BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

Topics

- What are **React Server Components** and why would you care?
- Using **Next.js** and the **App Router**
- Turning a React **Client Component into** a React **Server Component**
- **Updates and caching** with React Server Components
- **Querying the database** from a React Server Component
- **Suspense** & React Server Components
- React Server Components and **streaming**
- **Which components** are really React Server Components?
- Using **React Server Actions** to execute code on the server

See you in the next video

Personal introduction

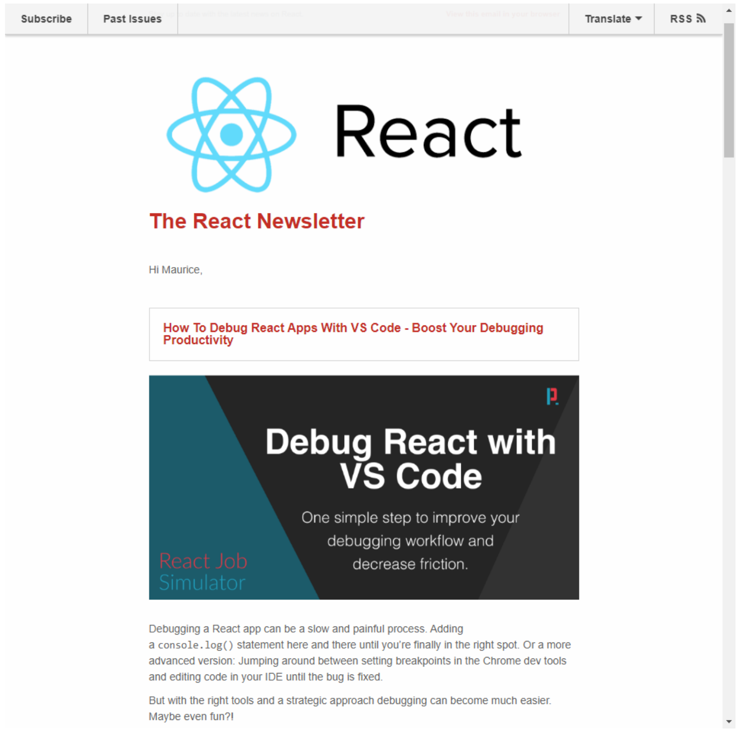
BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

Personal introduction

- Maurice de Beijer
- The Problem Solver
- Microsoft MVP
- Freelance developer/instructor
- Currently at https://someday.com/
- Twitter: @mauricedb
- Web: http://www.TheProblemSolver.nl
- E-mail: maurice.de.beijer@gmail.com

The React Newsletter



Subscribe Past Issues Translate RSS

React

The React Newsletter

Hi Maurice,

How To Debug React Apps With VS Code - Boost Your Debugging Productivity

Debug React with VS Code

One simple step to improve your debugging workflow and decrease friction.

React Job Simulator

Debugging a React app can be a slow and painful process. Adding a `console.log()` statement here and there until you're finally in the right spot. Or a more advanced version: Jumping around between setting breakpoints in the Chrome dev tools and editing code in your IDE until the bug is fixed.

But with the right tools and a strategic approach debugging can become much easier. Maybe even fun?!

See you in the next video

Prerequisites

BOOST YOUR DEVELOPER POTENTIAL

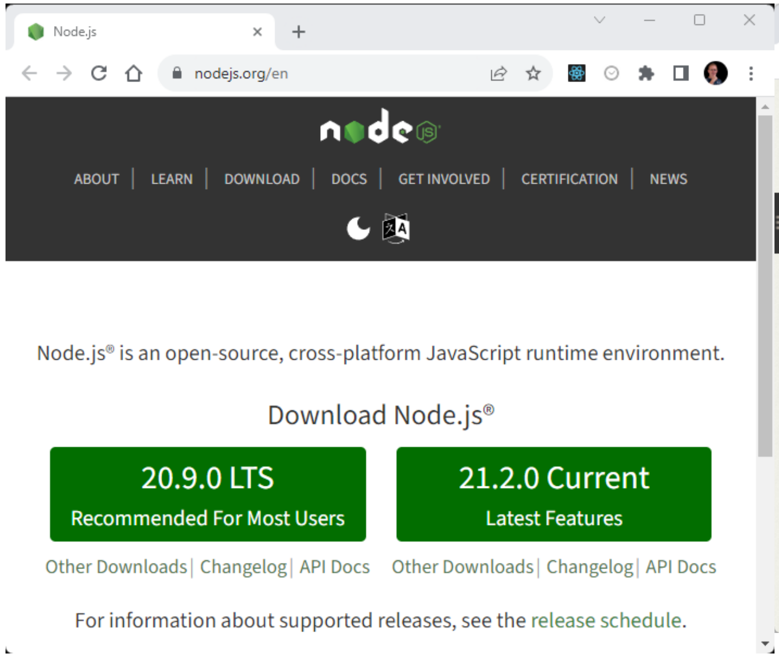
WITH REACT SERVER COMPONENTS

Prerequisites

Install Node & NPM

Install the GitHub repository

Install Node.js & NPM



nodejs.org/en

Node.js® is an open-source, cross-platform JavaScript runtime environment.

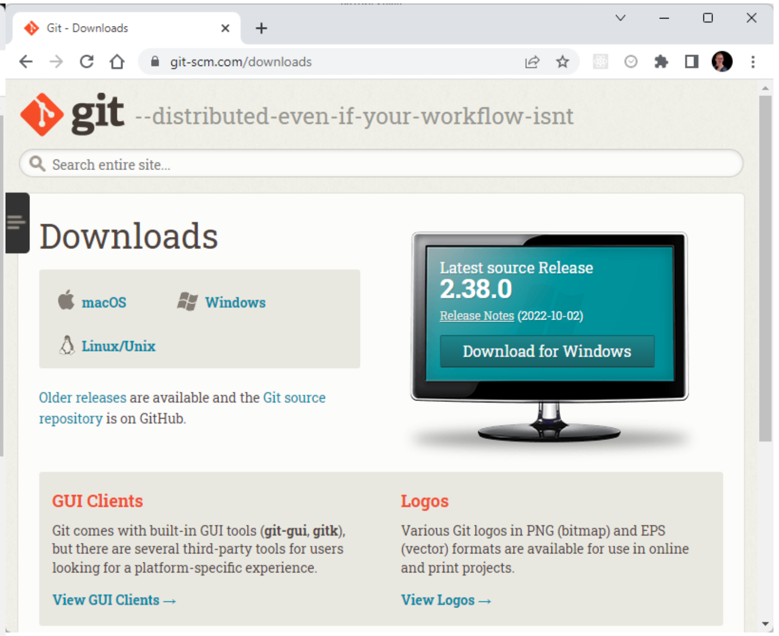
Download Node.js®

20.9.0 LTS
Recommended For Most Users

21.2.0 Current
Latest Features

Other Downloads | Changelog | API Docs

For information about supported releases, see the release schedule.



git-scm.com/downloads

git --distributed-even-if-your-workflow-isnt

Downloads

macOS Windows Linux/Unix

Latest source Release
2.38.0
Release Notes (2022-10-02)
Download for Windows

Older releases are available and the Git source repository is on GitHub.

GUI Clients

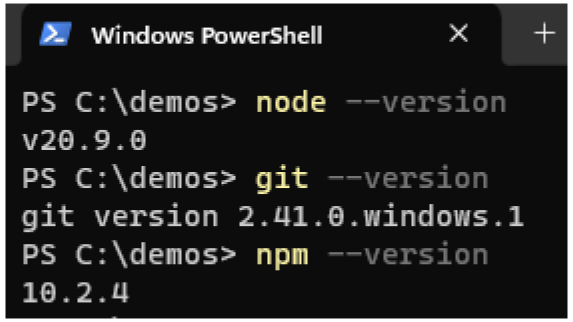
Git comes with built-in GUI tools (git-gui, gitk), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

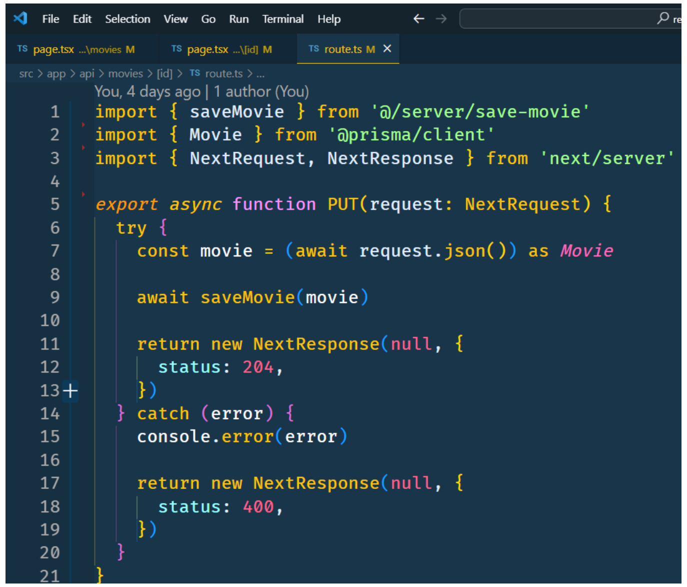
View Logos →



Windows PowerShell

```
PS C:\demos> node --version
v20.9.0
PS C:\demos> git --version
git version 2.41.0.windows.1
PS C:\demos> npm --version
10.2.4
```

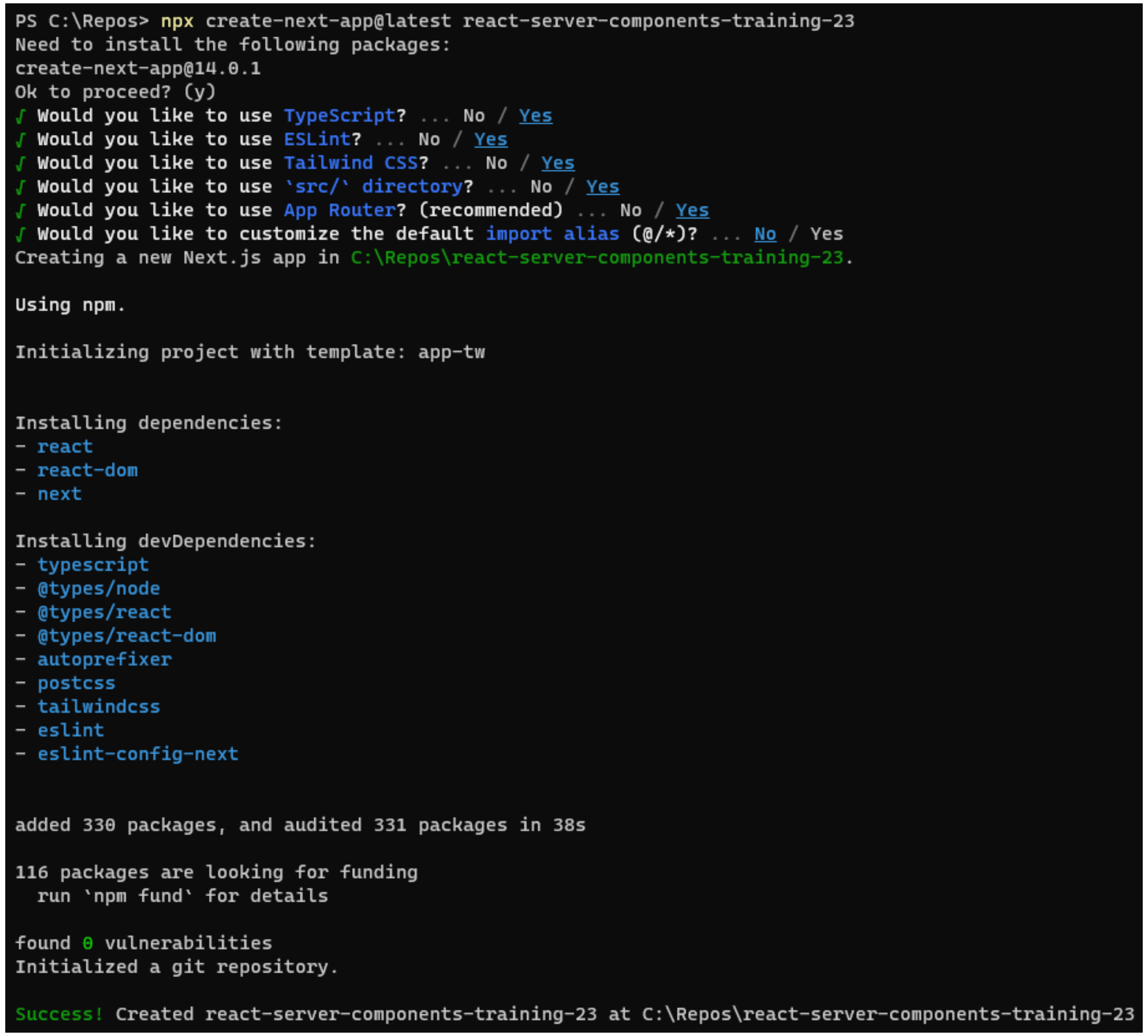
Following Along


File Edit Selection View Go Run Terminal Help
page.tsx ...\movies M
page.tsx ...\[id] M
route.ts M
src > app > api > movies > [id] > route.ts > ...
You, 4 days ago | 1 author (You)

```
import { saveMovie } from '@/server/save-movie'
import { Movie } from '@prisma/client'
import { NextRequest, NextResponse } from 'next/server'

export async function PUT(request: NextRequest) {
  try {
    const movie = (await request.json()) as Movie

    await saveMovie(movie)

    return new NextResponse(null, {
      status: 204,
    })
  } catch (error) {
    console.error(error)

    return new NextResponse(null, {
      status: 400,
    })
  }
}
```

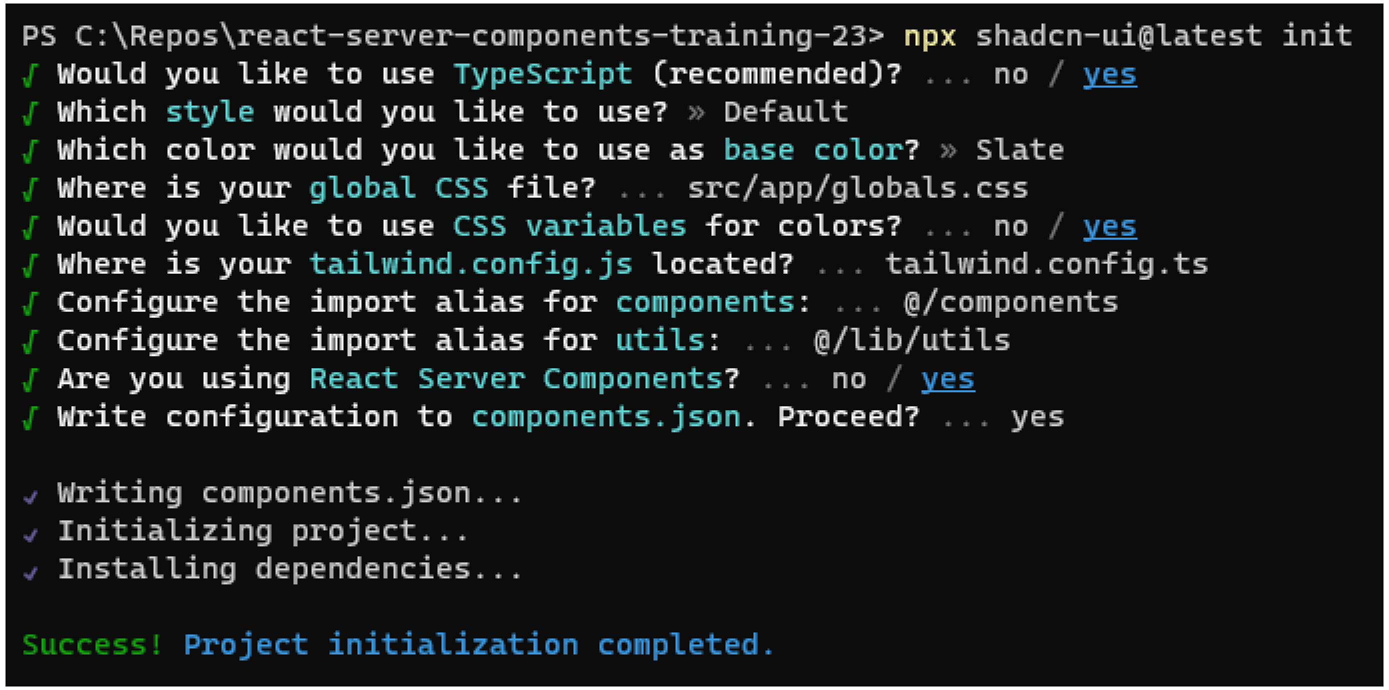
- Repo: https://bit.ly/rsc-training-23-github
- Slides: https://bit.ly/rsc-training-23-slides

Create a new Next.js app



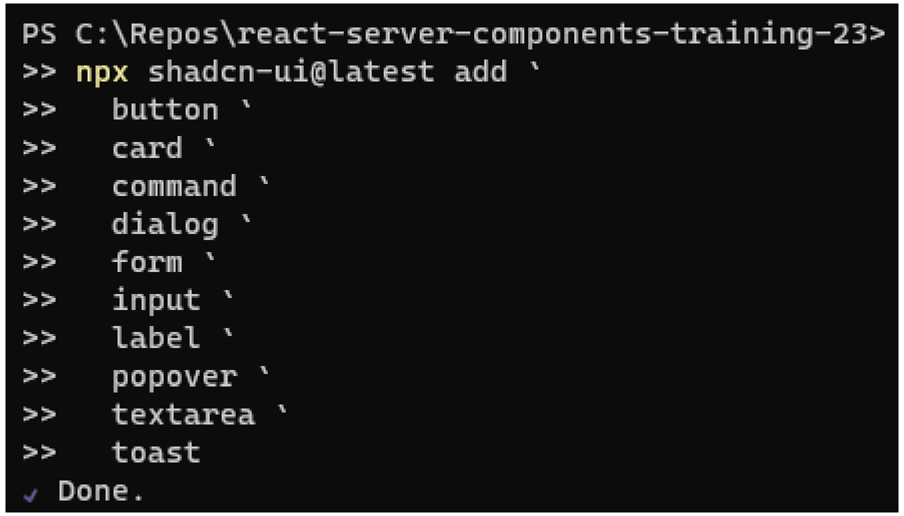
```
PS C:\Repos> npx create-next-app@latest react-server-components-training-23
Need to install the following packages:
create-next-app@14.0.1
Ok to proceed? (y)
√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like to use `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to customize the default import alias (@/*)? ... No / Yes
Creating a new Next.js app in C:\Repos\react-server-components-training-23.

Using npm.

Initializing project with template: app-tw


Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- autoprefixer
- postcss
- tailwindcss
- eslint
- eslint-config-next


added 330 packages, and audited 331 packages in 38s

116 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Initialized a git repository.

Success! Created react-server-components-training-23 at C:\Repos\react-server-components-training-23
```

Adding Shadcn support



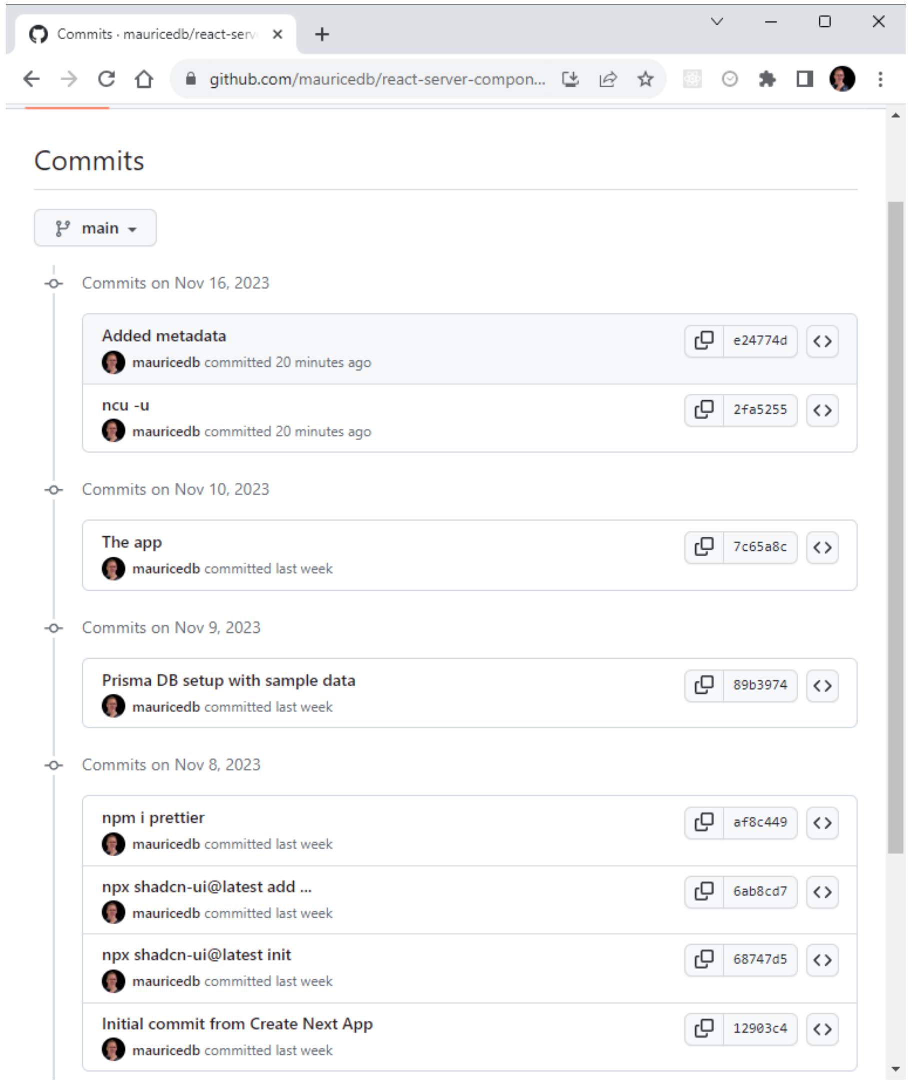
```
PS C:\Repos\react-server-components-training-23> npx shadcn-ui@latest init
√ Would you like to use TypeScript (recommended)? ... no / yes
√ Which style would you like to use? » Default
√ Which color would you like to use as base color? » Slate
√ Where is your global CSS file? ... src/app/globals.css
√ Would you like to use CSS variables for colors? ... no / yes
√ Where is your tailwind.config.js located? ... tailwind.config.ts
√ Configure the import alias for components: ... @/components
√ Configure the import alias for utils: ... @/lib/utils
√ Are you using React Server Components? ... no / yes
√ Write configuration to components.json. Proceed? ... yes

✓ Writing components.json...
✓ Initializing project...
✓ Installing dependencies...

Success! Project initialization completed.
```

Adding Shadcn components



```
PS C:\Repos\react-server-components-training-23>
>> npx shadcn-ui@latest add `
>>   button `
>>   card `
>>   command `
>>   dialog `
>>   form `
>>   input `
>>   label `
>>   popover `
>>   textarea `
>>   toast
✓ Done.
```

The changes


Commits · mauricedb/react-serv
github.com/mauricedb/react-server-compon...
main

Commits

Commits on Nov 16, 2023

Added metadata
mauricedb committed 20 minutes ago
e24774d

ncu -u
mauricedb committed 20 minutes ago
2fa5255

Commits on Nov 10, 2023

The app
mauricedb committed last week
7c65a8c

Commits on Nov 9, 2023

Prisma DB setup with sample data
mauricedb committed last week
89b3974

Commits on Nov 8, 2023

npm i prettier
mauricedb committed last week
af8c449

npx shadcn-ui@latest add ...
mauricedb committed last week
6ab8cd7

npx shadcn-ui@latest init
mauricedb committed last week
68747d5

Initial commit from Create Next App
mauricedb committed last week
12903c4

See you in the next video

Cloning the GitHub repository

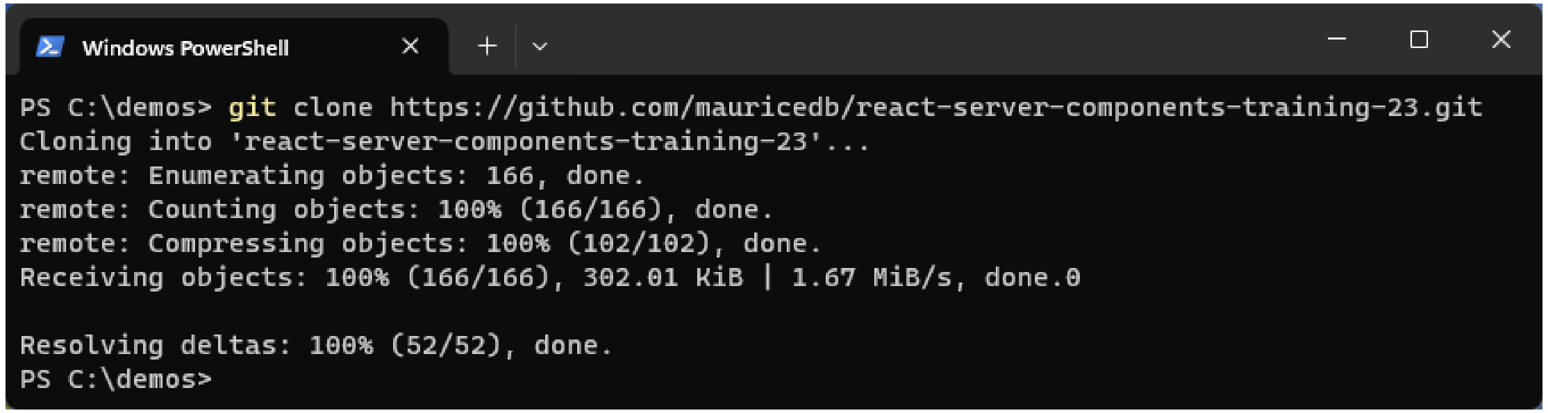
BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

Cloning the GitHub repository

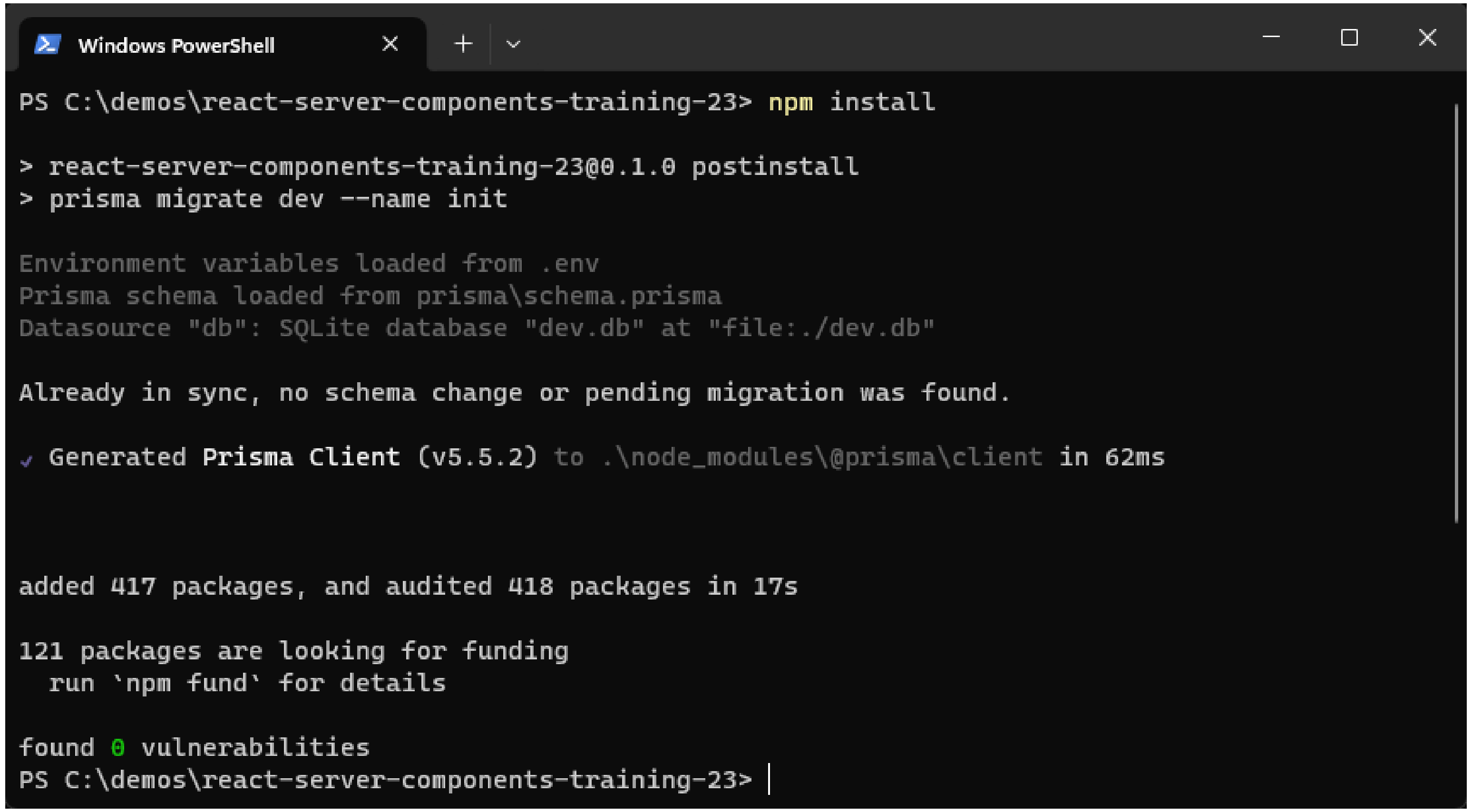
And running the application

Clone the GitHub Repository



```
PS C:\demos> git clone https://github.com/mauricedb/react-server-components-training-23.git
Cloning into 'react-server-components-training-23'...
remote: Enumerating objects: 166, done.
remote: Counting objects: 100% (166/166), done.
remote: Compressing objects: 100% (102/102), done.
Receiving objects: 100% (166/166), 302.01 KiB | 1.67 MiB/s, done.0

Resolving deltas: 100% (52/52), done.
PS C:\demos>
```

Install NPM Packages

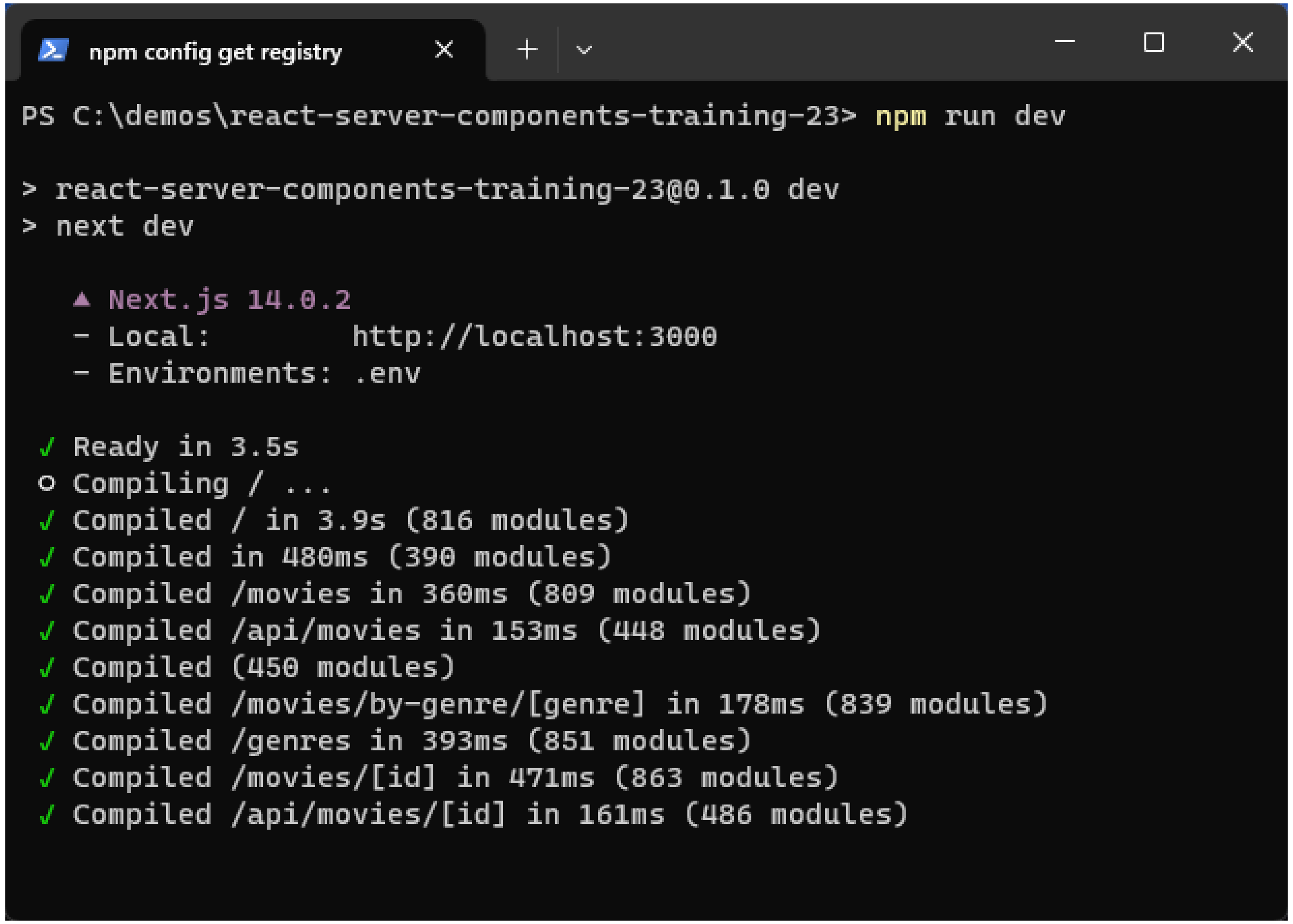

Windows PowerShell

```
PS C:\demos\react-server-components-training-23> npm install

> react-server-components-training-23@0.1.0 postinstall
> prisma migrate dev --name init

Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Datasource "db": SQLite database "dev.db" at "file:./dev.db"

Already in sync, no schema change or pending migration was found.

✓ Generated Prisma Client (v5.5.2) to .\node_modules\@prisma\client in 62ms



added 417 packages, and audited 418 packages in 17s

121 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\demos\react-server-components-training-23>
```

Start branch

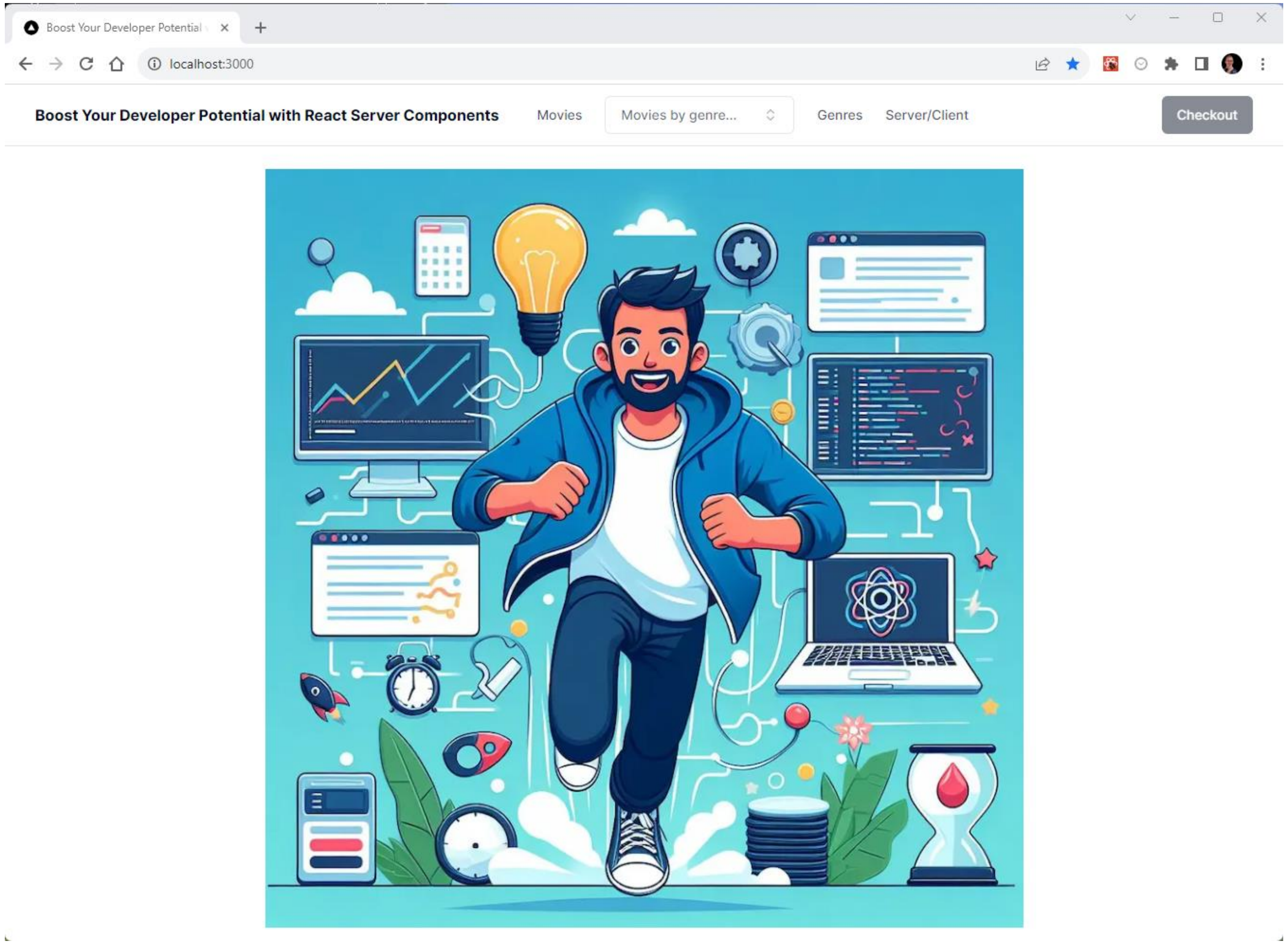
- Start with the **00-start** branch
  - `git checkout --track origin/00-start`

Start the application



```
PS C:\demos\react-server-components-training-23> npm run dev

> react-server-components-training-23@0.1.0 dev
> next dev

   ▲ Next.js 14.0.2
   - Local:        http://localhost:3000
   - Environments: .env

 ✓ Ready in 3.5s
 ○ Compiling / ...
 ✓ Compiled / in 3.9s (816 modules)
 ✓ Compiled in 480ms (390 modules)
 ✓ Compiled /movies in 360ms (809 modules)
 ✓ Compiled /api/movies in 153ms (448 modules)
 ✓ Compiled (450 modules)
 ✓ Compiled /movies/by-genre/[genre] in 178ms (839 modules)
 ✓ Compiled /genres in 393ms (851 modules)
 ✓ Compiled /movies/[id] in 471ms (863 modules)
 ✓ Compiled /api/movies/[id] in 161ms (486 modules)
```

The application


Boost Your Developer Potential
localhost:3000
Boost Your Developer Potential with React Server Components
Movies
Movies by genre...
Genres
Server/Client
Checkout

See you in the next video

What are React Server Components?

BOOST YOUR DEVELOPER POTENTIAL

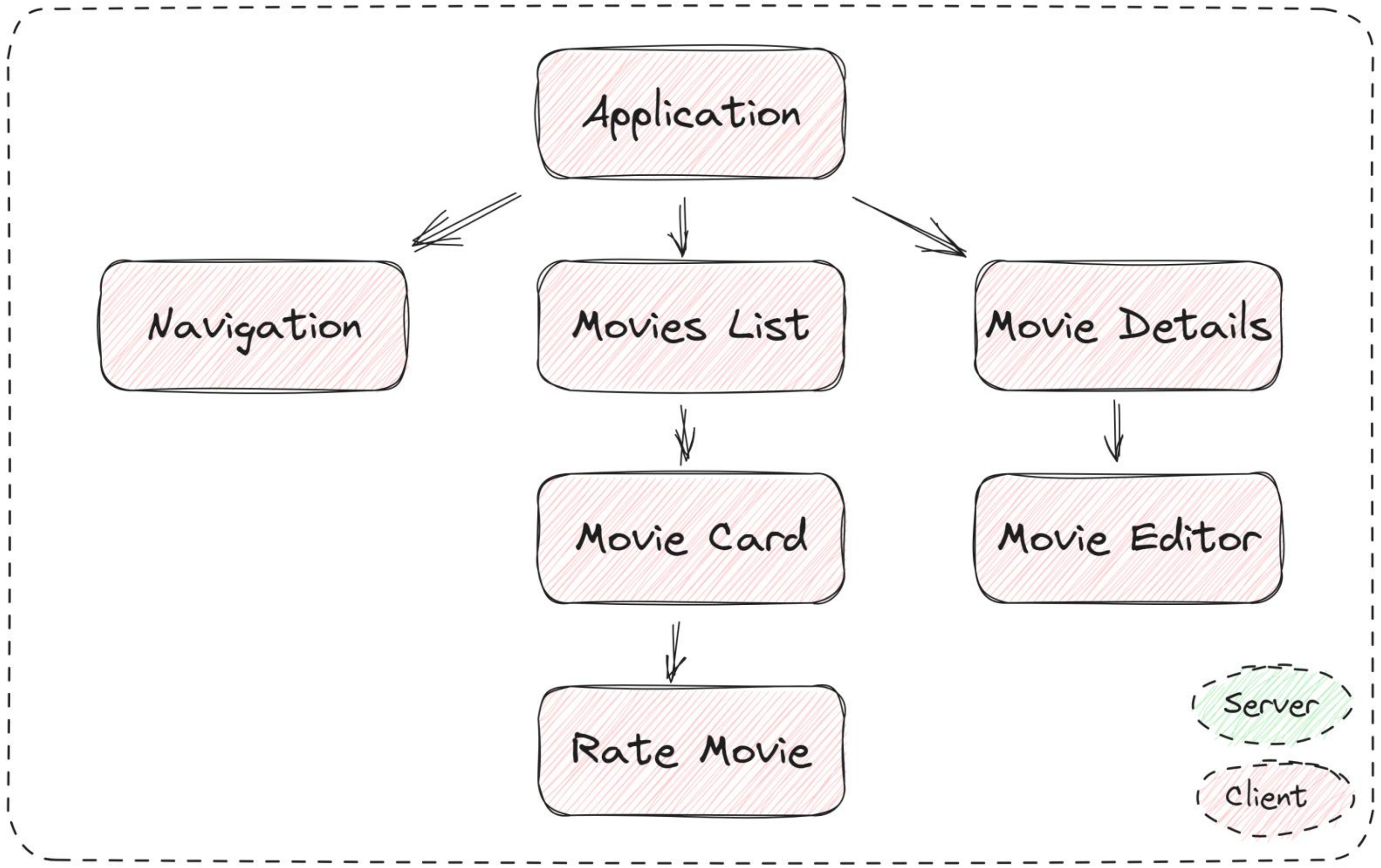
WITH REACT SERVER COMPONENTS

What are React Server Components?

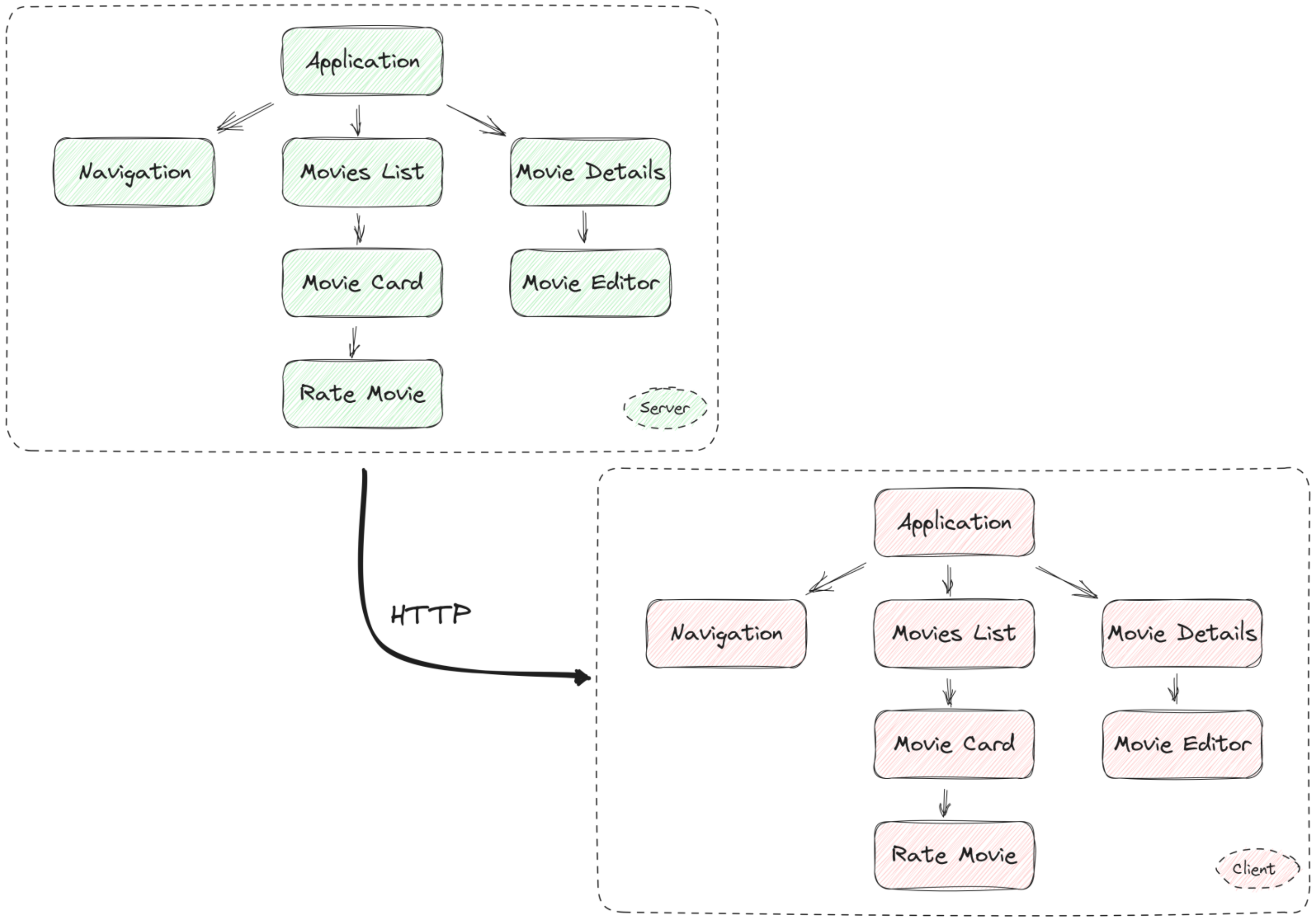
React Server Components

- React Server Components (RSC) **only execute on the server**
  - Traditionally React components always execute in the browser
- RSC are **not the same as Server Side Rendering**
  - With SSR components are executed both on the client and server
- Applications are a **combination of server and client components**
- The result: The back and front-end **code are more integrated**
  - Leading to **better type safety** ☺

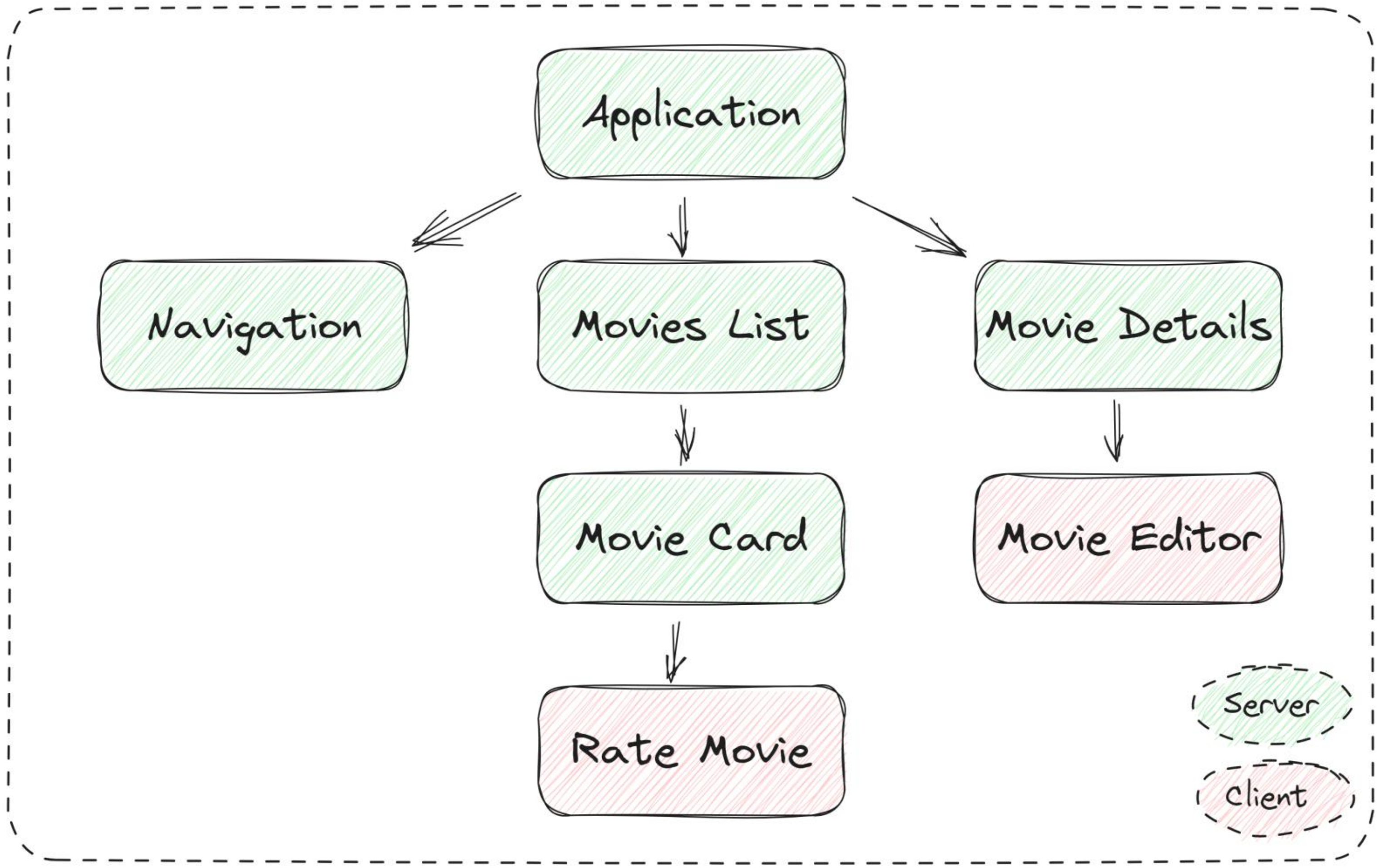
Before RSC


Application
Navigation
Movies List
Movie Details
Movie Card
Movie Editor
Rate Movie
Server
Client

Server Side Rendering



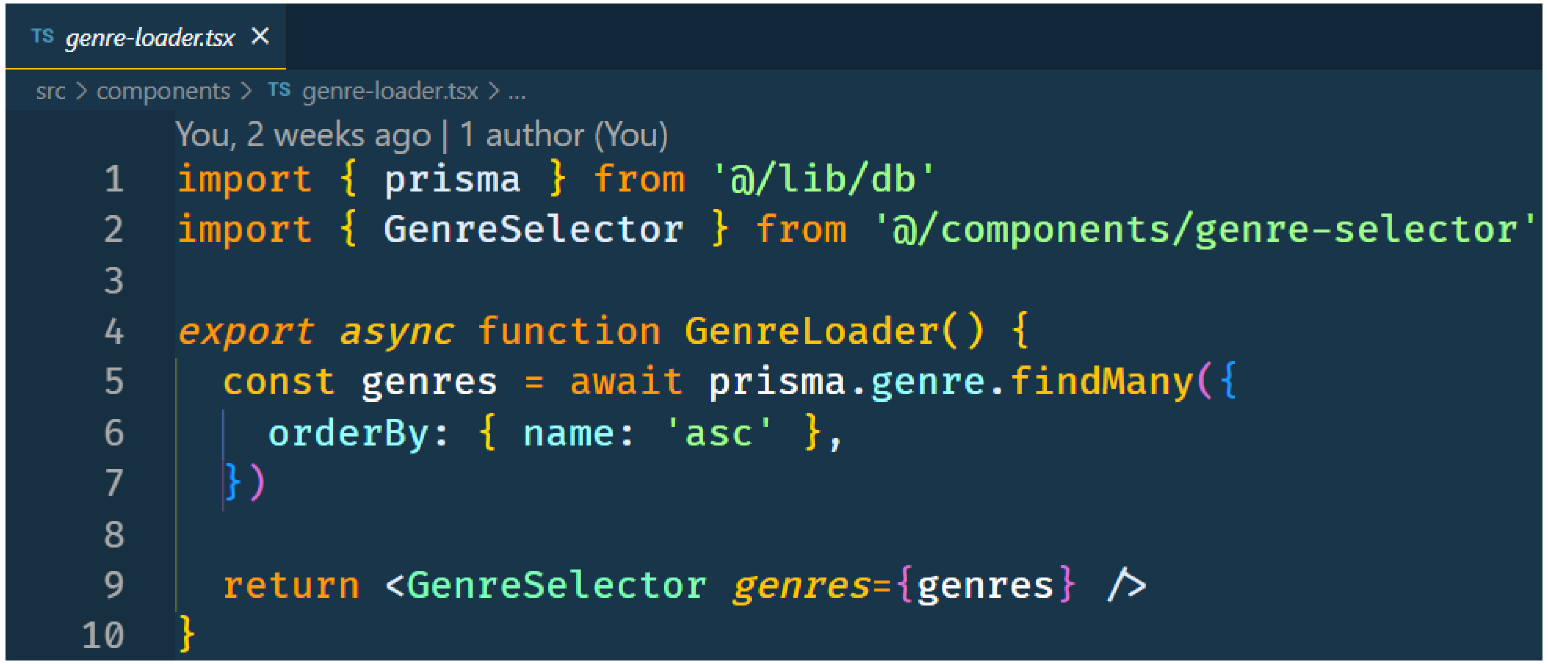
With RSC


Application
Navigation
Movies List
Movie Details
Movie Card
Movie Editor
Rate Movie
Server
Client

React Server Components

- Server components can be **asynchronous**
  - Great to load data from some API
- Server components **render just once**
  - No re-rendering with state changes or event handling
- The server component **code is not send to the browser**
  - Can safely use secure API key's etc.
  - Smaller bundle sizes
- React Server Components **can be authored in TypeScript**
  - RSC require TypeScript 5.1 or later

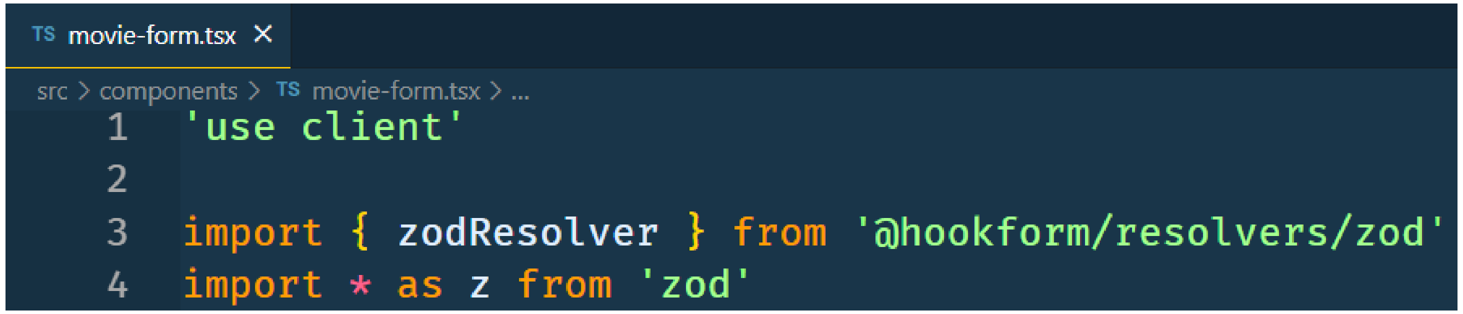
React Server Component


genre-loader.tsx
src > components > genre-loader.tsx > ...
You, 2 weeks ago | 1 author (You)

```tsx
import { prisma } from '@/lib/db'
import { GenreSelector } from '@/components/genre-selector'

export async function GenreLoader() {
  const genres = await prisma.genre.findMany({
    orderBy: { name: 'asc' },
  })

  return <GenreSelector genres={genres} />
}
```

React Client Components

- **Server components can render both server and client components**
  - Client components can only render other client components
- Adding **'use client'** to the top of a component makes it a client component
  - Used as a directive for the bundler to include this in the client JS bundle
- A client component is **still executed on the server** as part of SSR
  - When using Next.js



movie-form.tsx

src > components > movie-form.tsx > ...

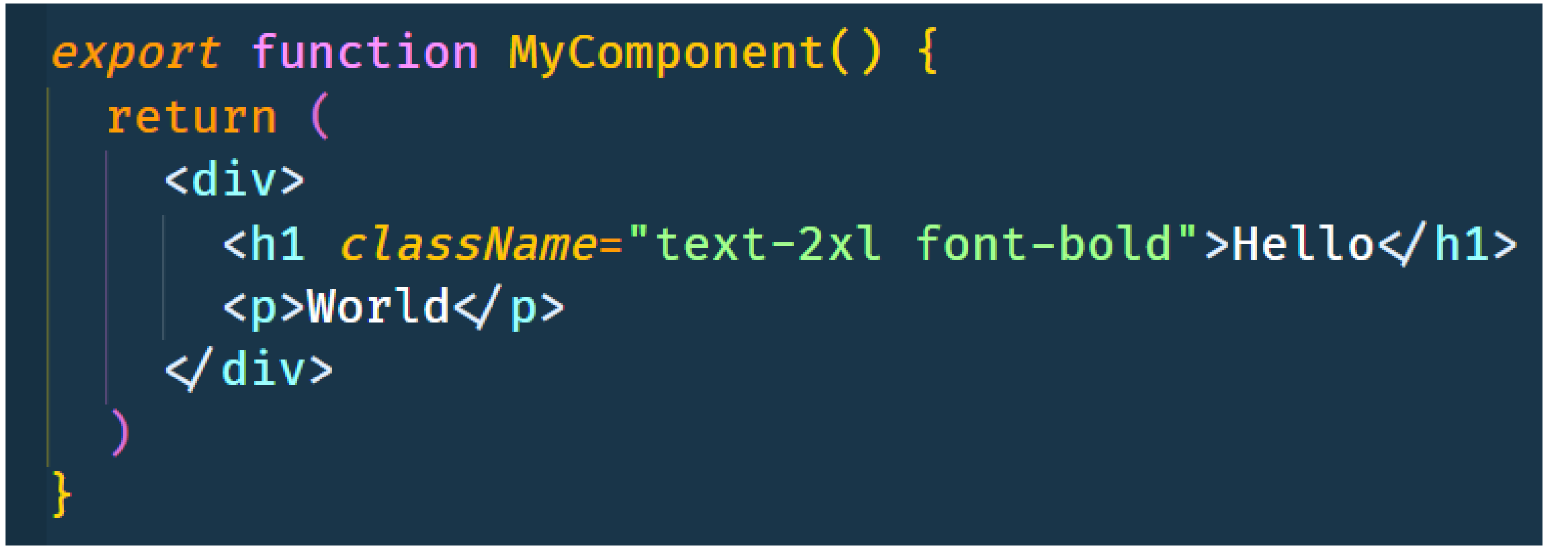
```
'use client'

import { zodResolver } from '@hookform/resolvers/zod'
import * as z from 'zod'
```

Rendering RSC’s

- **React Server Components are only rendered on the server**
  - And shipped to the client as a JSON like structure
  - The React Server Component Payload
- The client then **injects** these JSON objects **into the React tree**
  - Where it would previously have rendered these components themself

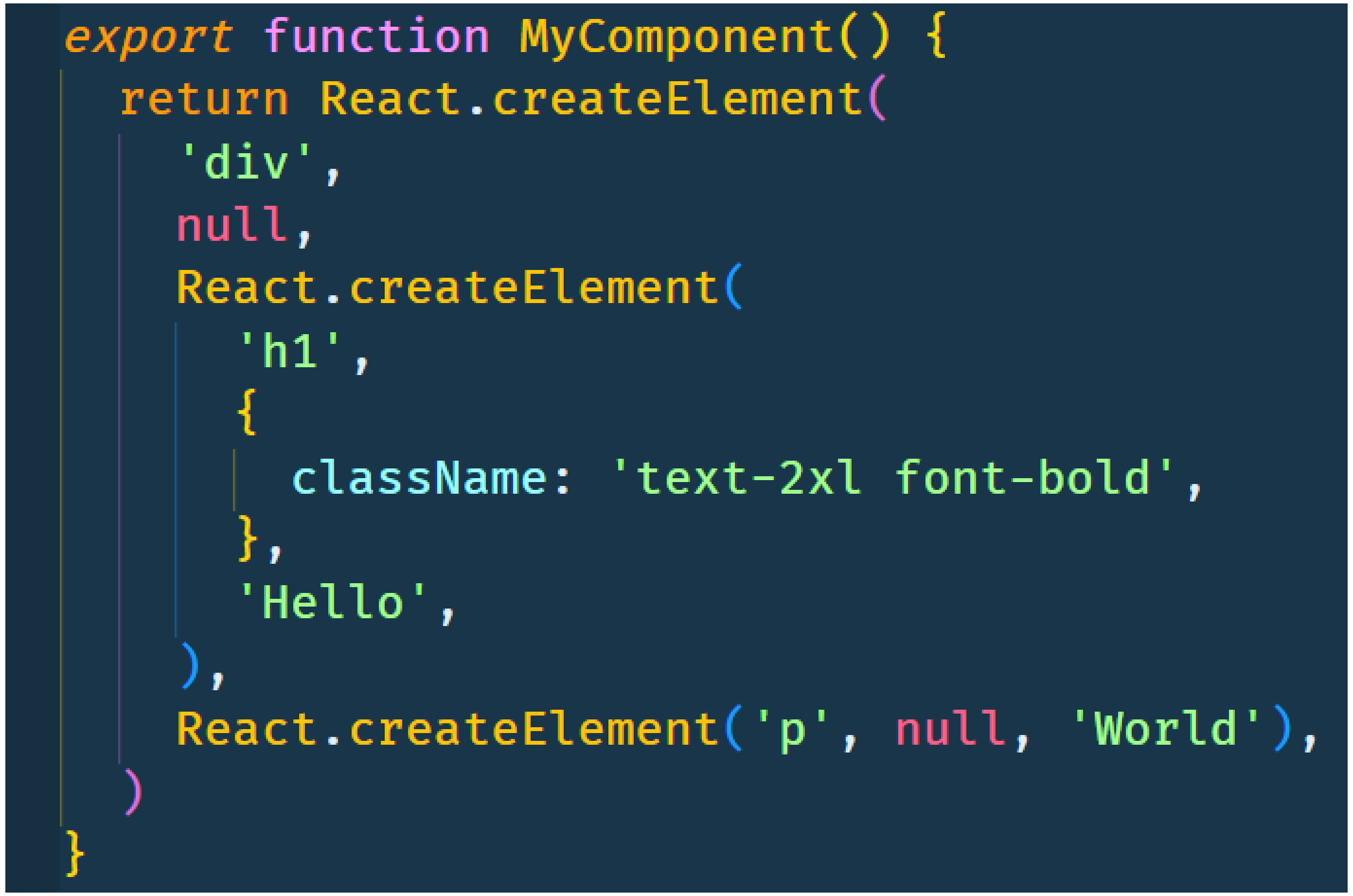
- ☞ **React already used a 2 step process** ☜
  - Components render to a virtual DOM
    - Just a series of JavaScript objects
  - Reconciliation maps the virtual DOM to the browser DOM
    - Or an HTML stream in the case or Server Side Rendering

Write JSX



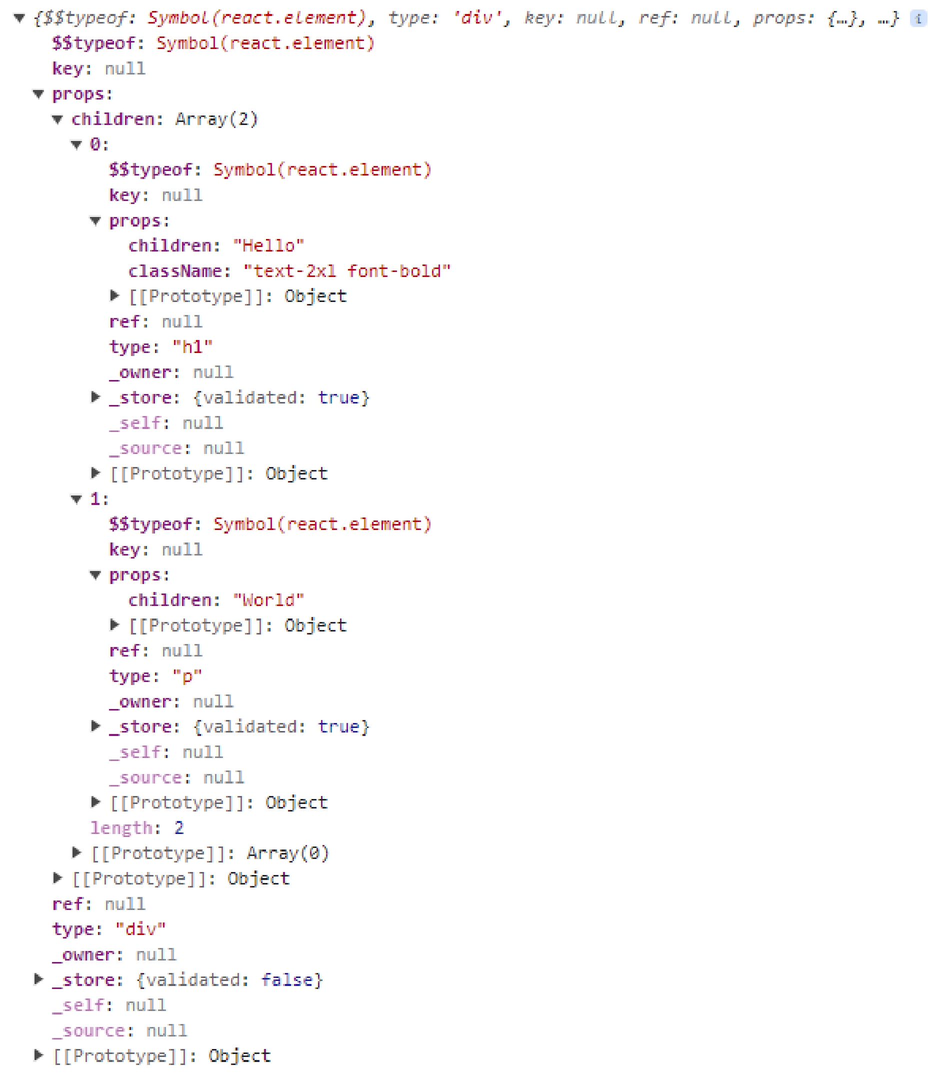
```
export function MyComponent() {
  return (
    <div>
      <h1 className="text-2xl font-bold">Hello</h1>
      <p>World</p>
    </div>
  )
}
```

Turned into createElement()



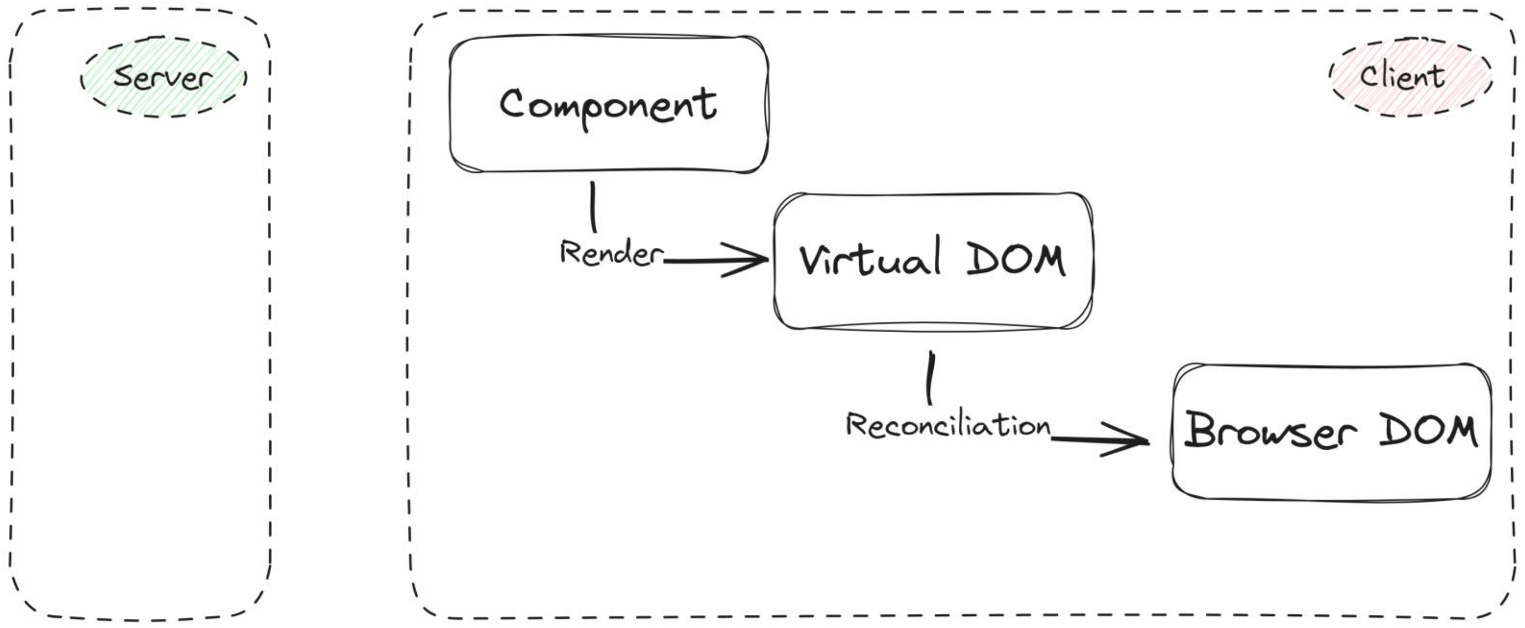
```
export function MyComponent() {
  return React.createElement(
    'div',
    null,
    React.createElement(
      'h1',
      {
        className: 'text-2xl font-bold',
      },
      'Hello',
    ),
    React.createElement('p', null, 'World'),
  )
}
```

Returns the Virtual DOM

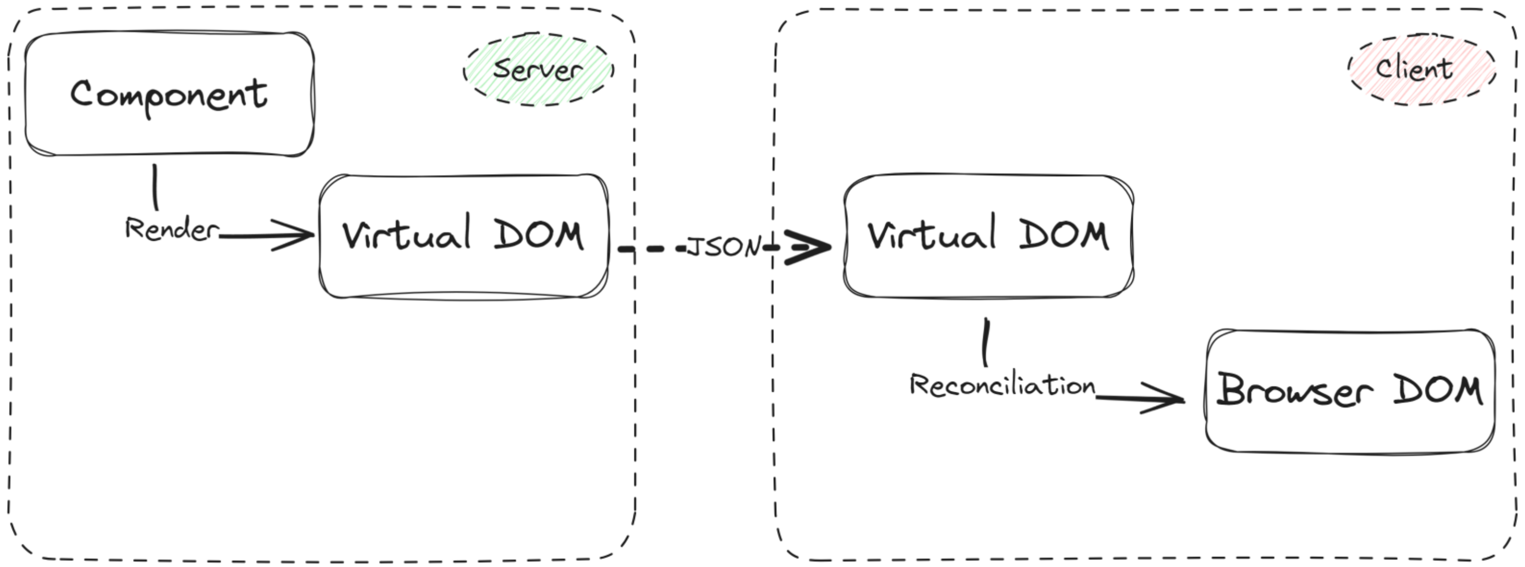


```
▼ {$$typeof: Symbol(react.element), type: 'div', key: null, ref: null, props: {…}, …} i
    $$typeof: Symbol(react.element)
    key: null
  ▼ props:
    ▼ children: Array(2)
      ▼ 0:
          $$typeof: Symbol(react.element)
          key: null
        ▼ props:
            children: "Hello"
            className: "text-2xl font-bold"
          ▶ [[Prototype]]: Object
          ref: null
          type: "h1"
          _owner: null
        ▶ _store: {validated: true}
          _self: null
          _source: null
        ▶ [[Prototype]]: Object
      ▼ 1:
          $$typeof: Symbol(react.element)
          key: null
        ▼ props:
            children: "World"
          ▶ [[Prototype]]: Object
          ref: null
          type: "p"
          _owner: null
        ▶ _store: {validated: true}
          _self: null
          _source: null
        ▶ [[Prototype]]: Object
        length: 2
      ▶ [[Prototype]]: Array(0)
    ▶ [[Prototype]]: Object
    ref: null
    type: "div"
    _owner: null
  ▶ _store: {validated: false}
    _self: null
    _source: null
  ▶ [[Prototype]]: Object
```

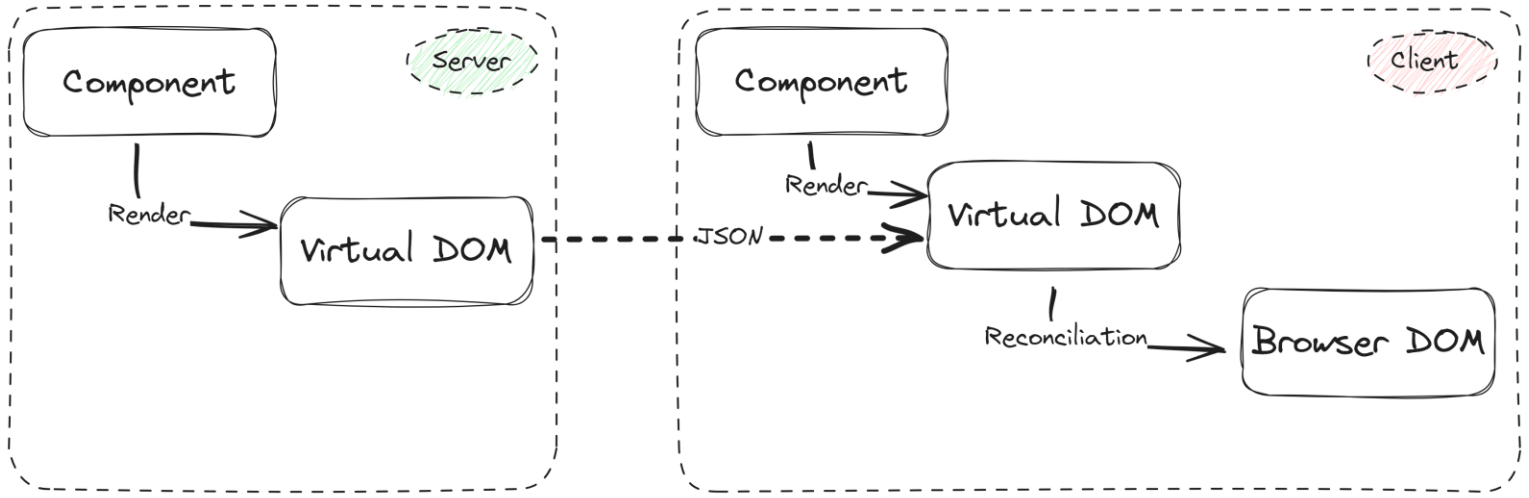
Before RSC


Server
Client
Component
Render
Virtual DOM
Reconciliation
Browser DOM

With RSC


Server
Component
Render
Virtual DOM
JSON
Client
Virtual DOM
Reconciliation
Browser DOM

With RSC and RCC



© ABL - The Problem Solver

Async transport

- RSC’s are **streamed asynchronously** to the client
  - Enables using Suspense boundaries while loading

Code bundling

- **Multiple JavaScript bundles** have to be made
    - The client and server have different code bundles
- **Server Component code is never executed on the client**
    - Can use **react-server-dom-webpack** or a similar package

See you in the next video

Next.js and the App Router

BOOST YOUR DEVELOPER POTENTIAL

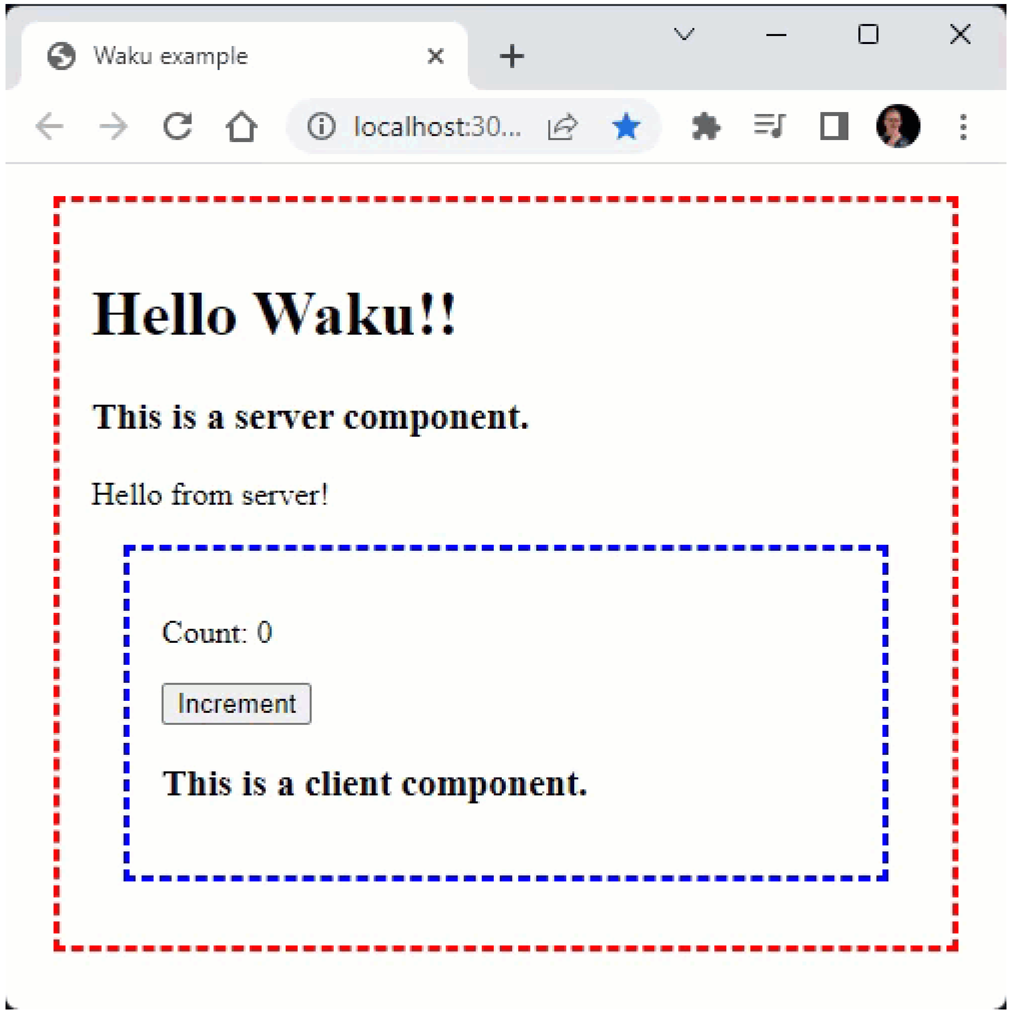
WITH REACT SERVER COMPONENTS

Next.js and the App Router

Next.js and the App Router

- **React is no longer just a client side library**
  - We need additional server side capabilities
  - As well as additional code bundling options
- **Next.js is the best production option available**
  - Shopify Hydrogen is also an option
  - ☞ Remix 2 doesn't support RSC yet ☜
- There are also more **experimental option**s
  - Waku from Daishi Kato
  - React Server Components Demo from the React team

Waku



Hello Waku!!

This is a server component.

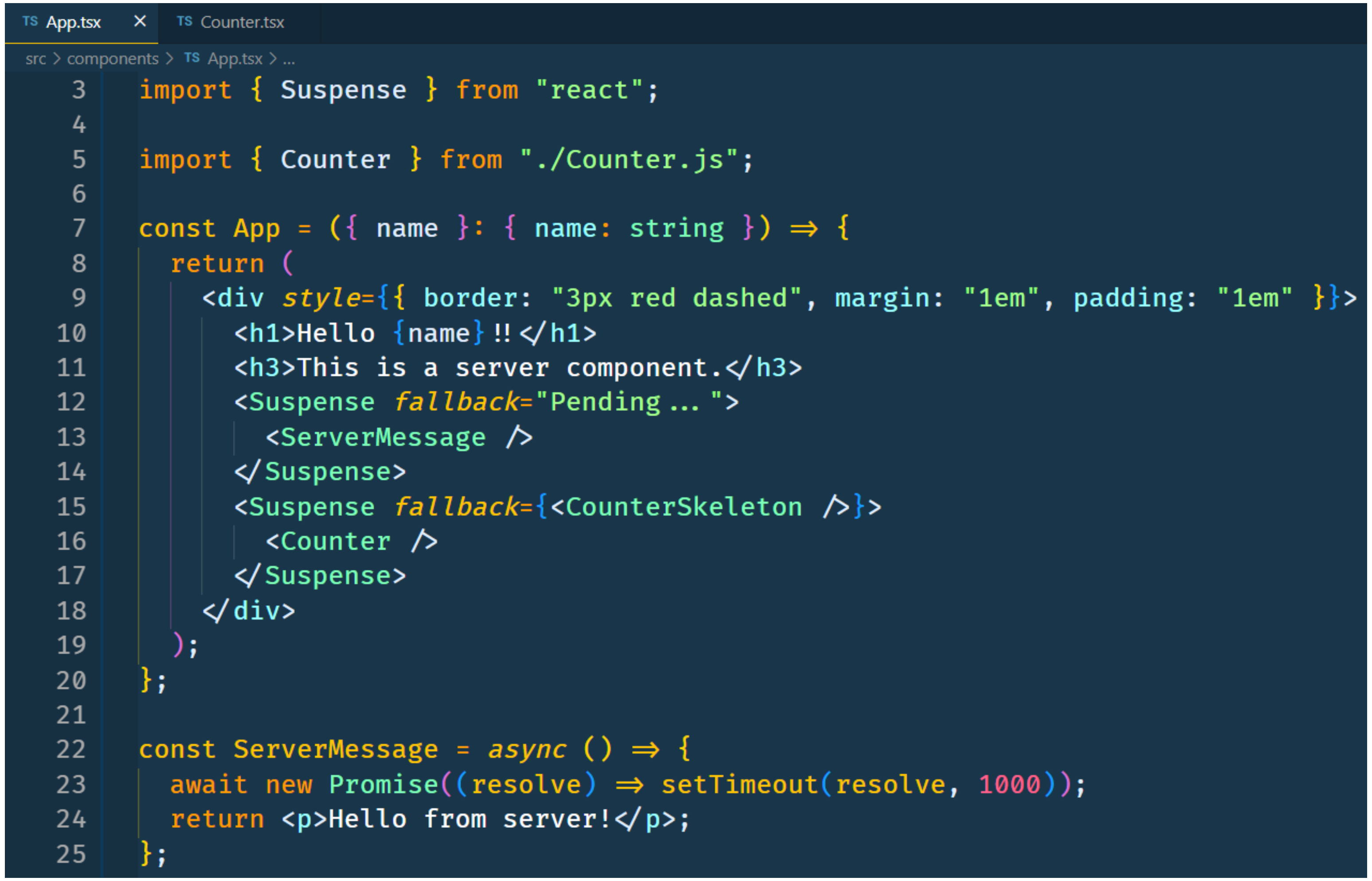
Hello from server!

Count: 0

Increment

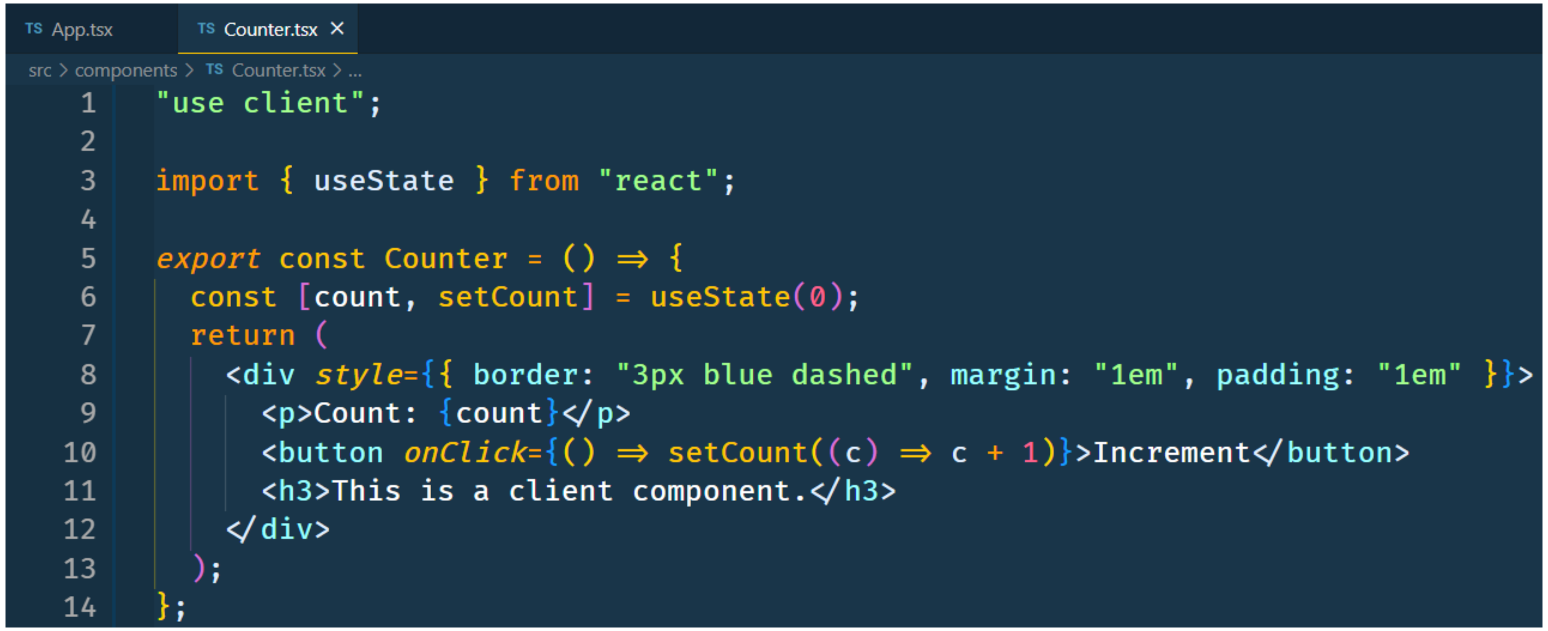
This is a client component.

Server Component



```tsx
import { Suspense } from "react";

import { Counter } from "./Counter.js";

const App = ({ name }: { name: string }) => {
  return (
    <div style={{ border: "3px red dashed", margin: "1em", padding: "1em" }}>
      <h1>Hello {name} !!</h1>
      <h3>This is a server component.</h3>
      <Suspense fallback="Pending ... ">
        <ServerMessage />
      </Suspense>
      <Suspense fallback={<CounterSkeleton />}>
        <Counter />
      </Suspense>
    </div>
  );
};

const ServerMessage = async () => {
  await new Promise((resolve) => setTimeout(resolve, 1000));
  return <p>Hello from server!</p>;
};
```

Client Component



```tsx
"use client";

import { useState } from "react";

export const Counter = () => {
  const [count, setCount] = useState(0);
  return (
    <div style={{ border: "3px blue dashed", margin: "1em", padding: "1em" }}>
      <p>Count: {count}</p>
      <button onClick={() => setCount((c) => c + 1)}>Increment</button>
      <h3>This is a client component.</h3>
    </div>
  );
};
```

See you in the next video

Turning a React Client Component into a Server Component

BOOST YOUR DEVELOPER POTENTIAL

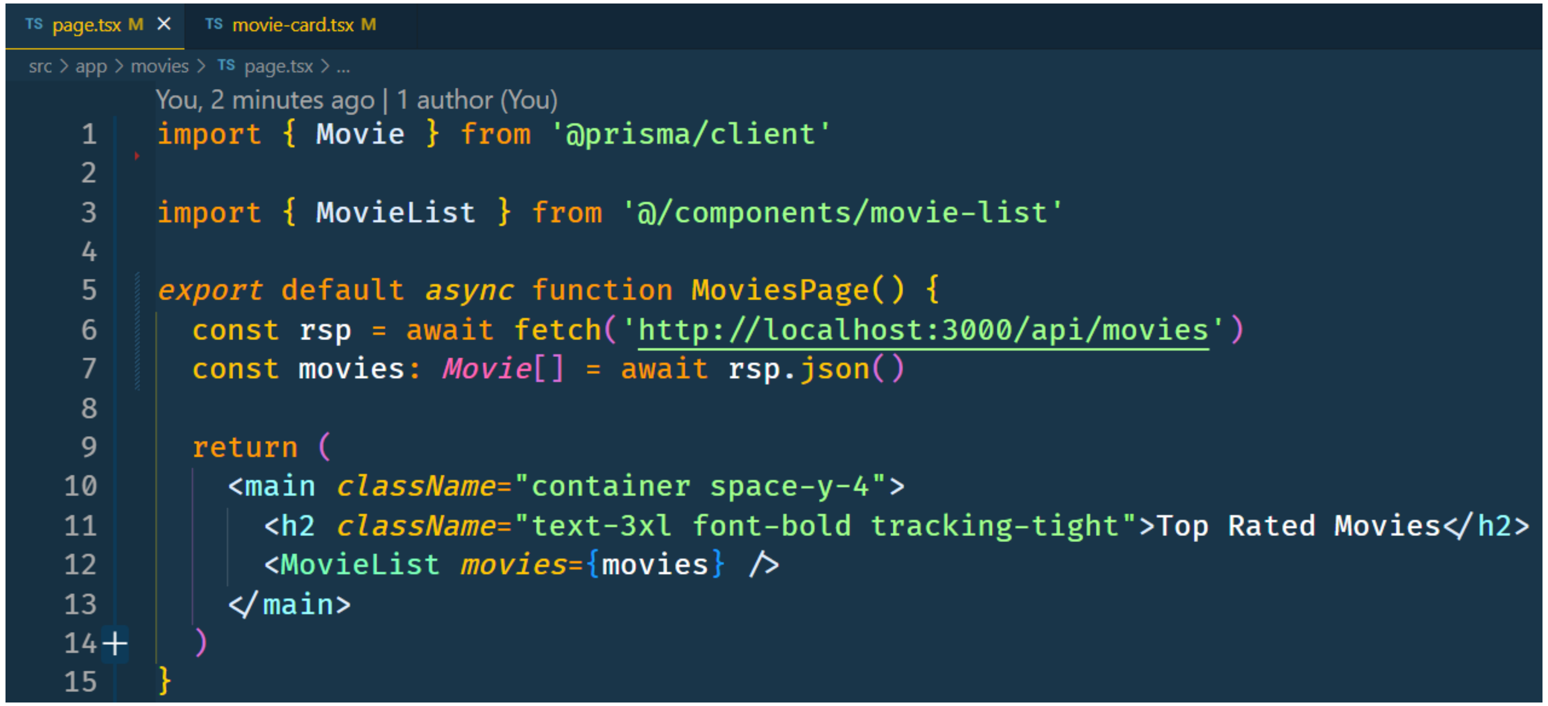
WITH REACT SERVER COMPONENTS

Turning a React Client Component into a Server Component

Client Component to Server Component

- **React Server Components normally perform better**
  - Only render once on the server
  - The code doesn’t need to be shipped to the browser
- **Can be async and await data** to be fetched
  - No need for a render/effect/re-render cycle in the browser
- **Components that don’t need client capabilities should be SRC’s**
  - State, effects, browser API’s etc. are client requirements

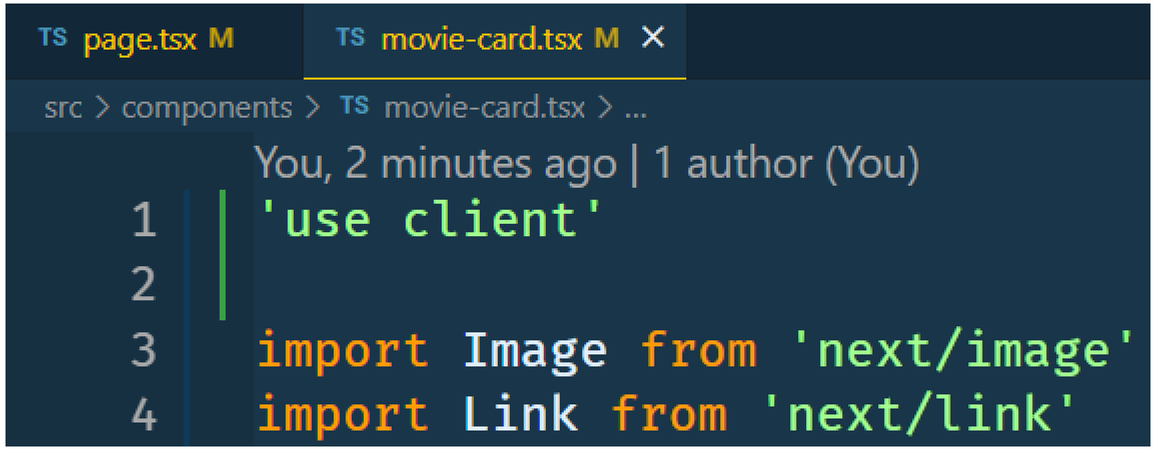
movies /page.tsx



TS page.tsx M
TS movie-card.tsx M
src > app > movies > TS page.tsx > ...
You, 2 minutes ago | 1 author (You)

```tsx
import { Movie } from '@prisma/client'

import { MovieList } from '@/components/movie-list'

export default async function MoviesPage() {
  const rsp = await fetch('http://localhost:3000/api/movies')
  const movies: Movie[] = await rsp.json()

  return (
    <main className="container space-y-4">
      <h2 className="text-3xl font-bold tracking-tight">Top Rated Movies</h2>
      <MovieList movies={movies} />
    </main>
  )
}
```

movie-card.tsx


TS page.tsx M
TS movie-card.tsx M ×
src > components > TS movie-card.tsx > ...
You, 2 minutes ago | 1 author (You)

```
'use client'

import Image from 'next/image'
import Link from 'next/link'
```

See you in the next video

Updating the movies by genre and the movie details pages

BOOST YOUR DEVELOPER POTENTIAL

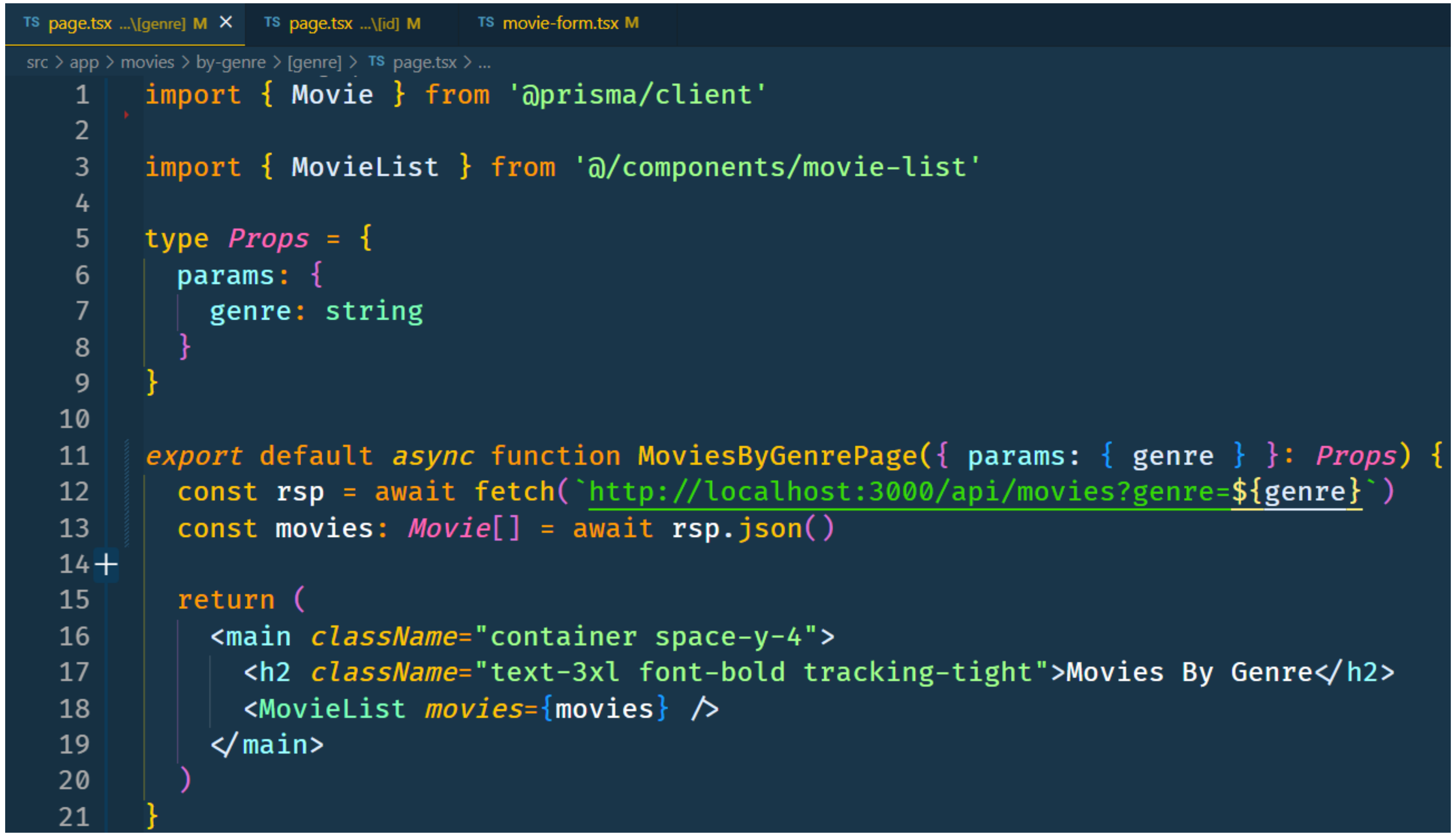
WITH REACT SERVER COMPONENTS

Updating the movies by genre and the movie details pages

Updating the movies by genre and the movie details pages

- The **MoviesByGenrePage** and **MoviePage** also fetch on the client
    - Lets make these React Server Components as well

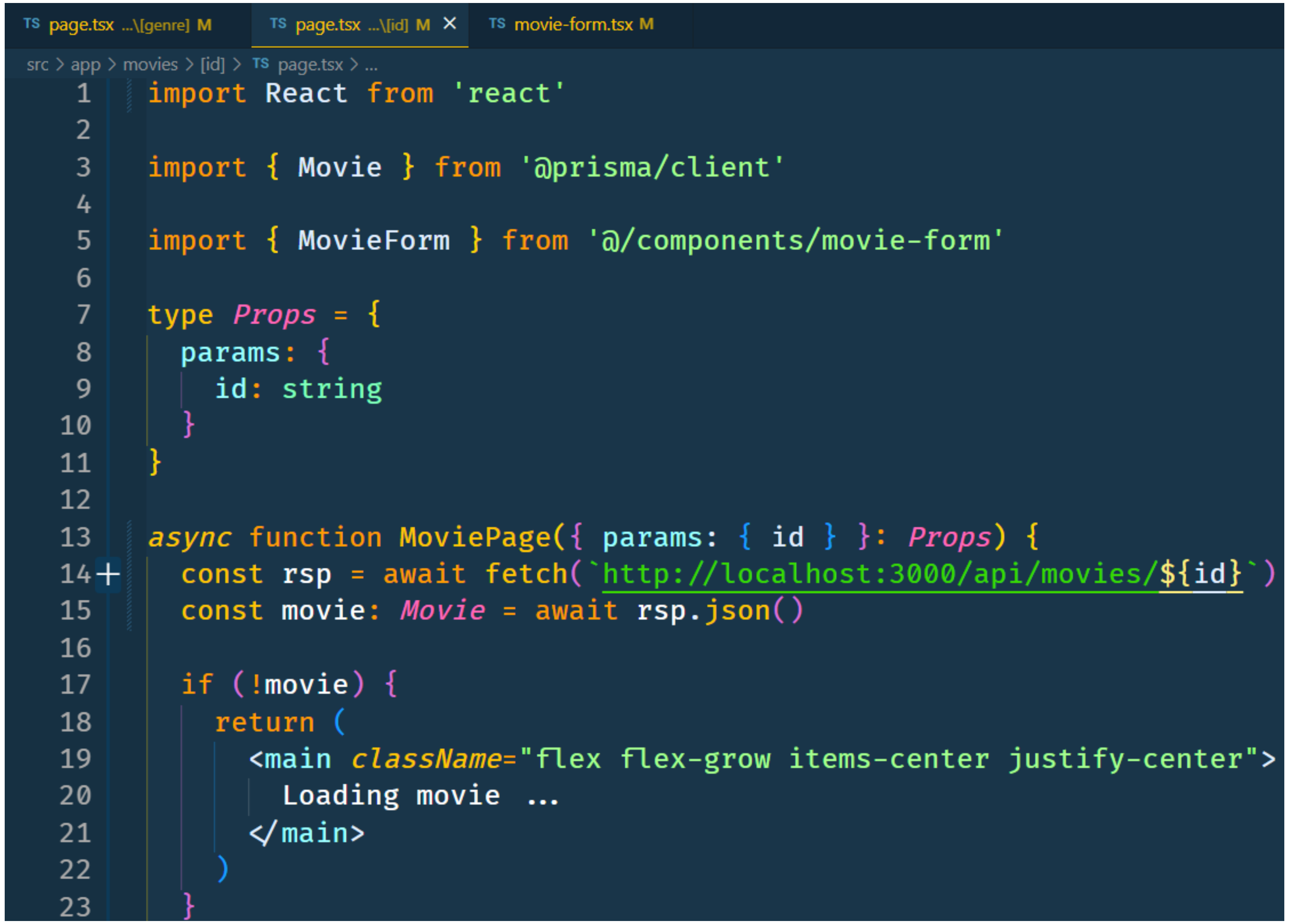
movies/by-genre /page.tsx



TS page.tsx ...\[genre] M
TS page.tsx ...\[id] M
TS movie-form.tsx M
src > app > movies > by-genre > [genre] > TS page.tsx > ...

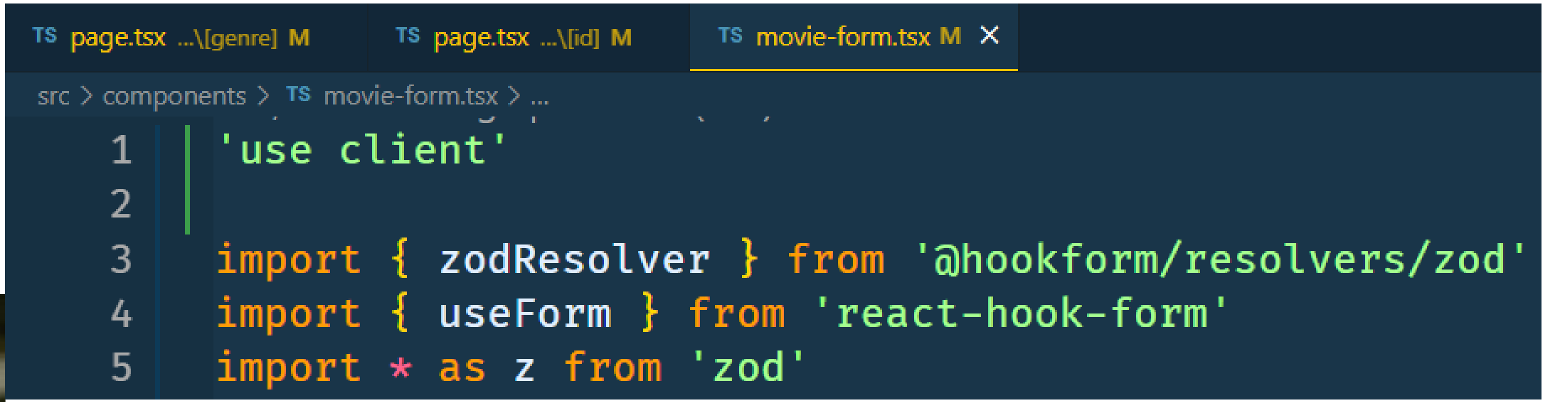
```tsx
import { Movie } from '@prisma/client'

import { MovieList } from '@/components/movie-list'

type Props = {
  params: {
    genre: string
  }
}

export default async function MoviesByGenrePage({ params: { genre } }: Props) {
  const rsp = await fetch(`http://localhost:3000/api/movies?genre=${genre}`)
  const movies: Movie[] = await rsp.json()

  return (
    <main className="container space-y-4">
      <h2 className="text-3xl font-bold tracking-tight">Movies By Genre</h2>
      <MovieList movies={movies} />
    </main>
  )
}
```

movies/[id] /page.tsx



```tsx
import React from 'react'

import { Movie } from '@prisma/client'

import { MovieForm } from '@/components/movie-form'

type Props = {
  params: {
    id: string
  }
}

async function MoviePage({ params: { id } }: Props) {
  const rsp = await fetch(`http://localhost:3000/api/movies/${id}`)
  const movie: Movie = await rsp.json()

  if (!movie) {
    return (
      <main className="flex flex-grow items-center justify-center">
        Loading movie ...
      </main>
    )
  }
```

movie-form.tsx


TS page.tsx ...\[genre] M
TS page.tsx ...\[id] M
TS movie-form.tsx M
src > components > TS movie-form.tsx > ...

```
'use client'

import { zodResolver } from '@hookform/resolvers/zod'
import { useForm } from 'react-hook-form'
import * as z from 'zod'
```

See you in the next video

Making the movie card mostly a RSC

BOOST YOUR DEVELOPER POTENTIAL

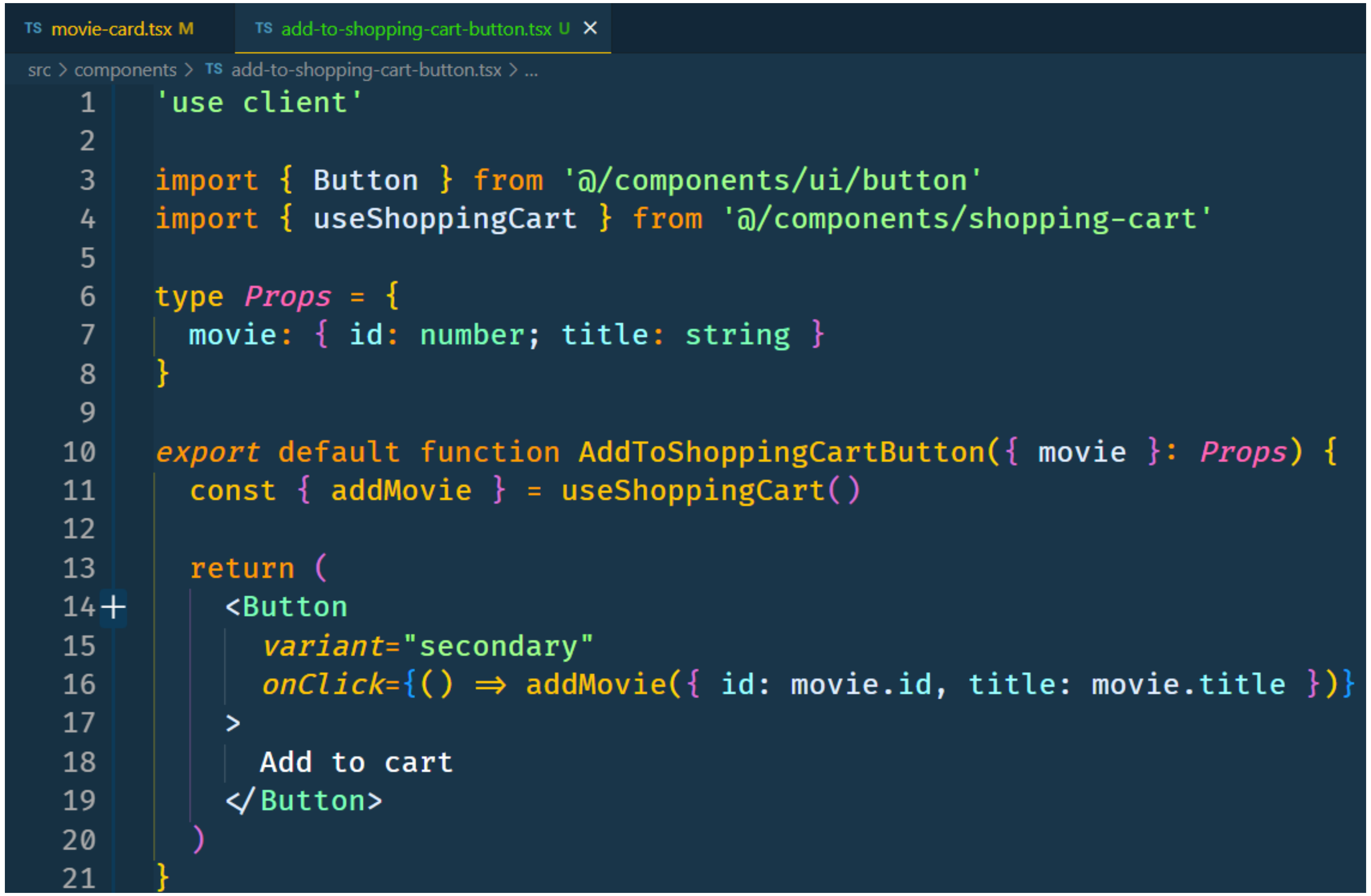
WITH REACT SERVER COMPONENTS

Making the movie card mostly a RSC

Making the movie card mostly a RSC

- The MovieCard is **not a very interactive component**
  - Only the Add to card button is interactive
- Recommended to **split it into two components**
  - An RSC with the movie card
  - And a client component with the button.
- Optionally: **use dynamic loading without SSR**
  - Clients with no JavaScript don't get a non operational button

add-to-shopping-cart-button.tsx

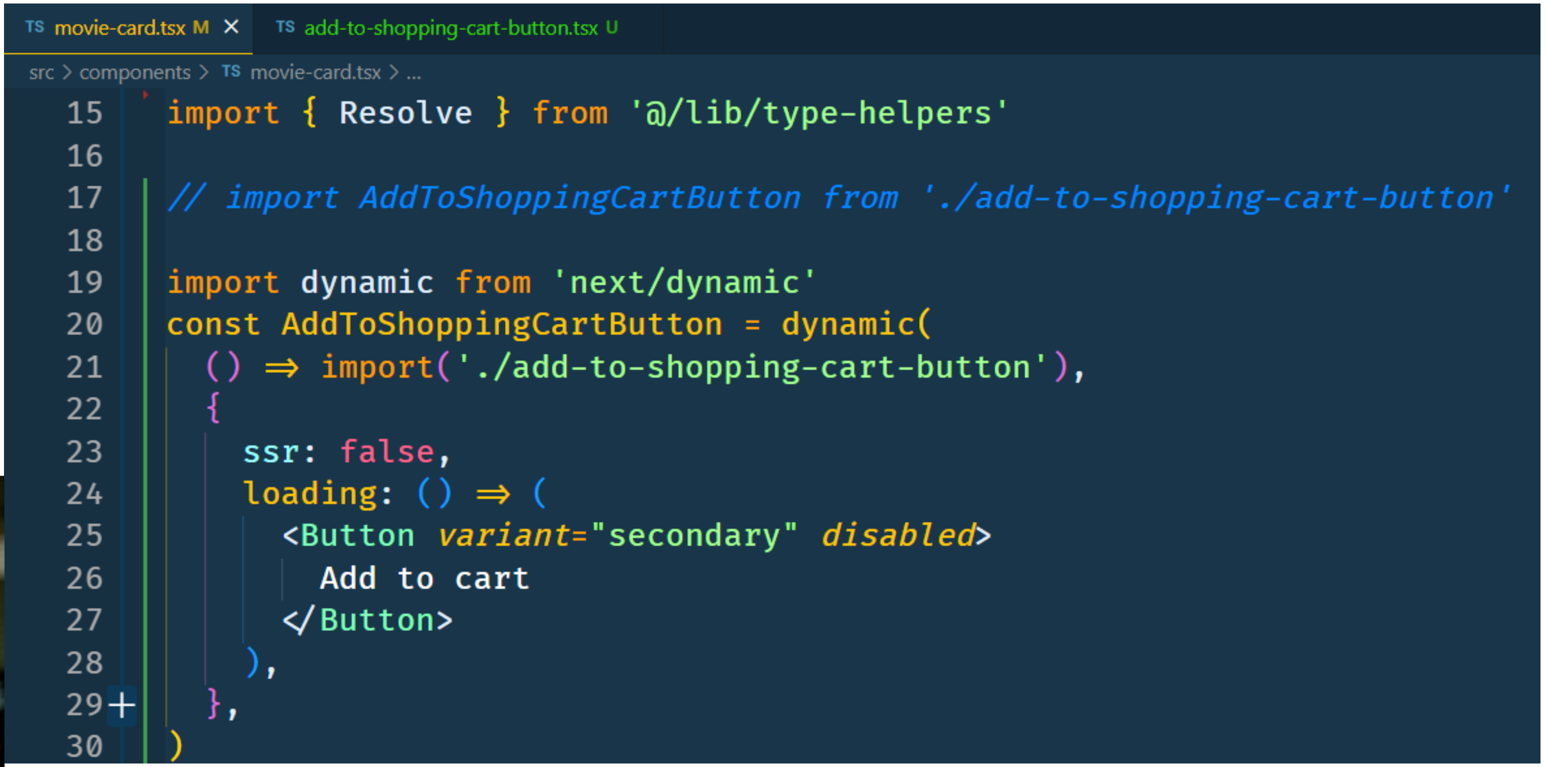


TS movie-card.tsx M   TS add-to-shopping-cart-button.tsx U ×

src > components > TS add-to-shopping-cart-button.tsx > ...

```tsx
'use client'

import { Button } from '@/components/ui/button'
import { useShoppingCart } from '@/components/shopping-cart'

type Props = {
  movie: { id: number; title: string }
}

export default function AddToShoppingCartButton({ movie }: Props) {
  const { addMovie } = useShoppingCart()

  return (
    <Button
      variant="secondary"
      onClick={() => addMovie({ id: movie.id, title: movie.title })}
    >
      Add to cart
    </Button>
  )
}
```

movie-card.tsx


TS movie-card.tsx M ×
TS add-to-shopping-cart-button.tsx U
src > components > TS movie-card.tsx > ...

```
import { Resolve } from '@/lib/type-helpers'

// import AddToShoppingCartButton from './add-to-shopping-cart-button'

import dynamic from 'next/dynamic'
const AddToShoppingCartButton = dynamic(
  () => import('./add-to-shopping-cart-button'),
  {
    ssr: false,
    loading: () => (
      <Button variant="secondary" disabled>
        Add to cart
      </Button>
    ),
  },
)
```

See you in the next video

Updates and caching in Next.js

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

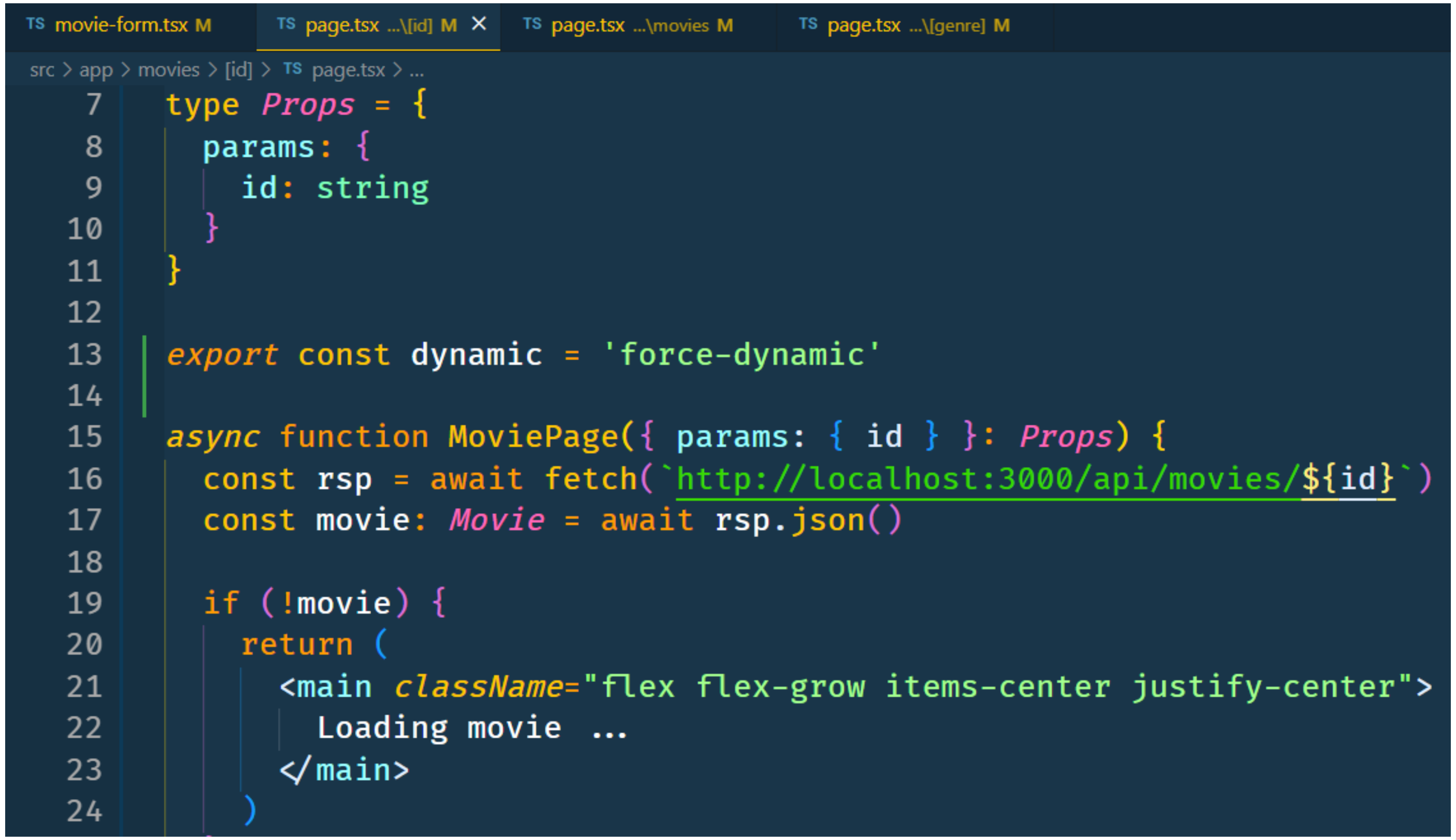
Updates and caching

In Next.js

Updates and caching

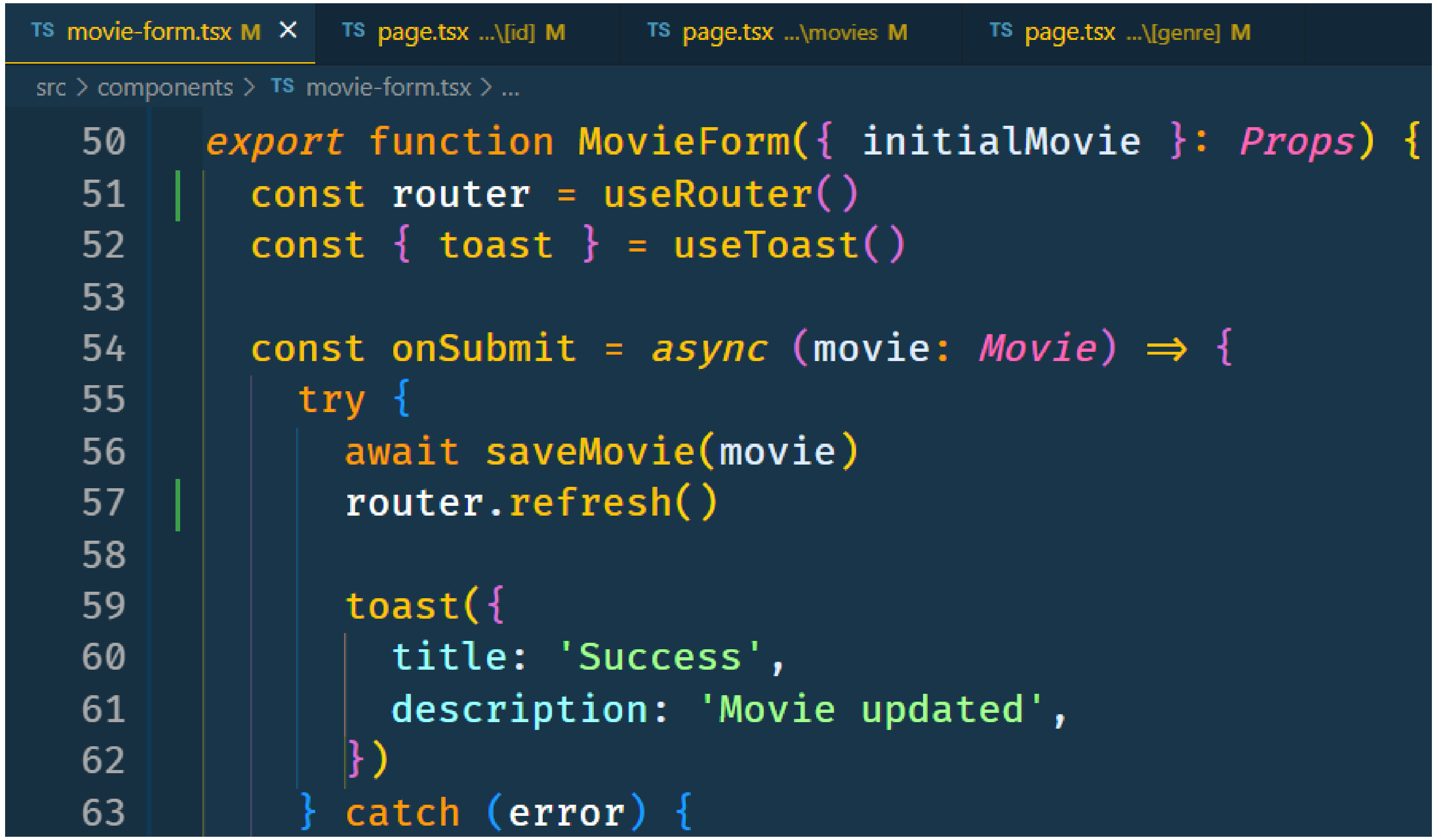
- Next.js does a lot of **optimizations using** **caching**
  - Both on the server and client
- The Next.js uses a **Data Cache** and **Full Router Cache** on the server
  - Use ***export const dynamic = 'force-dynamic'*** to make sure data on the server isn't cached
  - Can also be controlled at the fetch() level
- The Next.js uses a **Router Cache** on the client
  - Dynamically rendered routes are purged after 30 seconds
  - Call ***router.refresh()*** to immediately purge the cache
    - Make sure to use the router from '*next/navigation'*

movies/[id] /page.tsx



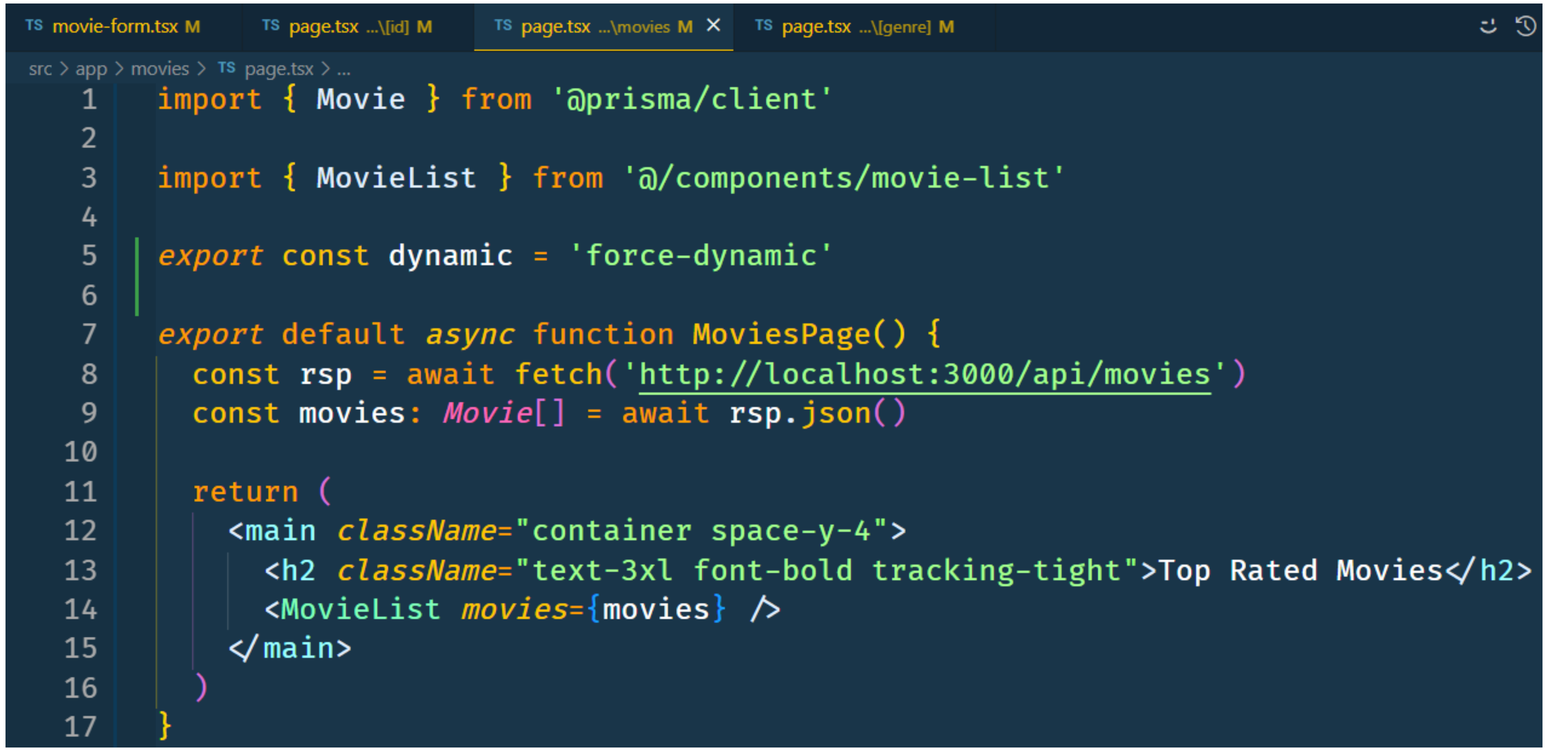
```
type Props = {
  params: {
    id: string
  }
}

export const dynamic = 'force-dynamic'

async function MoviePage({ params: { id } }: Props) {
  const rsp = await fetch(`http://localhost:3000/api/movies/${id}`)
  const movie: Movie = await rsp.json()

  if (!movie) {
    return (
      <main className="flex flex-grow items-center justify-center">
        Loading movie ...
      </main>
    )
```

movie-form.tsx


TS movie-form.tsx M ×
TS page.tsx ...\[id] M
TS page.tsx ...\movies M
TS page.tsx ...\[genre] M
src > components > TS movie-form.tsx > ...

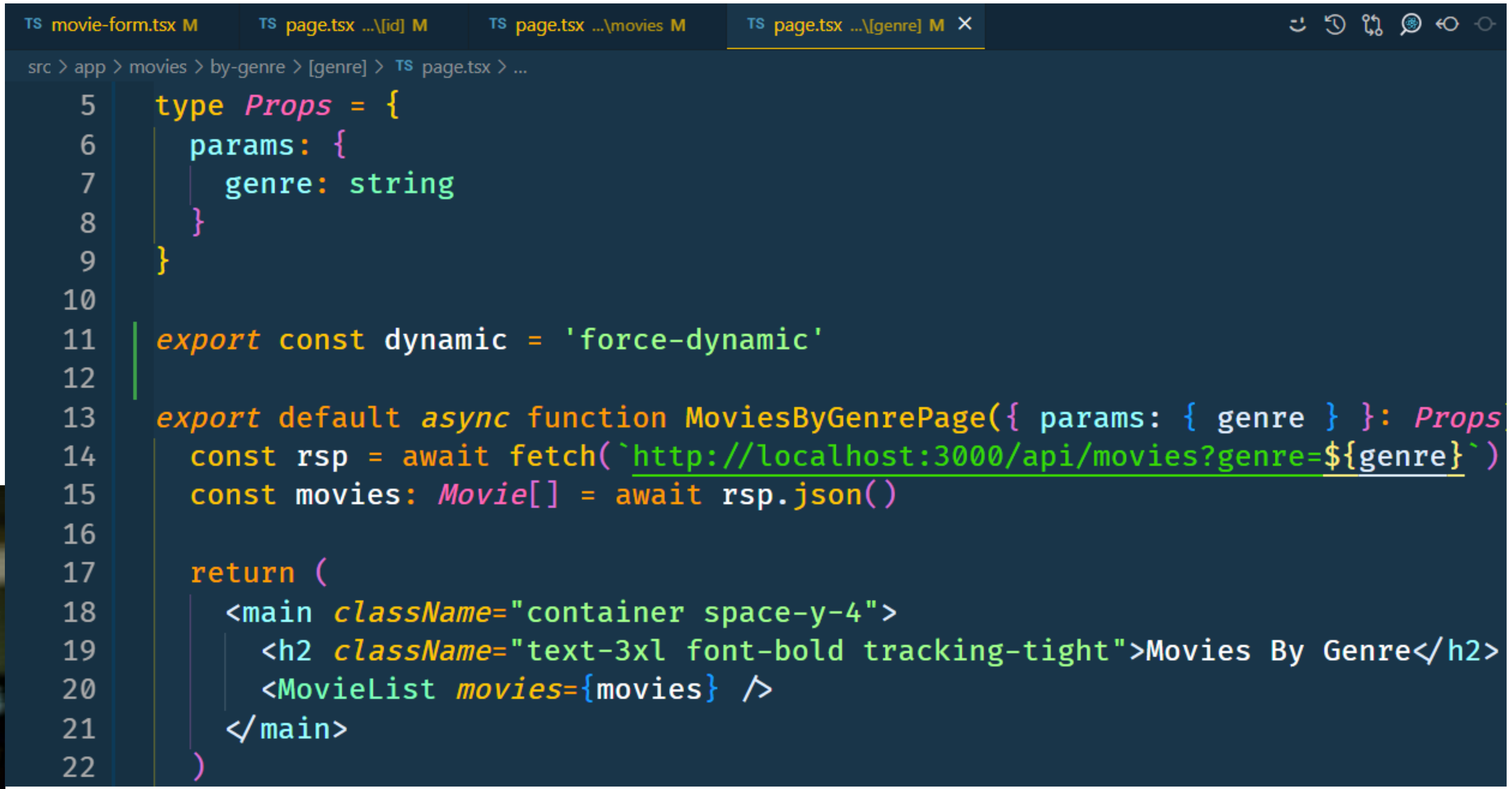
```tsx
export function MovieForm({ initialMovie }: Props) {
  const router = useRouter()
  const { toast } = useToast()

  const onSubmit = async (movie: Movie) => {
    try {
      await saveMovie(movie)
      router.refresh()

      toast({
        title: 'Success',
        description: 'Movie updated',
      })
    } catch (error) {
```

movies /page.tsx



```tsx
import { Movie } from '@prisma/client'

import { MovieList } from '@/components/movie-list'

export const dynamic = 'force-dynamic'

export default async function MoviesPage() {
  const rsp = await fetch('http://localhost:3000/api/movies')
  const movies: Movie[] = await rsp.json()

  return (
    <main className="container space-y-4">
      <h2 className="text-3xl font-bold tracking-tight">Top Rated Movies</h2>
      <MovieList movies={movies} />
    </main>
  )
}
```

movies/by-genre/[genre]/page.tsx



```tsx
type Props = {
  params: {
    genre: string
  }
}

export const dynamic = 'force-dynamic'

export default async function MoviesByGenrePage({ params: { genre } }: Props
  const rsp = await fetch(`http://localhost:3000/api/movies?genre=${genre}`)
  const movies: Movie[] = await rsp.json()

  return (
    <main className="container space-y-4">
      <h2 className="text-3xl font-bold tracking-tight">Movies By Genre</h2>
      <MovieList movies={movies} />
    </main>
  )
```

See you in the next video

Querying the database from a React Server Component

BOOST YOUR DEVELOPER POTENTIAL

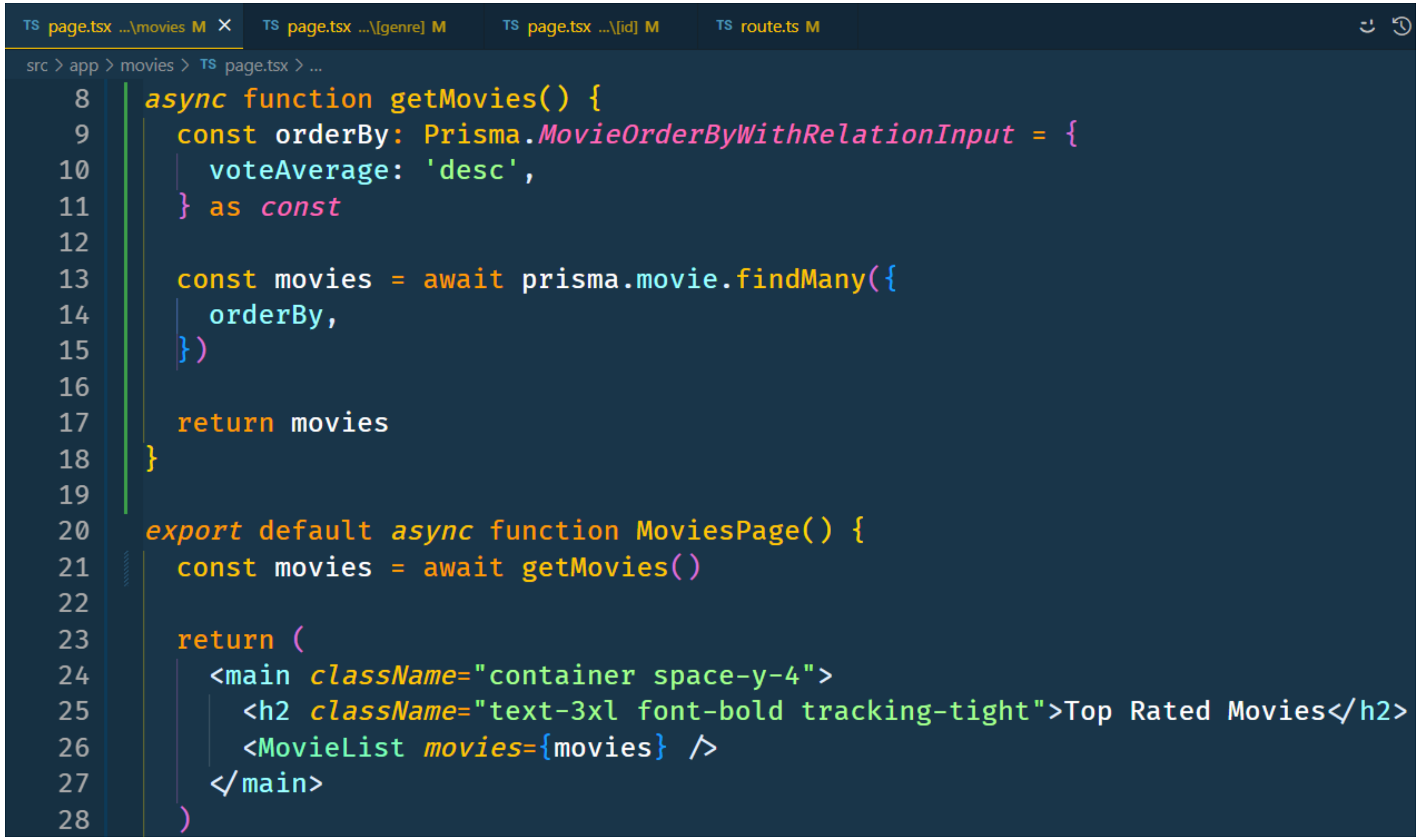
WITH REACT SERVER COMPONENTS

Querying the database from an RSC

Querying the database from an RSC

- **Using REST** to load the data results in **overhead**
  - Using the network to call back into the same application
  - Serializing and deserializing the data as JSON
- Because an **RSC** only runs on the server we **can use server side code**
  - Query the DB using Prisma directly
  - It's save to use secrets like database connection strings
- **Never executed in the browser**
  - Leads to smaller JavaScript bundle sizes

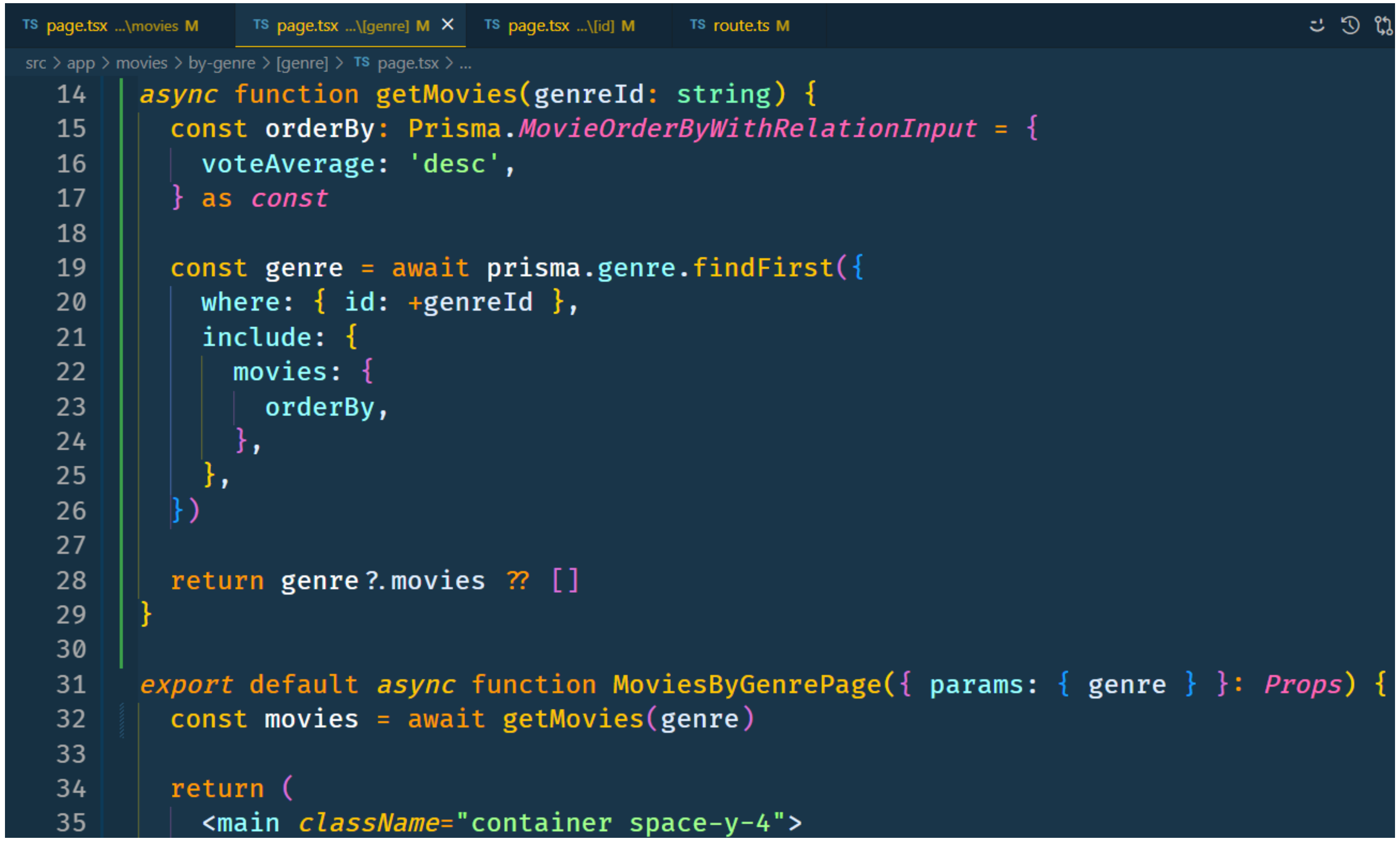
movies /page.tsx



TS page.tsx ...\movies M ×
TS page.tsx ...\[genre] M
TS page.tsx ...\[id] M
TS route.ts M
src > app > movies > TS page.tsx > ...

```tsx
async function getMovies() {
  const orderBy: Prisma.MovieOrderByWithRelationInput = {
    voteAverage: 'desc',
  } as const

  const movies = await prisma.movie.findMany({
    orderBy,
  })

  return movies
}

export default async function MoviesPage() {
  const movies = await getMovies()

  return (
    <main className="container space-y-4">
      <h2 className="text-3xl font-bold tracking-tight">Top Rated Movies</h2>
      <MovieList movies={movies} />
    </main>
  )
```

movies/by-genre /page.tsx

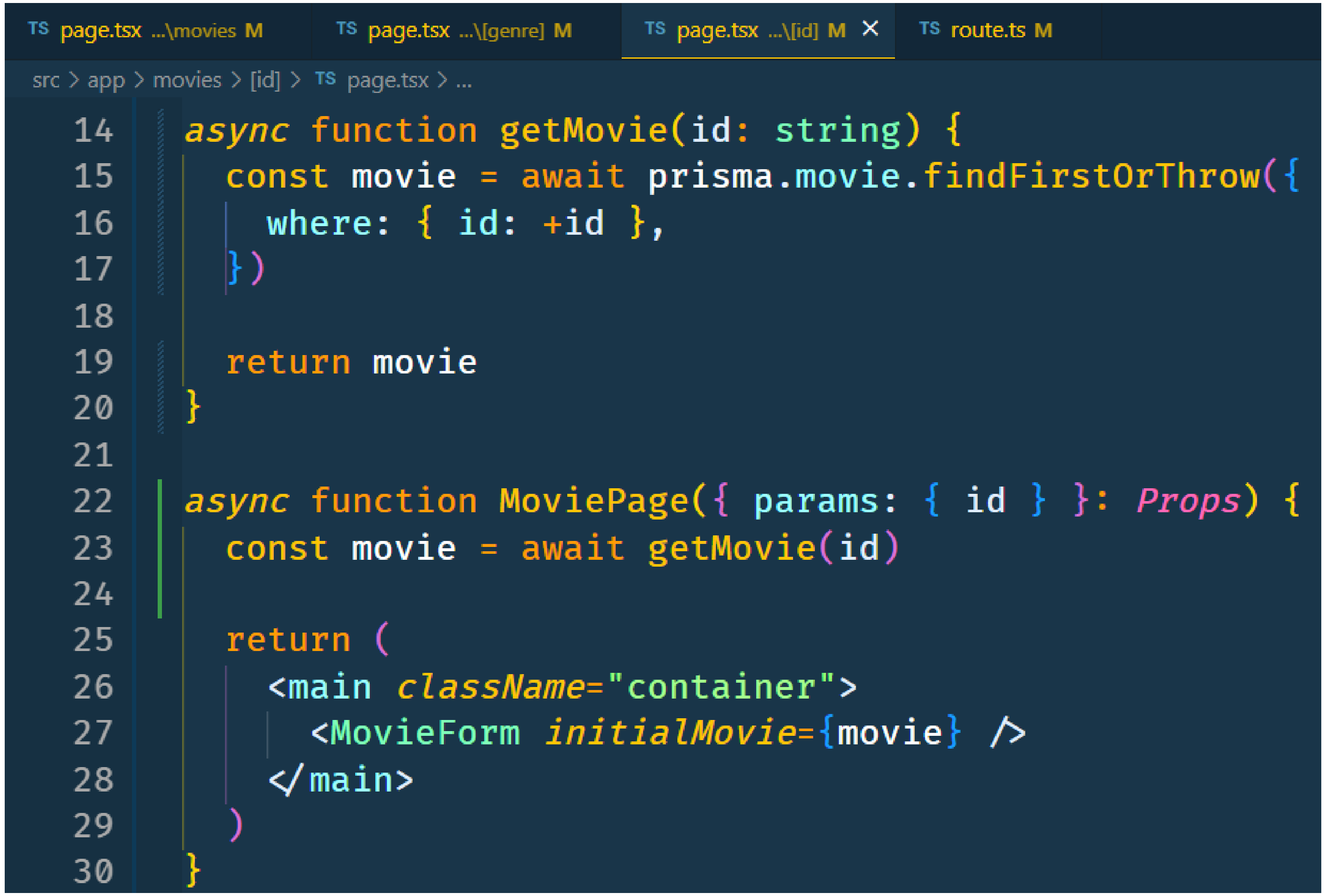


TS page.tsx ...\movies M | TS page.tsx ...\[genre] M × | TS page.tsx ...\[id] M | TS route.ts M

src > app > movies > by-genre > [genre] > TS page.tsx > ...

```tsx
async function getMovies(genreId: string) {
  const orderBy: Prisma.MovieOrderByWithRelationInput = {
    voteAverage: 'desc',
  } as const

  const genre = await prisma.genre.findFirst({
    where: { id: +genreId },
    include: {
      movies: {
        orderBy,
      },
    },
  })

  return genre?.movies ?? []
}

export default async function MoviesByGenrePage({ params: { genre } }: Props) {
  const movies = await getMovies(genre)

  return (
    <main className="container space-y-4">
```

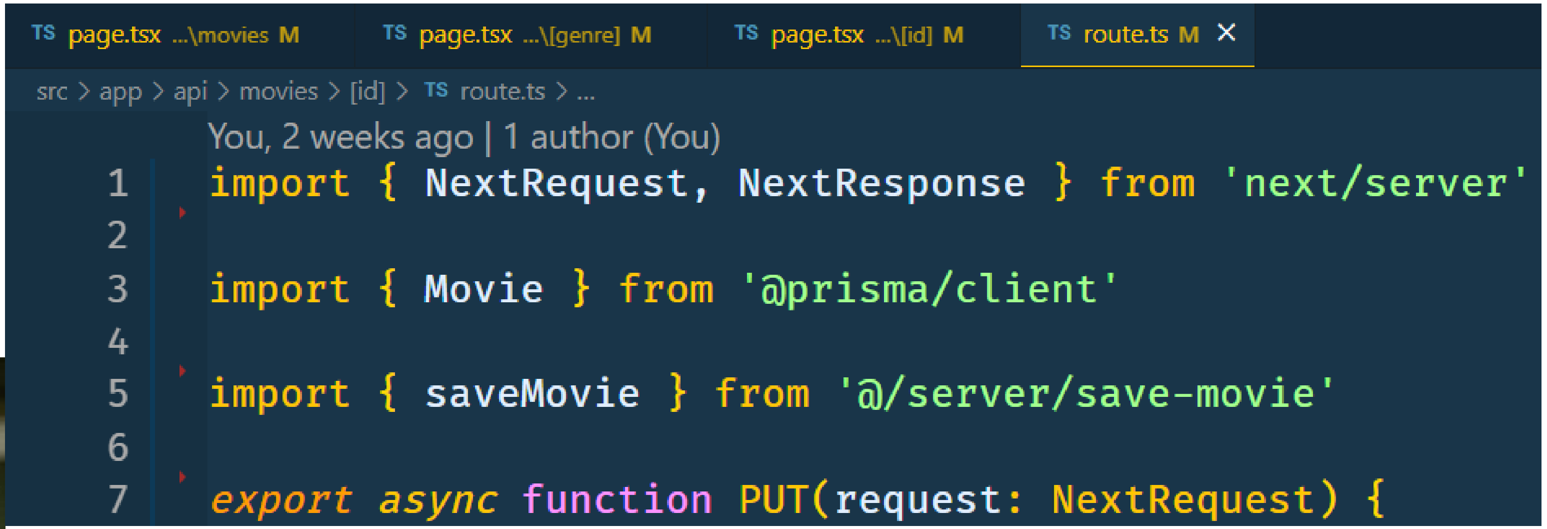
movies/[id] /page.tsx



TS page.tsx ...\movies M
TS page.tsx ...\[genre] M
TS page.tsx ...\[id] M
TS route.ts M
src > app > movies > [id] > TS page.tsx > ...

```tsx
async function getMovie(id: string) {
  const movie = await prisma.movie.findFirstOrThrow({
    where: { id: +id },
  })

  return movie
}

async function MoviePage({ params: { id } }: Props) {
  const movie = await getMovie(id)

  return (
    <main className="container">
      <MovieForm initialMovie={movie} />
    </main>
  )
}
```

api/movies/[id] /route.ts



TS page.tsx ...\movies M
TS page.tsx ...\[genre] M
TS page.tsx ...\[id] M
TS route.ts M ×
src > app > api > movies > [id] > TS route.ts > ...
You, 2 weeks ago | 1 author (You)

```
import { NextRequest, NextResponse } from 'next/server'

import { Movie } from '@prisma/client'

import { saveMovie } from '@/server/save-movie'

export async function PUT(request: NextRequest) {
```

See you in the next video

Prevent over fetching of data

BOOST YOUR DEVELOPER POTENTIAL

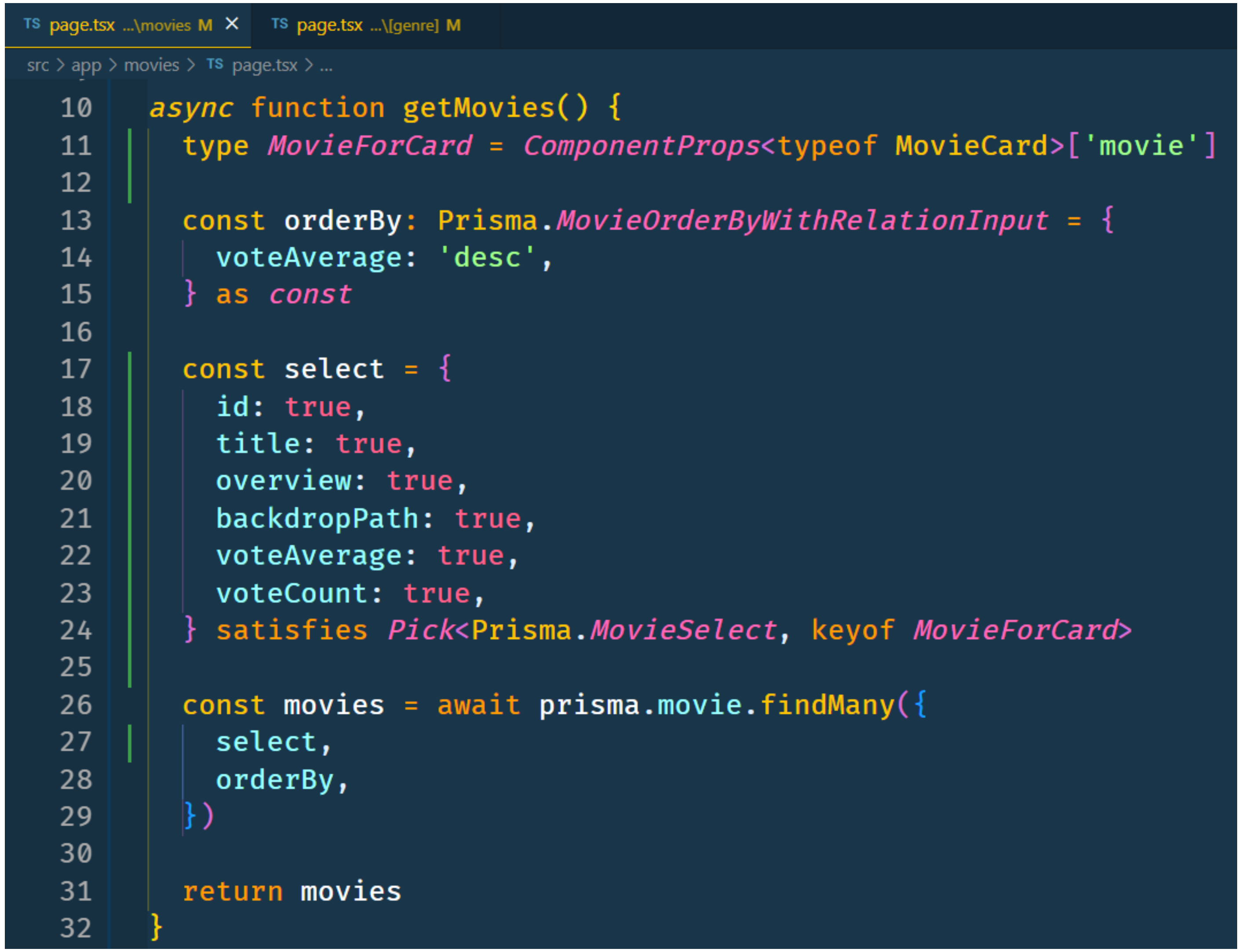
WITH REACT SERVER COMPONENTS

Prevent over fetching

Prevent over fetching

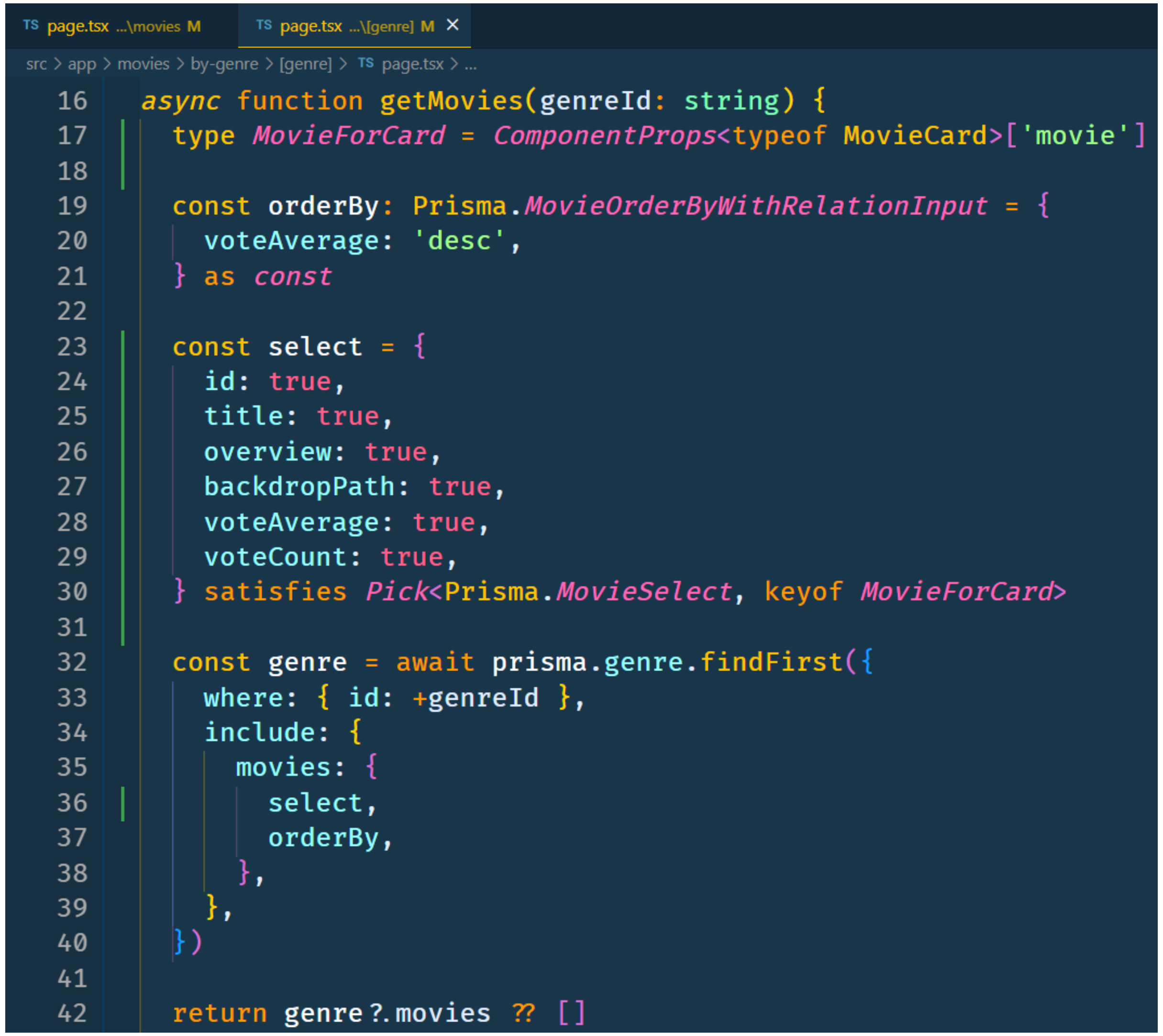
- **Colocation** of DB queries with components **enables more optimizations**
  - Fetch exactly the right amount of data
  - No more shared REST queries

movies /page.tsx



```tsx
async function getMovies() {
  type MovieForCard = ComponentProps<typeof MovieCard>['movie']

  const orderBy: Prisma.MovieOrderByWithRelationInput = {
    voteAverage: 'desc',
  } as const

  const select = {
    id: true,
    title: true,
    overview: true,
    backdropPath: true,
    voteAverage: true,
    voteCount: true,
  } satisfies Pick<Prisma.MovieSelect, keyof MovieForCard>

  const movies = await prisma.movie.findMany({
    select,
    orderBy,
  })

  return movies
}
```

movies/by-genre/page.tsx


TS page.tsx ...\movies M
TS page.tsx ...\[genre] M ×
src > app > movies > by-genre > [genre] > TS page.tsx > ...

```tsx
async function getMovies(genreId: string) {
  type MovieForCard = ComponentProps<typeof MovieCard>['movie']

  const orderBy: Prisma.MovieOrderByWithRelationInput = {
    voteAverage: 'desc',
  } as const

  const select = {
    id: true,
    title: true,
    overview: true,
    backdropPath: true,
    voteAverage: true,
    voteCount: true,
  } satisfies Pick<Prisma.MovieSelect, keyof MovieForCard>

  const genre = await prisma.genre.findFirst({
    where: { id: +genreId },
    include: {
      movies: {
        select,
        orderBy,
      },
    },
  })

  return genre?.movies ?? []
```

See you in the next video

Suspense and React Server Components

BOOST YOUR DEVELOPER POTENTIAL

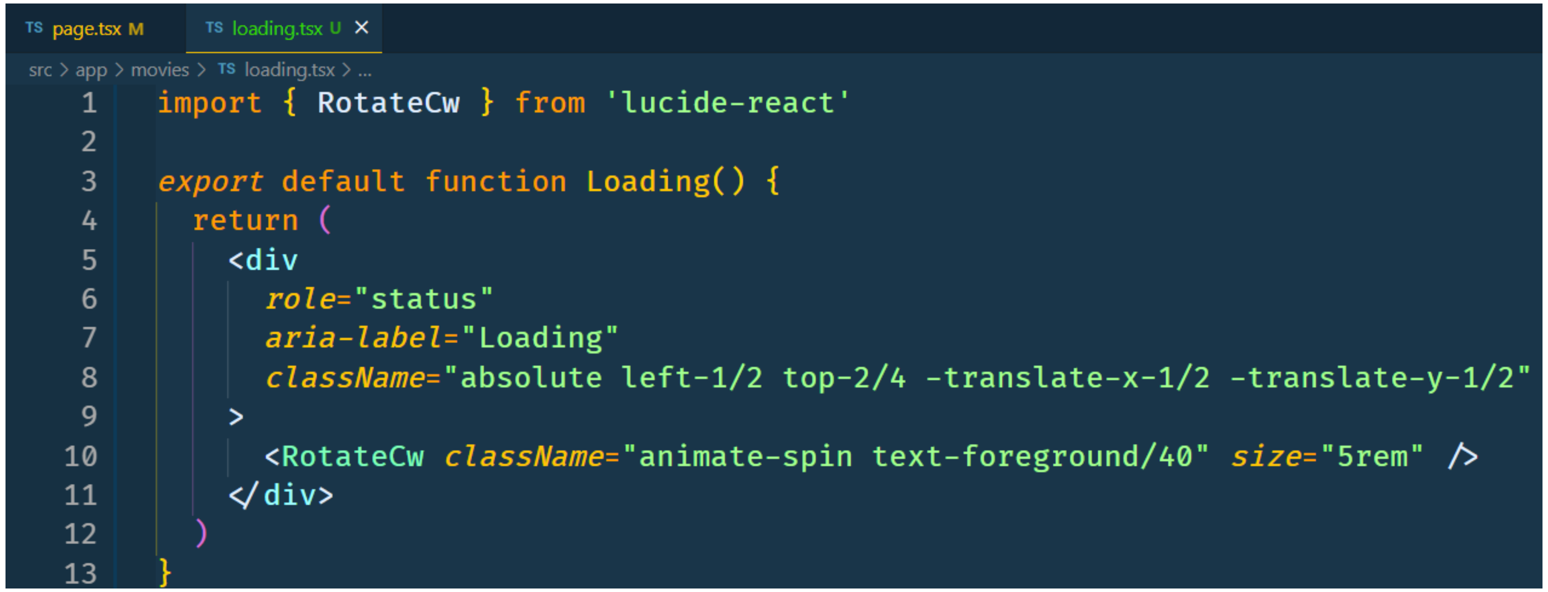
WITH REACT SERVER COMPONENTS

Suspense & RSC pages

Suspense & RSC pages

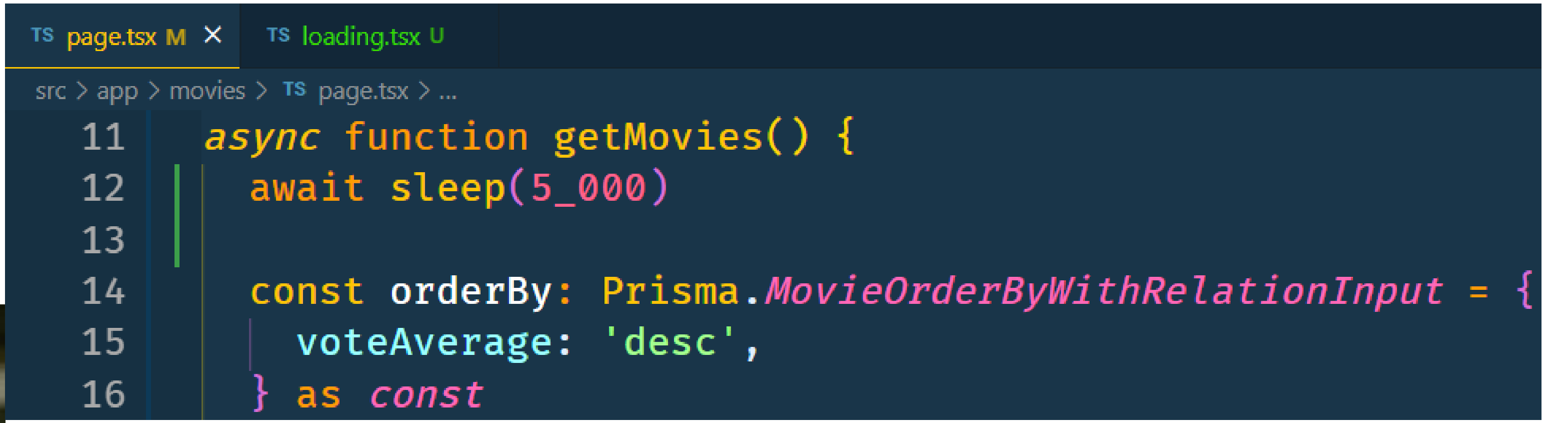
- React Server Components are **suspended until they resolve**
    - Can be controlled with **<Suspense /> boundaries**
- Next.js makes it easy to **suspend when rendering an async page**
    - **Add a loading.tsx**
    - They can be nested and the closest loading component will be used

movies /loading.tsx



```tsx
import { RotateCw } from 'lucide-react'

export default function Loading() {
  return (
    <div
      role="status"
      aria-label="Loading"
      className="absolute left-1/2 top-2/4 -translate-x-1/2 -translate-y-1/2"
    >
      <RotateCw className="animate-spin text-foreground/40" size="5rem" />
    </div>
  )
}
```

movies /page.tsx



```
async function getMovies() {
  await sleep(5_000)

  const orderBy: Prisma.MovieOrderByWithRelationInput = {
    voteAverage: 'desc',
  } as const
```

See you in the next video

React Server Components and streaming

BOOST YOUR DEVELOPER POTENTIAL

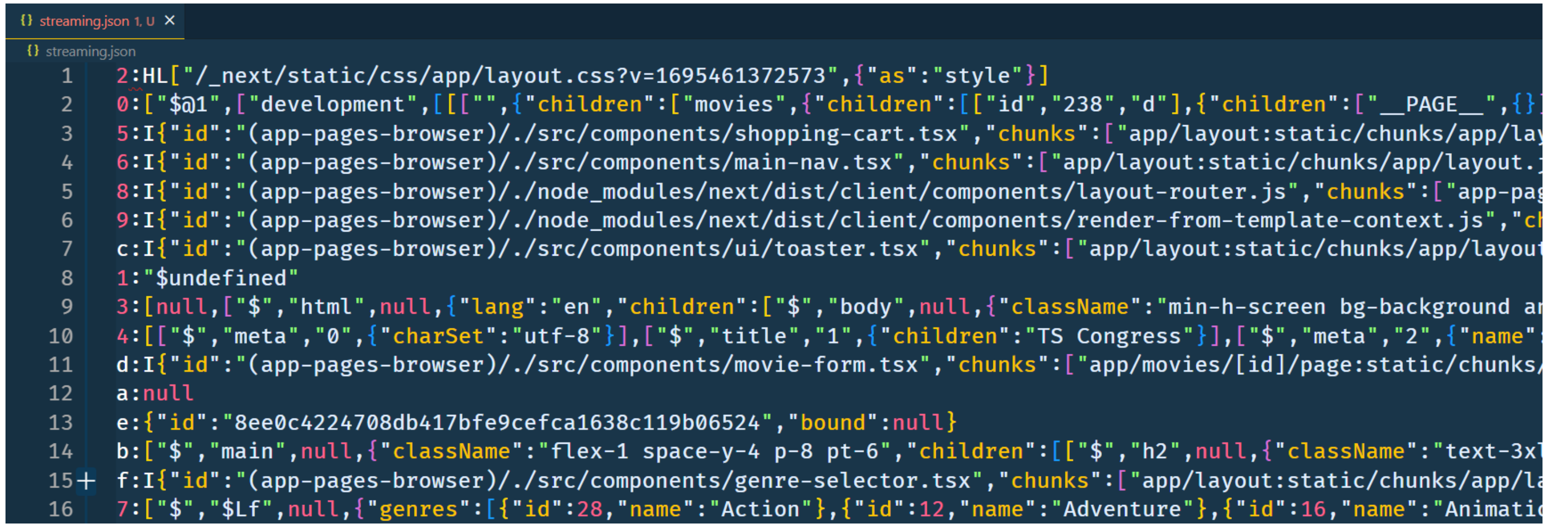
WITH REACT SERVER COMPONENTS

RSC and streaming

RSC and streaming

- **Async React Server Components are streamed to the browser**
  - Using the React Server Component Payload
  - On the client they are suspended until the component resolves
- **Server action responses** can also stream components back
  - After a ***revalidatePath()*** or a ***revalidateTag()***

RSC Payload



```
2:HL["/_next/static/css/app/layout.css?v=1695461372573",{"as":"style"}]
0:["$@1",["development",[[["",{"children":["movies",{"children":[["id","238","d"],{"children":["__PAGE__",{}
5:I{"id":"(app-pages-browser)/./src/components/shopping-cart.tsx","chunks":["app/layout:static/chunks/app/lay
6:I{"id":"(app-pages-browser)/./src/components/main-nav.tsx","chunks":["app/layout:static/chunks/app/layout.
8:I{"id":"(app-pages-browser)/./node_modules/next/dist/client/components/layout-router.js","chunks":["app-pag
9:I{"id":"(app-pages-browser)/./node_modules/next/dist/client/components/render-from-template-context.js","c
c:I{"id":"(app-pages-browser)/./src/components/ui/toaster.tsx","chunks":["app/layout:static/chunks/app/layou
1:"$undefined"
3:[null,["$","html",null,{"lang":"en","children":["$","body",null,{"className":"min-h-screen bg-background a
4:[["$","meta","0",{"charSet":"utf-8"}],["$","title","1",{"children":"TS Congress"}],["$","meta","2",{"name"
d:I{"id":"(app-pages-browser)/./src/components/movie-form.tsx","chunks":["app/movies/[id]/page:static/chunks/
a:null
e:{"id":"8ee0c4224708db417bfe9cefca1638c119b06524","bound":null}
b:["$","main",null,{"className":"flex-1 space-y-4 p-8 pt-6","children":[["$","h2",null,{"className":"text-3x
f:I{"id":"(app-pages-browser)/./src/components/genre-selector.tsx","chunks":["app/layout:static/chunks/app/la
7:["$","$Lf",null,{"genres":[{"id":28,"name":"Action"},{"id":12,"name":"Adventure"},{"id":16,"name":"Animati
```

See you in the next video

What is a server component?

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

What is a server component?

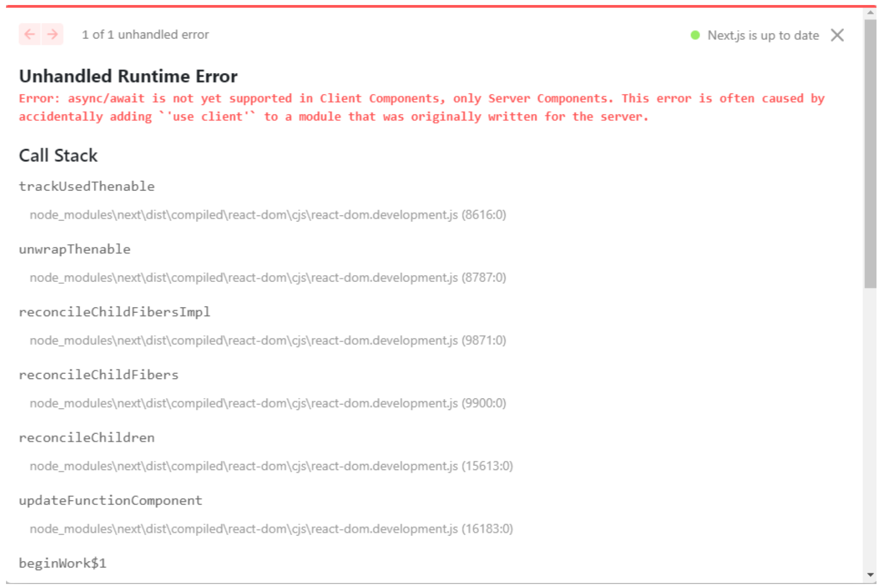
Afbeelding van Tumisu via Pixabay

What is a server component?

- What is a server component and what is not?
  - **Client components are marked with 'use client'**
- But **not all other components are server components**
  - With a component without 'use client' it depends on their parents
- If a component is a client component
  - Then **all components it renders are also client components**
- ☞ There is ***no 'use server'*** for server components ☜
  - The ***'use server'*** directive exists but is used for Server Actions
  - But there is a ***server-only*** NPM package

Async Client Components

- **Client components can't be asynchronous** yet
  - But the error doesn't reliably show up



1 of 1 unhandled error

Next.js is up to date

Unhandled Runtime Error

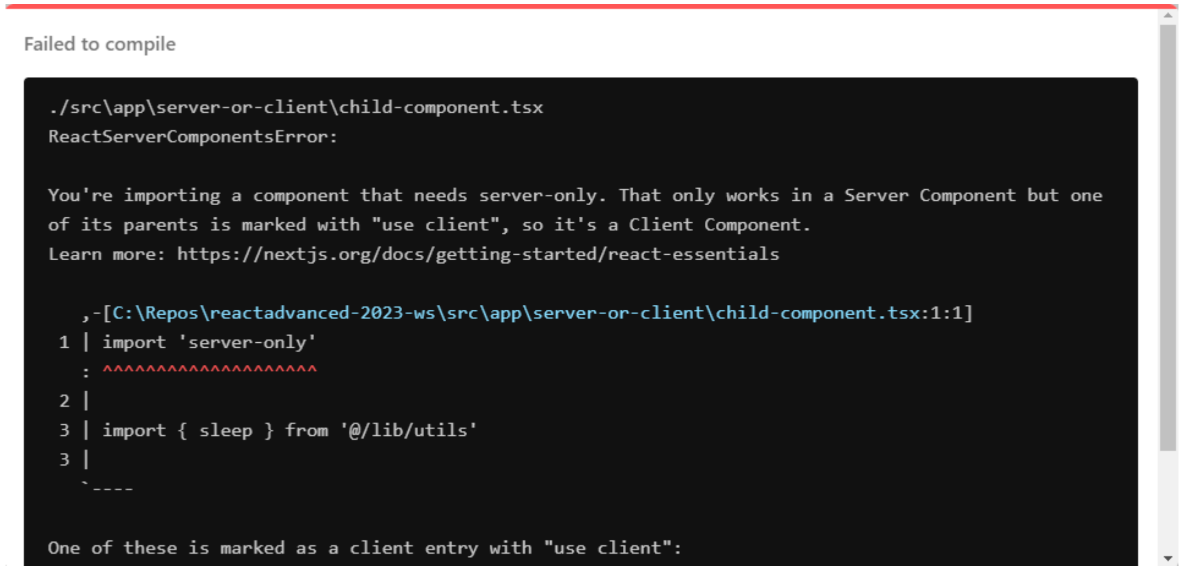
```
Error: async/await is not yet supported in Client Components, only Server Components. This error is often caused by
accidentally adding `'use client'` to a module that was originally written for the server.
```

Call Stack

```
trackUsedThenable
  node_modules\next\dist\compiled\react-dom\cjs\react-dom.development.js (8616:0)
unwrapThenable
  node_modules\next\dist\compiled\react-dom\cjs\react-dom.development.js (8787:0)
reconcileChildFibersImpl
  node_modules\next\dist\compiled\react-dom\cjs\react-dom.development.js (9871:0)
reconcileChildFibers
  node_modules\next\dist\compiled\react-dom\cjs\react-dom.development.js (9900:0)
reconcileChildren
  node_modules\next\dist\compiled\react-dom\cjs\react-dom.development.js (15613:0)
updateFunctionComponent
  node_modules\next\dist\compiled\react-dom\cjs\react-dom.development.js (16183:0)
beginWork$1
```

server-only

- Import the **server-only** NPM package
  - With components that must run on the server



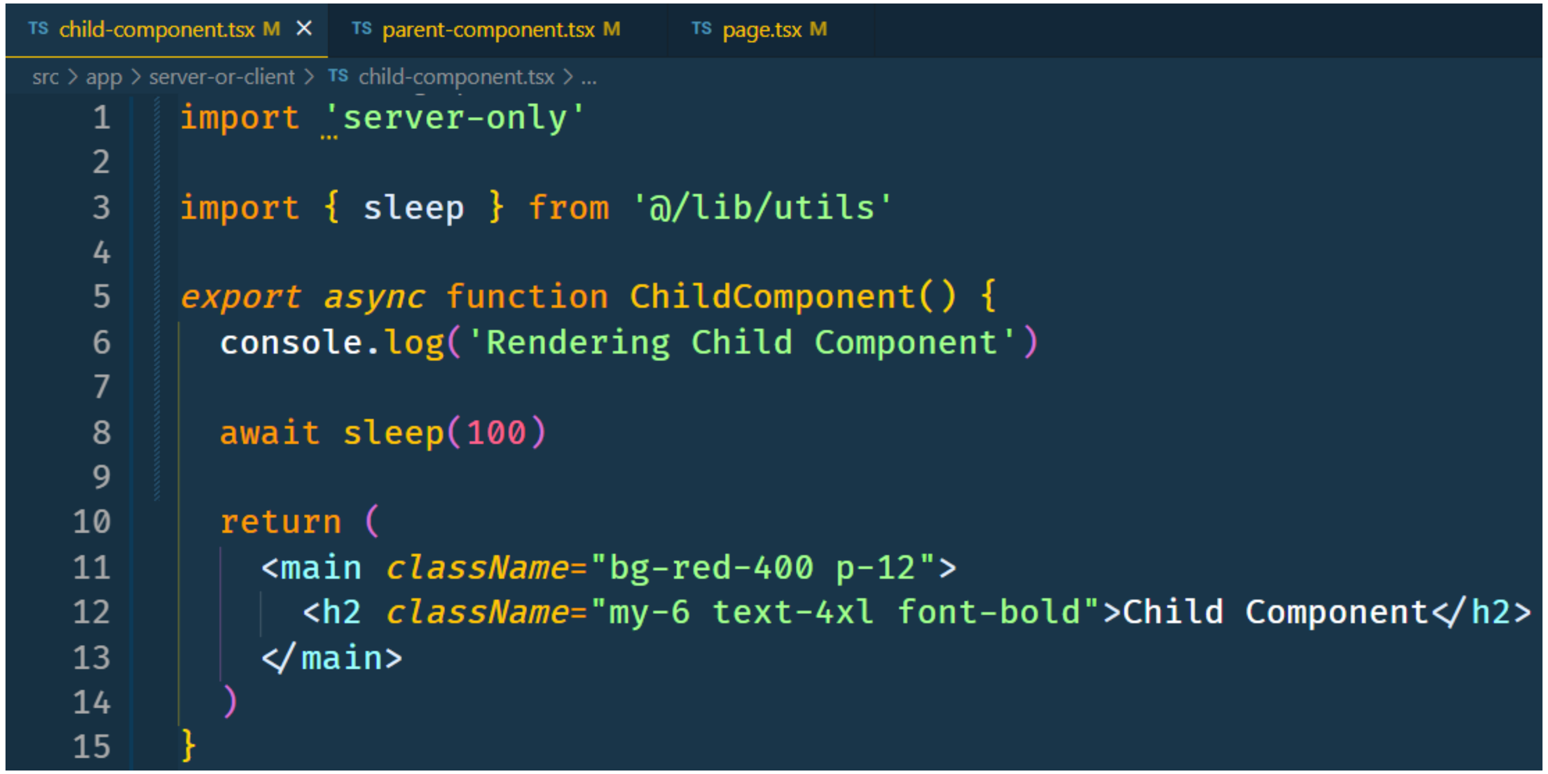
Failed to compile

```
./src\app\server-or-client\child-component.tsx
ReactServerComponentsError:

You're importing a component that needs server-only. That only works in a Server Component but one
of its parents is marked with "use client", so it's a Client Component.
Learn more: https://nextjs.org/docs/getting-started/react-essentials

   ,-[C:\Repos\reactadvanced-2023-ws\src\app\server-or-client\child-component.tsx:1:1]
 1 | import 'server-only'
   : ^^^^^^^^^^^^^^^^^^^^
 2 |
 3 | import { sleep } from '@/lib/utils'
 3 |
   `----

One of these is marked as a client entry with "use client":
```

Using an RSC as a child of a client component

- **A client component can have a server component as a child**
  - As long as it doesn't render it
- **Render the child server component** from another server component
  - 💡 And pass it as a children prop into the client component 💡

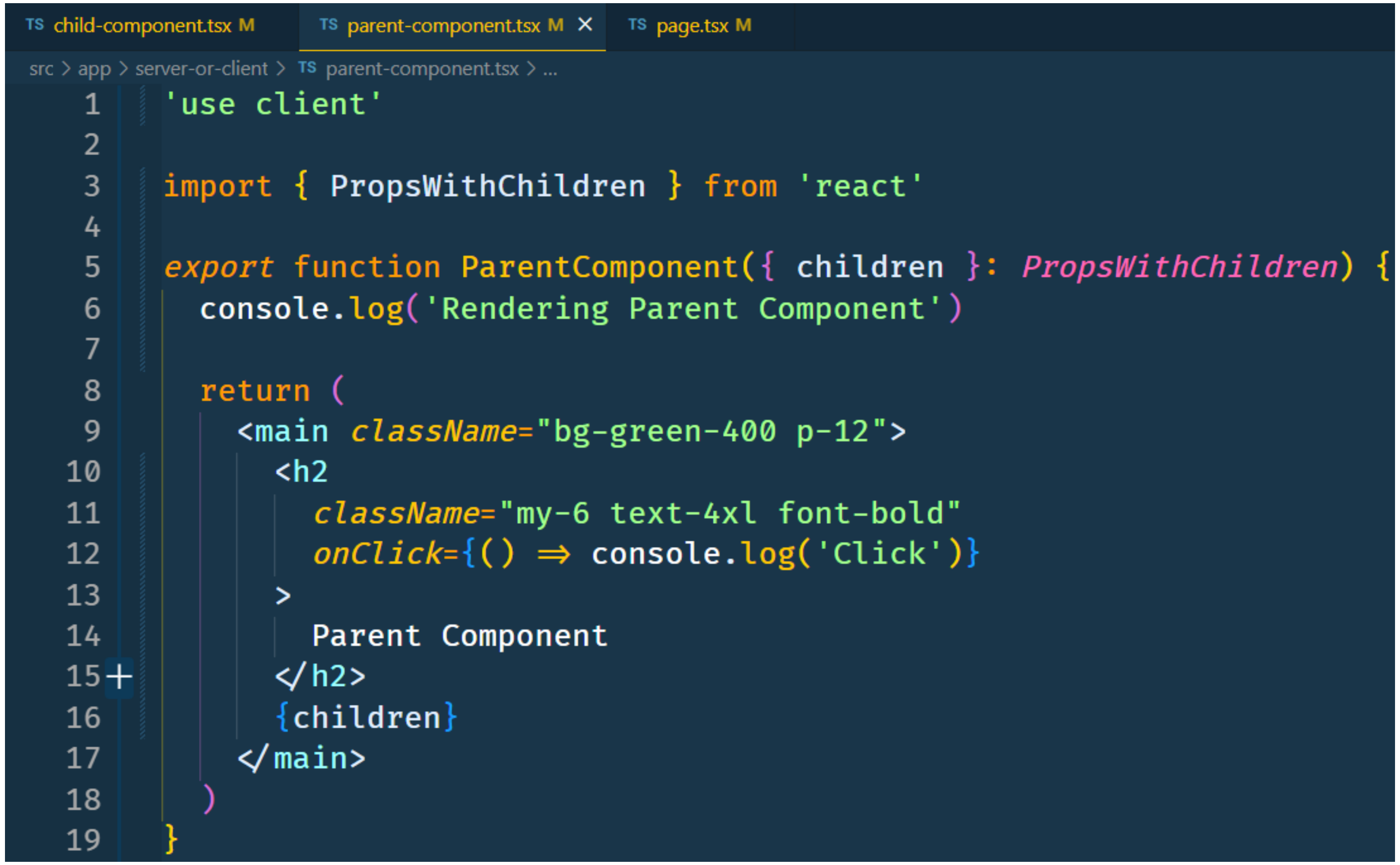
child-component.tsx



TS child-component.tsx M ×
TS parent-component.tsx M
TS page.tsx M
src > app > server-or-client > TS child-component.tsx > ...

```tsx
import 'server-only'

import { sleep } from '@/lib/utils'

export async function ChildComponent() {
  console.log('Rendering Child Component')

  await sleep(100)

  return (
    <main className="bg-red-400 p-12">
      <h2 className="my-6 text-4xl font-bold">Child Component</h2>
    </main>
  )
}
```

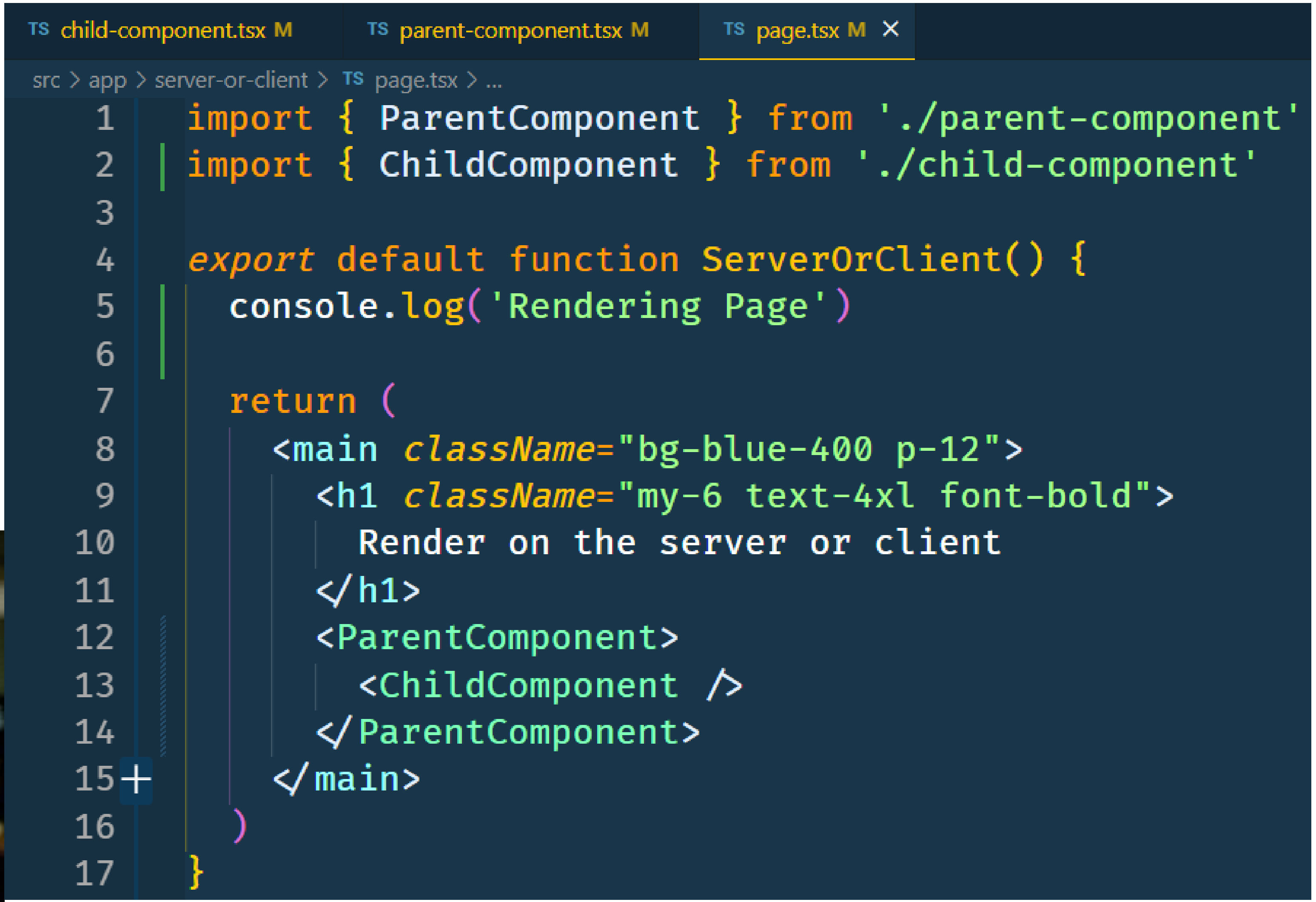
parent-component.tsx



TS child-component.tsx M
TS parent-component.tsx M
TS page.tsx M
src > app > server-or-client > TS parent-component.tsx > ...

```tsx
'use client'

import { PropsWithChildren } from 'react'

export function ParentComponent({ children }: PropsWithChildren) {
  console.log('Rendering Parent Component')

  return (
    <main className="bg-green-400 p-12">
      <h2
        className="my-6 text-4xl font-bold"
        onClick={() => console.log('Click')}
      >
        Parent Component
      </h2>
      {children}
    </main>
  )
}
```

server-or-client /page.tsx


TS child-component.tsx M
TS parent-component.tsx M
TS page.tsx M ×
src > app > server-or-client > TS page.tsx > ...

```tsx
import { ParentComponent } from './parent-component'
import { ChildComponent } from './child-component'

export default function ServerOrClient() {
  console.log('Rendering Page')

  return (
    <main className="bg-blue-400 p-12">
      <h1 className="my-6 text-4xl font-bold">
        Render on the server or client
      </h1>
      <ParentComponent>
        <ChildComponent />
      </ParentComponent>
    </main>
  )
}
```

See you in the next video

Loading the genres in the menu on the server

BOOST YOUR DEVELOPER POTENTIAL

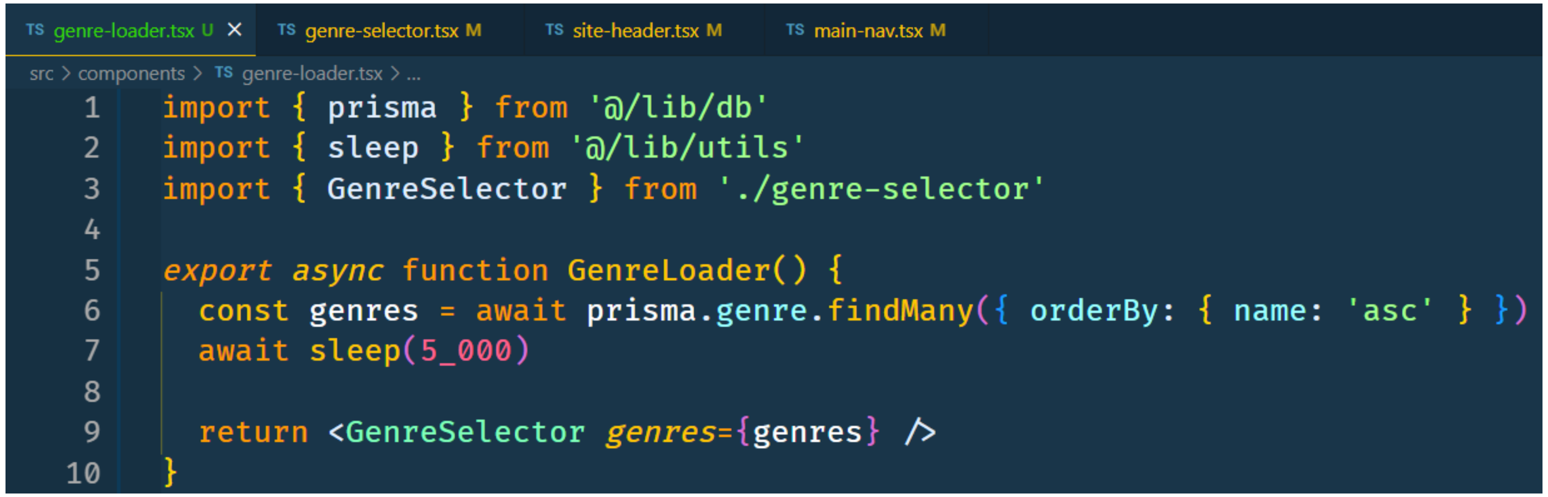
WITH REACT SERVER COMPONENTS

Loading the genres in the menu on the server

Loading the genres on the server

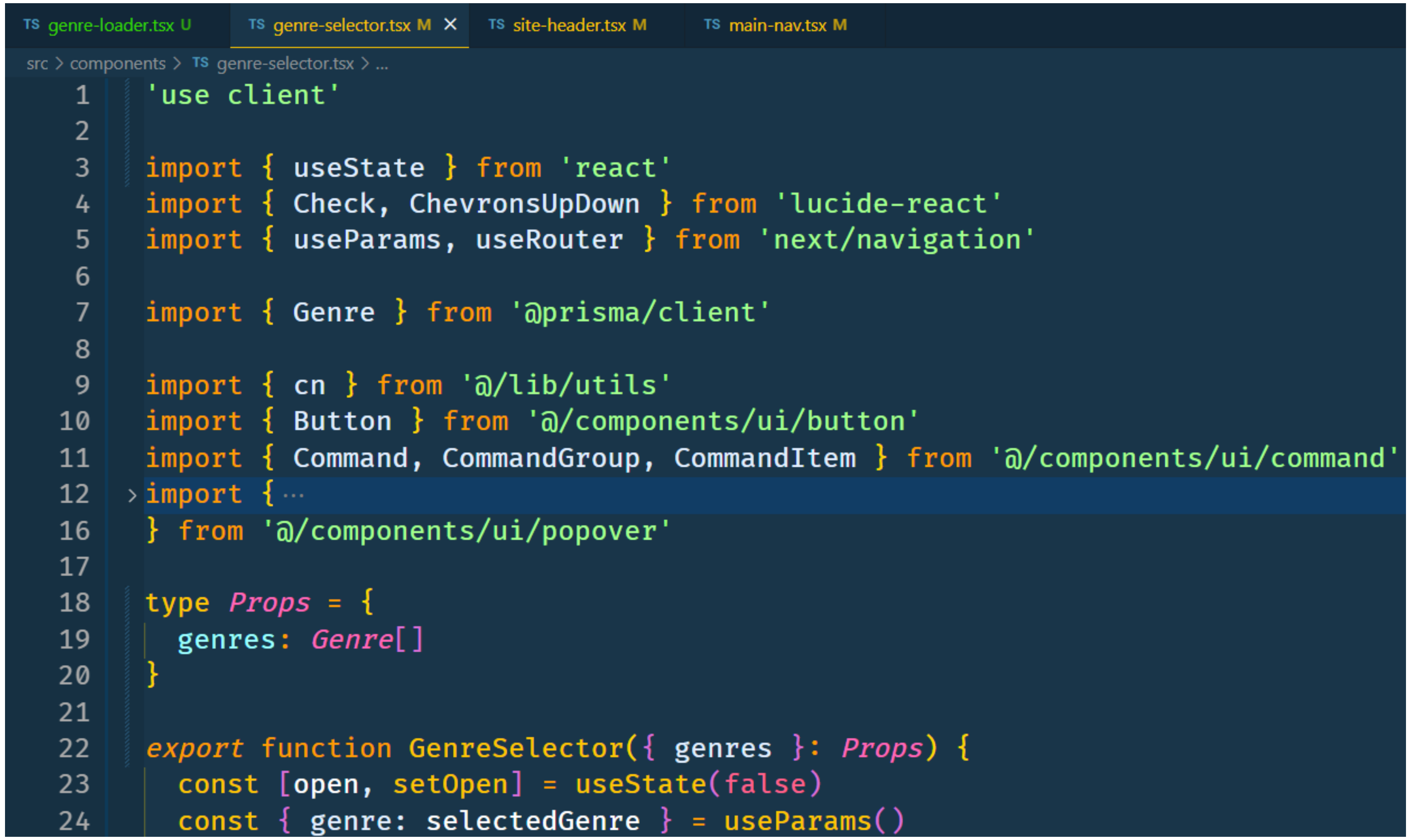
- **The <GenreSelector/> component can’t be rendered on the server**
  - The parent component <MainNav/> is a client component
- The <SiteHeader /> is a server component
  - It can access the database and load the genres

genre-loader.tsx



```tsx
import { prisma } from '@/lib/db'
import { sleep } from '@/lib/utils'
import { GenreSelector } from './genre-selector'

export async function GenreLoader() {
  const genres = await prisma.genre.findMany({ orderBy: { name: 'asc' } })
  await sleep(5_000)

  return <GenreSelector genres={genres} />
}
```

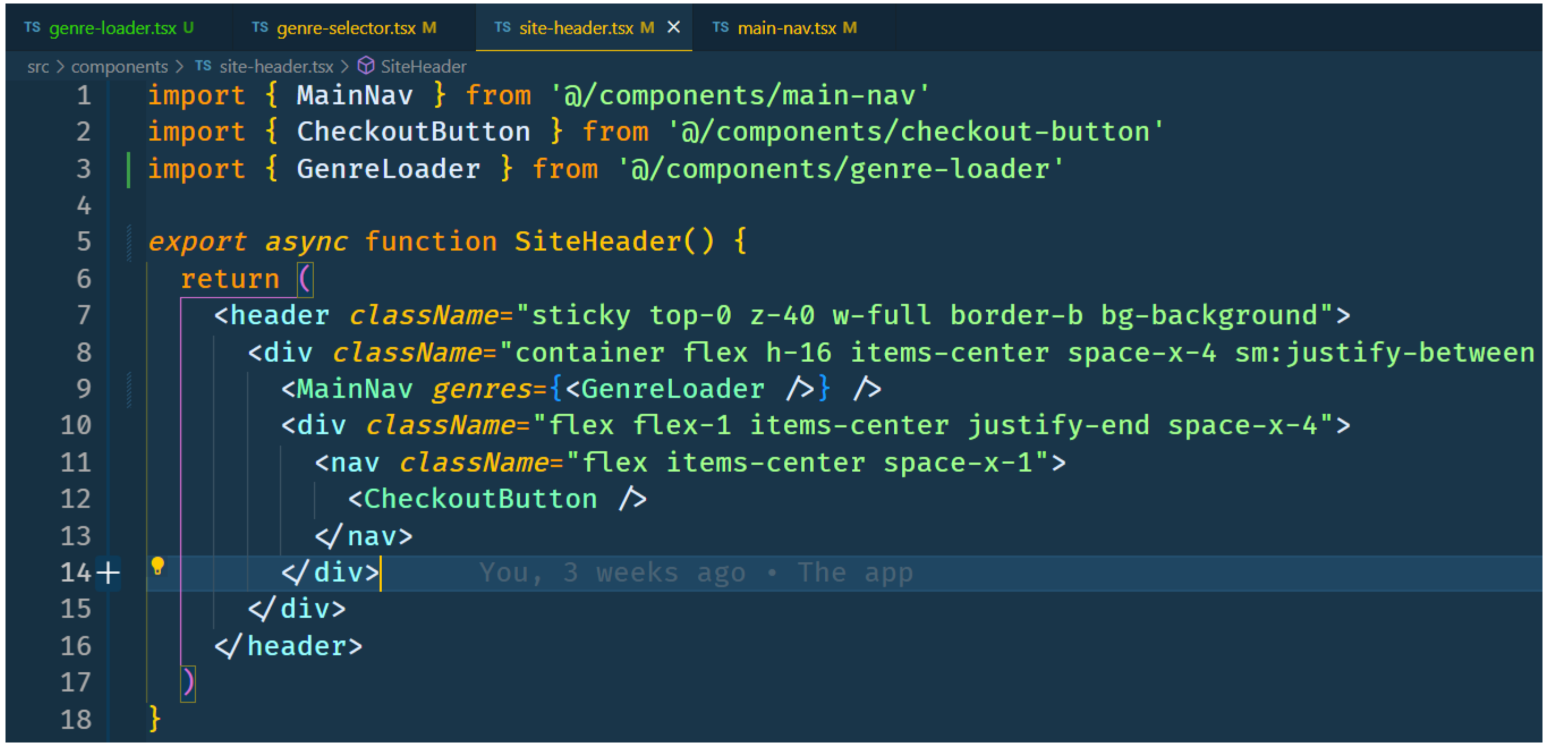
genre-selector.tsx



TS genre-loader.tsx U
TS genre-selector.tsx M ✕
TS site-header.tsx M
TS main-nav.tsx M
src > components > TS genre-selector.tsx > ...

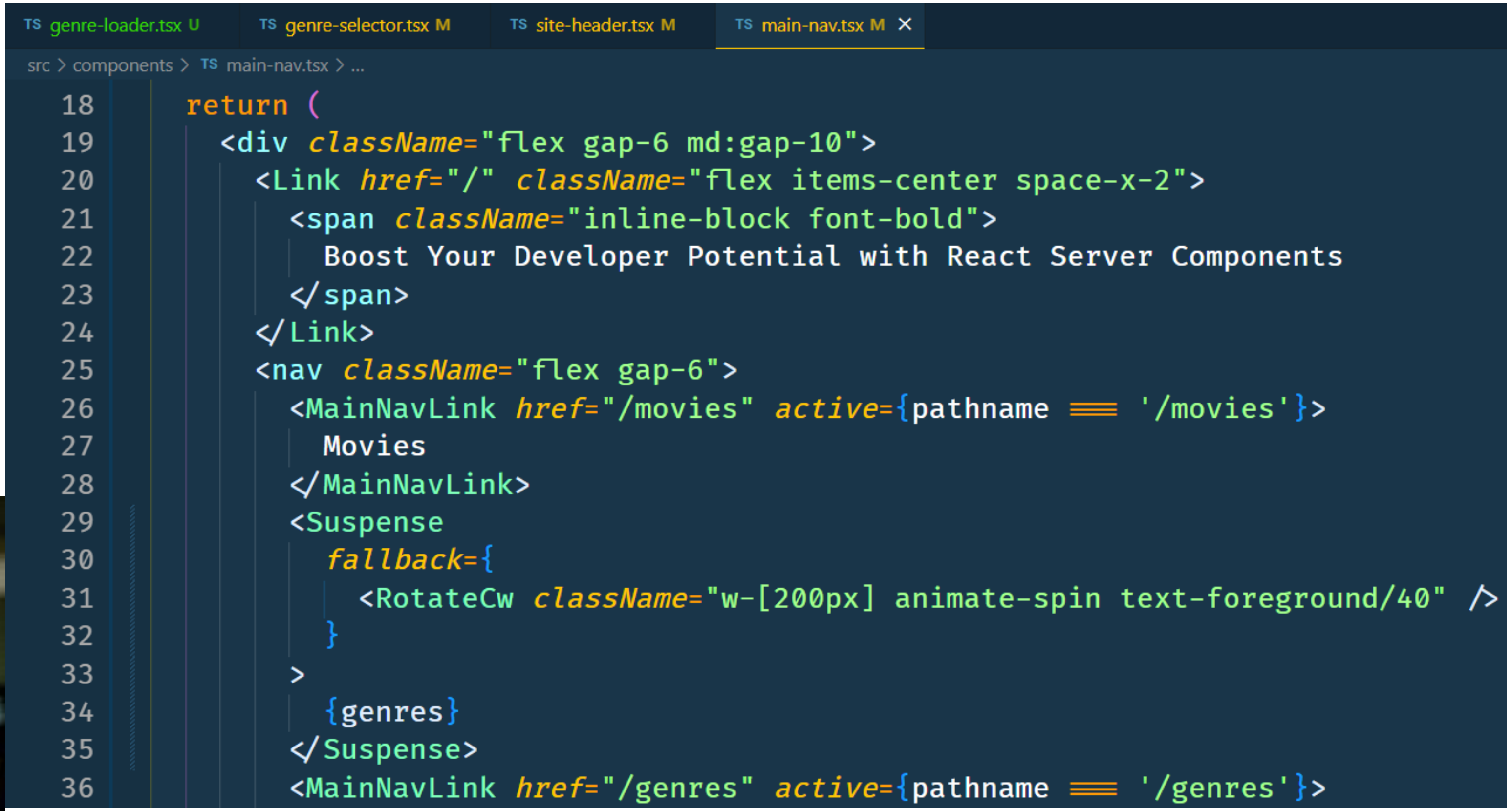
```tsx
'use client'

import { useState } from 'react'
import { Check, ChevronsUpDown } from 'lucide-react'
import { useParams, useRouter } from 'next/navigation'

import { Genre } from '@prisma/client'

import { cn } from '@/lib/utils'
import { Button } from '@/components/ui/button'
import { Command, CommandGroup, CommandItem } from '@/components/ui/command'
import {…
} from '@/components/ui/popover'

type Props = {
  genres: Genre[]
}

export function GenreSelector({ genres }: Props) {
  const [open, setOpen] = useState(false)
  const { genre: selectedGenre } = useParams()
```

site-header.tsx



```tsx
import { MainNav } from '@/components/main-nav'
import { CheckoutButton } from '@/components/checkout-button'
import { GenreLoader } from '@/components/genre-loader'

export async function SiteHeader() {
  return (
    <header className="sticky top-0 z-40 w-full border-b bg-background">
      <div className="container flex h-16 items-center space-x-4 sm:justify-between
        <MainNav genres={<GenreLoader />} />
        <div className="flex flex-1 items-center justify-end space-x-4">
          <nav className="flex items-center space-x-1">
            <CheckoutButton />
          </nav>
        </div>      You, 3 weeks ago • The app
      </div>
    </header>
  )
}
```

main-nav.tsx



```
  return (
    <div className="flex gap-6 md:gap-10">
      <Link href="/" className="flex items-center space-x-2">
        <span className="inline-block font-bold">
          Boost Your Developer Potential with React Server Components
        </span>
      </Link>
      <nav className="flex gap-6">
        <MainNavLink href="/movies" active={pathname === '/movies'}>
          Movies
        </MainNavLink>
        <Suspense
          fallback={
            <RotateCw className="w-[200px] animate-spin text-foreground/40" />
          }
        >
          {genres}
        </Suspense>
        <MainNavLink href="/genres" active={pathname === '/genres'}>
```

See you in the next video

BOOST YOUR DEVELOPER POTENTIAL

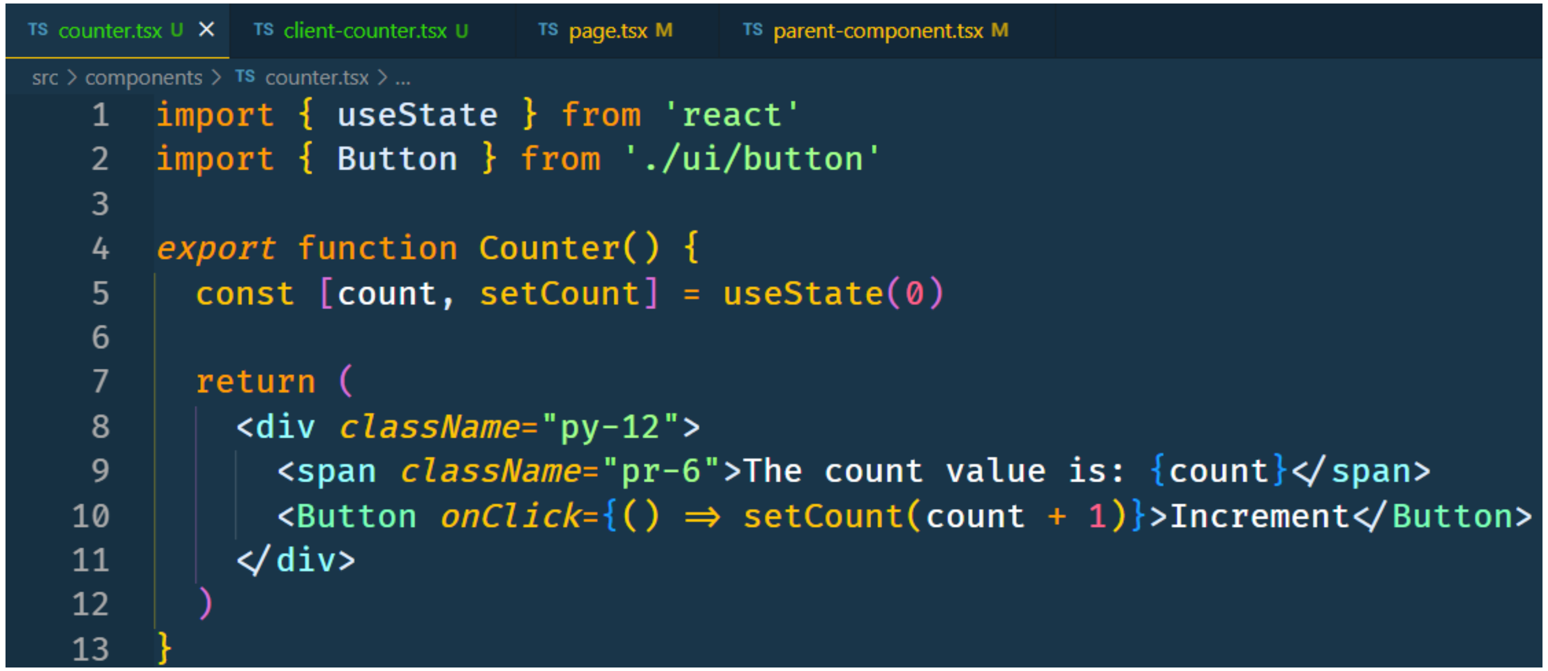
WITH REACT SERVER COMPONENTS

External Dependencies

External Dependencies

- **Not every external component includes `use client`** where required
  - Making them hard to use from a React Server Component
- Create a simple **wrapper file with `use client`**
  - And re-export the same component
- **This problem will go away over time**
  - When adding ‘use client` becomes the standard
  - Please create pull requests for open source NPM packages

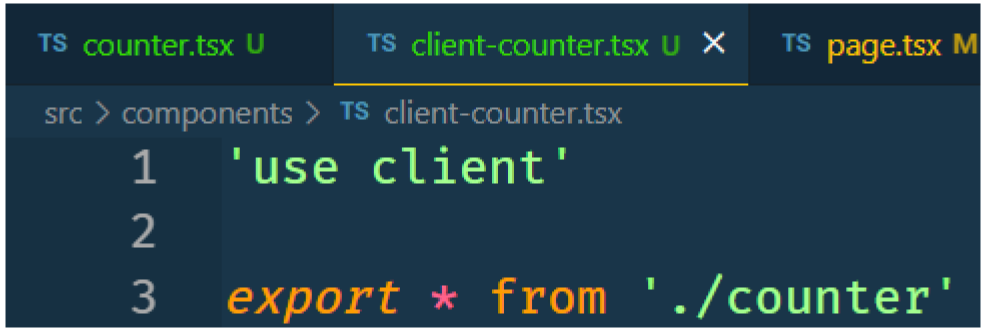
counter.tsx



TS counter.tsx U ×
TS client-counter.tsx U
TS page.tsx M
TS parent-component.tsx M
src > components > TS counter.tsx > ...

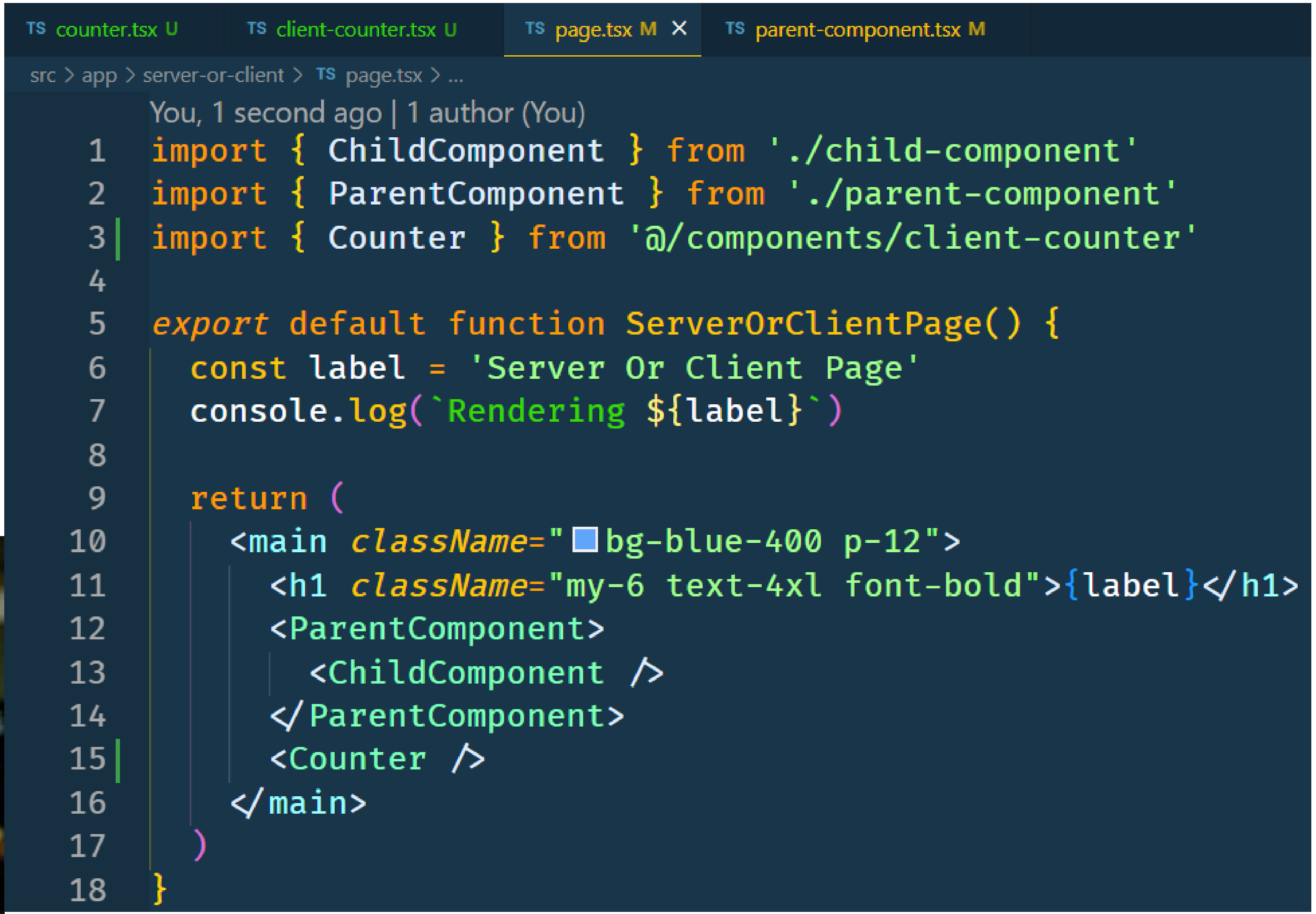
```tsx
import { useState } from 'react'
import { Button } from './ui/button'

export function Counter() {
  const [count, setCount] = useState(0)

  return (
    <div className="py-12">
      <span className="pr-6">The count value is: {count}</span>
      <Button onClick={() => setCount(count + 1)}>Increment</Button>
    </div>
  )
}
```

client-counter.tsx



```tsx
'use client'

export * from './counter'
```

page.tsx


TS counter.tsx U
TS client-counter.tsx U
TS page.tsx M
TS parent-component.tsx M
src > app > server-or-client > TS page.tsx > ...
You, 1 second ago | 1 author (You)

```tsx
import { ChildComponent } from './child-component'
import { ParentComponent } from './parent-component'
import { Counter } from '@/components/client-counter'

export default function ServerOrClientPage() {
  const label = 'Server Or Client Page'
  console.log(`Rendering ${label}`)

  return (
    <main className="bg-blue-400 p-12">
      <h1 className="my-6 text-4xl font-bold">{label}</h1>
      <ParentComponent>
        <ChildComponent />
      </ParentComponent>
      <Counter />
    </main>
  )
}
```

See you in the next video

BOOST YOUR DEVELOPER POTENTIAL

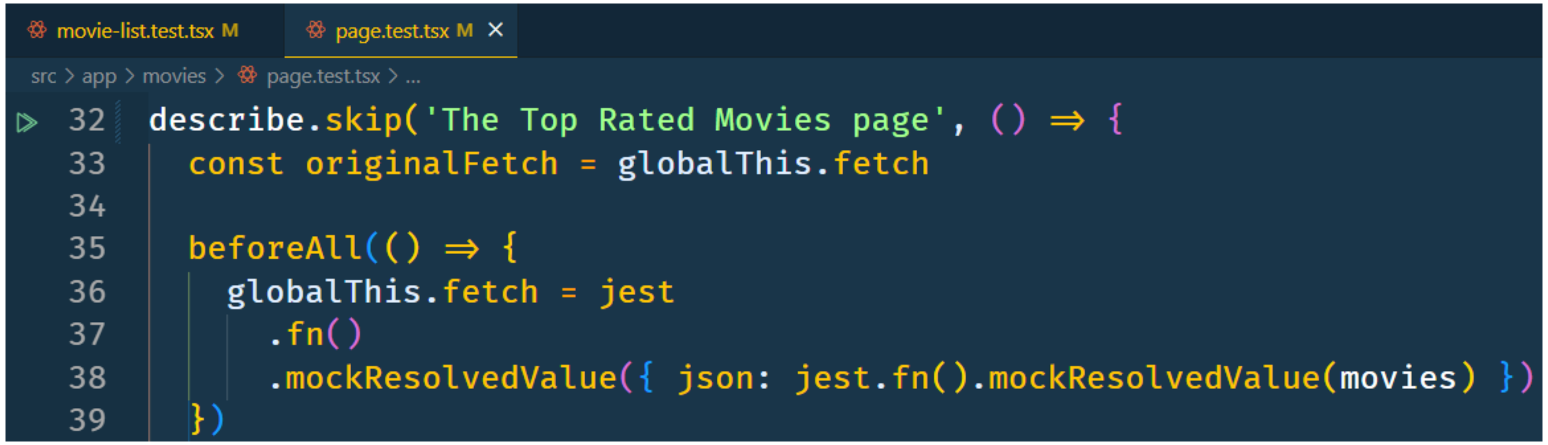
WITH REACT SERVER COMPONENTS

Unit Testing

Unit Testing

- **Unit testing** of async React Server Components **is still tricky**
  - There is no good support from React Testing Library
- **Consider using end to end testing** for async components
  - Tools like Cypress or Playwright work well
- Unit testing **can be done with a few hacks** now
  - Stay tuned for then this becomes better

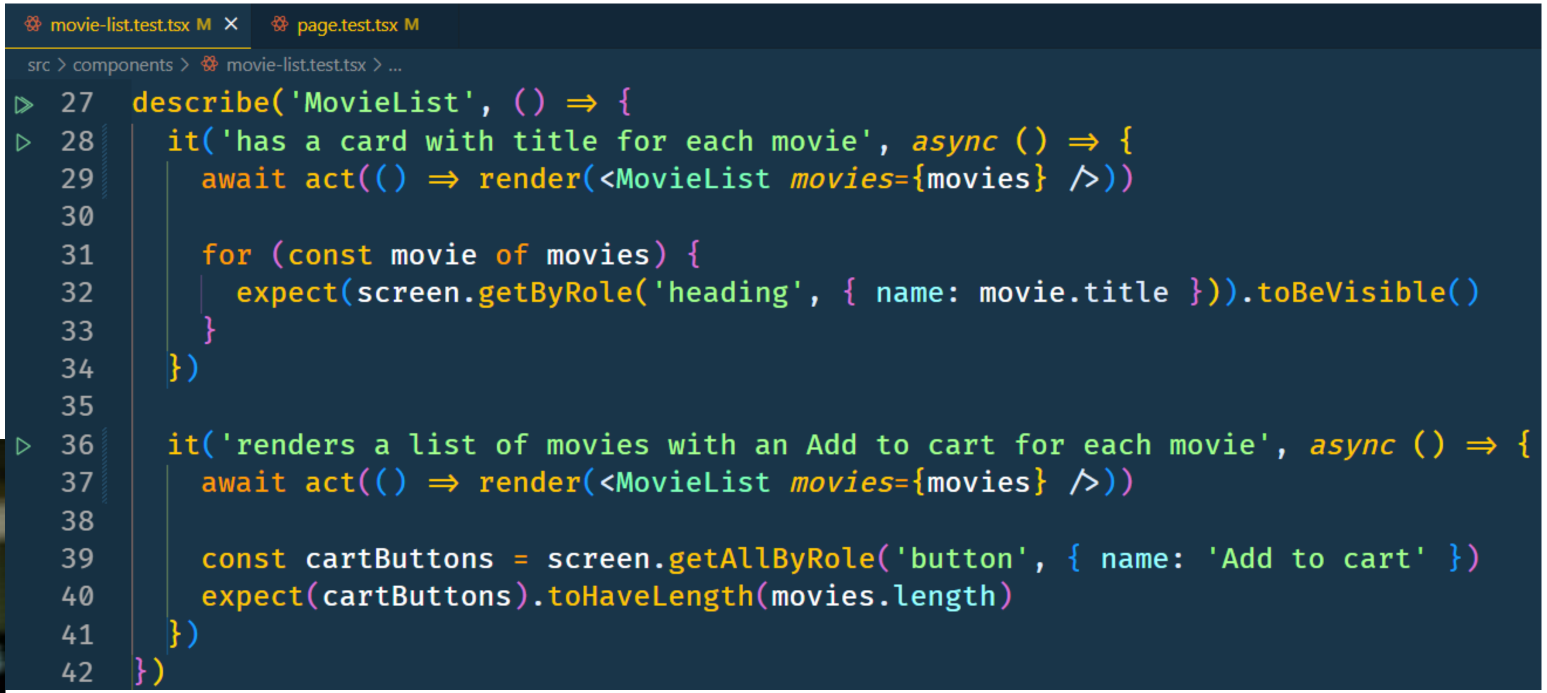
page.test.tsx


movie-list.test.tsx M page.test.tsx M ×

src > app > movies > page.test.tsx > ...

```
describe.skip('The Top Rated Movies page', () => {
  const originalFetch = globalThis.fetch

  beforeAll(() => {
    globalThis.fetch = jest
      .fn()
      .mockResolvedValue({ json: jest.fn().mockResolvedValue(movies) })
  })
```

movie-list.test.tsx



```tsx
describe('MovieList', () => {
  it('has a card with title for each movie', async () => {
    await act(() => render(<MovieList movies={movies} />))

    for (const movie of movies) {
      expect(screen.getByRole('heading', { name: movie.title })).toBeVisible()
    }
  })

  it('renders a list of movies with an Add to cart for each movie', async () => {
    await act(() => render(<MovieList movies={movies} />))

    const cartButtons = screen.getAllByRole('button', { name: 'Add to cart' })
    expect(cartButtons).toHaveLength(movies.length)
  })
})
```

See you in the next video

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

Testing async RSC’s

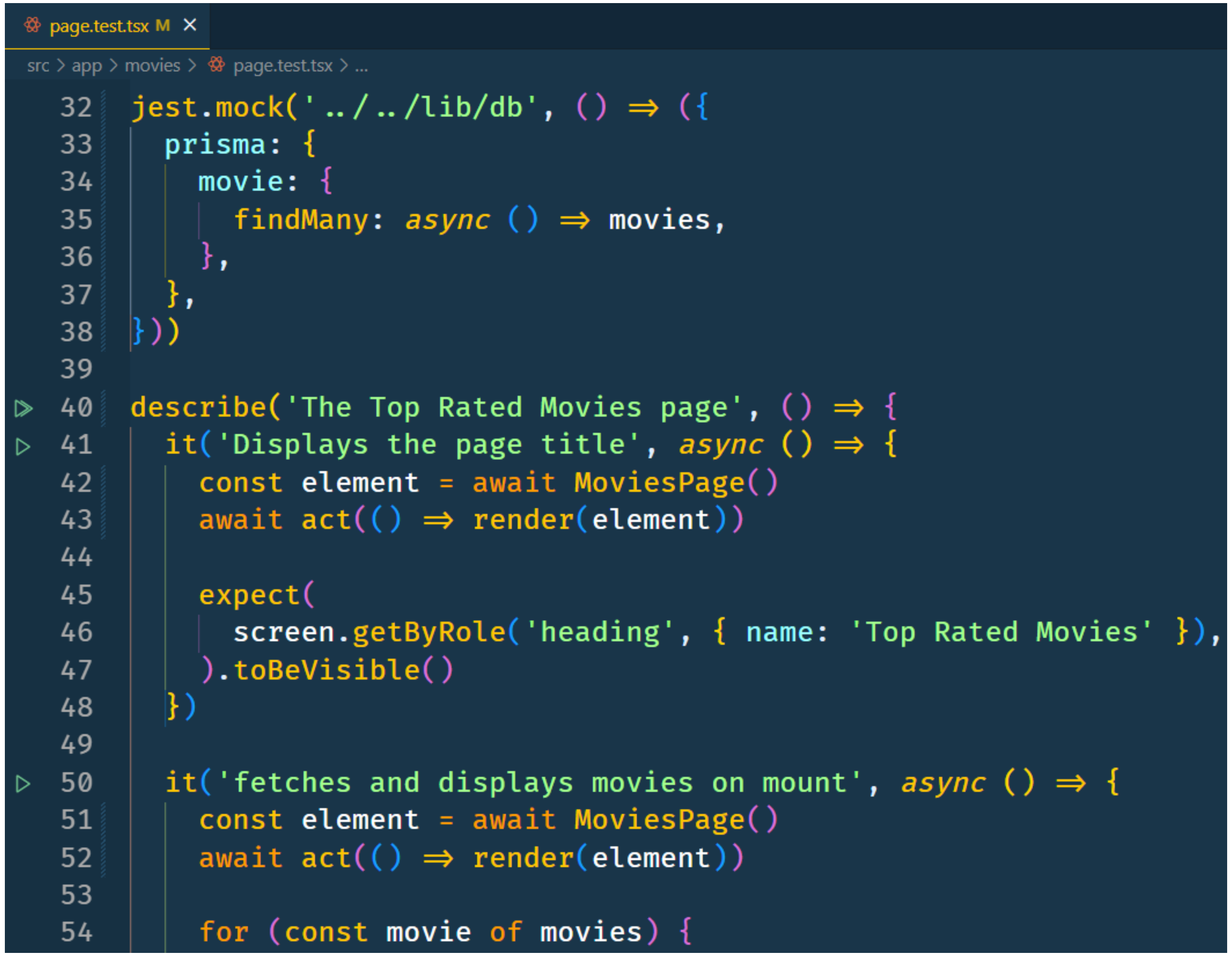
Testing async RSC's

- React Testing Library has **no support for async components** yet
  - As of February 2024
  - Hopefully that will be released soon
- **A component is just a function**
  - Call it as a normal function and await the rendered elements
- **Wrapping** an async RSC component **in <Suspense />** can also help

Recommendation

- **Unit test the normal components instead of the async RSC's**
  - Use async RSC to load data and pass this into normal components
- **Use end to end testing** if you need **to test logic in an async RSC**
  - But avoid this when possible as it is slower

page.test.tsx



```tsx
jest.mock('../../lib/db', () => ({
  prisma: {
    movie: {
      findMany: async () => movies,
    },
  },
}))

describe('The Top Rated Movies page', () => {
  it('Displays the page title', async () => {
    const element = await MoviesPage()
    await act(() => render(element))

    expect(
      screen.getByRole('heading', { name: 'Top Rated Movies' }),
    ).toBeVisible()
  })

  it('fetches and displays movies on mount', async () => {
    const element = await MoviesPage()
    await act(() => render(element))

    for (const movie of movies) {
```

See you in the next video

BOOST YOUR DEVELOPER POTENTIAL

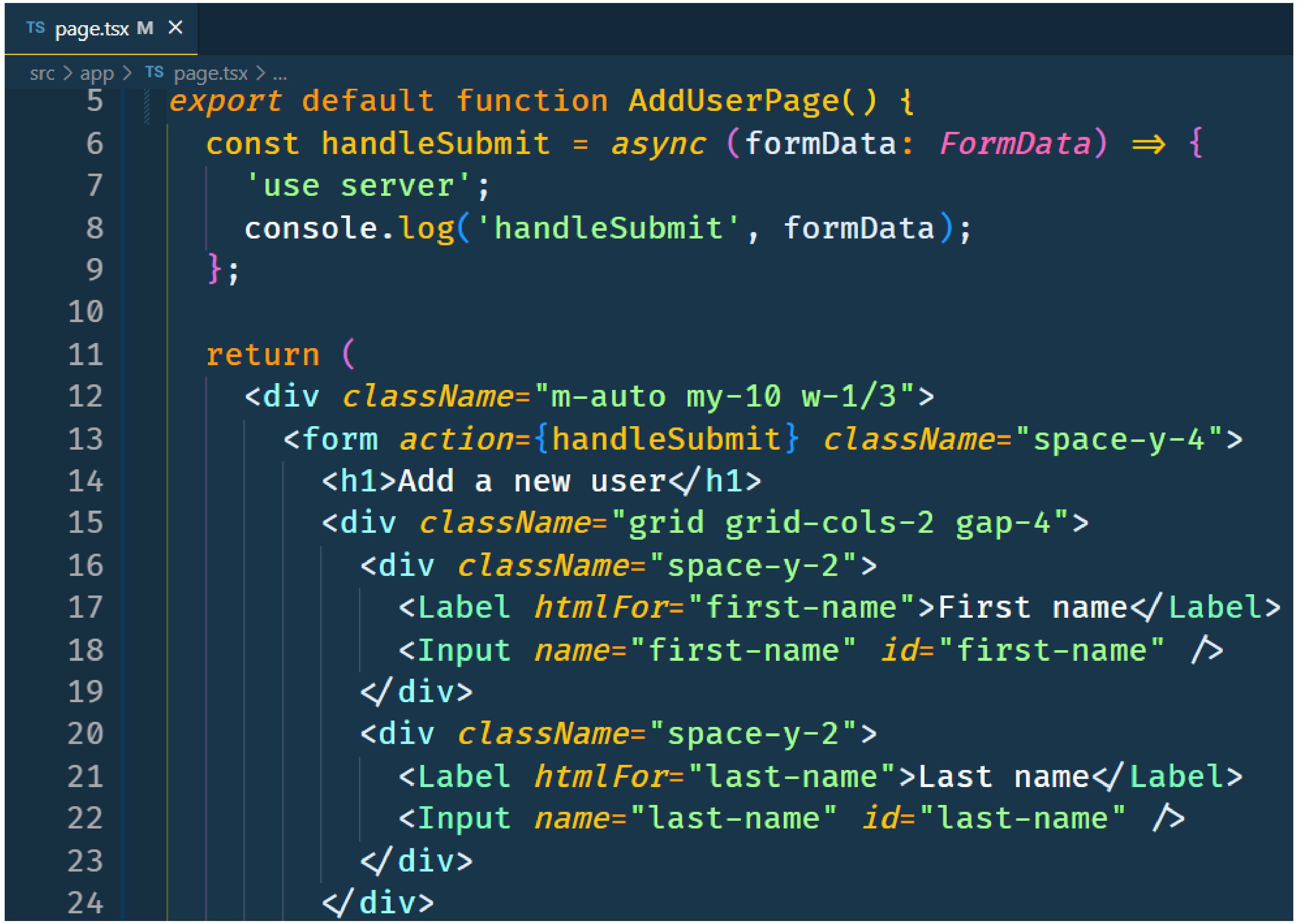
WITH REACT SERVER COMPONENTS

Server Actions

Server Actions

- **Server actions** are async functions that **are executed on the server**
  - Network serialization is done automatically
- **The 'use server' directive** marks a function as a server action
  - Can be added to the top of a file or individual function
- Server actions **can return a value** to the caller
  - Intended for mutations
  - Not to request large sets of data
- They **can be passed from a RSC to a client component as a prop**
  - Even though a function reference is normally not serializable
- Can also be used to **invalidate the client cache in Next.js**
  - When revalidatePath() or revalidateTag() is used on the server

Submitting a form


page.tsx M ×

src > app > page.tsx > ...

```tsx
export default function AddUserPage() {
  const handleSubmit = async (formData: FormData) => {
    'use server';
    console.log('handleSubmit', formData);
  };

  return (
    <div className="m-auto my-10 w-1/3">
      <form action={handleSubmit} className="space-y-4">
        <h1>Add a new user</h1>
        <div className="grid grid-cols-2 gap-4">
          <div className="space-y-2">
            <Label htmlFor="first-name">First name</Label>
            <Input name="first-name" id="first-name" />
          </div>
          <div className="space-y-2">
            <Label htmlFor="last-name">Last name</Label>
            <Input name="last-name" id="last-name" />
          </div>
        </div>
```

Server Actions and security

- **Server actions are network calls**
  - Just like another fetch request
- **Always treat input as untrusted**
  - Never assume client side validations etc. have been done

Server Actions & HTML forms

- An **HTML form** can call a server action using the **action prop**
  - This will even work when JavaScript is disabled in the browser
- **Form data** is passed as a FormData type parameter
- There are **several hooks** that make the client code more capable
  - useFormState
    - Allows updating form state based on the result of a form action
  - useFormStatus
    - Provides status information of the form submission

Calling a Server Action directly

- Server actions **can be called directly**
  - Just like any other async functions
- Arguments can be **any serializable data type**
  - Not just FormData

See you in the next video

Calling Server Actions from a <form />

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

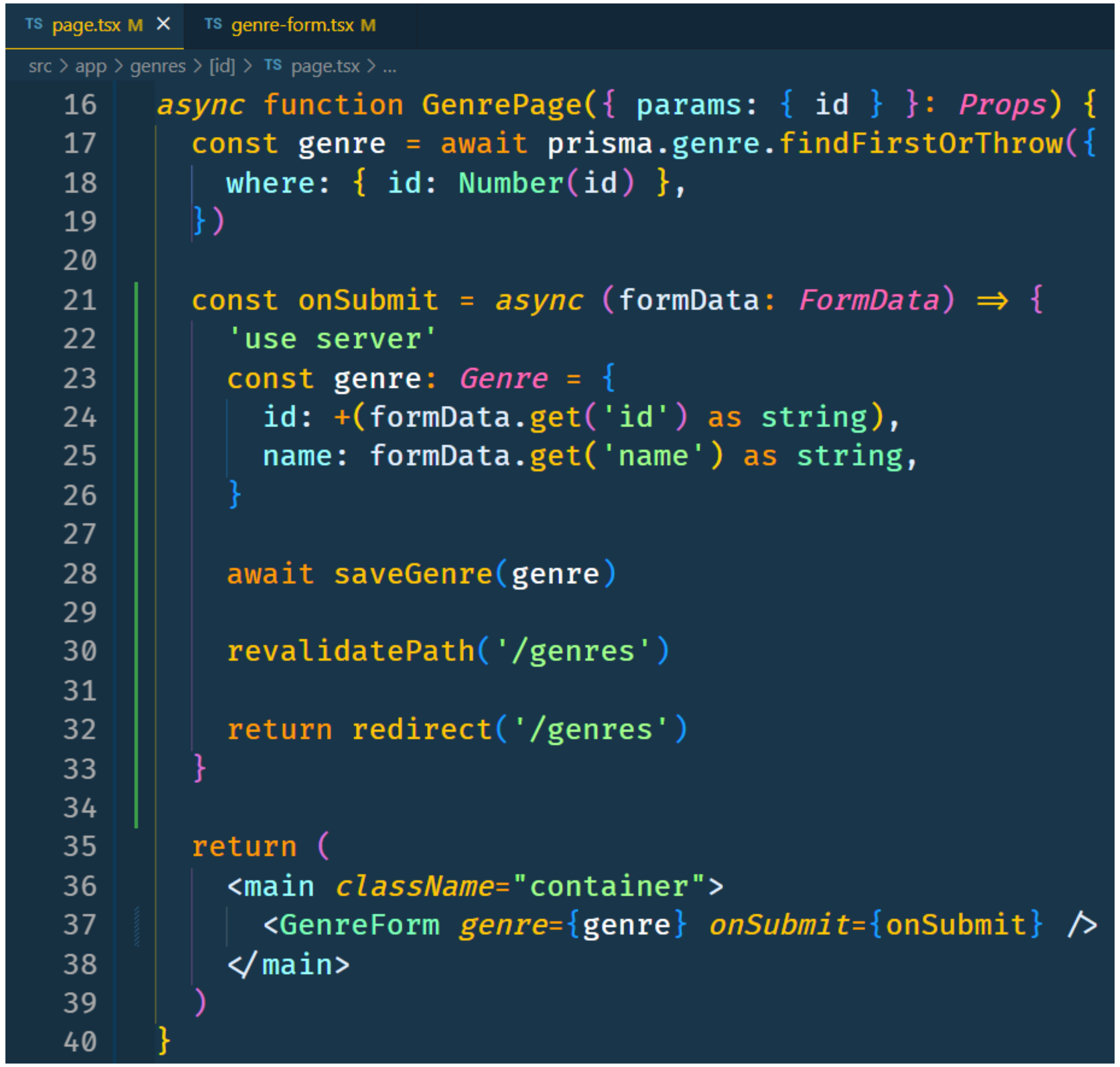
Calling Server Actions

From a <form />

Calling Server Actions

- **React Server Actions** are functions that we can **call on the client**
  - But then **execute on the server**
- Add the ***'use server'*** annotation
  - Can be at the top of a file or a single function
  - Not related to server components
- Can be passed as **the action of a client side <form />**
  - The forms data is passed as a **FormData** parameter
  - Even works if JavaScript is disabled ☺
- Can also be **called as a normal asynchronous function**
  - The network request is handled for you
- **Redirect to a different route** if required

/genres/[id] /page.tsx

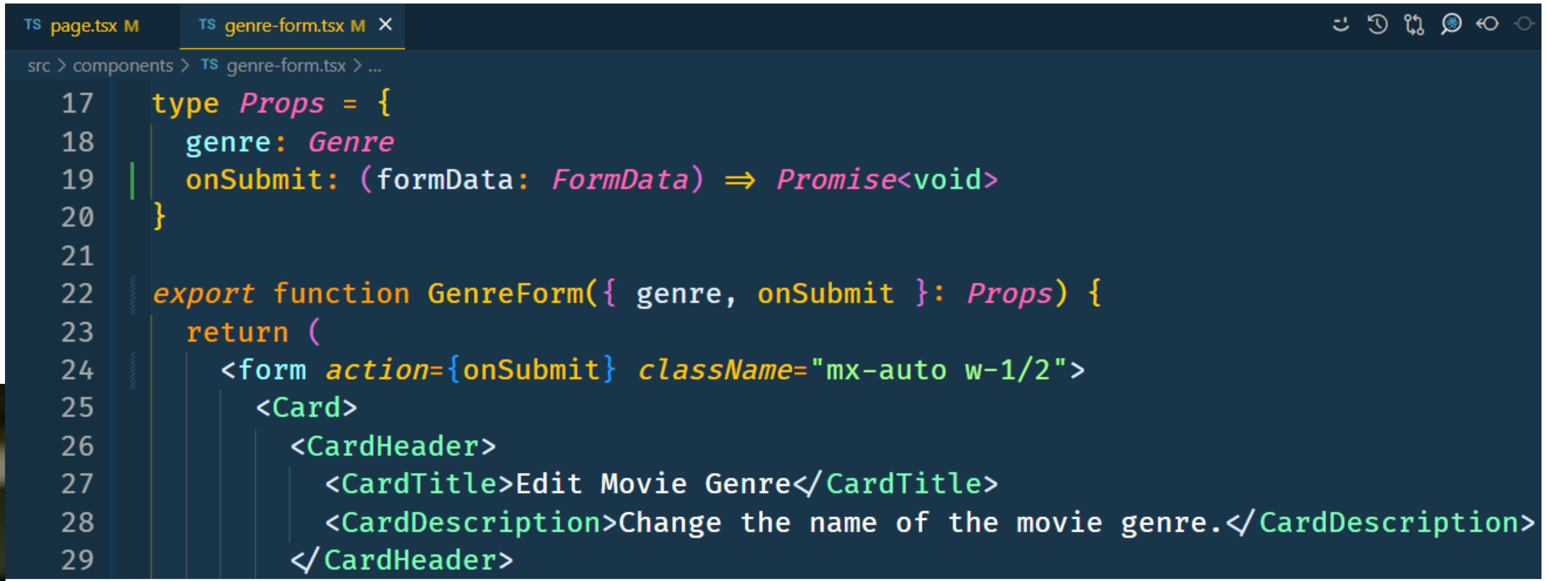


page.tsx M × genre-form.tsx M

src > app > genres > [id] > page.tsx > ...

```tsx
async function GenrePage({ params: { id } }: Props) {
  const genre = await prisma.genre.findFirstOrThrow({
    where: { id: Number(id) },
  })

  const onSubmit = async (formData: FormData) ⇒ {
    'use server'
    const genre: Genre = {
      id: +(formData.get('id') as string),
      name: formData.get('name') as string,
    }

    await saveGenre(genre)

    revalidatePath('/genres')

    return redirect('/genres')
  }

  return (
    <main className="container">
      <GenreForm genre={genre} onSubmit={onSubmit} />
    </main>
  )
}
```

genre-form.tsx



```
type Props = {
  genre: Genre
  onSubmit: (formData: FormData) ⇒ Promise<void>
}

export function GenreForm({ genre, onSubmit }: Props) {
  return (
    <form action={onSubmit} className="mx-auto w-1/2">
      <Card>
        <CardHeader>
          <CardTitle>Edit Movie Genre</CardTitle>
          <CardDescription>Change the name of the movie genre.</CardDescription>
        </CardHeader>
```

See you in the next video

Using the useFormStatus() hook

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

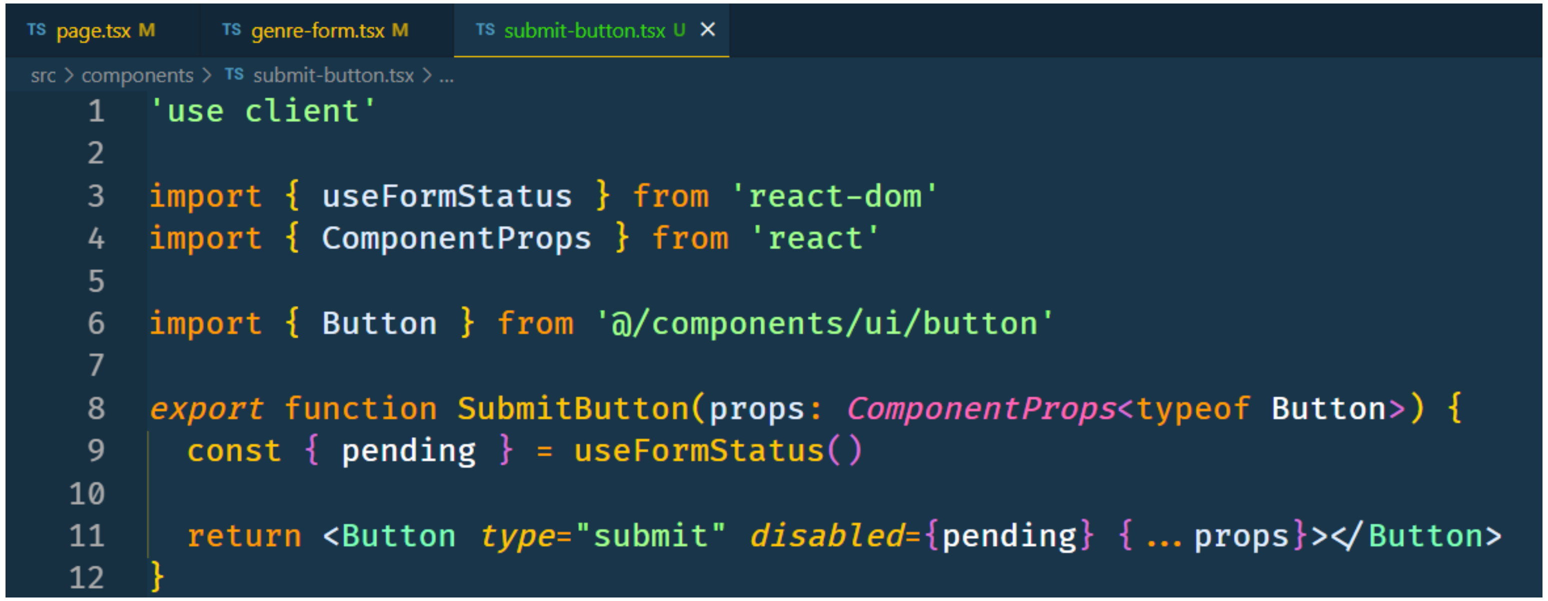
The useFormStatus() hook

With a <form />

The useFormStatus() hook

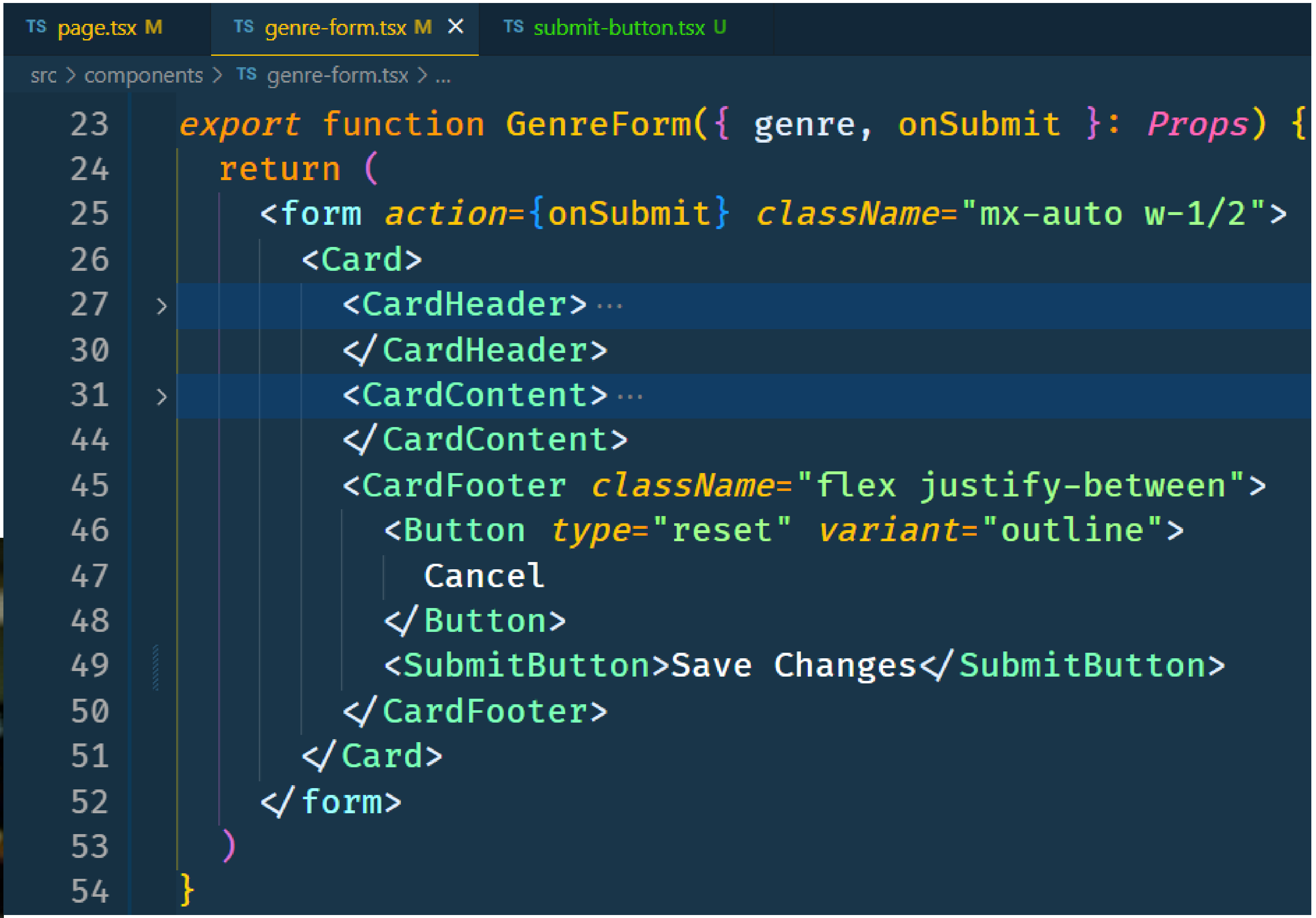
- The **useFormStatus() hook gives you form status information**
  - If a request is in progress and if so the form data
- It must be used in a **different component** than the form
  - Must be rendered as a child component of the <form/>

submit-button.tsx



```tsx
'use client'

import { useFormStatus } from 'react-dom'
import { ComponentProps } from 'react'

import { Button } from '@/components/ui/button'

export function SubmitButton(props: ComponentProps<typeof Button>) {
  const { pending } = useFormStatus()

  return <Button type="submit" disabled={pending} {...props}></Button>
}
```

genre-form.tsx



```tsx
export function GenreForm({ genre, onSubmit }: Props) {
  return (
    <form action={onSubmit} className="mx-auto w-1/2">
      <Card>
        <CardHeader>···
        </CardHeader>
        <CardContent>···
        </CardContent>
        <CardFooter className="flex justify-between">
          <Button type="reset" variant="outline">
            Cancel
          </Button>
          <SubmitButton>Save Changes</SubmitButton>
        </CardFooter>
      </Card>
    </form>
  )
}
```

See you in the next video

Using the useFormState() hook

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

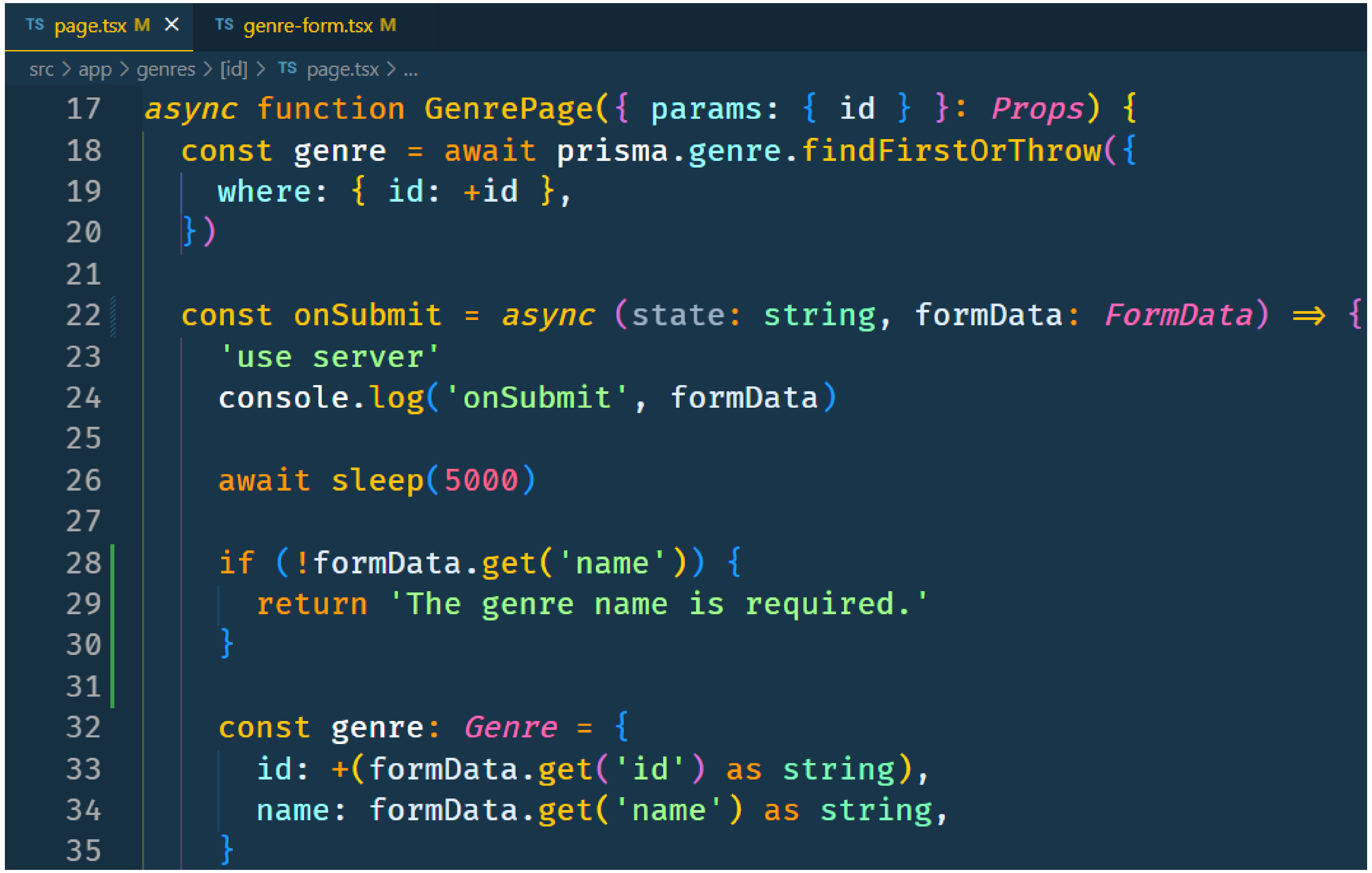
The useFormState() hook

With a <form />

The useFormState() hook

- The useFormState() allows you to **update state based on an action**
  - It takes the original action as a parameter and returns a new action
- The React Server Action **returns the new state**
  - And receives the previous state as the first parameter
- Still **works if JavaScript is disabled** ☺

/genres/[id] /page.tsx

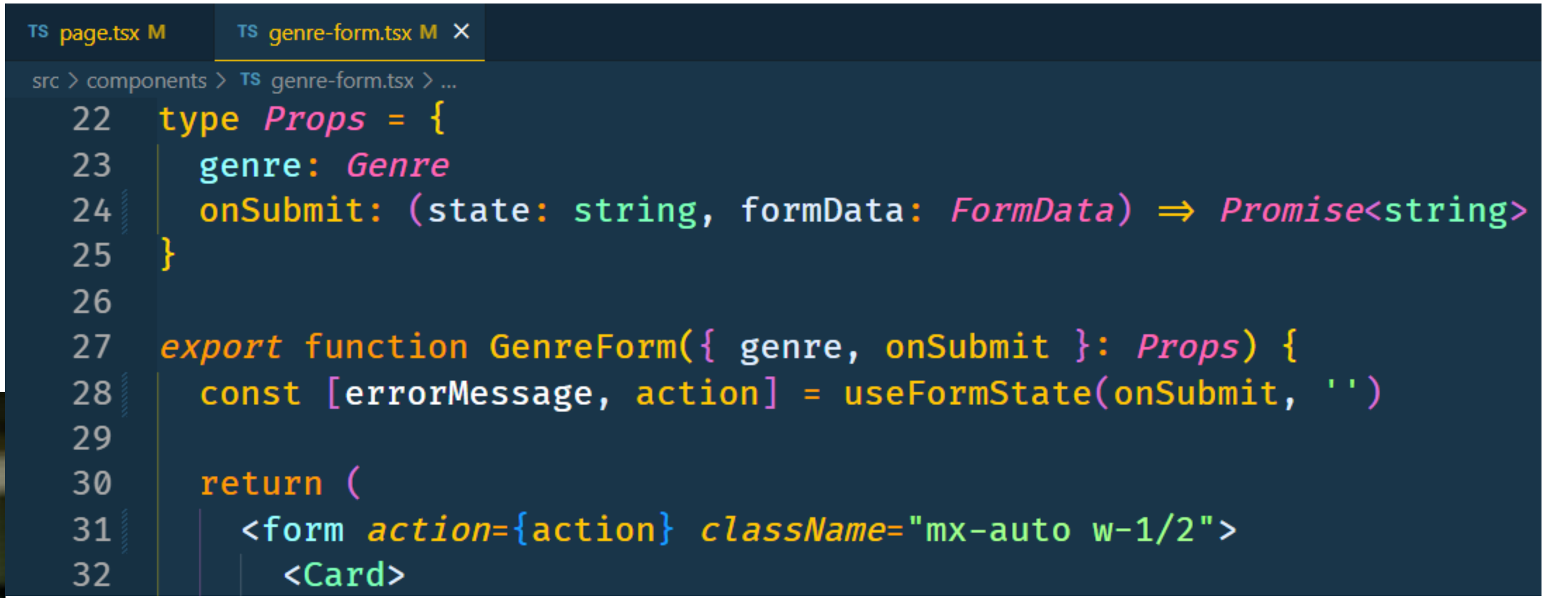


page.tsx M × genre-form.tsx M

src > app > genres > [id] > page.tsx > ...

```tsx
async function GenrePage({ params: { id } }: Props) {
  const genre = await prisma.genre.findFirstOrThrow({
    where: { id: +id },
  })

  const onSubmit = async (state: string, formData: FormData) => {
    'use server'
    console.log('onSubmit', formData)

    await sleep(5000)

    if (!formData.get('name')) {
      return 'The genre name is required.'
    }

    const genre: Genre = {
      id: +(formData.get('id') as string),
      name: formData.get('name') as string,
    }
```

genre-form.tsx



```tsx
type Props = {
  genre: Genre
  onSubmit: (state: string, formData: FormData) ⇒ Promise<string>
}

export function GenreForm({ genre, onSubmit }: Props) {
  const [errorMessage, action] = useFormState(onSubmit, '')

  return (
    <form action={action} className="mx-auto w-1/2">
      <Card>
```

See you in the next video

Using custom actions with a <button />

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

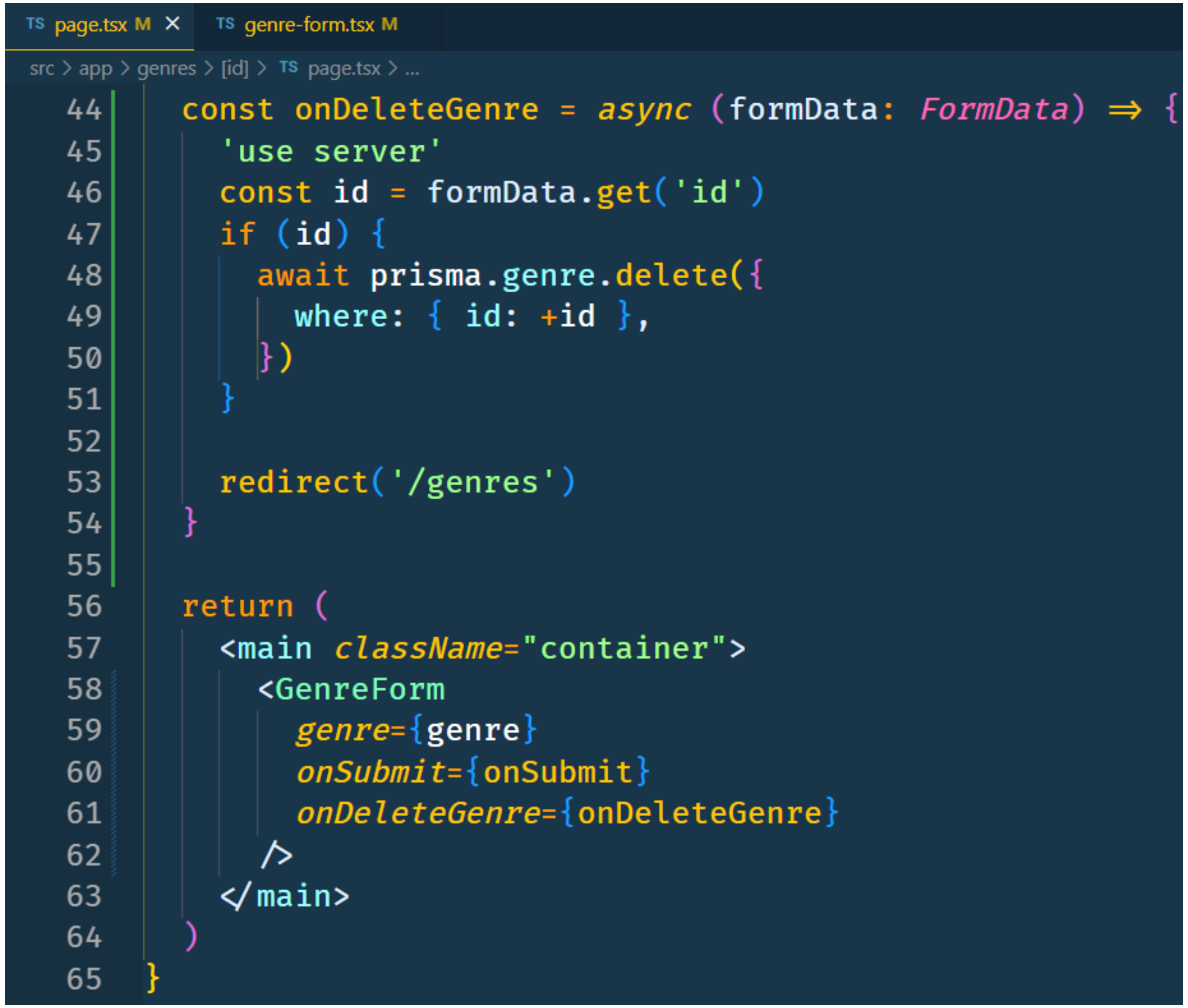
Using custom actions

With a <button />

Using custom actions

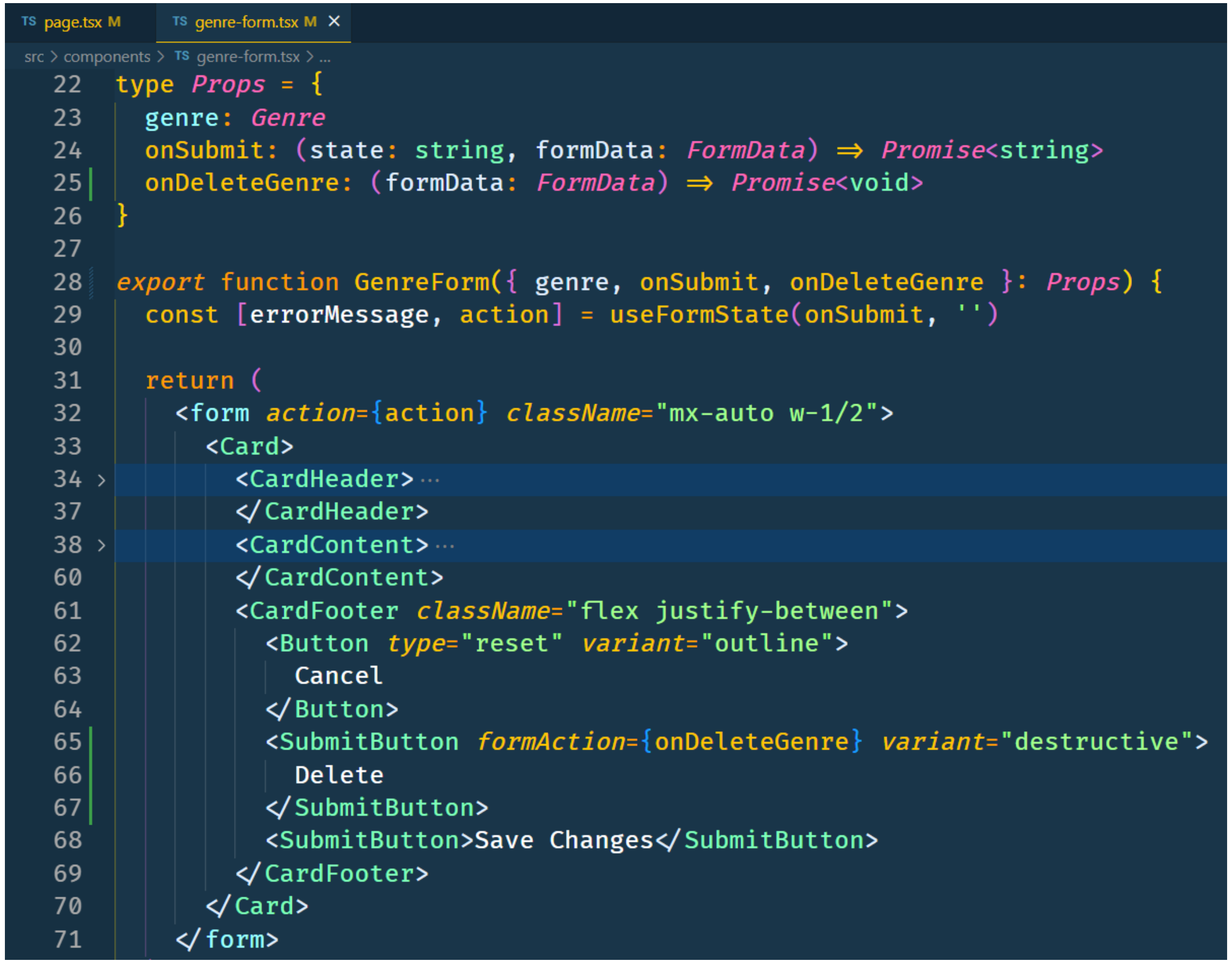
- A **submit button** can have a **formAction prop**
  - Overrides the form action
- Useful if you want **multiple different actions for a <form />**
  - Add to shopping cart or add to favorites for example
- Still **works if JavaScript is disabled** ☺

/genres/[id] /page.tsx



```tsx
  const onDeleteGenre = async (formData: FormData) => {
    'use server'
    const id = formData.get('id')
    if (id) {
      await prisma.genre.delete({
        where: { id: +id },
      })
    }

    redirect('/genres')
  }

  return (
    <main className="container">
      <GenreForm
        genre={genre}
        onSubmit={onSubmit}
        onDeleteGenre={onDeleteGenre}
      />
    </main>
  )
}
```

genre-form.tsx



```tsx
type Props = {
  genre: Genre
  onSubmit: (state: string, formData: FormData) => Promise<string>
  onDeleteGenre: (formData: FormData) => Promise<void>
}

export function GenreForm({ genre, onSubmit, onDeleteGenre }: Props) {
  const [errorMessage, action] = useFormState(onSubmit, '')

  return (
    <form action={action} className="mx-auto w-1/2">
      <Card>
        <CardHeader>···
        </CardHeader>
        <CardContent>···
        </CardContent>
        <CardFooter className="flex justify-between">
          <Button type="reset" variant="outline">
            Cancel
          </Button>
          <SubmitButton formAction={onDeleteGenre} variant="destructive">
            Delete
          </SubmitButton>
          <SubmitButton>Save Changes</SubmitButton>
        </CardFooter>
      </Card>
    </form>
```

See you in the next video

Calling Server Actions from any other code

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

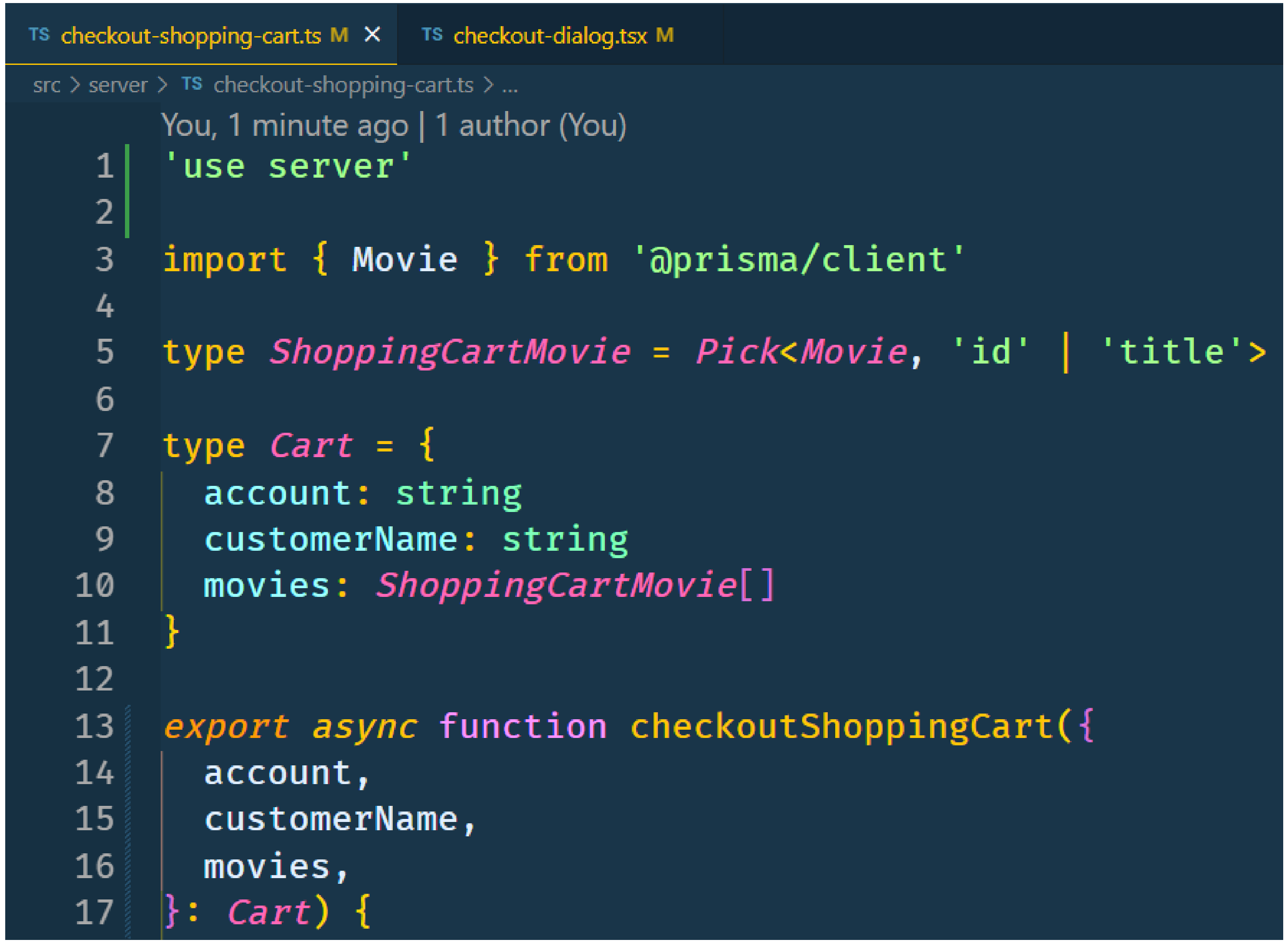
Calling Server Actions

From any other code

Calling Server Actions

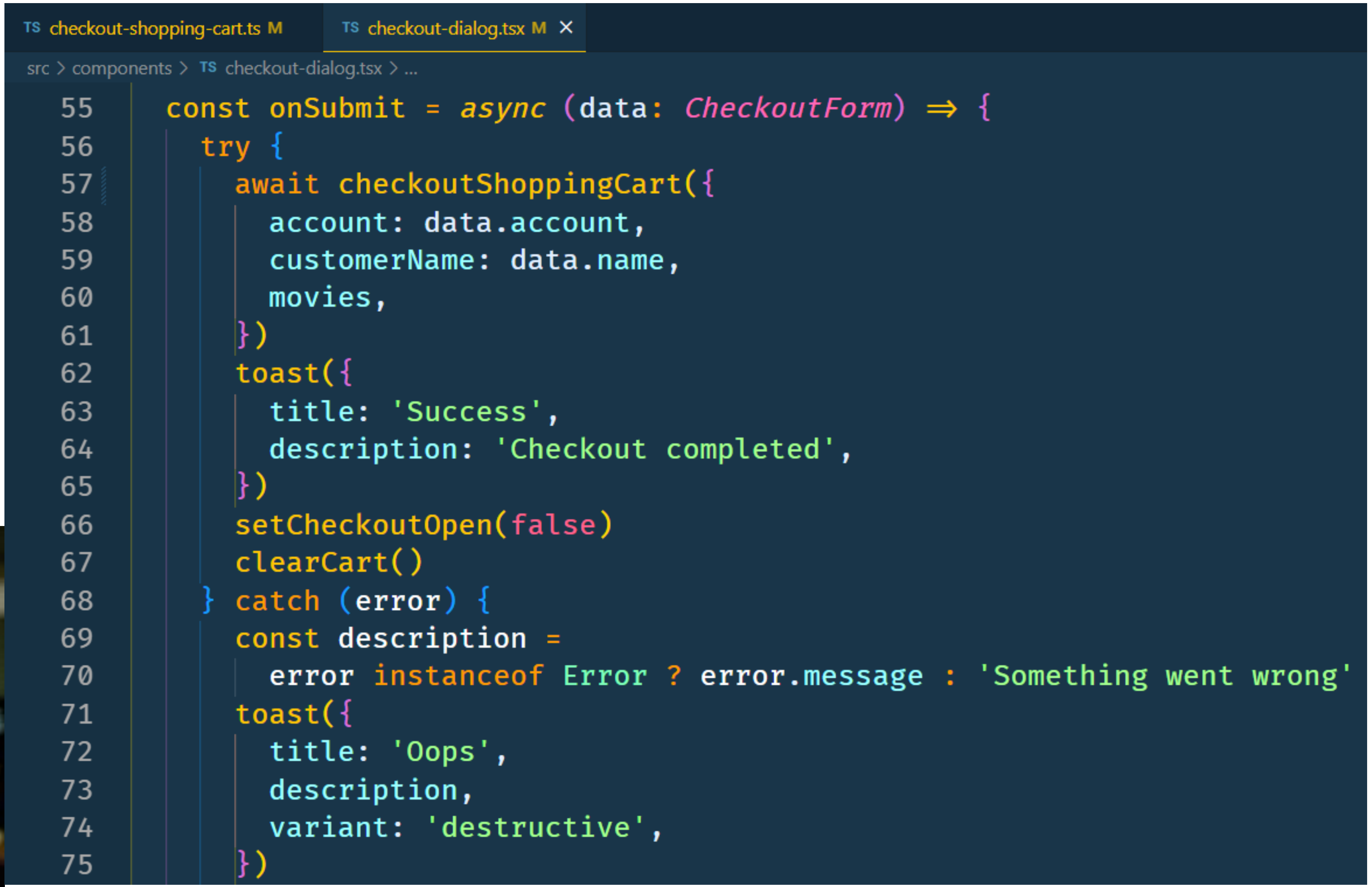
- **React Server Actions** can also used directly
  - Called as a normal asynchronous function
- It’s still an **HTTP post request** behind scenes
  - The network request is automatically handled for you

checkout-shopping-cart.ts


TS checkout-shopping-cart.ts M ×
TS checkout-dialog.tsx M
src > server > TS checkout-shopping-cart.ts > ...
You, 1 minute ago | 1 author (You)

```ts
'use server'

import { Movie } from '@prisma/client'

type ShoppingCartMovie = Pick<Movie, 'id' | 'title'>

type Cart = {
  account: string
  customerName: string
  movies: ShoppingCartMovie[]
}

export async function checkoutShoppingCart({
  account,
  customerName,
  movies,
}: Cart) {
```

checkout-dialog.tsx


checkout-shopping-cart.ts M
checkout-dialog.tsx M
src > components > checkout-dialog.tsx > ...

```
const onSubmit = async (data: CheckoutForm) => {
  try {
    await checkoutShoppingCart({
      account: data.account,
      customerName: data.name,
      movies,
    })
    toast({
      title: 'Success',
      description: 'Checkout completed',
    })
    setCheckoutOpen(false)
    clearCart()
  } catch (error) {
    const description =
      error instanceof Error ? error.message : 'Something went wrong'
    toast({
      title: 'Oops',
      description,
      variant: 'destructive',
    })
```

See you in the next video

BOOST YOUR DEVELOPER POTENTIAL

WITH REACT SERVER COMPONENTS

Recommendations with React Server Components

Recommendations

- **Start with Shared** components
  - Can run on the server or client as needed
  - Will default to act as Server Components
- Switch to **Server only components if needed**
  - When you need to use server side capabilities
- Only use **Client only components when absolutely needed**
  - Local state or side effects
  - Interactivity
  - Required browser API's
- Learn all about the **new capabilities of Next.js**
  - App Router
  - Caching

Conclusion

- React Server Components are a **great new addition to React**
  - Helps with keeping the client more responsive
  - Makes the application architecture easier
- **Use Next.js and the App Router**
  - Because you need a server
- React **Client Components**
  - Are components with state and interactivity and require 'use client'
- Control caching of React **Server Components**
  - Because Next.js is quite aggressive about caching
- **React Server Components are streamed**
  - And uses Suspense boundaries until they are done
- **Server Actions** are a great way to call back into the server
  - They also update the invalidated server components on the client

Thank you for joining

Share your thoughts