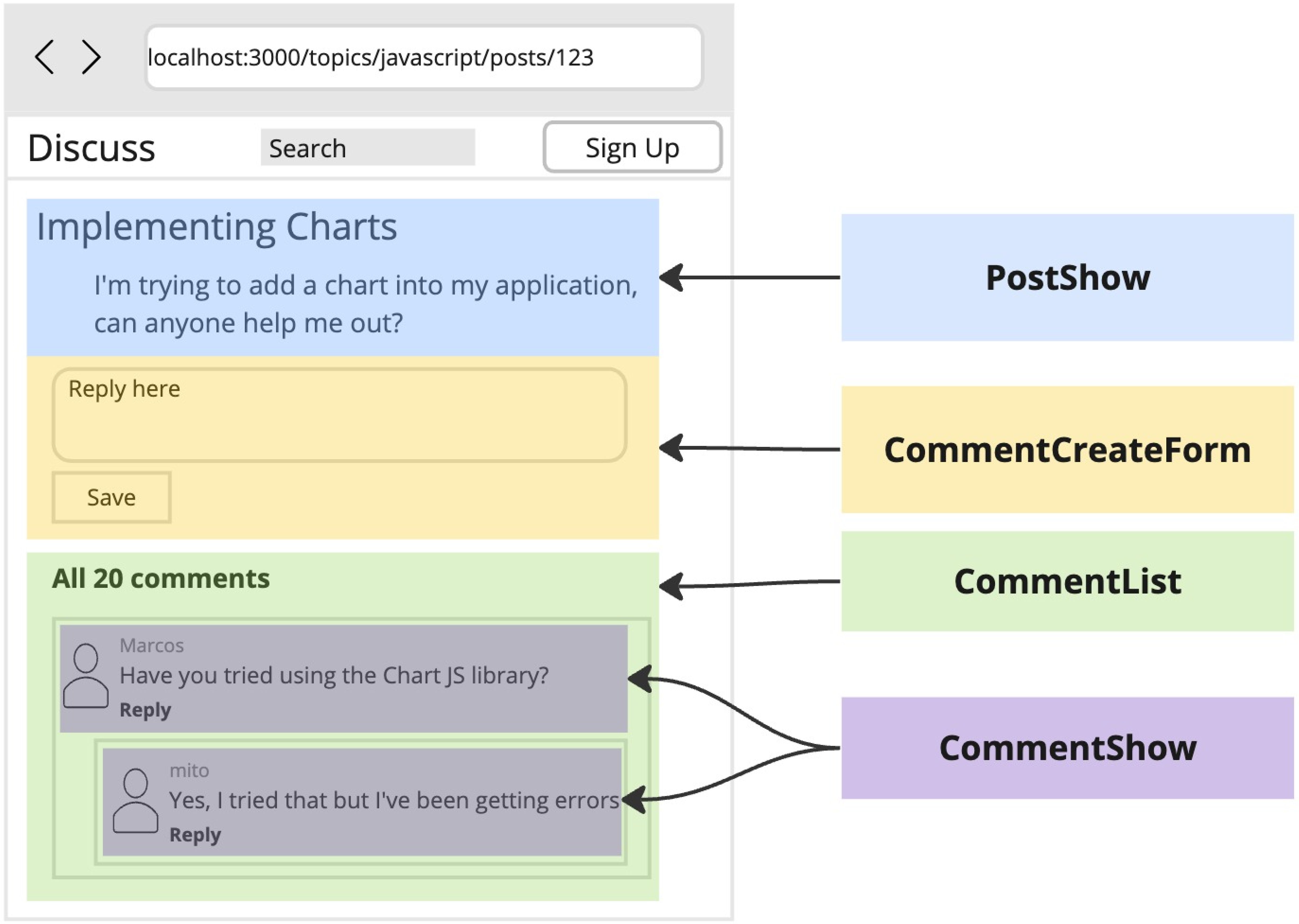
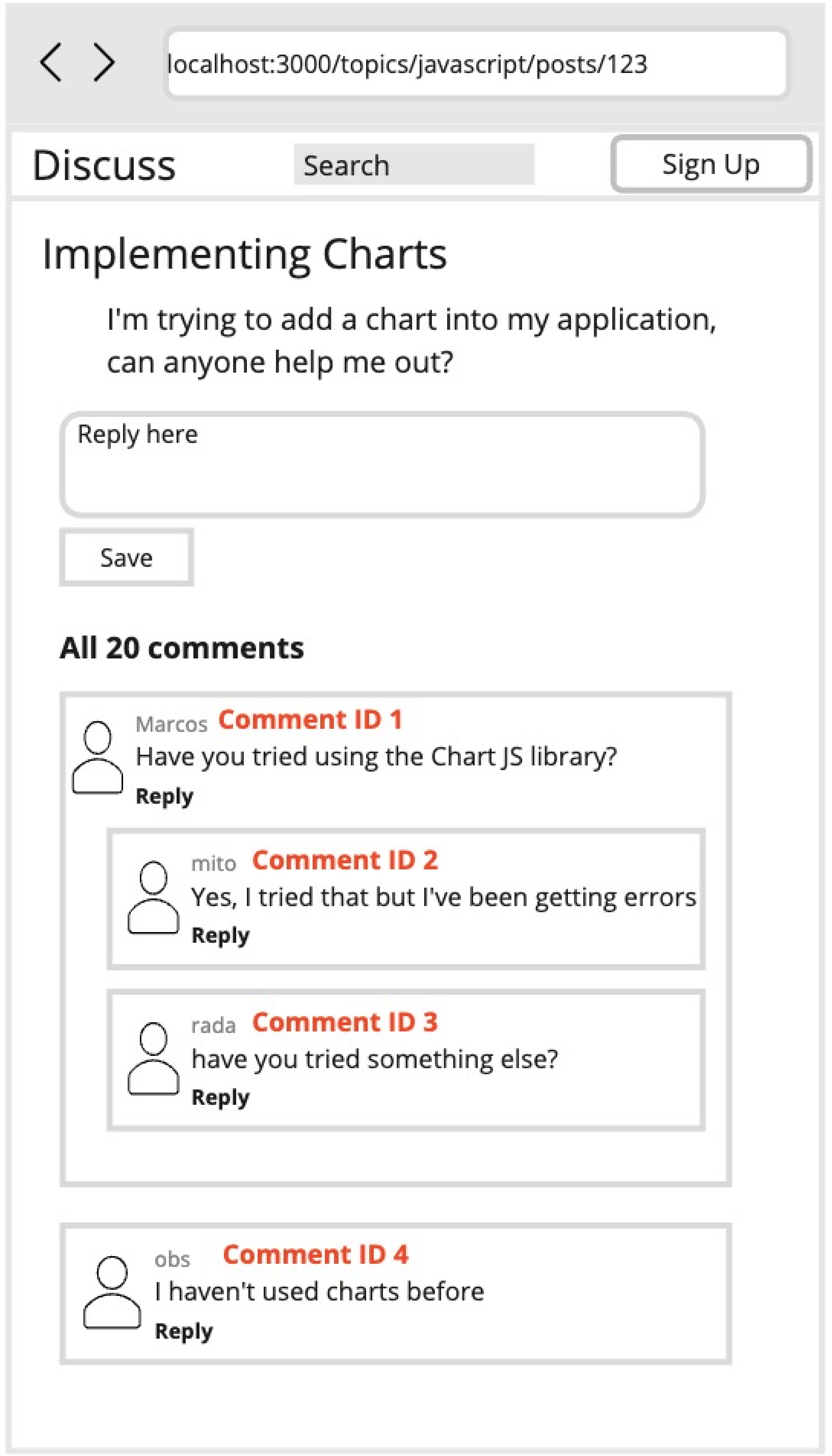
View a Post


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
PostShow
CommentCreateForm
CommentList
CommentShow

View a Post


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

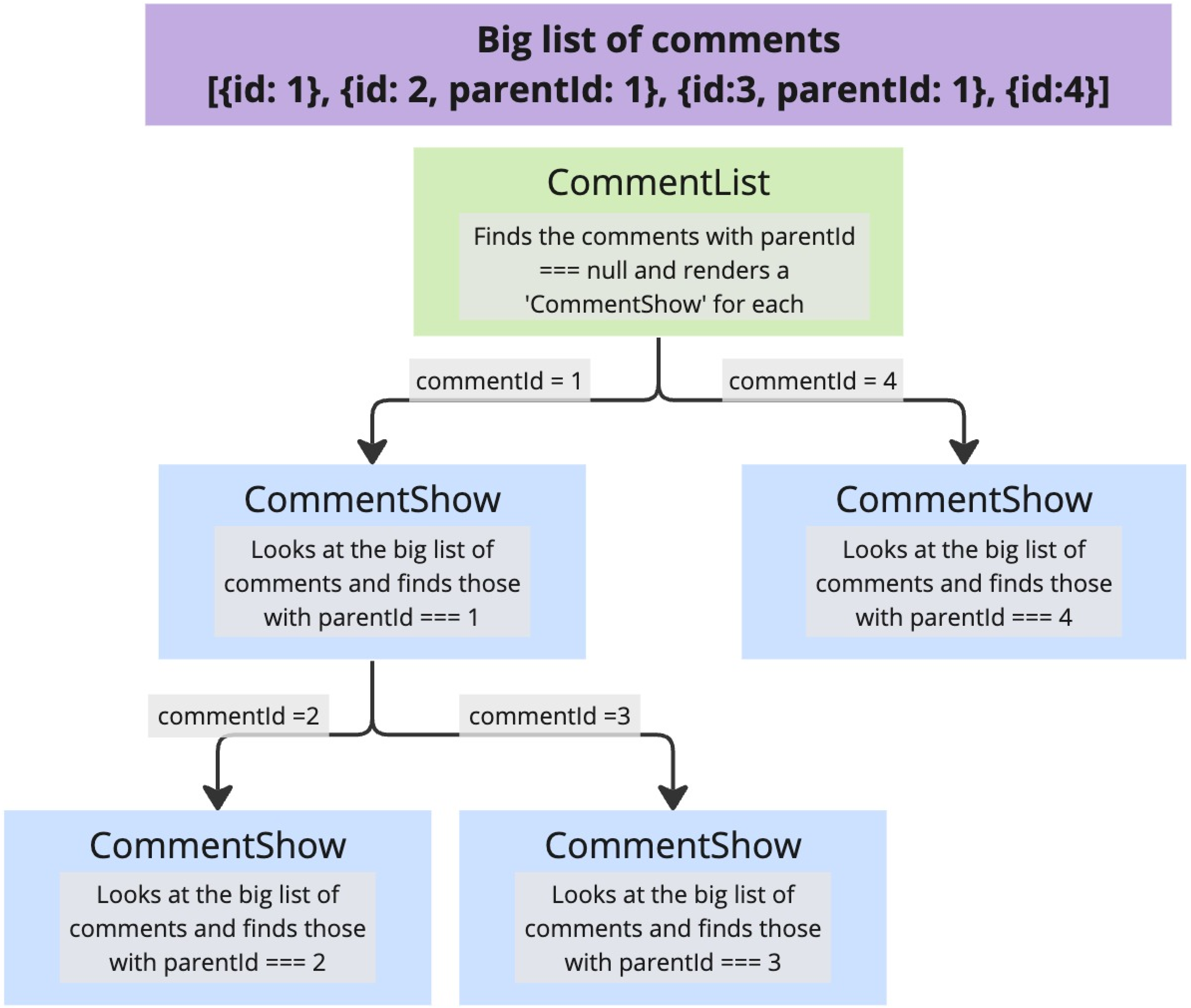
All 20 comments

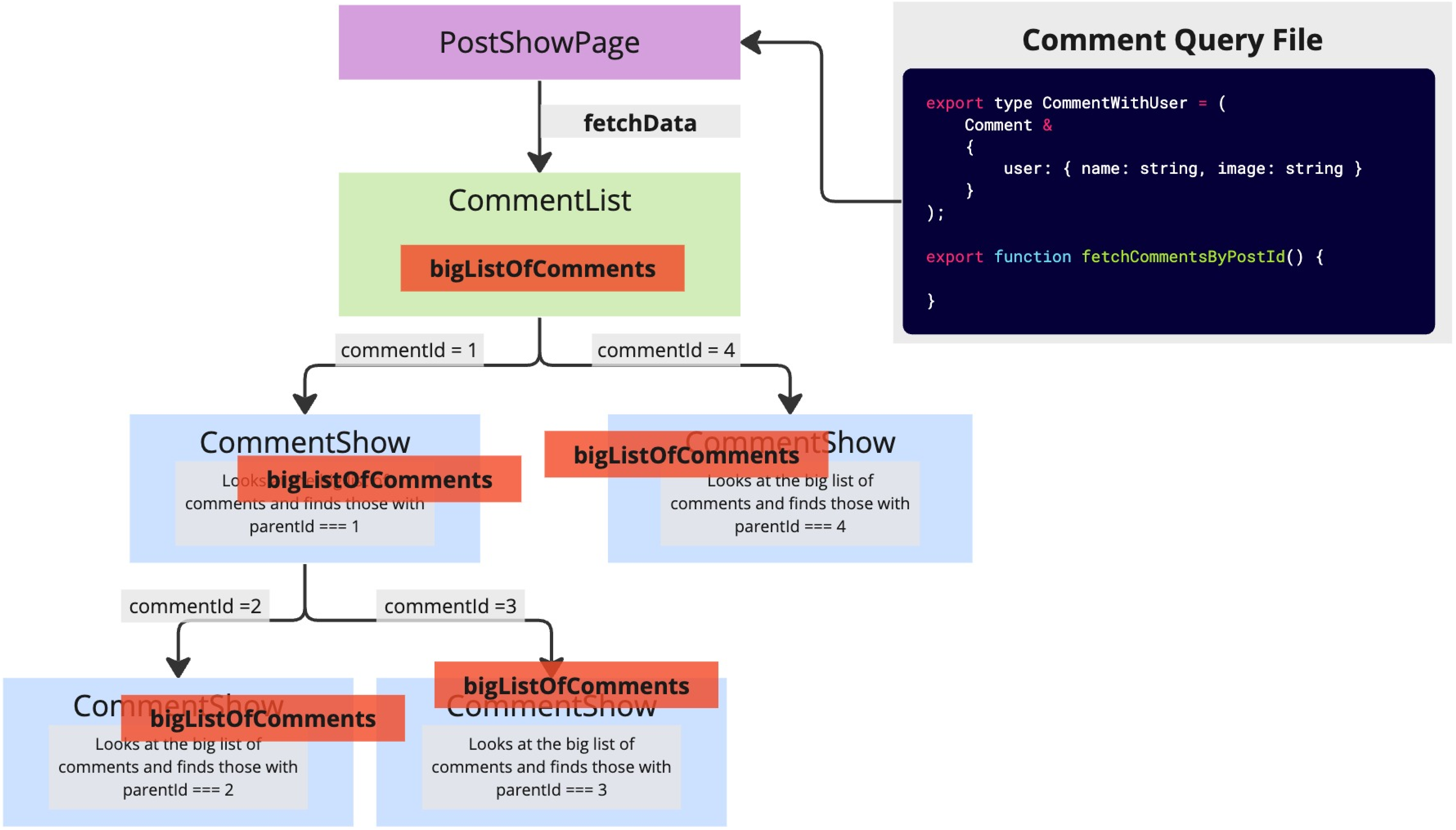
Marcos **Comment ID 1**
Have you tried using the Chart JS library?
Reply

mito **Comment ID 2**
Yes, I tried that but I've been getting errors
Reply

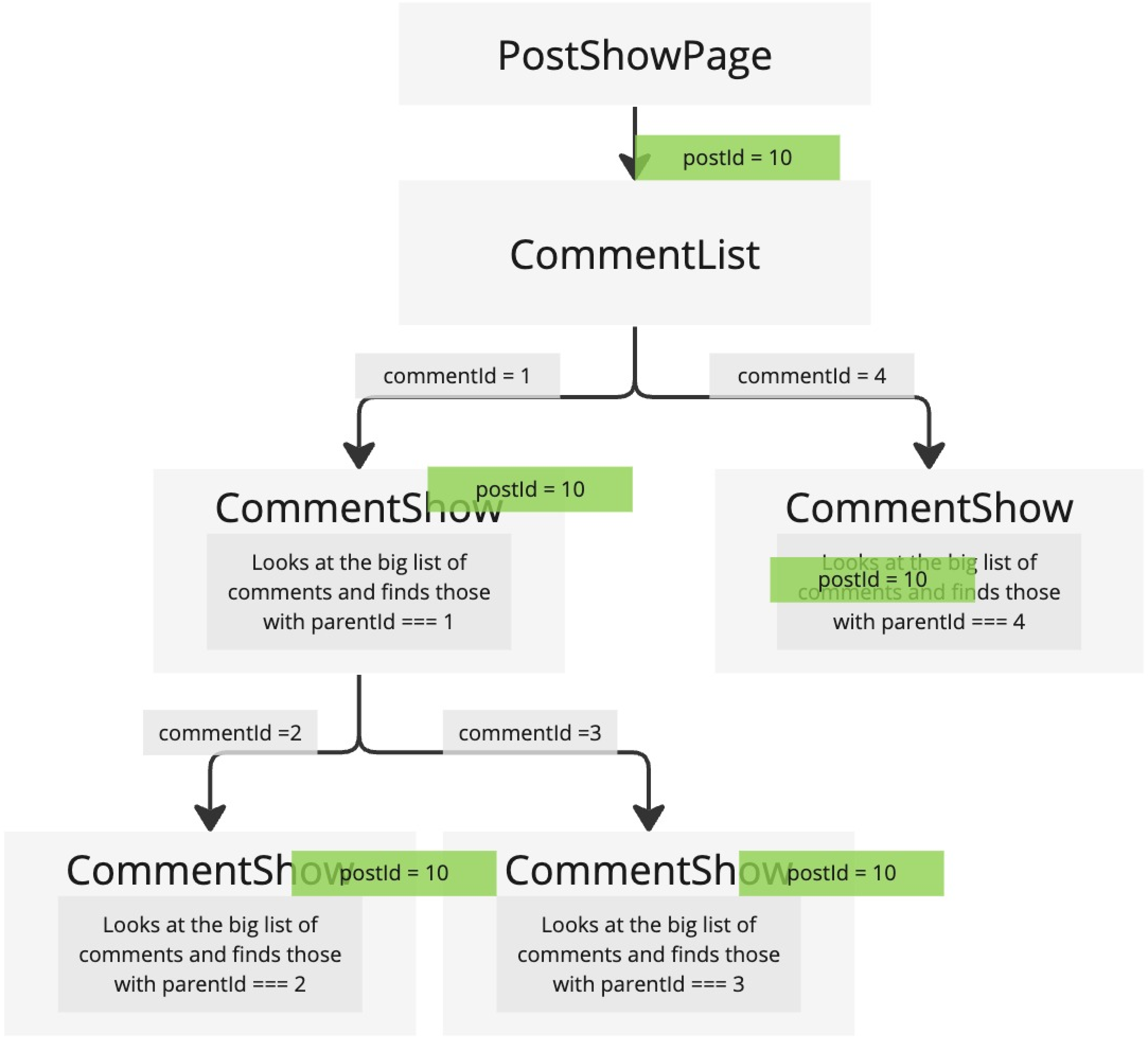
rada **Comment ID 3**
have you tried something else?
Reply

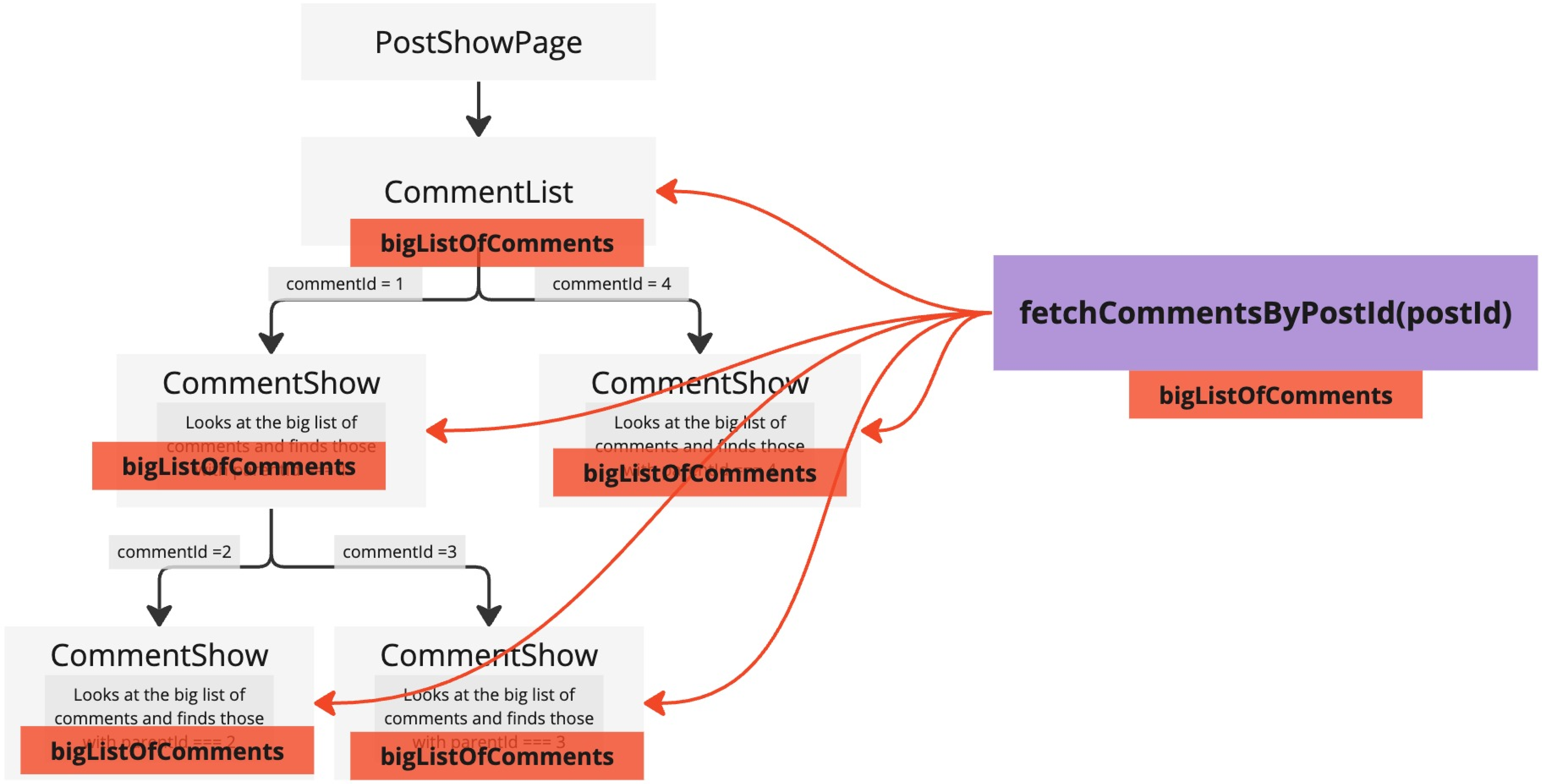
obs **Comment ID 4**
I haven't used charts before
Reply


Big list of comments
[{id: 1}, {id: 2, parentId: 1}, {id:3, parentId: 1}, {id:4}]
CommentList
Finds the comments with parentId === null and renders a 'CommentShow' for each
commentId = 1
commentId = 4
CommentShow
Looks at the big list of comments and finds those with parentId === 1
CommentShow
Looks at the big list of comments and finds those with parentId === 4
commentId =2
commentId =3
CommentShow
Looks at the big list of comments and finds those with parentId === 2
CommentShow
Looks at the big list of comments and finds those with parentId === 3


PostShowPage
fetchData
CommentList
bigListOfComments
commentId = 1
commentId = 4
CommentShow
bigListOfComments
Looks at the big list of comments and finds those with parentId === 1
CommentShow
bigListOfComments
Looks at the big list of comments and finds those with parentId === 4
commentId =2
commentId =3
CommentShow
bigListOfComments
Looks at the big list of comments and finds those with parentId === 2
CommentShow
bigListOfComments
Looks at the big list of comments and finds those with parentId === 3
Comment Query File

```
export type CommentWithUser = (
    Comment &
    {
        user: { name: string, image: string }
    }
);

export function fetchCommentsByPostId() {

}
```


PostShowPage
postId = 10
CommentList
commentId = 1
commentId = 4
CommentShow
postId = 10
Looks at the big list of comments and finds those with parentId === 1
CommentShow
postId = 10
Looks at the big list of comments and finds those with parentId === 4
commentId =2
commentId =3
CommentShow
postId = 10
Looks at the big list of comments and finds those with parentId === 2
CommentShow
postId = 10
Looks at the big list of comments and finds those with parentId === 3


PostShowPage
CommentList
bigListOfComments
commentId = 1
commentId = 4
CommentShow
Looks at the big list of comments and finds those with parentId === 1
bigListOfComments
CommentShow
Looks at the big list of comments and finds those with parentId === 4
bigListOfComments
commentId =2
commentId =3
CommentShow
Looks at the big list of comments and finds those with parentId === 2
bigListOfComments
CommentShow
Looks at the big list of comments and finds those with parentId === 3
bigListOfComments
fetchCommentsByPostId(postId)
bigListOfComments

Normally having components individually fetch their data is bad!

Leads to duplicate queries to the database

We can use another cache system to **deduplicate** these queries

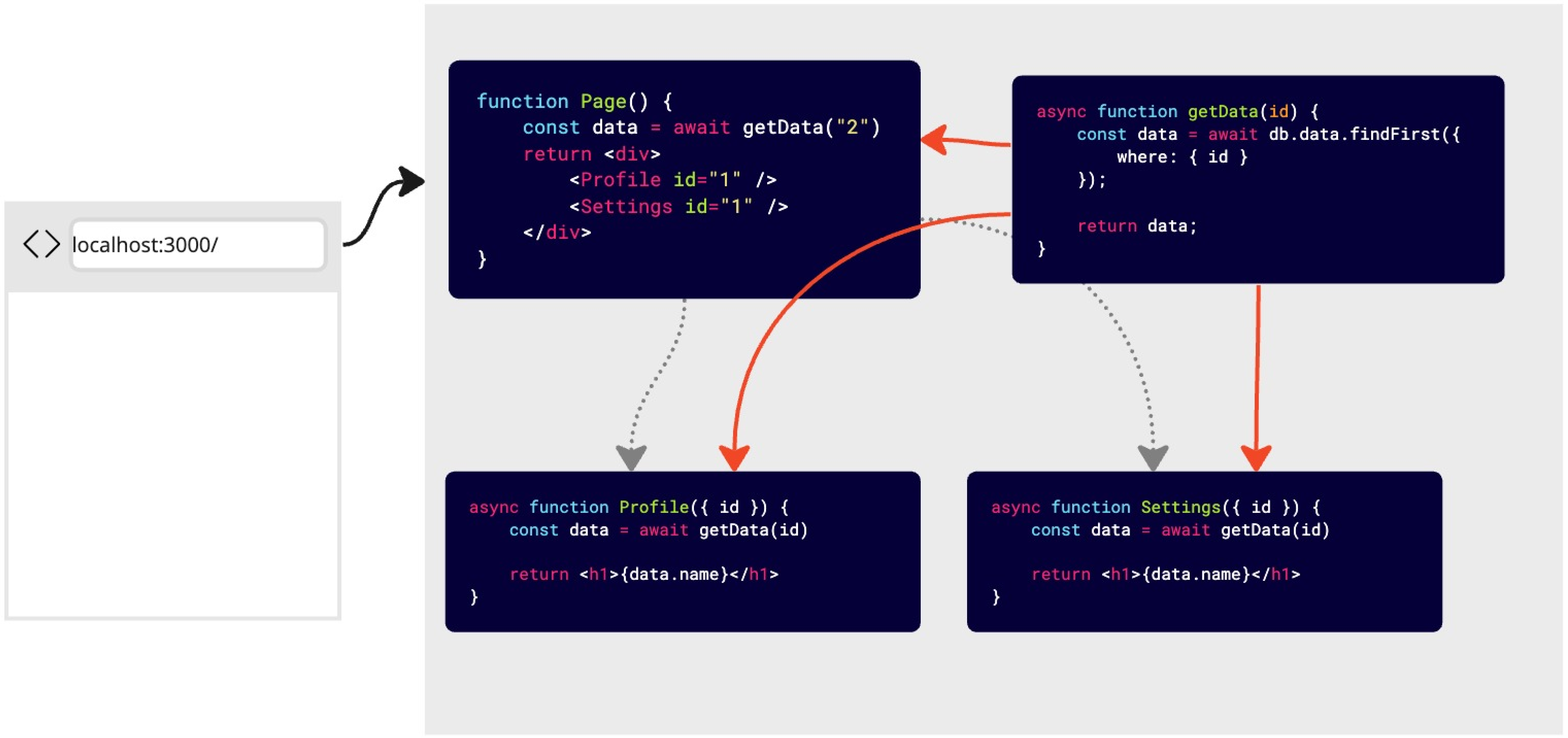
Caching

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	Requests made with 'fetch' or functions ran with 'cache' are deduplicated
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.

Next Server

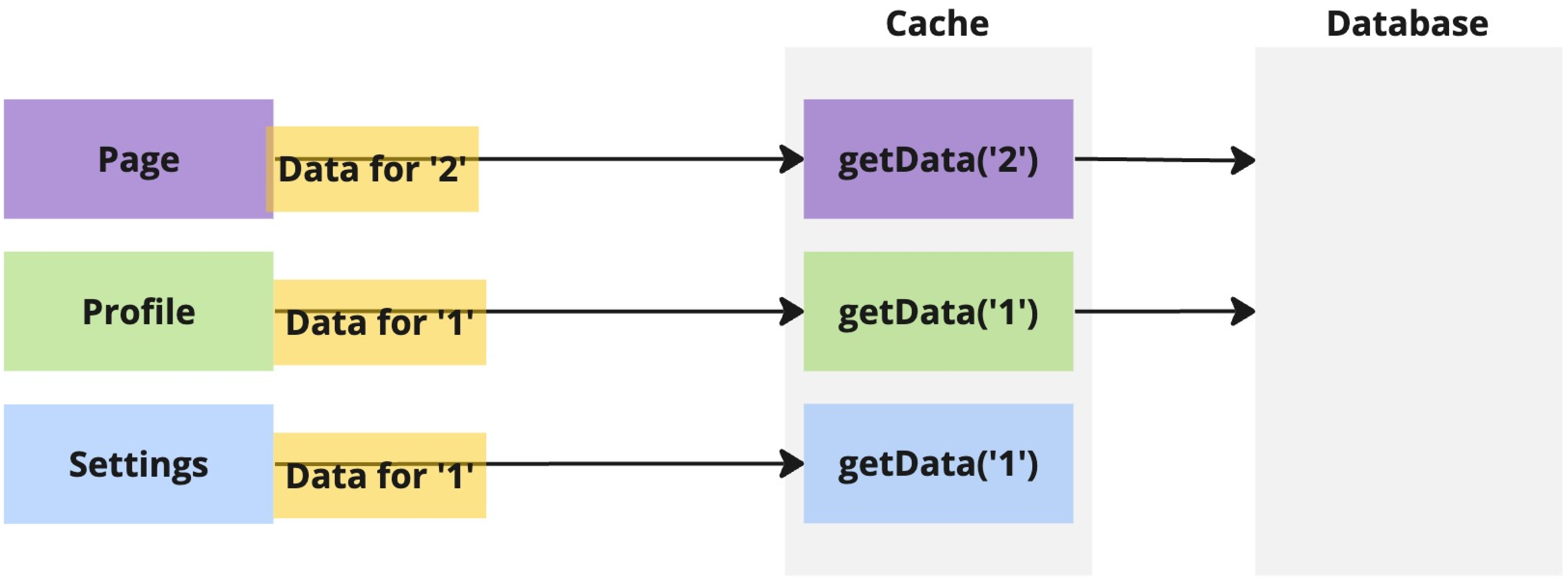


```
function Page() {
    const data = await getData("2")
    return <div>
        <Profile id="1" />
        <Settings id="1" />
    </div>
}
```

```
async function getData(id) {
    const data = await db.data.findFirst({
        where: { id }
    });

    return data;
}
```

```
async function Profile({ id }) {
    const data = await getData(id)

    return <h1>{data.name}</h1>
}
```

```
async function Settings({ id }) {
    const data = await getData(id)

    return <h1>{data.name}</h1>
}
```

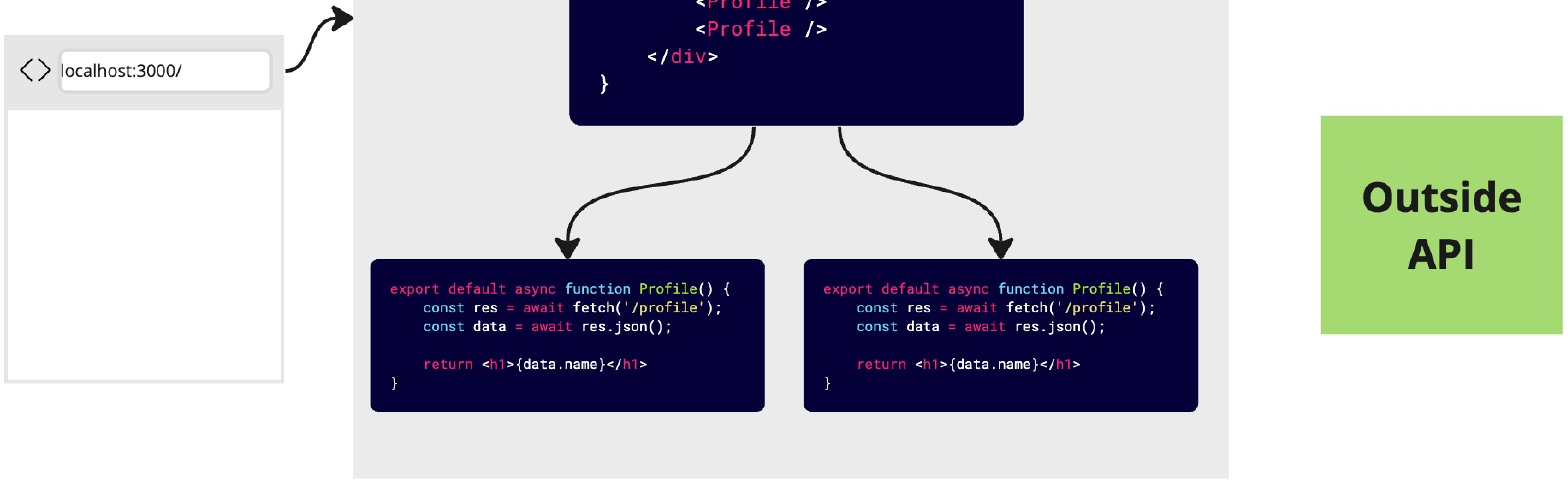

Cache
Database
Page
Data for '2'
getData('2')
Profile
Data for '1'
getData('1')
Settings
Data for '1'
getData('1')

The cache memoization system is cleared out between incoming requests

Automatically used with the built-in 'fetch' function

Can be used with other functions (like db queries) by using the '**cache**' function

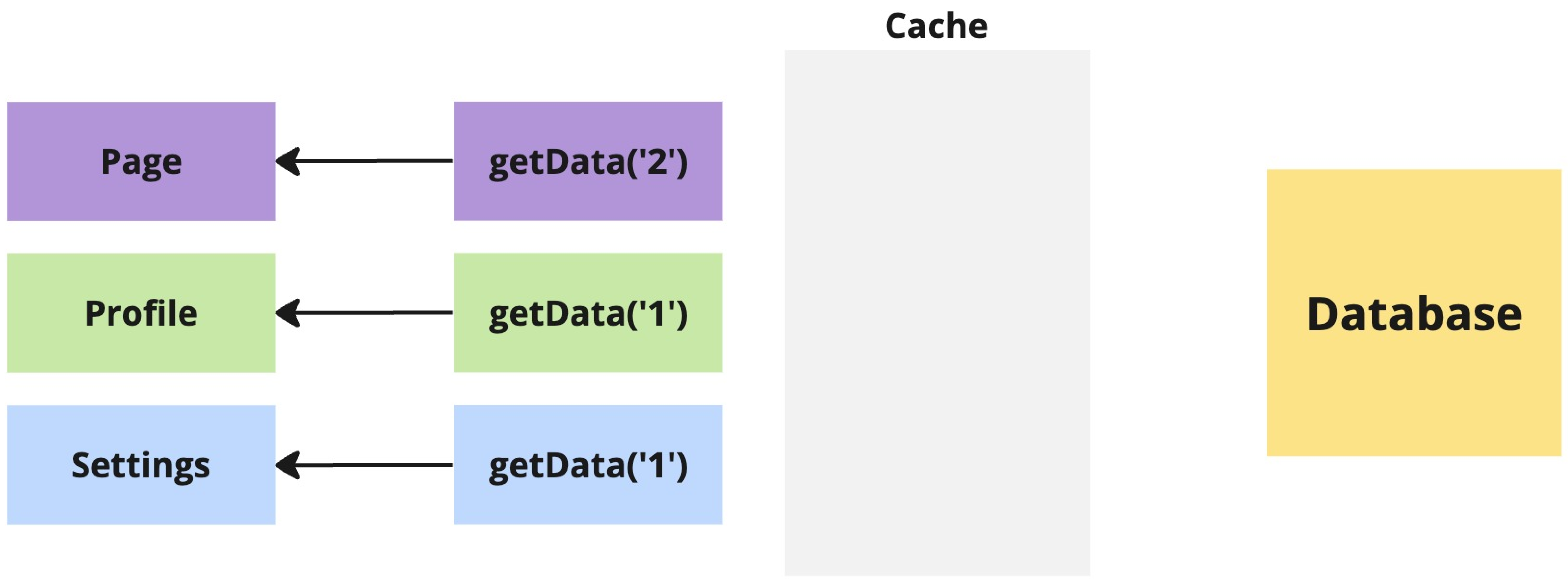
Next Server

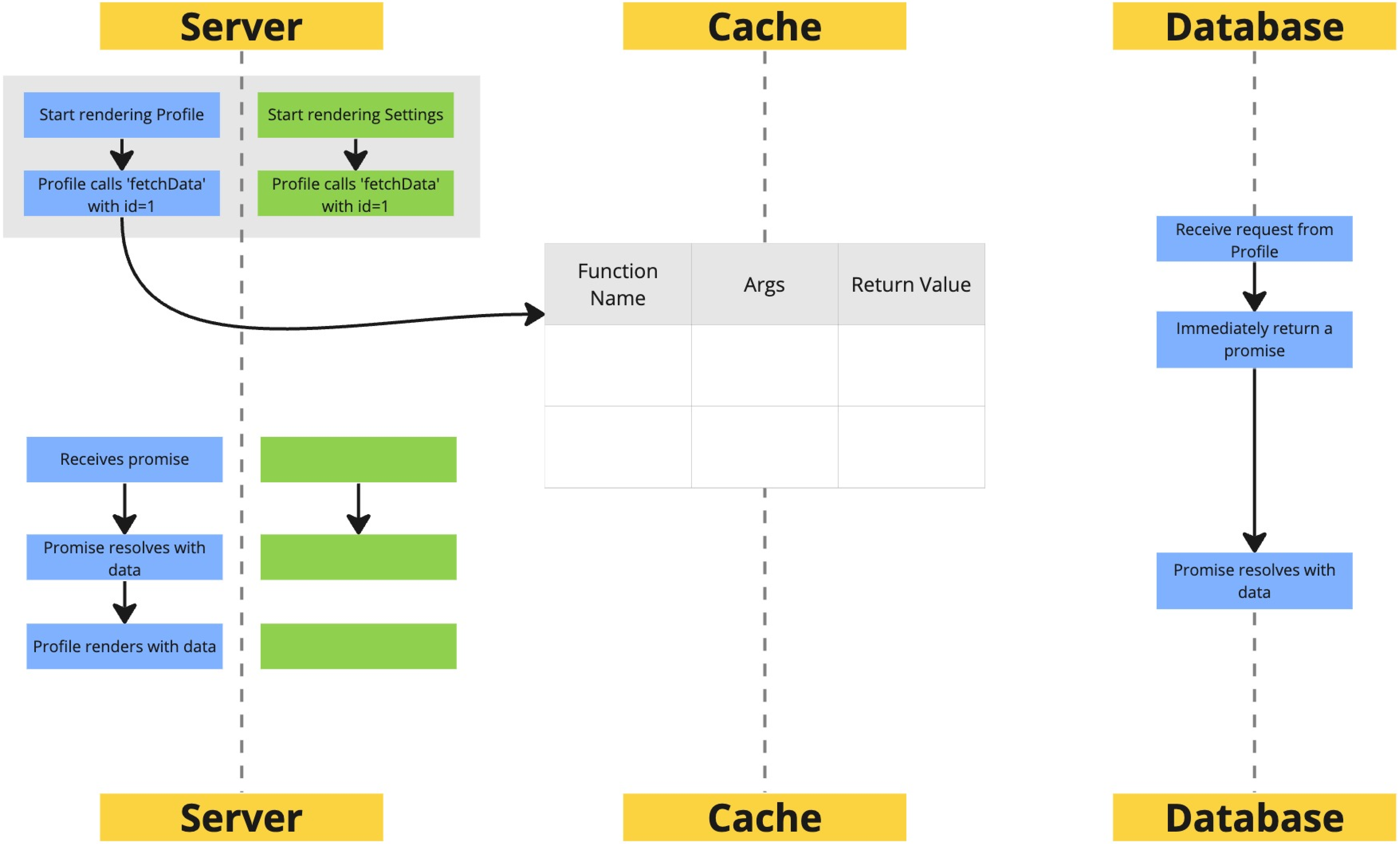


```
export default function Page() {
    return <div>
        <Profile />
        <Profile />
    </div>
}
```

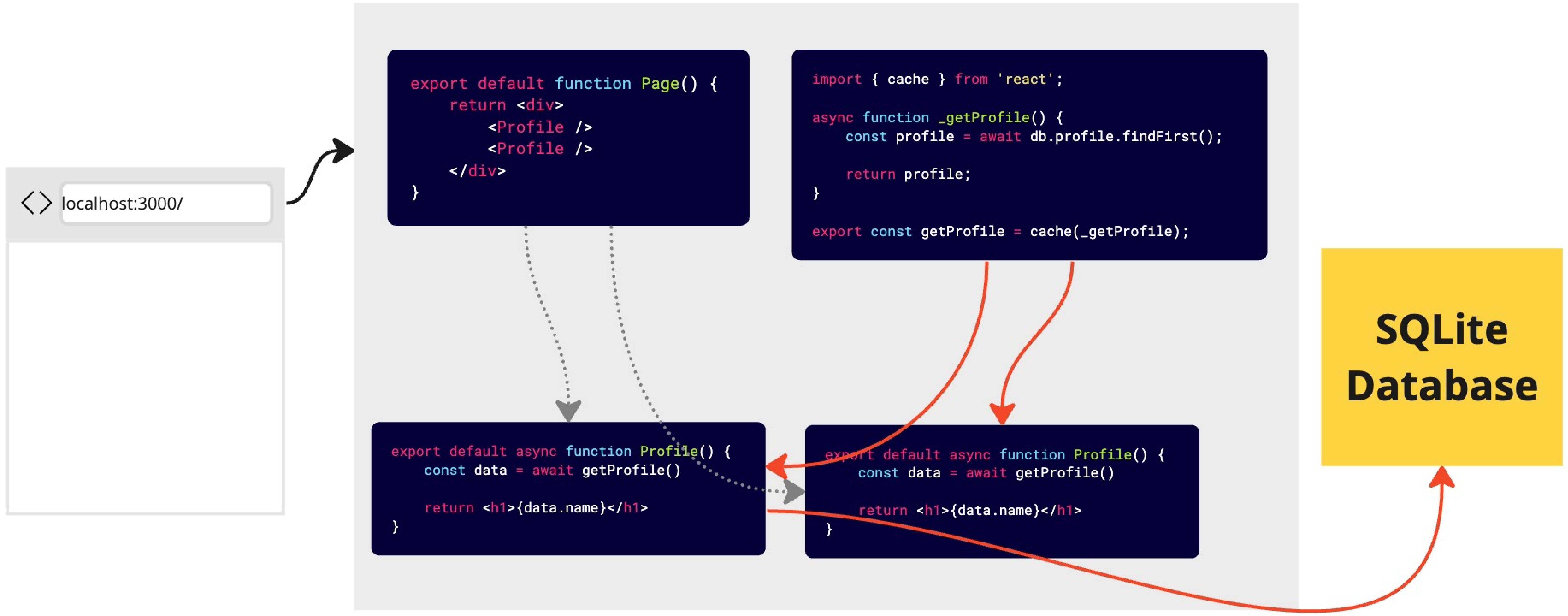
```
export default async function Profile() {
    const res = await fetch('/profile');
    const data = await res.json();

    return <h1>{data.name}</h1>
}
```

```
export default async function Profile() {
    const res = await fetch('/profile');
    const data = await res.json();

    return <h1>{data.name}</h1>
}
```


Cache
Page
getData('2')
Profile
getData('1')
Settings
getData('1')
Database


Server
Cache
Database
Start rendering Profile
Start rendering Settings
Profile calls 'fetchData' with id=1
Profile calls 'fetchData' with id=1
Receive request from Profile
Immediately return a promise
Function Name
Args
Return Value
Receives promise
Promise resolves with data
Profile renders with data
Promise resolves with data
Server
Cache
Database

Next Server

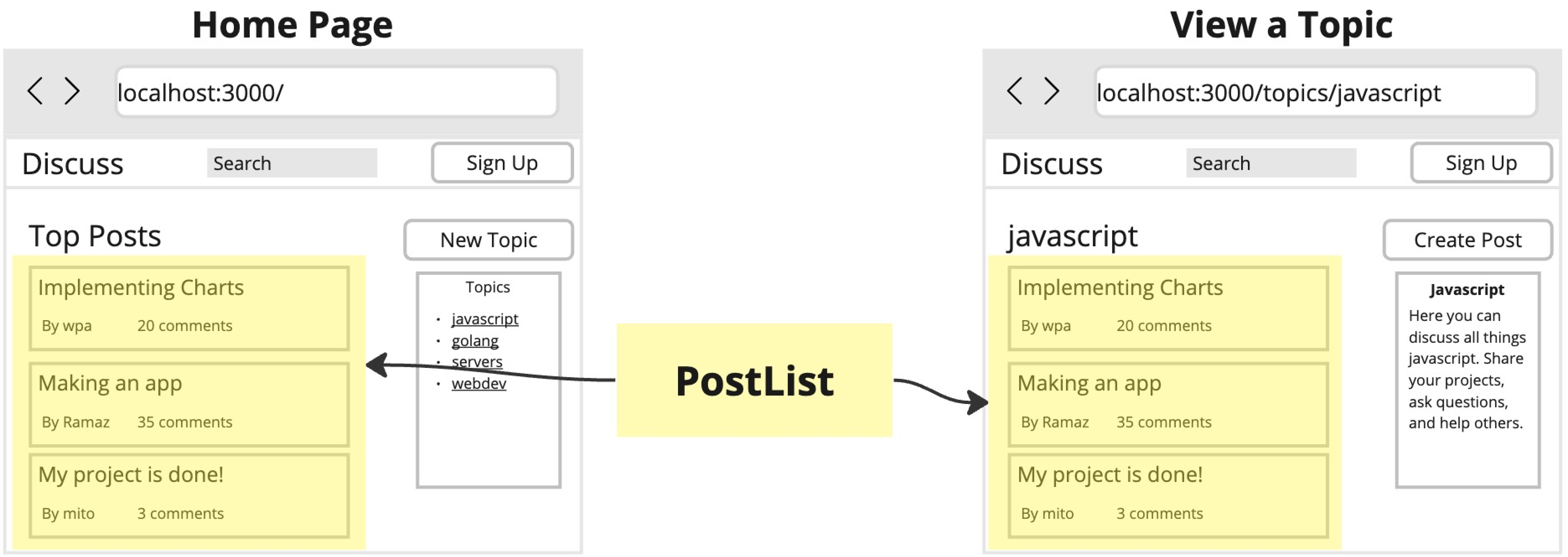


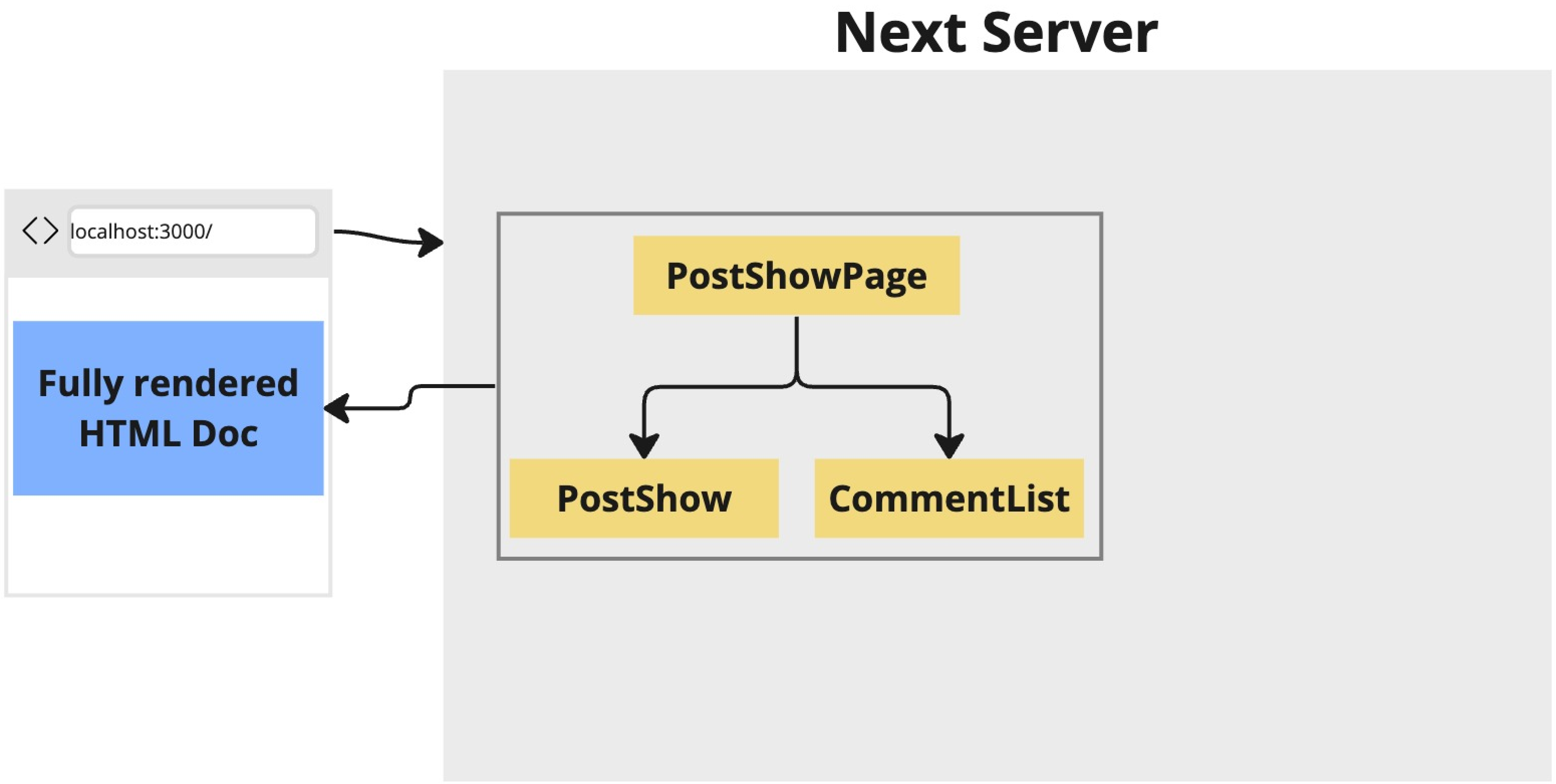
```
export default function Page() {
    return <div>
        <Profile />
        <Profile />
    </div>
}
```

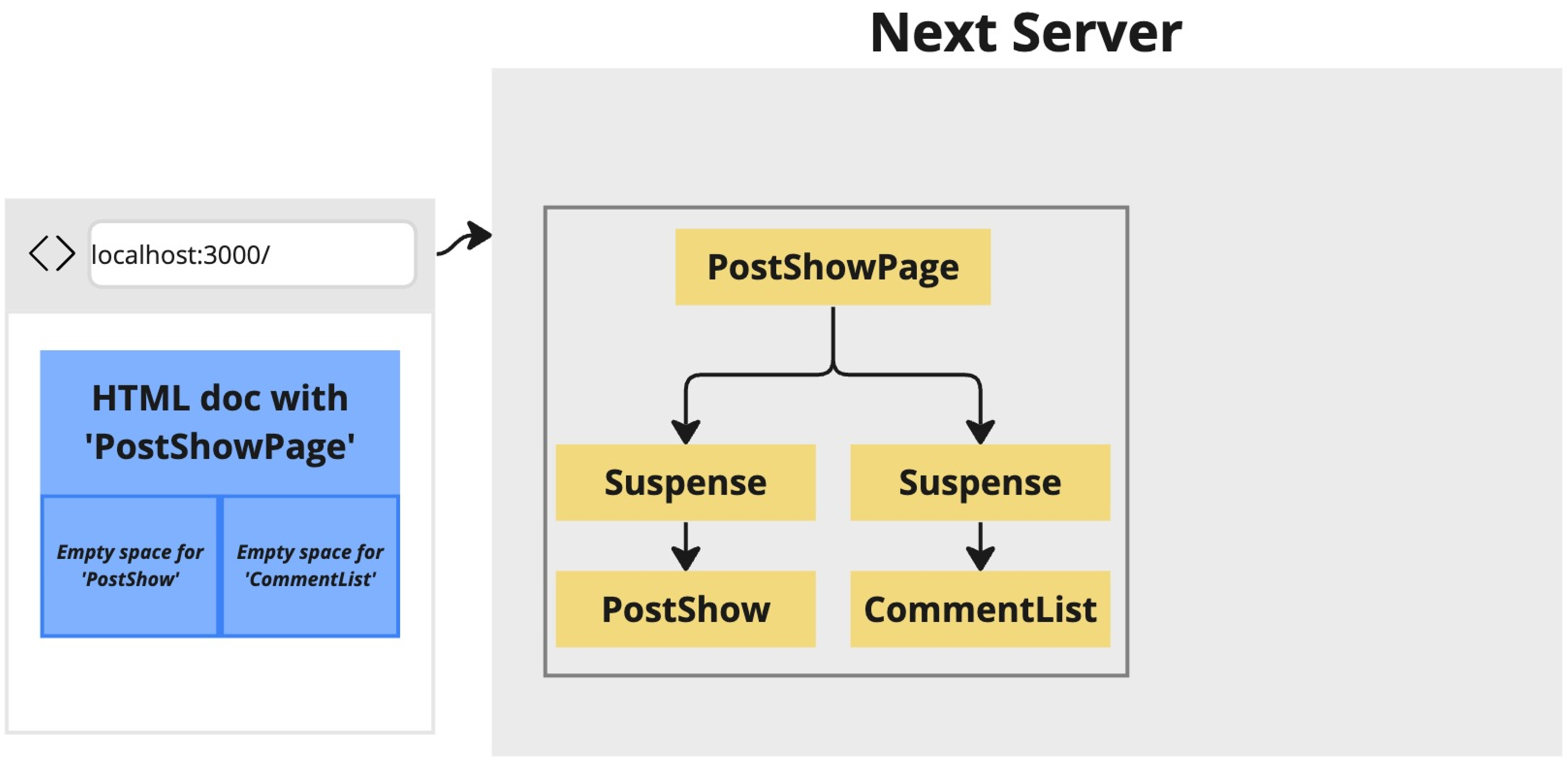
```
import { cache } from 'react';

async function _getProfile() {
    const profile = await db.profile.findFirst();

    return profile;
}

export const getProfile = cache(_getProfile);
```

```
export default async function Profile() {
    const data = await getProfile()

    return <h1>{data.name}</h1>
}
```

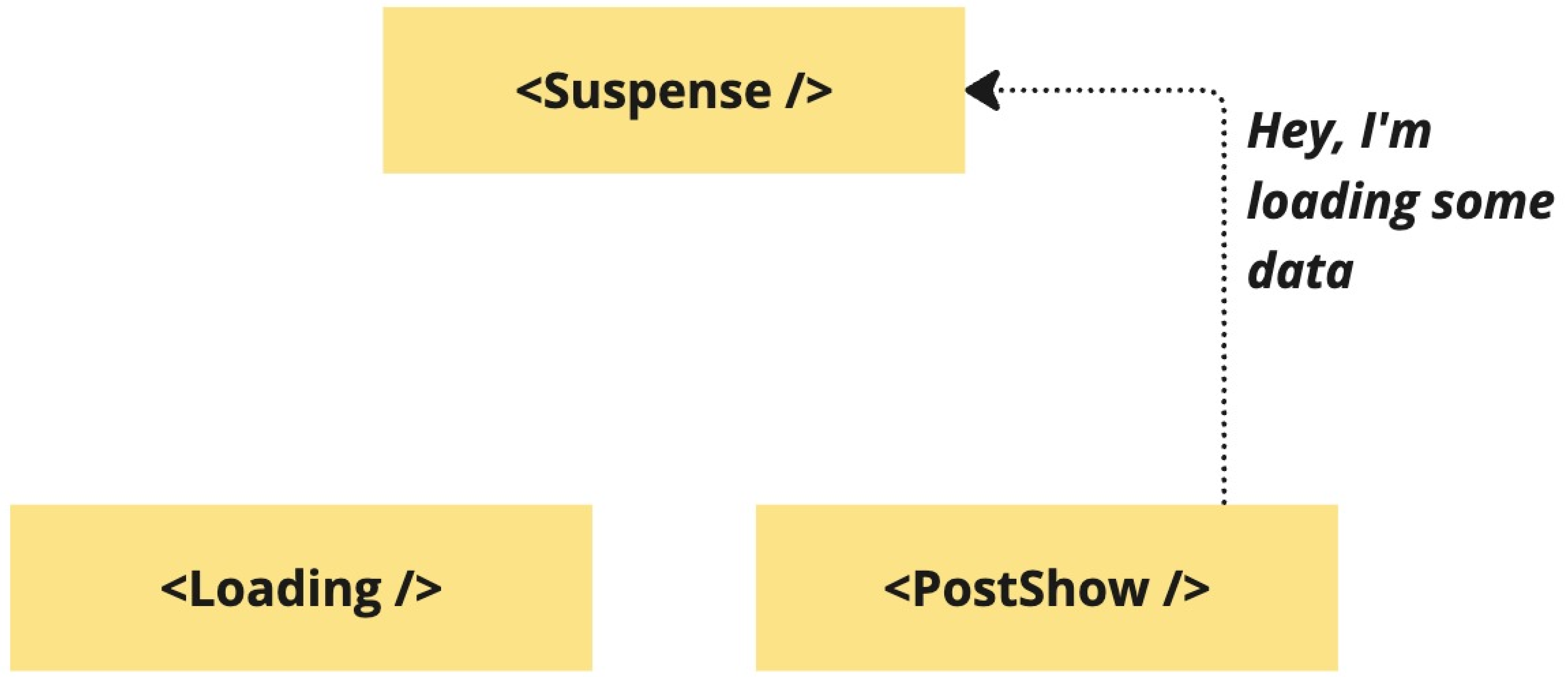
```
export default async function Profile() {
    const data = await getProfile()

    return <h1>{data.name}</h1>
}
```

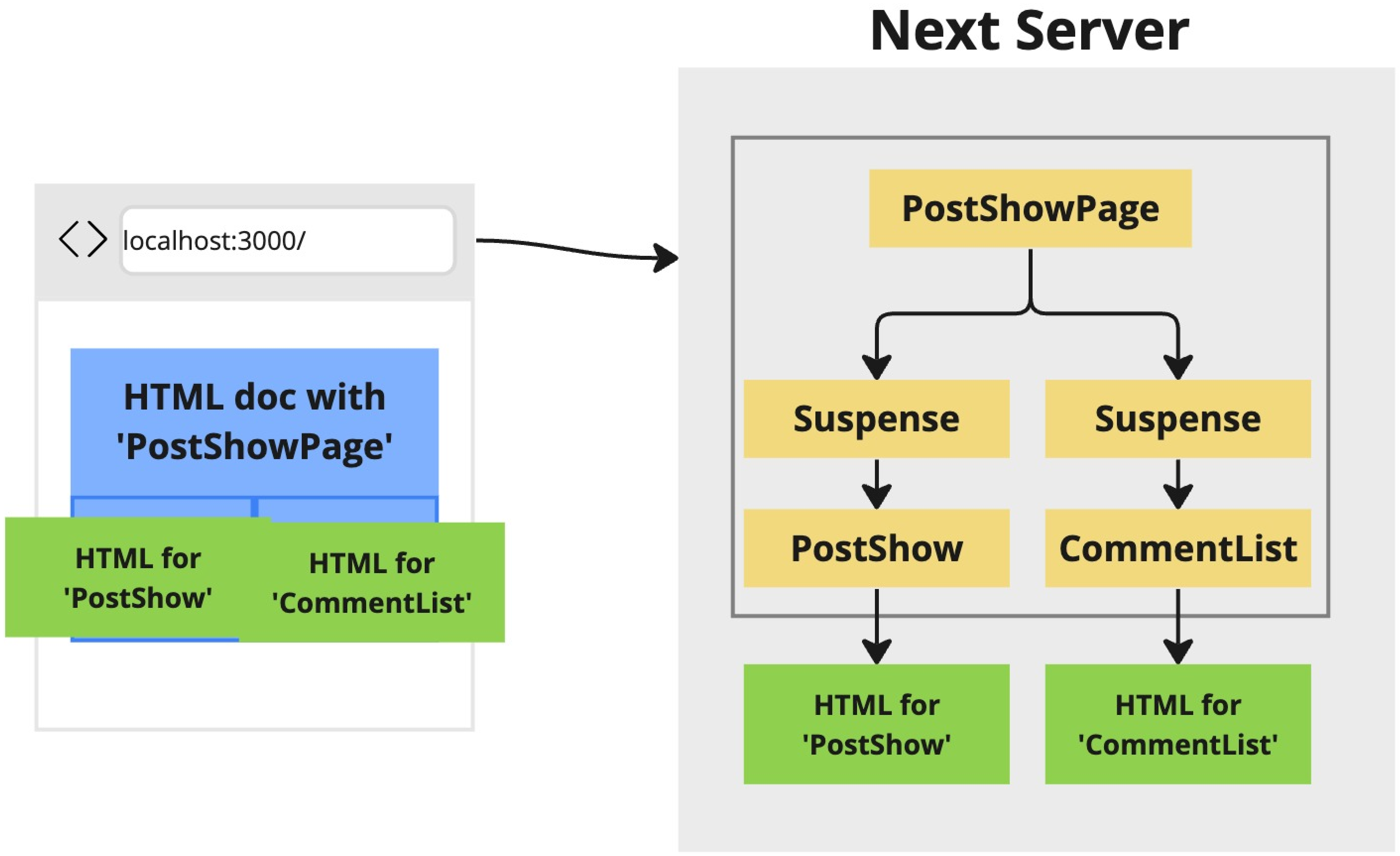

Home Page
localhost:3000/
Discuss
Search
Sign Up
Top Posts
New Topic
Implementing Charts
By wpa
20 comments
Making an app
By Ramaz
35 comments
My project is done!
By mito
3 comments
Topics
javascript
golang
servers
webdev
PostList
View a Topic
localhost:3000/topics/javascript
Discuss
Search
Sign Up
javascript
Create Post
Implementing Charts
By wpa
20 comments
Making an app
By Ramaz
35 comments
My project is done!
By mito
3 comments
Javascript
Here you can discuss all things javascript. Share your projects, ask questions, and help others.


Next Server
localhost:3000/
Fully rendered HTML Doc
PostShowPage
PostShow
CommentList

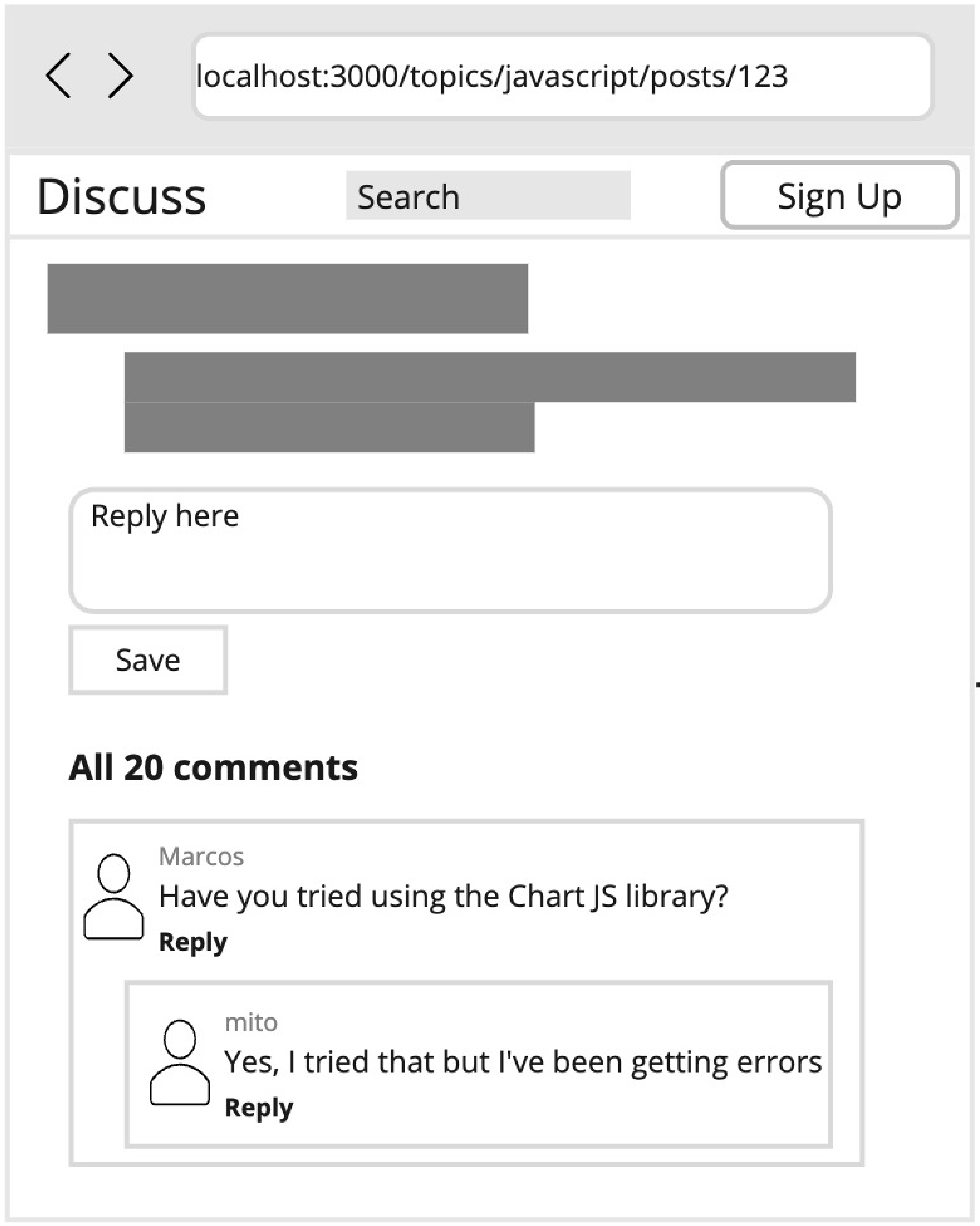

Next Server
localhost:3000/
HTML doc with 'PostShowPage'
Empty space for 'PostShow'
Empty space for 'CommentList'
PostShowPage
Suspense
Suspense
PostShow
CommentList

```
<Suspense fallback={<Loading />}>
    <PostShow />
</Suspense>
```




Next Server
localhost:3000/
HTML doc with 'PostShowPage'
HTML for 'PostShow'
HTML for 'CommentList'
PostShowPage
Suspense
Suspense
PostShow
CommentList
HTML for 'PostShow'
HTML for 'CommentList'

View a Post

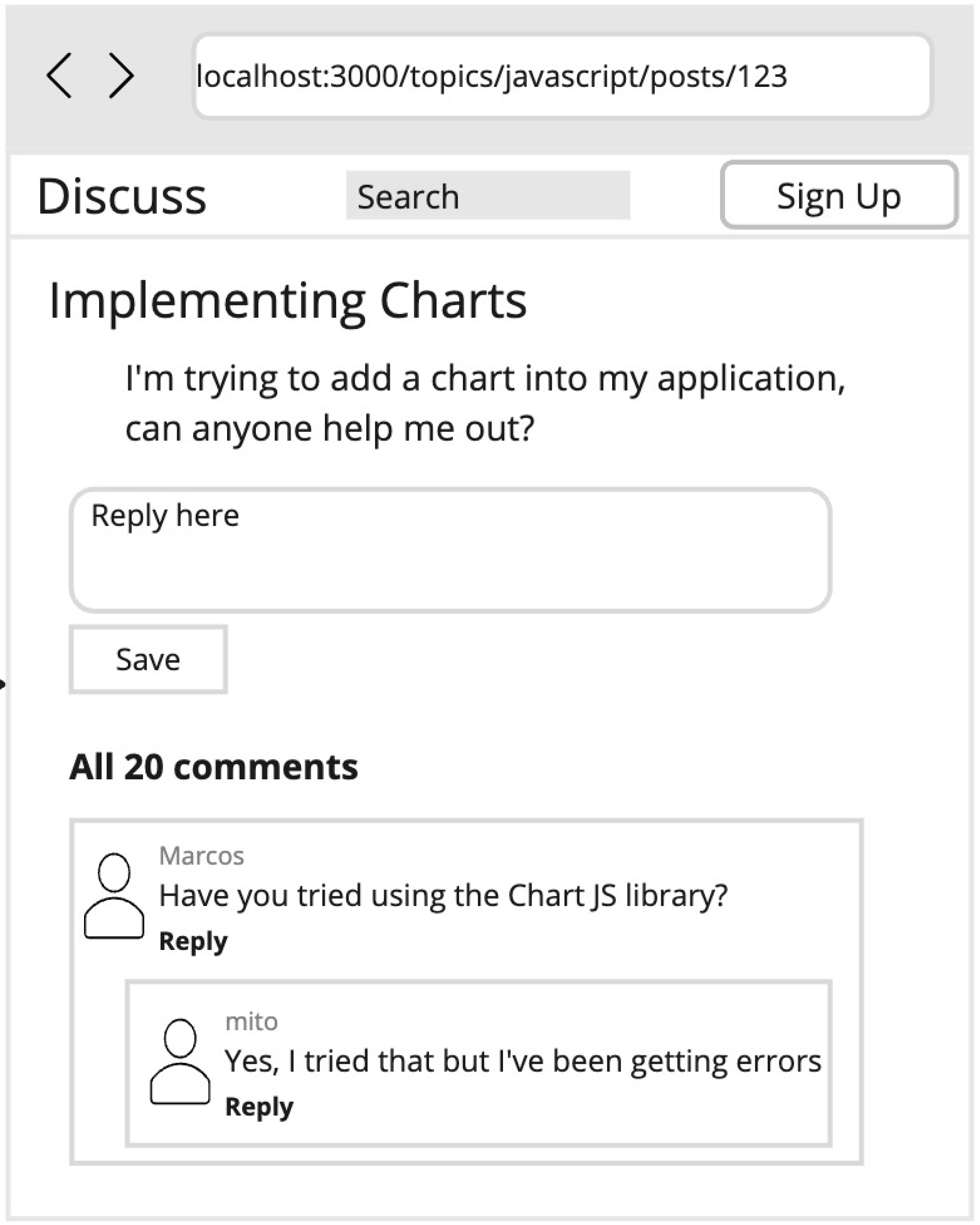


All 20 comments

Marcos
Have you tried using the Chart JS library?
Reply

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

View a Post



Implementing Charts

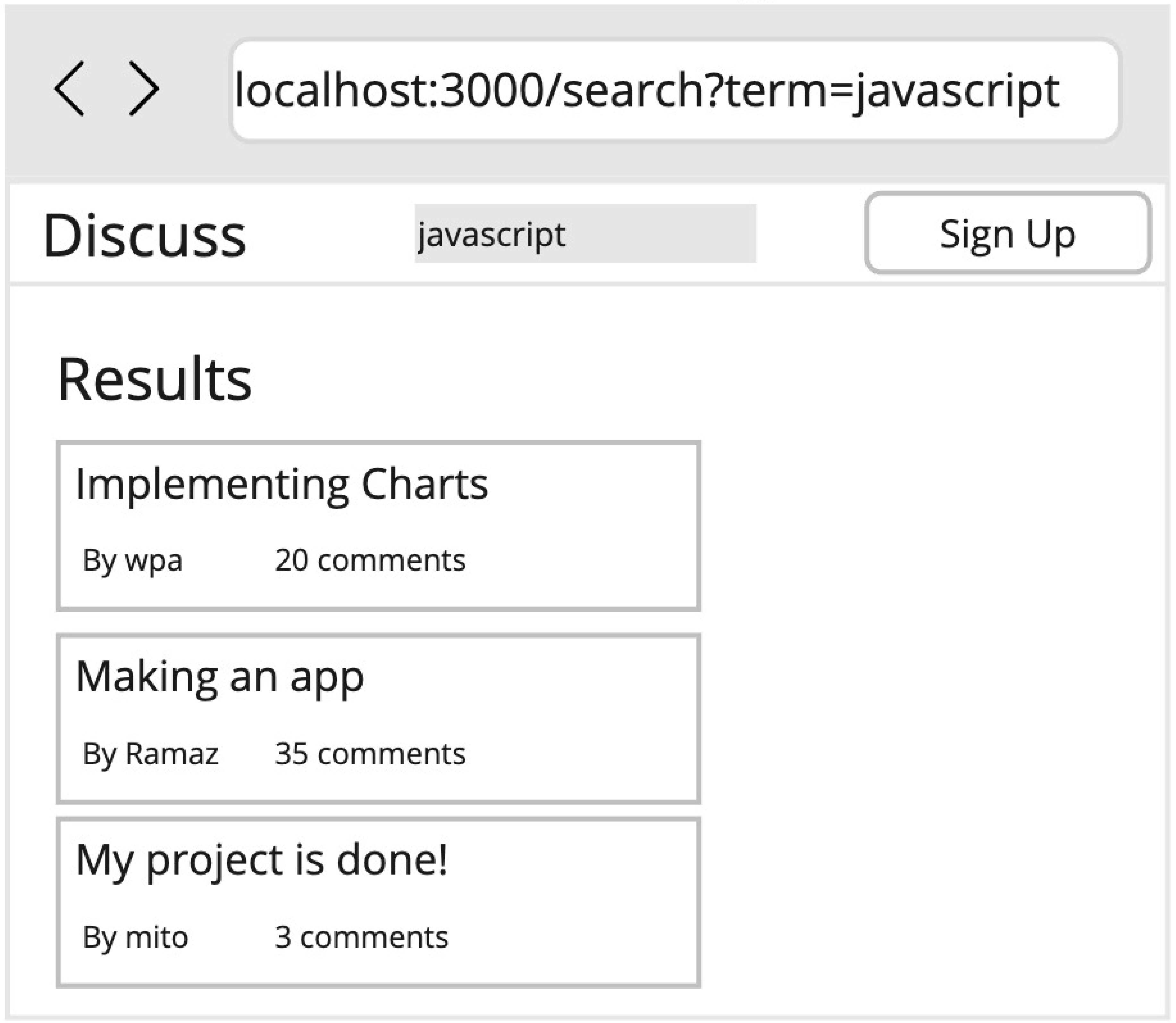
I'm trying to add a chart into my application, can anyone help me out?

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

Search Page



Users should be able to bookmark or share the URL of the search results page

The search input's default value should come from the query string

Page components receive the query string data through the 'searchParams' prop

```
interface SearchPageProps {
    searchParams: {
        term: string;
    }
}

function SearchPage({ searchParams}: SearchPageProps) {
    return <div>
        {searchParams.term}
    </div>
}
```

Client components can get query string data with 'useSearchParams'

```
'use client';

import { useSearchParams } from 'next/navigation'

function SearchInput() {
    const searchParams = useSearchParams();

    return <div>
        {searchParams.term}
    </div>
}
```

Client components with 'useSearchParams' need to be wrapped with 'Suspense' or you'll get a strange warning at build time

```
'use client';

import { useSearchParams } from 'next/navigation'

function SearchInput() {
    const searchParams = useSearchParams();

    return <div>
        {searchParams.term}
    </div>
}
```

```
function Page() {
    return <div>
        <Suspense>
            <SearchInput />
        </Suspense>
    </div>
}
```

Pages that reference 'searchParams' will be marked as 'dynamic' for purposes of build time caching

```tsx
interface SearchPageProps {
    searchParams: {
        term: string;
    }
}

function SearchPage({ searchParams}: SearchPageProps) {
    return <div>
        {searchParams.term}
    </div>
}
```

This route is dynamic!

1	When initially displayed, search input should take its default value from the 'term' query string param
2	When the user enters a term and presses 'enter', run a server action to redirect a user to '/search?term=<term>'
3	When the user enters a term and presses 'enter', run a server action to redirect a user to '/search?term=<term>'

Design your Server Actions ahead of time!

Build your app every now and then to check your caching status

Wrap slow-loading components with Suspense to enable streaming

Consider using that query function data fetching pattern