FrankenJS • 30th November 2023

Web development like 15 years ago – the new fullstack frameworks

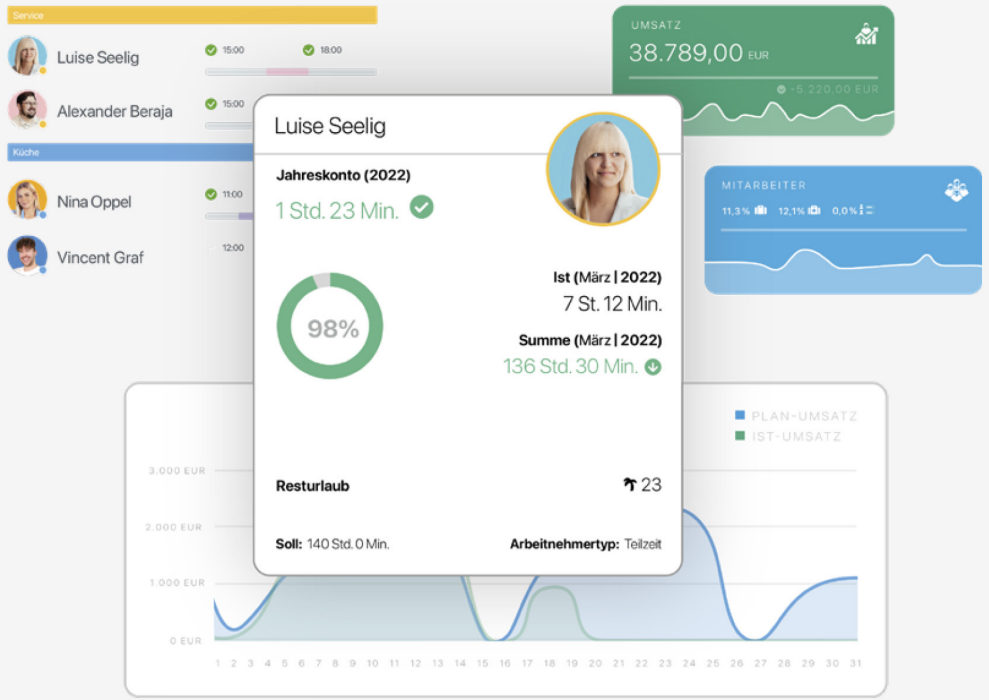
Sonja Feitsch • E2N GmbH

Hi, its me 👋

- During my studies in biology I discovered my interests in web development.
- Since 2015 I worked with PHP, JavaScript/TypeScript and React.
- Joined e2n and working as frontend developer.

The smart software for employee management

- We believe that digitalization is a key to success for companies.
- We optimize processes within employee management.
- We enable companies to save time and money with e2n.

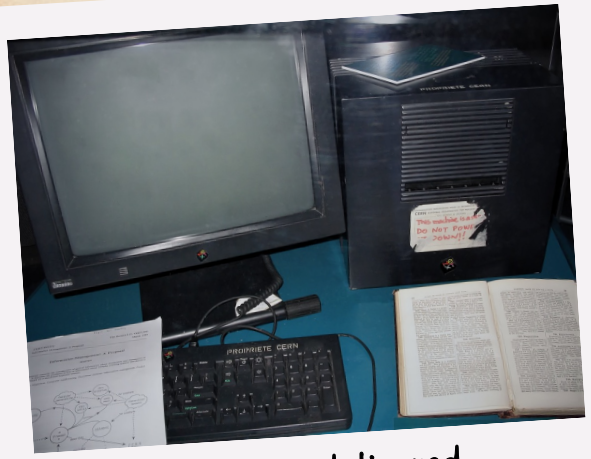


e2n

Why this talk?

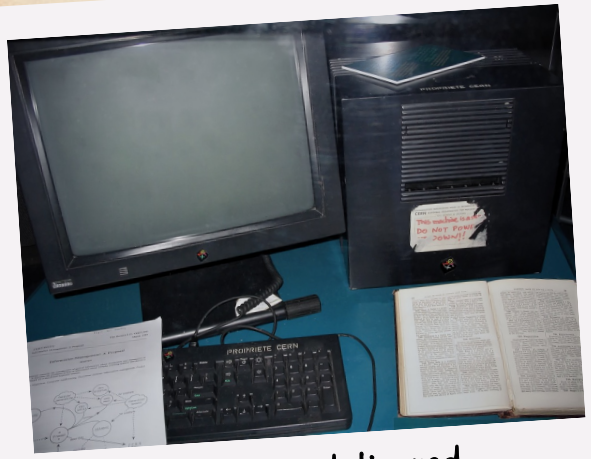
Tell me, how was web development in the past...

Tell me, how was web development in the past...



Static HTML was delivered by a server

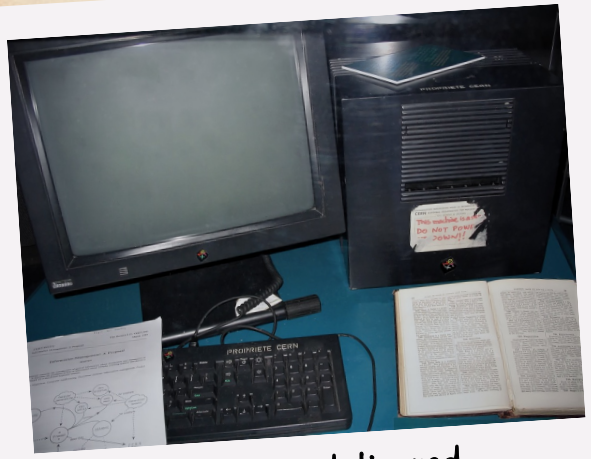
Tell me, how was web development in the past...



Static HTML was delivered by a server

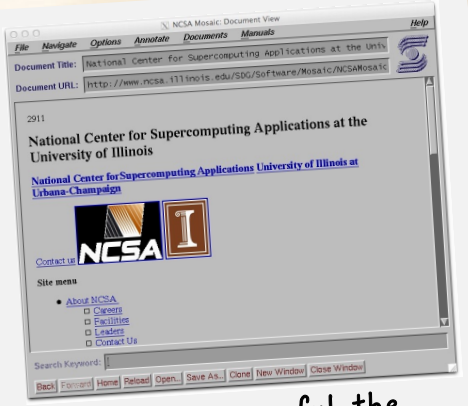
Feeding HTML with data using languages like PHP

Tell me, how was web development in the past...



Static HTML was delivered by a server

Feeding HTML with data using languages like PHP



Servers were powerful, the browser only displayed the document

Browsers got powerful and JavaScript got good...

Browsers got powerful and JavaScript got good...

Enable HTML manipulation, event handling and animations directly on the client

Browsers got powerful and JavaScript got good...

jQuery
write less, do more.

Enable HTML manipulation, event handling and animations directly on the client

Me, adding interactivity using jQuery

Browsers got powerful and JavaScript got good...

jQuery
write less, do more.

Enable HTML manipulation, event handling and animations directly on the client

Me, adding interactivity using jQuery

Dev News

May, 2013

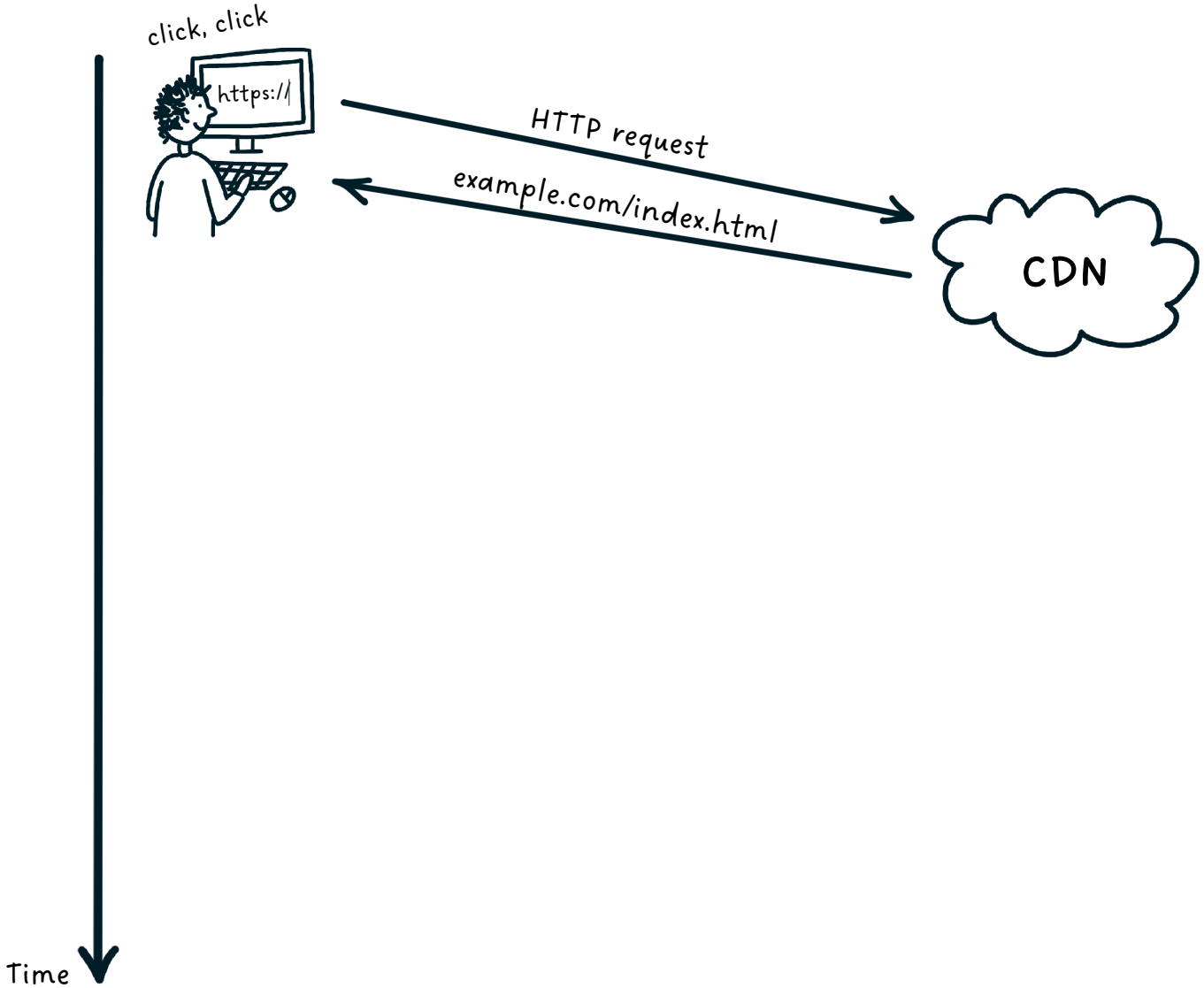
Big fail at JSConf

Facebook React is not getting much love at JsConf. Mixin HTML in JS seems like a bad idea.

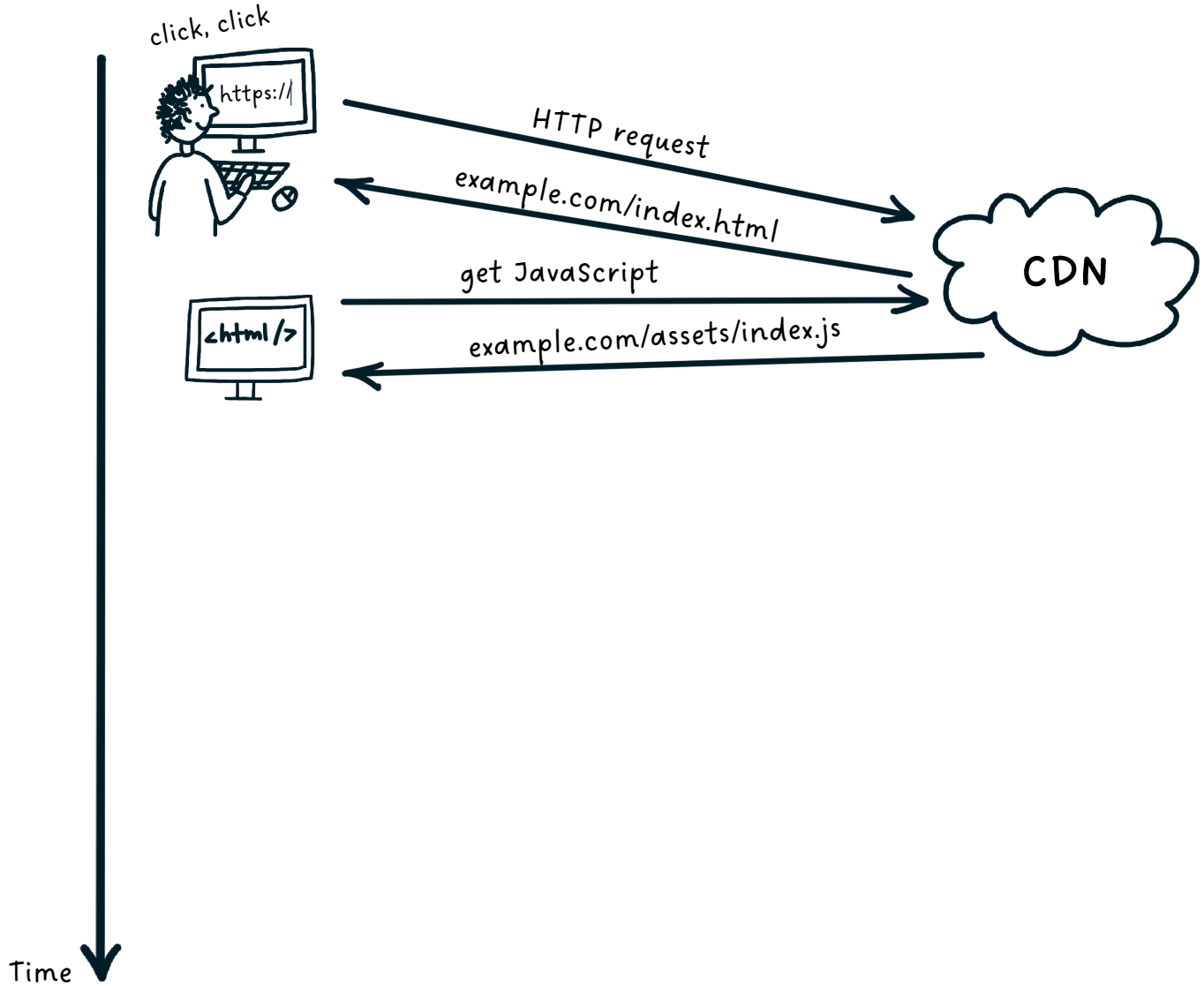
Since 2012 React, Angular and similar frameworks changed web development

How does a Single Page Application work?

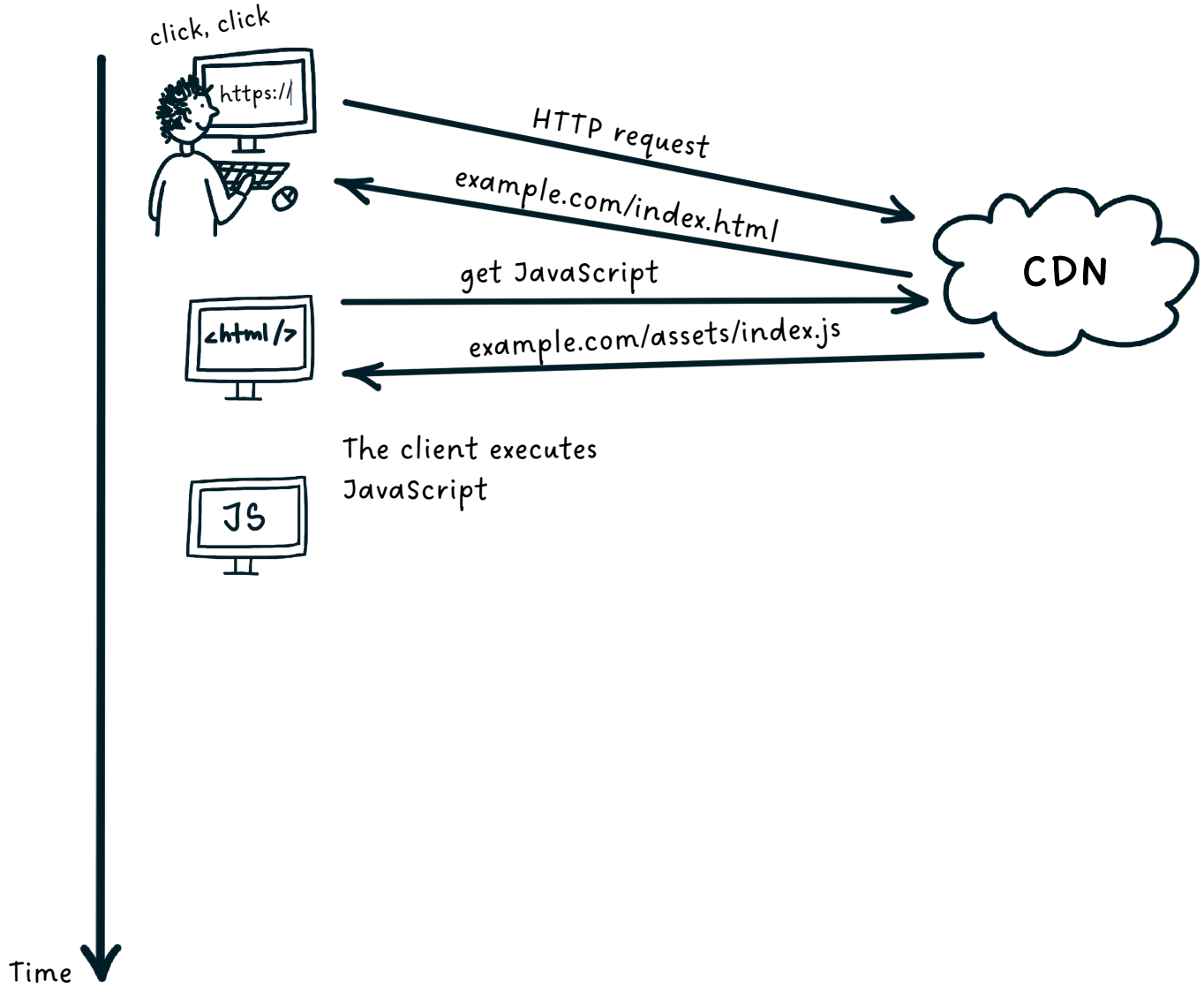
How does a Single Page Application work?



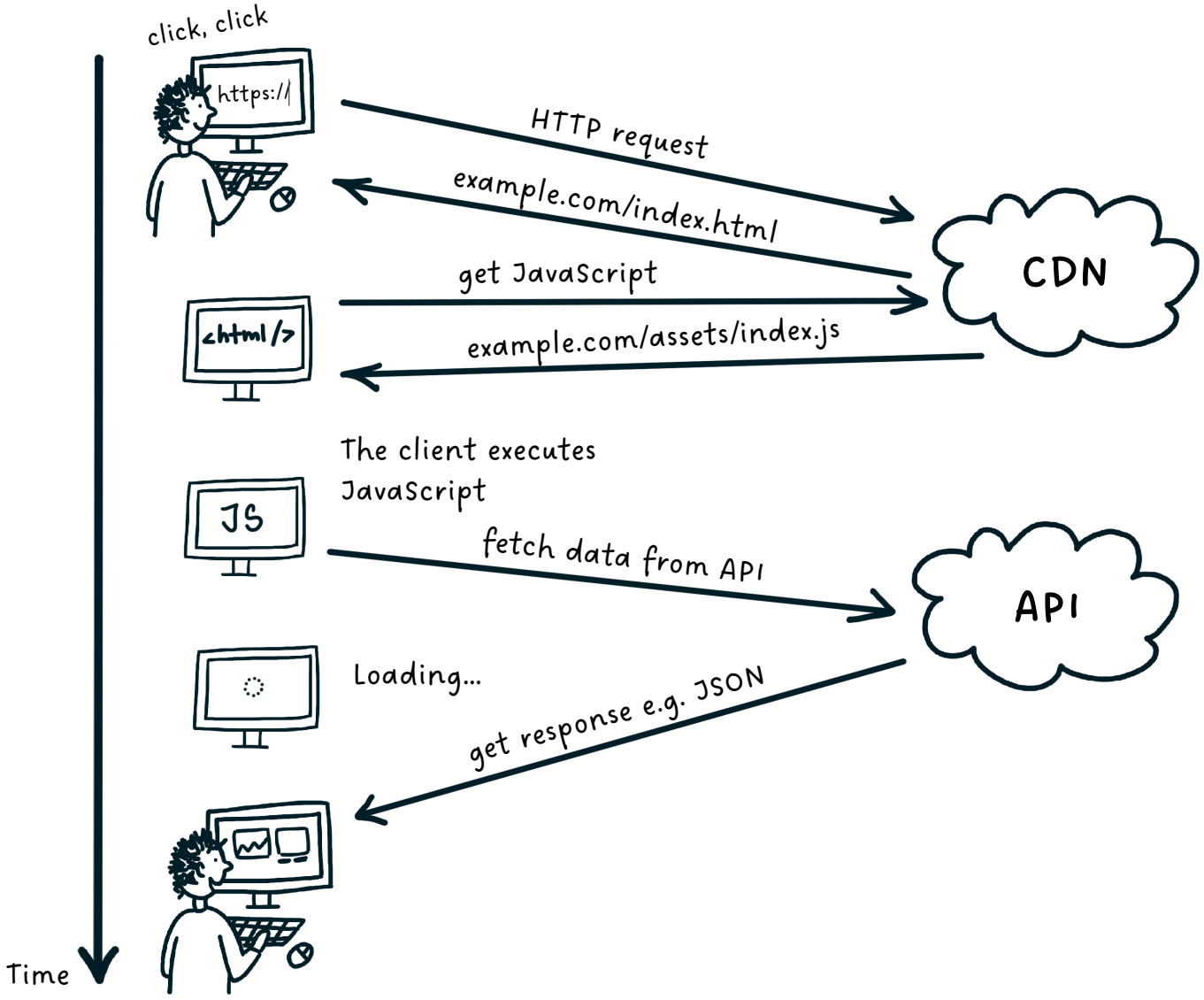
How does a Single Page Application work?



How does a Single Page Application work?

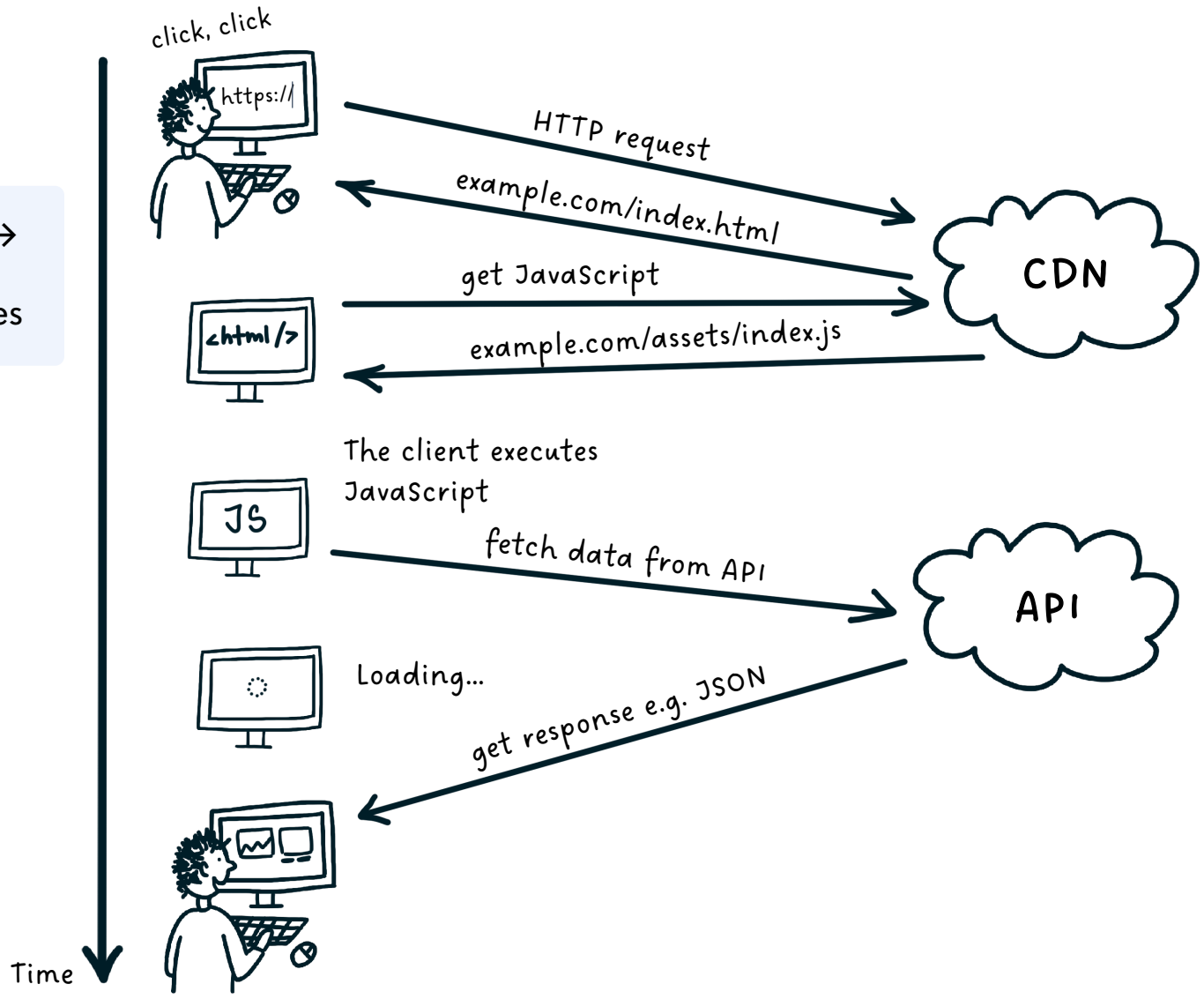


How does a Single Page Application work?



How does a Single Page Application work?

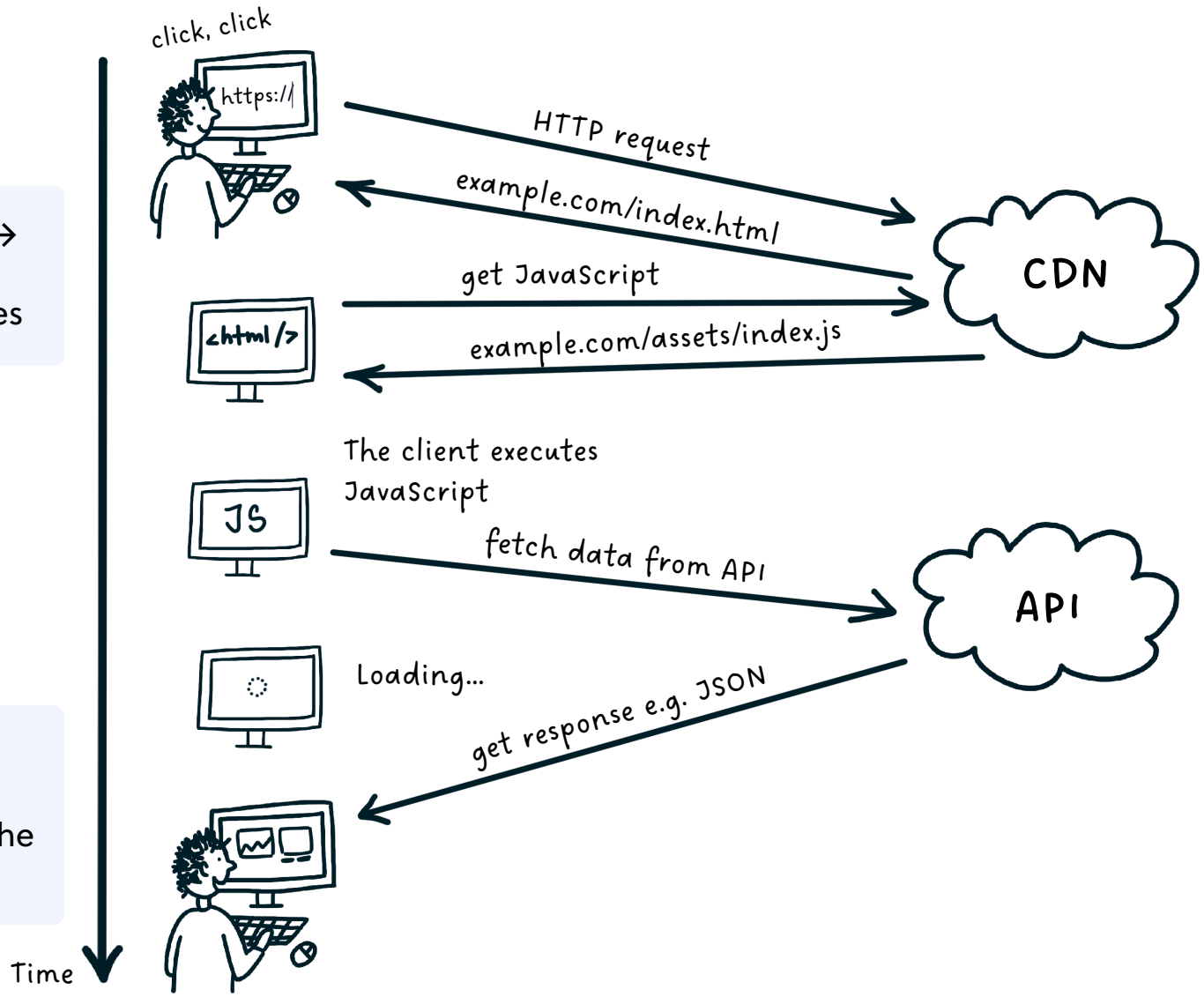
⚠ Single HTML-file → difficult to get metadata for routes



How does a Single Page Application work?

⚠ Single HTML-file → difficult to get metadata for routes

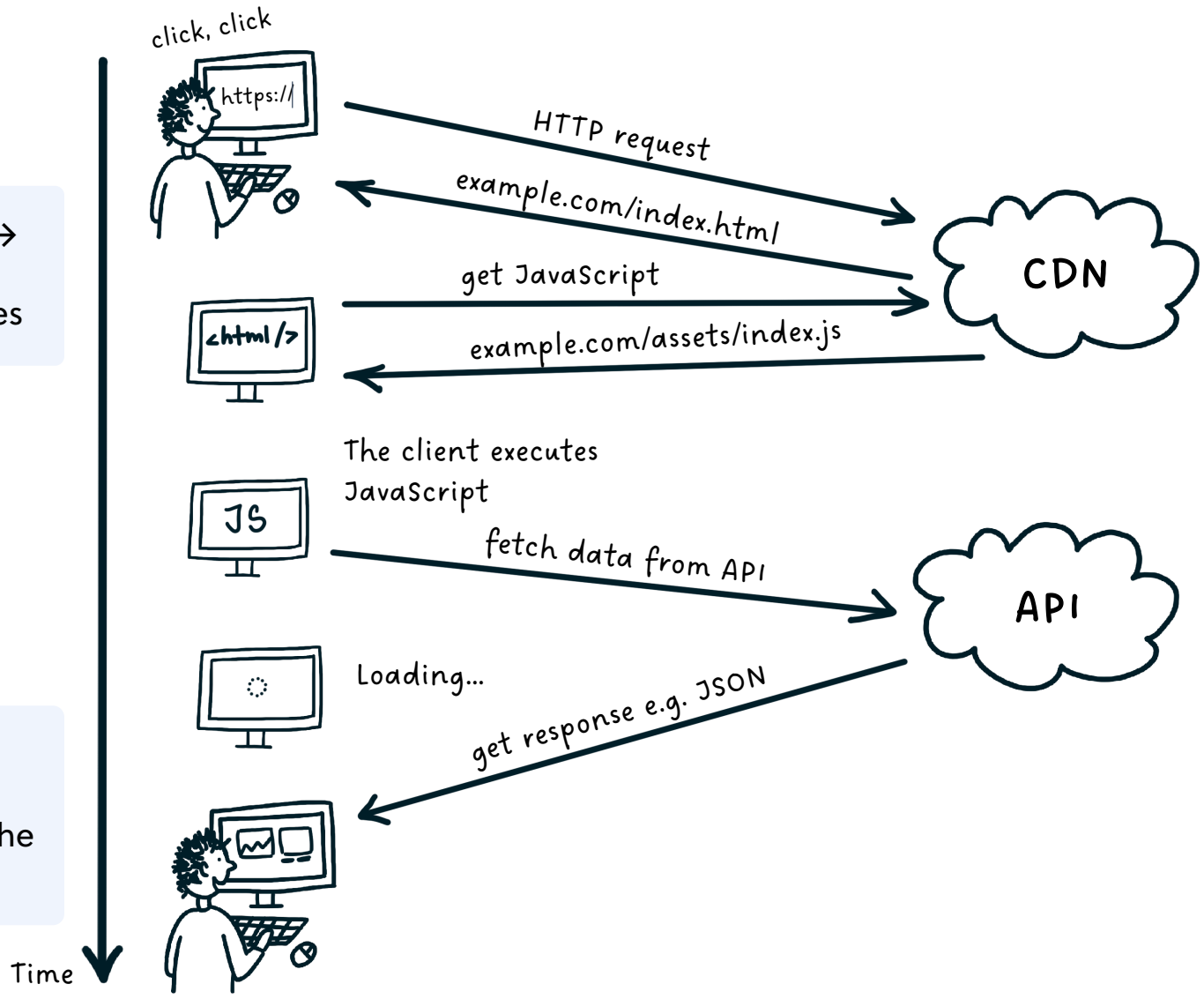
⚠ The client needs Javascript to view and interact with the page



How does a Single Page Application work?

⚠ Single HTML-file → difficult to get metadata for routes

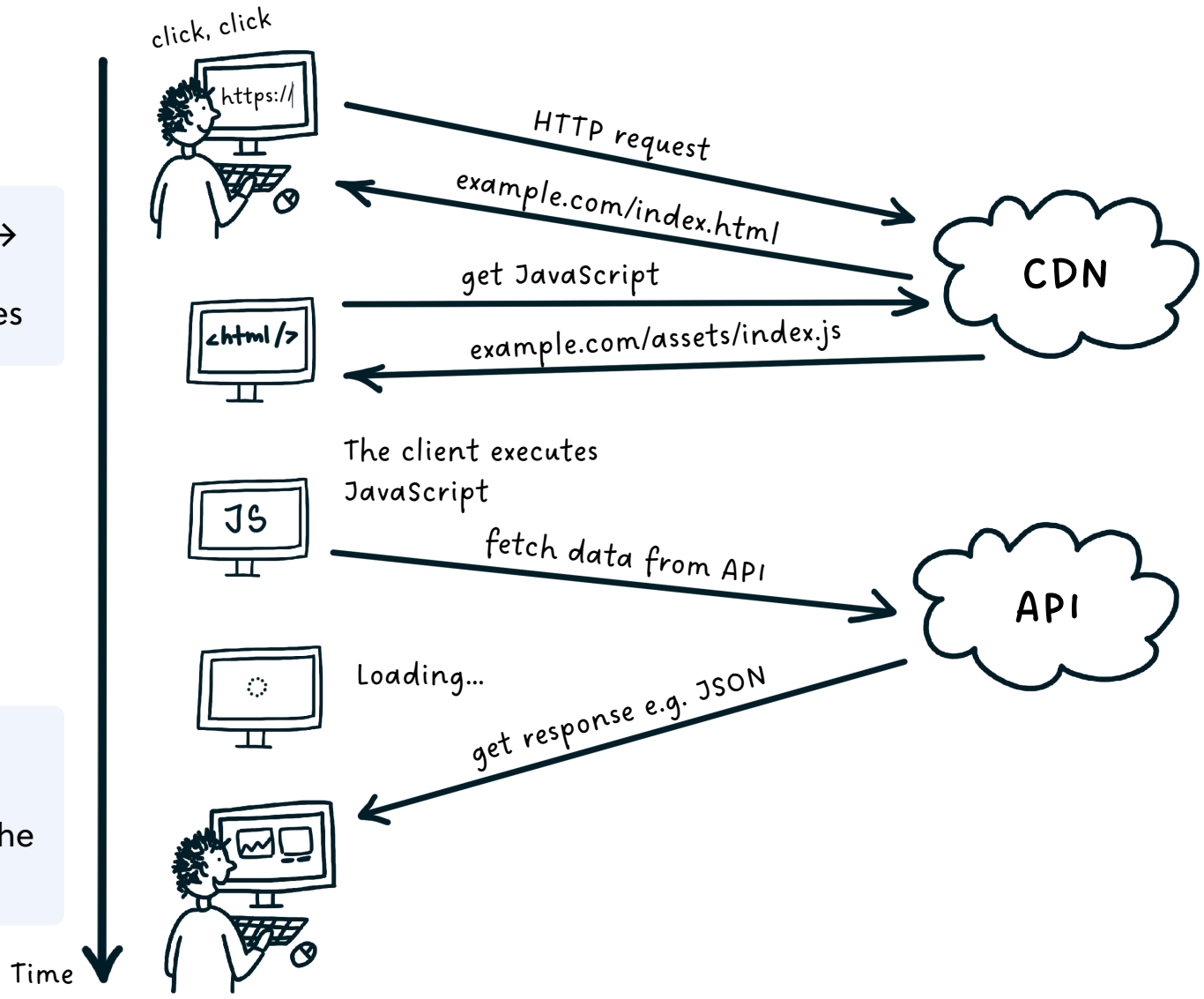
⚠ The client needs Javascript to view and interact with the page



⚠ The bigger a page, the longer it takes until the page is viewable and interactable

How does a Single Page Application work?

- Single HTML-file → difficult to get metadata for routes
- The client needs Javascript to view and interact with the page



- The bigger a page, the longer it takes until the page is viewable and interactable

? Back to the roots?

Not everything in the past was bad.

Combine ideas from back then with modern technologies.

Back to the roots with fullstack frameworks

Back to the roots with fullstack frameworks

- New frameworks in the JavaScript ecosystem: Next.js, React Remix, Nuxt, SvelteKit

Back to the roots with fullstack frameworks

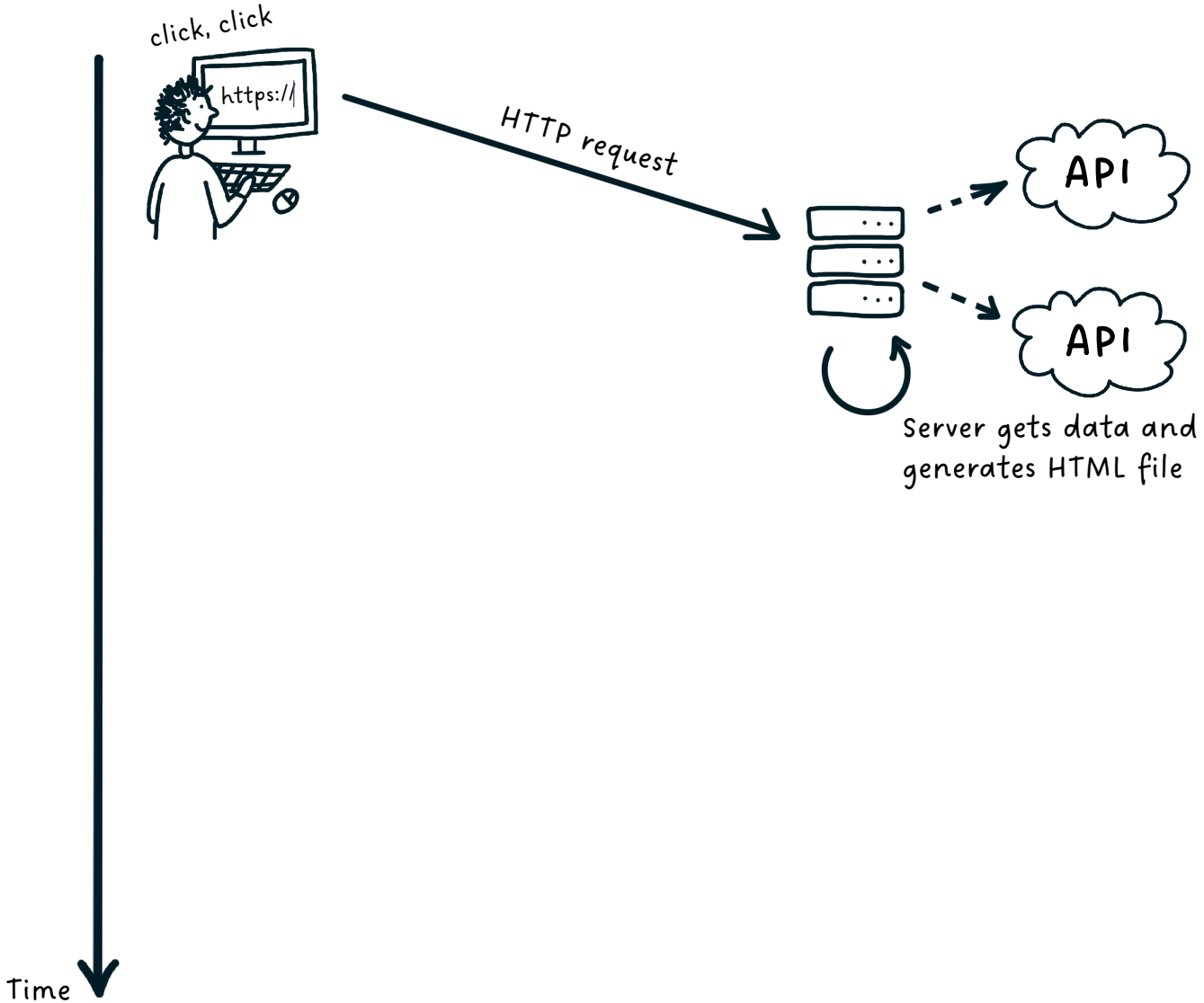
- New frameworks in the JavaScript ecosystem: Next.js, **React Remix**, Nuxt, SvelteKit
- Remix got open-sourced in 2021 by Micheal Jackson

Back to the roots with fullstack frameworks

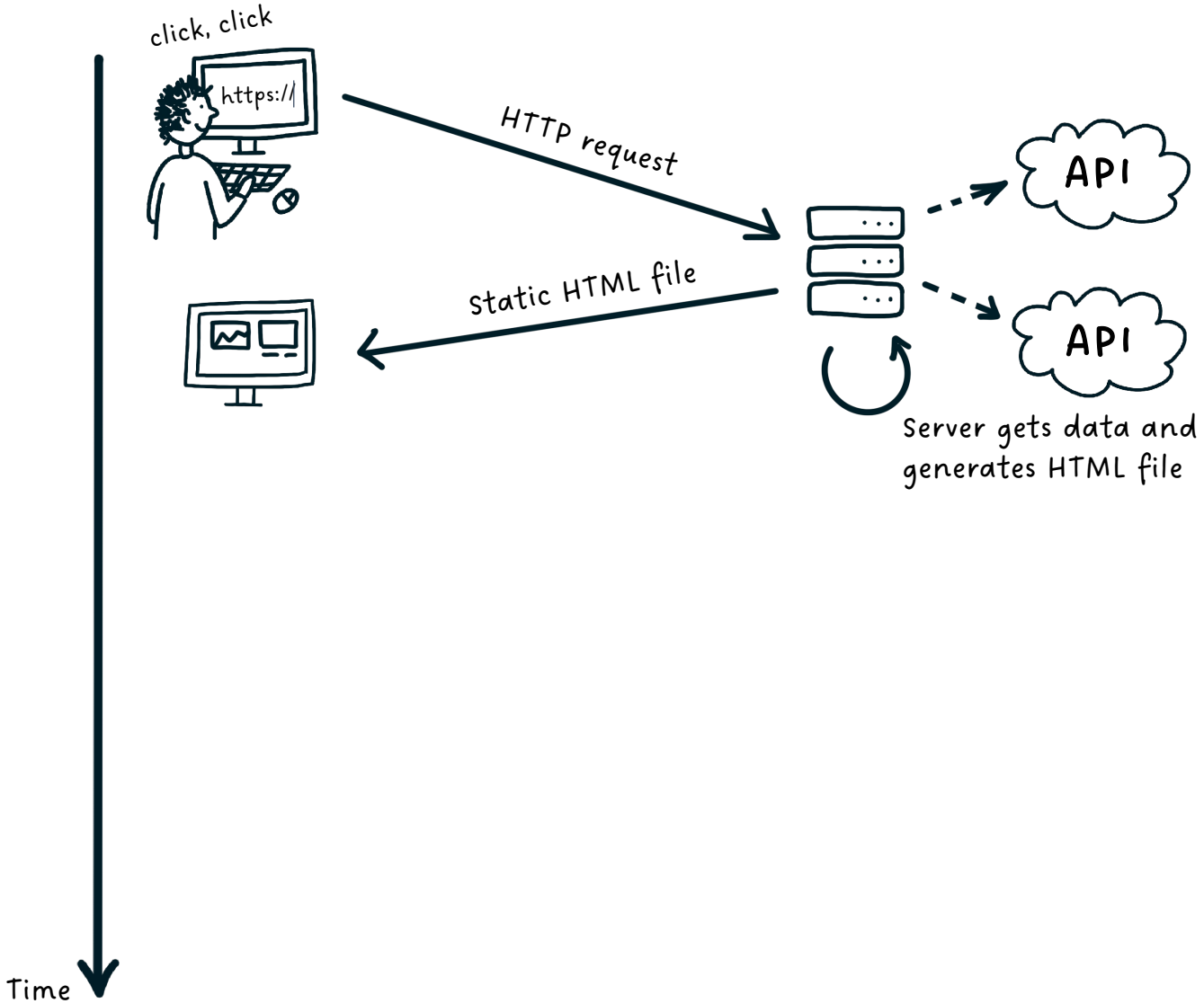
- New frameworks in the JavaScript ecosystem: Next.js, **React Remix**, Nuxt, SvelteKit
- Remix got open-sourced in 2021 by Micheal Jackson
- Progressive Enhancement and close to HTML5 standards

React Remix: Server-side rendering with Hydration

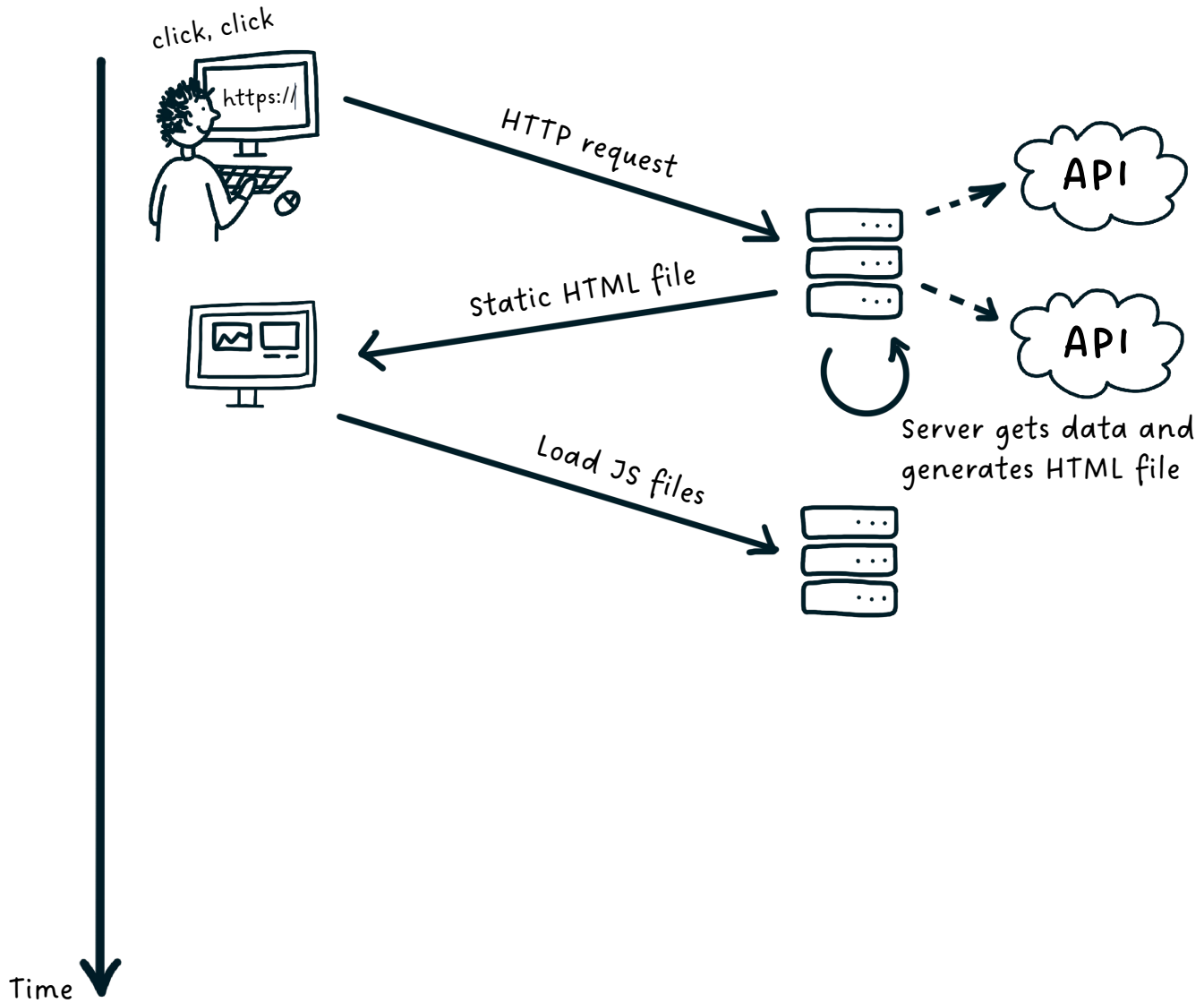
React Remix: Server-side rendering with Hydration



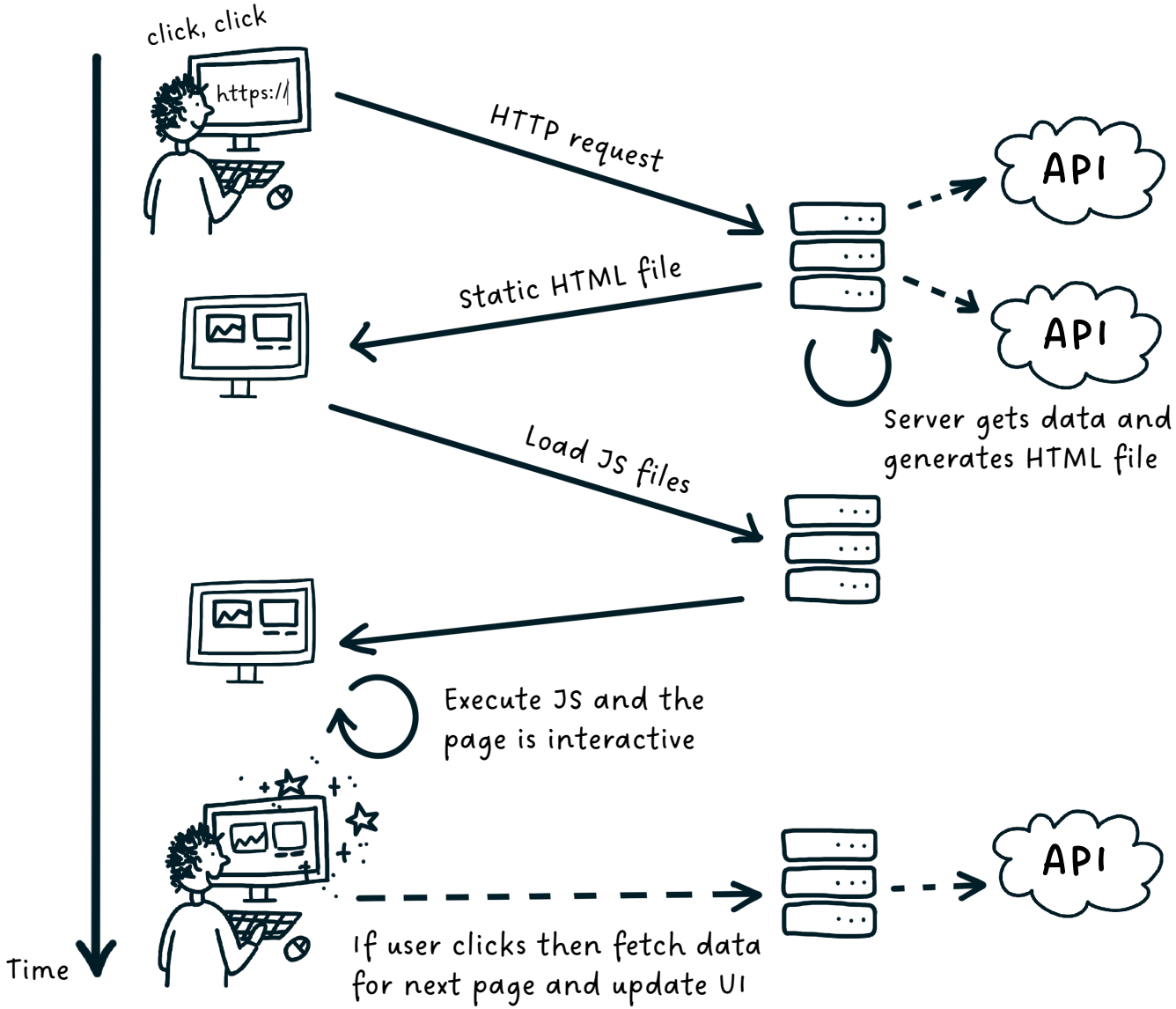
React Remix: Server-side rendering with Hydration



React Remix: Server-side rendering with Hydration



React Remix: Server-side rendering with Hydration



Example: Booking an event room

Meine Buchung

Gib den Wunschzeitraum für Deine Veranstaltung ein.
Prüfe mit einem Klick ob sie verfügbar ist.

Von

TT . MM . JJJJ , -- : --

Bis

TT . MM . JJJJ , -- : --

Verfügbarkeit prüfen

Impressum | Datenschutz

View on Github https://github.com/sonjafeitsch/meine-nische-booking

Meine Nische - Dashboard | Meine Nische – Buchung

localhost:3000

Lesezeichen importi... | Sonjas Dashboard -... | A simple strategy fo... | How to set up Amaz... | Dashboard Sidebar ... | Remix: Das neue Fr... | MFA not working wh... | A Beginners Guide t... | format dayjs utc obj...

Meine Buchung

Gib den Wunschzeitraum für Deine Veranstaltung ein.
Prüfe mit einem Klick ob sie verfügbar ist.

Von

TT . MM . JJJJ , -- : --

Bis

TT . MM . JJJJ , -- : --

Verfügbarkeit prüfen

Impressum | Datenschutz

Example: Booking an event room

View on Github https://github.com/sonjafeitsch/meine-nische-booking

Remix: Handle user interaction

Structure of _index.tsx

```tsx
function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();

  return (
    <>
      <Form method="post">
        <TextField label="Von" id="start" name="start" type="datetime-local" />
        <TextField label="Bis" id="end" name="end" type="datetime-local" />
        <Button type="submit">
          {state === "submitting"
            ? "Events werden gesucht…"
            : "Verfügbarkeit prüfen"}
        </Button>
      </Form>
      {!error && actionData && (
        <Events events={actionData} />
      )}
    </>
  );
}
```

```tsx
export async function action({ request }: ActionArgs) {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  if (!start || !end) {
    return json({ events: [] });
  }

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await getEvents(start, end);
  return json({ events });
}
```

Remix: Handle user interaction

Handle loading state of form

Render form

Structure of _index.tsx

```tsx
function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();

  return (
    <>
      <Form method="post">
        <TextField label="Von" id="start" name="start" type="datetime-local" />
        <TextField label="Bis" id="end" name="end" type="datetime-local" />
        <Button type="submit">
          {state === "submitting"
            ? "Events werden gesucht…"
            : "Verfügbarkeit prüfen"}
        </Button>
      </Form>
      {!error && actionData && (
        <Events events={actionData} />
      )}
    </>
  );
}
```

```tsx
export async function action({ request }: ActionArgs) {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  if (!start || !end) {
    return json({ events: [] });
  }

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await getEvents(start, end);
  return json({ events });
}
```

Remix: Handle user interaction

Handle loading state of form

Render form

action is called when form is submitted, form values are sent via POST method

Call another function on the server

Structure of _index.tsx

```tsx
function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();

  return (
    <>
      <Form method="post">
        <TextField label="Von" id="start" name="start" type="datetime-local" />
        <TextField label="Bis" id="end" name="end" type="datetime-local" />
        <Button type="submit">
          {state === "submitting"
            ? "Events werden gesucht…"
            : "Verfügbarkeit prüfen"}
        </Button>
      </Form>
      {!error && actionData && (
        <Events events={actionData} />
      )}
    </>
  );
}
```

```tsx
export async function action({ request }: ActionArgs) {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  if (!start || !end) {
    return json({ events: [] });
  }

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await getEvents(start, end);
  return json({ events });
}
```

Remix: Handle user interaction

Handle loading state of form

Render form

action is called when form is submitted, form values are sent via POST method

Call another function on the server

Structure of _index.tsx

```tsx
function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();

  return (
    <>
      <Form method="post">
        <TextField label="Von" id="start" name="start" type="datetime-local" />
        <TextField label="Bis" id="end" name="end" type="datetime-local" />
        <Button type="submit">
          {state === "submitting"
            ? "Events werden gesucht…"
            : "Verfügbarkeit prüfen"}
        </Button>
      </Form>
      {!error && actionData && (
        <Events events={actionData} />
      )}
    </>
  );
}
```

Client

```tsx
export async function action({ request }: ActionArgs) {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  if (!start || !end) {
    return json({ events: [] });
  }

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await getEvents(start, end);
  return json({ events });
}
```

Server

What makes a SPA so powerful?

State management: the glitter dust of an interactive SPA

The state is an important part of what makes the application dynamic and interactive.

State management: the glitter dust of an interactive SPA

The state is an important part of what makes the application dynamic and interactive.

And with Remix? 🤔

No more state management 😱

```tsx
export const action = async ({ request }: ActionArgs) => {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await createEvent(start, end);

  return redirect("/book?view=finished");
};

function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();
  const [searchParams] = useSearchParams();
  const view = searchParams.get("view") || "";

  return (
    <>
      {view !== "finished" ? (
        <Form method="post">
          <TextField
            label="Von"
            id="start"
            name="start"
            type="datetime-local"
          />
          <TextField label="Bis" id="end" name="end" type="datetime-local" />
          <Button type="submit">
            {state === "submitting"
              ? "Events werden gesucht…"
              : "Verfügbarkeit prüfen"}
          </Button>
        </Form>
      ) : (
        <div>Vielen Dank für Deine Anfrage.</div>
      )}
      {!error && actionData && <Events events={actionData} />}
    </>
  );
}
```

No more state management 😱

Possible Solution: store state of different views in URL

```tsx
export const action = async ({ request }: ActionArgs) => {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await createEvent(start, end);

  return redirect("/book?view=finished");
};

function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();
  const [searchParams] = useSearchParams();
  const view = searchParams.get("view") || "";

  return (
    <>
      {view !== "finished" ? (
        <Form method="post">
          <TextField
            label="Von"
            id="start"
            name="start"
            type="datetime-local"
          />
          <TextField label="Bis" id="end" name="end" type="datetime-local" />
          <Button type="submit">
            {state === "submitting"
              ? "Events werden gesucht…"
              : "Verfügbarkeit prüfen"}
          </Button>
        </Form>
      ) : (
        <div>Vielen Dank für Deine Anfrage.</div>
      )}
      {!error && actionData && <Events events={actionData} />}
    </>
  );
}
```

No more state management 😱

Possible Solution: store state of different views in URL

Read state from URL search params

```tsx
export const action = async ({ request }: ActionArgs) => {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await createEvent(start, end);

  return redirect("/book?view=finished");
};

function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();
  const [searchParams] = useSearchParams();
  const view = searchParams.get("view") || "";

  return (
    <>
      {view !== "finished" ? (
        <Form method="post">
          <TextField
            label="Von"
            id="start"
            name="start"
            type="datetime-local"
          />
          <TextField label="Bis" id="end" name="end" type="datetime-local" />
          <Button type="submit">
            {state === "submitting"
              ? "Events werden gesucht…"
              : "Verfügbarkeit prüfen"}
          </Button>
        </Form>
      ) : (
        <div>Vielen Dank für Deine Anfrage.</div>
      )}
      {!error && actionData && <Events events={actionData} />}
    </>
  );
}
```

No more state management 😱

Possible Solution: store state of different views in URL

Read state from URL search params

Other solutions:

- Cookies
- LocalStorage
- Remix offers helpers for form submission and validation error state

```tsx
export const action = async ({ request }: ActionArgs) => {
  const formData = await request.formData();
  const start = formData.get("start");
  const end = formData.get("end");

  invariant(typeof start === "string", "Startdatum muss ein Text sein.");
  invariant(typeof end === "string", "Enddatum muss ein Text sein.");

  const events = await createEvent(start, end);

  return redirect("/book?view=finished");
};

function IndexPage({ error }: { error?: string }) {
  const { state } = useNavigation();
  const actionData = useActionData<typeof action>();
  const [searchParams] = useSearchParams();
  const view = searchParams.get("view") || "";

  return (
    <>
      {view !== "finished" ? (
        <Form method="post">
          <TextField
            label="Von"
            id="start"
            name="start"
            type="datetime-local"
          />
          <TextField label="Bis" id="end" name="end" type="datetime-local" />
          <Button type="submit">
            {state === "submitting"
              ? "Events werden gesucht…"
              : "Verfügbarkeit prüfen"}
          </Button>
        </Form>
      ) : (
        <div>Vielen Dank für Deine Anfrage.</div>
      )}
      {!error && actionData && <Events events={actionData} />}
    </>
  );
}
```

Developing with Remix feels like web development 15 years ago

Developing with Remix feels like web development 15 years ago – but with more comfort

Developing with Remix feels like web development 15 years ago – but with more comfort

- Easy setup & easy deployment
- Good framework for simple web apps & SEO optimized applications
- Simply add some code around the existing "server side view" without changing how it works fundamentally
- Differs from classic React and you have to reconsider common patterns

A group of developers discussing wheather they should implement a single page application or a server side application • generated by Midjourney

Closing thoughts

- Fullstack frameworks add more variety to develop applications
- Consider type of application: static site, mostly static or dynamic web app
- What are the use cases and who should bear the costs?