

# 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

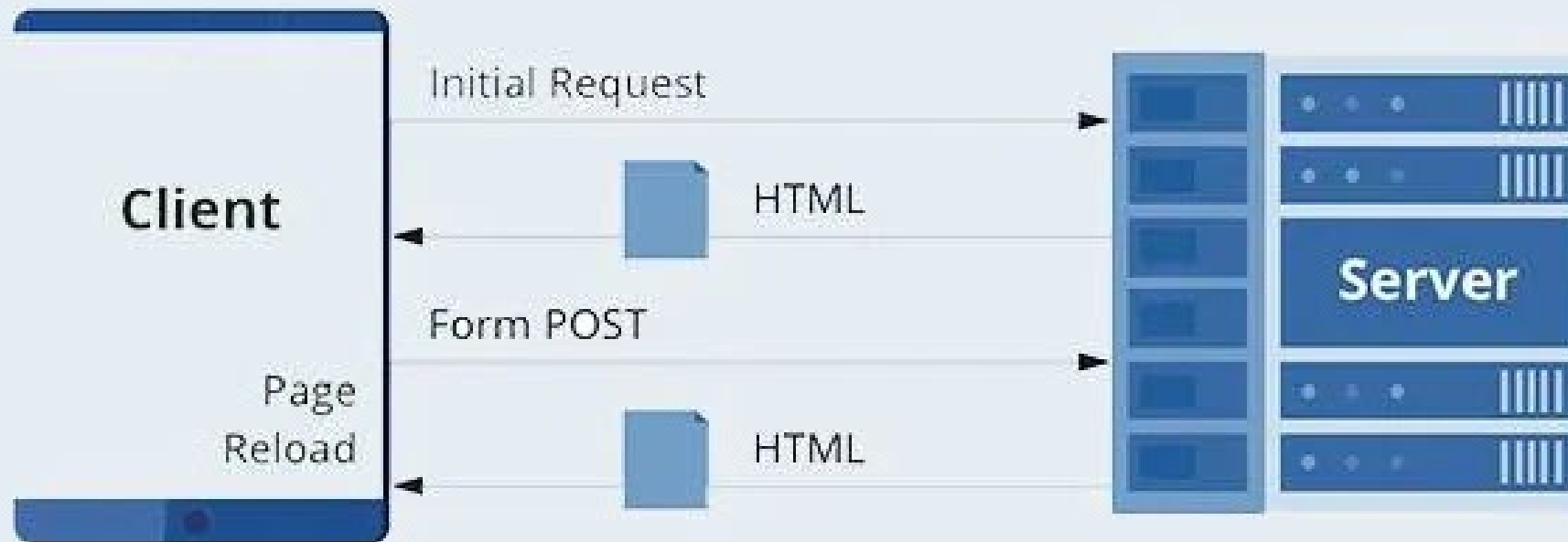
# Round 1

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 mpa`
- `cd mpa`
- `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
```

# Todo (MPA)

Submit

(1) 🍌 My First Todo

ElementsConsoleSourcesNetworkPerformanceMemory

1

FilterInvertHide data URLSHide extension URLs

AllFetch/XHRDocCSSJSFontImgMediaManifestWSWasmOtherBlocked response cookies

Blocked requests3rd-party requests

100 ms200 ms300 ms400 ms500 ms600 ms

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

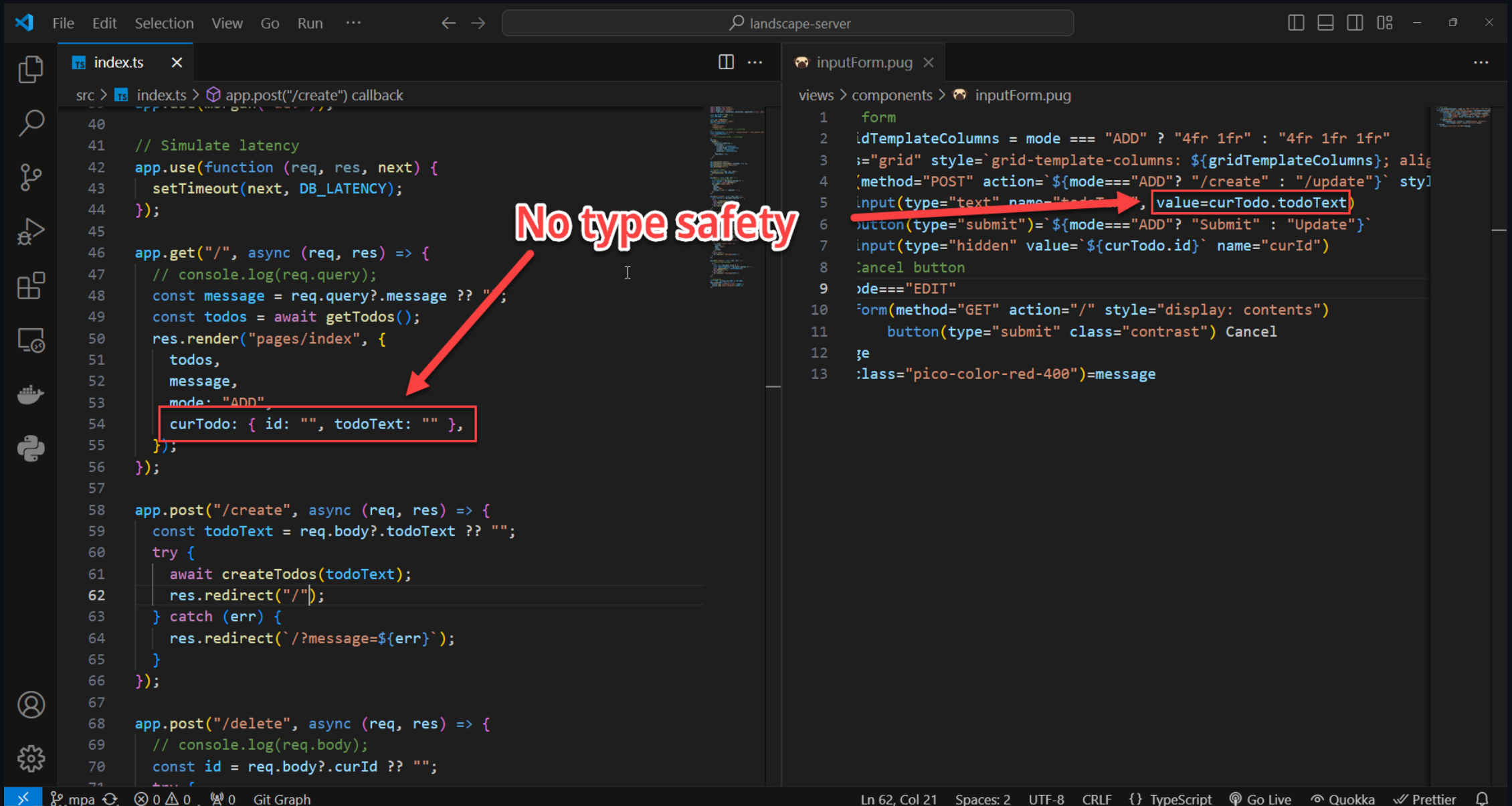
261497: Fullstack Development

*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.

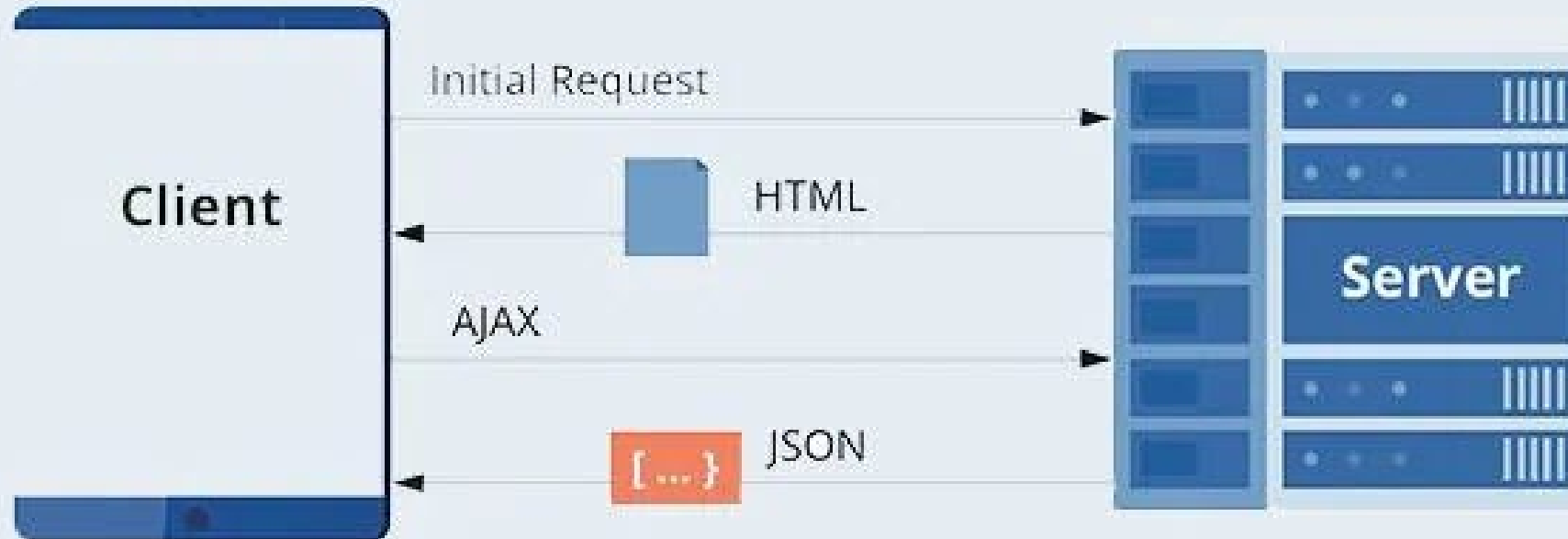
```
form(action="/delete" method="POST" style="display: contents")
  input(type="hidden" value={`${todo.id}`} name="curId")
  button(type="submit" class="contrast" style="margin-bottom: 0") 🗑️
form(action="/edit" method="POST" style="display: contents")
  input(type="hidden" value={`${todo.id}`} name="curId")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



# 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





# 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

Network

Filter

All Fetch/XHR Doc CSS JS Font Img Media Manifest WS Wasm Other

Blocked requests 3rd-party requests

100 ms 200 ms 300 ms 400 ms 500 ms 600 ms

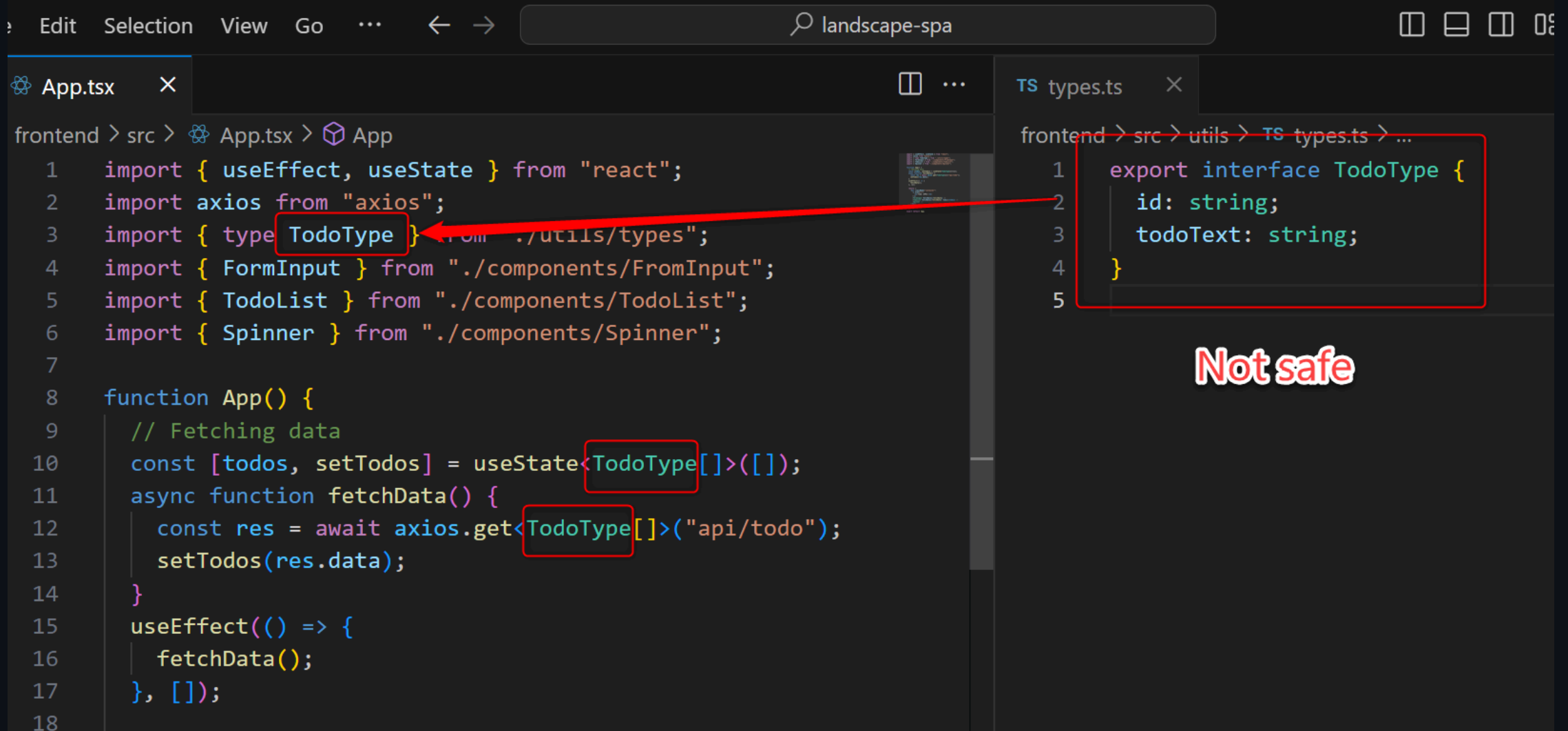
Name

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

Headers Preview Response Initiator >>

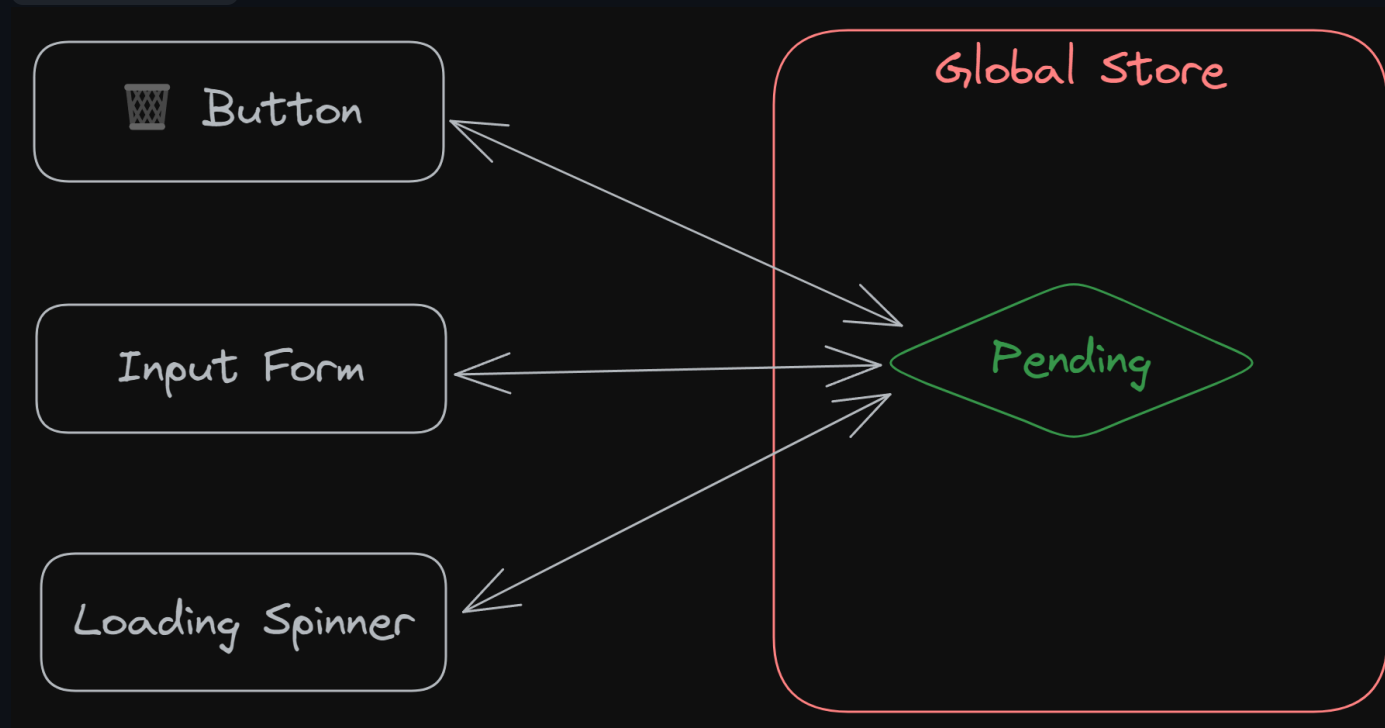
```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <link rel="icon" type="image/svg+xml"
6     <meta name="viewport" content="width=
7     <title>Vite + React + TS</title>
8     <script type="module" crossorigin src
9     <link rel="stylesheet" crossorigin hr
10  </head>
11  <body>
12    <main id="root"></main>
13  </body>
14 </html>
15
```

5 requests | 1.5 kB transferred | 343 kB resources | Finish: 5



# 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
<i>MPA</i>		
./src	2	161
./views	3	42
<i>SPA</i>		
./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 🤪



36



Reply



[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):

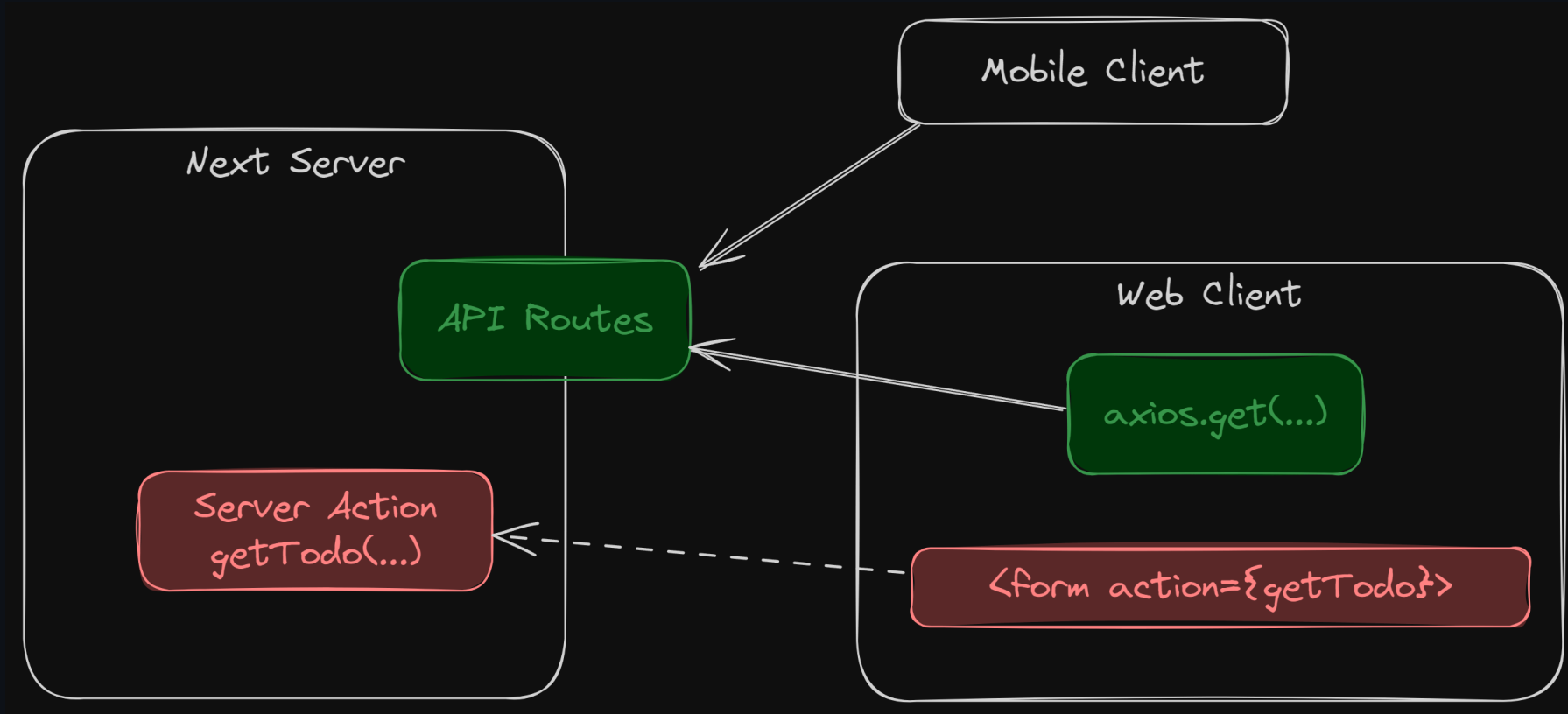
```
<form action="/action_page.php" method="get">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>
```

[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



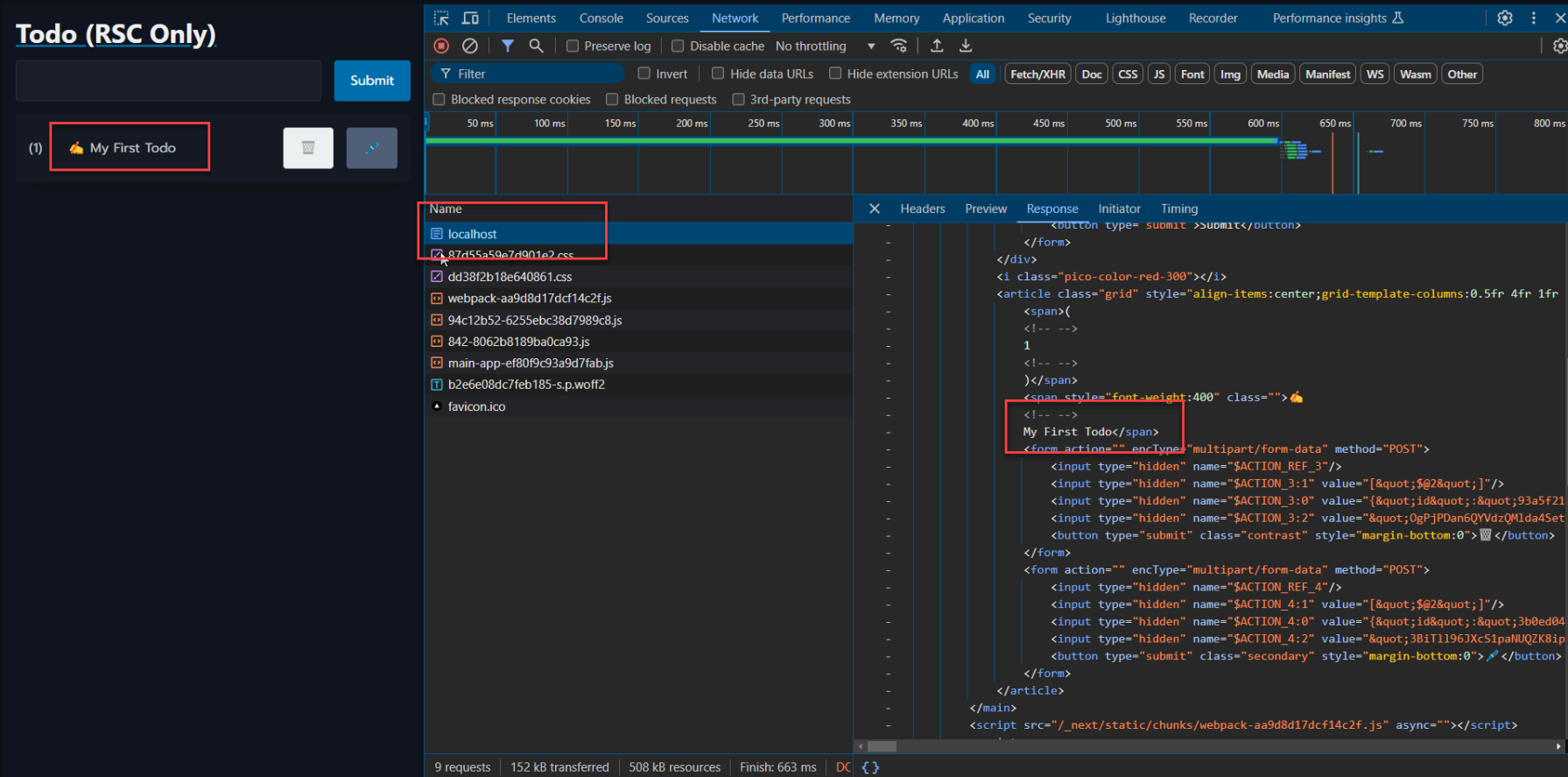
# Form action

- No need to define new endpoints.
- Accept `form-data`
- Colocation
  - Type safe
  - Can "bind" data that is passed through 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`

# No blank HTML





# No page refresh

Take that, PHP!

### Todo (RSC Only)

Submit

(1) 🍌 My First Todo

(2) 🍌 My Second Todo (Edited)

(3) 🍌 My Third Todo

50000 ms100000 ms150000 ms200000 ms250000 ms300000 ms

Name	Status	Type	Initiator	Size	Time
localhost	200	document	Other	4.4 kB	595 ...
87d55a59e7d901e2.css	200	stylesheet	(index):0	12.7 kB	9 ms
dd38f2b18e640861.css	200	stylesheet	(index):0	8.6 kB	10 ms
webpack-aa9d8d17dcf14c2f.js	200	script	(index):0	2.0 kB	9 ms
94c12b52-6255ebc38d7989c8.js	200	script	(index):0	54.2 kB	12 ms
842-8062b8189ba0ca93.js	200	script	(index):0	32.0 kB	10 ms
main-app-ef80f9c93a9d7fab.js	200	script	(index):0	809 B	7 ms
b2e6e08dc7feb185-s.p.wolf2	200	font	87d55a59e7d901e2.css	11.0 kB	2 ms
• favicon.ico	200	x-icon	Other	26.2 kB	4 ms
localhost	303	fetch	842-8062b8189ba0ca93.js:1	2.4 kB	1.10 s
• favicon.ico	200	x-icon	Other	26.2 kB	9 ms
?message=&curlId=&mode=ADD	303	fetch	842-8062b8189ba0ca93.js:1	2.2 kB	1.09 s
c7be417d571ead1f-c.p.wolf2	200	font	87d55a59e7d901e2.css	10.7 kB	2 ms
• favicon.ico	200	x-icon	Other	26.2 kB	4 ms
?mode=EDIT&curlId=122832	303	fetch	842-8062b8189ba0ca93.js:1	2.3 kB	1.06 s
• favicon.ico	200	x-icon	Other	26.2 kB	4 ms
localhost	303	fetch	842-8062b8189ba0ca93.js:1	2.4 kB	1.06 s
• favicon.ico	200	x-icon	Other	26.2 kB	6 ms
?message=Empty%20Text&curlId=&mode=ADD	303	fetch	842-8062b8189ba0ca93.js:1	2.6 kB	1.06 s
• favicon.ico	200	x-icon	Other	26.2 kB	4 ms

20 requests | 306 kB transferred | 673 kB resources | Finish: 5.1 min | DOMContentLoaded: 634 ms | Load: 652 ms

# 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

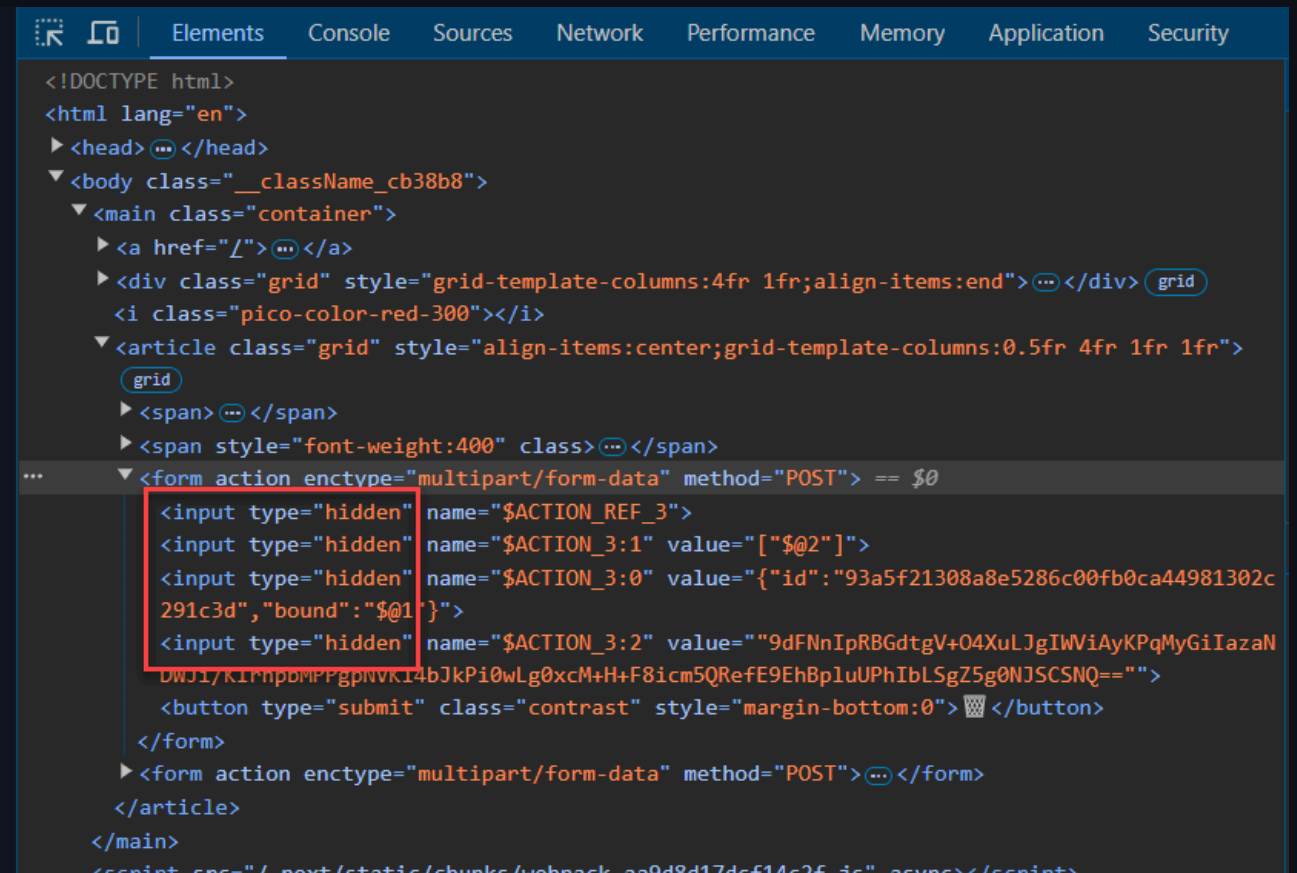
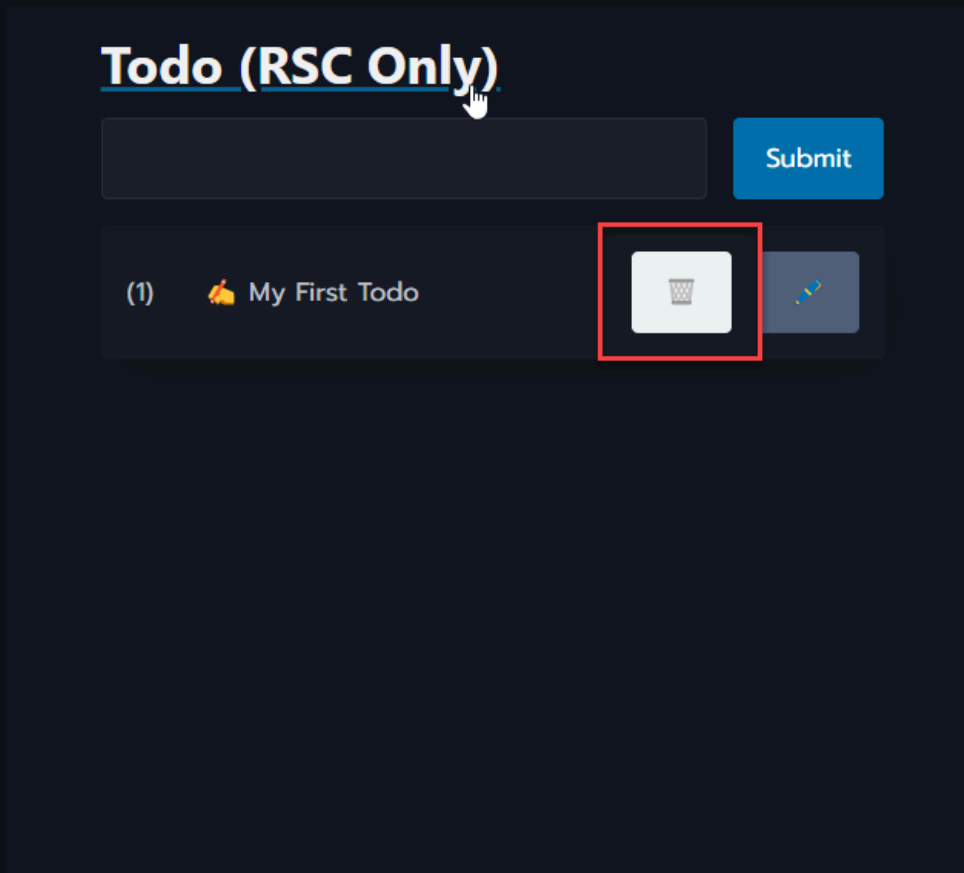
`./src/components/FormInput.tsx` *(Slightly modified)*

```
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
    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

```
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.

# 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).
- Both `revalidatePath` and `redirect` do not trigger page refresh. (Nice!)

# Missing UX/DX

- UX
  - Form not resetting after submission.
  - 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 (form `input` ).
  - *Way too complicated (compared with MPA)*

# Hybrid (RSC + RCC)

- `git clone -b rsc-client-v2 https://github.com/fullstack-67/landscape-hybrid.git rsc-client`
- `cd rsc-client`
- `pnpm install`
- `npm run build`
- `npm 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.curTodo]);
  return (
    <>
      {...}
    </>
  );
};
```

# SSR

## Todo (RSC + RCC)

Submit

(1) 🍌 My First Todo

🗑️

✏️

(2) 🍌 My Second Todo

🗑️

✏️

(3) 🍌 My Third Todo

🗑️

✏️

Network tab showing the response for the first request (localhost).

Response content (HTML):

```
<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 async function Home() {
  const todos = await getTodos();
  return (
    <main className="container">
      <a href="/">
        <h1>Todo (RSC + RCC)</h1>
      </a>
      <FormInput />
      <TodoList todos={todos} /> // ➡ Inject data to client component (SSR)
      <Spinner />
    </main>
  );
}
```

# No page refresh

## Todo (RSC + RCC)

Submit

(1) 🍌 My First Todo

(2) 🍌 My Second Todo

(3) 🍌 My Third Todo

The screenshot shows the Chrome DevTools Network tab. The top bar includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, and Security. The Network tab is active, showing a waterfall chart and a list of requests. The list of requests is as follows:

Name	Status	Type	Initiator	Size	Time
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-3afc1eca04d50e.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...

The bottom of the Network tab shows summary statistics: 17 requests, 48.8 kB transferred, 541 kB resources, Finish: 30.41 s, DOMContentLoaded: 597 ms, Load: 597 ms.

# Interactivity

- Elements disabled
- Spinner

## Todo (RSC + RCC)

(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

```
const ButtonSubmit: FC<PropsButtonSubmit> = ({ setMessage }) => {  
  // ...  
  const submit = actionCreateTodo.bind(null, inputText); //👉 Bind data to server action  
  // ...  
  function handleClick() {  
    setPending(true);  
    //👉 Call server action from click event  
    submit().then((res) => {  
      setMessage(res.message); //👉 Get return data from server action  
    }); //...  
  }  
  // ...  
  return <button onClick={handleClick}>Submit</button>;  
};
```

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



# Type safety

./src/components/FormInput.tsx

```
const submit = actionCreateTodo.bind(null, inputText);  
function handleClick() {  
  setPending(true);  
  submit()  
    .then((res) => {  
      setMessage(res.);  
    })  
    .catch((err) => {  
      console.log(err);  
    })  
    .finally(() => {  
      setPending(false);  
      setInputText("");  
    });  
}
```

message

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

# Flashing UI

```
src > app > actionsAndDb.ts > actionUpdateTodo
1  "use server";
2  import { revalidatePath } from "next/cache";
3  // import { redirect } from "next/navigation";
4
5  export async function actionCreateTodo(todoText: string) {
6    try {
7      await createTodos(todoText);
8    } catch (err: any) {
9      if (typeof err === "string") {
10        return { message: err };
11      } else {
12        return { message: "Unknown Error" };
13      }
14    }
15    revalidatePath("/");
16    return { message: "" };
17  }
18
19  export async function actionUpdateTodo(curId: string, todoTextUpdated: string) {
20    try {
21      await updateTodo(curId, todoTextUpdated);
22    } catch (err) {
23      if (typeof err === "string") {
24        return { message: err };
25      } else {
26        return { message: "Unknown Error" };
27      }
28    }
29    revalidatePath("/");
30    return { message: "" };
31  }

```

```
src > components > FormInput.tsx > FormInput
55 const ButtonSubmit: FC<PropsButtonSubmit> = ({ setMessage }) => {
56   state.pending,
57   state.setPending,
58   state.inputText,
59   state.setInputText,
60 }
61
62 const submit = actionCreateTodo.bind(null, inputText);
63
64 function handleClick() {
65   setPending(true);
66   submit()
67     .then((res) => {
68       setMessage(res.message);
69     })
70     .catch((err) => {
71       console.log(err);
72     })
73     .finally(() => {
74       setPending(false);
75       setInputText("");
76     });
77 }
78
79 if (mode !== "ADD") return <>/>;
80
81 return (
82   <button disabled={pending} onClick={handleClick}>
83     Submit
84   </button>
85 );
86
87
88
89


```

Does not wait for this to finish

# Flashing UI

- `revalidatePath` does not block server action from returning.
- I cannot set the `pending` state at the right time.
- [Github Issue](#)

# Line of codes

Type	# Line
MPA	203
SPA	504
<b>RSC</b>	292
<b>RSC + RCC</b>	 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



# HTMX Demo

- `git clone -b htmx-demo https://github.com/fullstack-67/landscape-server htmx-demo`
- `cd htmx-demo`
- `pnpm install`
- `npm run dev`

# Todo app (HTMX)


# Get started

- `git clone -b htmx https://github.com/fullstack-67/landscape-server htmx`
- `cd htmx`
- `pnpm install`
- `npm run dev`

# 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
<b>HTMX</b>	 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, $n \leq 10^5$ )
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