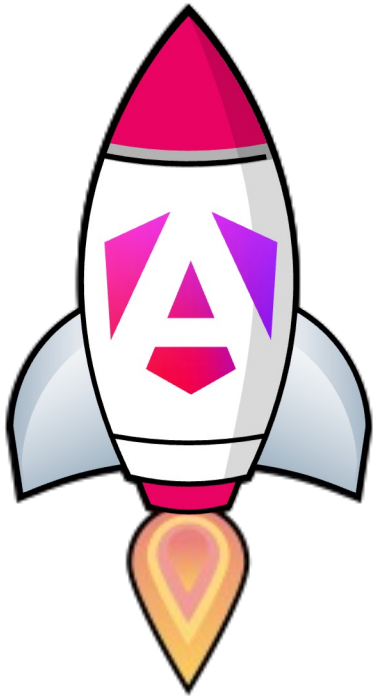
ANGULAR
ARCHITECTS

Initial Load
Assets & Build

Alexander Thalhammer | @LX_T

Outline - Initial Load Performance



- Assets & Build
- Lazy Loading & Deferrable Views
- SSR & SSG

Assets & Build

- Use web performance best practices
- Use **NgOptimizedImage** (since NG 14.2.0)
- Use build optimization & Tree Shaking
- Avoid large 3rd party deps
  - like CSS or component frameworks

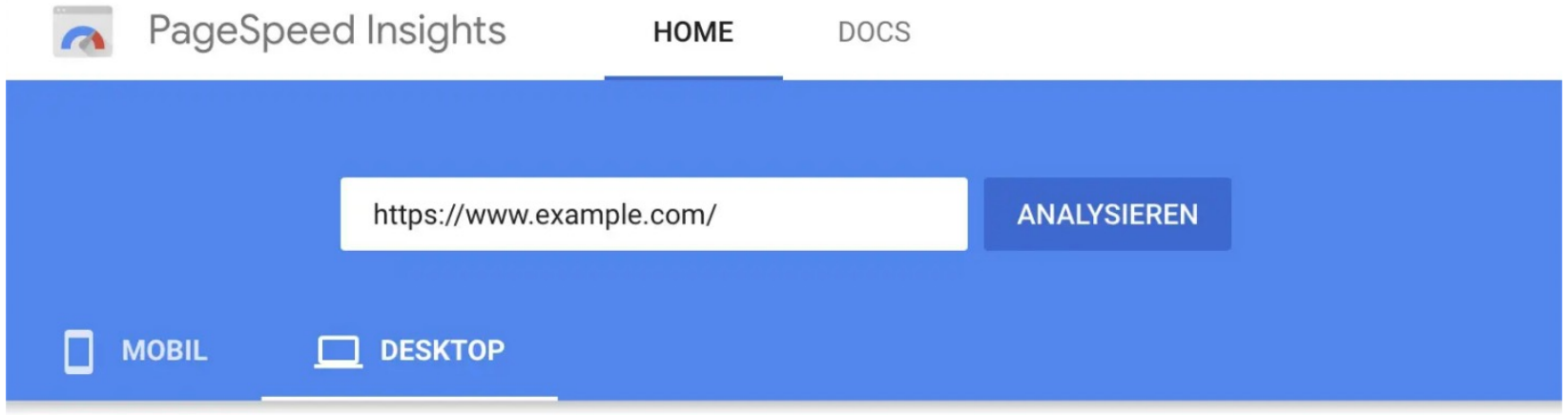
Web Performance - Identify Issues

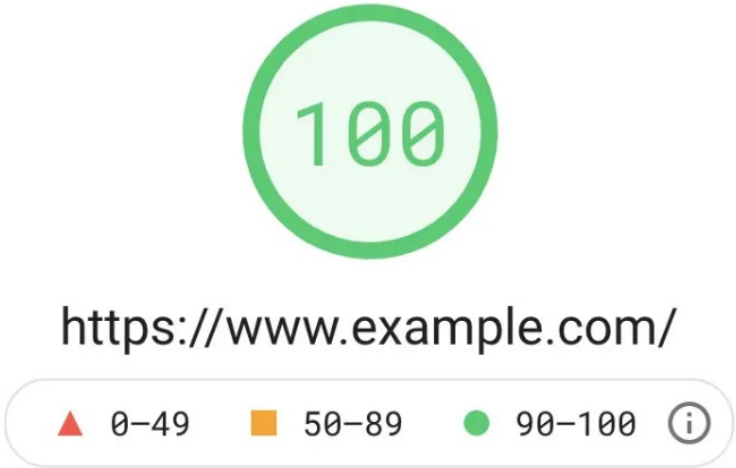
– Lighthouse & PageSpeed Insights

– WebPageTest.org or

– Chrome DevTools

Web Performance - Best Practices





Web Performance – Issues & Solutions

- *Slow server infrastructure → HTTP/3 not HTTP/1.1, CDN*
- *Browser Caching not configured correctly → Configure it*
- *Compression not configured correctly → Brotli or Gzip*
- *Images not optimized → Use .webp, .avif or .svg*
- *Images not properly sized → Use srcsets*
- *Unused JS code or CSS styles → Clean up & lazy load assets*
- *Too large assets, many assets → Clean up & lazy load assets*
- …

NgOptimizedImage

lazyLoading / defer

NgOptimizedImage

ANGULAR
ARCHITECTS

Use NgOptimizedImage (since NG 14.2.0)

- Problem: Lighthouse or PageSpeed image errors / warnings
- Identify: Lighthouse & PageSpeed / WebPageTest or DevTools
- Solution: Use NgOptimizedImage's **ngSrc** instead of **src** attr.
  - Takes care of intelligent **lazy loading** of images outside viewport
  - **Prioritization** of critical images ("above-the-fold")
  - Together with an **image provider** it creates **.webp** format for us ☺
  - Also creates **srcset & sizes attributes** (for responsive sizes, since NG 15)
    - Also supports high-res devices ("Retina images")

Demo

ngSrc

ANGULAR
ARCHITECTS

Build optimization

Use Build Optimization – Problem

– Too large build
– Downloading the App takes too much time / ressources

Identify
– CSS / JS Files not minimized
– Unused JS code included in the build

Use Build Optimization – Solution

- Use production build
  - *ng b(uild) (--c production)*
- Set up angular.json correctly
- New builder in NG17

```
"builder": "@angular-devkit/build-angular:application",
[…]
"production": {
  "optimization": true
},
```

```
"@angular-devkit/build-angular:browser",

"production": {
  "buildOptimizer": true,
  "optimization": true,
  "vendorChunk": true
}
```

Demo

NG Build Configuration

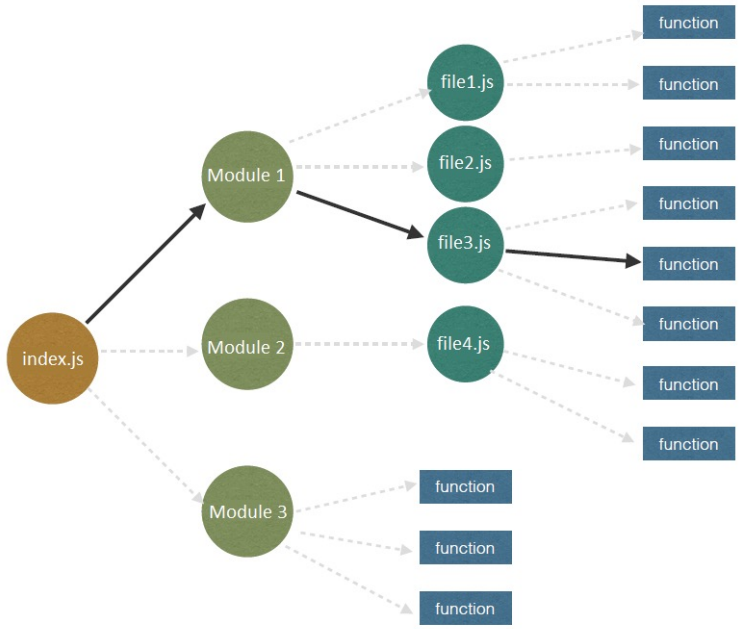
ANGULAR
ARCHITECTS

Advantages of Angular Ivy (since V9)

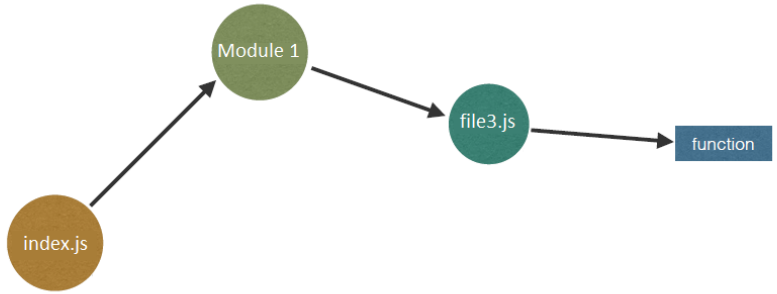
- Angular ***ViewEngine*** itself was not tree-shakable
- Default since NG 10, for libs default since NG 12
- AOT per default → You don't need to include the compiler!
- ***Ivy*** also does a lot of under the hood optimization
- Tools can easier analyse the code
  - Remove unneeded parts of frameworks
  - Called ***Tree Shaking***
    - Also 3rd party and
    - Even our own libs

Tree Shaking

Before Tree Shaking



After Tree Shaking



Avoid large 3rd party deps / CSS frameworks

- Problem: *Importing large 3rd party libraries, not treeshakable*
  - *moment*
  - *lodash*
  - *charts*
  - ...
- Identify: Source Map Analyzer or Webpack Bundle Analyzer
- Solution 1: Remove / replace that lib / framework
  - *moment → luxon, day.js or date-fns*
  - *lodash → ~~lodash-es~~ → es-toolkit*
  - *charts → charts.js*
  - ...
- Solution 2: Lazy load that thing

Demo

Large 3rd Party Deps

ANGULAR
ARCHITECTS

Lab 03 Initial Load

NgOptimizedImage / NG Prod Mode / 3rd party deps

Assets & Build

- Use web performance best practices
- Use **NgOptimizedImage** (since NG 14.2.0)
- Use build optimization & Tree Shaking
- Avoid large 3rd party deps / CSS frameworks

Recap

References

- Optimize the bundle size of an Angular application
  - https://www.youtube.com/watch?v=19T3O7XWJkA

- Angular Docs
  - NgOptimizedImage
  - NG Build

ANGULAR
ARCHITECTS