Fullstack Development

Stack Overflow 2024 Developer Survey

- Developer types
- Databases
- Web framework
- Tools

Fullstack Landscape

> Diagram

Inspired from this VDO.

Case Study

> How to over-engineer "todo" apps

5 Ways to make Todo apps

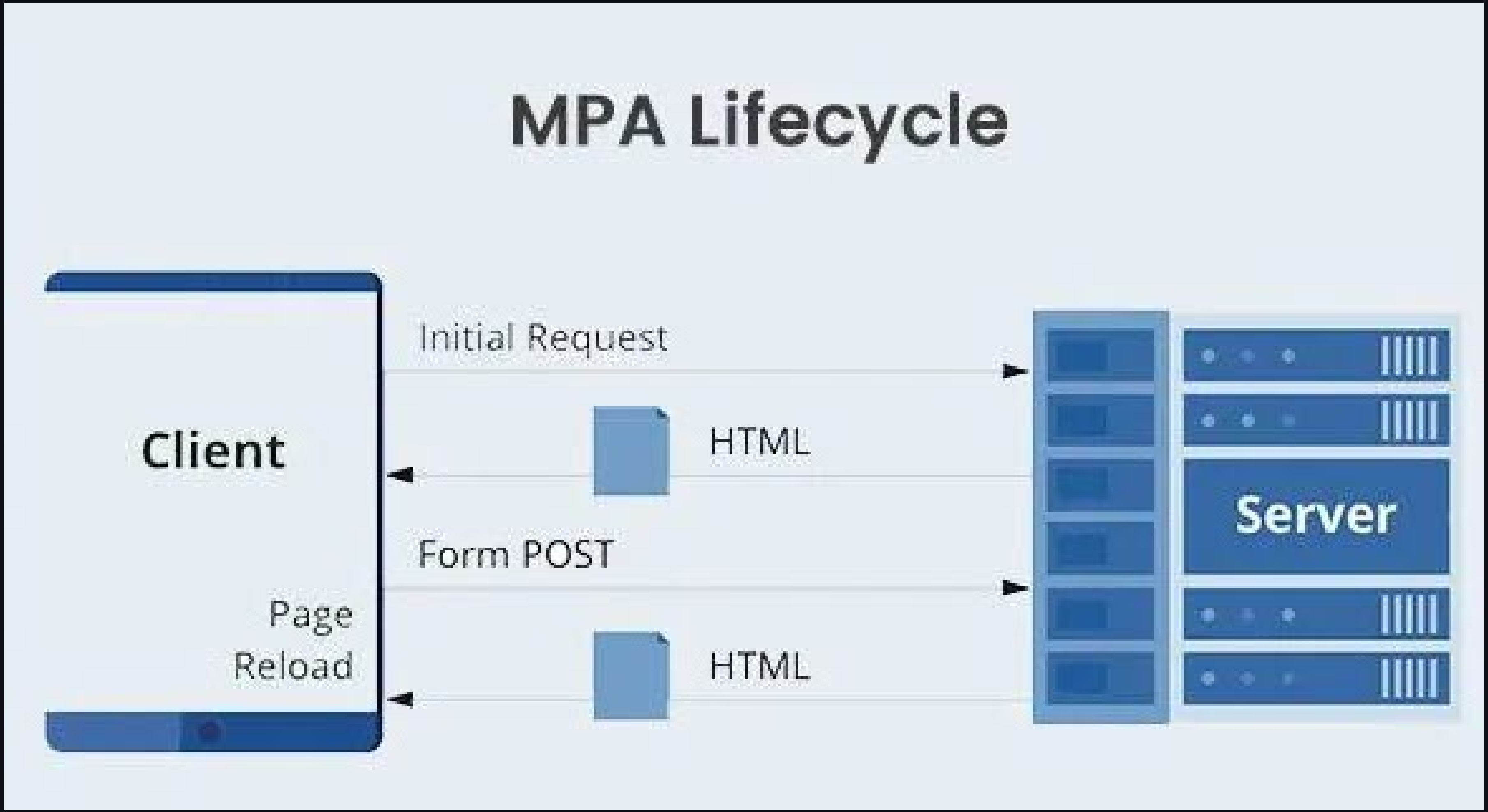
- Multi-Page Applications (MPA)
- Single-Page Applications (SPA)
- React Server Components (RSC)
- RSC + Client Components (React's New Architecture)
- HTMX

Round 1

Multi-Page Application (MPA) vs Single-Page Application (SPA)

Multi-page application

- Loads a new page every time you perform an action.
- Traditional web applications.
- Use server-side technologies
    - PHP, Ruby on Rails, ASP.NET, Java, and Node JS.
- Can include JavaScript (`script`) for client-side interactivity


MPA Lifecycle
Client
Initial Request
HTML
Form POST
Page Reload
HTML
Server

Todo app (MPA)

- `Express JS` + `Pug` (renderer)

Get started

- `git clone https://github.com/fullstack-68/landscape-mpa.git mpa`
- `cd mpa`
- `pnpm install`
- `pnpm run build`
- `pnpm run start`

Endpoint

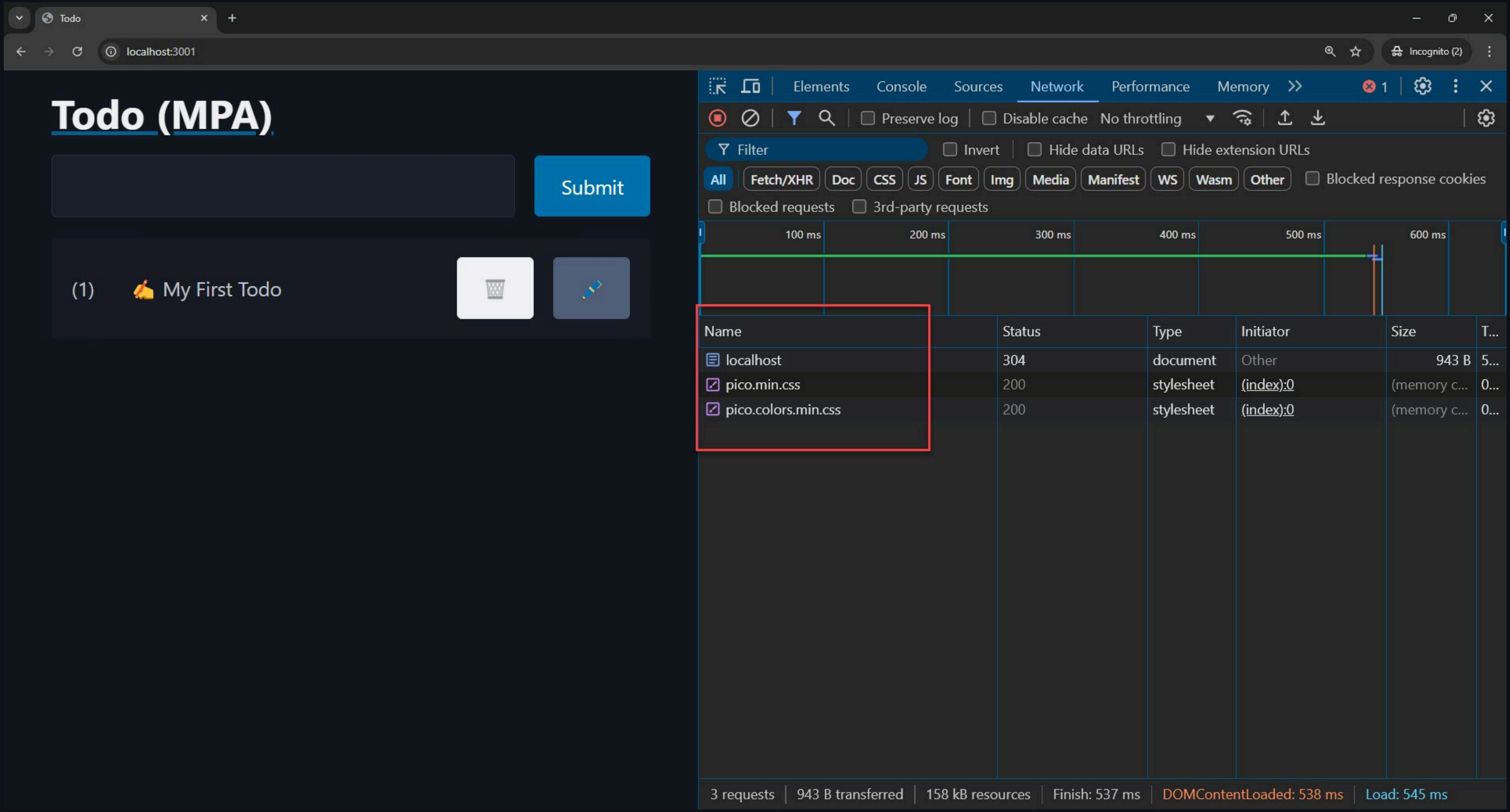
./src/index.ts

```
app.get("/", async (req, res) => {
  const message = req.query?.message ?? "";
  const todos = await getTodos();
  // Output HTML
  res.render("pages/index", {
    todos,
    message,
    mode: "ADD",
    curTodo: { id: "", todoText: "" },
  });
});
```

Renderer

`./view/pages/index.pug`

```pug
body
    main(class="container")
        a(href="/")
            h1 Todo (MPA)
        div(id="todoform")
            include ../components/inputform.pug
        div(id="todolist")
            include ../components/todolist.pug
```



Name	Status	Type	Initiator	Size	T...
localhost	304	document	Other	943 B	5...
pico.min.css	200	stylesheet	(index):0	(memory c...	0...
pico.colors.min.css	200	stylesheet	(index):0	(memory c...	0...

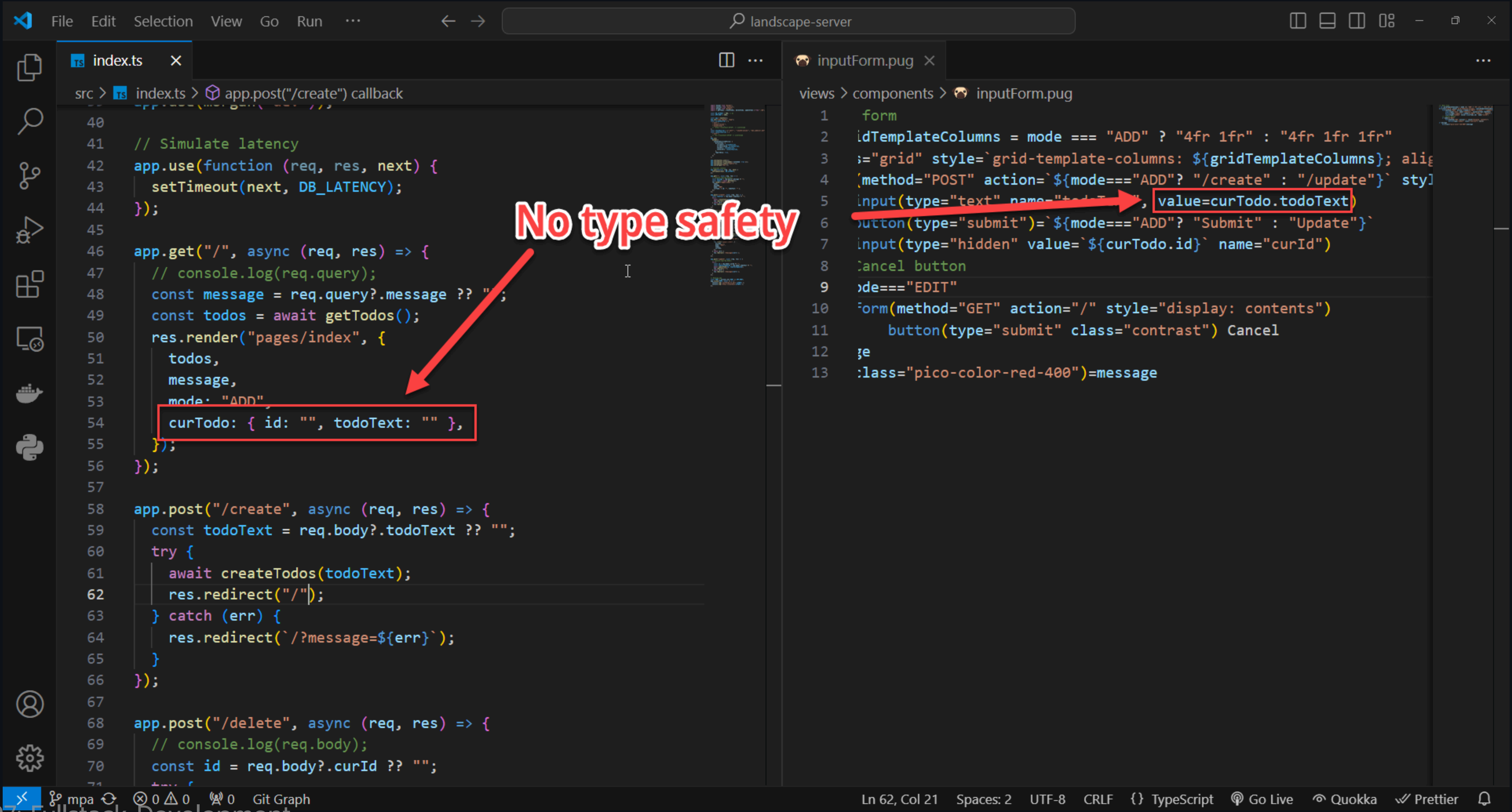
3 requests | 943 B transferred | 158 kB resources | Finish: 537 ms | DOMContentLoaded: 538 ms | Load: 545 ms

Use incognito mode to avoid loading chrome extensions.

Note

- Every button is wrapped in a separate form
  - Need to trigger different endpoints.
- Need to use `input(type="hidden)` to encode additional information.

```
form(action="/delete" method="POST" style="display: contents")
  input(type="hidden" value=`${todo.id}` name="curId")
  button(type="submit" class="contrast" style="margin-bottom: 0") 🗑️
form(action="/edit" method="POST" style="display: contents")
  input(type="hidden" value=`${todo.id}` name="curId")
  button(type="submit" class="secondary" style="margin-bottom: 0") 🖊️
```

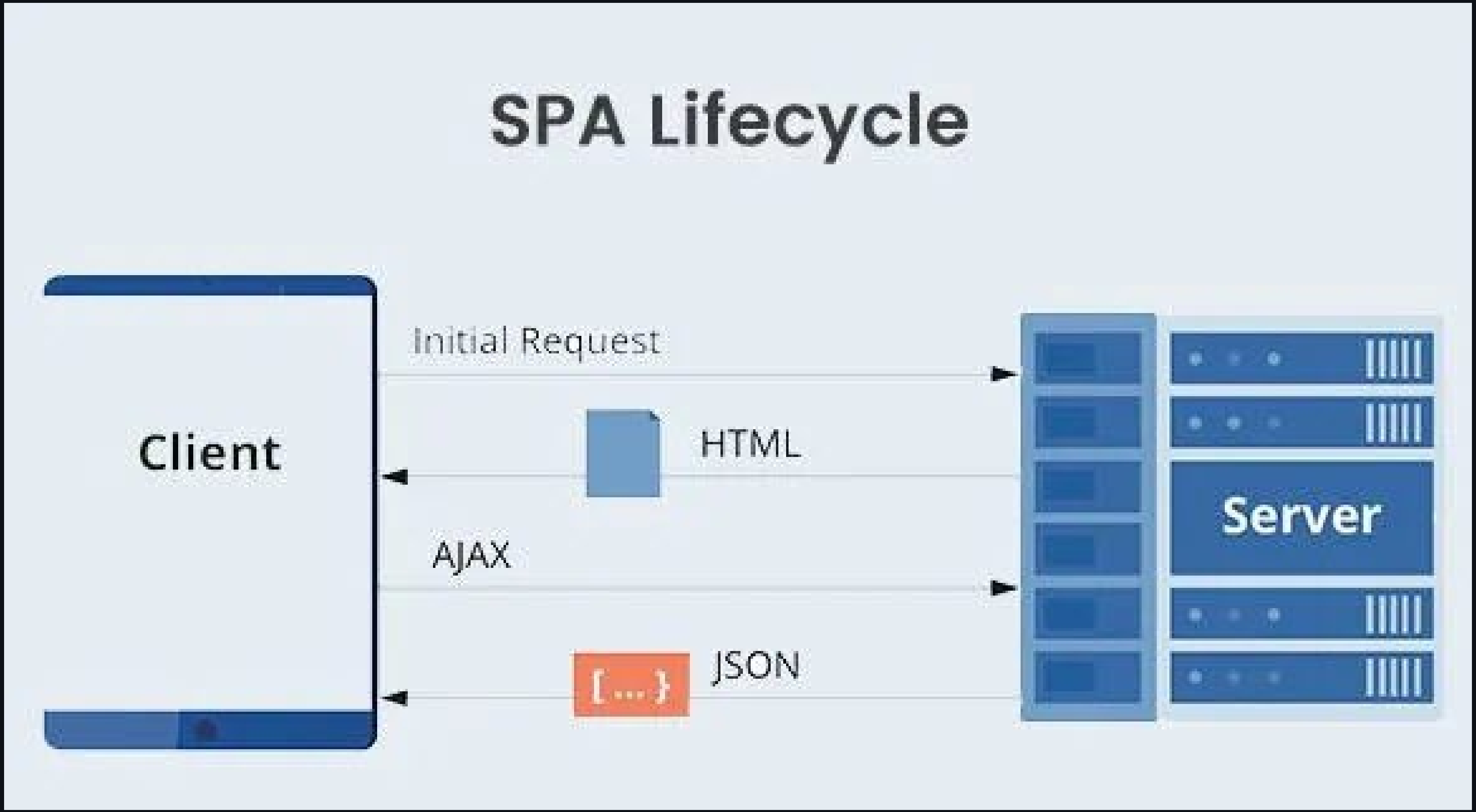

File Edit Selection View Go Run ···
landscape-server
index.ts
src > index.ts > app.post("/create") callback
inputForm.pug
views > components > inputForm.pug
No type safety
mpa 0 0 0 Git Graph
Ln 62, Col 21 Spaces: 2 UTF-8 CRLF {} TypeScript Go Live Quokka Prettier

```typescript

// Simulate latency
app.use(function (req, res, next) {
  setTimeout(next, DB_LATENCY);
});

app.get("/", async (req, res) => {
  // console.log(req.query);
  const message = req.query?.message ?? "";
  const todos = await getTodos();
  res.render("pages/index", {
    todos,
    message,
    mode: "ADD",
    curTodo: { id: "", todoText: "" },
  });
});

app.post("/create", async (req, res) => {
  const todoText = req.body?.todoText ?? "";
  try {
    await createTodos(todoText);
    res.redirect("/");
  } catch (err) {
    res.redirect(`/?message=${err}`);
  }
});

app.post("/delete", async (req, res) => {
  // console.log(req.body);
  const id = req.body?.curId ?? "";
  try {
```

```pug
 form
idTemplateColumns = mode === "ADD" ? "4fr 1fr" : "4fr 1fr 1fr"
s="grid" style=`grid-template-columns: ${gridTemplateColumns}; alig
(method="POST" action=`${mode==="ADD"? "/create" : "/update"}` sty]
input(type="text" name="todoText", value=curTodo.todoText)
button(type="submit")=`${mode==="ADD"? "Submit" : "Update"}`
input(type="hidden" value=`${curTodo.id}` name="curId")
Cancel button
ode==="EDIT"
form(method="GET" action="/" style="display: contents")
    button(type="submit" class="contrast") Cancel
ge
class="pico-color-red-400")=message
```

Single-page application

- Single-page application
  - Loads a single HTML page and dynamically updates the content as the user interacts with the app.
- Use frontend and backend frameworks separately.


SPA Lifecycle
Client
Initial Request
HTML
AJAX
{...}
JSON
Server

Todo app (SPA)

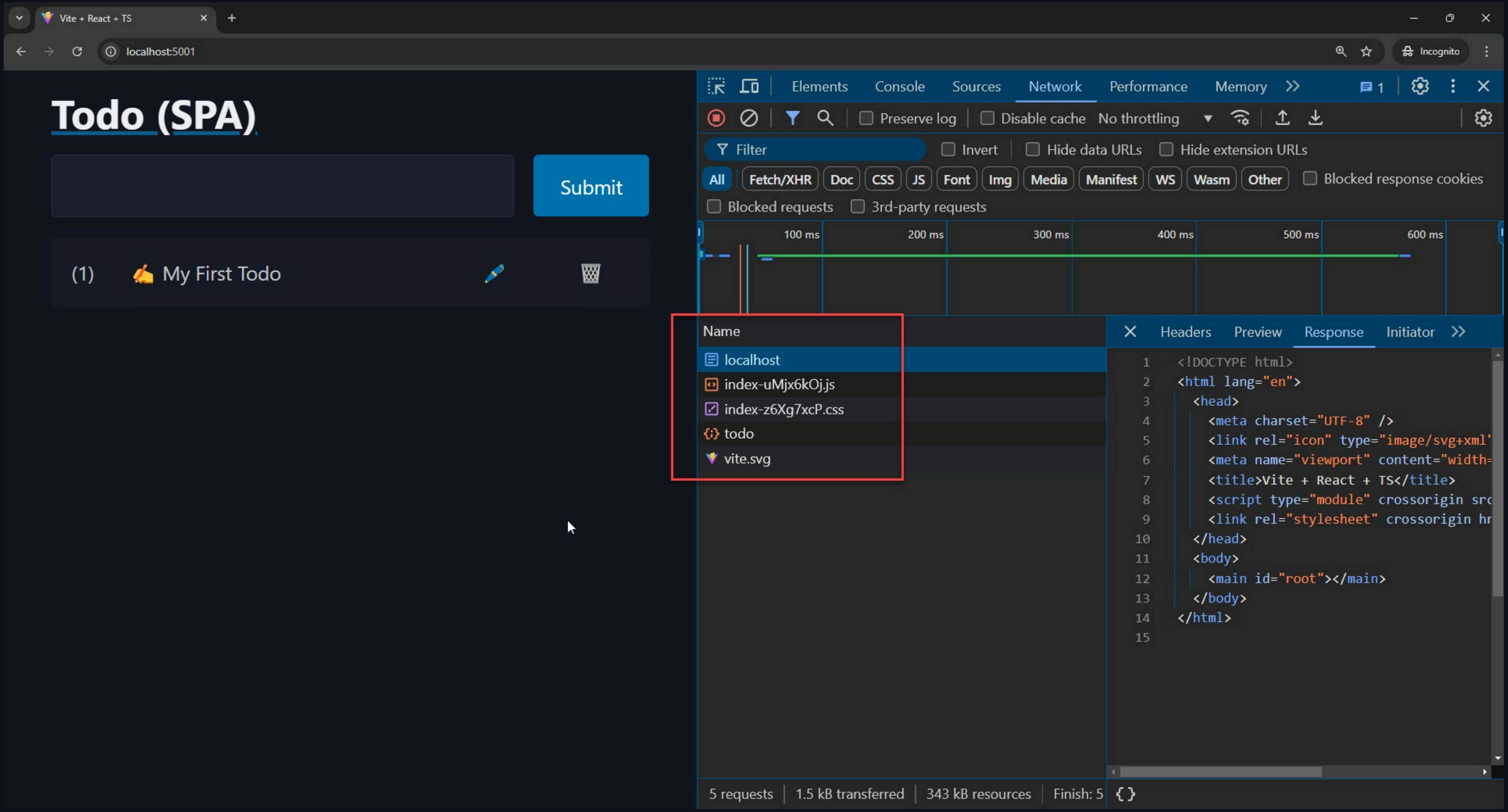
- `Express JS` + `React`

Get started

- `git clone https://github.com/fullstack-68/landscape-spa.git spa`
- Backend
  - `cd spa/backend`
  - `pnpm install`
  - `pnpm run build`
  - `pnpm run start`

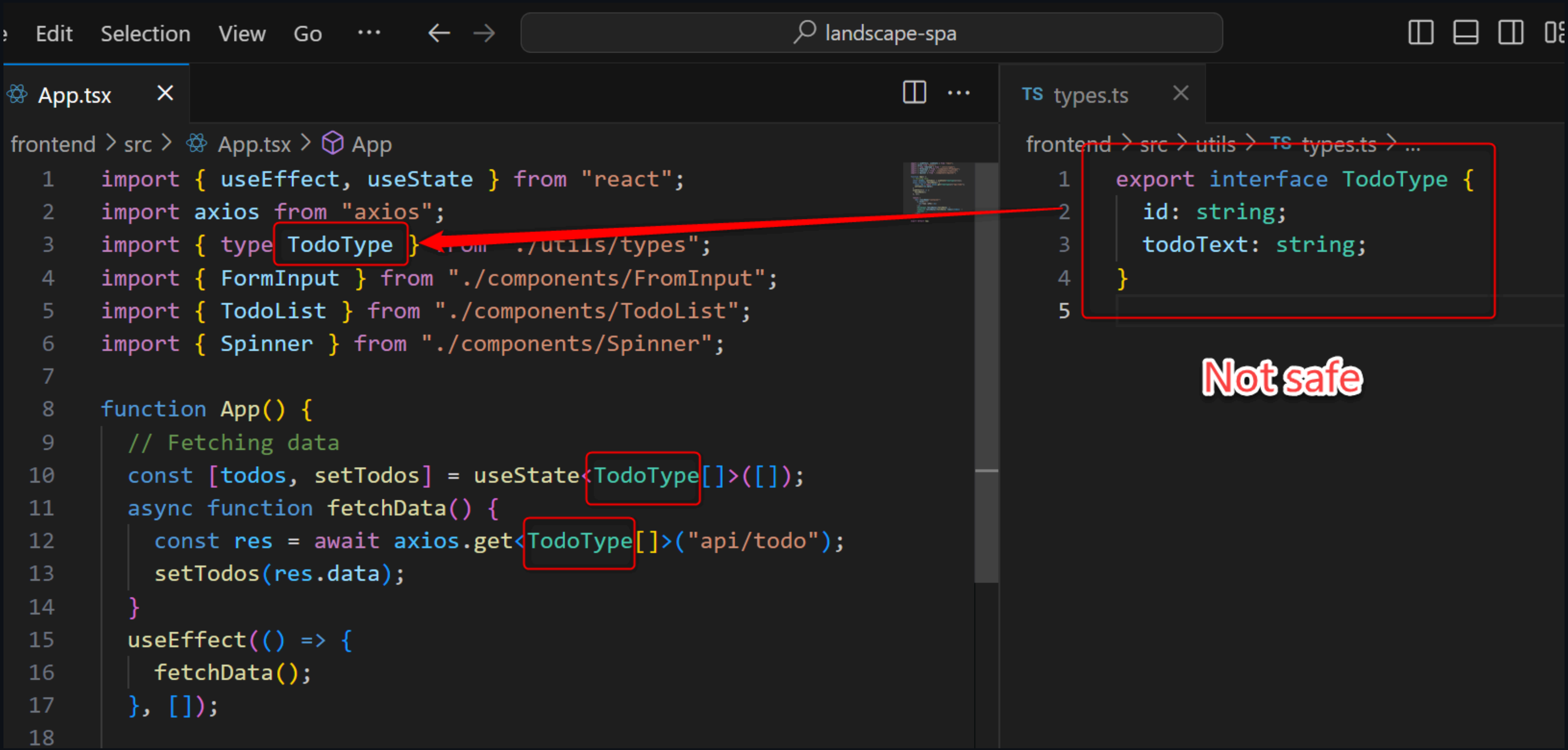
Get started (cont.)

- Frontend
    - `cd spa/frontend`
    - `pnpm install`
    - `pnpm run build`
    - `pnpm run preview`



Name
localhost
index-uMjx6kOj.js
index-z6Xg7xcP.css
todo
vite.svg

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml"
    <meta name="viewport" content="width=
    <title>Vite + React + TS</title>
    <script type="module" crossorigin src
    <link rel="stylesheet" crossorigin hr
  </head>
  <body>
    <main id="root"></main>
  </body>
</html>
```

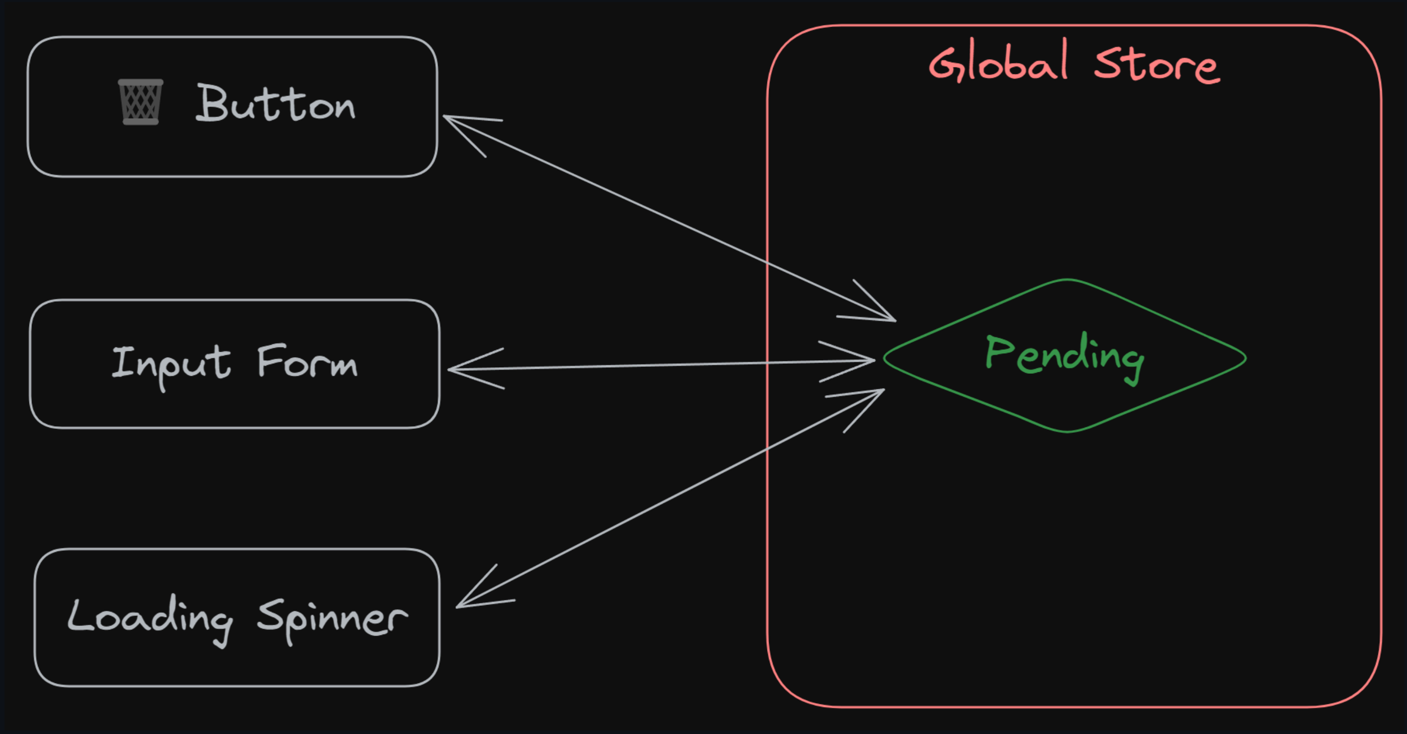

Edit
Selection
View
Go
landscape-spa
App.tsx
frontend > src > App.tsx > App
types.ts
frontend > src > utils > types.ts > ...
Not safe

```tsx
import { useEffect, useState } from "react";
import axios from "axios";
import { type TodoType } from "./utils/types";
import { FormInput } from "./components/FromInput";
import { TodoList } from "./components/TodoList";
import { Spinner } from "./components/Spinner";

function App() {
  // Fetching data
  const [todos, setTodos] = useState<TodoType[]>([]);
  async function fetchData() {
    const res = await axios.get<TodoType[]>("api/todo");
    setTodos(res.data);
  }
  useEffect(() => {
    fetchData();
  }, []);

```

```ts
export interface TodoType {
  id: string;
  todoText: string;
}

```

Global store

- `zustand`



App comparison (UX)

Item	MPA	SPA
No page-refresh	❌	✅
Spinner	❌	✅
Element disabling	❌	✅

App comparison (technical)

Item	MPA	SPA
Amount of `JS` loaded	✅	❌
HTML content (SEO)	✅	❌
State in URL	✅	❌

DX

Item	MPA	SPA
Number of frameworks	1 ✅	2 ❌
Complexity	Less ✅	More ❌
Lines of code	Less ✅	More ❌
Type Safety	Less ❌	More ✅
Hot reloading	Partial ❌	Full ✅

Amount of Codes

Dir	# Files	Total Lines
MPA		
`./src`	2	161
`./views`	3	42
SPA		
`./backend/src`	2	144
`./frontend/src`	10	😭 360

Total: MPA=203, SPA=504

Round 2

> Back to the server. (Next JS)

Server-Side Rendering (SSR)

- `SSR`
  - Generating the HTML for a web page on the server before sending it to the client's browser.
- `CSR` (Client-Side Rendering)
  - Browser loads a minimal HTML file and fetches and renders the content using JavaScript.
- See this explanation.

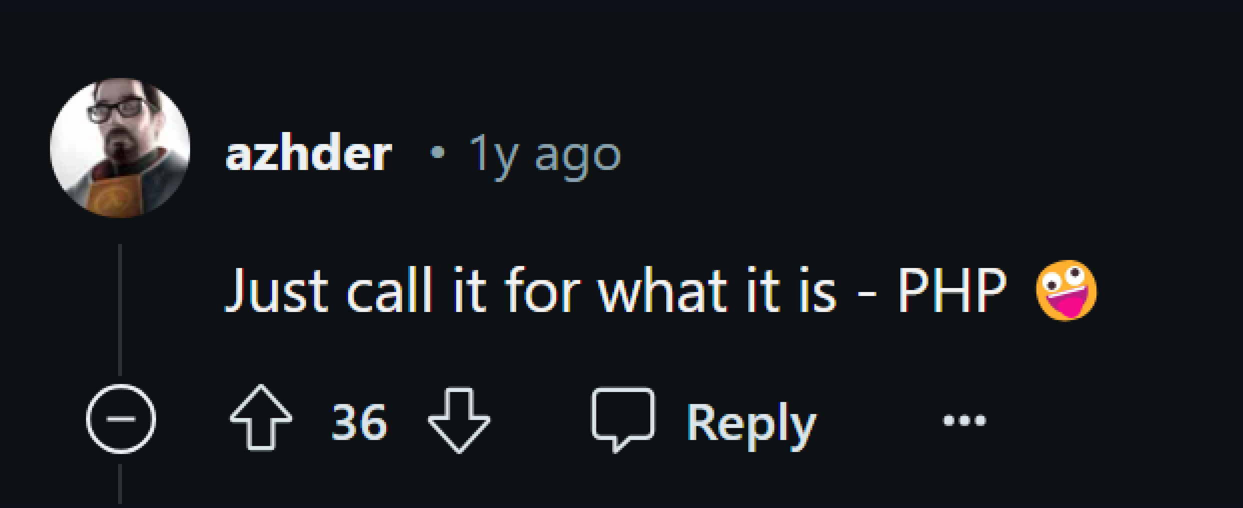
Next JS

- `V12`
  - "Full SSR" (*with DB query*) can only be done through top-level component (*page level*).
  - Use `getServerSideProps` function.
- `V13` (and above)
  - Full SSR can be done through a special components called
    - *React Server Components*.

Server component

- Run *exclusively* on the server.
- Generate static HTML.
  - No interactivity (event handlers)
- It's code isn't included in the JS bundle.
  - Never re-render.
  - Output is static without change in router level.
- No hook
  - `useState`, `useEffect` 😍

Server component


azhder • 1y ago

Just call it for what it is - PHP 🤪

Link

Client component

- The "standard" React components we're familiar with.
- Client Components render on *both* the client and the server.
  - *Still have SSR.*

	Render on server?	**Render on client?**
Server Component	✅	
Client Component	✅	✅

Why server component?

- First "official" way to run server-exclusive code in React.
- Performance
  - Server Components don't get included in our JS bundles.
  - Faster load time
  - Real use-case
- Less complications
  - Dependency arrays, stale closures, memoization, …
  - *(All of these are caused by things changing.)*

Missing piece

React

> *If RSC output is static, how can I mutate data then?*

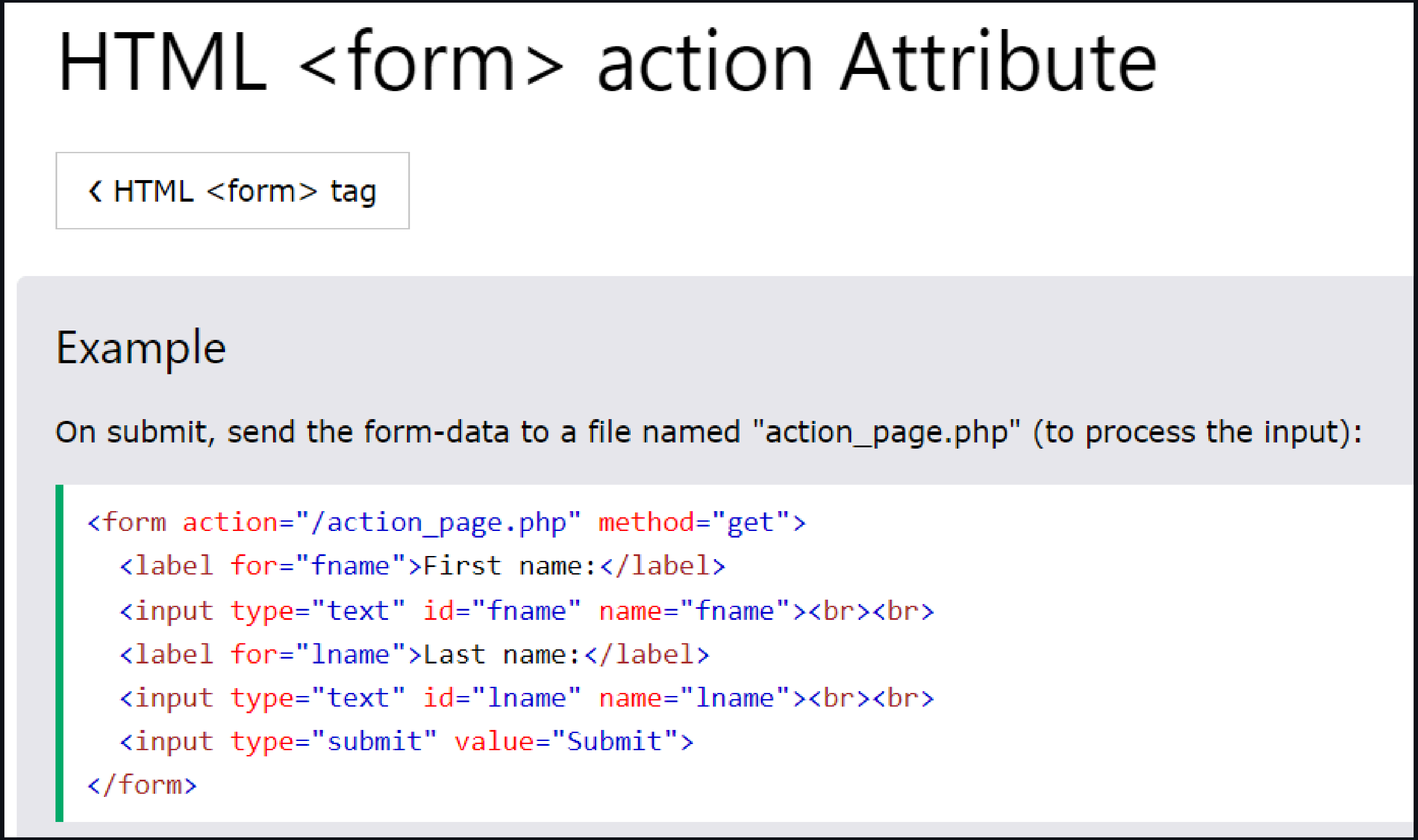
Missing piece

React

> *If RSC output is static, how can I mutate data then?*

PHP

> *I had solved this problem before you were born, kid.*


HTML <form> action Attribute

❮ HTML <form> tag

Example

On submit, send the form-data to a file named "action_page.php" (to process the input):

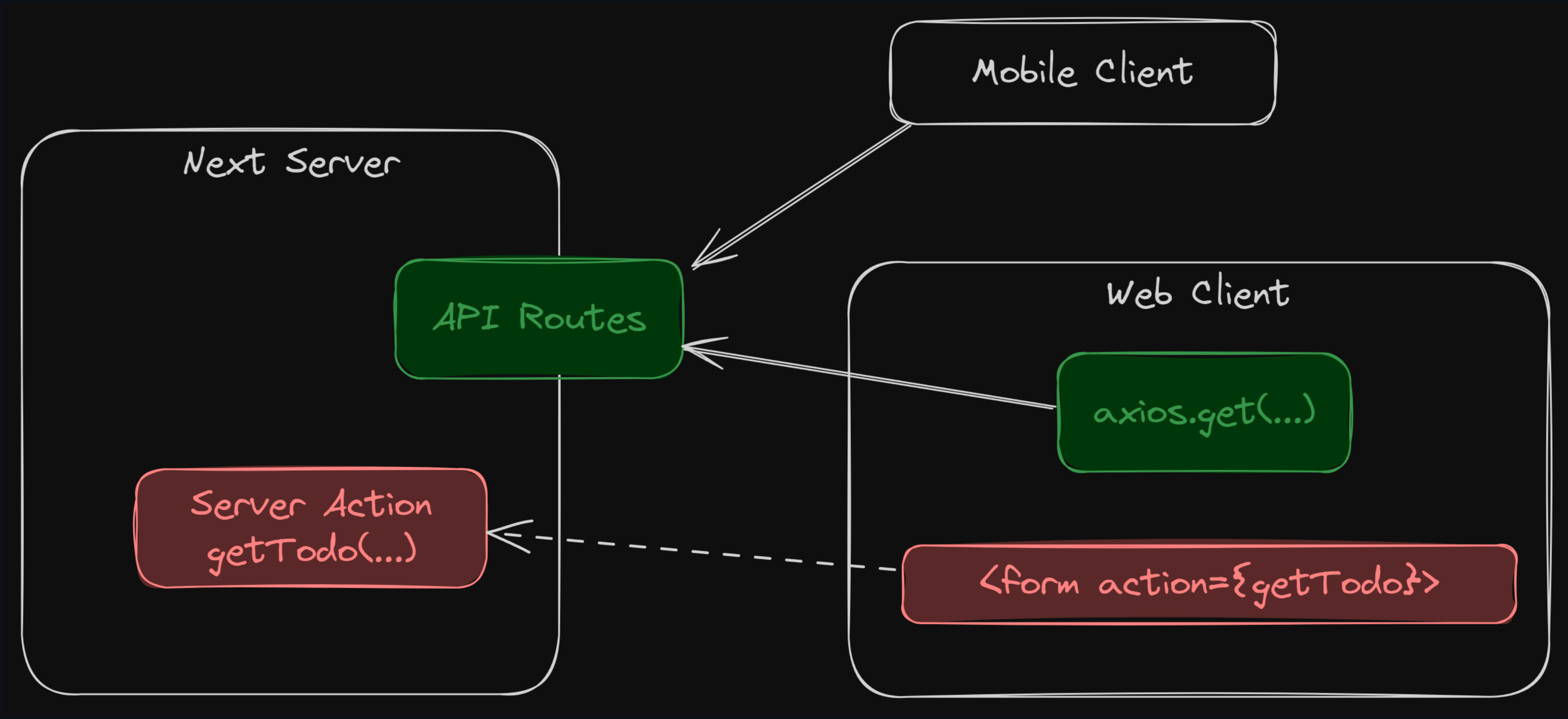
```
<form action="/action_page.php" method="get">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>
```

Link

React's new architecture

- Client component
- Server component
- Server action
  - Asynchronous functions that are executed on the server.
  - Alternative to API routes.

Server Action vs API Route



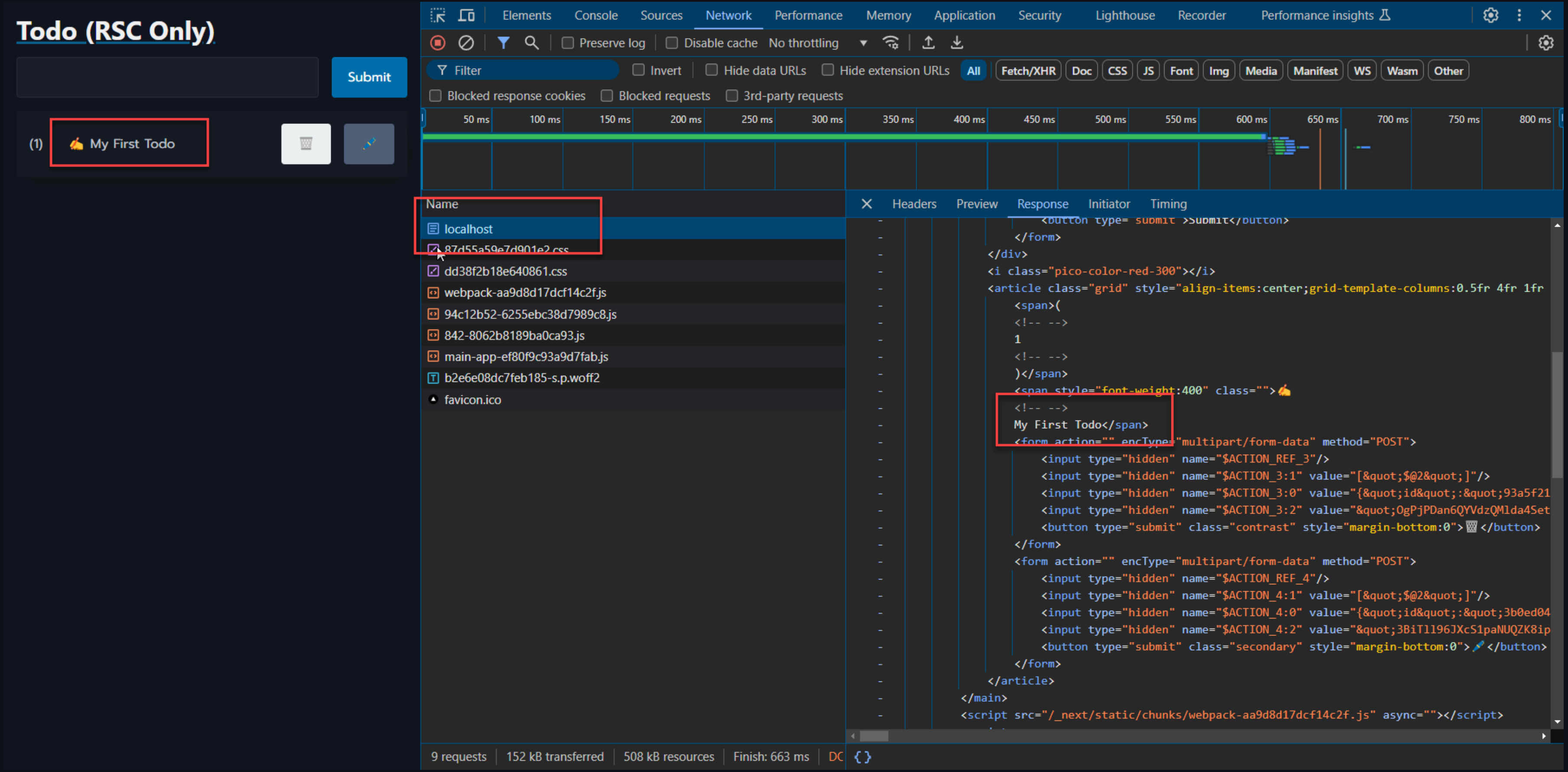
Form action

- No need to define new endpoints.
- Accept `form-data`
- Colocation
    - Type safe
    - Can "bind" data that is passed thought components' props. 🤯
- Can trigger RSC update without refreshing page. 👍

Todo App (RSC Only)

- `git clone https://github.com/fullstack-68/landscape-rsc-only.git rsc-only`
- `cd rsc-only`
- `pnpm install`
- `pnpm run build`
- `pnpm run start`

No blank HTML

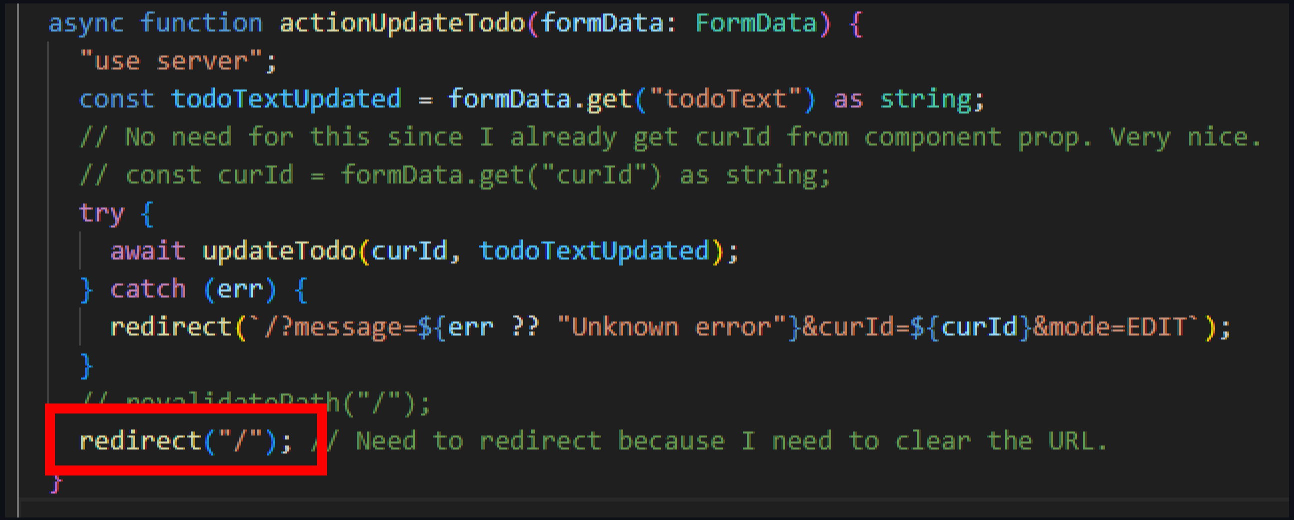

Todo (RSC Only)
Submit
(1)
✍ My First Todo
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
Hide extension URLs
All Fetch/XHR Doc CSS JS Font Img Media Manifest WS Wasm Other
Blocked response cookies
Blocked requests
3rd-party requests
Name
Headers Preview Response Initiator Timing

```
localhost
87d55a59e7d901e2.css
dd38f2b18e640861.css
webpack-aa9d8d17dcf14c2f.js
94c12b52-6255ebc38d7989c8.js
842-8062b8189ba0ca93.js
main-app-ef80f9c93a9d7fab.js
b2e6e08dc7feb185-s.p.woff2
favicon.ico
```

```
                <button type="submit">Submit</button>
            </form>
        </div>
        <i class="pico-color-red-300"></i>
        <article class="grid" style="align-items:center;grid-template-columns:0.5fr 4fr 1fr
            <span>(
            <!-- -->
            1
            <!-- -->
            )</span>
            <span style="font-weight:400" class="">✍
            <!-- -->
            My First Todo</span>
            <form action="" encType="multipart/form-data" method="POST">
                <input type="hidden" name="$ACTION_REF_3"/>
                <input type="hidden" name="$ACTION_3:1" value="[&quot;$@2&quot;]"/>
                <input type="hidden" name="$ACTION_3:0" value="{&quot;id&quot;:&quot;93a5f21
                <input type="hidden" name="$ACTION_3:2" value="&quot;OgPjPDan6QYVdzQMlda4Set
                <button type="submit" class="contrast" style="margin-bottom:0">🗑</button>
            </form>
            <form action="" encType="multipart/form-data" method="POST">
                <input type="hidden" name="$ACTION_REF_4"/>
                <input type="hidden" name="$ACTION_4:1" value="[&quot;$@2&quot;]"/>
                <input type="hidden" name="$ACTION_4:0" value="{&quot;id&quot;:&quot;3b0ed04
                <input type="hidden" name="$ACTION_4:2" value="&quot;3BiTl196JXcS1paNUQZK8ip
                <button type="submit" class="secondary" style="margin-bottom:0">🖊</button>
            </form>
        </article>
    </main>
    <script src="/_next/static/chunks/webpack-aa9d8d17dcf14c2f.js" async=""></script>
```

```
9 requests | 152 kB transferred | 508 kB resources | Finish: 663 ms | DC
```

Page refresh is back



```
async function actionUpdateTodo(formData: FormData) {
  "use server";
  const todoTextUpdated = formData.get("todoText") as string;
  // No need for this since I already get curId from component prop. Very nice.
  // const curId = formData.get("curId") as string;
  try {
    await updateTodo(curId, todoTextUpdated);
  } catch (err) {
    redirect(`/?message=${err ?? "Unknown error"}&curId=${curId}&mode=EDIT`);
  }
  // revalidatePath("/");
  redirect("/"); // Need to redirect because I need to clear the URL.
}
```

Server components

`./src/app/page.tsx`

```tsx
export default async function Home({ params, searchParams }: PageProps) {
  const todos = await getTodos();
  //...
  return <main className="container">...</main>;
}
```

- Async function
- Data fetching without `useEffect`.
  - It only runs once on the *server*. (Try `console.log`)
- Returns HTML to client.

Server Action

`./src/components/FormInput.tsx` *(Slightly modified)*

```tsx
export const FormInput: FC<Props> = async ({ message, mode, curId }) => {
  async function actionCreateTodo(formData: FormData) {
    "use server";
    const todoText = formData.get("todoText") as string; // Receive form-data
    await createTodos(todoText); // DB stuff
    redirect("/?message=&curId=&mode=ADD"); // Update content without refreshing page (Nice!)
  }

  return (
    <form action={actionCreateTodo} style={{ display: "contents" }}>
      <input type="hidden" name="curId" value={curId ?? ""} />
      <button type="submit">{mode === "ADD" ? "Submit" : "Update"}</button>
    </form>
  );
};
```

No need to create endpoint manually.

Automatic binding

`./src/components/TodoList.tsx`

- "Binding" `todo` in the server action
- No need to use `form-data`. Type safety!
- No need to include hidden `input` field.

Automatic binding

```
const ButtonDelete: FC<{ todo: Todo }> = ({ todo }) => {
  async function actionDeleteTodo(formData: FormData) {
    "use server";
    await deleteTodo(todo.id); //👈👈👈👈
    revalidatePath("/");
  }
  return (
    <form action={actionDeleteTodo}>
      <button type="submit">🗑️</button>
    </form>
  );
};
```

Side note

(for my future self)

- `revalidatePath`
  - Used when there is not change in url (params).
- `redirect`
  - Used when you need to change the url (change client states in the url).

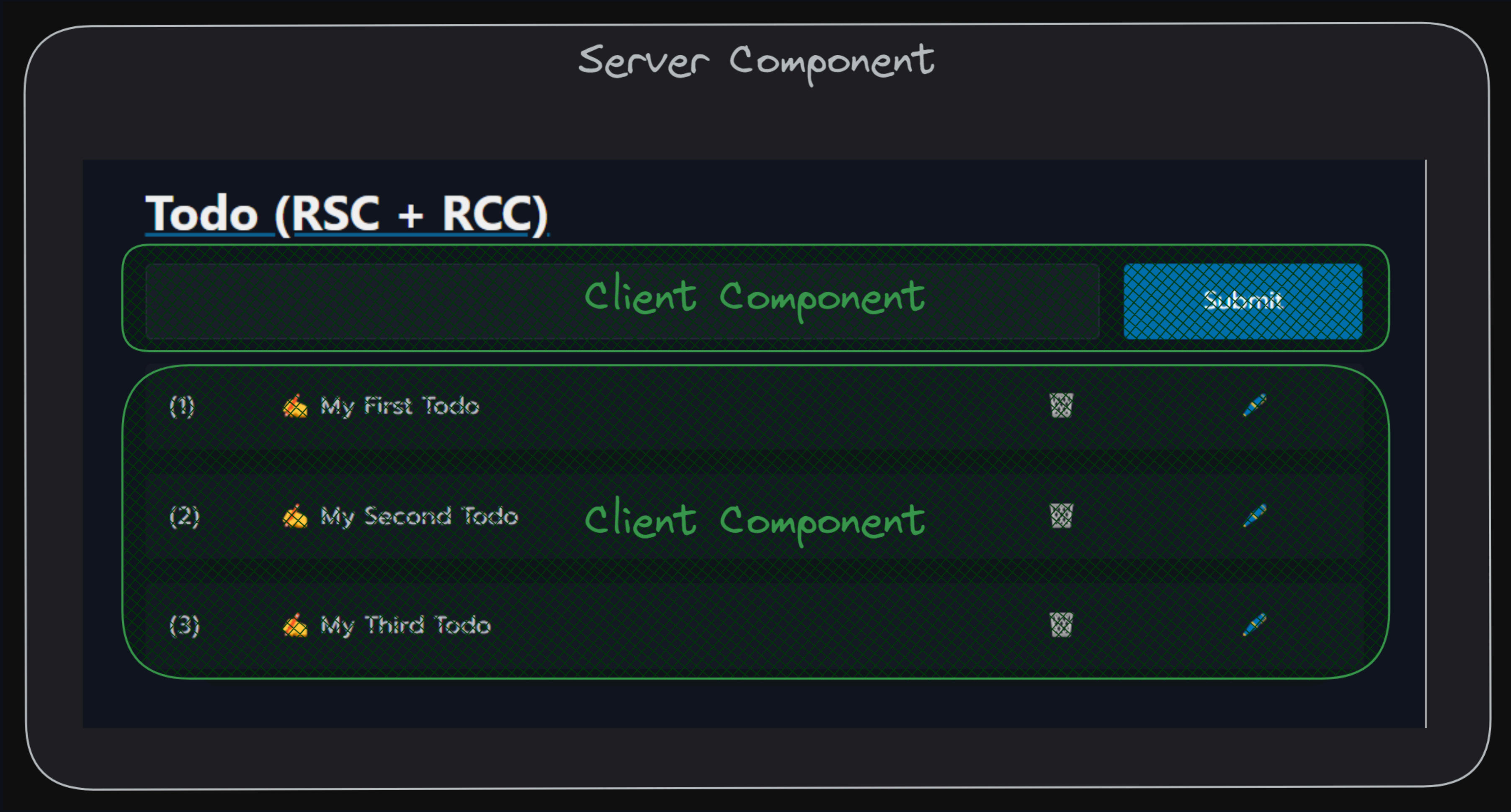
Missing UX/DX

- `UX`
  - No loading spinner
- `DX`
  - Client state is accessed from url. (`searchParam` in `page.tsx`)
    - Not type safety. Need validation.
  - *Way too complicated (compared with MPA)*

Hybrid (RSC + RCC)

- `git clone https://github.com/fullstack-68/landscape-hybrid.git rsc-client`
- `cd rsc-client`
- `pnpm install`
- `pnpm run build`
- `pnpm run start`

Structure

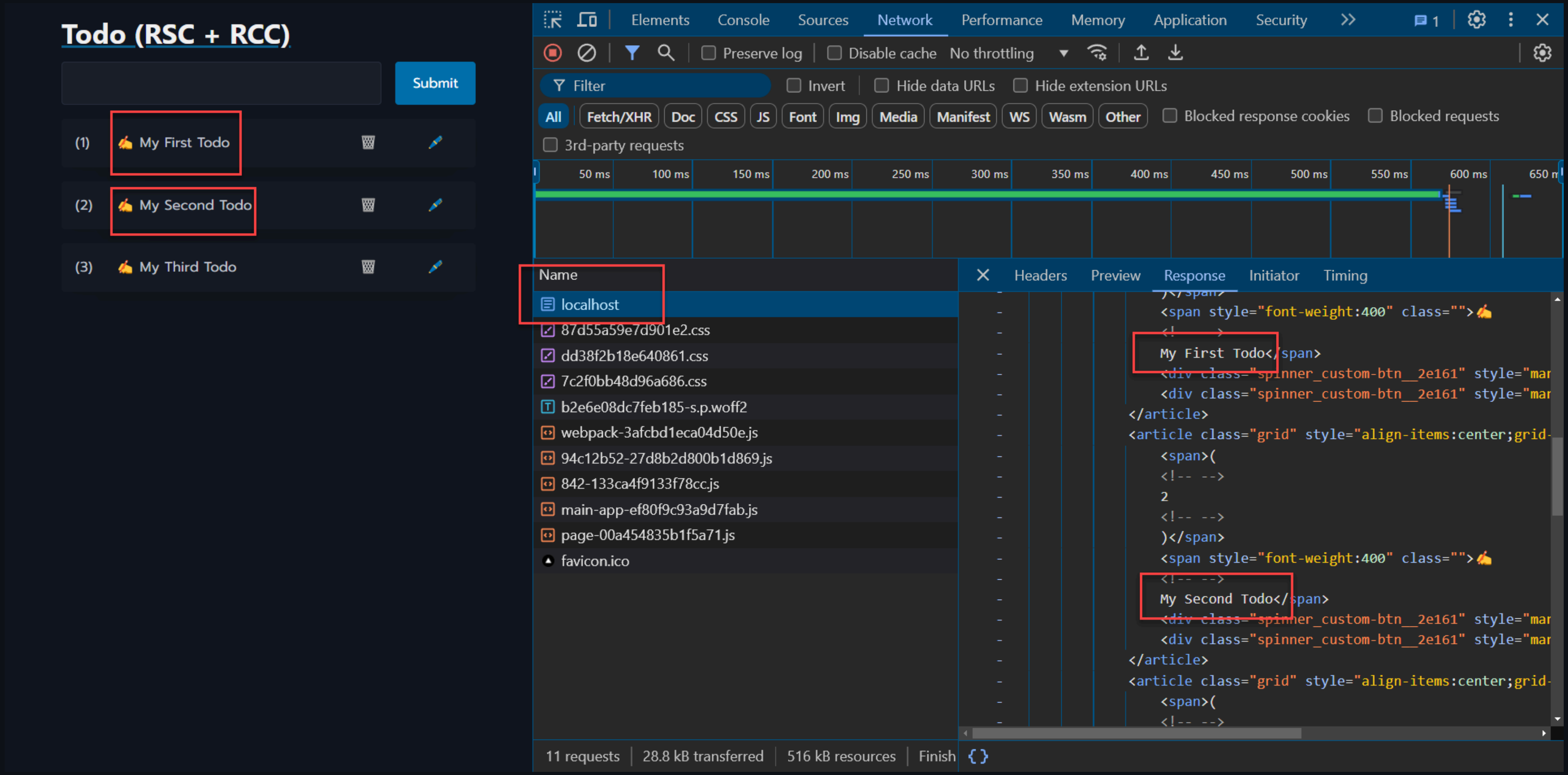

Server Component
Todo (RSC + RCC)
Client Component
Submit
(1) ✍ My First Todo
(2) ✍ My Second Todo
Client Component
(3) ✍ My Third Todo

Client component

`./src/components/TodoList.tsx`

```
"use client";  //👈 Mark component as client component
//...
export const TodoList: FC<Props> = ({ todos }) => {
  const { curTodo } = useStore((state) => state);
  return (
    <>
      {...}
    </>
  );
};
```

SSR



Name
localhost
87d55a59e7d901e2.css
dd38f2b18e640861.css
7c2f0bb48d96a686.css
b2e6e08dc7feb185-s.p.woff2
webpack-3afcbd1eca04d50e.js
94c12b52-27d8b2d800b1d869.js
842-133ca4f9133f78cc.js
main-app-ef80f9c93a9d7fab.js
page-00a454835b1f5a71.js
favicon.ico

```
        <span style="font-weight:400" class="">✍
        <!-- -->
        My First Todo</span>
        <div class="spinner_custom-btn__2e161" style="mar
        <div class="spinner_custom-btn__2e161" style="mar
    </article>
    <article class="grid" style="align-items:center;grid-
        <span>(
        <!-- -->
        2
        <!-- -->
        )</span>
        <span style="font-weight:400" class="">✍
        <!-- -->
        My Second Todo</span>
        <div class="spinner_custom-btn__2e161" style="mar
        <div class="spinner_custom-btn__2e161" style="mar
    </article>
    <article class="grid" style="align-items:center;grid-
        <span>(
        <!-- -->
```

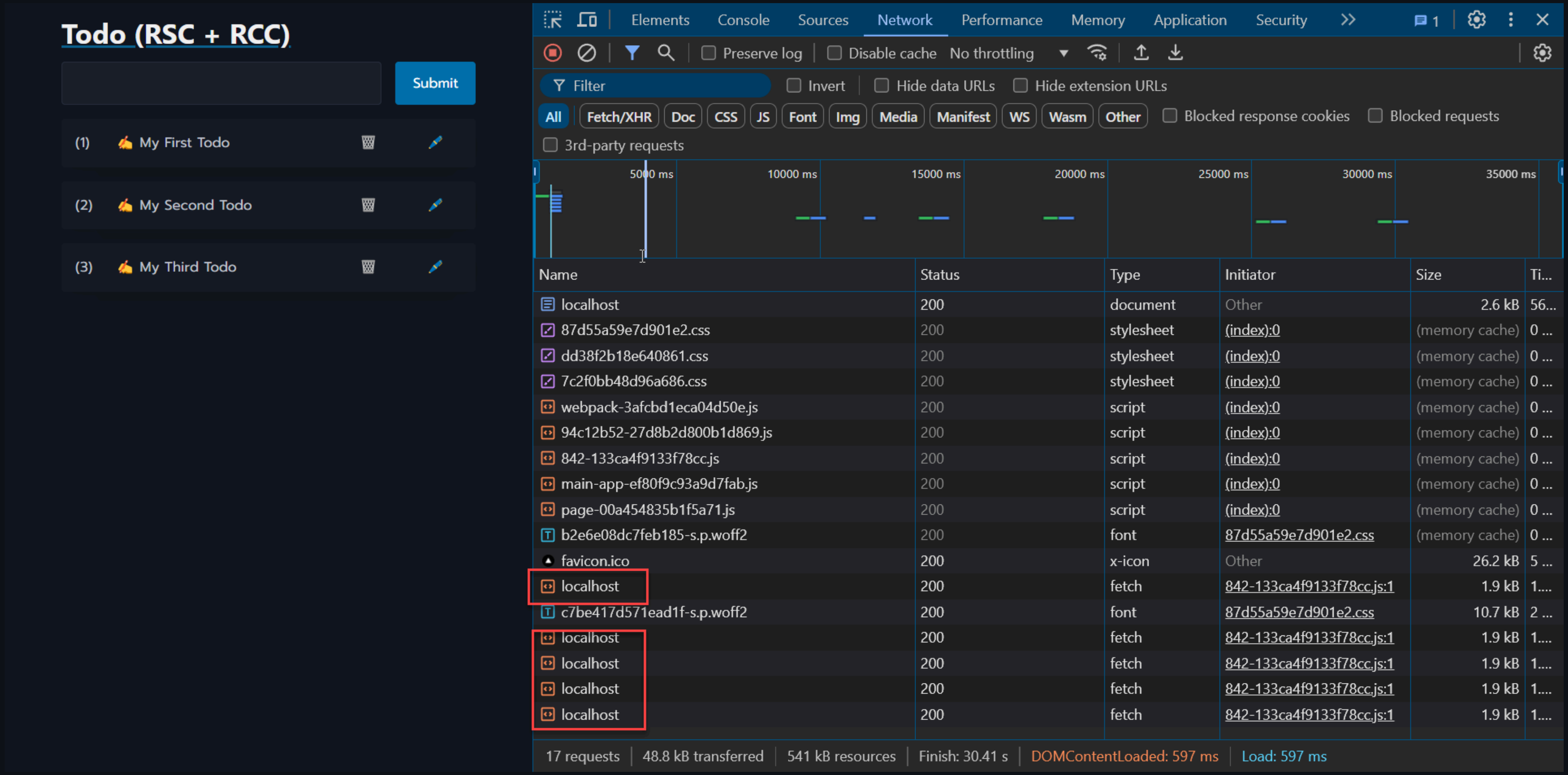
11 requests | 28.8 kB transferred | 516 kB resources | Finish

SSR

`./src/app/page.tsx`

```
export default function Home({ params, searchParams }: PageProps) {
  const todosPromise = getTodos();
  return (
    <main className="container">
      <a href="/">
        <h1>Todo (RSC + RCC)</h1>
      </a>
      <FormInput />
      <Suspense fallback={<div>Loading...</div>}>
        <TodoList todosPromise={todosPromise} /> //👈 Inject data to client
        component (SSR)
      </Suspense>
      <Spinner />
    </main>
  );
}
```

No page refresh



Todo (RSC + RCC)

Submit

(1) ✍ My First Todo

(2) ✍ My Second Todo

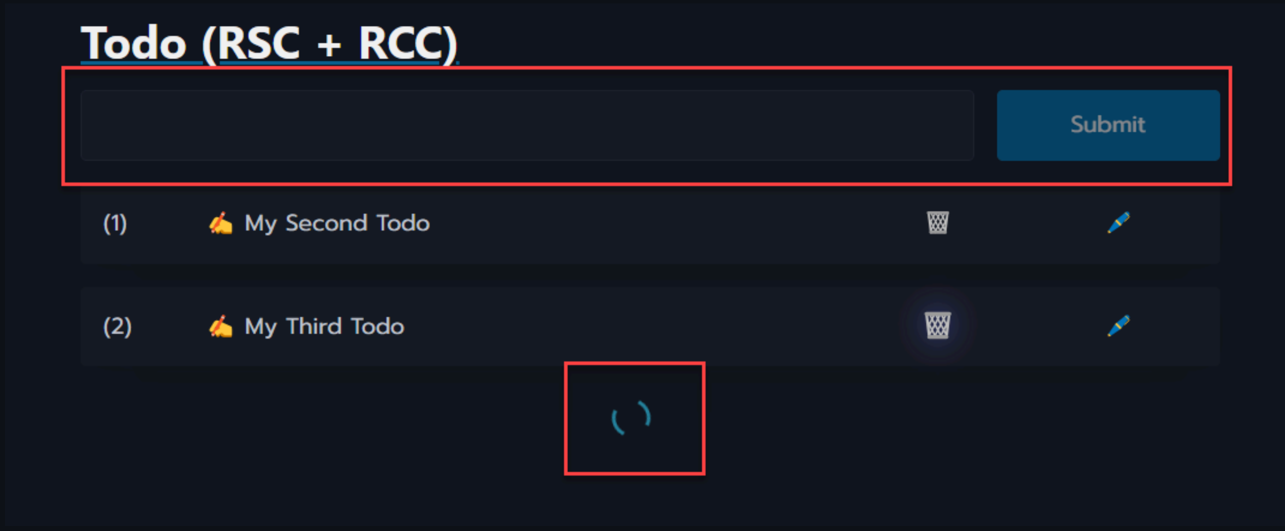
(3) ✍ My Third Todo

Name	Status	Type	Initiator	Size	Ti...
localhost	200	document	Other	2.6 kB	56...
87d55a59e7d901e2.css	200	stylesheet	(index):0	(memory cache)	0 ...
dd38f2b18e640861.css	200	stylesheet	(index):0	(memory cache)	0 ...
7c2f0bb48d96a686.css	200	stylesheet	(index):0	(memory cache)	0 ...
webpack-3afcbd1eca04d50e.js	200	script	(index):0	(memory cache)	0 ...
94c12b52-27d8b2d800b1d869.js	200	script	(index):0	(memory cache)	0 ...
842-133ca4f9133f78cc.js	200	script	(index):0	(memory cache)	0 ...
main-app-ef80f9c93a9d7fab.js	200	script	(index):0	(memory cache)	0 ...
page-00a454835b1f5a71.js	200	script	(index):0	(memory cache)	0 ...
b2e6e08dc7feb185-s.p.woff2	200	font	87d55a59e7d901e2.css	(memory cache)	0 ...
favicon.ico	200	x-icon	Other	26.2 kB	5 ...
localhost	200	fetch	842-133ca4f9133f78cc.js:1	1.9 kB	1....
c7be417d571ead1f-s.p.woff2	200	font	87d55a59e7d901e2.css	10.7 kB	2 ...
localhost	200	fetch	842-133ca4f9133f78cc.js:1	1.9 kB	1....
localhost	200	fetch	842-133ca4f9133f78cc.js:1	1.9 kB	1....
localhost	200	fetch	842-133ca4f9133f78cc.js:1	1.9 kB	1....
localhost	200	fetch	842-133ca4f9133f78cc.js:1	1.9 kB	1....

17 requests | 48.8 kB transferred | 541 kB resources | Finish: 30.41 s | DOMContentLoaded: 597 ms | Load: 597 ms

Interactivity

- Elements disabled
- Spinner



(1) ✍ My Second Todo

(2) ✍ My Third Todo

Server action in a separate file

`./src/app/actionAdnDb.ts` *(Slightly modified)*

```
"use server"; //👈 Mark functions in this file as server actions.
//...
export async function actionCreateTodo(todoText: string) {
  //...
  await createTodos(todoText);
  //...
  revalidatePath("/");
  return { message: "" };
}
```

- You can import this function into client component.
- You don't have to use `form-data` anymore.
- You can `return` data back to client component.

Server action in client component

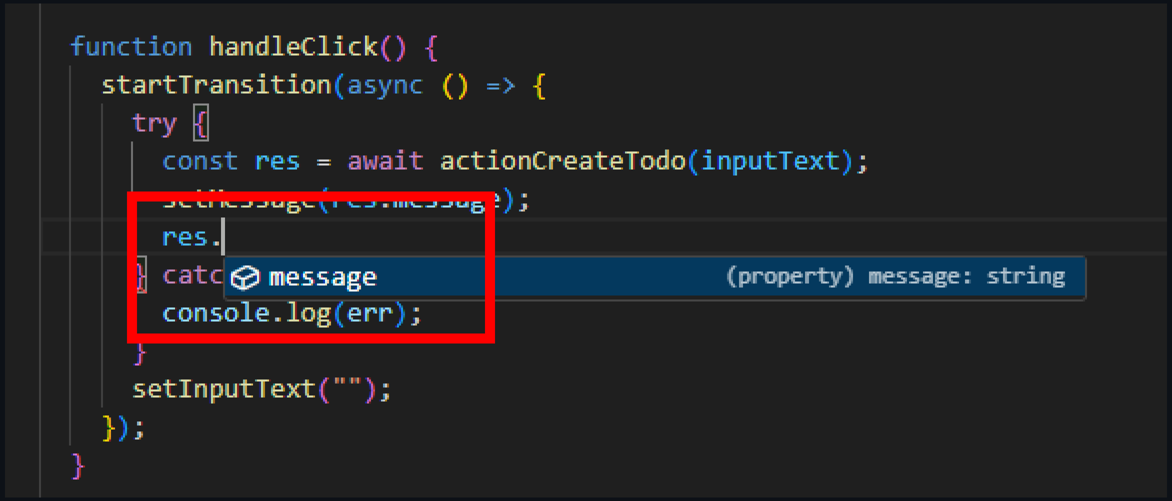
`./src/components/FormInput.tsx`

```tsx
const ButtonSubmit: FC<PropsButtonSubmit> = ({ setMessage }) => {
  // ...
  function handleClick() {
    startTransition(async () => {
      const res = await actionCreateTodo(inputText);
      setMessage(res.message);
      setInputText("");
    });
  }
  // ...
  return <button onClick={handleClick}>Submit</button>;
};
```

- I can trigger server action anywhere, not just `form`.

Type safety

`./src/components/FormInput.tsx`



```
function handleClick() {
  startTransition(async () => {
    try {
      const res = await actionCreateTodo(inputText);
      setMessage(res.message);
      res.
    } catc  message                       (property) message: string
      console.log(err);
    }
    setInputText("");
  });
}
```

- You get type safety from client component to database levels 👍.

Line of codes

Type	**# Line**
MPA	203
SPA	504
RSC	292
RSC + RCC	😭 527

(In Next JS project, I counted `src` *dir.)*

> Have we gone too far?

Round 3

> Back to basic

HTMX

What exactly is HTMX?

> Small JS library that can swap parts of UI with ***HTML response*** from a server.

2023 JavaScript Rising Stars

Get started

- `git clone https://github.com/fullstack-68/landscape-htmx.git htmx`
- `cd htmx`
- `pnpm install`
- `pnpm run build`
- `pnpm run start`

Todo app

- The stack is very similar to MPA (`express` + `pug`).
- SSR enabled *(duh!)*
- No reloading
- Spinner enabled
- Form input disabled during submission.

Code count

Type	**# Line**
MPA	203
SPA	504
RSC	292
RSC + RCC	527
HTMX	☺️ 259

> *Take that NextJS!*

In addition

- Hypermedia as the Engine of Application State (HATEOAS)
- HTMX + Alpine.js

Summary

Architecture	Usage
MPA	Backend-critical (financial, ERP, ทะเบียน)
SPA	Dashboards, editor
RSC	Blogs, brochure sites
RSC + RCC	Highly interactive website with latest technology (to justify high price)
HTMX	Apps that finish on time