Fullstack Development

Stack Overflow 2024 Developer Survey

- Developer types
- Databases
- Web framework
- Tools

Fullstack Landscape

> 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

Todo app (MPA)

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

Get started

- `npm install -g pnpm`
  - I am using `pnpm`.
- `git clone -b mpa https://github.com/fullstack-67/landscape-server`
- `cd landscape-server`
- `pnpm install`
- `npm run dev`

Endpoint

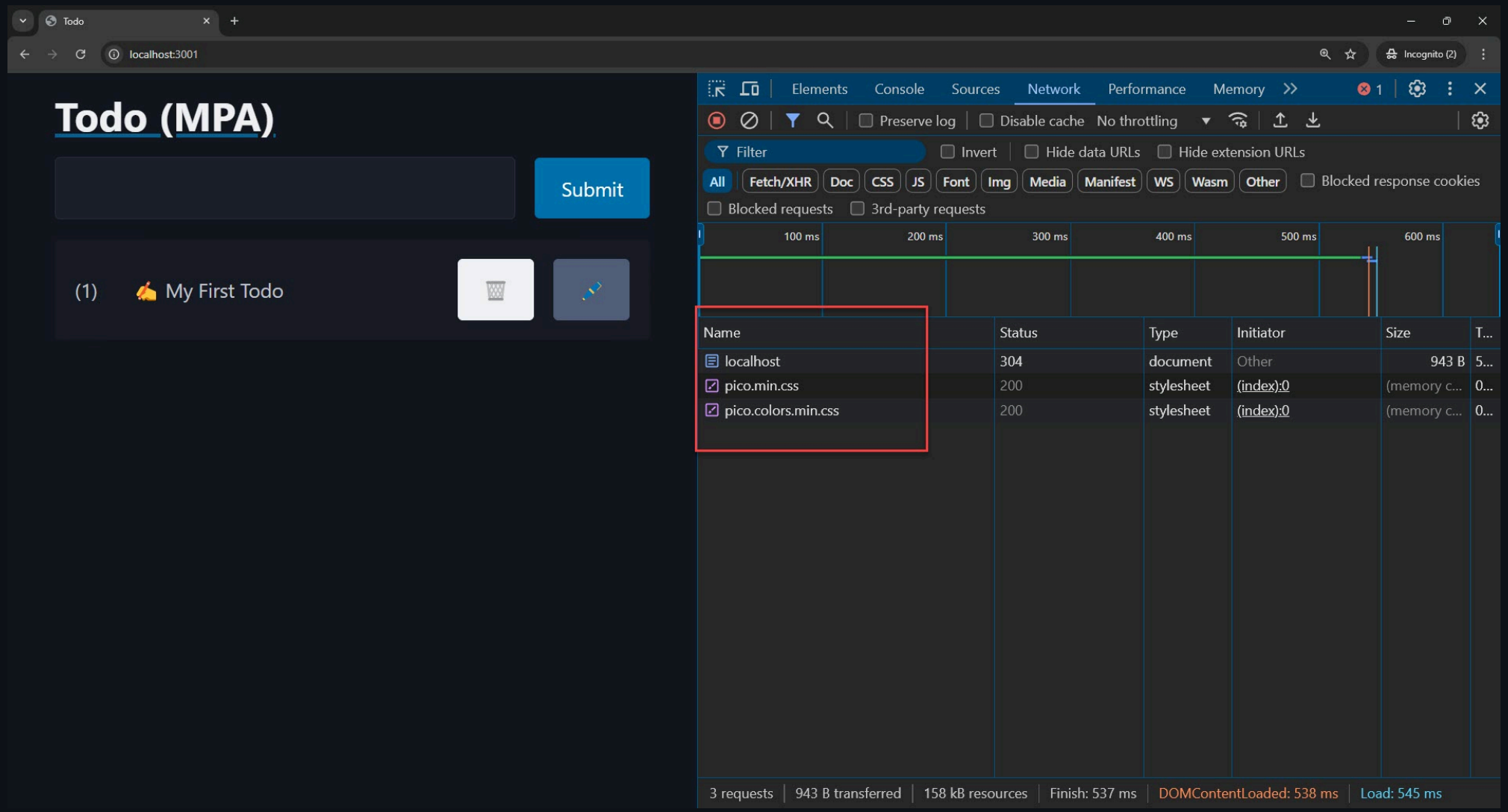
./src/index.ts

```
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`

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



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

```ts
// 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 ?? "";
```

No type safety

inputForm.pug

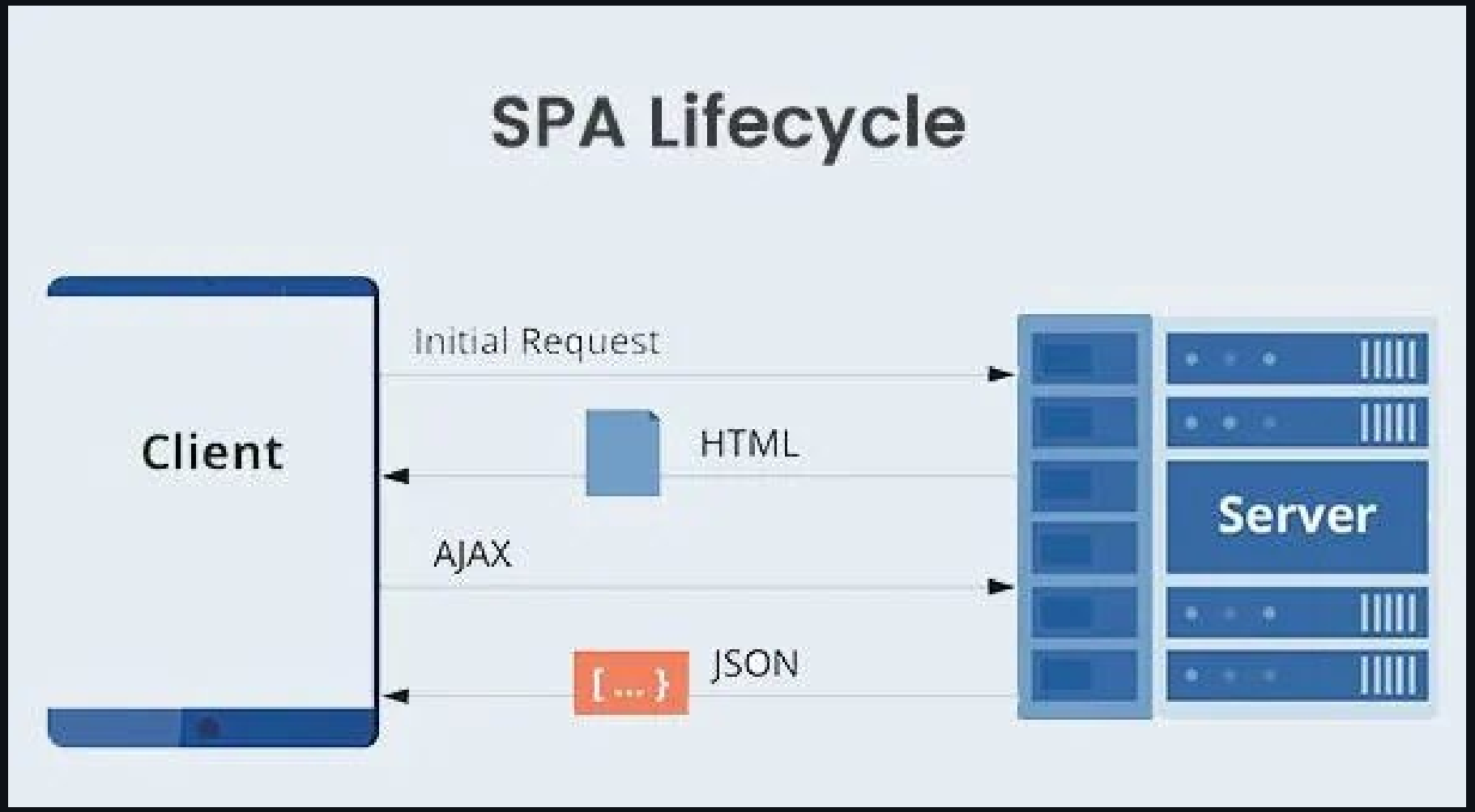
views > components > inputForm.pug

```
 form
idTemplateColumns = mode === "ADD" ? "4fr 1fr" : "4fr 1fr 1fr"
s="grid" style=`grid-template-columns: ${gridTemplateColumns}; alig
'method="POST" action=`${mode===="ADD"? "/create" : "/update"}` styl
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
```

mpa 0 0 0 Git Graph Ln 62, Col 21 Spaces: 2 UTF-8 CRLF {} TypeScript Go Live Quokka Prettier

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.

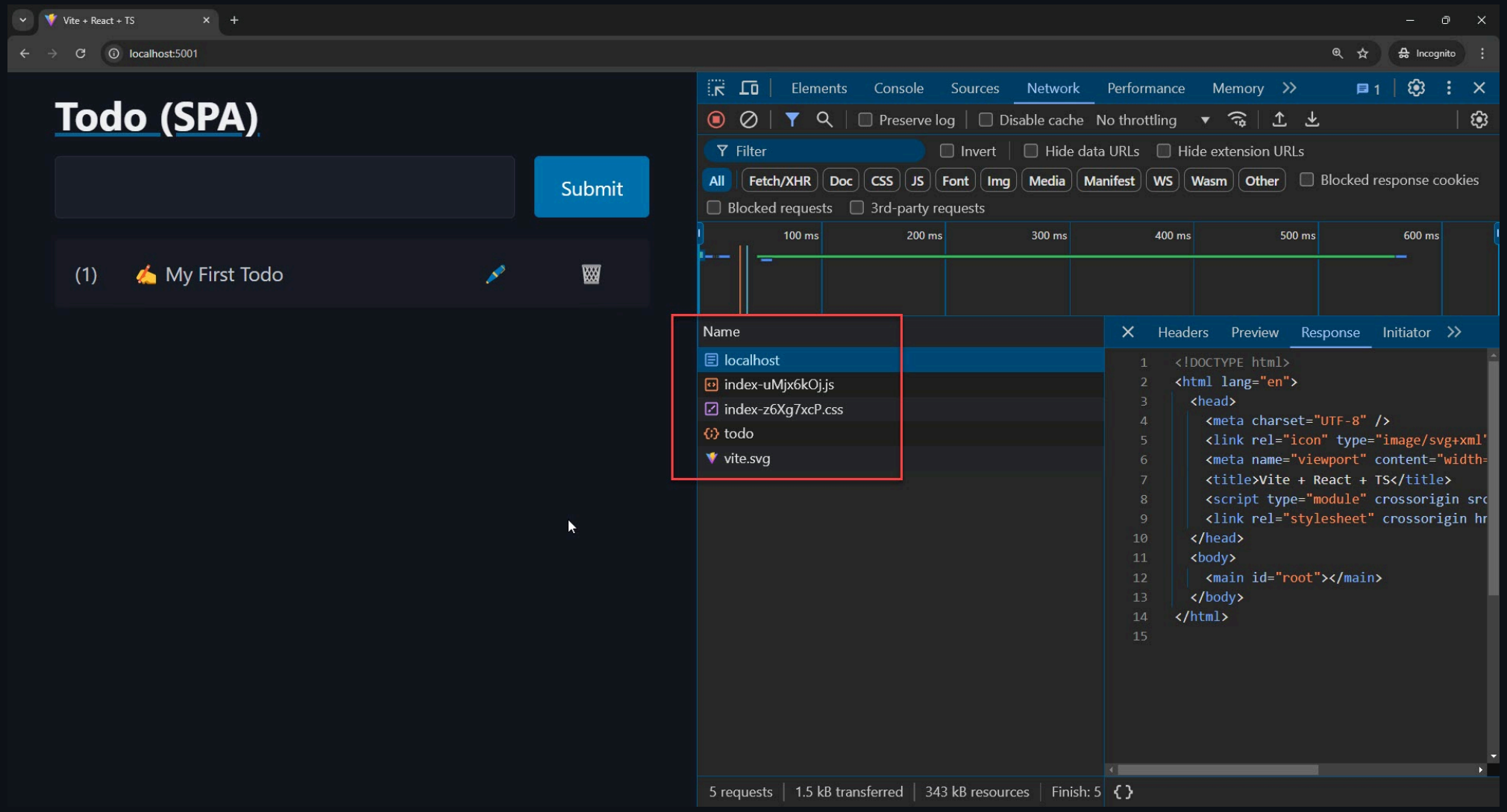

SPA Lifecycle
Client
Initial Request
HTML
AJAX
JSON
[... }
Server

Todo app (SPA)

- `Express JS` + `React`

Get started

- `git clone https://github.com/fullstack-67/landscape-spa`
- Backend
  - `cd backend`
  - `pnpm install`
  - `npm run dev`
- Frontend
  - `cd frontend`
  - `pnpm install`
  - `npm run build`
  - `npm run preview`


Vite + React + TS
localhost:5001
Todo (SPA)
Submit
(1) My First Todo
Elements
Console
Sources
Network
Performance
Memory
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
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
Name
localhost
index-uMjx6kOj.js
index-z6Xg7xcP.css
todo
vite.svg
Headers
Preview
Response
Initiator
<!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>
5 requests
1.5 kB transferred
343 kB resources
Finish: 5

Edit Selection View Go ··· landscape-spa

App.tsx

frontend > src > App.tsx > App

```
1  import { useEffect, useState } from "react";
2  import axios from "axios";
3  import { type TodoType } from "./utils/types";
4  import { FormInput } from "./components/FromInput";
5  import { TodoList } from "./components/TodoList";
6  import { Spinner } from "./components/Spinner";
7
8  function App() {
9    // Fetching data
10   const [todos, setTodos] = useState<TodoType[]>([]);
11   async function fetchData() {
12     const res = await axios.get<TodoType[]>("api/todo");
13     setTodos(res.data);
14   }
15   useEffect(() => {
16     fetchData();
17   }, []);
18
```

types.ts

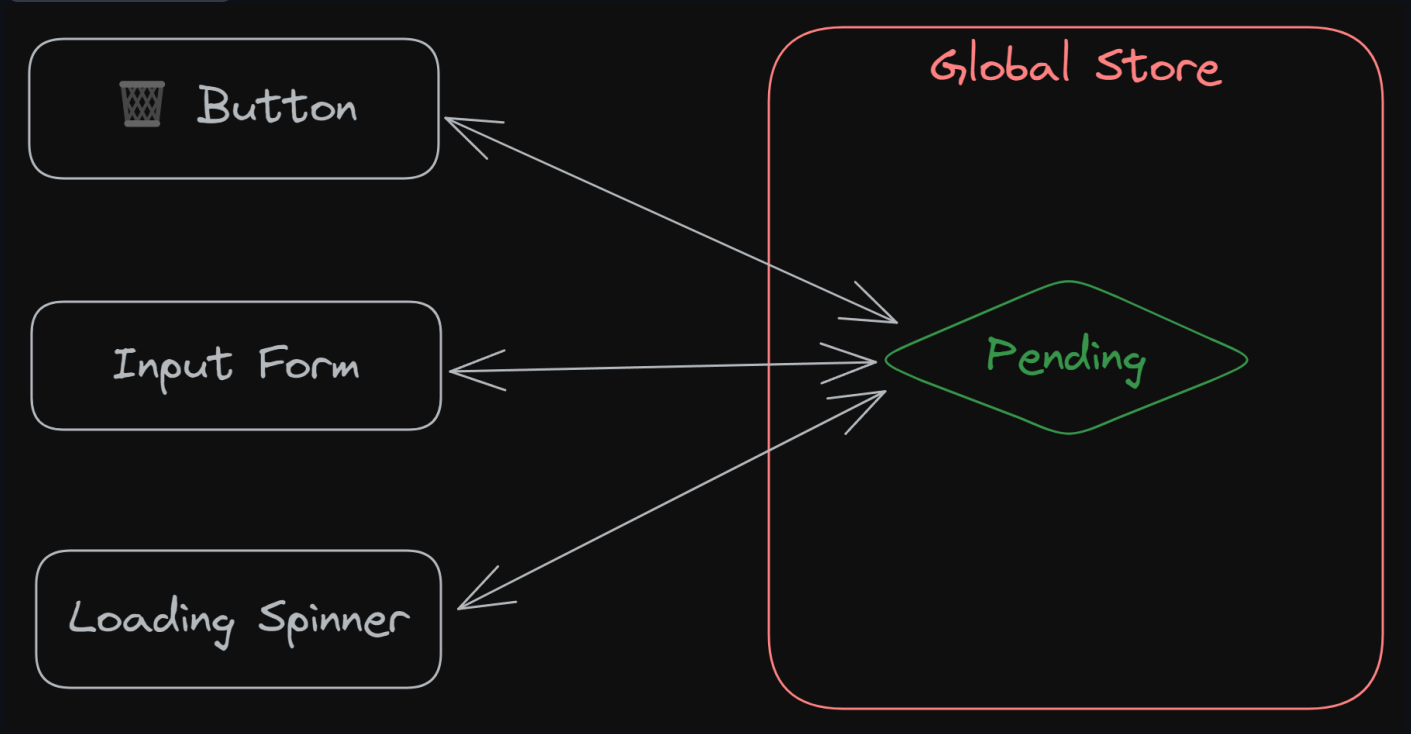
frontend > src > utils > types.ts > ...

```
1  export interface TodoType {
2    id: string;
3    todoText: string;
4  }
5
```

Not safe

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

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):

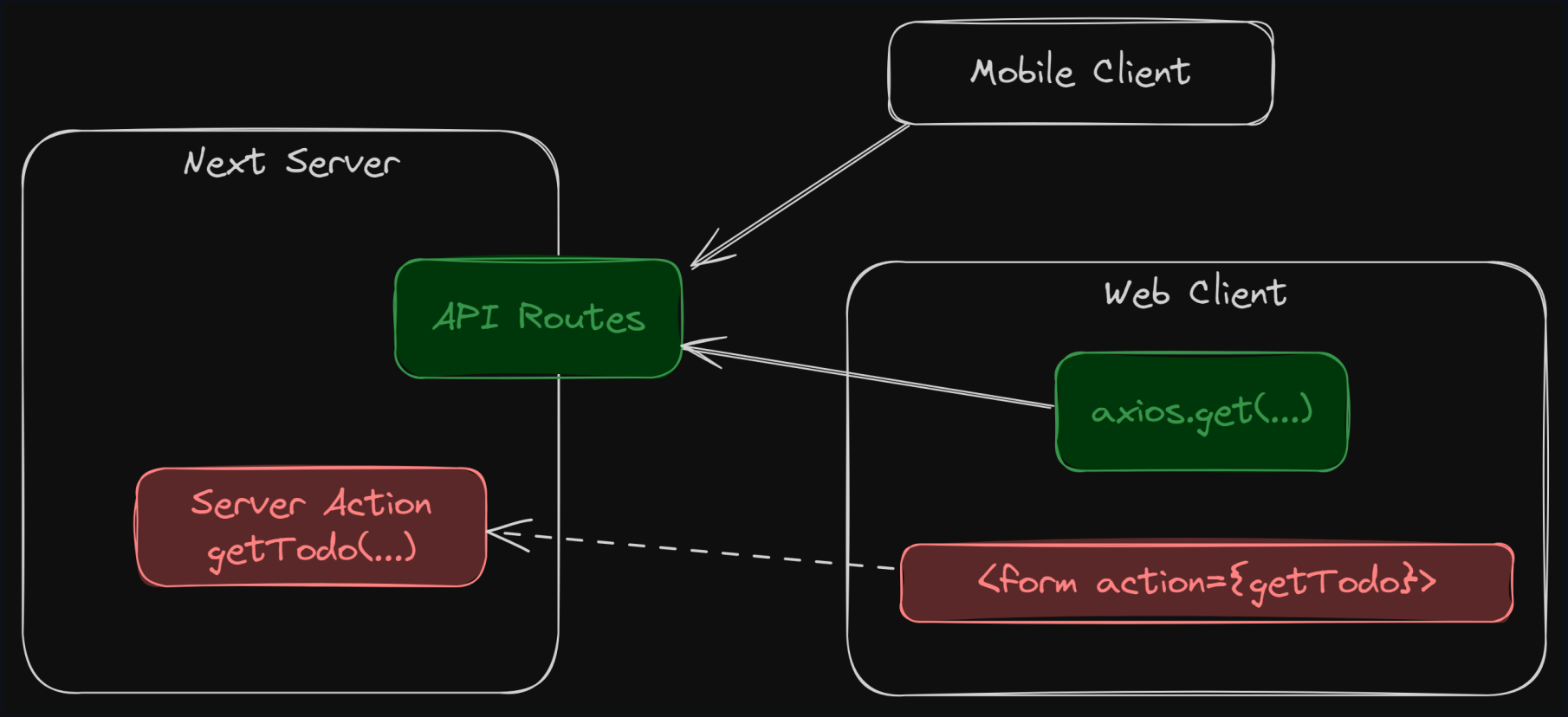
```
<form action="/action_page.php" method="get">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>
```

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 -b rsc-only https://github.com/fullstack-67/landscape-hybrid.git rsc-only`
- `cd rsc-only`
- `pnpm install`
- `npm run build`
- `npm run start`

Server components

`./src/app/page.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)*

```
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
    revalidatePath("/"); // Change RSC content without refreshing
  }

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

./src/components/TodoList.tsx

```
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>
  );
};
```

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

Missing UX/DX

- `UX`
  - Form not resetting after submission.
  - Page refresh when changing "client" states (error messsage).
  - No loading spinner
- `DX`
  - Client state is accessed from url. (`searchParam` in `page.tsx`)
    - Not type safety. Need validation.
  - Still need to include hidden input field (Update Button).

Hybrid (RSC + RCC)

- `git clone -b rsc-client https://github.com/fullstack-67/landscape-hybrid.git rsc-client`
- `cd rsc-client`
- `pnpm install`
- `npm run build`
- `npm run start`

Server action in client component

Line of codes

Type	# Line
MPA	203
SPA	504
RSC	292
RSC + RCC	475

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

Round 3

> Enlightment

Hypermedia-driven application - HTMX