

[HOME](#) [AGENDA](#)

bitbash

[SPEAKERS](#) [TICKETS](#)

WINTERGLOW EDITION



26/27 JAN 2024

37
37

DAYS
TO GO

POWERED BY **infoSupport**
Solid Innovator

Mastering React Server Components

Unlocking the Future of React Development

Maurice de Beijer
@mauricedb



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



Topics

- What are **React Server Components** and why would you care?
- Turning a React **Client Component** into a React **Server Component**
- Using **Next.js** and the **App Router**
- **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**

Type it out
by hand?

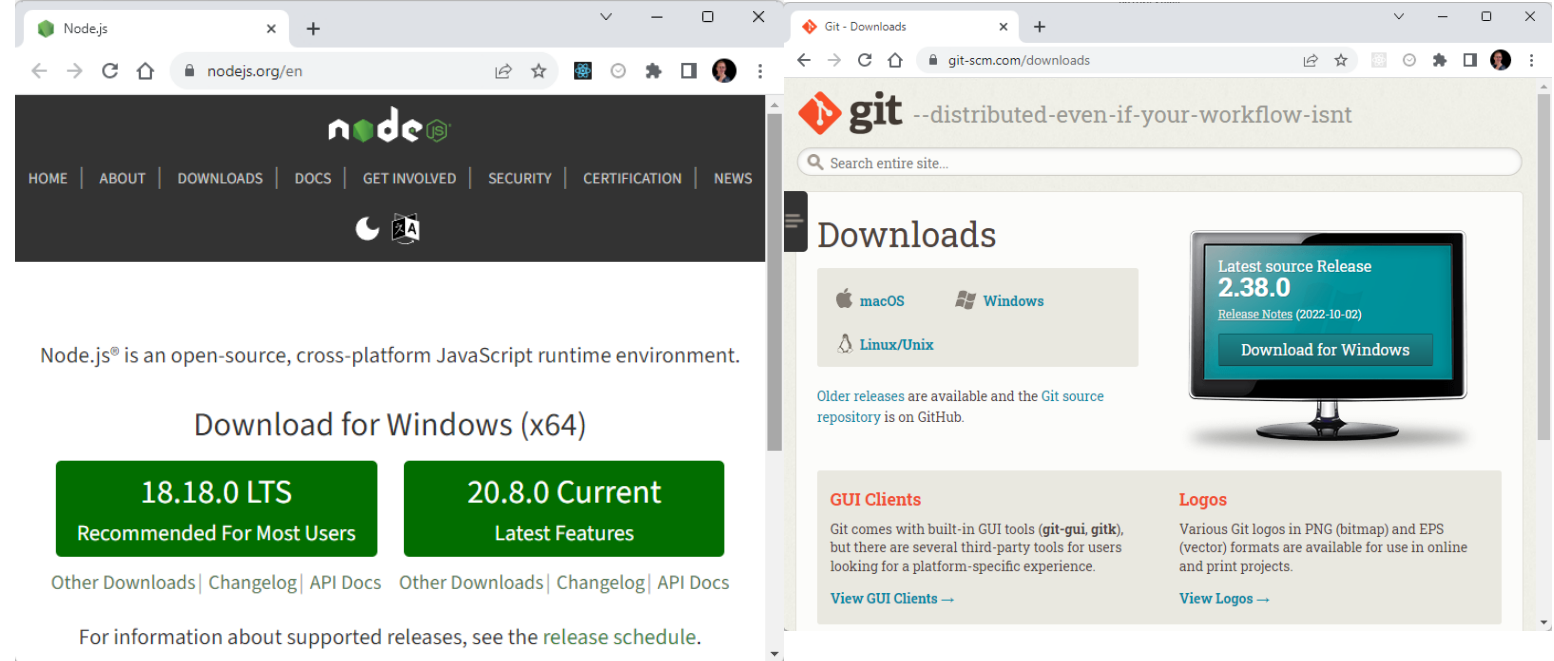
"Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now!"

Prerequisites

Install Node & NPM

Install the GitHub repository

Install Node.js & NPM



```
Windows PowerShell
PS C:\Users\mauri> node --version
v18.18.0
PS C:\Users\mauri> git --version
git version 2.41.0.windows.1
PS C:\Users\mauri> npm --version
10.1.0
```

Following Along



```
File Edit Selection View Go Run Terminal Help
TS page.tsx ...movies M TS page.tsx ...[id] M TS routes.ts M X
src > app > api > movies > [id] > TS routes.ts > ...
You, 4 days ago | 1 author (You)
1 import { saveMovie } from '@server/save-movie'
2 import { Movie } from '@prisma/client'
3 import { NextRequest, NextResponse } from 'next/server'
4
5 export async function PUT(request: NextRequest) {
6   try {
7     const movie = (await request.json()) as Movie
8
9     await saveMovie(movie)
10
11     return new NextResponse(null, {
12       status: 204,
13     })
14   } catch (error) {
15     console.error(error)
16
17     return new NextResponse(null, {
18       status: 400,
19     })
20   }
21 }
```

- Repo: <https://github.com/mauricedb/bitbash-rsc-2024>
- Slides: <https://bit.ly/bitbash-rsc-2024>

Create a new Next.js app

```
Windows PowerShell
PS C:\Repos> npx create-next-app@latest react-berlin-2023-ws
Need to install the following packages:
create-next-app@14.0.3
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-berlin-2023-ws.

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

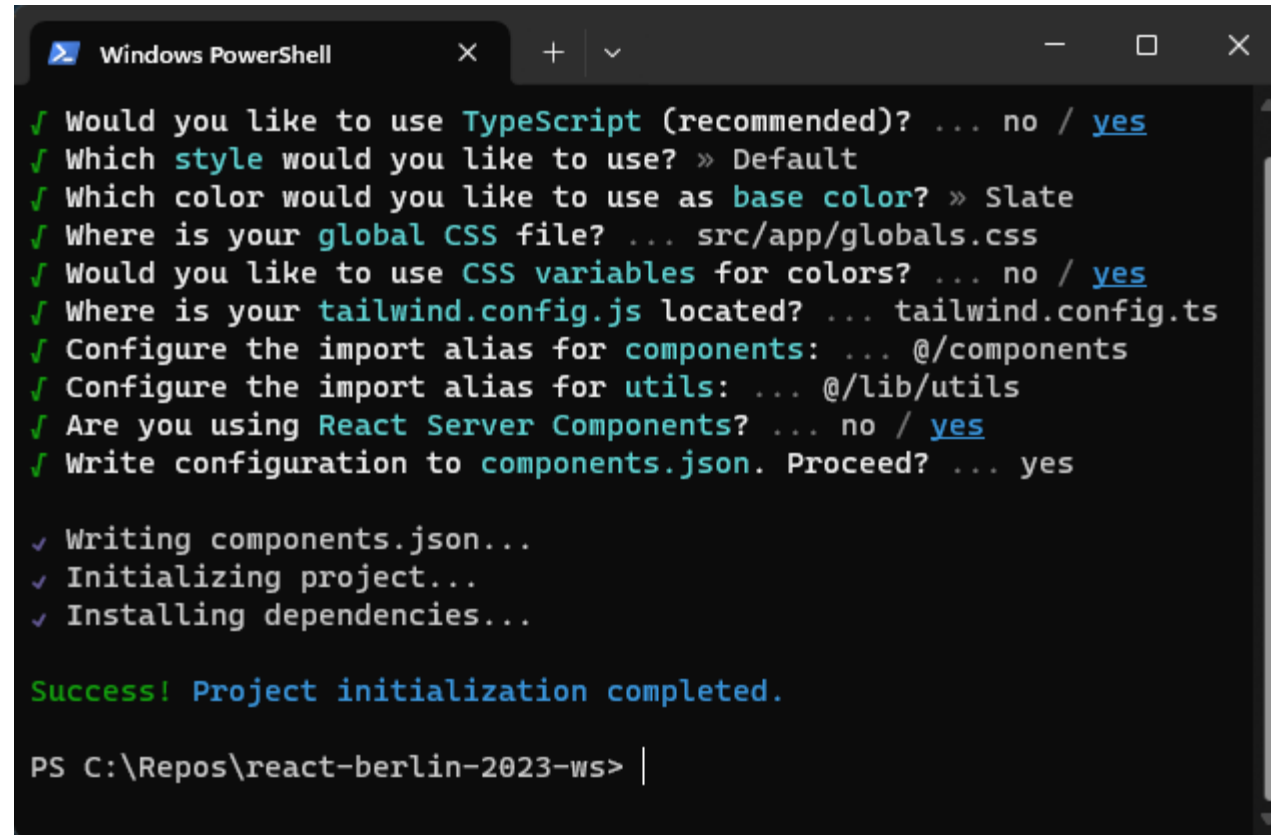
added 332 packages, and audited 333 packages in 24s

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

found 0 vulnerabilities
Initialized a git repository.

Success! Created react-berlin-2023-ws at C:\Repos\react-berlin-2023-ws
PS C:\Repos> |
```

Adding Shadcn support



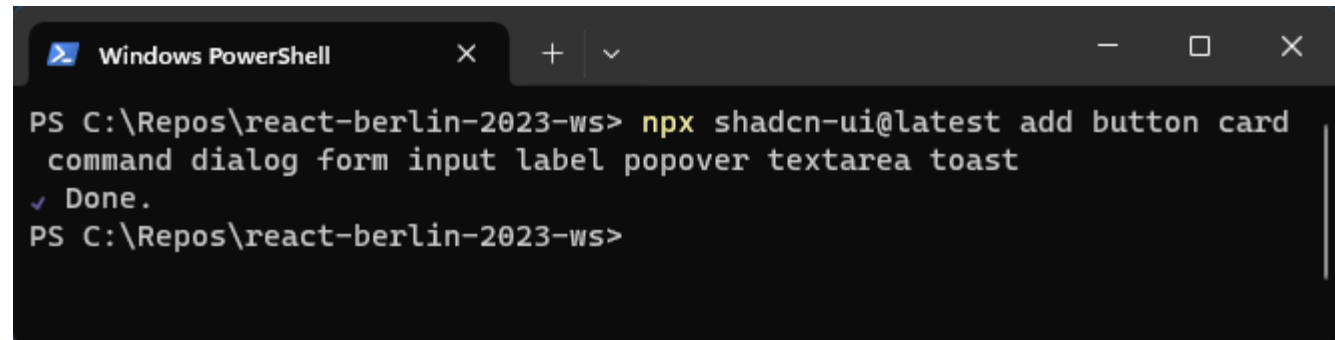
```
Windows PowerShell
✓ Would you like to use TypeScript (recommended)? ... no / yes
✓ Which style would you like to use? » Default
✓ Which color would you like to use as base color? » Slate
✓ Where is your global CSS file? ... src/app/globals.css
✓ Would you like to use CSS variables for colors? ... 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.

PS C:\Repos\react-berlin-2023-ws> |
```

Adding Shadcn components



```
Windows PowerShell
PS C:\Repos\react-berlin-2023-ws> npx shadcn-ui@latest add button card
command dialog form input label popover textarea toast
✓ Done.
PS C:\Repos\react-berlin-2023-ws>
```

The changes



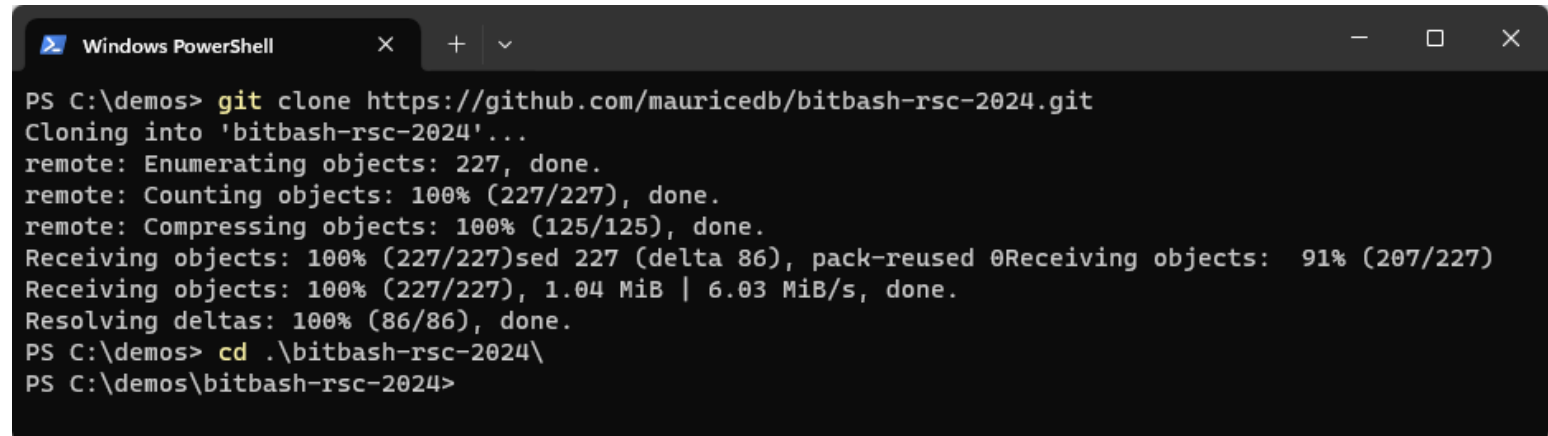
Commits

main

Commits on Jan 12, 2024

Calling Server Actions directly	mauricedb committed 5 days ago	8cef23	<>
Calling Server Actions From a <form />	mauricedb committed 5 days ago	b6b38fa	<>
Loading the genres on the server	mauricedb committed 5 days ago	21e623d	<>
Using an RSC as a child of a client component	mauricedb committed 5 days ago	5c96728	<>
Site layout as an RSC	mauricedb committed 5 days ago	31906a8	<>
Suspense & RSC pages	mauricedb committed 5 days ago	50f1083	<>
Prevent over fetching	mauricedb committed 5 days ago	52c1abe	<>
Querying the database from a RSC	mauricedb committed 5 days ago	fddf6a0	<>
Updates and caching	mauricedb committed 5 days ago	35998ca	<>
Turning a React Client Component into a Server Component	mauricedb committed 5 days ago	93be3cc	<>

Clone the GitHub Repository



```
Windows PowerShell
PS C:\demos> git clone https://github.com/mauricedb/bitbash-rsc-2024.git
Cloning into 'bitbash-rsc-2024'...
remote: Enumerating objects: 227, done.
remote: Counting objects: 100% (227/227), done.
remote: Compressing objects: 100% (125/125), done.
Receiving objects: 100% (227/227) | 1.04 MiB | 6.03 MiB/s, done.
Receiving objects: 100% (227/227), 1.04 MiB | 6.03 MiB/s, done.
Resolving deltas: 100% (86/86), done.
PS C:\demos> cd .\bitbash-rsc-2024\
PS C:\demos\bitbash-rsc-2024>
```

Install NPM Packages

```
Windows PowerShell
PS C:\demos\bitbash-rsc-2024> npm install

> react-berlin-2023-ws@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"

SQLite database dev.db created at file:./dev.db

Applying migration `20231206113326_init`

The following migration(s) have been applied:

migrations/
├─ 20231206113326_init/
└─ migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v5.6.0) to .\node_modules\@prisma\client in 59ms

Running seed command `ts-node --compiler-options {"module":"CommonJS"} prisma/seed.ts` ...

The seed command has been executed.

Update available 5.6.0 -> 5.8.1
Run the following to update
npm i --save-dev prisma@latest
npm i @prisma/client@latest

added 419 packages, and audited 420 packages in 19s

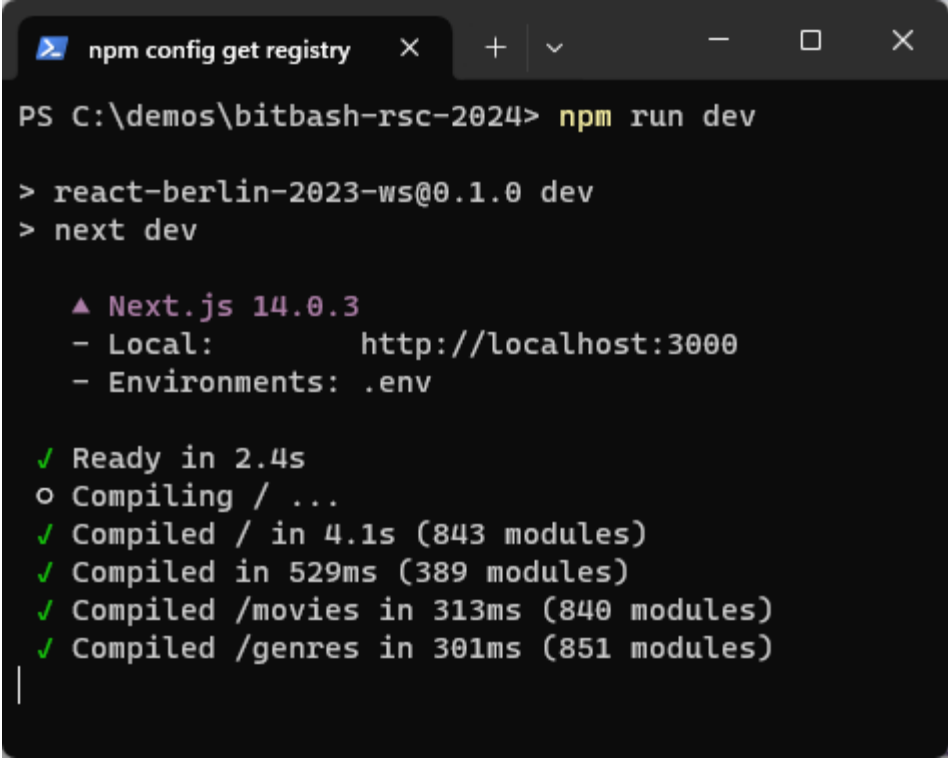
122 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\demos\bitbash-rsc-2024>
```

Start branch

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

Start the application



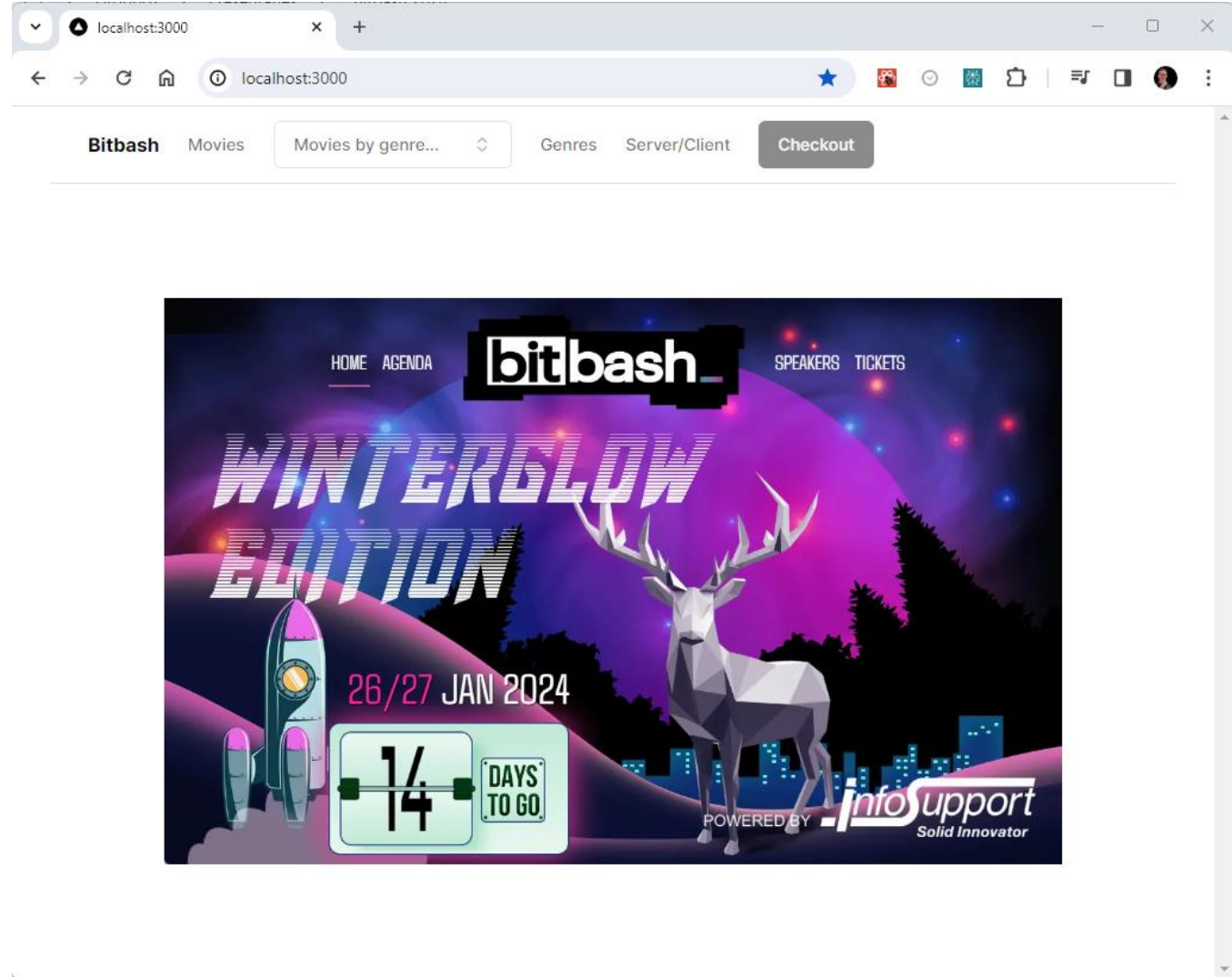
```
npm config get registry x + v - □ x
PS C:\demos\bitbash-rsc-2024> npm run dev


> react-berlin-2023-ws@0.1.0 dev
> next dev

  ▲ Next.js 14.0.3
    - Local:      http://localhost:3000
    - Environments: .env

✓ Ready in 2.4s
○ Compiling / ...
✓ Compiled / in 4.1s (843 modules)
✓ Compiled in 529ms (389 modules)
✓ Compiled /movies in 313ms (840 modules)
✓ Compiled /genres in 301ms (851 modules)
```


The application



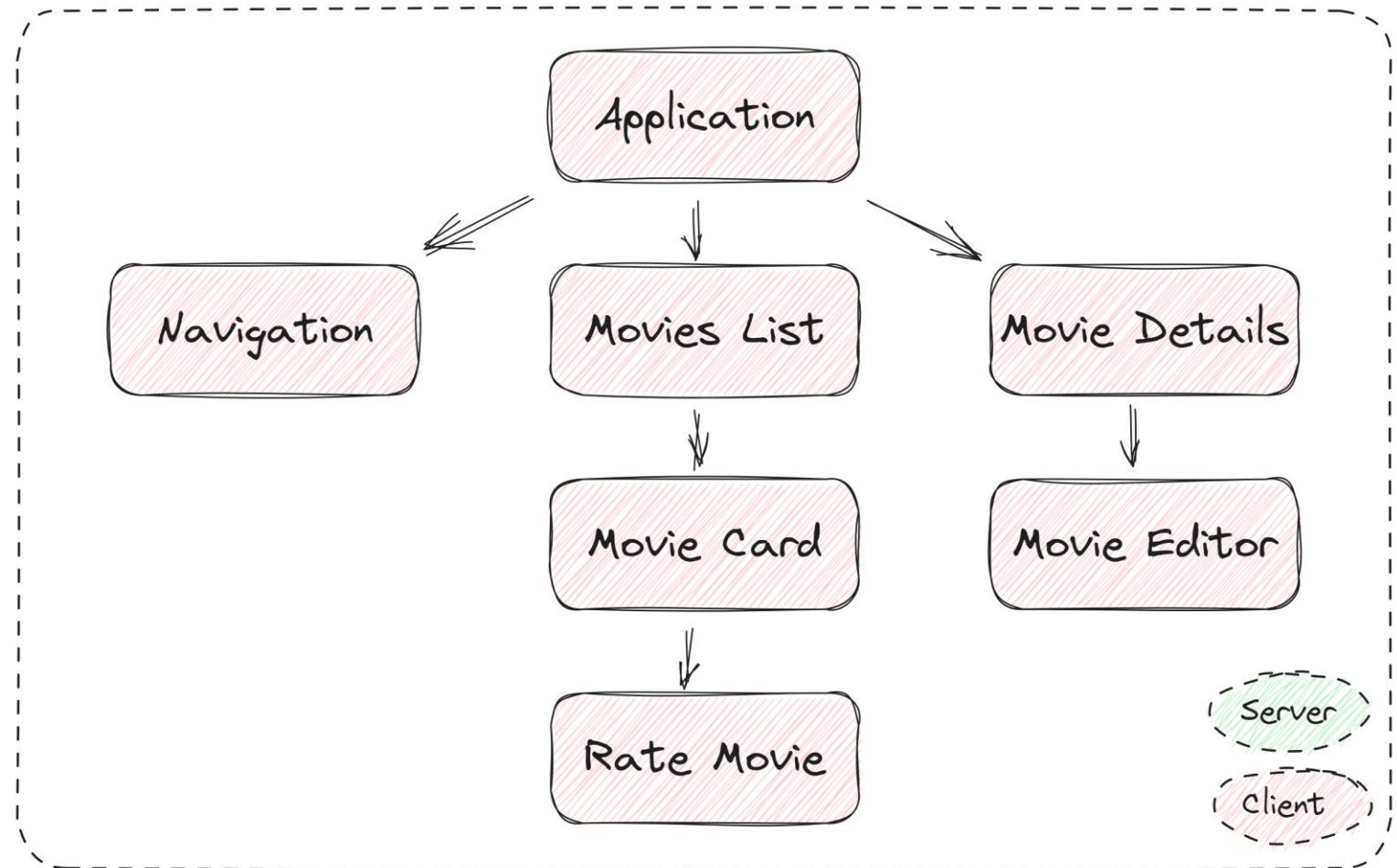


What are React Server Components?

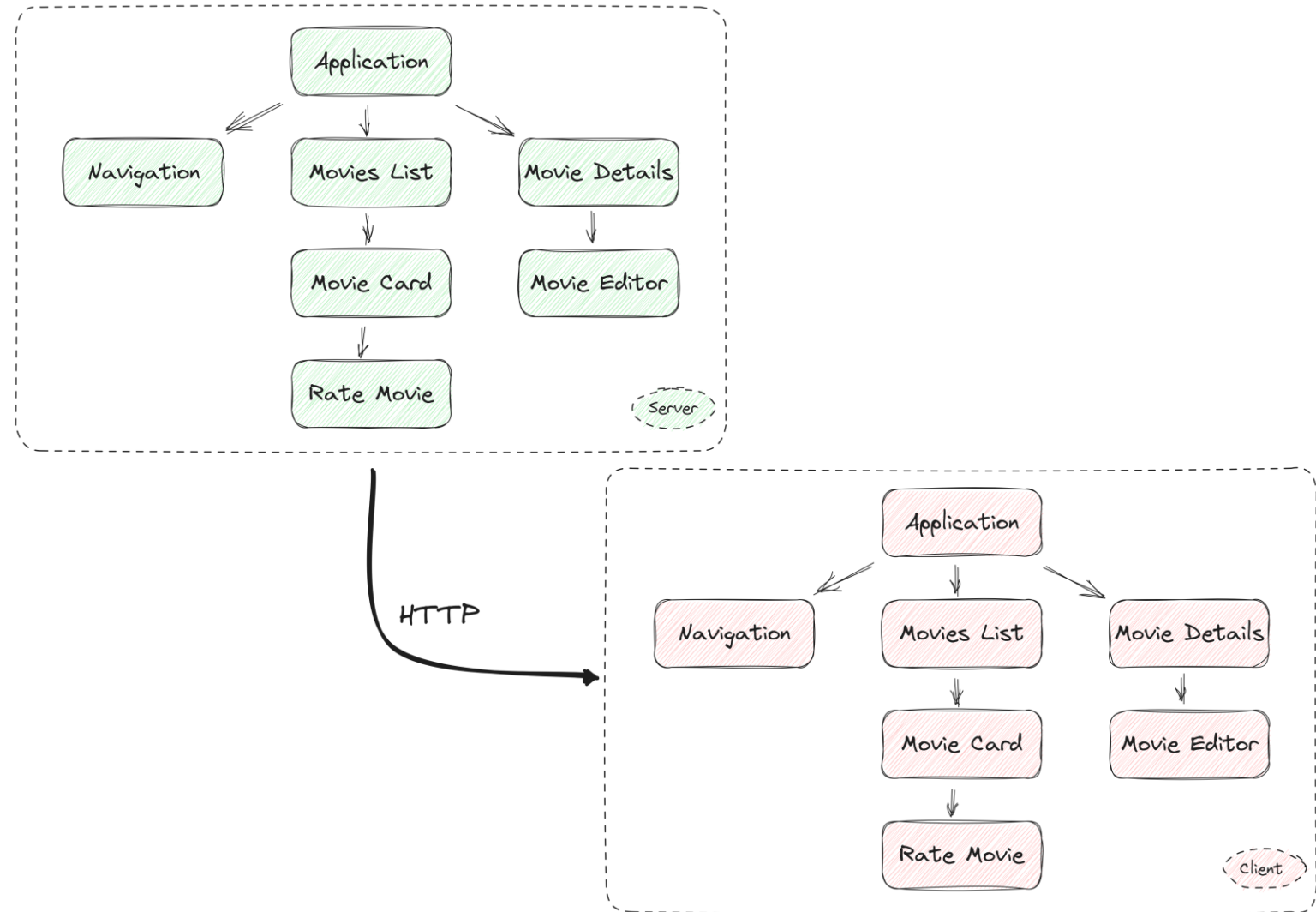
React Server Components

- React Server Components (RSC) **only execute on the server**
 - Traditionally React components always execute in the browser
 - The server can even be a build server instead of a runtime server
- 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** 😊

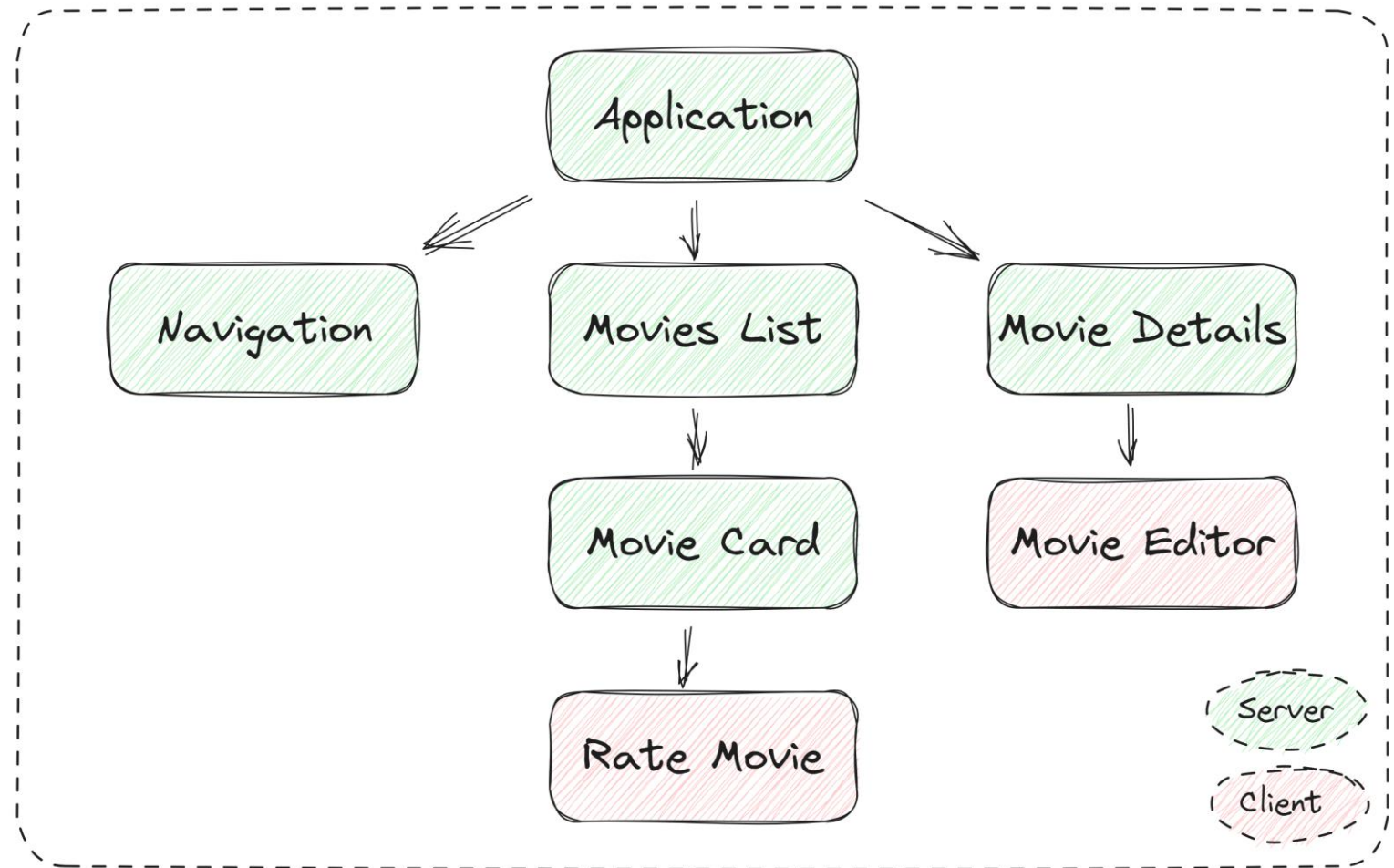
Before RSC



Server Side Rendering



With RSC



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 require TypeScript 5.1 📌


React Server Component

```
TS genre-loader.tsx ×
src > components > TS genre-loader.tsx > ...
You, 2 weeks ago | 1 author (You)
1 import { prisma } from '@lib/db'
2 import { GenreSelector } from '@components/genre-selector'
3
4 export async function GenreLoader() {
5   const genres = await prisma.genre.findMany({
6     orderBy: { name: 'asc' },
7   })
8
9   return <GenreSelector genres={genres} />
10 }
```


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

```
TS movie-form.tsx ×
src > components > TS movie-form.tsx > ...
1  'use client'
2
3  import { zodResolver } from '@hookform/resolvers/zod'
4  import * as z from 'zod'
```



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 RSC's**
 - State, effects, browser API's etc. are client requirements

movies /page.tsx

```
TS page.tsx ...\movies M X TS movie-card.tsx M TS page.tsx ...[id] M TS movie-form.tsx M
src > app > movies > TS page.tsx > ...
11 export default async function MoviesPage({ searchParams: { genre } }: Props)
12   async function fetchMovies() {
13     const url = genre ? `/api/movies?genre=${genre}` : '/api/movies'
14     const rsp = await fetch(`http://localhost:3000${url}`)
15     const movies = await rsp.json()
16     return movies as Movie[]
17   }
18
19   const movies = await fetchMovies()
20
```

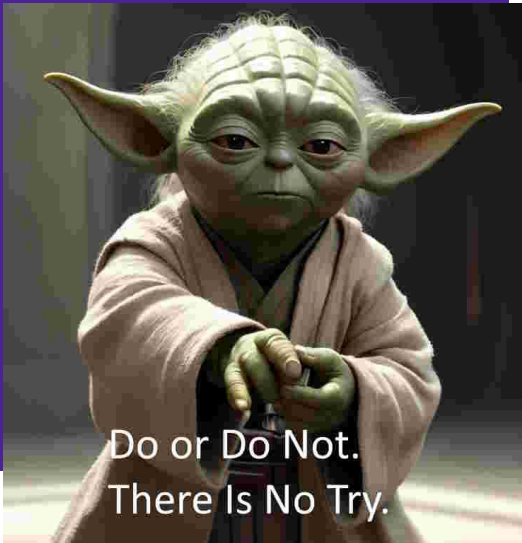
movie-card.tsx

```
TS page.tsx ...\movies M TS movie-card.tsx M X TS page.tsx ...[id] M
src > components > TS movie-card.tsx > ...
1 | 'use client'
2 |
3 | import Image from 'next/image'
4 | import Link from 'next/link'
```

movies/[id]
/page.tsx

```
TS page.tsx ...\movies M TS movie-card.tsx M TS page.tsx ...\[id] M X TS movie-form.tsx M
src > app > movies > [id] > TS page.tsx > ...
12 const MoviePage: FC<Props> = async ({ params: { id } }) => {
13   async function fetchMovie() {
14     const rsp = await fetch(`http://localhost:3000/api/movies/${id}`)
15     const movie = await rsp.json()
16     return movie as Movie
17   }
18
19   const movie = await fetchMovie()
```

movie-form.tsx



```
TS page.tsx ...\movies M TS movie-card.tsx M TS page.tsx ...\[id] M TS movie-form.tsx M X
src > components > TS movie-form.tsx > ...
  You, 5 minutes ago | 1 author (You)
1  'use client'
2
3  import { zodResolver } from '@hookform/resolvers/zod'
4  import * as z from 'zod'
```



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 options**
 - [Waku](#) from Daishi Kato
 - [React Server Components Demo](#) from the React team

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 themselves
- 📌 **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 of Server Side Rendering

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



Updates and caching

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

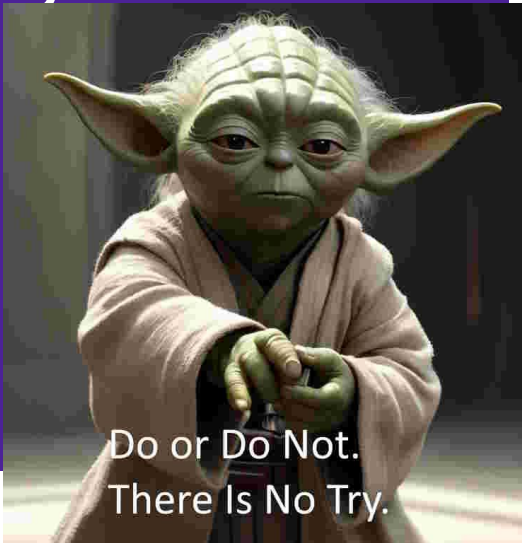
```
TS movie-form.tsx M TS page.tsx .../[id] M X TS page.tsx .../movies M
src > app > movies > [id] > TS page.tsx > ...
6   type Props = {
7     params: {
8       id: string
9     }
10  }
11
12  export const dynamic = 'force-dynamic'
13
14  const MoviePage: FC<Props> = async ({ params: { id } }) => {
15    async function fetchMovie() {
16      const rsp = await fetch(`http://localhost:3000/api/movies/${id}`)
17      const movie = await rsp.json()
18      return movie as Movie
19    }

```

movie- form.tsx

```
TS movie-form.tsx M X TS page.tsx ...\[id] M TS page.tsx ...\movies M
src > components > TS movie-form.tsx > ...
42 export const MovieForm: FC<Props> = ({ initialMovie }) => {
43   const { toast } = useToast()
44   const router = useRouter()
45
46   const onSubmit = async (movie: Movie) => {
47     try {
48       await saveMovie(movie)
49
50       router.refresh()
51
52       toast({
53         title: 'Success',
54         description: 'Movie updated',
55       })
56     } catch (error) {
57       toast({
58         title: 'Error',
59         description: 'Something went wrong',
60       })
61     }
62   }
63 }
```


movies /page.tsx



Do or Do Not.
There Is No Try.

```
TS movie-form.tsx M TS page.tsx ...[id] M TS page.tsx ...movies M X
src > app > movies > TS page.tsx > ...
5  type Props = {
6    searchParams: {
7      genre?: string
8    }
9  }
10
11  export const dynamic = 'force-dynamic'
12
13  export default async function MoviesPage({ searchParams: { genre } }: Props) {
14    async function fetchMovies() {
15      const url = genre ? `/api/movies?genre=${genre}` : '/api/movies'
16      const rsp = await fetch(`http://localhost:3000${url}`)
17      const movies = await rsp.json()
18      return movies as Movie[]
19    }
20  }
```

Querying the database from an RSC

Querying the database from an RSC

- Because an **RSC** only runs on the server we **can use server side code**
 - Query the DB using Prisma directly
 - It's safe to use secrets like database connection strings
- **Never executed in the browser**
 - Leads to smaller JavaScript bundle sizes

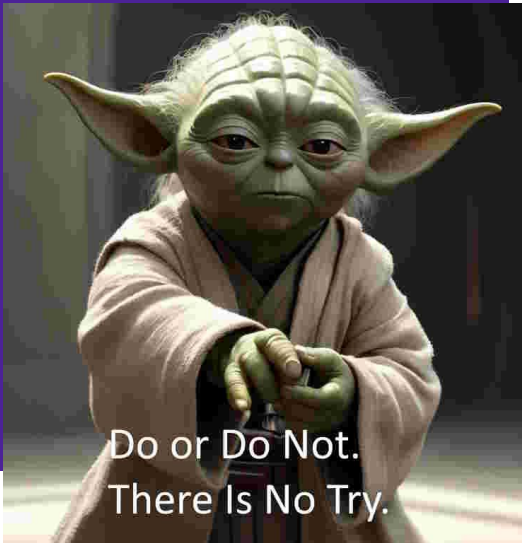
movies /page.tsx

```
TS page.tsx ...\movies M X TS page.tsx ...\[id] M TS route.ts M
src > app > movies > TS page.tsx > ...
14  async function getMovies(genreId: string | undefined) {
15    const orderBy: Prisma.MovieOrderByWithRelationInput = {
16      voteAverage: 'desc',
17    } as const
18
19    > if (genreId) { ...
30  } else {
31    const movies = await prisma.movie.findMany({
32      orderBy,
33    })
34
35    return movies
36  }
37 }
38 +
39 export default async function MoviesPage({ searchParams: { genre } }: Props) {
40   const movies = await getMovies(genre)
41
42   return (
43     <main className="flex-1 space-y-4 p-8 pt-6">
```

movies/[id]
/page.tsx

```
TS page.tsx ...\movies M  TS page.tsx ...\[id] M ×  TS route.ts M
src > app > movies > [id] > TS page.tsx > ...
12   export const dynamic = 'force-dynamic'
13
14   async function getMovie(id: string) {
15     const movie = await prisma.movie.findFirstOrThrow({
16       where: { id: +id },
17     })
18
19     return movie
20   }
21
22   const MoviePage: FC<Props> = async ({ params: { id } }) => {
23     const movie = await getMovie(id)
24
25     if (!movie) {
```

api/movies/[id] /route.ts



```
File Edit Selection View Go Run Terminal Help
TS page.tsx ...\movies M TS page.tsx ...\[id] M TS route.ts M X
src > app > api > movies > [id] > TS route.ts > ...
You, 4 days ago | 1 author (You)
1 import { saveMovie } from '@server/save-movie'
2 import { Movie } from '@prisma/client'
3 import { NextRequest, NextResponse } from 'next/server'
4
5 export async function PUT(request: NextRequest) {
6   try {
7     const movie = (await request.json()) as Movie
8
9     await saveMovie(movie)
10
11     return new NextResponse(null, {
12       status: 204,
13     })
14   } catch (error) {
15     console.error(error)
16
17     return new NextResponse(null, {
18       status: 400,
19     })
20   }
21 }
```

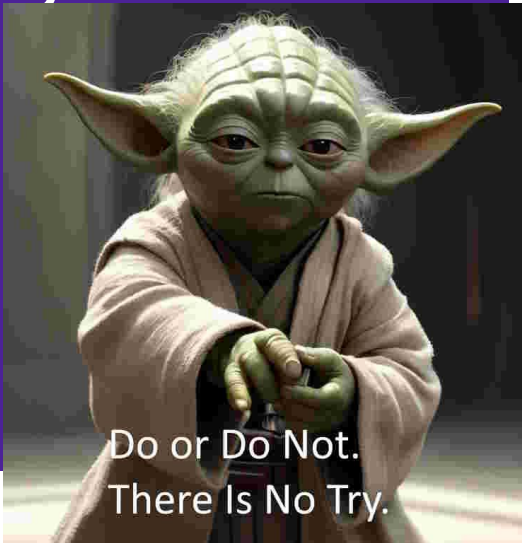


Prevent over fetching

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



```
TS page.tsx M x
src > app > movies > TS page.tsx > ...
13 type MovieForCard = ComponentProps<typeof MovieCard>['movie']
14
15 export const dynamic = 'force-dynamic'
16
17 async function getMovies(genreId: string | undefined): Promise<MovieForCard[]> {
18   const orderBy: Prisma.MovieOrderByWithRelationInput = {
19     voteAverage: 'desc',
20   } as const
21
22   const select = {
23     id: true,
24     title: true,
25     overview: true,
26     backdropPath: true,
27     voteAverage: true,
28     voteCount: true,
29   } satisfies Prisma.MovieSelect
30
31+   if (genreId) {
32     const genre = await prisma.genre.findFirst({
33       where: { id: +genreId },
34       include: {
35         movies: {
36           orderBy,
37           select,
38         },
39       },
40     })
41   }
42 }
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS AZURE GITLENS COMMENTS

```
- wait compiling...
- event compiled client and server successfully in 319 ms (736 modules)
```

REPOSITORIES
COMMITTS
BRANCHES
REMOVES
STASHES
TAGS
WORKTREES

Break time





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

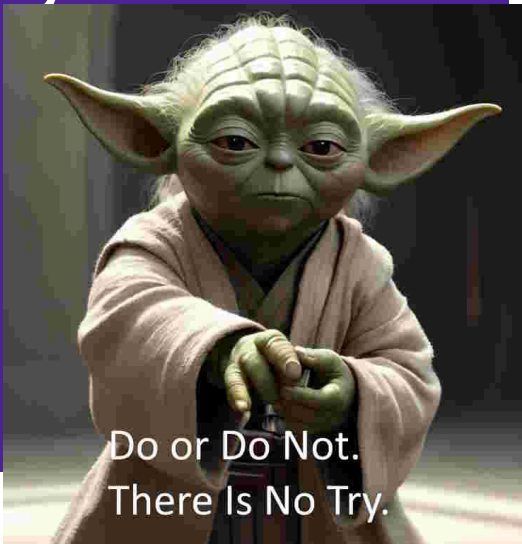
movies /loading.tsx

```
TS page.tsx ...\movies M TS loading.tsx U X TS page.tsx ...[id] M
src > app > movies > TS loading.tsx > ...
1 import { RotateCw } from 'lucide-react'
2
3 export default function Loading() {
4   return (
5     <div
6       role="status"
7       aria-label="Loading"
8       className="absolute left-1/2 top-2/4 -translate-x-1/2 -translate-y-1/2"
9     >
10       <RotateCw className="animate-spin text-foreground/40" size="5rem" />
11     </div>
12   )
13 }
```

movies /page.tsx

```
TS page.tsx ...\movies M X TS loading.tsx ...\movies U TS page.tsx ...\[id] M TS loading.tsx ...\[id] U
src > app > movies > TS page.tsx > ...
18  async function getMovies(genreId: string | undefined): Promise<MovieForCard[]> {
19    const orderBy: Prisma.MovieOrderByWithRelationInput = {
20      voteAverage: 'desc',
21    } as const
22
23    const select = {
24      id: true,
25      title: true,
26      overview: true,
27      backdropPath: true,
28+   voteAverage: true,
29      voteCount: true,
30    } satisfies Prisma.MovieSelect
31
32    await sleep(1_000)
33
```

movies/[id]
/page.tsx



Do or Do Not.
There Is No Try.

```
TS page.tsx ...\movies M    TS loading.tsx ...\movies U    TS page.tsx ...\[id] M X    TS loading.tsx ...\[id] U
src > app > movies > [id] > TS page.tsx > ...
15   async function getMovie(id: string) {
16       const movie = await prisma.movie.findFirstOrThrow({
17           where: { id: +id },
18       })
19
20       await sleep(1_000)
21
22       return movie
23   }
```

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

```
{} streaming.json 1.0 X
{} streaming.json
1 2:HL["/_next/static/css/app/layout.css?v=1695461372573",{ "as": "style" }]
2 0:["$@1",["development",[[["",{ "children": ["movies",{ "children": ["id", "238", "d"],{ "children": ["__PAGE__", {]}
3 5:I{"id": "(app-pages-browser)"/./src/components/shopping-cart.tsx", "chunks": ["app/layout:static/chunks/app/lay
4 6:I{"id": "(app-pages-browser)"/./src/components/main-nav.tsx", "chunks": ["app/layout:static/chunks/app/layout.
5 8:I{"id": "(app-pages-browser)"/./node_modules/next/dist/client/components/layout-router.js", "chunks": ["app-pag
6 9:I{"id": "(app-pages-browser)"/./node_modules/next/dist/client/components/render-from-template-context.js", "ch
7 c:I{"id": "(app-pages-browser)"/./src/components/ui/toaster.tsx", "chunks": ["app/layout:static/chunks/app/layou
8 1:"$undefined"
9 3:[null,["$","html",null,{ "lang": "en", "children": ["$","body",null,{ "className": "min-h-screen bg-background ar
10 4:[["$","meta","0",{ "charSet": "utf-8"}],["$","title","1",{ "children": "TS Congress"}],["$","meta","2",{ "name":
11 d:I{"id": "(app-pages-browser)"/./src/components/movie-form.tsx", "chunks": ["app/movies/[id]/page:static/chunks/
12 a:null
13 e:{ "id": "8ee0c4224708db417bfe9cefca1638c119b06524", "bound": null }
14 b:["$","main",null,{ "className": "flex-1 space-y-4 p-8 pt-6", "children": ["$","h2",null,{ "className": "text-3x
15+ f:I{"id": "(app-pages-browser)"/./src/components/genre-selector.tsx", "chunks": ["app/layout:static/chunks/app/la
16 7:["$","$Lf",null,{ "genres": [{ "id": 28, "name": "Action"}, { "id": 12, "name": "Adventure"}, { "id": 16, "name": "Animati
```



Site layout as an RSC

Site layout as an RSC

- A **layout.tsx** is typically a **React Server Component**
 - But can be a client component if required
- **Render server and/or client components** as needed

layout.tsx

```
TS layout.tsx M X TS main-nav.tsx M TS shopping-cart.tsx M TS genre-selector.tsx M
src > app > TS layout.tsx > ...
1  import './globals.css'
2
3  import type { PropsWithChildren } from 'react'
4  import type { Metadata } from 'next'
5  import { Inter } from 'next/font/google'
```

main-nav.tsx

```
TS layout.tsx M TS main-nav.tsx M X TS shopping-cart.tsx M TS genre-selector.tsx M
src > components > TS main-nav.tsx > ...
1  | 'use client'
2  |
3  | import Link from 'next/link'
4  | import { usePathname, useSearchParams } from 'next/navigation'
5  |
6  | import { cn } from '@lib/utils'
7  | import { Button } from '@components/ui/button'
8  | import { useShoppingCart } from '../shopping-cart'
9  | import { GenreSelector } from '../genre-selector'
10 |
11 | export function MainNav() {
12 |   const { itemCount, checkout } = useShoppingCart()
13 |   const pathname = usePathname()
14 |   const searchParams = useSearchParams()
15 |   const hasGenreParam = searchParams?.has('genre')
```

shopping-cart.tsx

```
TS layout.tsx M TS main-nav.tsx M TS shopping-cart.tsx M X TS genre-selector.tsx M
src > components > TS shopping-cart.tsx > ...
1  'use client'
2
3  import {
4    ComponentProps,
5    PropsWithChildren,
6    createContext,
7    useContext,
8    useState,
9  } from 'react'
10
11 import { CheckoutDialog } from '@components/checkout-dialog'
12
13 type ShoppingCartMovies = ComponentProps<typeof CheckoutDialog>['movies']
14 type ShoppingCartMovie = ShoppingCartMovies[0]
15 +
16 const ShoppingCartContext = createContext({
```

genre-selector.tsx



```
TS layout.tsx M TS main-nav.tsx M TS shopping-cart.tsx M TS genre-selector.tsx M X
src > components > TS genre-selector.tsx > ...
1  'use client'
2
3  import { useEffect, useState } from 'react'
4  import { Check, ChevronsUpDown } from 'lucide-react'
5  import { useRouter, useSearchParams } from 'next/navigation'
6  import { Genre } from '@prisma/client'
7
8  import { cn } from '@lib/utils'
9  import { Button } from '@components/ui/button'
10 import { Command, CommandGroup, CommandItem } from '@components/ui/command'
11 import {
12   Popover,
13   PopoverContent,
14   PopoverTrigger,
15 } from '@components/ui/popover'
16
17 export function GenreSelector() {
```




What is a server component?

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

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 x TS parent-component.tsx M TS page.tsx M
src > app > server-or-client > TS child-component.tsx > ...
1  import 'server-only'
2
3  import { prisma } from '@lib/db'
4
5  export async function ChildComponent() {
6    console.log('Rendering Child Component')
7    const movie = await prisma.movie.findFirstOrThrow()
8
9    return (
10     <main className="bg-red-400 p-12">
11       <h2 className="my-6 text-4xl font-bold">Child Component</h2>
12       <p>{movie.title}</p>
13     </main>
14   )
15 }
```

parent- component.tsx

```
TS child-component.tsx M TS parent-component.tsx M X TS page.tsx M
src > app > server-or-client > TS parent-component.tsx > ...
1 'use client'
2
3 import { PropsWithChildren } from 'react'
4
5 export function ParentComponent({ children }: PropsWithChildren) {
6   console.log('Rendering Parent Component')
7
8   return (
9     <main className="bg-green-400 p-12">
10      <h2
11        className="my-6 text-4xl font-bold"
12        onClick={() => console.log('Click')}
13      >
14        Parent Component
15      </h2>
16      {children}
17    </main>
18  )
19 }
```

server-or-client /page.tsx



```
TS child-component.tsx M TS parent-component.tsx M TS page.tsx M X
src > app > server-or-client > TS page.tsx > ...
1 import { ChildComponent } from './child-component'
2 import { ParentComponent } from './parent-component'
3
4 export default function ServerOrClientPage() {
5   console.log('Rendering Server Or Client Page')
6
7   return (
8     <main className="bg-blue-400 p-12">
9       <h1 className="my-6 text-4xl font-bold">Server Or Client Page</h1>
10      <ParentComponent>
11        <ChildComponent />
12      </ParentComponent>
13    </main>
14  )
15 }
```



Loading the genres on the server

Loading the genres on the server

- **Splitting the GenreSelector** in a client and a server component
 - Client component for interactivity
 - Server component for data loading
- The **MainNav component** still needs to be a client component
 - The **GenreSelector/Loader** can be injected as a prop

genre-selector.tsx

```
TS genre-selector.tsx M X TS genre-loader.tsx U TS site-header.tsx M TS main-nav.tsx M
src > components > TS genre-selector.tsx > ...
17  type Props = {
18    genres: Genre[]
19  }
20
21  export function GenreSelector({ genres }: Props) {
22    const [open, setOpen] = useState(false)
23    const searchParams = useSearchParams()
24    const selectedGenre = searchParams?.get('genre') ?? ''
25    const { push } = useRouter()
26    const items = genres.map((genre) => ({
27      value: genre.id.toString(),
28      label: genre.name,
29    }))
30
31    return (
32      <Popover open={open} onOpenChange={setOpen}>
```

genre-loader.tsx

```
TS genre-selector.tsx M TS genre-loader.tsx U X TS site-header.tsx M TS main-nav.tsx M
src > components > TS genre-loader.tsx > ...
1  import 'server-only'
2
3  import { prisma } from '@lib/db'
4  import { GenreSelector } from '../genre-selector'
5  import { sleep } from '@lib/utils'
6
7  export async function GenreLoader() {
8    // await sleep(2500)
9
10   const genres = await prisma.genre.findMany({
11     orderBy: {
12       name: 'asc',
13     },
14   })
15
16   return <GenreSelector genres={genres} />
17 }
```

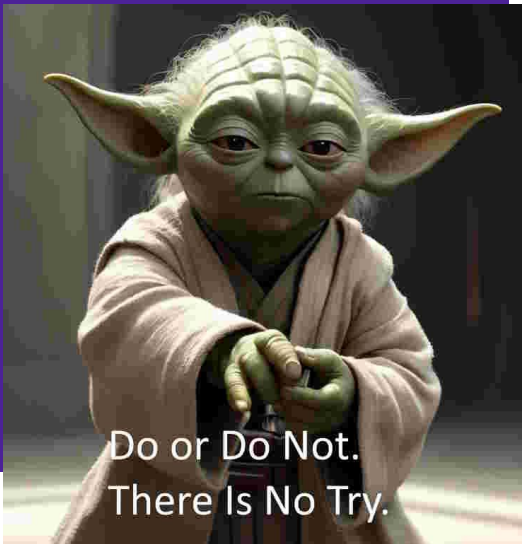
site-header.tsx

```
TS genre-selector.tsx  TS genre-loader.tsx  TS site-header.tsx X  TS main-nav.tsx
src > components > TS site-header.tsx > ...
1  import { MainNav } from '@components/main-nav'
2  import { GenreLoader } from './genre-loader'
3
4  export function SiteHeader() {
5    return (
6      <header className="sticky top-0 z-40 w-full border-b bg-background">
7        <div className="container flex h-16">
8          <MainNav genreSelector={<GenreLoader />} />
9        </div>
10     </header>
11   )
12 }
```

main-nav.tsx

```
TS genre-selector.tsx M TS genre-loader.tsx U TS site-header.tsx M TS main-nav.tsx M X
src > components > TS main-nav.tsx > ...
1  'use client'
2
3  import Link from 'next/link'
4  import { usePathname, useSearchParams } from 'next/navigation'
5  import { ReactNode, Suspense } from 'react'
6  import { RotateCw } from 'lucide-react'
7
8  import { cn } from '@lib/utils'
9  import { Button } from '@components/ui/button'
10 import { useShoppingCart } from '../shopping-cart'
11
12 type Props = {
13   genreSelector: ReactNode
14 }
15
16 export function MainNav({ genreSelector }: Props) {
```

main-nav.tsx



```
TS genre-selector.tsx M TS genre-loader.tsx U TS site-header.tsx M TS main-nav.tsx M X
src > components > TS main-nav.tsx > ...
47      <Suspense
48        fallback={
49          <Button
50            variant="outline"
51            className="w-[200px] justify-between text-foreground/60"
52            disabled
53          >
54            Movies by genre ...
55            <ChevronsUpDown className="ml-2 h-4 w-4 shrink-0 opacity-50" />
56          </Button>
57        }
58      >
59        {genreSelector}
60      </Suspense>
```

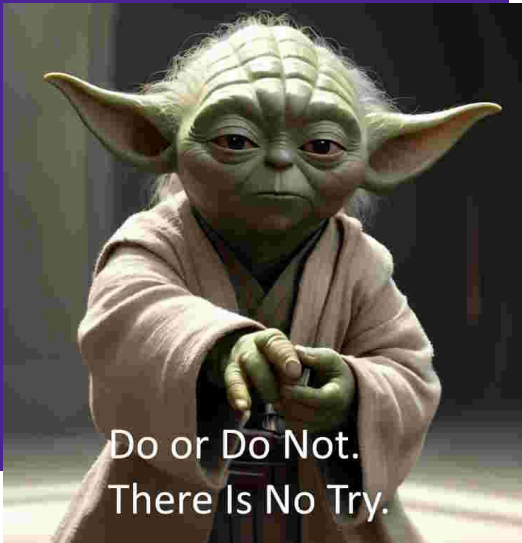
Calling Server Actions

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 😊

genre-form.tsx



Do or Do Not.
There Is No Try.

```
TS genre-form.tsx M X
src > components > TS genre-form.tsx > ...

22 export function GenreForm({ genre }: Props) {
23   const onSubmit = async (formData: FormData) => {
24     'use server'
25
26     const genre: Genre = {
27       id: +(formData.get('id') as string),
28       name: formData.get('name') as string,
29     }
30
31     await saveGenre(genre)
32
33     redirect('/genres')
34   }
35
36   return (
37     <form action={onSubmit} className="mx-auto w-1/2">
38       <Card>
39         <CardHeader>
40           <CardTitle>Edit Movie Genre</CardTitle>
```

Calling Server Actions

Directly

Calling Server Actions

- Can also be **called as a normal asynchronous function**
 - The network request is handled for you
- Optionally use **the useTransition() hook**
 - For feedback while the server action is executing

checkout-shopping-cart.ts

```
JS next.config.js M TS checkout-shopping-cart.ts M X TS checkout-dialog.tsx M
src > server > TS checkout-shopping-cart.ts > ...
1  'use server'
2
3  import { Movie } from '@prisma/client'
4
5  type ShoppingCartMovie = Pick<Movie, 'id' | 'title'>
6
7  type Cart = {
8    account: string
9    customerName: string
10   movies: ShoppingCartMovie[]
11 }
12
13 export async function checkoutShoppingCart({
14   account,
15   customerName,
16   movies,
17 }: Cart) {
```

checkout-dialog.tsx



```
JS next.config.js M    TS checkout-shopping-cart.ts M    TS checkout-dialog.tsx M X
src > components > TS checkout-dialog.tsx > ...
53     const onSubmit = async (data: CheckoutForm) => {
54       try {
55         await checkoutShoppingCart({
56           account: data.account,
57           customerName: data.name,
58           movies,
59         })
60       toast({
61         title: 'Success',
62         description: 'Checkout completed',
63       })
```



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 use 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

