**\*\* ./app/search/page.tsx \*\***

import PostList from "@/components/posts/post-list"

import { fetchPostsBySearchTerm } from "@/db/queries/posts"

import { redirect } from "next/navigation"

interface Props {

searchParams: {

term: string

}

}

export default async function SearchPage({ searchParams }: Props) {

const { term } = searchParams

if (!term) {

redirect('/')

}

return <div>

<PostList fetchData={() => fetchPostsBySearchTerm(term)} />

</div>

}

**\*\* ./app/layout.tsx \*\***

import type { Metadata } from 'next'

import { Inter } from 'next/font/google'

import './globals.css'

import { Providers } from './providers'

import { Header } from '@/components/header'

const inter = Inter({ subsets: ['latin'] })

export const metadata: Metadata = {

title: 'Create Next App',

description: 'Generated by create next app',

}

export default function RootLayout({

children,

}: Readonly<{

children: React.ReactNode

}>) {

return (

<html lang='en'>

<body className={inter.className}>

<div className='container mx-auto px-4 max-w-6xl'>

<Providers>

<Header />

{children}

</Providers>

</div>

</body>

</html>

)

}

**\*\* ./app/api/auth/[...nextauth]/route.ts \*\***

export { GET, POST } from '@/auth'

**\*\* ./app/page.tsx \*\***

import PostList from "@/components/posts/post-list";

import { TopicCreateForm } from "@/components/topics/topic-create-form";

import { TopicList } from "@/components/topics/topic-list";

import { fetchTopPosts } from "@/db/queries/posts";

import { Divider } from "@nextui-org/react";

export default function HomePage() {

// need to add time-based revalidation

return <div className="grid grid-cols-4 gap-4 p-4">

<div className="col-span-3">

<h1 className="text-xl m-2">Top Posts</h1>

<PostList fetchData={fetchTopPosts} />

</div>

<div className="border shadow py-3 px-2">

<TopicCreateForm />

<Divider className="my-2" />

<h3 className="text-lg">Topics</h3>

<TopicList />

</div>

</div>

}

**\*\* ./app/topics/[slug]/posts/new/page.tsx \*\***

export default function PostCreatePage() {

return <div>Post Create Page</div>

}

**\*\* ./app/topics/[slug]/posts/[postId]/page.tsx \*\***

import CommentCreateForm from "@/components/comments/comment-create-form";

import CommentList from "@/components/comments/comment-list";

import PostShow from "@/components/posts/post-show";

import { PostShowLoading } from "@/components/posts/post-show-loading";

import { fetchCommentsByPostId } from "@/db/queries/comments";

import { paths } from "@/paths";

import Link from "next/link";

import { Suspense } from "react";

interface PostShowPageProps {

params: {

slug: string;

postId: string;

};

}

export default async function PostShowPage({ params }: PostShowPageProps) {

const { slug, postId } = params;

return (

<div className="space-y-3">

<Link className="underline decoration-solid" href={paths.topicShow(slug)}>

{"< "}Back to {slug}

</Link>

<Suspense fallback={<PostShowLoading />}>

<PostShow postId={postId} />

</Suspense>

<CommentCreateForm postId={postId} startOpen />

{/\* <CommentList fetchData={() => fetchCommentsByPostId(postId)} /> \*/}

<CommentList postId={postId} />

</div>

);

}

**\*\* ./app/topics/[slug]/page.tsx \*\***

import { PostCreateForm } from '@/components/posts/post-create-form'

import PostList from '@/components/posts/post-list'

import { fetchPostsByTopicSlug } from '@/db/queries/posts'

interface Props {

params: {

slug: string

}

}

export default function TopicShowPage({ params }: Props) {

const { slug } = params

return (

<div className='grid grid-cols-4 gap-4 p-4'>

<div className='col-span-3'>

<h1 className='text-2xl font-bold mb-2'>{slug}</h1>

<PostList fetchData={() => fetchPostsByTopicSlug(slug)} />

</div>

<div>

<PostCreateForm slug={slug} />

</div>

</div>

)

}

**\*\* ./app/providers.tsx \*\***

'use client'

import { NextUIProvider } from '@nextui-org/react'

import { SessionProvider } from 'next-auth/react'

interface Props {

children: React.ReactNode

}

export function Providers({ children }: Props) {

return (

<SessionProvider>

<NextUIProvider>{children}</NextUIProvider>

</SessionProvider>

)

}

**\*\* ./components/comments/comment-list.tsx \*\***

import CommentShow from "@/components/comments/comment-show";

import { CommentWithAuthor, fetchCommentsByPostId } from "@/db/queries/comments";

interface CommentListProps {

// fetchData: () => Promise<CommentWithAuthor[]>

postId: string

}

export default async function CommentList({ postId }: CommentListProps) {

// const comments = await fetchData()

const comments = await fetchCommentsByPostId(postId)

const topLevelComments = comments.filter(

(comment) => comment.parentId === null

);

const renderedComments = topLevelComments.map((comment) => {

return (

<CommentShow

key={comment.id}

commentId={comment.id}

// comments={comments}

postId={postId}

/>

);

});

return (

<div className="space-y-3">

<h1 className="text-lg font-bold">All {comments.length} comments</h1>

{renderedComments}

</div>

);

}

**\*\* ./components/comments/comment-show.tsx \*\***

import Image from "next/image";

import { Button } from "@nextui-org/react";

import CommentCreateForm from "@/components/comments/comment-create-form";

import { CommentWithAuthor, fetchCommentsByPostId } from "@/db/queries/comments";

interface CommentShowProps {

commentId: string;

// comments: CommentWithAuthor[]

postId: string

}

export default async function CommentShow({ commentId, postId }: CommentShowProps) {

const comments = await fetchCommentsByPostId(postId)

const comment = comments.find((c) => c.id === commentId);

if (!comment) {

return null;

}

const children = comments.filter((c) => c.parentId === commentId);

const renderedChildren = children.map((child) => {

return (

<CommentShow key={child.id} commentId={child.id} postId={postId} />

);

});

return (

<div className="p-4 border mt-2 mb-1">

<div className="flex gap-3">

<Image

src={comment.user.image || ""}

alt="user image"

width={40}

height={40}

className="w-10 h-10 rounded-full"

/>

<div className="flex-1 space-y-3">

<p className="text-sm font-medium text-gray-500">

{comment.user.name}

</p>

<p className="text-gray-900">{comment.content}</p>

<CommentCreateForm postId={comment.postId} parentId={comment.id} />

</div>

</div>

<div className="pl-4">{renderedChildren}</div>

</div>

);

}

**\*\* ./components/comments/comment-create-form.tsx \*\***

"use client";

import { useFormState } from "react-dom";

import { useEffect, useRef, useState } from "react";

import { Textarea, Button } from "@nextui-org/react";

import { createComment } from "@/actions/create-comment";

import { FormButton } from "../form-button";

interface CommentCreateFormProps {

postId: string;

parentId?: string;

startOpen?: boolean;

}

export default function CommentCreateForm({

postId,

parentId,

startOpen,

}: CommentCreateFormProps) {

const [open, setOpen] = useState(startOpen);

const ref = useRef<HTMLFormElement | null>(null);

const [formState, action] = useFormState(

createComment.bind(null, { postId, parentId }),

{ errors: {} }

);

useEffect(() => {

if (formState.success) {

ref.current?.reset();

if (!startOpen) {

setOpen(false);

}

}

}, [formState, startOpen]);

const form = (

<form action={action} ref={ref}>

<div className="space-y-2 px-1">

<Textarea

name="content"

label="Reply"

placeholder="Enter your comment"

isInvalid={!!formState.errors.content}

errorMessage={formState.errors.content?.join(", ")}

/>

{formState.errors.\_form ? (

<div className="p-2 bg-red-200 border rounded border-red-400">

{formState.errors.\_form?.join(", ")}

</div>

) : null}

<FormButton>Create Comment</FormButton>

</div>

</form>

);

return (

<div>

<Button size="sm" variant="light" onClick={() => setOpen(!open)}>

Reply

</Button>

{open && form}

</div>

);

}

**\*\* ./components/search-input.tsx \*\***

'use client'

import { search } from "@/actions/search"

import { Input } from "@nextui-org/react"

import { useSearchParams } from "next/navigation"

export function SearchInput() {

const searchParams = useSearchParams()

return (

<form action={search}>

<Input name="term" defaultValue={searchParams.get('term') || ""} />

</form>

)

}

**\*\* ./components/posts/post-show-loading.tsx \*\***

import { Skeleton } from "@nextui-org/react";

export function PostShowLoading() {

return (

<div className="m-4">

<div className="my-2">

<Skeleton className="h-8 w-48" />

</div>

<div className="p-4 border rounded space-y-2">

<Skeleton className="h-6 w-32" />

<Skeleton className="h-6 w-32" />

<Skeleton className="h-6 w-32" />

</div>

</div>

)

}

**\*\* ./components/posts/post-create-form.tsx \*\***

'use client'

import { Button, Input, Popover, PopoverContent, PopoverTrigger, Textarea } from "@nextui-org/react"

import { FormButton } from "../form-button"

import { useFormState } from "react-dom"

import { createPost } from "@/actions/create-post"

interface Props {

slug: string

}

export function PostCreateForm({ slug }: Props) {

const [formState, action] = useFormState(createPost.bind(null, slug), { errors: {} })

return (

<Popover placement="left">

<PopoverTrigger>

<Button color="primary">

Create a Post

</Button>

</PopoverTrigger>

<PopoverContent>

<form action={action}>

<div className="flex flex-col gap-4 p-4 w-80">

<h3 className="text-lg">Create a Post</h3>

<Input

name="title"

label="Title"

labelPlacement="outside"

placeholder="Title"

isInvalid={!!formState.errors.title}

errorMessage={formState.errors.title?.join(', ')}

/>

<Textarea

name="content"

label="Content"

labelPlacement="outside"

placeholder="Content"

isInvalid={!!formState.errors.content}

errorMessage={formState.errors.content?.join(', ')}

/>

{formState.errors.\_form ?

<div className="rounded p-2 bg-red-200 border border-red-400">

{formState.errors.\_form.join(', ')}

</div> : null

}

<FormButton>

Create Post

</FormButton>

</div>

</form>

</PopoverContent>

</Popover>

)

}

**\*\* ./components/posts/post-show.tsx \*\***

import { db } from "@/db";

import { notFound } from "next/navigation";

interface PostShowProps {

postId: string

}

export default async function PostShow({postId}: PostShowProps) {

await new Promise(resolve => setTimeout(resolve, 2500))

const post = await db.post.findFirst({

where: { id: postId }

})

if (!post) {

notFound()

}

return (

<div className="m-4">

<h1 className="text-2xl font-bold my-2">{post.title}</h1>

<p className="p-4 border rounded">{post.content}</p>

</div>

);

}

**\*\* ./components/posts/post-list.tsx \*\***

import Link from 'next/link';

import { PostWithData } from '@/db/queries/posts';

import { paths } from '@/paths';

interface Props {

fetchData: () => Promise<PostWithData[]>

}

export default async function PostList({ fetchData}: Props) {

const posts = await fetchData()

const renderedPosts = posts.map((post) => {

const topicSlug = post.topic.slug;

if (!topicSlug) {

throw new Error('Need a slug to link to a post');

}

return (

<div key={post.id} className="border rounded p-2">

<Link href={paths.postShow(topicSlug, post.id)}>

<h3 className="text-lg font-bold">{post.title}</h3>

<div className="flex flex-row gap-8">

<p className="text-xs text-gray-400">By {post.user.name}</p>

<p className="text-xs text-gray-400">

{post.\_count.comments} comments

</p>

</div>

</Link>

</div>

);

});

return <div className="space-y-2">{renderedPosts}</div>;

}

**\*\* ./components/header-auth.tsx \*\***

'use client'

import { signIn, signOut } from "@/actions/auth"

import { Avatar, Button, NavbarItem, Popover, PopoverContent, PopoverTrigger } from "@nextui-org/react"

import { useSession } from "next-auth/react"

export function HeaderAuth() {

const session = useSession()

let authContent: React.ReactNode

if (session.status === 'loading') {

return null

}

if (session.data?.user) {

authContent = <Popover placement='left'>

<PopoverTrigger>

<Avatar src={session.data?.user.image || ''} />

</PopoverTrigger>

<PopoverContent>

<div className='p-4'>

<form action={signOut}>

<Button type="submit">Sign Out</Button>

</form>

</div>

</PopoverContent>

</Popover>

} else {

authContent = (

<>

<NavbarItem>

<form action={signIn}>

<Button

type='submit'

color='secondary'

variant='bordered'>

Sign In

</Button>

</form>

</NavbarItem>

<NavbarItem>

<form action={signIn}>

<Button type='submit' color='primary' variant='flat'>

Sign Up

</Button>

</form>

</NavbarItem>

</>

)

}

return authContent

}

**\*\* ./components/form-button.tsx \*\***

import { Button } from "@nextui-org/react"

import { useFormStatus } from "react-dom"

interface Props {

children: React.ReactNode

}

export function FormButton({ children }: Props) {

const { pending } = useFormStatus()

return <Button type="submit" isLoading={pending}>

{children}

</Button>

}

**\*\* ./components/profile.tsx \*\***

'use client'

import { useSession } from "next-auth/react"

export function Profile() {

const session = useSession()

if (session.data?.user) {

return <div>From client: user is signed in</div>

}

return <div>From client: user is NOT signed in</div>

}

**\*\* ./components/header.tsx \*\***

import {

Input,

Navbar,

NavbarBrand,

NavbarContent,

NavbarItem,

} from '@nextui-org/react'

import Link from 'next/link'

import { HeaderAuth } from './header-auth'

import { SearchInput } from './search-input'

import { Suspense } from 'react'

export function Header() {

return (

<Navbar className='shadow mb-6'>

<NavbarBrand>

<Link href='/' className='font-bold'>

Discuss

</Link>

</NavbarBrand>

<NavbarContent justify='center'>

<NavbarItem>

<Suspense>

<SearchInput />

</Suspense>

</NavbarItem>

</NavbarContent>

<NavbarContent justify='end'>

<HeaderAuth />

</NavbarContent>

</Navbar>

)

}

**\*\* ./components/topics/topic-create-form.tsx \*\***

'use client'

import { createTopic } from '@/actions/create-topic'

import {

Button,

Input,

Popover,

PopoverContent,

PopoverTrigger,

Textarea,

} from '@nextui-org/react'

import { useFormState } from 'react-dom'

import { FormButton } from '../form-button'

export function TopicCreateForm() {

const [formState, action] = useFormState(createTopic, { errors: {} })

return (

<Popover placement='left'>

<PopoverTrigger>

<Button color='primary'>Create a Topic</Button>

</PopoverTrigger>

<PopoverContent>

<form action={action}>

<div className='flex flex-col gap-4 p-4 w-80'>

<h3 className='text-lg'>Create a Topic</h3>

<Input

name='name'

label='Name'

labelPlacement='outside'

placeholder='Name'

isInvalid={!!formState.errors.name}

errorMessage={formState.errors.name?.join(', ')}

/>

<Textarea

name='description'

label='Description'

labelPlacement='outside'

placeholder='Describe your topic'

isInvalid={!!formState.errors.description}

errorMessage={formState.errors.description?.join(', ')}

/>

{formState.errors.\_form ?

<div className='rounded p-2 bg-red-200 border border-red-400'>

{formState.errors.\_form.join(', ')}

</div> : null}

<FormButton>Submit</FormButton>

</div>

</form>

</PopoverContent>

</Popover>

)

}

**\*\* ./components/topics/topic-list.tsx \*\***

import { db } from "@/db"

import { paths } from "@/paths"

import { Chip } from "@nextui-org/react"

import Link from "next/link"

interface Props {

topics: string[]

}

export async function TopicList() {

const topics = await db.topic.findMany()

const renderedTopics = topics.map(topic => {

return (

<div key={topic.id}>

<Link href={paths.topicShow(topic.slug)}>

<Chip color="warning" variant="shadow">

{topic.slug}

</Chip>

</Link>

</div>

)

})

return <div className="flex flex-row flex-wrap gap-2">

{renderedTopics}

</div>

}

**\*\* ./actions/search.ts \*\***

'use server'

import { redirect } from "next/navigation"

export async function search(formData: FormData) {

const term = formData.get('term')

if (typeof term !== 'string' || !term) {

redirect('/')

}

redirect(`/search?term=${term}`)

}

**\*\* ./actions/create-post.ts \*\***

'use server'

import { auth } from "@/auth"

import { db } from "@/db"

import { paths } from "@/paths"

import { Post } from "@prisma/client"

import { revalidatePath } from "next/cache"

import { redirect } from "next/navigation"

import { z } from "zod"

interface CreatePostFormState {

errors: {

title?: string[],

content?: string[],

\_form?: string[]

}

}

export async function createPost(slug: string, formState: CreatePostFormState, formData: FormData): Promise<CreatePostFormState> {

const createPostSchema = z.object({

title: z.string().min(3),

content: z.string().min(10)

})

const result = createPostSchema.safeParse({

title: formData.get('title'),

content: formData.get('content')

})

if (!result.success) {

return { errors: result.error.flatten().fieldErrors }

}

const session = await auth()

if (!session || !session.user) {

return { errors: { \_form: ['You must be signed in to do this']}}

}

const topic = await db.topic.findFirst({

where: { slug }

})

if (!topic) {

return { errors: { \_form: ['Cannot find topic']}}

}

let post: Post

try {

post = await db.post.create({

data: {

title: result.data.title,

content: result.data.content,

userId: session.user.id,

topicId: topic.id

}

})

} catch (err: unknown) {

if (err instanceof Error) {

return { errors: { \_form: [err.message] } }

} else {

return { errors: { \_form: ['Unknown error'] } }

}

}

revalidatePath(paths.topicShow(slug))

redirect(paths.postShow(slug, post.id))

}

**\*\* ./actions/create-topic.ts \*\***

'use server'

import { auth } from '@/auth'

import { db } from '@/db'

import { paths } from '@/paths'

import { Topic } from '@prisma/client'

import { revalidatePath } from 'next/cache'

import { redirect } from 'next/navigation'

import { z } from 'zod'

interface CreateTopicFormState {

errors: {

name?: string[]

description?: string[]

\_form?: string[]

}

}

export async function createTopic(formState: CreateTopicFormState, formData: FormData): Promise<CreateTopicFormState> {

const createTopicSchema = z.object({

name: z

.string()

.min(3)

.regex(/^[a-z-]+$/, {

message: 'Must be lowercase letters or dashes without spaces',

}),

description: z.string().min(10),

})

// await new Promise(resolve => setTimeout(resolve, 2500)) // for testing

const result = createTopicSchema.safeParse({

name: formData.get('name'),

description: formData.get('description')

})

if (!result.success) {

return { errors: result.error.flatten().fieldErrors }

}

const session = await auth()

if (!session || !session.user) {

return { errors: { \_form: ['You must be signed in to do this']}}

}

let topic: Topic

try {

topic = await db.topic.create({

data: {

slug: result.data.name,

description: result.data.description

}

})

} catch (err: unknown) {

if (err instanceof Error) {

return { errors: { \_form: [err.message]}}

} else {

return { errors: { \_form: ['Unknown error']}}

}

}

revalidatePath('/')

redirect(paths.topicShow(topic.slug))

}

**\*\* ./actions/create-comment.ts \*\***

"use server";

import { revalidatePath } from "next/cache";

import { z } from "zod";

import { auth } from "@/auth";

import { db } from "@/db";

import { paths } from "@/paths";

const createCommentSchema = z.object({

content: z.string().min(3),

});

interface CreateCommentFormState {

errors: {

content?: string[];

\_form?: string[];

};

success?: boolean;

}

export async function createComment(

{ postId, parentId }: { postId: string; parentId?: string },

formState: CreateCommentFormState,

formData: FormData

): Promise<CreateCommentFormState> {

const result = createCommentSchema.safeParse({

content: formData.get("content"),

});

if (!result.success) {

return {

errors: result.error.flatten().fieldErrors,

};

}

const session = await auth();

if (!session || !session.user) {

return {

errors: {

\_form: ["You must sign in to do this."],

},

};

}

try {

await db.comment.create({

data: {

content: result.data.content,

postId: postId,

parentId: parentId,

userId: session.user.id,

},

});

} catch (err) {

if (err instanceof Error) {

return {

errors: {

\_form: [err.message],

},

};

} else {

return {

errors: {

\_form: ["Something went wrong..."],

},

};

}

}

const topic = await db.topic.findFirst({

where: { posts: { some: { id: postId } } },

});

if (!topic) {

return {

errors: {

\_form: ["Failed to revalidate topic"],

},

};

}

revalidatePath(paths.postShow(topic.slug, postId));

return {

errors: {},

success: true,

};

}

**\*\* ./actions/auth.ts \*\***

'use server'

import \* as auth from '@/auth'

export async function signIn() {

return auth.signIn('github')

}

export async function signOut() {

return auth.signOut()

}

**\*\* ./db/queries/comments.ts \*\***

import { Comment } from "@prisma/client";

import { db } from "..";

import { cache } from "react";

export type CommentWithAuthor = (Comment &

{ user: {

name: string | null

image: string | null

}

})

function fetchCommentsFn(postId: string): Promise<CommentWithAuthor[]> {

console.log('fetching')

return db.comment.findMany({

where: { postId },

include: {

user: { select: { name: true, image: true }

}}

})

}

export const fetchCommentsByPostId = cache(fetchCommentsFn)

// export const fetchCommentsByPostId = (postId: string): Promise<CommentWithAuthor[]> => {

// console.log('fetching')

// return db.comment.findMany({

// where: { postId },

// include: {

// user: { select: { name: true, image: true }

// }}

// })

// }

**\*\* ./db/queries/posts.ts \*\***

import { Post } from "@prisma/client";

import { db } from "..";

export type PostWithData = (Post & {

topic: { slug: string }

user: { name: string | null }

\_count: { comments: number }

})

export function fetchPostsBySearchTerm(term: string): Promise<PostWithData[]> {

return db.post.findMany({

include: {

topic: { select: { slug: true } },

user: { select: { name: true, image: true }},

\_count: { select: { comments: true }}

},

where: {

OR: [

{ title: { contains: term }},

{ content: { contains: term }}

]

}

})

}

export function fetchPostsByTopicSlug(slug: string): Promise<PostWithData[]> {

return db.post.findMany({

where: { topic: { slug }},

include: {

topic: { select: { slug: true }},

user: { select: { name: true }},

\_count: { select: { comments: true }}

}

})

}

export function fetchTopPosts(): Promise<PostWithData[]> {

return db.post.findMany({

orderBy: [

{ comments: { \_count: "desc"} }

],

include: {

topic: { select: { slug: true }},

user: { select: { name: true }},

\_count: { select: { comments: true }}

},

take: 5

})

}

**\*\* ./db/index.ts \*\***

import { PrismaClient } from "@prisma/client";

export const db = new PrismaClient()

**\*\* ./paths.ts \*\***

export const paths = {

home() { return '/'},

topicShow(topicSlug: string) {

return `/topics/${topicSlug}`

},

postCreate(topicSlug: string) {

return `/topics/${topicSlug}/posts/new`

},

postShow(topicSlug: string, postId: string) {

return `/topics/${topicSlug}/posts/${postId}`

}

}

**\*\* ./auth.ts \*\***

import GitHub from "@auth/core/providers/github"

import { PrismaAdapter } from "@auth/prisma-adapter"

import NextAuth from "next-auth"

import { db } from "./db"

const clientId = process.env.GITHUB\_CLIENT\_ID

const clientSecret = process.env.GITHUB\_CLIENT\_SECRET

if (!clientId || !clientSecret) {

throw new Error('Missing github oath credentials')

}

export const { handlers: { GET, POST }, auth, signOut, signIn } = NextAuth({

adapter: PrismaAdapter(db),

providers: [

GitHub({ clientId, clientSecret })

],

callbacks: {

// To fix a current bug in nextauth (should be removed eventually)

async session({ session, user }: any) {

if (session && user) {

session.user.id = user.id

}

return session

}

}

})