ANGULAR ARCHITECTS

JavaScript & Angular Days

https://LXT.dev

Alexander Thalhammer

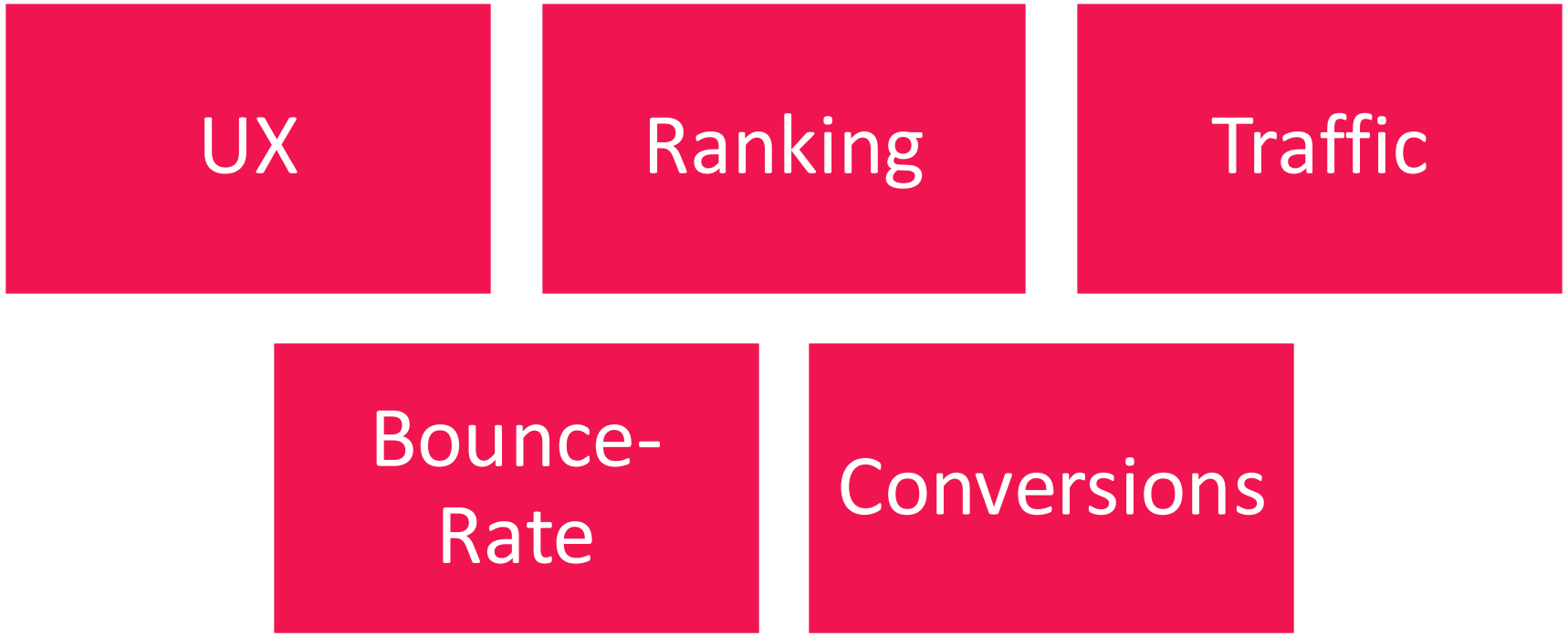
# SSR, SSG, Deferrable Views und Incremental Hydration in NG 19

Alexander Thalhammer, ANGULARarchitects.io

Git: https://github.com/L-X-T/ssr-ih-ng19-days

# Warum Performance-Optimierung?

ANGULAR ARCHITECTS



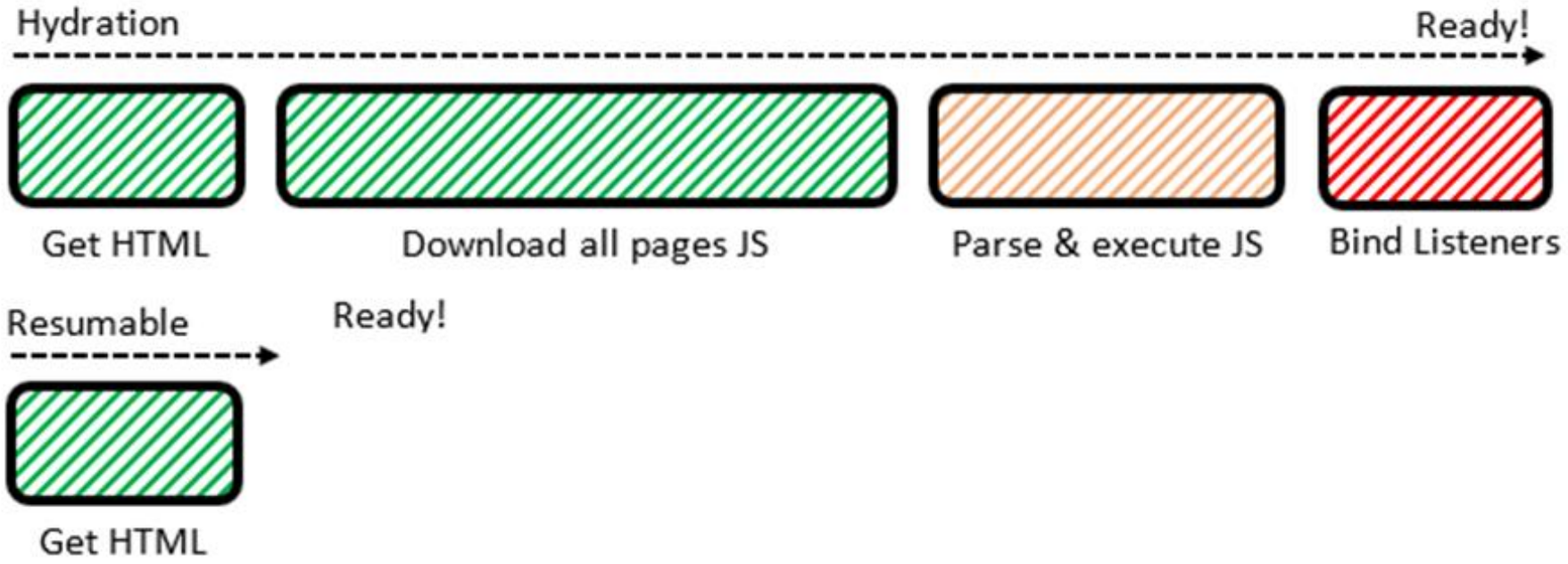
# Warum Server-Side Rendering?

ANGULAR ARCHITECTS

Initial Load verbessern

Ressourcen sparen

# Warum Server-Side Rendering?




https://www.acldigital.com/blogs/zero-hydration-qwik-future-web

ANGULAR
ARCHITECTS

# Warum
# setzt ihr es nicht ein?

Alexander Thalhammer

# Challenges

ANGULAR ARCHITECTS

Server

Login

Live-Content

Übersetzung

Federation

Instagram

# Realität

# Hi, it's me → @LX_T

ANGULAR ARCHITECTS

- Alexander Thalhammer from Graz, Austria (est. 1983)
  - **Angular Software Tree GmbH** (est. 2019)
- Web Dev for 24 years
  - Since 2017 Angular Dev
- Since 2021 Angular Evangelist, Coach & Consultant
  - Member of the **Angular Architects**

# @LX_T → Lieblingsthemen

ANGULAR ARCHITECTS

Performance 🚀

Accessibility ♿

Best Practices 📈

# Agenda

ANGULAR ARCHITECTS

1) Server-Side Rendering (SSR)
2) Prerendering während Build (SSG)
3) Hydration (NG >= 16)
4) Event Replay (NG >= 18)
5) Hybrid Rendering (NG => 19)
6) Deferrable Views (NG >= 17)
7) Incremental Hydration (NG >= 19)

# Labs

ANGULAR ARCHITECTS

Audit

@defer

Incremental Hydration

[repository] https://github.com/L-X-T/ssr-ih-ng19-days

ANGULAR
ARCHITECTS

# First things first

Alexander Thalhammer

# First things first: Begriffe klären

ANGULAR ARCHITECTS

- Lazy Loading === Defer Loading
  - Lazy Loading für Router-basiertes Nachladen mit **loadChildren()** / **loadComponent()**
  - Defer für Template-basiertes Nachladen mittels **@defer**
- Hydration → HTML mit JS (Event Handlers) anreichern
- Client-side Rendering, Server-side Rendering, Static Site Generation
- above-the-fold

# First things first: Performance messen

ANGULAR ARCHITECTS

- Performance Deiner Angular App messen
- Verbesserungen messbar machen
- Wie messen wir? 🤔

# Angular App mit Google Lighthouse testen

ANGULAR ARCHITECTS

- ng b(uild)
- npm i - g serve
- serve dist/[appName]/browser -s
- In Chrome http://localhost:3000 öffnen, mit Lighthouse (DevTools) testen

ANGULAR ARCHITECTS

# Lab: Performance messen mit Lighthouse

https://github.com/L-X-T/ssr-ih-ng19-days/blob/main/labs/01-audit.md

# 1) SSR @angular/ssr (v17)

ANGULAR ARCHITECTS

- ersetzt Universal Package
  - Community Lösung
- standardmäßig in: **ng new**
- oder: **ng add @angular/ssr**

# 1) SSR – Grundlagen

ANGULAR ARCHITECTS

- **(fertiges) HTML** wird am Server erzeugt und vom Client heruntergeladen
- Wie klassische Websites
- Bessere Performance

# 1) SSR – SEO Considerations

ANGULAR ARCHITECTS

- Wenn SSR für Suchmaschinen, unbedingt Meta Tags implementieren

```
export class SeoComponent {
    private readonly title = inject(Title);
    private readonly meta = inject(Meta);

    constructor() {
        // set SEO metadata
        this.title.setTitle("My fancy page/route title. Ideal length 60-70 chars");
        this.meta.addTag({ name: "description", content: "My fancy meta description. Ideal length 120-150 characters." });
    }
}
```

# 1) SSR – Typische Fehler

ANGULAR ARCHITECTS

- Valides HTML erforderlich (anyway good practice ☺)
- Gewisse Browser Features nicht nutzbar, z.B.
  - document, window, localStorage, sessionStorage
- private readonly document = inject(DOCUMENT);
- private readonly platform = inject(PLATFORM_ID);

# 1) SSR – platfrom check

ANGULAR ARCHITECTS

```
    constructor() {
        if (isPlatformBrowser(this.platform)) {
            console.warn("browser");
            // Safe to use document, window, localStorage, etc. :-)
            console.log(document);
        }

        if (isPlatformServer(this.platform)) {
            console.warn("server");
            // Not smart to use document here, however, we can inject it ;-)
            console.log(this.document);
        }
    }
}
```

# 1) SSR – Browser only render hooks

ANGULAR ARCHITECTS

- afterNextRender()
- afterRender() → wird voraussichtlich umbenannt in afterEachRender()

# 1) SSR – caution with ng serve in v19

ANGULAR ARCHITECTS

- disable hot module reloading
- ng serve –no-hmr

ANGULAR ARCHITECTS

# DEMO SSR mit ng serve --no-hmr

https://github.com/L-X-T/ssr-ih-ng19-days

# 2) Prerendering während des Builds (SSG)

ANGULAR ARCHITECTS

- On-demand-Rendering am Server benötigt Zeit & Ressourcen
- Lösung: Prerendering (SSG)
  - Alle statische Inhalte können schon vorgerendert werden
- Nachteile
  - Funktioniert nicht für Live-Inhalte
  - Verzögert die Build-Time
- Vorteile
  - Sehr schnelle Ladezeiten
  - Kein Node.js/Express Server nötig

ANGULAR ARCHITECTS

# DEMO SSG mit ng build

https://github.com/L-X-T/ssr-ih-ng19-days

# 3) non-destructive HYDRATION (v16)

ANGULAR ARCHITECTS

- neuerdings auch **full-application hydration**
- statisches HTML wird mit JavaScript versorgt
- konkret werden alle Angular Event Handler "(click)= doSth()" angehängt

# 3) warum non-destructive?

ANGULAR ARCHITECTS

- bis Angular 16 wurde der DOM komplett (zerstört und) ersetzt
  - daraus ergab sich ein Flickern beim initialen Laden der Angular App
  - Hydration vermeidet Layout-Verschiebungen (schlechte UX)

- erst mit diesem Feature SSR in Angular verwendbar (ab v16)

```
export const appConfig: ApplicationConfig = {
    providers: [
        provideClientHydration(), // use v16 hydration
    ],
};
```

# 3) Hydration Trouble Shooting

ANGULAR ARCHITECTS

- manche Komponenten (z.B. externe Libs) sind nicht kompatibel
- hydration kann für einzelne Kompenten deaktiviert werden mittels

```
<app-example ngSkipHydration />
```

- oder via host binding

```
@Component({
    host: { ngSkipHydration: "true" },
})
class DryComponent {}
```

# 4) Event Replay (v18)

ANGULAR ARCHITECTS

- User-Events werden gefangen und nach Hydration noch einmal gefeuert

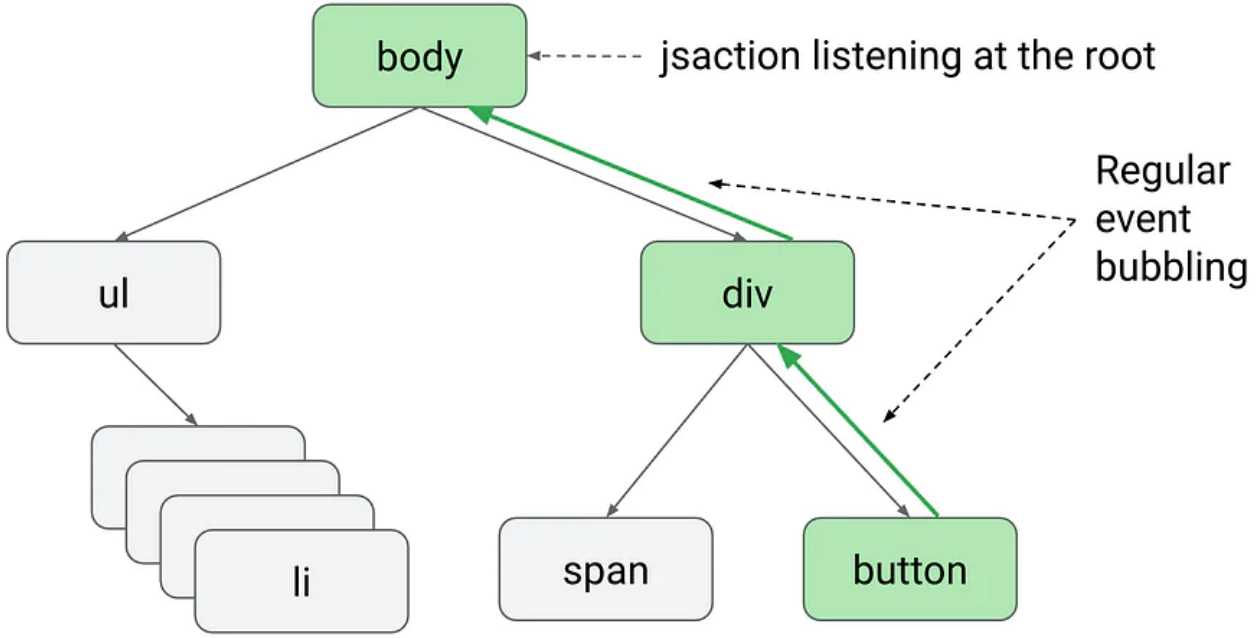
- Das ermöglicht User-Interaktionen während des intialen Ladens der App

```
export const appConfig: ApplicationConfig = {
    providers: [
        provideClientHydration(
            withEventReplay(), // use hydration with v18 event replay
        ),
    ],
};
```

# 4) Event Replay – Details

ANGULAR ARCHITECTS

- JSAction (mini-Lib von Googles Wiz Framework) fängt am body Element alle Events ab und feuert sie nach dem initialen Laden erneut



# 5) Hyrbid Rendering (v19)

ANGULAR ARCHITECTS

- Ab NG 19 können wir für jede Route den Modus wählen:
  - RenderMode.Prerender (SSG) für statischen Content
  - RenderMode.Server (SSR) für dynamischen Content
  - RenderMode.Client (CSR) für benutzerabhängigen Content

```
/* src/app/app.routes.server.ts */
// imports [...]

export const serverRoutes: ServerRoute[] = [
    { path: "ssr", renderMode: RenderMode.Server },
    { path: "ssg", renderMode: RenderMode.Prerender },
    { path: "csr", renderMode: RenderMode.Client },
];
```

# 5) Hyrbid Rendering – 301 & 404

ANGULAR ARCHITECTS

- Zusätzlich können Umleitungen oder Fehlerseiten ausgeliefert werden

```
export const serverRoutes: ServerRoute[] = [
    // [...],
    { path: "redirect", renderMode: RenderMode.Server, status: 301 },
    {
        path: "error",
        renderMode: RenderMode.Server,
        status: 404,
        headers: {
            "Cache-Control": "no-cache",
        },
    },
    { path: "**", renderMode: RenderMode.Server },
];
```

ANGULAR ARCHITECTS

# DEMO Hybrid Rendering

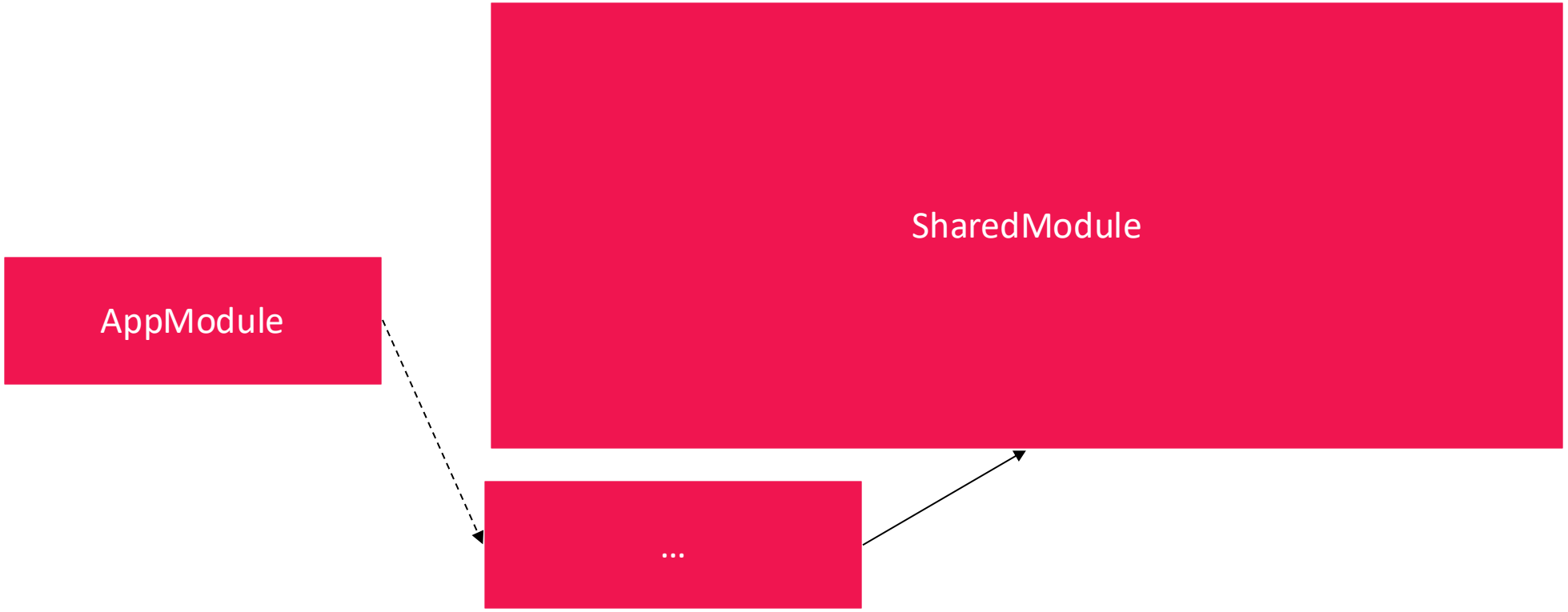
https://github.com/jeanmeche/ssr-v19

# Standalone Components (v14)

ANGULAR ARCHITECTS

# NgModules – Häufiges Problem

ANGULAR ARCHITECTS




**Root Module** **Feature Module(s)** **Shared Module**

# Digression – Angular Migrations (Pflicht!)

ANGULAR ARCHITECTS

| | | |
|---|---|---|
| standalone | build w/ esbuild & vite | signal inputs & queries |
| control flow | lazy-loaded routes | cleanup imports & self-closing tags |

ANGULAR ARCHITECTS

# Angular Docs – Migration

https://angular.dev/reference/migrations

# 6) Deferrable Views mit @defer (v17)

ANGULAR ARCHITECTS

- Nur mit Standalone-Komponenten
- Alternative zum Router-basierten Lazy Loading in Angular
  - um den initial Load zu verkleinern → Performance
  - ohne eigene Route
- Sehr gute DX im Vergleich zu früheren Lösungen mit **ViewContainerRef**

# 6) Deferrable Views – Beispiele

ANGULAR ARCHITECTS

- Widgets die auf mehreren Routen verwendet werden
- Nachladen großer 3rd-Party-Dependencies (Charts, Tabellen, …)
- Erzwingt CSR → Kombination mit SSR für userabhängigen Content (Preis)
- Nicht geeignet für above-the-fold Content (initial nur Platzhalter sichtbar)

# 6) Deferrable Views – Trigger Intro

ANGULAR ARCHITECTS

- Triggert (lazy) Laden der Komponente
  - z.B. on viewport (wie ngOptimizedImage)

```
@defer (on viewport) {
  <aa-lazy-component />
}
```

# 6) Deferrable Views – Trigger Details

ANGULAR ARCHITECTS

- on
  - immediate
  - idle (*default*)
  - viewport (like ngOptimizedImage)
  - hover (mouseover & focusin)
  - interaction (click & keydown)
  - timer(4200ms) (use case → demo)

- when
  - boolean Variable oder
  - Signal<boolean> oder
  - Funktion

- *beides one-way ticket*
- Kombination mehrerer

# 6) Deferrable Views – Prefetch

ANGULAR ARCHITECTS

- Mit **prefetch** kann das deferrte JavaScript vorgeladen werden
  - vergleichbar mit der PreloadingStrategy beim routerbasierten Lazy Loading

- Beschleunigt den Ladevorgang wenn er @defer trigger ausgelöst wird

```
@defer (on viewport; prefetch on idle) {
  <aa-lazy-component />
}
```

# 6) Deferrable Views – Placeholder & Loading

ANGULAR ARCHITECTS

- @placeholder
- @loading
- @error

```
@defer (on viewport; prefetch on idle) {
  <aa-lazy-component />
} @placeholder (minimum 500ms) {
  <img width="420" height="420" alt="lazy component placeholder" src="placeholder.avif" />
} @loading (after 500ms; minimum 1s) {
  <img width="420" height="420" alt="lazy is loading spinner" src="spinner.avif" />
} @error {
  <p>Why do I exist?</p>
}
```

# 6) Deferrable Views – Best Practices

ANGULAR ARCHITECTS

- Vorsicht bei verschachtelten **@defer**-Blöcken
  - sollten unterschiedliche Trigger haben
- Vorsicht bei **@defer** das above-the-fold und während des Initial Loads getriggert wird
  - es sollten keine Layout Shifts auftreten (schlechte UX)

# 6) Deferrable Views – Kombination mit SSR

ANGULAR ARCHITECTS

- In Kombination mit SSR wird am Server der **@placeholder** gerendert
  - bzw. nichts wenn kein @placeholder
- Erst im Client werden **@defer** Blöcke geladen und die Trigger aktiviert
  - somit via @defer im Client-Side Rendering erzwingbar
- Erst mit Incremental Hydration wird Content am Server vor-gerendert

ANGULAR ARCHITECTS

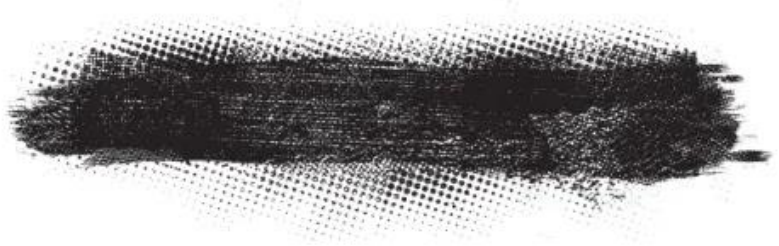
# Lab: @defer

https://github.com/L-X-T/ssr-ih-ng19-days/blob/main/labs/02-defer.md

# 7) Incremental Hydration (v19)

ANGULAR ARCHITECTS

The *FUTURE* is



incremental hydration

# 7) Incremental Hydration (v19)

ANGULAR ARCHITECTS

- basiert auf **Hydration**, **@defer** und **Event Replay**
- ermöglicht **noch besseres** Initiales Laden, weil
  - weniger JavaScript ausgeliefert wird → partielles Nachladen von JS on demand
- **@defer**-Blöcke nun auch above-the-fold möglich, weil
  - der Content (statt nur @placeholder) schon am Server gerendert wird

# 7) Incremental Hydration – Trigger

ANGULAR ARCHITECTS

- gleiche Trigger wie **@defer**
- **on** (immediate, idle, viewport, hover, interaction)
- **when** (boolean Variable, Signal<boolean> oder Funktion)
- **never** (für statischen Content ohne Interaktivität)

# 7) Incremental Hydration – Szenarios

ANGULAR ARCHITECTS

- Für above-the-fold Inhalte nutzbar
- Dynamischer below-the-fold Inhalt
- **Produkte** im Online-Shop
- **Formulare** und **Actions**
- Interaktive **Tabellen**

ANGULAR ARCHITECTS

# Lab: Incremental Hydration

https://github.com/L-X-T/ssr-ih-ng19-days/blob/main/labs/03-incremental-hydration.md

# Recap – Why SSR & Incremental Hydration

ANGULAR ARCHITECTS

1) Faster Angular Apps (LCP & INP)
2) Better UX
3) Everybody happy ☺

# Recap – Hydration steps

ANGULAR ARCHITECTS

1) Build Application with Angular
2) Server-Side Rendering with node.js
3) Transfer to / Load by client (HTTP)
4) Browser Parsing & Rendering
5) Full app Hydration
6) Full app Event Replay

if(@defer) Load, Parse, Render, Hydrate & Event Replay

if(hydrate) Load, Parse, Hydrate & Event Replay

# Conclusion

ANGULAR ARCHITECTS

Initial Load verbessern durch SSR

SSG (Prerender) für statischen Content

@defer für Komponenten below-the-fold

Incremental Hydration für above-the-fold

Ausprobieren & Herumspielen!

# Ressources

ANGULAR ARCHITECTS

- angular.dev/guide/performance - official SSR docs
- angular.dev/guide/incremental-hydration - official IH docs
  - youtube.com/watch?v=v5KTJGEYLsM – IH intro by Jessica Janiuk
- angulararchitects.io/blog/complete-guide-for-server-side-rendering-ssr-in-angular/
- angulararchitects.io/training/angular-performance-optimization-workshop/

# Danke für die Aufmerksamkeit

ANGULAR ARCHITECTS

[X] @LX_T → *Updates zu Angular*

[web] https://lxt.dev → *Slides & Source*

Remote Workshops
und Consulting

https://angulararchitects.io