Zurück in die Zukunft mit statischen Webseiten

Ein Blick auf das Astro Framework

Timo Zander (timozander.de)

enterJS 2023

Was?

Web-Framework mit Fokus auf Performance und statischen Inhalten

+ UI-Frameworks

Static Site Generators (SSG)

NEXT.JS

Nuxt

SVELTEKIT

Remix

11ty

Gridsome

astro

Warum Astro?

- 🗒️ Fokus auf Inhalts-lastigen Webseiten
- 🚀 Hohe Performance durch null Client-JavaScript
- 🏝️ Interaktivität dank Islands-Architektur
- 🧩 Bring your own Framework

Islands Architektur

Interaktive Elemente
werden **nachgeladen**

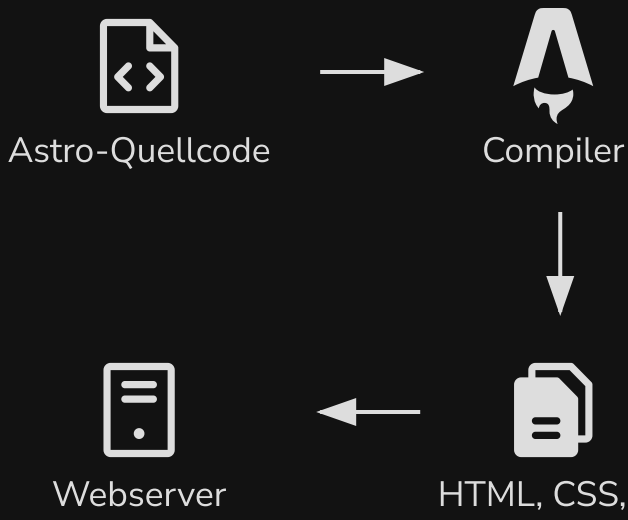
Statische Elemente sind
sofort verfügbar

MPA-Architektur

	Single-Page Application (SPA)	Multi-Page Application (MPA)
Rendering	Client	Server
Navigation	Client	Server
State-Management	Client	Server
Client-JavaScript	viel	wenig bis keins

Rendering Modi von Astro

Static Site Generation (SSG)



Server-side Rendering (SSR)

Astro in der Praxis

Ein neues Projekt mit Astro erstellen

Ein Astro-Projekt zum Leben erwecken

```
$ pnpm create astro@latest
```

oder online auf astro.new

Starten des Entwicklungs-Servers mit `pnpm dev`

Ein neues Astro Projekt

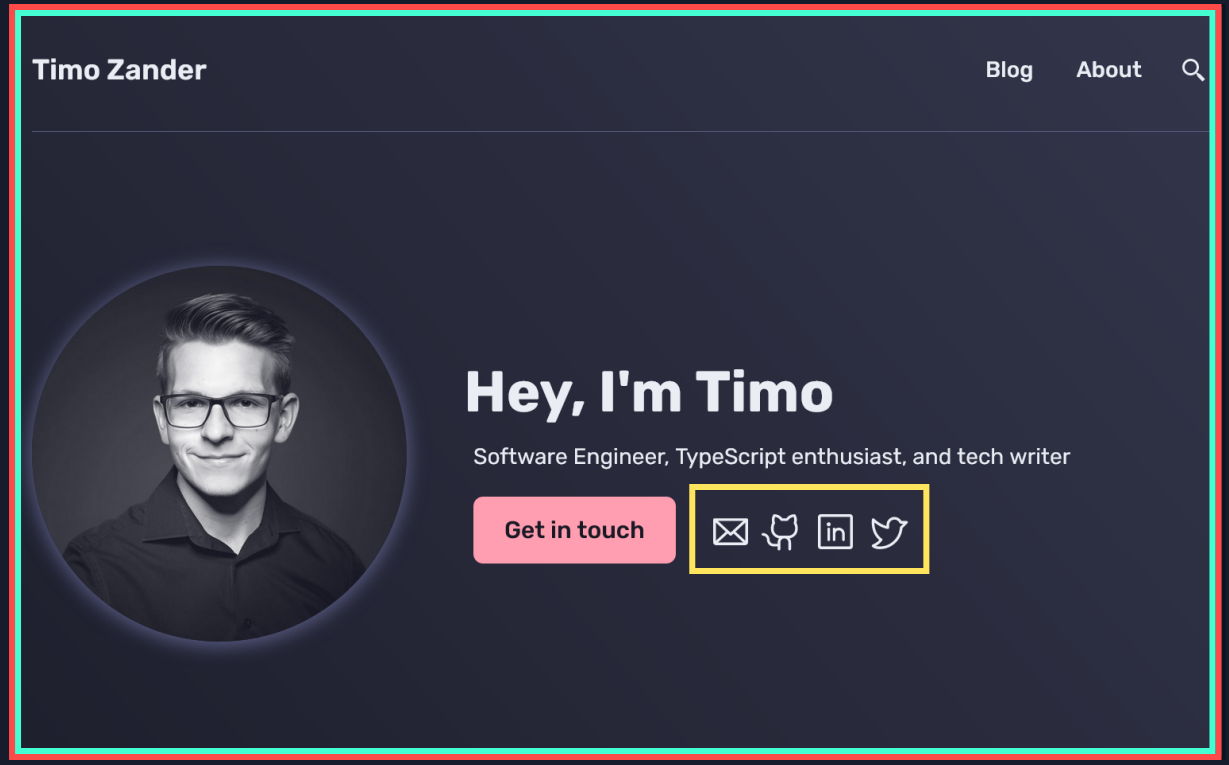
```
├── README.md
├── astro.config.mjs
├── node_modules/
├── package.json
├── pnpm-lock.yaml
├── public/
│   └── ...
├── src/
│   └── ...
└── tsconfig.json
```

Das `src` Verzeichnis

```
├ src
  ├── components
  │   └── Card.astro
  ├── layouts
  │   └── Layout.astro
  ├── pages
  │   └── index.astro
  └── env.d.ts
```

Astro-Webseiten bestehen aus **Komponenten**, **Layouts** und **Seiten**.

Anatomie einer Astro Seite



- Seite
- Layout
- Komponente

Astro Komponenten

src/components/Date.astro

```
---
const date = new Date();
---

<p>Heute ist {date.toLocaleDateString()}</p>
```

src/components/LinkButton.astro

```astro
---
export type ButtonType = "primary" | "default";
interface Props {
  href: string;
  type?: ButtonType;
  disabled?: boolean;
  target?: HTMLAttributeAnchorTarget;
}
const { href, type = "default", disabled = false, target = undefined } = Astro.props;
---
<a
  type="button"
  href={disabled ? "#" : href}
  tabindex={disabled ? "-1" : "0"}
  class={`button-${type}`}
  target={target}
>
  <slot />
</a>

<style>
/* ... */
</style>
```

Props in Astro-Komponenten

```
  <LinkButton
    href={email.href}
    type="primary"
    className={"cta-button"}
    target="_blank"
  >
    Get in touch
  </LinkButton>
```

Props sind **typisiert**

```
---
// ...
const {
  href, type = "default",
  disabled = false, target = undefined
} = Astro.props;
---
<a
  type="button"
  href={disabled ? "#" : href}
  tabindex={disabled ? "-1" : "0"}
  class={`button-${type}`}
  target={target}
>
  <slot />
</a>
```

Frontend-Frameworks nutzen

src/pages/index.astro

```
---
import MyVueComponent from './components/MyComponent.vue';
---

<div>
  <MyVueComponent />
</div>
```

Komponenten werden **automatisch** statisch gerendert

Pages und Navigation

Die index.astro Page

```
---
import Layout from "../layouts/Layout.astro";
import Date from "../components/Date.astro";
---

<Layout title="Welcome to Astro.">
  <main>
    <h1>
      Welcome to <span class="text-gradient">Astro</span>
    </h1>

    <Date />
  </main>
</Layout>
```

File-based routing

Der **Dateiname** bestimmt die **URL** der Seite

```
├── pages/
│   ├── index.astro        → /
│   ├── about.md           → /about
│   ├── blog/
│   │   ├── index.astro    → /blog
│   │   └── [slug].astro   → /blog/:slug
│   └── rss.xml.ts         → /rss.xml
```

Islands-Architektur angewandt

Client-seitiges JavaScript mit Astro

`script` in Komponenten

```
---
console.log("Hello aus dem Build!")
---

<button>Test</button>

<script>
  console.log("Hello aus dem Browser!")
</script>
```

Verarbeitung von client-side JavaScript

1. Importe werden ge-bundled
2. Das JavaSkript wird zum HTML-Head hinzugefügt
3. Skripte werden **nicht** gedoppelt
4. TypeScript wird unterstützt

Skripte "pur" verwenden

```
---
console.log("Hello aus dem Build!")
---

<button>Test</button>

<script is:inline>
  console.log("Hello aus dem Browser!")
</script>
```

`is:inline` Skript-Tags werden **nicht** vom Build verarbeitet

Client-Rendering mit Direktiven steuern

```
---
// Wichtig: keine .astro Komponente
import LongTask from "../components/LongTask.vue";
---

<LongTask client:load />

<LongTask client:idle />

<p class="spacer">Scroll down</p>

<LongTask client:visible />
```

Dynamische Inhalte und statische Seiten

Content Collections

```
- src/pages/
  ├── newsletter/
  │   ├── new-subcriber.md
  │   ├── featured-links-may-2023.md
  │   └── ...
  ├── blog/
  │   ├── welcome-post.md
  │   ├── what-i-learned-2022-post.md
  │   └── ...
  └── authors/
      ├── zander.json
      └── ...
```

Einträge typisieren

```ts
// src/content/config.ts
import { z, defineCollection } from 'astro:content';

const blogCollection = defineCollection({
  type: 'content', // oder 'data'
  schema: z.object({
    title: z.string(),
    tags: z.array(z.string()),
    image: z.string().optional(),
  }),
});

export const collections = {
  blog: blogCollection,
};
```

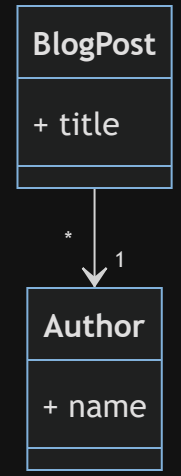
Inhalte aus Collections nutzen

```
import { getCollection } from 'astro:content';

const blogPosts = await getCollection('blog');
```

```
<section>
  <h2>Blog-Einträge</h2>
  {
    blogPosts.map((post) ⇒ (
      <a href={"blog/" + post.slug}>{post.data.title}</a>
    ))
  }
</section>
```

Daten referenzieren

```
const blog = defineCollection({
  type: 'content',
  schema: z.object({
    title: z.string(),
    author: reference('authors'),
    relatedPosts: z.array(reference('blog')),
  })
});

const authors = defineCollection({
  type: 'data',
  schema: z.object({
    name: z.string(),
  })
});
```



> Jedem Anfang wohnt ein Zauber inne.

- Loslegen unter astro.new
- Nutzt Astro für euren Blog, ein Portfolio, …
- …und **nicht** für interaktive Dashboards und co
- Lernt das Framework zu lieben (oder zu hassen!)

Alle Unterlagen findet Ihr auf timozander.de