();
          setImage(data.secure_url);
        } else {
          console.log("Image upload failed");
        }
      } catch (err) {
        console.log("Error uploading image:", err);
      }

      setLoading(false);
    }
  };

  const createBlog = async () => {
    try {
      const response = await fetch(
        `${process.env.NEXT_PUBLIC_API}/admin/blog`,
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            title,
            content,
            category,
            image,
          }),
        }
      );

      if (response.ok) {
        router.push("/dashboard/admin");
        toast.success("Blog created successfully");
      } else {
        const errorData = await response.json();
        toast.error(errorData.err);
```

```
      }
    } catch (err) {
      console.log("err => ", err);
      toast.error("An error occurred while creating the blog");
    }
  };

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
          <p>Create Blog</p>
          <label className="text-secondary">Blog title</label>
          <input
            type="text"
            value={title}
            onChange={(e) => setTitle(e.target.value)}
            className="form-control p-2 my-2"
          />

          <label className="text-secondary">Blog content</label>
          <ReactQuill
            className="border rounded my-2"
            value={content}
            onChange={setContent}
          />

          <label className="text-secondary">Blog category</label>
          <input
            type="text"
            value={category}
            onChange={(e) => setCategory(e.target.value)}
            className="form-control p-2 my-2"
          />

          {image && (
            <img src={image} alt="preview" style={{ width: "100px" }} />
          )}

          <div className="d-flex justify-content-between mt-3">
            <button className="btn btn-outline-secondary">
              <label className="mt-2" htmlFor="upload-button">
                {loading ? "Uploading..." : "Upload image"}
              </label>
              <input
                id="upload-button"
                type="file"
                accept="image/*"
                hidden
                onChange={uploadImage}
              />
            </button>

            <button
```

```
              diasabled={loading}
              className="btn bg-primary text-light"
              onClick={createBlog}
            >
              Save
            </button>
          </div>
        </div>
      </div>
    </div>
  );
}
```

Create blog API

```
// npm i slugify

// app/api/admin/blog/route.js
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Blog from "@/models/blog";
import slugify from "slugify";
import { getToken } from "next-auth/jwt";

export async function GET(request) {
  return NextResponse.json({ message: "Hello from Blog GET endpoint" });
}

export async function POST(req) {
  const _req = await req.json();
  console.log("_req => ", _req);
  await dbConnect();

  try {
    const { title, content, category, image } = _req;

    // Check if required fields are filled
    switch (true) {
      case !title:
        return NextResponse.json({ err: "Title is required" }, { status:
400 });
      case !content:
        return NextResponse.json(
          { err: "Content is required" },
          { status: 400 }
        );
      case !category:
        return NextResponse.json(
          { err: "Category is required" },
          { status: 400 }
        );
```

```
    }

    // Check if blog with that title already exists
    const existingBlog = await Blog.findOne({
      slug: slugify(title?.toLowerCase()),
    });

    // get token to get current user's id
    const token = await getToken({
      req,
      secret: process.env.NEXTAUTH_SECRET,
    });

    if (existingBlog) {
      return NextResponse.json(
        {
          err: "Blog with that title already exists",
        },
        { status: 409 }
      );
    } else {
      const blog = await Blog.create({
        title,
        content,
        category,
        image: image ? image : null,
        postedBy: token.user._id,
        slug: slugify(title?.toLowerCase()),
      });

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

Blogs list API with Pagination

```
// blog NOT blogs

// api/blog/route.js
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Blog from "@/models/blog";
```

```
import queryString from "query-string";

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

  // console.log("req.url ==================> ", req.url);
  const searchParams = queryString.parseUrl(req.url).query;
  // console.log("searchParams => ", searchParams.page);

  const { page } = searchParams || {}; // Use searchParams.page instead of
req.query.page
  const pageSize = 6;

  try {
    const currentPage = Number(page) || 1; // Set default page to 1 if not
provided
    const skip = (currentPage - 1) * pageSize; // Calculate the number of
documents to skip
    const totalBlogs = await Blog.countDocuments({}); // Get the total
count of blogs

    const blogs = await Blog.find({})
      .populate("postedBy", "name")
      .skip(skip)
      .limit(pageSize)
      .sort({ createdAt: "-1" }); // Retrieve the paginated blogs

    console.log(blogs.length);

    return NextResponse.json(
      {
        blogs,
        currentPage,
        totalPages: Math.ceil(totalBlogs / pageSize),
      },
      { status: 200 }
    );
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Blog list on home page

```
// npm i query-string

// app/page.js

import queryString from "query-string";
import Link from "next/link";

async function getBlogs(searchParams) {
  const urlParams = {
    page: searchParams.page || 1,
  };

  const searchQuery = new URLSearchParams(urlParams).toString();
  console.log("searchQuery => ", searchQuery); // page=1

  // blog NOT blogs
  const response = await fetch(`${process.env.API}/blog?${searchQuery}`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      // Add any additional headers if required
    },
    // cache: "no-store", // NEVER USE THIS ANYWHERE
    next: { revalidate: 1 },
  });

  if (!response.ok) {
    console.log("Failed to fetch blogs => ", response);
    throw new Error("Failed to fetch blogs.");
  }

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

export default async function Home({ searchParams = { page: "1" } }) {
  // console.log("searchParams => ", searchParams);
  const data = await getBlogs(searchParams);
  const { blogs, currentPage, totalPages } = data;

  const hasPreviousPage = currentPage > 1;
  const hasNextPage = currentPage < totalPages;

  return (
    <main>
      <p className="text-center lead fw-bold">Blogs {blogs?.length}</p>

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

      <div className="d-flex justify-content-center">
        <nav aria-label="Page navigation">
          <ul className="pagination">
            {hasPreviousPage && (
```

```
              <li className="page-item">
                <Link
                  className="page-link px-3"
                  href={`?page=${currentPage - 1}`}
                >
                  Previous
                </Link>
              </li>
            )}

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

            {hasNextPage && (
              <li className="page-item">
                <Link
                  className="page-link px-3"
                  href={`?page=${currentPage + 1}`}
                >
                  Next
                </Link>
              </li>
            )}
          </ul>
        </nav>
      </div>
    </main>
  );
}
```

Use blog cards to display list of blogs

```
// components/blogs/ListBlog.js
import BlogCard from "./BlogCard";

export default function ListBlog({ blogs }) {
  return (
    <div className="container mb-5">
```

```
        <div className="row g-4">
          {blogs?.map((blog) => (
            <div className="col-lg-4" key={blog.id}>
              <BlogCard blog={blog} />
            </div>
          ))}
        </div>
      </div>
    );
}


// components/blog/BlogCard

// npm i dayjs
import Link from "next/link";
import dayjs from "dayjs";
import relativeTime from "dayjs/plugin/relativeTime";

dayjs.extend(relativeTime);

export default function BlogCard({ blog }) {
  return (
    <div className="card" className="mb-4">
      <div style={{ height: "200px", overflow: "hidden" }}>
        <img
          src={blog?.image || "/images/new-wave.jpeg"}
          className="card-img-top"
          style={{ objectFit: "cover", height: "100%", width: "100%" }}
          alt={blog.title}
        />
      </div>
      <div className="card-body">
        <h5 className="card-title">
          <Link href={`/blog/${blog.slug}`}>{blog.title}</Link>
        </h5>
        <div className="card-text">
          <div
            dangerouslySetInnerHTML={{
              __html:
                blog.content.length > 160
                  ? `${blog.content.substring(0, 160)}...`
                  : blog.content,
            }}
          ></div>
        </div>
      </div>
      <div className="card-footer d-flex justify-content-between">
        <small className="text-muted">Category: {blog.category}</small>
        <small className="text-muted">
          Author: {blog.postedBy?.name || "Admin"}
        </small>
      </div>
      <div className="card-footer d-flex justify-content-between">
```

```
          <small>❤ {blog?.likes?.length} likes</small>
          <small className="text-muted">
            Posted {dayjs(blog.updatedAt).fromNow()}
          </small>
        </div>
      </div>
    );
}

// show list of blogs
<ListBlog blogs={blogs} />
```

Single blog API

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

export async function GET(req, context) {
  await dbConnect();
  // console.log("req ==================> ", context);

  try {
    const blog = await Blog.findOne({ slug: context.params.slug
}).populate(
      "postedBy",
      "name"
    );
    return NextResponse.json(blog, { status: 200 });
  } catch (err) {
    console.log(err);
    return NextResponse.json(
      {
        err: "Server error. Please try again.",
      },
      { status: 500 }
    );
  }
}
```

Blog view page

```
// app/blog/[slug]/page

// npm i dayjs
// public/images/new-wave.jpg

import dayjs from "dayjs";
```

```
import relativeTime from "dayjs/plugin/relativeTime";

dayjs.extend(relativeTime);

async function getBlog(slug) {
  const apiUrl = `${process.env.API}/blog/${slug}`;

  const options = {
    method: "GET",
    next: { revalidate: 1 },
    // cache: "no-store", // required to update likes later
  };

  try {
    const response = await fetch(apiUrl, options);

    if (!response.ok) {
      throw new Error(
        `Failed to fetch: ${response.status} ${response.statusText}`
      );
    }

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

export default async function BlogViewPage({ params }) {
  //   console.log("params in single blog view => ", params);
  const blog = await getBlog(params.slug);

  return (
    <main>
      <p className="text-center lead fw-bold">Blogs</p>
      <BlogList blogs={blogs} />

      {/* <pre>{JSON.stringify(blogs, null, 4)}</pre> */}
      <div className="d-flex justify-content-center">
        <nav aria-label="Page navigation">
          <ul className="pagination">
            {hasPreviousPage && (
              <li className="page-item">
                <Link
                  className="page-link px-3"
                  href={`/?page=${currentPage - 1}`}
                  as={`/?page=${currentPage - 1}`}
                >
                  Previous
                </Link>
              </li>
            )}
```

```
            {Array.from({ length: totalPages }, (_, index) => {
              const page = index + 1;
              return (
                <li
                  key={page}
                  className={`page-item${
                    currentPage === page ? " active" : ""
                  }`}
                >
                  <Link
                    className="page-link"
                    href={`/?page=${page}`}
                    // use 'as' to avoid interpreting it as a separate
route
                    as={`/?page=${page}`}
                  >
                    {page}
                  </Link>
                </li>
              );
            })}

            {hasNextPage && (
              <li className="page-item">
                <Link
                  className="page-link px-3"
                  href={`/?page=${currentPage + 1}`}
                  as={`/?page=${currentPage + 1}`}
                >
                  Next
                </Link>
              </li>
            )}
          </ul>
        </nav>
      </div>
    </main>
  );
}
```

Loading page (default)

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

```
    );
  }
```

Blog like and unlike API

```
// api/user/blog/like/route

// make sure to use next: {revalidate: 1}
// do not use cache: "no-store"

import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Blog from "@/models/blog";
import { getToken } from "next-auth/jwt";

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

  const _req = await req.json();

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

  try {
    const updated = await Blog.findByIdAndUpdate(
      blogId,
      { $addToSet: { likes: token.user._id } },
      { new: true }
    );

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

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

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

  const _req = await req.json();

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

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

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

Blog like component

```
// components/blog/BlogLike
"use client";
import { useState } from "react";
import { useSession } from "next-auth/react";
import toast from "react-hot-toast";
import { useRouter, usePathname } from "next/navigation";
import { set } from "mongoose";

export default function BlogLike({ blog }) {
  const { data, status } = useSession();
  // console.log("blog", blog);
  const [likes, setLikes] = useState(blog?.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({
            blogId: blog._id,
          }),
        };

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

        const data = await response.json();
        // console.log("blog liked response => ", data);
        setLikes(data.likes);
        toast.success("Blog liked");
        router.refresh(); // only works in server components
      }
    } catch (err) {
      console.log(err);
      toast.error("Error liking blog");
    }
  };

  const handleUnlike = async () => {
    try {
      const options = {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
```

```
        },
        body: JSON.stringify({
          blogId: blog._id,
        }),
      };

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

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

  // 🖤

  return (
    <>
      <small className="pointer">
        <span onClick={handleLike}>❤ {likes?.length} likes</span>
      </small>
    </>
  );
}
```

How to like/unlike in single blog view

```
// app/blog/[slug]/page
<div className="card-footer d-flex justify-content-between">
  <BlogLike blog={blog} />
  <small className="text-muted">Posted {dayjs(blog.updatedAt).fromNow()}
</small>
</div>;

// show blog like/unlike in home page
{
  /* <small>❤️ {blog?.likes?.length} likes</small> */
```

```
}
<BlogLike blog={blog} />;
```

User liked blogs API

```
// api/user/liked-blogs/route
import { NextResponse } from "next/server";
import dbConnect from "@/utils/dbConnect";
import Blog from "@/models/blog";
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-blogs => ", token);

  try {
    // Find blogs that have the user's _id in their likes array
    const likedBlogs = await Blog.find({ likes: token.user._id });

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

Liked blogs page

```
// app/dashboard/user/layout
import Link from "next/link";

export default function UserLayout({ children }) {
  return (
    <>
      <nav className="nav justify-content-center">
        <Link className="nav-link" href="/dashboard/user">
          Dashboard
        </Link>
      </nav>
```

```
      {children}
    </>
  );
}

// Fetch liked blogs, sending Cookie in fetch request headers

import { cookies } from "next/headers";
import ListBlog from "@/components/blog/ListBlog";

async function getLikedBlogs() {
  const nextCookies = cookies();
  const nextAuthSessionToken = nextCookies.get("next-auth.session-token");

  // console.log("nextAuthSessionToken", nextAuthSessionToken);

  const apiUrl = `${process.env.API}/user/liked-blogs`;

  const options = {
    method: "GET",
    // cache: "no-store",
    next: { revalidate: 1 },
    headers: {
      Cookie: `next-auth.session-token=${nextAuthSessionToken?.value}`,
    },
  };

  try {
    const response = await fetch(apiUrl, options);

    if (!response.ok) {
      throw new Error(
        `Failed to fetch: ${response.status} ${response.statusText}`
      );
    }

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

export default async function UserDashboard() {
  const likedBlogs = await getLikedBlogs();

  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p>Liked Blogs</p>
          <br />
          <ListBlog blogs={likedBlogs} />
```

```
        </div>
      </div>
    </div>
  );
}
```

Blogs list for admin

```
<Link className="nav-link" href="/dashboard/admin/blog/list">
  Blogs List
</Link>
```

```
// List of blogs for admin
// app/dashboard/admin/blog/list/page
import Link from "next/link";
import queryString from "query-string";

async function getBlogs(searchParams) {
  const urlParams = {
    page: searchParams.page || 1,
  };

  const searchQuery = queryString.stringify(urlParams);

  const apiUrl = `${process.env.API}/blog?${searchQuery}`;

  const options = {
    method: "GET",
    // cache: "no-store",
    next: { revalidate: 1 },
  };

  try {
    const response = await fetch(apiUrl, options);

    if (!response.ok) {
      throw new Error(
        `Failed to fetch: ${response.status} ${response.statusText}`
      );
    }

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

```
export default async function AdminBlogsList({ searchParams }) {
  const data = await getBlogs(searchParams);
  const { blogs, currentPage, totalPages } = data;

  const hasPreviousPage = currentPage > 1;
  const hasNextPage = currentPage < totalPages;

  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p>Blogs List</p>
          <hr />
          {blogs.map((blog, index) => (
            <div key={blog._id} className="d-flex justify-content-
between">
              <p>{blog.title}</p>
              <Link
                href={`/dashboard/admin/blog/update/${blog.slug}`}
                className="text-danger"
              >
                Update
              </Link>
            </div>
          ))}
        </div>
      </div>

      <div className="d-flex justify-content-center">
        <nav aria-label="Page navigation">
          <ul className="pagination">
            {hasPreviousPage && (
              <li className="page-item">
                <Link
                  className="page-link px-3"
                  href={`?page=${currentPage - 1}`}
                >
                  Previous
                </Link>
              </li>
            )}

            {hasNextPage && (
              <li className="page-item">
                <Link
                  className="page-link px-3"
                  href={`?page=${currentPage + 1}`}
                >
                  Next
                </Link>
              </li>
            )}
          </ul>
        </nav>
```

```
      </div>
    </div>
  );
}
```

Blog update and delete API

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

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

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

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

    if (!updatedBlog) {
      return res.status(404).json({
        error: "Blog not found",
      });
    }

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

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

  await dbConnect();

  try {
    const deletedBlog = await Blog.findByIdAndDelete(context.params.id);
```

```
    if (!deletedBlog) {
      return res.status(404).json({
        error: "Blog not found",
      });
    }

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

Blog update and delete page

```
// app/dashboard/admin/blog/update/[slug]/page.js
"use client";
import { useState, useEffect } from "react";
import { useRouter } from "next/navigation";
import toast from "react-hot-toast";
import dynamic from "next/dynamic";
const ReactQuill = dynamic(() => import("react-quill"), { ssr: false });
import "react-quill/dist/quill.snow.css";

export default function AdminBlogUpdate({ params }) {
  const [id, setId] = useState("");
  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");
  const [category, setCategory] = useState("");
  const [image, setImage] = useState("");
  const [loading, setLoading] = useState(false);

  const router = useRouter();

  useEffect(() => {
    getBlog();
  }, [params]);

  async function getBlog() {
    try {
      const response = await fetch(
        `${process.env.NEXT_PUBLIC_API}/blog/${params.slug}`
      );

      if (!response.ok) {
```

```
          throw new Error("Network response was not ok");
        }

        const data = await response.json();

        setId(data._id);
        setTitle(data.title);
        setContent(data.content);
        setCategory(data.category);
        setImage(data.image);
        setPreview(data.image);
      } catch (error) {
        console.error("Error fetching blog:", error);
        // Handle error state or show a message to the user
      }
    }

    // cloudinary - click on settings icon > preset > unsigned
    const uploadImage = async (e) => {
      const file = e.target.files[0];
      if (file) {
        setLoading(true);
        const formData = new FormData();
        formData.append("file", file);
        formData.append(
          "upload_preset",
          process.env.NEXT_PUBLIC_CLOUDINARY_UPLOAD_PRESET
        ); // replace with your upload_preset

        // upload to cloudinary
        try {
          const response = await
  fetch(process.env.NEXT_PUBLIC_CLOUDINARY_URL, {
            method: "POST",
            body: formData,
          });

          if (response.ok) {
            const data = await response.json();
            setImage(data.secure_url);
          } else {
            console.log("Image upload failed");
          }
        } catch (err) {
          console.log("Error uploading image:", err);
        }

        setLoading(false);
      }
    };

    const updateBlog = async () => {
      try {
        const response = await fetch(
```

```
        `${process.env.NEXT_PUBLIC_API}/admin/blog/${id}`,
        {
          method: "PUT",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify({
            title,
            content,
            category,
            image,
          }),
        }
      );

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

      // router.back();
      window.location.href = "/dashboard/admin/blog/list";
      toast.success("Blog updated successfully");
    } catch (error) {
      console.error("Error updating blog:", error);
      toast.error("Failed to update blog");
    }
  };

  const handleDelete = async () => {
    try {
      const response = await fetch(
        `${process.env.NEXT_PUBLIC_API}/admin/blog/${id}`,
        {
          method: "DELETE",
        }
      );

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

      // router.back();
      window.location.href = "/dashboard/admin/blog/list";
      toast.success("Blog deleted successfully");
    } catch (error) {
      console.error("Error deleting blog:", error);
      toast.error("Failed to delete blog");
    }
  };

  return (
    <div className="container mb-5">
      <div className="row">
        <div className="col">
```

```
            <p>Update blog</p>
            <label className="text-secondary">Blog title</label>
            <input
              type="text"
              value={title}
              onChange={(e) => setTitle(e.target.value)}
              className="form-control p-2 my-2"
            />

            <label className="text-secondary">Blog content</label>
            <ReactQuill
              className="border rounded my-2"
              value={content}
              onChange={setContent}
            />

            <label className="text-secondary">Blog category</label>
            <input
              type="text"
              value={category}
              onChange={(e) => setCategory(e.target.value)}
              className="form-control p-2 my-2"
            />

            {image && (
              <img src={image} alt="preview" style={{ width: "100px" }} />
            )}

            <div className="d-flex justify-content-between mt-3">
              <button className="btn btn-outline-secondary">
                <label className="mt-2" htmlFor="upload-button">
                  {loading ? "Uploading..." : "Upload image"}
                </label>
                <input
                  id="upload-button"
                  type="file"
                  accept="image/*"
                  hidden
                  onChange={uploadImage}
                />
              </button>

              <button className="btn bg-primary text-light" onClick=
{updateBlog}>
                Update
              </button>
            </div>

            <div className="d-flex justify-content-end mt-5">
              <button
                diasabled={loading}
                onClick={handleDelete}
                className="btn btn-sm btn-outline-danger"
              >
```

```
              Delete
            </button>
          </div>
        </div>
      </div>
    );
}
```

Blog search context

```
// context/search.js
"use client";
import { createContext, useState, useContext } from "react";
import { useRouter } from "next/navigation";

export const SearchContext = createContext();

export const SearchProvider = ({ children }) => {
  const [searchQuery, setSearchQuery] = useState("");
  const [searchResults, setSearchResults] = useState([]);

  const router = useRouter();

  const fetchSearchResults = async (e) => {
    e.preventDefault();
    try {
      const response = await fetch(
        `${process.env.API}/search?searchQuery=${searchQuery}`
      );

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

      const data = await response.json();
      setSearchResults(data);
      // console.log("search results => ", data);
      router.push(`/search?searchQuery=${searchQuery}`);
    } catch (error) {
      console.error("Error fetching search results:", error);
    }
  };

  return (
    <SearchContext.Provider
      value={{
        searchQuery,
        setSearchQuery,
        searchResults,
        setSearchResults,
```

```
        fetchSearchResults,
      }}
    >
      {children}
    </SearchContext.Provider>
  );
};

export const useSearch = () => useContext(SearchContext);
```

Search provider

```
// Wrap root layout with search provider
import { SearchProvider } from "@/context/search";

// app/layout
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <SessionProvider>
          <SearchProvider>
            <Toaster position="top-left" />
            <TopNav />
            {children}
          </SearchProvider>
        </SessionProvider>
      </body>
    </html>
  );
}
```

Search form

```
// TopNav
import { useSearch } from "@/context/search";

// ...
const { searchQuery, setSearchQuery, fetchSearchResults } = useSearch();

// ...

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

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

Search API

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

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

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

  try {
    const blogs = await Blog.find({
      $or: [
        { title: { $regex: searchQuery, $options: "i" } }, // Case-
insensitive search on the title field
        { content: { $regex: searchQuery, $options: "i" } }, // Case-
insensitive search on the content field
        { category: { $regex: searchQuery, $options: "i" } }, // Case-
insensitive search on the category field
      ],
    }).sort({ createdAt: -1 });

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

Search page

```
// app/search/page
"use client";
import { useEffect } from "react";
```

```
import BlogList from "@/components/blog/BlogList";
import { useSearchParams } from "next/navigation";
import { useSearch } from "@/context/search";

export default function SearchPage() {
  // context
  const { setSearchQuery, searchResults, setSearchResults } = useSearch();
  // console.log("searchQuery in search page =====> ", searchQuery);

  const searchParams = useSearchParams();
  const query = searchParams.get("searchQuery");

  // to fetch results on page load based on query
  useEffect(() => {
    if (query) {
      console.log(
        "Got search params in search page => ",
        searchParams.get("searchQuery")
      );
      setSearchQuery(query);
      fetchResultsOnLoad(query);
    }
  }, [query]);

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

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

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

  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <p>Search result {searchResults.length}</p>
          {/* <pre>{JSON.stringify(searchResults, null, 4)}</pre> */}
          {searchResults ? <BlogList blogs={searchResults} /> : ""}
        </div>
      </div>
    </div>
  );
}
```

Add Cors (optional)

```
// ...
// add cors support - optional
if (url.startsWith("/api")) {
  NextResponse.next().headers.append("Access-Control-Allow-Origin", "*");
}
// client side protection
if (url?.includes("/admin") && userRole !== "admin") {
  return NextResponse.redirect(new URL("/", req.url));
}
```

Deploy to vercel

```
npm i -g vercel@latest
vercel
vercel --prod
```