HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

CLOUD
COMPUTING

Metaframeworks (Astro)

Prof. Dr.-Ing. Thorsten Weiss

Version 1.0

Meta-Frameworks

CLOUD COMPUTING

Wichtige Meta-Frameworks

Framework	Library / Basis	Rendering-Modi	API-Handling	Performance	DX / Community	Besonderes	Nachteile
Next.js	React / JS/TS	SSR, SSG, ISR, CSR	Ja (API-Routen)	Hängt an React Performance	Sehr groß, gut dokumentiert	ISR (Incremental Static Regeneration) Vercel-optimiert	Build-Komplexität, Performance, Abstraktion.
Nuxt	Vue / JS/TS	SSR, SSG, Hybrid, (SPA)	Ja (Server API)	Gut, etwas träger bei großen Projekten	Groß, Vue-Community	Auto-Imports, File-based Routing, Nuxt UI Pro	Langsamere Builds
SolidStart  Beta Status	SolidJS / TS	SSR, SSG, Streaming, Edge, SPA	Ja, mit Node oder reinen API-Calls.	Sehr schnell (Finegrained reactivity)	Klein, aktiv wachsend	Extrem kleiner Output, moderne Architektur, server-first	Noch instabil, kleine Community, keine Stable Seeds.
SvelteKit	Svelte / JS/TS	SSR, SSG, SPA	Ja	Sehr schnell, kein virtueller DOM	Mittelgroß, sehr bleibt.	Wenig Boilerplate, reactive Syntax	Neue Lernkurve, Ökosystem im Aufbau
Astro 	Framework-agnostisch (JS/TS + JSX/Vue/Svelte/etc.)	SSG, SSR, Partial Hydration	Server functions sind eingeschränkt, aber Node nutzbar.	Sehr schnell für statische Seiten	Wächst stark, gute Docs	Inselarchitektur, Multi-Framework-Support	Nicht ideal für interaktive Webapps

SSR = Server-Side Rendering

SSG = Static Site Generation

CSR = Client-Side Rendering

ISR = Incremental Static Regeneration

Astro – File Based Routing

CLOUD COMPUTING

File-Based Routing

- In SolidJS haben wir die Umschaltung der Seite mit einem Solid-Router erledigt. Dies ist für SPA auch die gängige Vorgehensweise.
- Sollen jedoch SSR und SSG integriert werden, so werden diese nicht wie bei SPA als ganze Apps oder Chunks ausgeliefert, sondern für jede Seite extra. Um hierbei ein einfaches Handling zu ermöglichen, bietet sich **File-Based Routing** an.

UI-Routen Beispiel: SolidStart

- Die Mechanismen sind bei den Meta-Frameworks sehr ähnlich.
- Die Routen ergeben sich aus dem Ordnernamen.
- Beispiel die Route /about ruft die Komponente about.jsx auf. Diese kann je nach Anforderung SSR oder SSG sein.

Datei / Ordner	URL-Pfad
src/pages/index.jsx	/ (Startseite)
src/pages/about.jsx	/about
src/pages/blog/index.jsx	/blog
src/pages/blog/[slug].jsx	/blog/[beliebig]
src/pages/[...404].jsx	Catch-All für 404

- Die Route [slug].jsx steht für „irgendeine Route“, sprich einen Platzhalter. Hier kann z.B. ein Blogeintrag eingefügt werden.

API-Routen

- Astro wie auch die meisten anderen Meta-Frameworks bieten API-Routen an. Damit können serverless-Funktions, die über Routen angesprochen werden umgesetzt werden. Bei kleineren einfachen Projekten, können so API-Calls auch ohne extra Node/Bun/etc. Server aufgebaut werden.
- Beispiel: Aufruf durch /api/person – wenn person.js im ordner /routes/api/ liegt.

```
const dbPath = path.resolve("./src/database/", 'database.db');
console.log(dbPath);

export const GET: APIRoute = async ({ params, request }) => {
  let db = new sqlite(dbPath);
  let personsFromDb = await db.prepare('SELECT id, prename, surname, city, age, img FROM Persons').all();
  console.log("personFromDb", personsFromDb);
  db.close();
  return new Response(JSON.stringify(
      {
        persons: personsFromDb,
        success: "ok",
        message: ""
      }), {
        status: 200,
      })
}
```

Zur Rückgabe wird ein Response Objekt erzeugt.

Der Body, Statuscode und Headers etc. können zurückgegeben werden.

Astro – Layouts

CLOUD COMPUTING

- Zur einheitlicher Gestaltung und Aufbau bietet Astro Layouts an.
- Diese sind im Ordner Layouts zu finden.
- Hierdurch kann ein Rahmen einer SSR/SSG Seite definiert und wiederverwendet werden.
- SEO und HTML der Seite werden definiert – siehe Layout.astro im Beispielprojekt.

- Hier ein Ausschnitt:

```
<div class="w-full">
    <div class="mx-auto max-w-screen-lg px-4">
      <Navbar/>
      <slot />
      <Footer />
      <style is:global>
      img {
        content-visibility: auto;
      }
      </style>
    </div>
  </div>
```

Wenn das Layout verwendet wird, wird immer eine Navbar oben und ein Footer unten eingefügt. Dort wo der Platzhalter <slot /> steht, wird die Seite eingefügt. (siehe beim Abschnitt SSG/SSR).

- Es können auch mehrere Layouts definiert werden, sodass verschiedene Darstellungen wiederverwendet werden können.

Astro – Modi

CLOUD COMPUTING

- Astro unterstützt SSR – die Seite wird so bei jedem Aufruf auf dem Server neu gebaut.
- Wenn nicht allzu viele Daten gesammelt werden müssen, ist dieser Vorgang in Astro sehr schnell umgesetzt. Im Gegensatz zu anderen Systemen müssen Dokumente nicht aufwendig aus einer Datenbank geladen werden, sondern File-Based.

- Die Konfiguration von Astro wird in astro.config.mjs festgelegt.

```
import { defineConfig } from 'astro/config';
import mdx from '@astrojs/mdx';
import tailwind from '@astrojs/tailwind';
import solid from "@astrojs/solid-js"

export default defineConfig({
    output: "server",
    site: 'https://my-url.de/‘,
    integrations: [mdx(), tailwind(), solid()]
});
```

Hier kann entweder server oder static gewählt werden. Bei Server braucht es zum Deploy einen Adapter wie Node, Vercel, Deno, Netlify, Cloudfare, Partytown.

Integrations sind das Besondere an Astro. Hierdurch können Frontendlibraries und viele andere Elemente einfach in Astro integriert werden.

- Astro ist vorwiegend für Conent-Driven Webseiten konzipiert. Es kennt die zwei Modi static und server. Je nach Konfiguration stehen verschiedene Modi zur Verfügung. Da hier API-Routen und SSR und SSG gezeigt werden sollen wählen wir **output 'server'.**

	output: 'static'	output: 'server'
HTML statisch	ja	mit prerender = true
Islands (`client:*`)	ja	ja
`client:only`	ja	ja
API-Routen	nein	ja
SSR mit DB	nein	ja

- Für SSG muss der Parameter **prerender = true** gesetzt werden.

- Astro kann in seinen Komponenten einen JS-Teil voranstellen, der mit --- eingeleitet wird.
- So kann JS Code vor dem Rendern ausgeführt werden. Es braucht also keine zusätzlichen Dateien, was SSR und SSG sehr komfortabel macht.

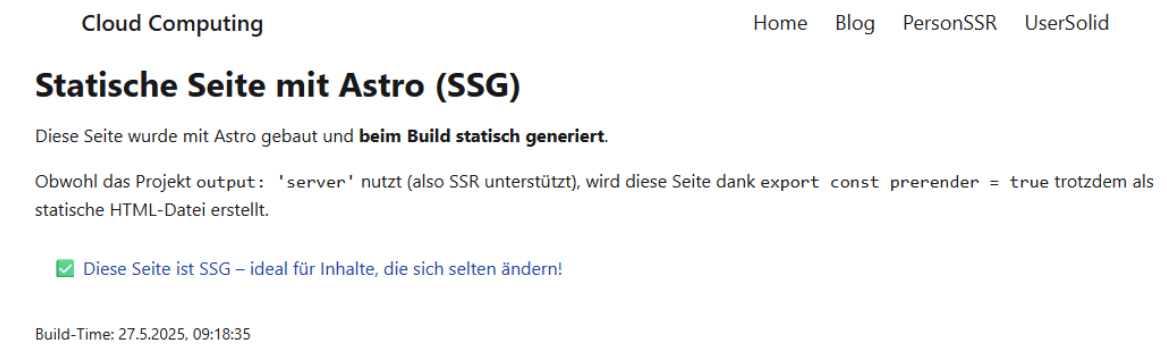
Astro – SSG

CLOUD COMPUTING

- Der Parameter prerender = true muss oben in dem JS Teil gesetzt werden.
- Astro unterstützt Layouts (siehe unten)

```
---
export const prerender = true;
import Layout from '../layouts/Layout.astro';
---

<Layout title="SSG mit Astro">
  <h1 class="text-3xl font-bold text-blue-600 mb-4">Statische Seite mit Astro (SSG)</h1>

  <p class="mb-4">
    Diese Seite wurde mit Astro gebaut und <strong>beim Build statisch generiert</strong>.
  </p>

  <p class="mb-4">
    Obwohl das Projekt <code class="bg-gray-200 px-1 rounded">output: 'server'</code> nutzt (also SSR
unterstützt),
    wird diese Seite dank
    <code class="bg-gray-200 px-1 rounded">export const prerender = true</code>
    trotzdem als statische HTML-Datei erstellt.
  </p>

  <div class="bg-blue-50 border border-blue-200 text-blue-800 px-4 py-3 rounded">
    ☑ Diese Seite ist SSG – ideal für Inhalte, die sich selten ändern!
  </div>

  <p class="mt-6 text-sm text-gray-500">
    Build-Time (nur zur Demo): {new Date().toLocaleString()}
  </p>
</Layout>
```

Beim Bauen der Seite würde das aktuelle Datum fest in die statische Seite integriert, da die Seite nach dem Build nicht mehr verändert wird.



Astro – SSR

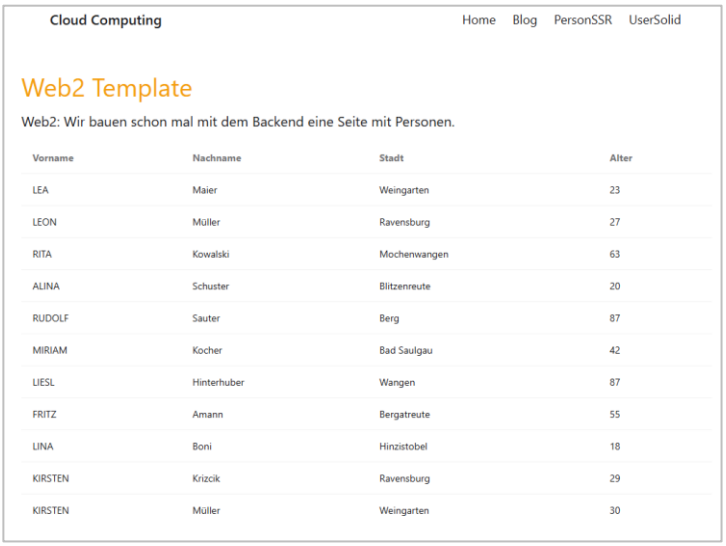
CLOUD COMPUTING

- SSR wird von Astro standardmäßig genutzt, wenn output: server gewählt wird.
- Nun können in dem JS Teil Daten gefetched und aufbereitet werden, bevor die Daten gerendert werden.
- Die Daten, die zum Client geschickt werden, sind dort statisch. Daten können über einfache Forms aber noch übertragen werden. Nur komplexe UI-Logik ist so nicht möglich. Dazu gibt es Island (siehe unten)
- Hier ein Beispiel für SSR – im JS-Teil wird ein Array definiert. Im Teil unten wird dieses Array iteriert. **WICHTIG: Das passiert auf dem Server!**

```
---
import Layout from "@layouts/Layout.astro";
let persons = [
        { prename: "Lea", surname: "Maier", age: 23, city: "Weingarten"},
        { prename: "Leon", surname: "Müller", age: 27 , city: "Ravensburg"},
        { prename: "Rita", surname: "Kowalski", age: 63 , city: "Mochenwangen"},
        { prename: "Alina", surname: "Schuster", age: 20 , city: "Blitzenreute"},
        { prename: "Rudolf", surname: "Sauter", age: 87 , city: "Berg"},
        { prename: "Miriam", surname: "Kocher", age: 42, city: "Bad Saulgau"},
        { prename: "Liesl", surname: "Hinterhuber", age: 87, city: "Wangen"},
        { prename: "Fritz", surname: "Amann", age: 55 , city: "Bergatreute"},
        { prename: "Lina", surname: "Boni", age: 18 , city: "Hinzistobel"},
        { prename: "Kirsten", surname: "Krizcik", age: 29, city: "Ravensburg"},
        { prename: "Kirsten", surname: "Müller", age: 30, city: "Weingarten"}
    ]
---

<Layout title="Persons">
  <section class="grid py-10">
    <h1 class="text-4xl text-amber-500">Web2 Template</h1>
    <p class="text-xl py-4">Web2: Wir bauen schon mal mit dem Backend eine Seite mit Personen.</p>
    <table class="w-full table">
      <thead>
        <tr>
          <th>Vorname</th>
          <th>Nachname</th>
          <th>Stadt</th>
          <th>Alter</th>
        </tr>
      </thead>
      <tbody>
       { persons.map((person) =>
          <tr>
            <td>{ person.prename.toUpperCase() }</td>
            <td>{ person.surname }</td>
            <td>{ person.city }</td>
            <td>{ person.age }</td>
          </tr>)}
      </tbody>
  </table>
  </section>
</Layout>
```

Dieses Array wird auf dem Server gebildet. Hier können auch DB-Calls gemacht werden.
Hinweis: Zu allem in diesem Block hat der Client keinen Zugriff.

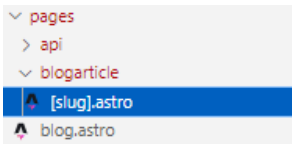
Das Array wird iteriert und nach der Ausführung wird Das HTML an den Client geschickt. Dort verhält es sich wie eine „statische" Seite



Astro – SSR mit Markdown

CLOUD COMPUTING

- Astro unterstützt die Erstellung von dynamischen Routen.
- Die Route blogarticle enthält eine Datei **[slug].astro**. Slug steht hier für eine beliebige Unter-Route.
- Wenn also ein Link aufgerufen wird wie **http://localhost:3000/blogarticle/anonymjson** dann wird die Datei **[slug].astro** aufgerufen und die Route **anonymjs** wird in dem Parameter **slug** übergeben.



- Bei dem Beispiel hier, soll eine Blogbeitrag gerendert werden, der als Markdown vorliegt.
- Es können also einfach Artikel als Markdown ins File-System gelegt werden und mit Astro können diese gerendert werden.

```
---
import { getCollection } from 'astro:content';
import Layout from '../../layouts/Layout.astro';
import dayjs from 'dayjs';

const { slug } = Astro.params;
const entries = await getCollection('blog');

const entry = entries.find(e => e.slug === slug);
console.log("Entry ", entry)

if (!entry) {
  throw new Error(`Kein Eintrag gefunden für slug: ${slug}`);
}

const { Content } = await entry.render();
---

<Layout title={entry.data.title}>

   <div class="mx-auto max-w-3xl mt-14">
      <h1
        class="text-4xl lg:text-5xl font-bold lg:tracking-tight mt-1 lg:leading-tight">
        {entry.data.title}
      </h1>
      <div class="flex gap-2 mt-3 items-center flex-wrap md:flex-nowrap">
        <span class="text-gray-400">
          {entry.data.author}
        </span>
        <span class="text-gray-400">•</span>
        <time
          class="text-gray-400"
          datetime={entry.data.publishDate.toISOString()}>
          {dayjs(entry.data.publishDate).format("DD.MM.YYYY")}
        </time>
      </div>
    </div>

    <div class="mx-auto prose prose-lg mt-6 max-w-3xl">
      <Content />
    </div>
    <div class="text-center mt-8">
      <a
        href="/blog"
        class="bg-gray-100 px-5 py-3 rounded-md hover:bg-gray-200 transition"
        >← Back to Blog</a
      >
    </div>

</Layout>
```

slug ist der die Route, z.B. /anonymjs

getCollection holt alle Dateien aus dem Ordner content/blog

Hier wird die Datei, die zum slug passt gesucht. Entry enthält nun auch die Infos aus dem Prefix, also Infos zum Autor etc.

Das Markdown-File wird gerendert (siehe nächste Seite).

Daten aus dem Prefix werden gerendert.

Hier kommt der eigentliche Blogbeitrag rein.

Astro – SSR mit Markdown

CLOUD COMPUTING

- Markdown wird in der vorliegenden Konfiguration seit den letzten Astro Versionen gut unterstützt.
- Durch den JS-Teil, der ebenfalls mit --- begonnen und beendet wird, können Eigenschaften festgelegt werden, die dann später zum Rendern genutzt werden.
- Beispiel ist das Feld publish: true, das als Schalter zum Rendern genutzt werden kann.
- Der Client hat auf die MD-Dateien keinen Zugriff.
- Dieses Vorgehen macht Astro so stark im Bereich Content.

```
---
title: Anonyme Funktionen und JSON
author: Chat Tschibitti
description: Hier werden anonyme Funktionen in Javascript beschrieben.
publish: true
publishDate: 2024-04-13
mydata: 3
image: {
    src: "javascript.jpg",
    alt: "Mediendesign"
}
---

# Anonyme Funktionen in JavaScript

Anonyme Funktionen sind Funktionen ohne Namen. In JavaScript sind sie ein zentrales Konzept,
besonders im funktionalen Programmierstil. Sie werden oft dort verwendet, wo Funktionen „on the fly“
benötigt werden – zum Beispiel als Argumente für andere Funktionen.

### Syntax und Beispiel

Eine anonyme Funktion kann etwa so aussehen:
```

- Die Übersicht der Blogs erfolgt in der Seite blog.astro

Astro – SSR mit Markdown

CLOUD COMPUTING

- In blog.astro wird eine Liste aller Blogs dargestellt, die in dem ordner content/blog enthalten sind und deren publishData vor dem heutigen Tag sind und deren publish-Flag auf true gesetzt ist.

```
import { getCollection } from "astro:content";
import { Image } from 'astro:assets';
import Layout from "@layouts/Layout.astro";
import dayjs from "dayjs"

const publishedBlogEntries = await getCollection("blog", ({ data }) => {
  return data.publish && data.publishDate < new Date();
});

publishedBlogEntries.sort(function (a, b) {
  return b.data.publishDate.valueOf() - a.data.publishDate.valueOf();
});
---

<Layout title="Blog" active="blog">
    <main class="mt-16">
      <ul class="grid gap-16 max-w-4xl mx-auto">
        {
          publishedBlogEntries.map((blogPostEntry, index) => (
            <li>
              <a href={`/blogarticle/${blogPostEntry.slug}`}>
                <div class="grid md:grid-cols-2 gap-5 md:gap-10 items-center">
                <Image
                    src={"/images/" + blogPostEntry.data.image.src }
                    alt={blogPostEntry.data.image.alt}
                    sizes="(max-width: 800px) 100vw, 800px"
                    width="200"
                    height="355"
                    loading={index <= 2 ? "eager" : "lazy"}
                    decoding={index <= 2 ? "sync" : "async"}
                    class="w-full rounded-md"
                />
                 <div>
                  <h2 class="text-3xl font-semibold leading-snug tracking-tight mt-1 ">
                    {blogPostEntry.data.title}
                  </h2>

                  <div class="flex gap-2 mt-3">
                    <span class="text-gray-400">
                      {blogPostEntry.data.author}
                    </span>
                    <span class="text-gray-400">•</span>
                    <time
                      class="text-gray-400"
                      datetime={dayjs(blogPostEntry.data.publishDate).format("DD.MM.YYYY")}>
                      {dayjs(blogPostEntry.data.publishDate).format("DD.MM.YYYY")}
                    </time>
                  </div>
                </div>
              </div>
            </a>
          </li>
        ))
        }
      </ul>
    </main>
</Layout>
```

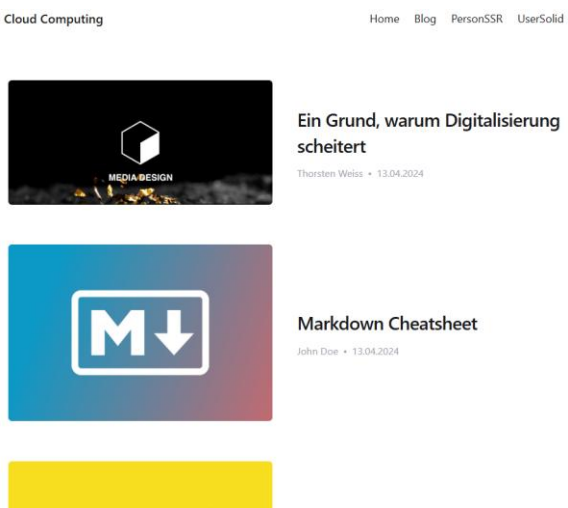
getCollection holt alle Dateien aus dem Ordner content/blog und danach werden diese gefiltert.

Nach Datum sortieren

Nach Datum sortieren

Iterieren und Bild mit Kurztext darstellen.

Datum Kurztext etc.



Astro – Islands

CLOUD COMPUTING

- In Astro besteht die Möglichkeit, alle 5 großen Frontendlibraries als Island in SSR / SSG Seiten zu laden. So werden Teile der Seite dynamisch.
- Es ist zudem möglich, eine komplette SPA, wie sie im Teil SolidJS besprochen wurde, ins Frontend zu laden und das Routing etc. alles in der SPA zu machen.
- Dies ermöglicht beispielsweise eine Entwicklung einer Seite, die SEO etc. mit SSR und SSG erledigt und einen separaten Login-Bereich hat, der nicht SEO-relevant ist und alle Möglichkeiten der SPA bereitstellt. Dies macht Astro sehr flexibel.
- Weiterhin können auch mehrere Libraries genutzt werden, z.B. könnte React und Solid gleichezeitig in einem Projekt und sogar auf einer Seite genutzt werden. Hierbei können allerdings gegenseitige Effekte auftreten. Daher ist eine genaue Analyse hier erforderlich.
- Das Laden von Islands ist sehr einfach. In einer Astro-Datei wird mit dem prop client:xxx festgelegt, wo der Code läuft. Beispielsweise könnte Solid so zur Generierung von Code im Server für SSR und für intelligente Komponenten im Client genutzt werden.
- Hier ein Beispiel einer Solid Komponente ähnlich zu der im Teil SolidJS – diese läuft im Client. Die Daten werden hier von dem API-Endpoint /api/person geladen.

```
---
import Layout from "@layouts/Layout.astro";
import SolidPage from "@solid/solid_pages/PersonCrud"
import config from "../utils/config";
const theme = config.theme;
---

<Layout title="Web2">
  <section class=" py-10 place-items-center text-center">
    <SolidPage client:only="solid" theme={theme} />
  </section>
</Layout>
```

- Die Navbar und der Footer sind statisch. Die Island reaktiv. Zu beachten ist, dass jedes mal wenn die Route geändert wird und wieder zu UserSolid zurück gekehrt wird, alles neugeladen wird.
- Daher ist für eine SPA ein Solid Router empfehlenswerter. Die Islands sind für interaktive Teile auf einer statischen Seite gedacht.