Zurück in die Zukunft mit statischen Webseiten

19.6.2023, 10:25:14

1/31 Zurück in die Zukunft mit statischen Webseiten

- Willkommen in Darmstadt
- Timo Zander, Astro Framework
- Astro Hype
- heute: Warum Astro
- Presentation + Live Coding. Fragen am Ende
- Folien + Code auf Webseite, timozander.de

2/31 Was?

- Doch was ist Astro
- Web-Framework mit Fokus auf Inhalte: Blogs, Portfolios, E-Commerce
- Inhalte: aus verschiedenen Quellen
- Build durch Compiler (Integrationen und Plugins! Tailwind, React, ...)
- Statische Ausgabe: "Reine" HTML Dateien
- Deployment: Webserver, Edge, ...
- einreihen in Konkurrenz (ÜBERLEITUNG)

3/31 Static Site Generators (SSG)

- Konkurrenz: SSGs
- Auswahl an Frameworks: welches nimmt man?

- jeder hat Lieblings Framework
- React vs Vue Kampf, "Svelte ist Revolution"
- Wahl des Tools nach Framework -> hilft nicht
- Next und Nuxt als Platzhirsche -> natürliche Wahl
- spezialisierte Tools: Docusaurus als Beispiel

- viele von euch fragen sich: WARUM
- warum noch ein Framework
- Kritisch beäugt bei Neuem -> GUT SO!

- Zeigen was Astro besonders macht (4 Punkte)

4/31 Warum Astro?

- klarer Fokus
- Vergleich mit Next.js = kann alles
- Astro sagt klar: Inhalte-lastige Webseiten (und hat entsprechend Features)
- Wer JIRA nachbauen will, ist hier falsch

- PERFORMANCE: per default 0 client-JS
- doch ganz ohne JS ist nicht gut (Beispiele)
- Deshalb: Island-Architektur
- sorgt für Performance: schnelle Time-to-interactive!

- Framework-agnostisch
- EINFACH zu installiert

5/31 Islands Architektur

- Webseite in "Inseln" aufgeteilt
- statisch: wird in HTML-Datei statisch ausgeliefert
- dynamisch: erfordert Client-JS. Entweder rendern oder ladern

- per Default alles statisch
- hilft für Performance: Keine Gedanken an Performance
- Client-JS immer ein bewusstes Opt-in

6/31 MPA-Architektur

- Astro hat MPA-Architektur
- im Gegensatz zu React und co

- SPA: nur eine index.html Datei, leer
- Client-JS rendert alles
- JS übernimmt Navigation, State-Management ...

- MPA: "traditionell" wie PHP etc.
- Server rendert HTML (mit Inhalten)
- Navigation = neue Request an Server, neue HTML Datei
- SESSIONS auf Server für State

- Spektrum: von SPA bis MPA
- z.B. serverseitig gerenderte SPAs, erhöht Initial-Performance
- kein gut oder schlecht
- für Astros Zwecke: MPA besser

7/31 Rendering Modi von Astro

1. Statisches Rendern (wie erwähnt)

   - Build wird ausgeführt
   - Compiler gibt fertige HTML, CSS, JS Datei aus
   - keine Spur mehr von Astro
   - kostengünstig irgendwo statisch hosten (Nginx ...)
   - in Stein gemeißelt: Nur Build ändert Inhalte

- klingt wie Rückschritt, aber:
- automatisieren mit Build-Pipeline
- Beispiel von meinem Blog
- wenige Minuten zwischen Commit + Live

klick

2. Serverseitiges Rendern

   - Request an Server (z.B. "About me")
   - Astro-Runtime rendert die Seite, gibt on demand HTML zurück
   - Vorteil: Datenbanken, APIs, Echtzeit
   - Astro Fokus auf statischen Inhalten
   - deshalb heute: Schwerpunkt auf SSG
   - Syntax zu 90% gleich

8/31 Astro in der Praxis

wie sieht ein Astro Projekt aus

9/31 Ein Astro-Projekt zum Leben erwecken

- schnell erstellt
- CLI oder online
- vielzahl von Templates
- vom leeren Projekt bis "Kitchen sink"
- heute: leeres Projekt
- reminder: Sourcecode online

10/31 Ein neues Astro Projekt

- überwiegend Standard

klick

- Astro-Config: Add-ons, Verhalten des Frameworks, ...
- z.B. vue als "integration" hinzufügen

klick

- interessant sind public und src
- public: Dateien und Assets, welche *nicht* vom Build berührt werden (z.B. robots.txt)
- src: Quellcode, folgt jetzt

11/31 Das `src` Verzeichnis

- Bausteine: Komponenten, Layouts, Seiten
- Seite: wie man denkt, eigene URL, eigene HTML-Datei
- Layouts: Skelett
  - HTML Basics wie DOCTYPE oder `<head>`
  - wird mit Inhalten und Komponenten gefüllt
- Komponente: wie in UI-Frameworks. Button, Dropdown, ...
- Beispiel persönlicher Blog

12/31 Anatomie einer Astro Seite

- Beispiel: persönliche Seite (mit Astro)
- Startseite = Page

- Layout: gesamte Seite, von `<head>` bis `</body>`
- Social Media Icons: Komponente
- wird wiederverwendet im Footer
- Layouts und Astro-Komponenten technisch nahezu gleich!

weiter mit Komponenten

13/31 Astro Komponenten

- simpel aufgebaut
- optionaler JS-Header mit Fences
  - da optional: jede HTML-Datei ist gültige Astro-Komponente
- beliebiger HTML-Code mit JSX-Syntax
- JS-Code in geschweiften Klammern

- Besonderheit von SSG: Code wird nur beim Build ausgeführt
- muss man sich gewöhnen, da JS-Code nicht immer "frisch"
- Datum bleibt also fix auf Build-Zeit

weitere, realitätsnahe Komponente...

14/31

- (wall of code)
- LinkButton von meinem Blog
- zuerst JS

klick

- TypeScript Typen: definieren und exportieren klappt
- `ButtonType` woanders wiederverwenden
- `Props` sind die Props der Komponente (Konvention)
- sorgt für Auto-Complete und Typisierung

15/31 Props in Astro-Komponenten

- Syntax ist wie bei React, Vue usw.
- Auto-Complete dank Props interface

16/31

- Props können im Astro.props Objekt abgerufen werden
- Nutzung im Markup
- bekannte JSX-Syntax
- Slots

17/31 Frontend-Frameworks nutzen

- Clue: statt Astro-Komponenten eigenes FE-Framework nutzen
- einfach offizielle Plugins für Vue, React, Svelte, Preact, Solid, Lit, Alpine,...
- `pnpm astro add vue` -> fertig
- Astros Kern bleibt: alles statisch gerendert per default

18/31 Pages und Navigation

- Komponente klar
- jetzt: Pages und Navigation

Astro nutzt MPA -> mehrere Pages

19/31 Die `index.astro` Page

- 1 URL = 1 HTML Datei = 1 Astro Page
- Beispiel-Page
- Pages sind "Kleber" zwischen Layouts und Komponenten
- Komponente nicht genutzt -> nicht im Output
- JS Header mit 3 Fences
- Erinnerung: nur im Build ausgeführt

20/31 File-based routing

- File-based Routing
- Verzeichnisstruktur bestimmt URLs (Slide erklären)
- Dateitypen: .astro, .md, .html, .ts

klick

- unter Blog: eckige Klammern im Dateinamne
- dynamische Route
- nicht nur statische Pfade wie Home oder About
- Beispiel Blog: nicht für jeden Beitrag Seite copy-pasten

LIVE CODING 01-start

21/31 Islands-Architektur angewandt

- wie funktioniert Islands nun?
- standard ist kein Client-JS
- wie schaffe ich nun JS auf Client?

22/31 `script` in Komponenten

- JS an 2 Stellen:
- Header im Build
- `<script>` Tag im Browser

LIVE CODING 02-start

23/31 Verarbeitung von client-side JavaScript

- ihr seht: Astro scheint zu optimieren
- Script nur einmal da -> was tut Astro?

- Skript-Tags in Build Pipeline (Code optimierung und Import bundling)

1. Importe werden gebundled (imports nicht in extra Dateien)
2. Skripte werden in den head verschoben, keine Duplikate
3. TypeScript ist eingebaut: Transpilieren + Checken

24/31 Skripte "pur" verwenden

- Skripte 1:1 im Build behalten
- neben `is:inline` noch andere Direktiven

Was geht also nicht?

- keine Importe (relativ oder npm Packages)
- kein Verschieben in den head
- kein Bundling in externe Datei -> kein defer (async laden) MEHR DETAILS

25/31 Client-Rendering mit Direktiven steuern

- bei UI-Frameworks (React, Vue, ...) -> clientseitig rendern aktivieren
- Steuern WANN gerendert/ausgeführt wird mit client-Direktiven
- nicht für Astro Komponenten -> immer SSR

1. load -> direkt am Anfang. "Überlebenswichtige" Sachen
2. idle -> wenn Client idled (schon interaktiv)
3. visible -> Mittelweg, wenn Komponente sichtbar wird

    - Komponente im Footer: nicht jeder scrollt
    - kann Zeit sparen

seit Astro 2.6: eigene Direktiven

26/31 Dynamische Inhalte und statische Seiten

- Basics fertig
- jetzt los ziehen und Astro Seite bauen
- Spaß weil: gute Doks + hilfreiche Fehler (eben gesehen)

- Fader Beigeschmack: Dynamische Inhalte?
- niemals so "live" wie nicht-SSG Lösung, klar
- aber neuer Blog Post sollte kein Riesen Aufwand sein

jetzt: Wie Astro dynamische Inhalte unterstützt

27/31 Content Collections

- oft kein CMS oder DB nötig
- einfacher Blog und Portfolio: Inhalte strukturiert als Dateien
- Performance egal: Build ggf. länger, aber who cares (wenige Sekunden)

- daher: Content Collections
- strukturierte Inhalte in Dateien

- Blog-Posts und Newsletter Einträge
- Markdown (Inhalte) oder JSON (reine Daten)
- Markdown hat Frontmatter-Header mit Attributen (z.B. title)

28/31 Einträge typisieren

- Zod ist integriert
- umfangreiche Typ-API für exotische Edge Cases
- sorgt für typisierte Auto-Complete und Build-Check

klick

- Collection für Blog-Posts
- Zod validiert Frontmatter-Header

klick

- export benötigt blog Key
- key = Ordnername

29/31 Inhalte aus Collections nutzen

- Abruf mit `getCollection()`
- slug: URL-Friendly Version des Dateinamens (unique ID)
- data Attribut in Schleife: Frontmatter-Attribute (typisiert!!)
- bei Typ-Verletzung: Build schlägt fehl

LIVE CODING 04-start

30/31 Daten referenzieren

- JSON in Content Collections neu seit Astro 2.5
- vereinfachte, Datei-basierte Datenbank
- Referenzen möglich!

- sorgt für Datenintegrität
- wie in SQL, Blog-Post muss immer Autor haben (kein "von undefined undefined")

optional: LIVE CODING 05-start

31/31

Heinrich Heine

- Empfehlung: Wenn gefallen, dann probiert Astro aus
- für Astro-taugliches Projekt (und nicht für Webanwendung)
- falls gefällt: großartig
- falls nicht: genauso gut - Findet das Tool, was für EUCH am besten passt
- noch viel mehr zu entdecken: Asset Optimierung, Endpoints, ...