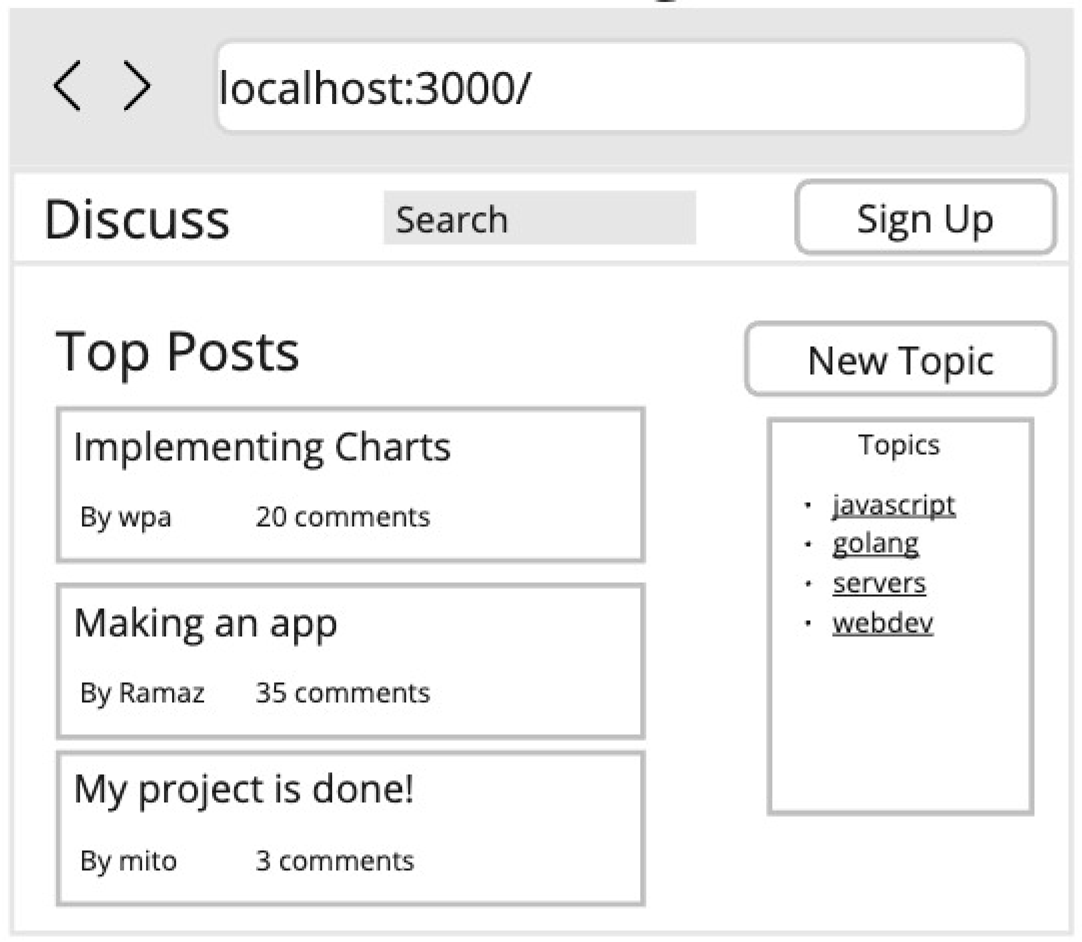
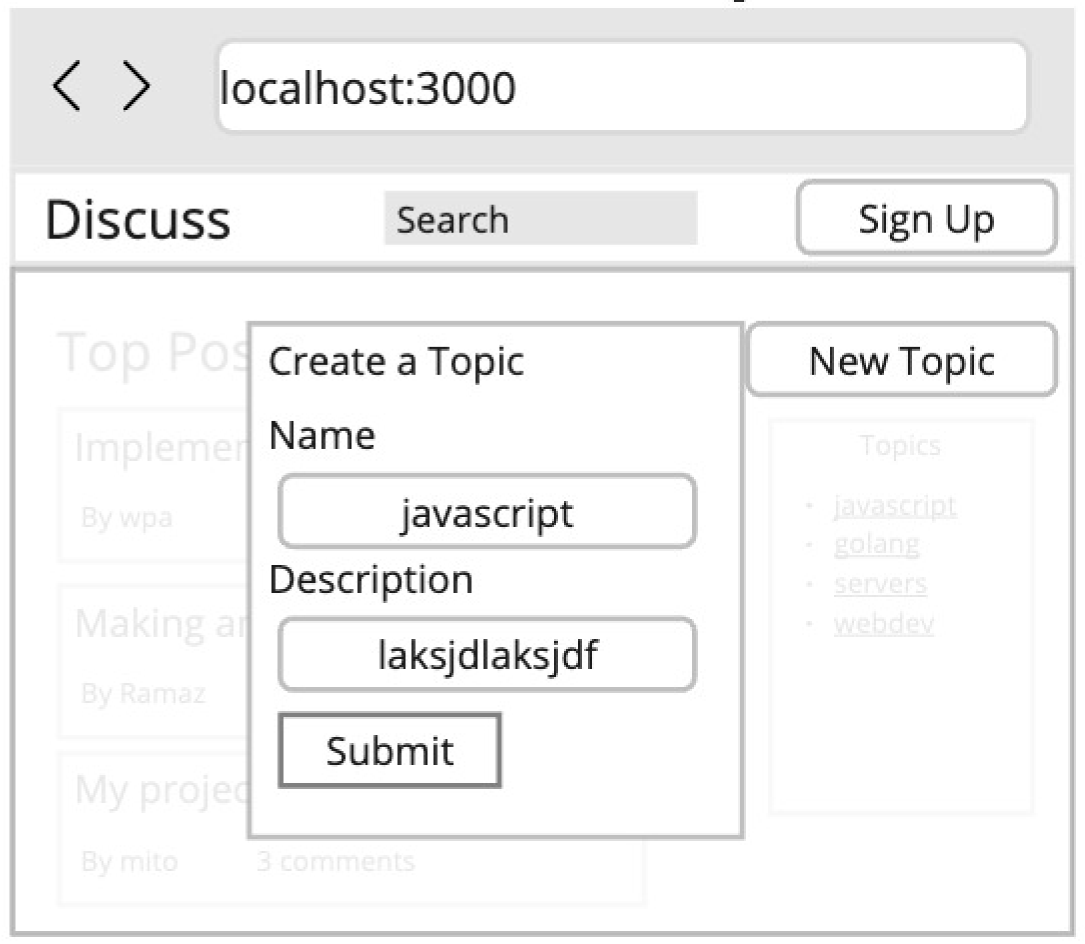
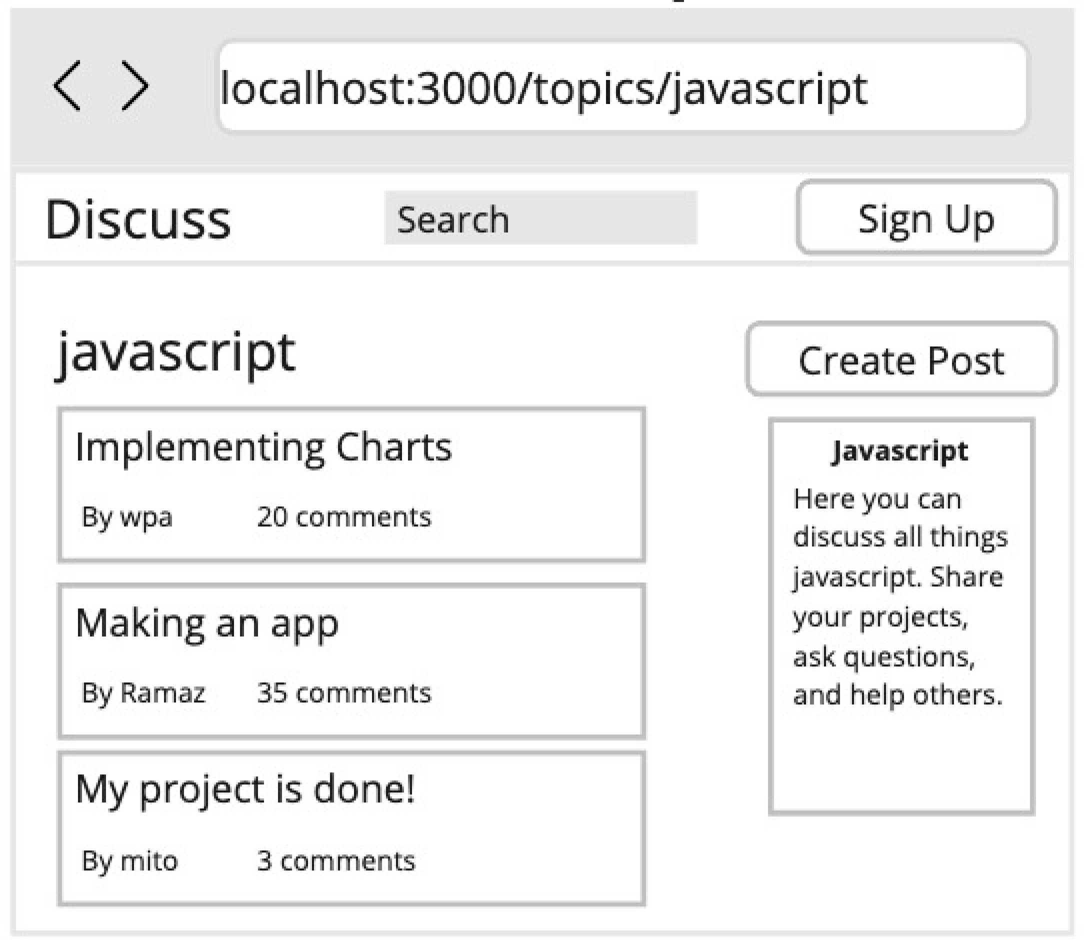
Home Page



Create a Topic

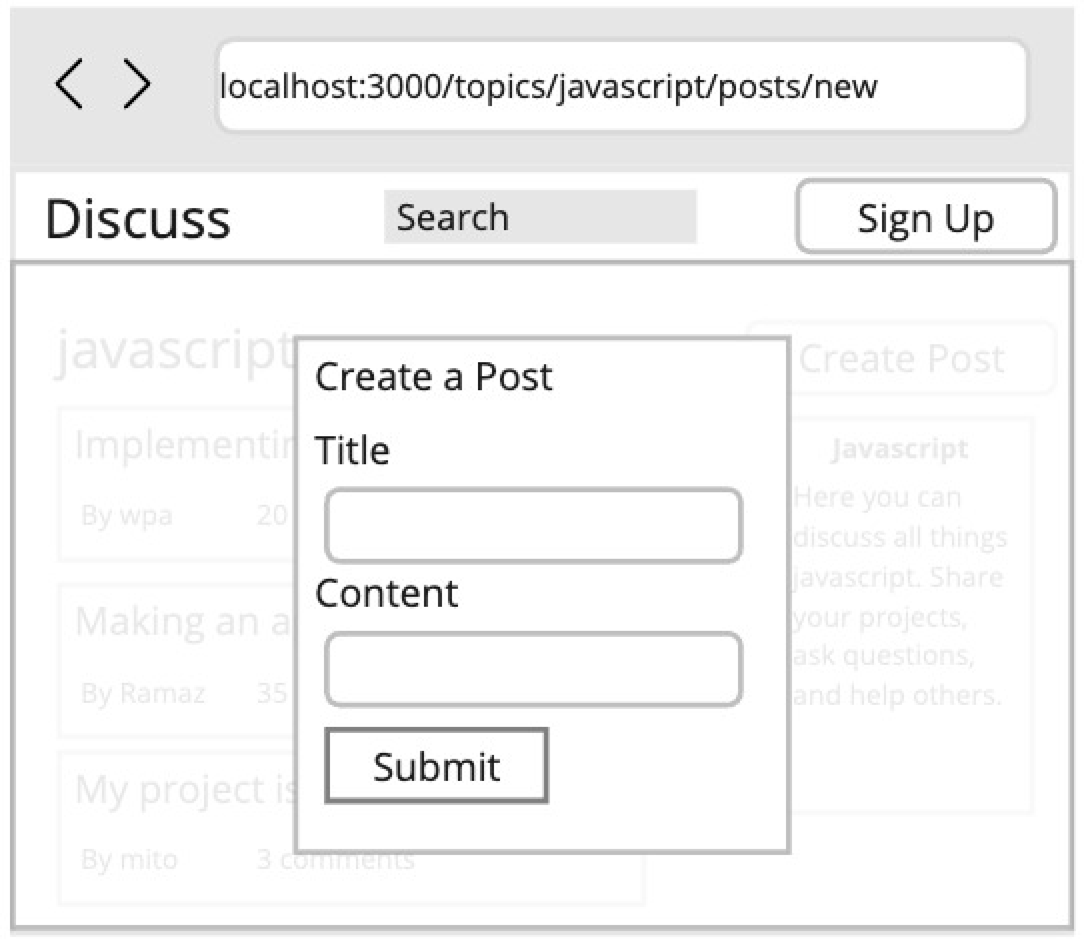


View a Topic

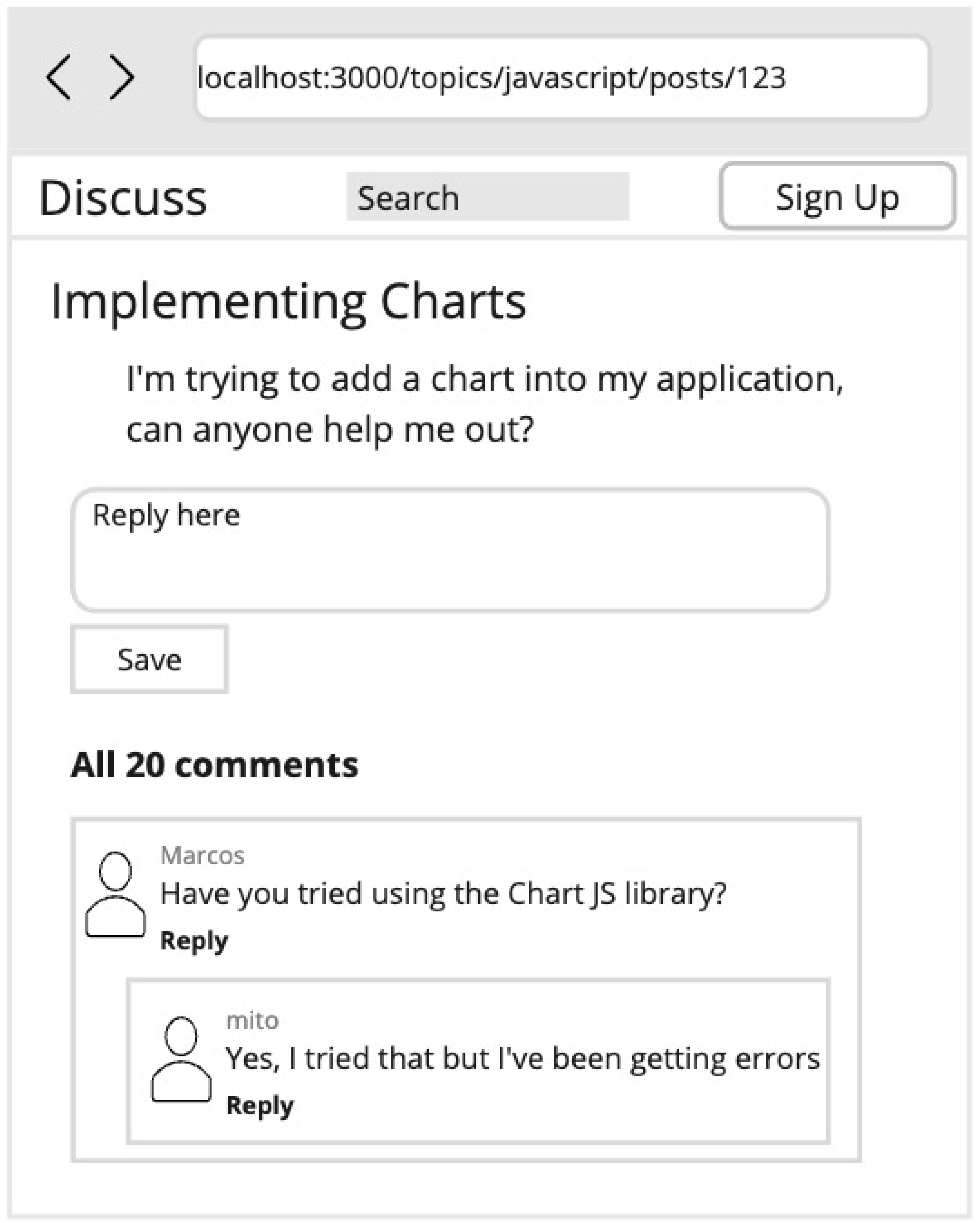


Here you can discuss all things javascript. Share your projects, ask questions, and help others.

Create a Post



View a Post

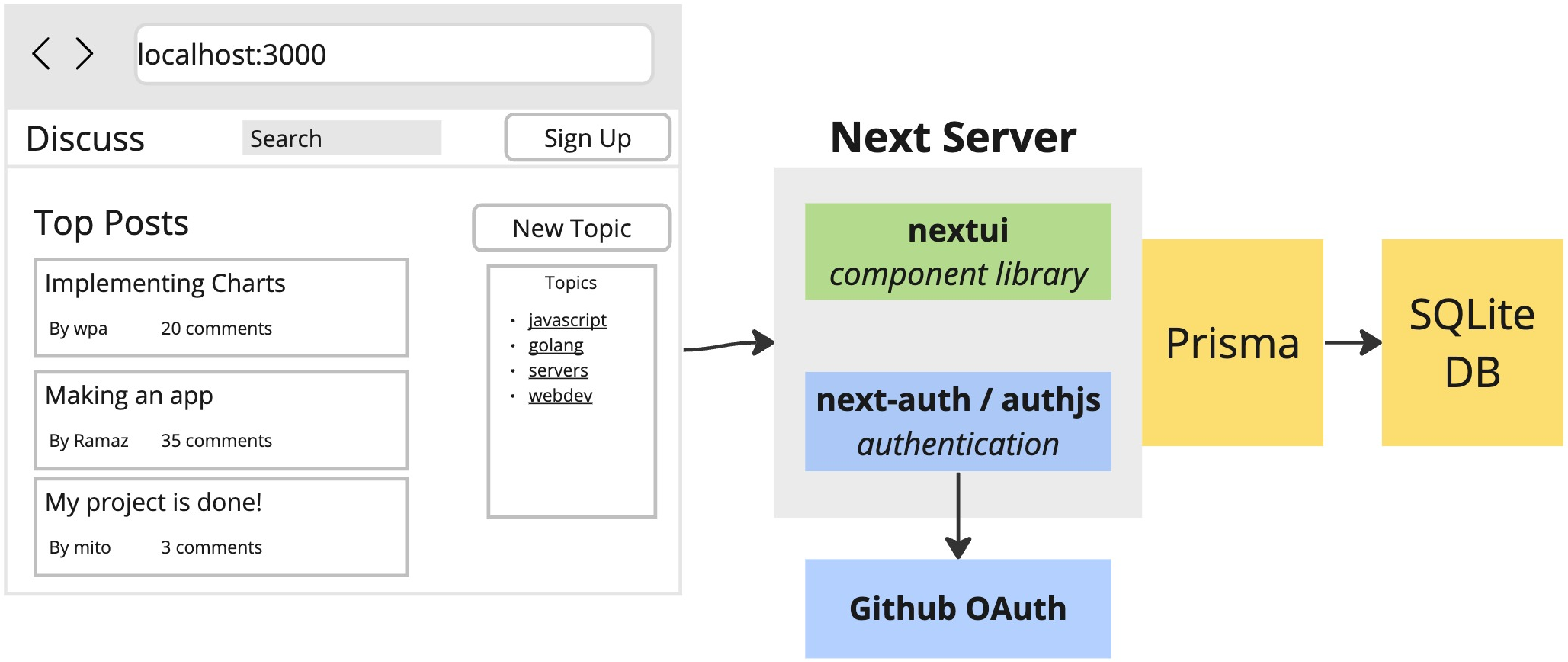


Implementing Charts

I'm trying to add a chart into my application, can anyone help me out?

Have you tried using the Chart JS library?

Yes, I tried that but I've been getting errors


localhost:3000
Discuss
Search
Sign Up
Top Posts
New Topic
Implementing Charts
By wpa
20 comments
Making an app
By Ramaz
35 comments
My project is done!
By mito
3 comments
Topics
javascript
golang
servers
webdev
Next Server
nextui
component library
next-auth / authjs
authentication
Github OAuth
Prisma
SQLite DB

Project Design

Always important to do some upfront thinking about how to design your project

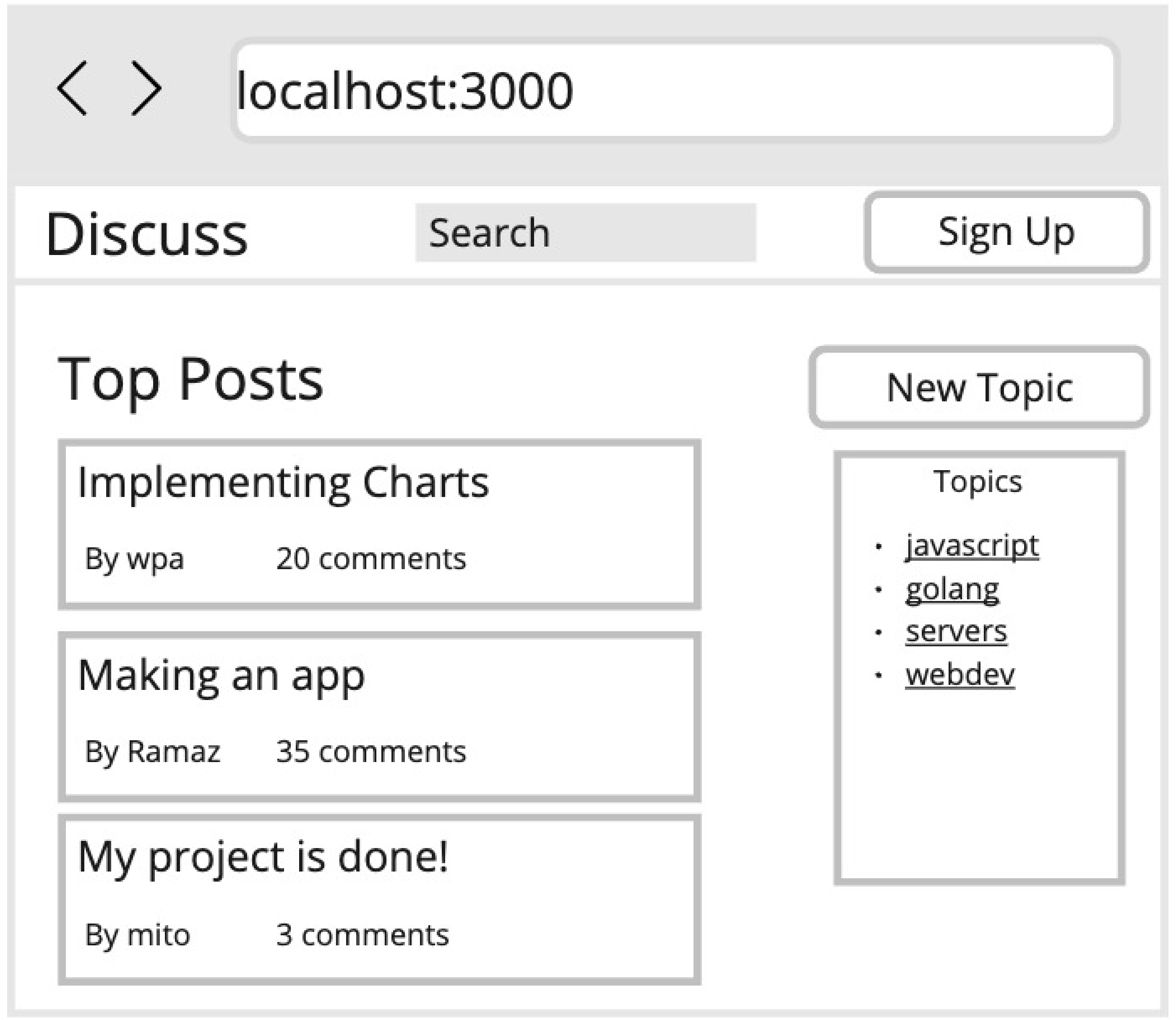
With Next, it is **twice as important**

Trying to solve caching issues later in your project is challenging! Easier up front!

Scaffold your different page.tsx files so you know what routes exist immediately

Upfront Design for Caching

1. Identify the different types of data in your app that will be changing over time
2. Find the different places where this data can change


localhost:3000

Discuss | Search | Sign Up

Top Posts | New Topic

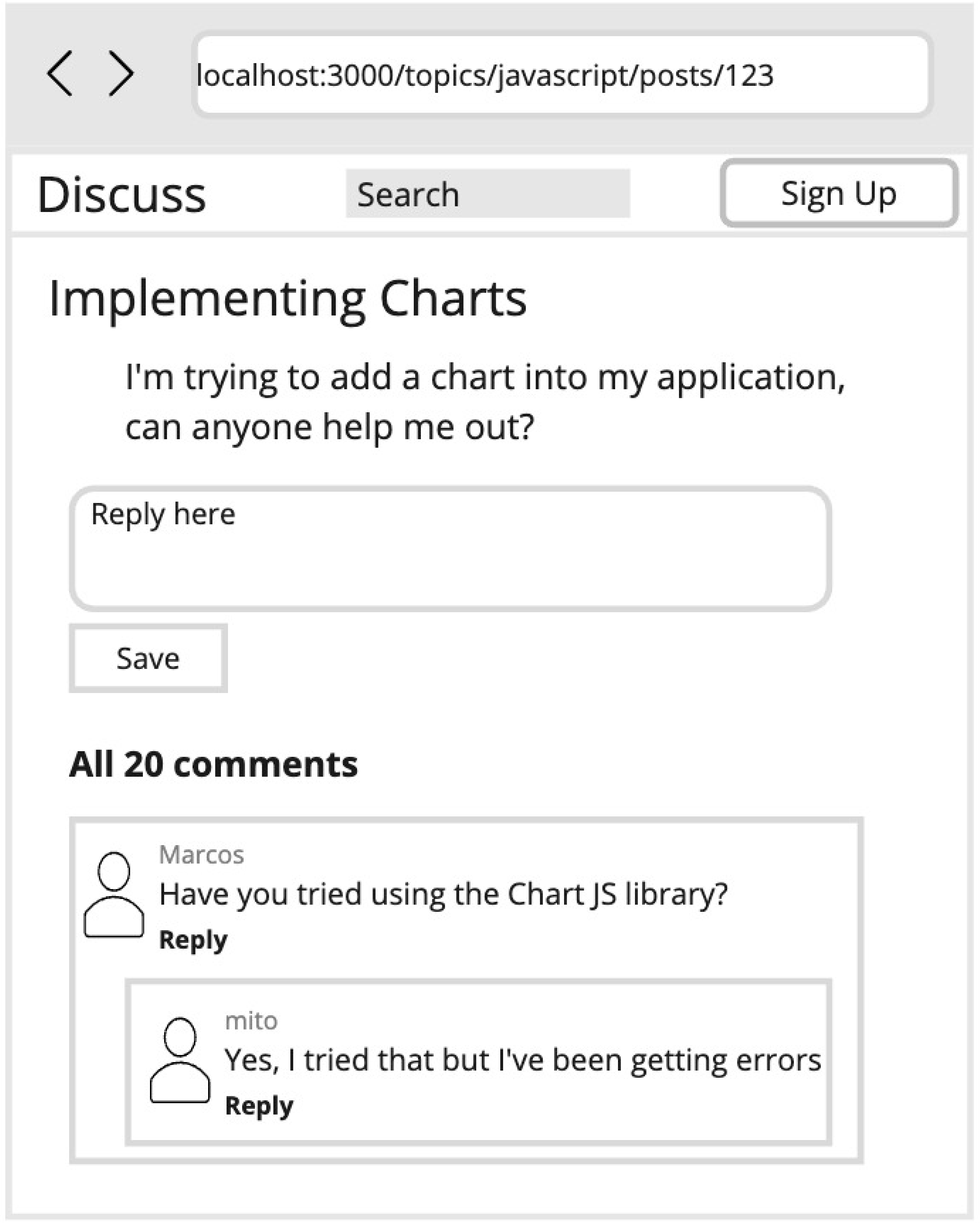
Implementing Charts
By wpa 20 comments

Making an app
By Ramaz 35 comments

My project is done!
By mito 3 comments

Topics
- javascript
- golang
- servers
- webdev

Types of Data


localhost:3000/topics/javascript/posts/123

Discuss | Search | Sign Up

Implementing Charts

I'm trying to add a chart into my application, can anyone help me out?

Reply here

Save

All 20 comments

Marcos
Have you tried using the Chart JS library?
Reply

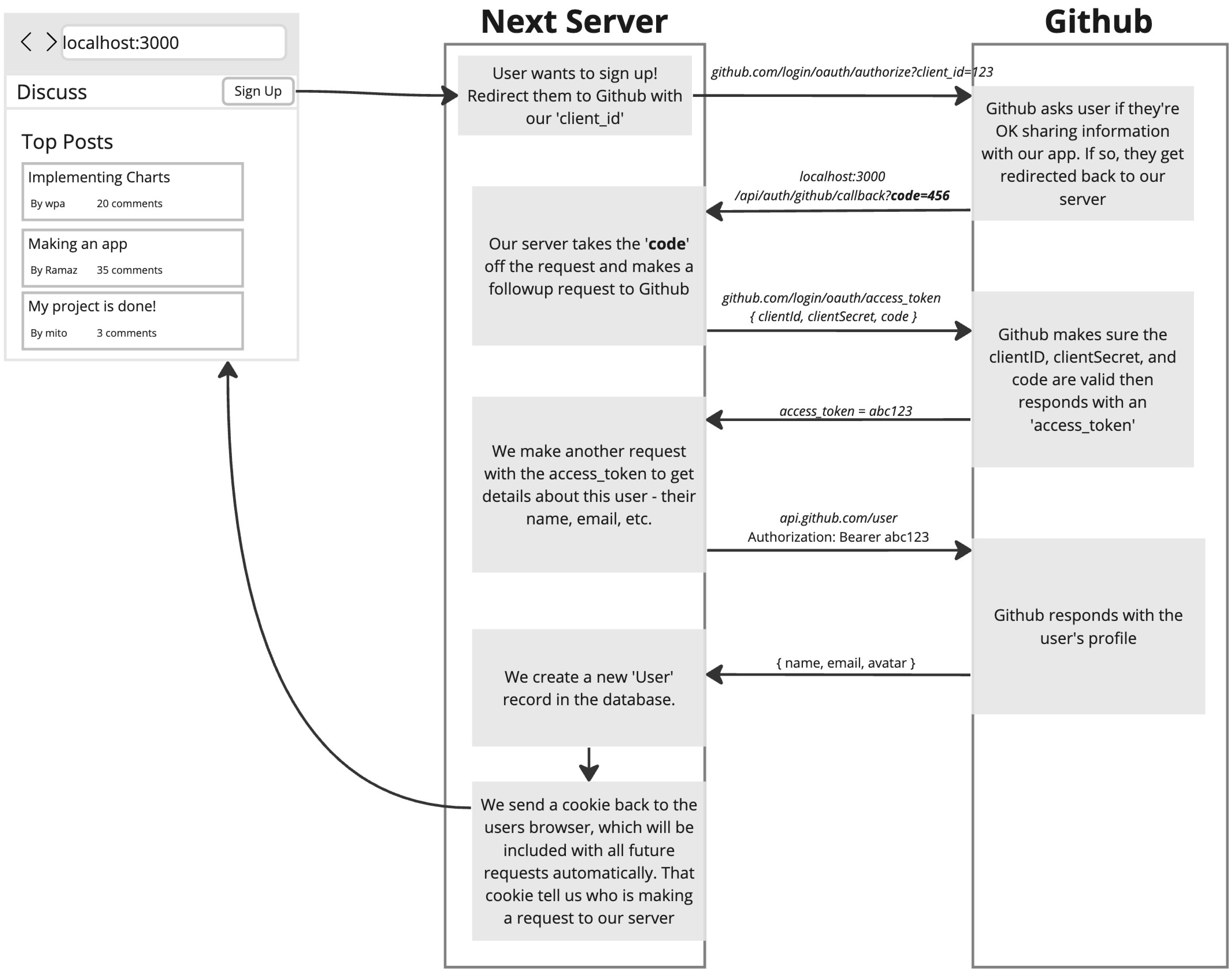
mito
Yes, I tried that but I've been getting errors
Reply

NextUI Setup

nextui.org/docs/frameworks/nextjs

Auth Setup

1	**github.com/settings/applications/new** Create an OAuth app and generate a client_id and client_secret
2	Add **AUTH_SECRET, GITHUB_CLIENT_ID, GITHUB_CLIENT_SECRET** to a .env.local file
3	Install these packages: **@auth/core@0.18.1 @auth/prisma-adapter@1.0.6 next-auth@5.0.0-beta.3**
4	Make a 'auth.ts' file in the 'src' folder. Set up NextAuth and the PrismaAdapter in there
5	Set up the 'app/api/auth/[...nextauth]/route.ts' file to handle the requests between Githubs servers and ours
6	Make server actions to signin/signout the user (Optional, but highly recommended)


localhost:3000
Discuss
Sign Up
Top Posts
Implementing Charts
By wpa
20 comments
Making an app
By Ramaz
35 comments
My project is done!
By mito
3 comments
Next Server
Github
User wants to sign up! Redirect them to Github with our 'client_id'
github.com/login/oauth/authorize?client_id=123
Github asks user if they're OK sharing information with our app. If so, they get redirected back to our server
localhost:3000 /api/auth/github/callback?**code=456**
Our server takes the '**code**' off the request and makes a followup request to Github
github.com/login/oauth/access_token { clientId, clientSecret, code }
Github makes sure the clientID, clientSecret, and code are valid then responds with an 'access_token'
access_token = abc123
We make another request with the access_token to get details about this user - their name, email, etc.
api.github.com/user
Authorization: Bearer abc123
Github responds with the user's profile
{ name, email, avatar }
We create a new 'User' record in the database.
We send a cookie back to the users browser, which will be included with all future requests automatically. That cookie tell us who is making a request to our server

Sign In / Sign Up

```tsx
import * as actions from '@/actions';

export default function Page() {
    return <form action={actions.signIn}>
        <button type="submit">Sign Up</button>
    </form>
}
```

Sign Out

```tsx
import * as actions from '@/actions';

export default function Page() {
    return <form action={actions.signOut}>
        <button type="submit">Sign Out</button>
    </form>
}
```

See if a user is signed in from a Server Component

```tsx
import { auth } from '@/auth';

export default async function Page() {
    const session = await auth();

    if (session?.user) {
        return <div>Signed In!</div>
    } else {
        return <div>Signed out</div>
    }
}
```

See if a user is signed in from a Client Component

Requires a 'SessionProvider' to be set up in the 'providers.tsx' file

```tsx
'use client'

import { useSession } from 'next-auth/react';

export default function Profile() {
    const session = useSession();

    if (session.data?.user) {
        return <div>Signed In!</div>
    } else {
        return <div>Signed out</div>
    }
}
```

Recommended Initial Design

1. Identify all the different routes you want your app to have + the data that each shows
2. Make 'path helper' functions
3. Create your routing folders + page.tsx files based on step #1
4. Identify the places where data changes in your app
5. Make empty server actions for each of those
6. Add in comments on what paths you'll need to revalidate for each server action

Caching

Next implements caching in several locations.

Can lead to unexpected behavior

Data Cache	Responses from requests made with '**fetch**' are stored and used across requests.
Router Cache	'Soft' navigation between routes are cached in the browser and reused when a user revisits a page.
Request Memoization	Make two or more 'GET' requests with 'fetch' during a user's request to your server? Only one 'GET' is actually executed.
Full Route Cache	**At build time**, Next decides if your route is ***static*** or ***dynamic***. If it is static, the page is rendered and the result is stored. In production, users are given this pre-rendered result.

Time-Based → Every X seconds, ignore the cached response and fetch new data

Every 3 seconds the next request to this route will trigger a rerender

```
export const revalidate = 3;

export default async function Page() {
    const snippets = await db
        .snippets.findMany();

    return <div>{snippets.map(..)}</div>
}
```

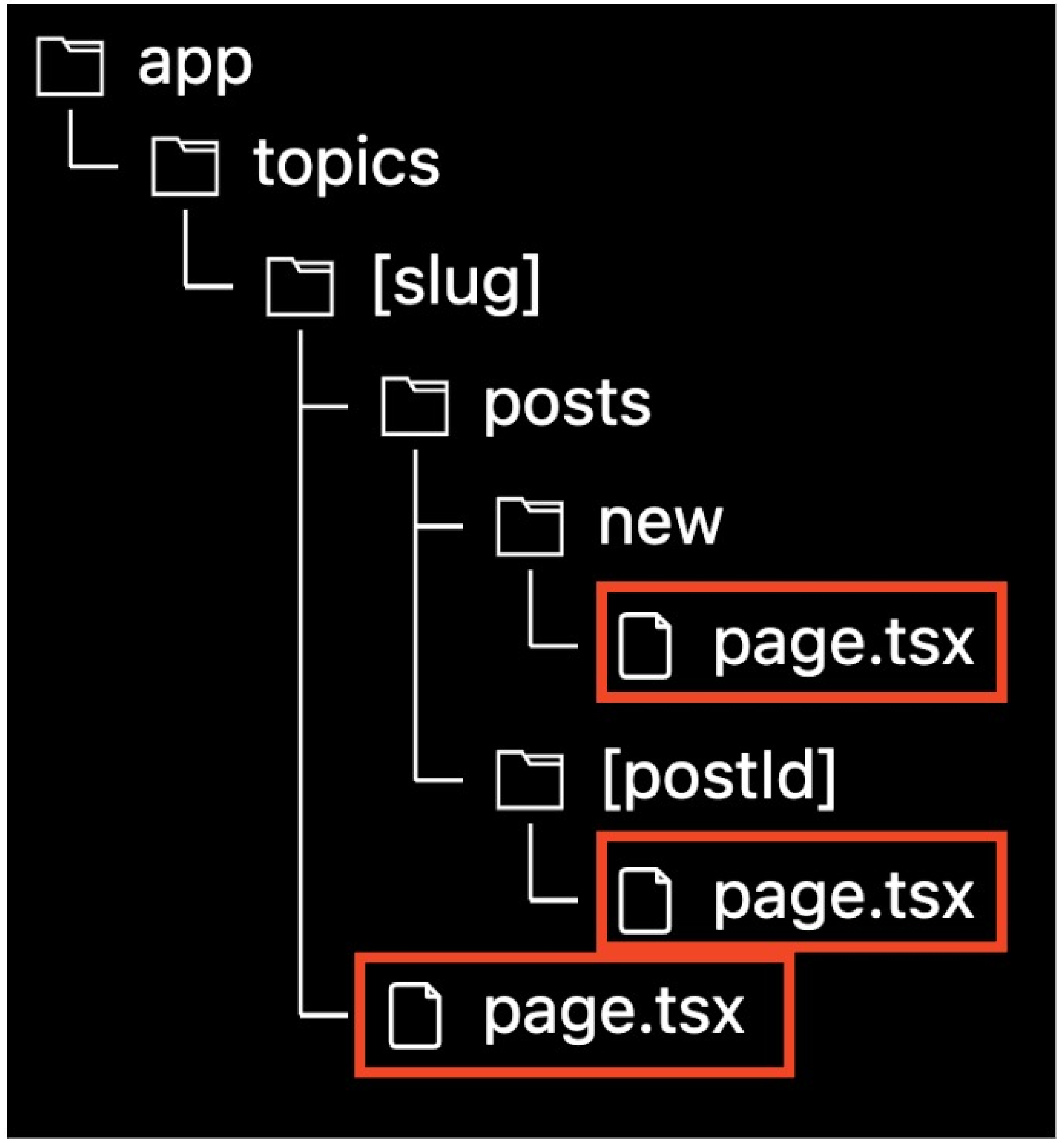
On-Demand → Forcibly purge a cached response

Dump cache for everything in a page

```js
import { revalidatePath } from "next/cache";

// When we think data that the '/snippets'
// route uses has changed...
revalidatePath('/snippets');
```

Page Name	Path	Data shown
Home Page	/	Many posts, many topics
Topic Show	/topics/[slug]	A single and many posts
Create a post	/topics/[slug]/posts/new	A single topic and many posts
Show a post	/topics/[slug]/posts/[postId]	A single post and many comments



In Some Component

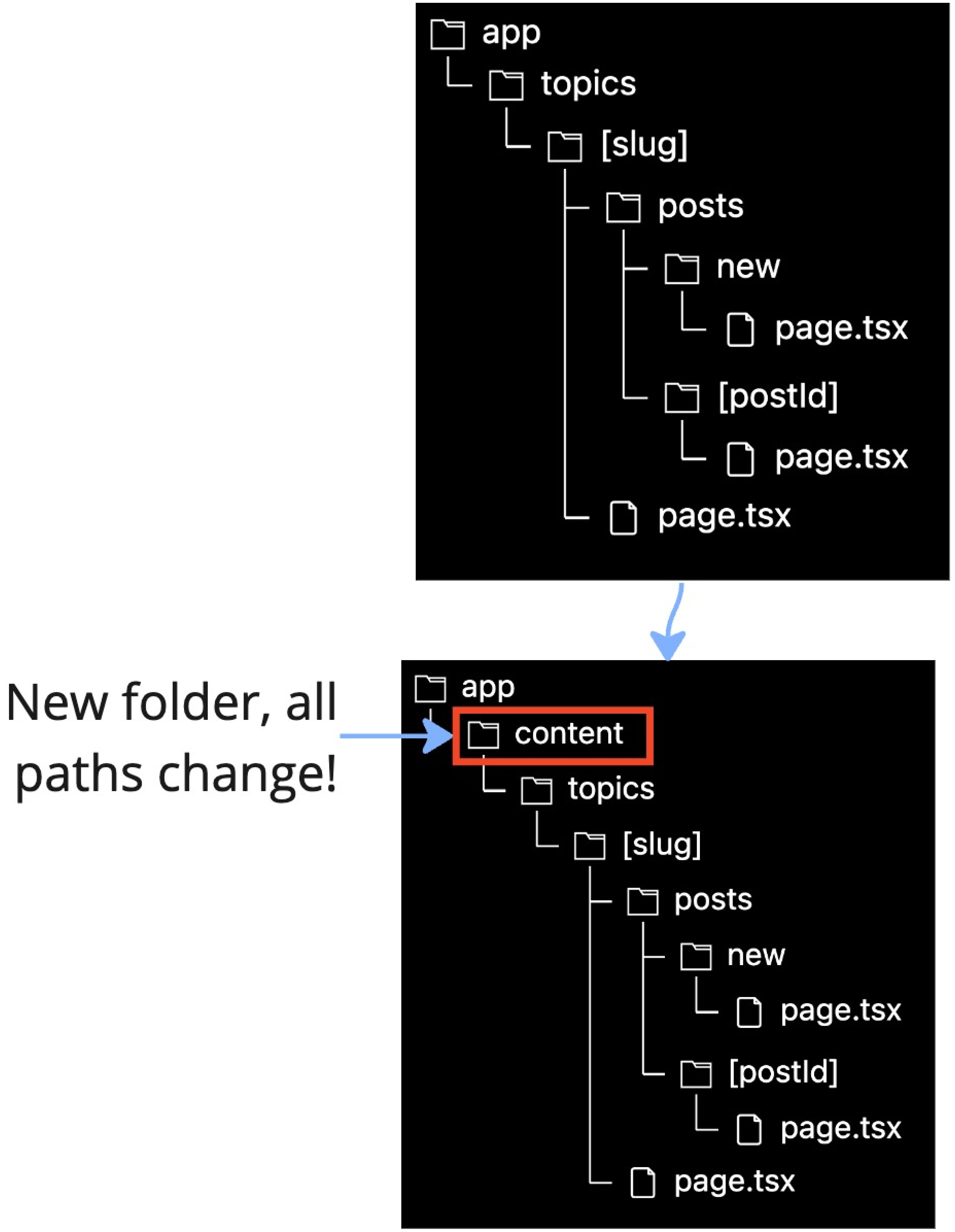
```
<Link href={`/topics/${topic.slug}/posts/new`}>
    Create
</Link>
```

In Some Other Component

```
<Link href={`/topics/${topic.slug}/posts/${postId}`}>
    View
</Link>
```

In a Server Action

```
revalidatePath(`/topics/${topic.slug}`);
```



```
app
└ topics
  └ [slug]
    ├ posts
    │ ├ new
    │ │ └ page.tsx
    │ └ [postId]
    │   └ page.tsx
    └ page.tsx
```

New folder, all paths change!

```
app
└ content
  └ topics
    └ [slug]
      ├ posts
      │ ├ new
      │ │ └ page.tsx
      │ └ [postId]
      │   └ page.tsx
      └ page.tsx
```

In Some Component

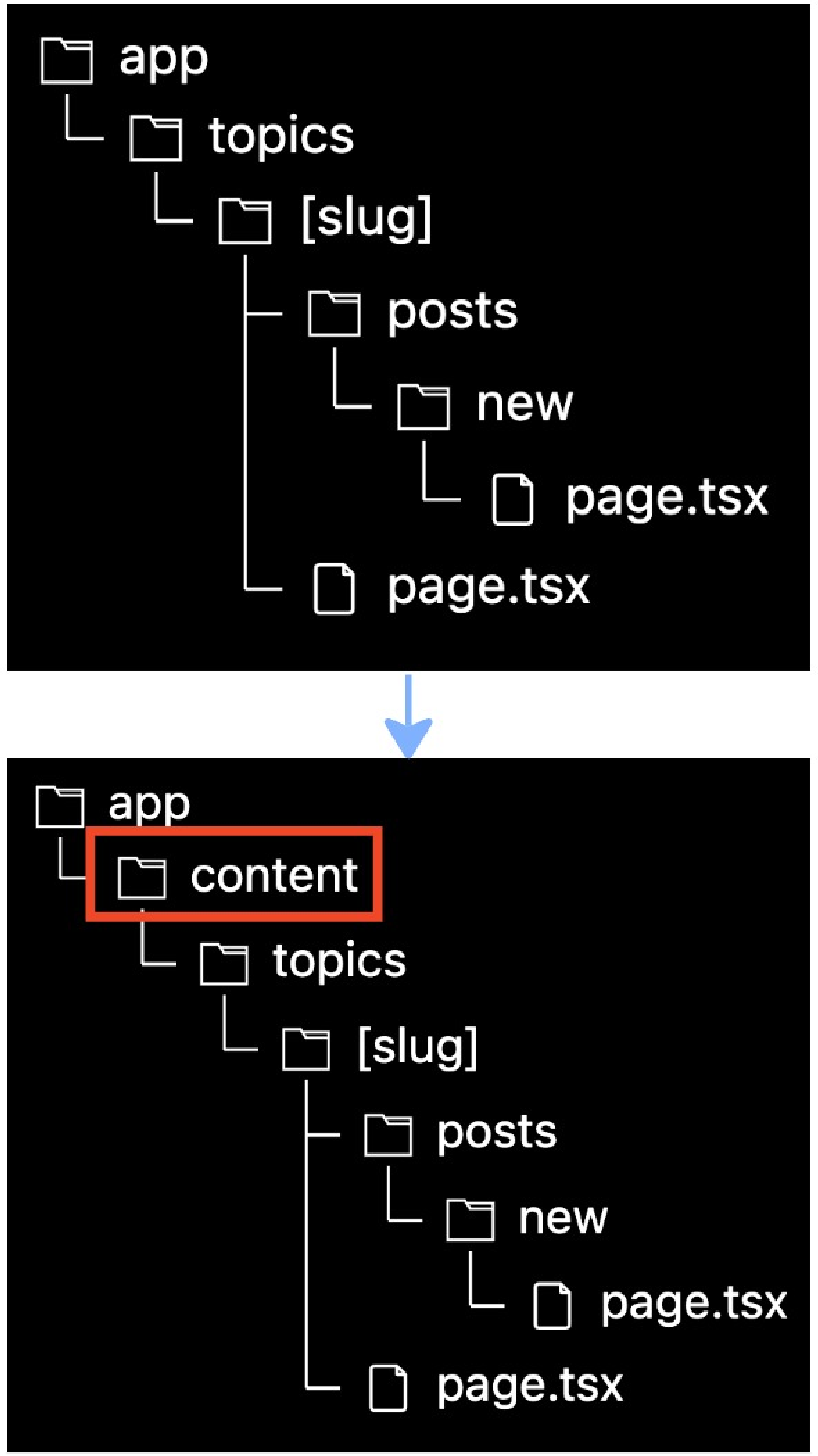
```
<Link href={`/content/topics/${topic.slug}/posts
    /new`}>
    Create
</Link>
```

In Some Other Component

```
<Link href={`/content/topics/${topic.slug}/posts
    /${postId}`}>
    View
</Link>
```

In a Server Action

```
revalidatePath(`/content/topics/${topic.slug}`);
```



```
app
└ topics
   └ [slug]
      ├ posts
      │  └ new
      │     └ page.tsx
      └ page.tsx
```

```
app
└ content
   └ topics
      └ [slug]
         ├ posts
         │  └ new
         │     └ page.tsx
         └ page.tsx
```

Path Helpers

```
const paths = {
    homePage() {
        return '/'
    },
    topicShowPath(slug: string) {
        return `/content/topics/${slug}`
    },
    postCreatePath(slug: string) {
        return `/content/topics/${slug}/posts
            /new`
    },
    postShowPath(slug: string, postId: string) {
        return `/content/topics/${slug}/posts
            /${postId}`
    }
}
```

In Some Component

```
<Link href={paths.postCreatePath(topic.slug)}>
    Create
</Link>
```

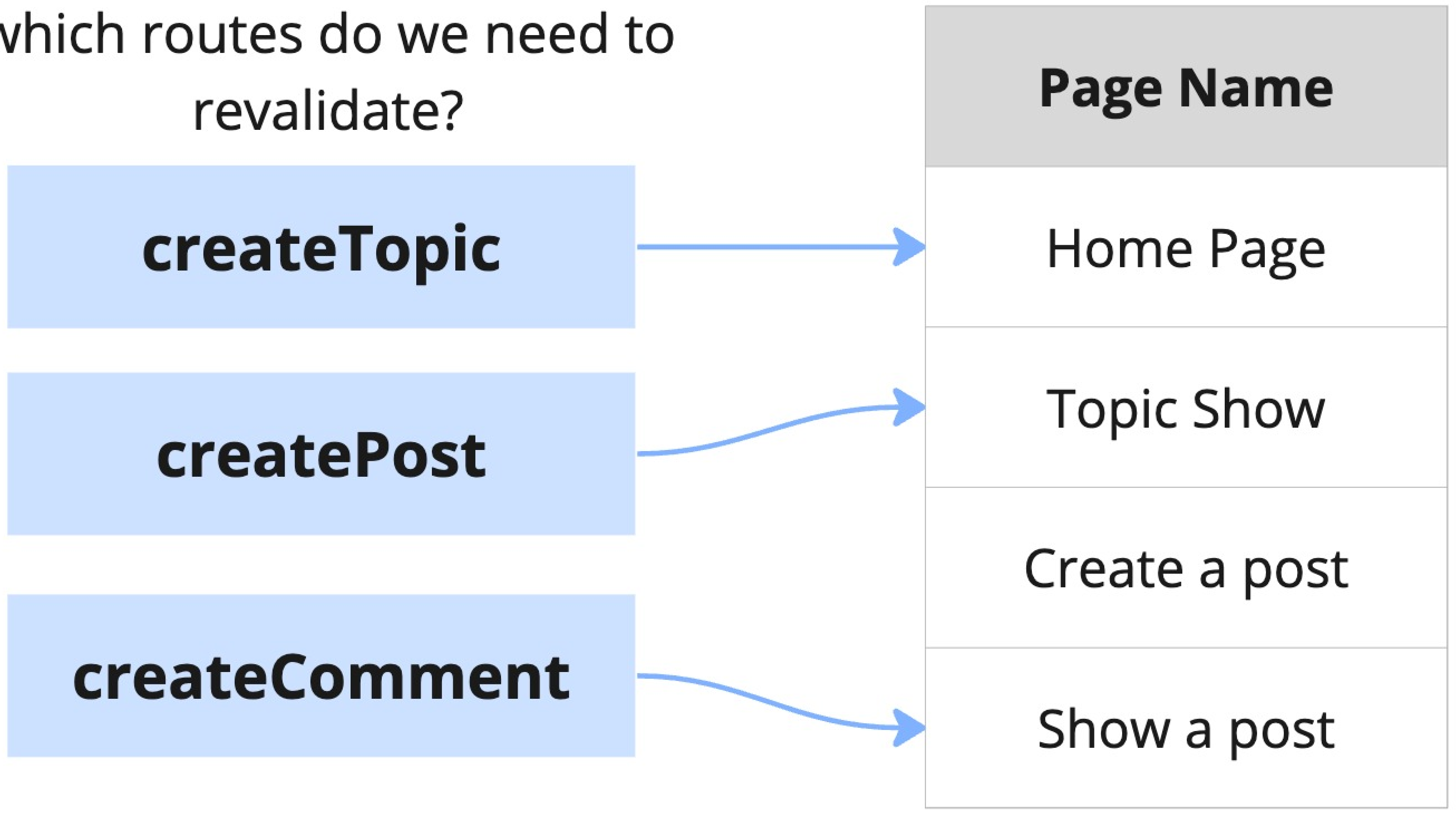
In Some Other Component

```
<Link href={`/topics/${topic.slug}/posts/${postId}`}>
    View
</Link>
```

In a Server

```
revalidatePath(`/topics/${topic.slug}`);
```

When we call each server action which routes do we need to revalidate?


createTopic
createPost
createComment
Page Name
Home Page
Topic Show
Create a post
Show a post