Weniger Komplexität, mehr Effizienz

Moderne Web-Apps mit Server Side Rendering

Raphael Zimmermann
Ergon Informatik AG
18.09.2024

I ❤️ the Web

I 💔 the "modern" Web

Registration

Name:

Email:

Phone:

Address:

Submit

```html
<form method="POST" action="/submit-form">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name" required>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

  <label for="phone">Phone:</label>
  <input type="tel" id="phone" name="phone" pattern="[0-9]{10}" required>

  <label for="address">Address:</label>
  <textarea id="address" name="address" required></textarea>

  <input type="submit" value="Submit">
</form>
```

```tsx
import React, { useState, FormEvent } from 'react';

function CustomerForm() {
  const [name, setName] = useState<string>('');
  const [email, setEmail] = useState<string>('');
  const [phone, setPhone] = useState<string>('');
  const [address, setAddress] = useState<string>('');
  const [outcome, setOutcome] = useState<string>('');

  const handleSubmit = async (event: FormEvent<HTMLFormElement>): Promise<void> => {
    event.preventDefault();

    const formData = {
      name,
      email,
      phone,
      address,
    };

    const response = await fetch('/submit-form', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
    });

    if (response.ok) {
      setOutcome('Form submitted successfully!');
    } else {
      setOutcome('Error submitting form.');
    }
  }

  return (
    <div>
      {outcome && <p>{outcome}</p>}
      <form onSubmit={handleSubmit}>
        <label htmlFor="name">Name:</label>
        <input type="text" id="name" name="name" value={name} onChange={(event) => setName(event.target.value)} required />

        <label htmlFor="email">Email:</label>
        <input type="email" id="email" name="email" value={email} onChange={(event) => setEmail(event.target.value)} required />

        <label htmlFor="phone">Phone:</label>
        <input type="tel" id="phone" name="phone" pattern="[0-9]{10}" value={phone} onChange={(event) => setPhone(event.target.value)} required />

        <label htmlFor="address">Address:</label>
        <textarea id="address" name="address" value={address} onChange={(event) => setAddress(event.target.value)} required></textarea>

        <button type="submit">Submit</button>
      </form>
    </div>
  );
}

export default CustomerForm;
```

Browser Feature neu implementiert

Browser Feature neu implementiert

Identisches "HTML" + Event-Handler

Server Side Rendering =

weniger Komplexität

schnellere Entwicklung

gleichwertige UX

Agenda

- Wie sind wir überhaupt da gelandet?
- Server vs. Client-Generated Views
- HTML over the Wire Ansatz
- Zusammenfassung

Das Web um ~2007

- Weniger Dynamik
- Dynamische Inhalte mit Flash
- Full Page Reload
- Problem: Browser Kompatibilität

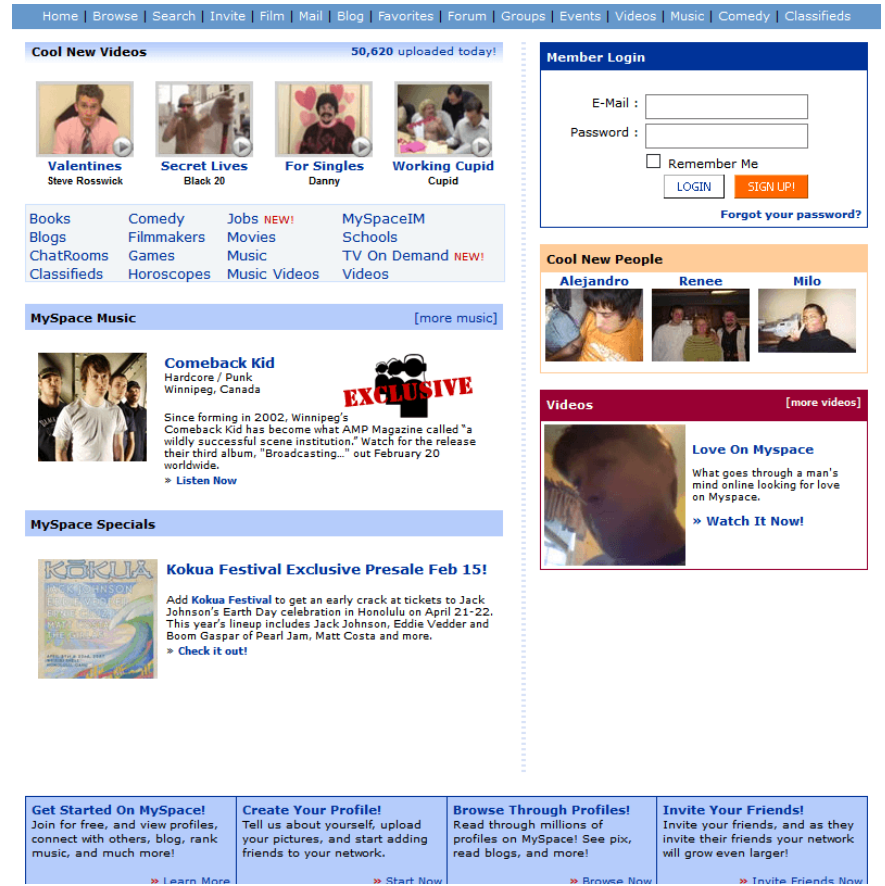
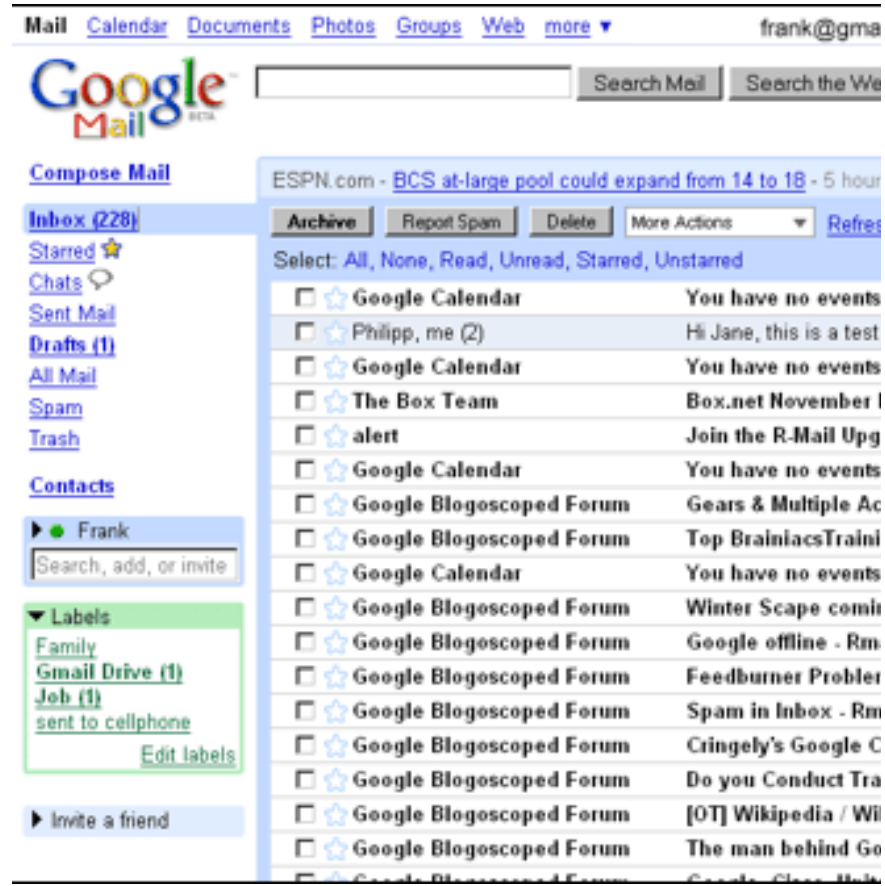


Bild Quelle: webdesignmuseum.org

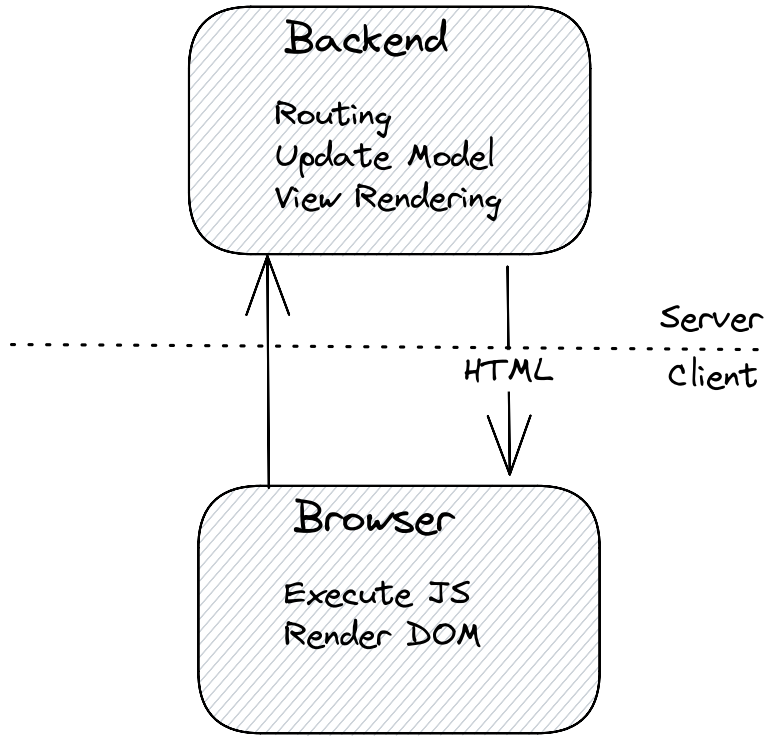
Neue Generation von Webanwendungen

- Mehr Dynamik erwünscht
- Insbesondere Partial Page Updates
- Ajax wird Populär
- Neue Probleme

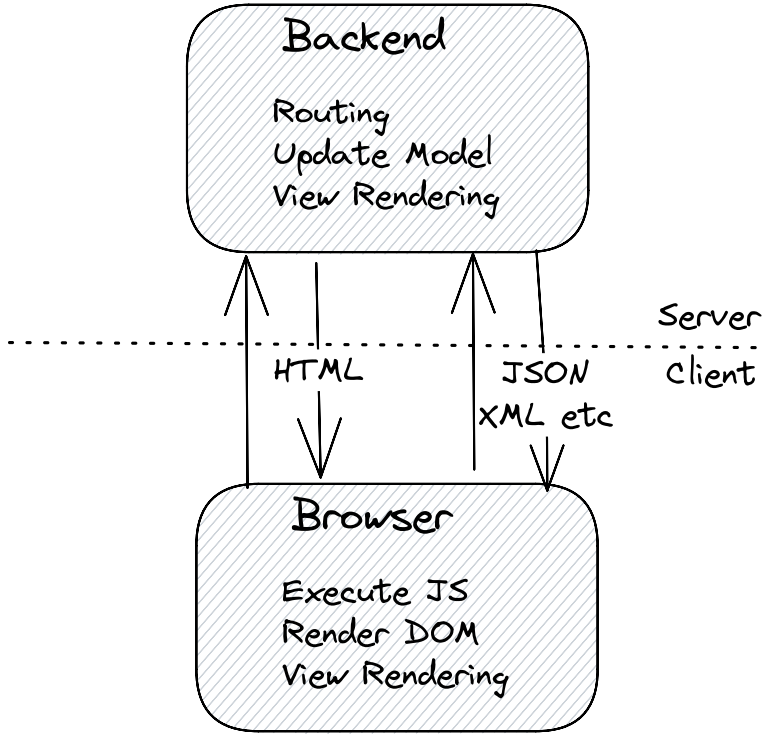


Quelle: blogoscoped.com

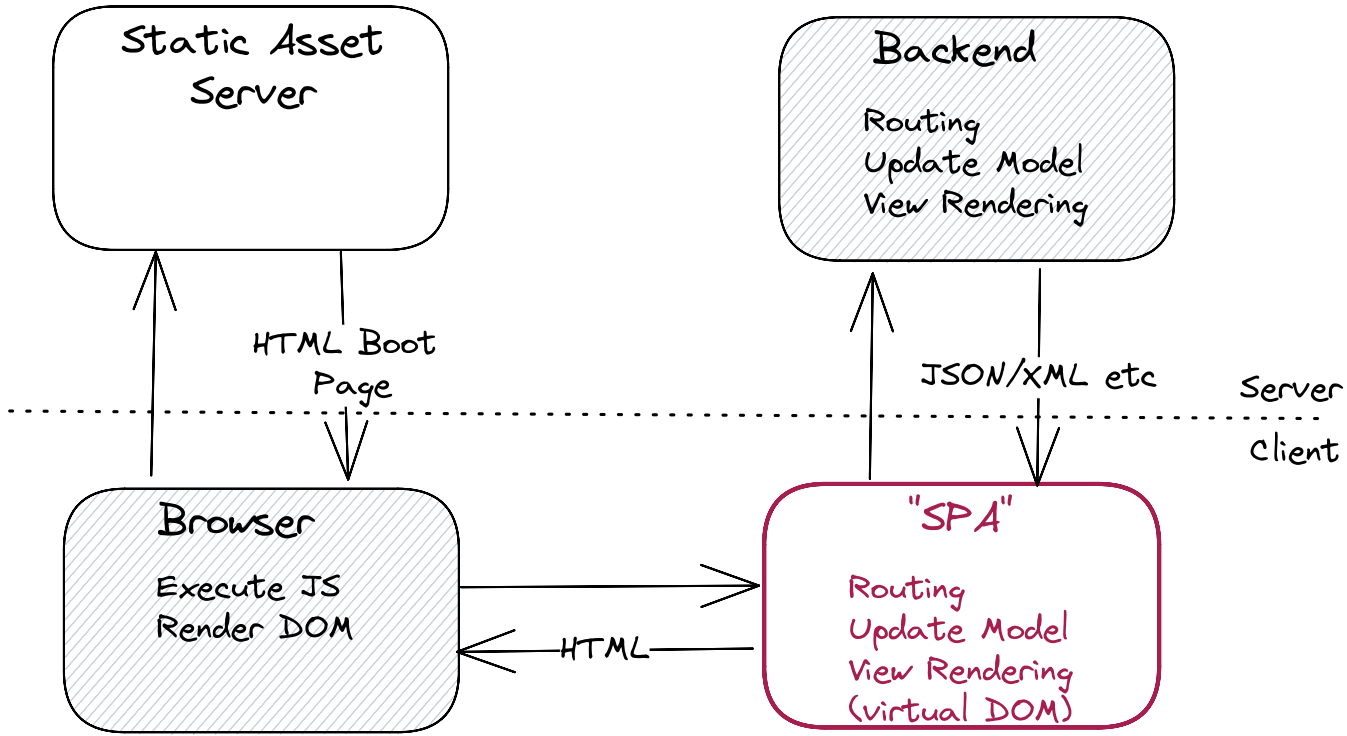
Server-Generated Views



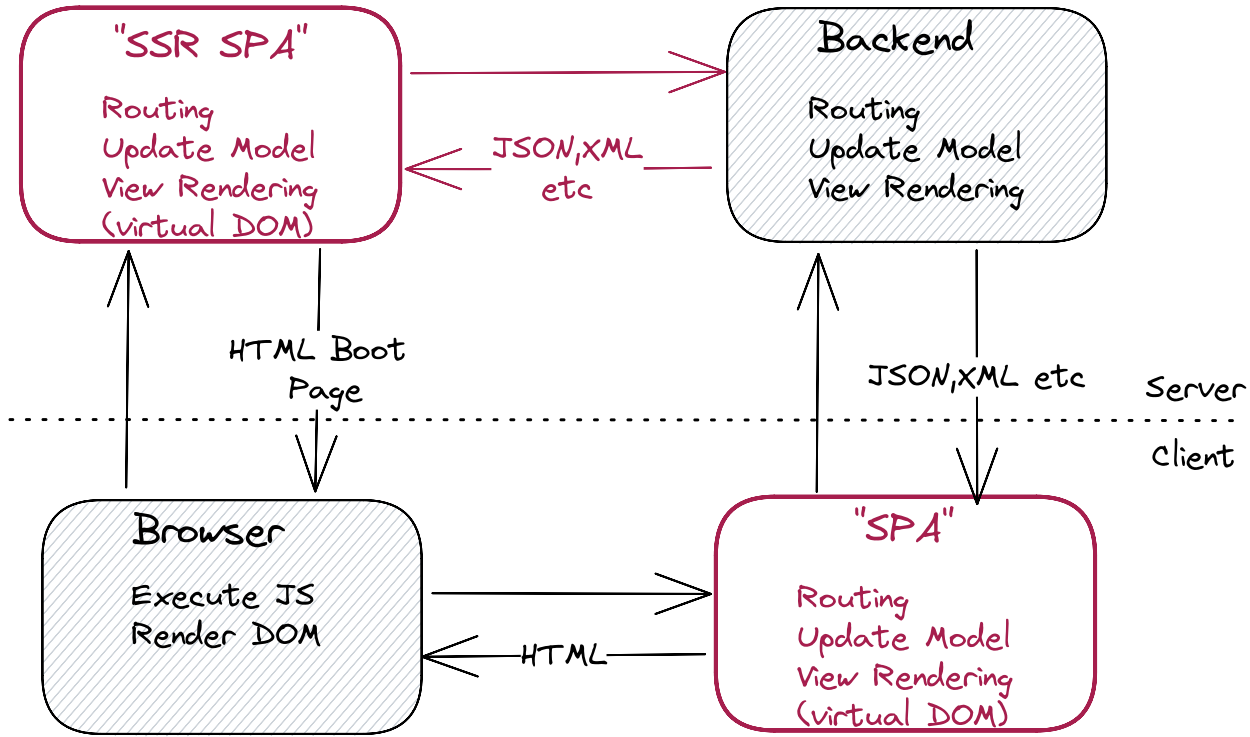
Server-Generated Views + Ajax



Client-Generated Views



Pre-Rendered Client-Generated Views

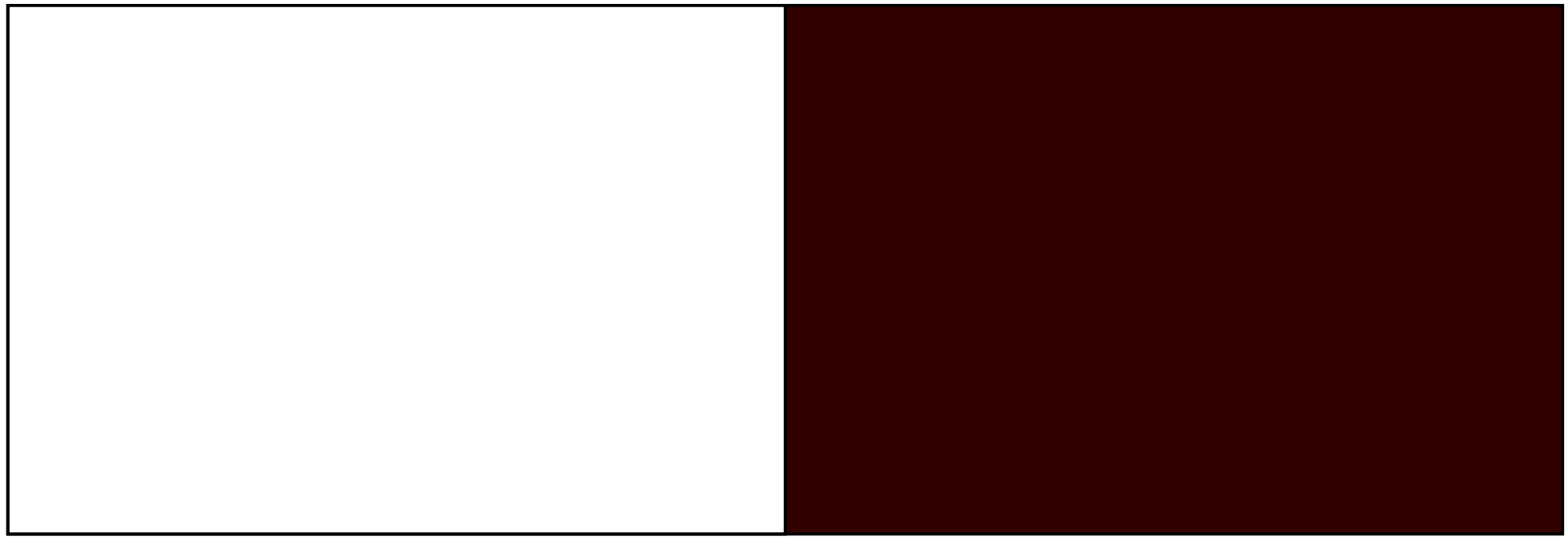


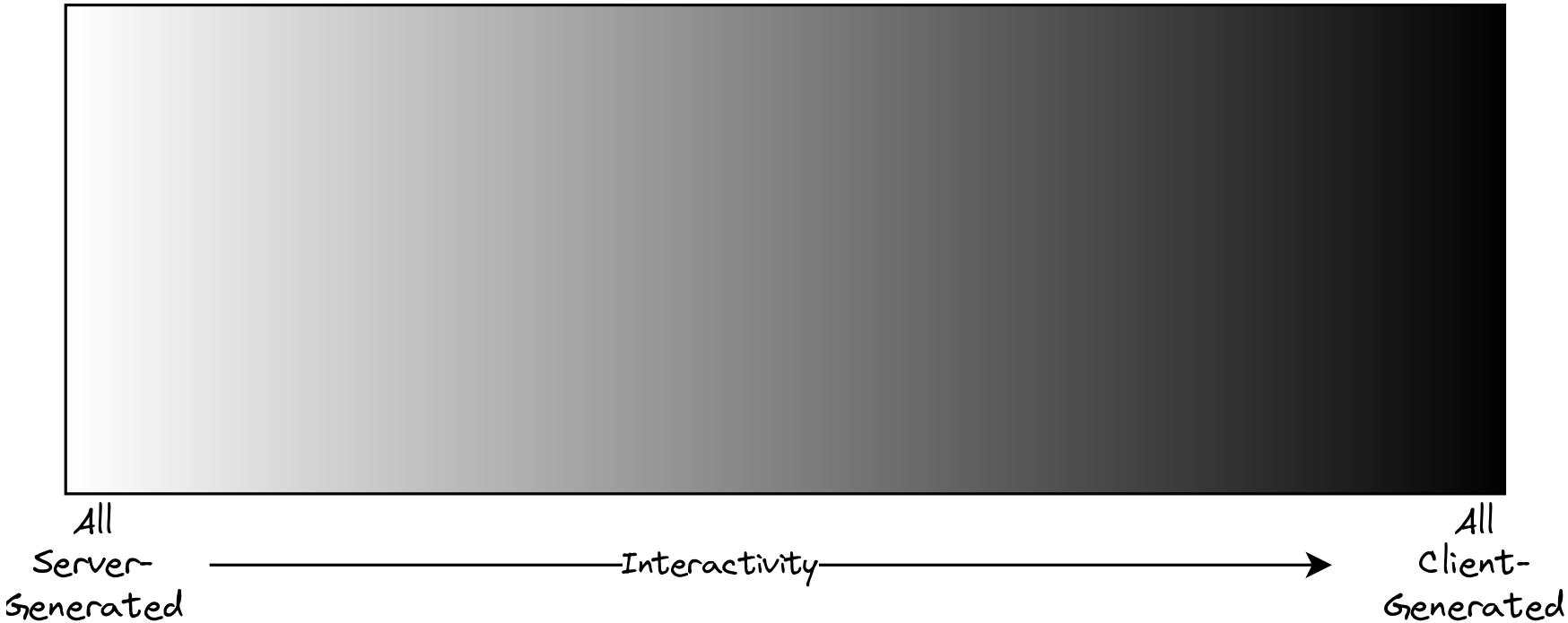
Herausforderungen von Client-Generated Views

- Initial Page Load ist langsam
- SEO nach wie vor nicht so gut
- Benötigt viel spezifisches Know-How
  - um Routing, Forms, Reactivity "richtig" zu machen
- "Wasserfall"-Netzwerk calls
- Viel Last auf Client Geräte
- **Frontend Cache**

Zwei Codebasen

- 2 Deployments
- 2 Sprachen / Ökosysteme
- Fördert Trennung von Frontend und Backend
- Definition von API-Contracts
- Benötigt mehr Entwickler
- Kostet alles Zeit und Geld 💸


All
Server-
Generated
All
Client-
Generated

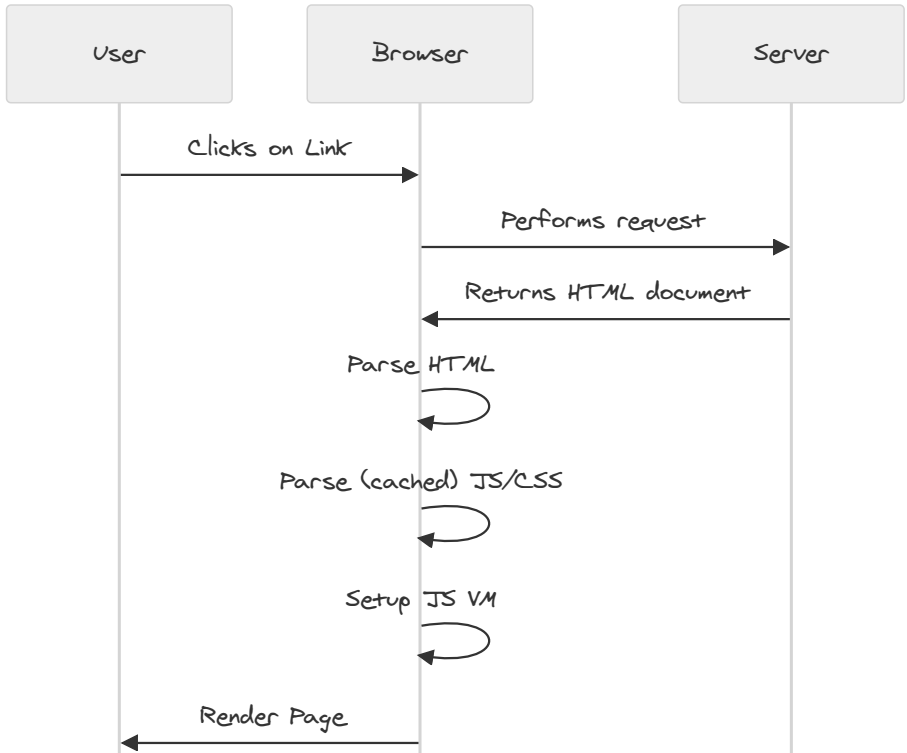

All
Server-
Generated
Interactivity
All
Client-
Generated

HTML Over The Wire

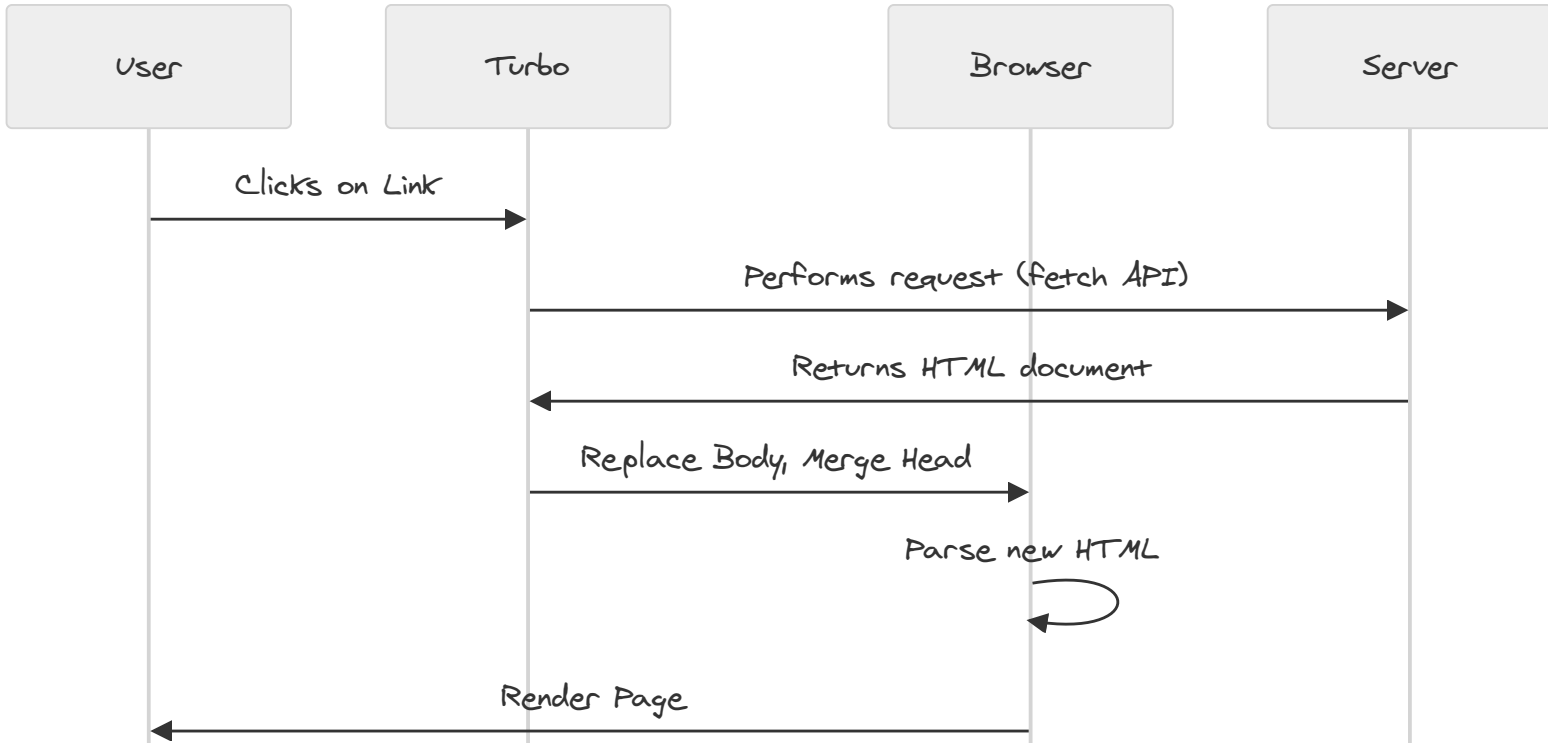
- Wir generieren all unser HTML auf dem Server
- senden HTML anstelle von JSON/XML zum Client

Hotwired Turbo

Navigation without Turbo Drive (Full Page Reload)

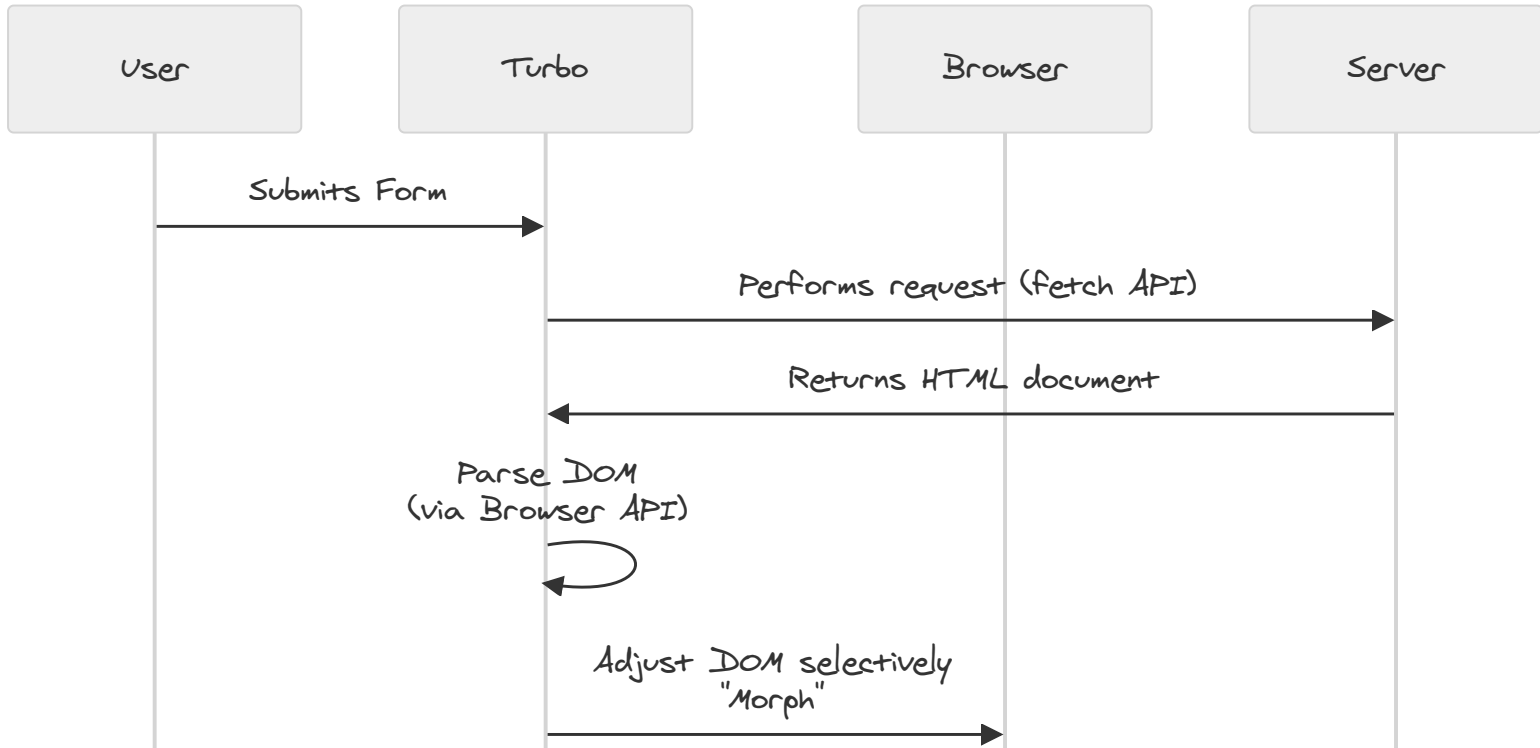


Navigation with Turbo Drive

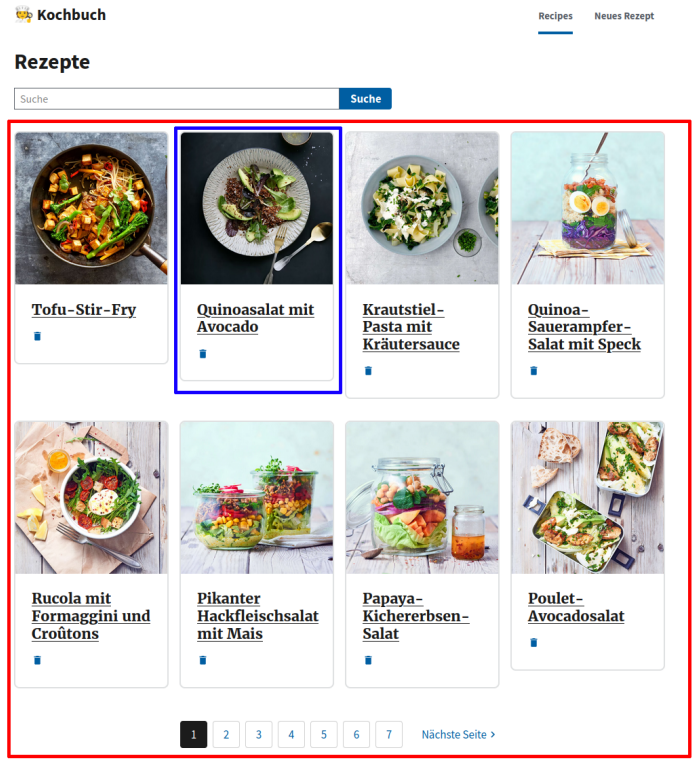


Demo

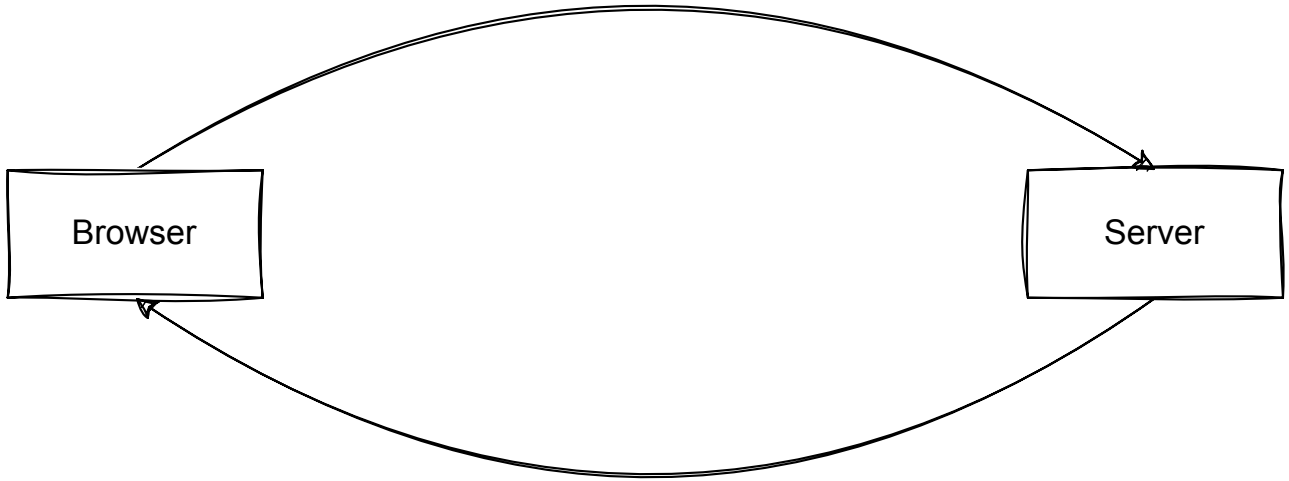
Turbo Morph

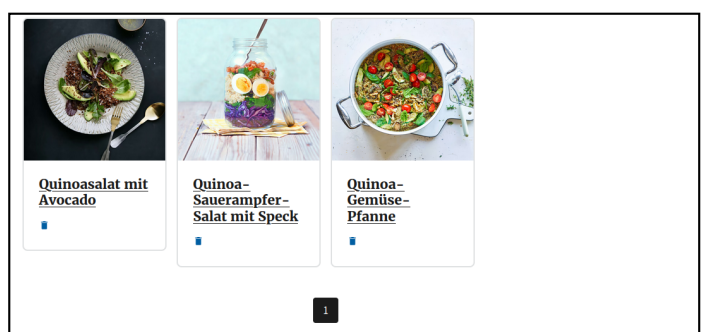

User
Turbo
Browser
Server
Submits Form
Performs request (fetch API)
Returns HTML document
Parse DOM
(via Browser API)
Adjust DOM selectively
"Morph"

Turbo Frames



Quinoa Suche





It’s just HTML

```
<turbo-frame id="recipe-list">
  <div>...</div>
</turbo-frame>
<turbo-frame id="recipe-42">
  <img src=".." />

</turbo-frame>
```

Demo

Server Push mit Turbo Streams

Client

```
const eventSource = new EventSource("/events");
Turbo.session.connectStreamSource(eventSource);
```

Server

```
sseService.send(event.sseEmitterId, renderHtml())
```

HTML

```
<turbo-stream action="replace" target="recipe-42">
  <template>
    <img src="..."/>
    ...
```

Demo

Interaktivität

Beispiel: Password Visibility

Stimulus.js: HTML

```
<div data-controller="password-visibility">
  <input type="password" data-password-visibility-target="input" spellcheck="false" />

  <button type="button" data-action="click->password-visibility#toggle">
    <span data-password-visibility-target="icon">Eye</span>
    <span data-password-visibility-target="icon" class="hidden">Eye Slash</span>
  </button>
</div>
```

Quelle: stimulus-components.com

Stimulus.js: JavaScript

```
// <… data-controller="password-visibility" …>
stimulus.register("password-visibility", class extends Controller {
    static targets = [
      "input",  // <… data-password-visibility-target="input" …>
      "icon",   // <… data-password-visibility-target="icon" …>
    ];

    connect() {
      this.hidden = this.inputTarget.type === "password";
    }

    toggle(e) { // <… data-action="password-visibility#toggle" …>
      e.preventDefault();

      this.inputTarget.type = this.hidden ? "text" : "password";
      this.hidden = !this.hidden;

      this.iconTargets.forEach((icon) => icon.classList.toggle("hidden"));
    }
  }
);
```

Quelle: stimulus-components.com

Server-Generated Views

➕ HTML rendern ist einfach

➕ Eine Codebasis

➕ Weniger Komplexität

➕ **einfaches Ausführungs-Modell**

👉 **Erhöht Produktivität**

➖ Fehlende (Component) Libraries

- 👉 `Web Components`
- 👉 `CSS Frameworks`

➖ Libraries müssen auch gut verstanden werden

➖ Fehlende Framework Integrationen

Wann soll ich jetzt was einsetzen?

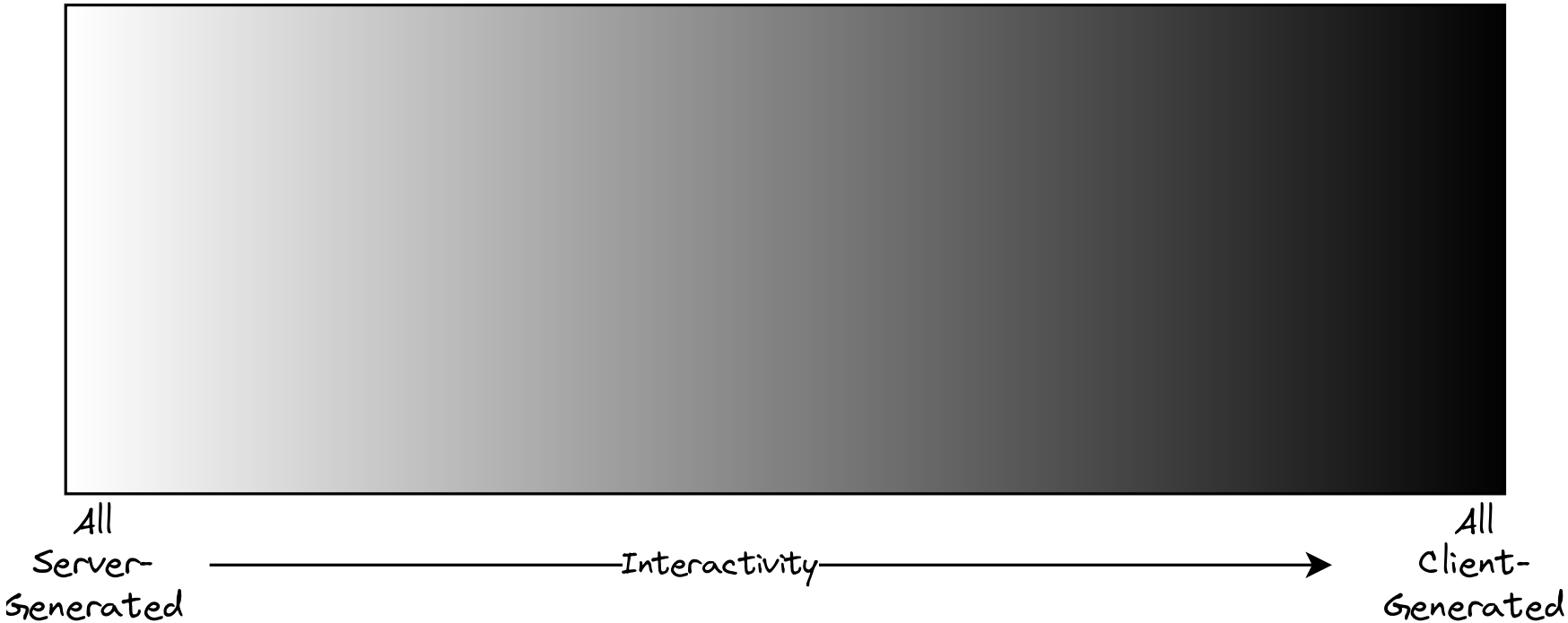
Server-Generated

- Default!
- Auch wenn API vorhanden
  (Bsp. Mobile Apps)

Client-Generated

- E2E Encryption (Bsp. Threema, Whatsapp etc.)
- 100% Offline fähig
- Sehr sehr komplexe Komponenten (Bsp. Figma)
- Organisatorische Trennung von Frontend und Backend

Zusammenfassung



🙋‍♂️ hi@raphael.li 👨‍💼 raphael.zimmermann@ergon.ch raphiz

Weiterführende Resourcen

- Hotwire - Offizielle Webseite
- Demo Anwendung auf GitHub
- RailsConf 2016 - Turbolinks 5: I Can’t Believe It’s Not Native! by Sam Stephenson
- Hotwire Turbo Chat with Micronaut Views
- Hotwire Weekly Newsletter
- Resource-oriented Client Architecture - ROCA
- Hypermedia Systems
- Modern web apps without JavaScript bundling or transpiling
- JavaScript modules - Performance recommendations (v8.dev)

CSS Frameworks

- US Design System (USWDS)
- Red Hat Patternfly
- Canonical Vanilla
- GitHub Primer
- Franken UI
- HyperUI
- …

… und viele Weitere

Web Component Libraries

- Shoelace
- Lion
- IBM Carbon
- Adobe Spectrum
- …