Techniques for lazy loading TS JS in A

Vojtech Mašek

Head of engineering

@ flowup

/ vmasek

/ VojtechMasek

/ @vmasek

Import size/cost plugin

- Plugin for  /  : *“Import cost”*
- Overview of what & how much we are importing

```
import 'fast-deep-equal';  1.29 kB (gzip: 603 B)
import 'hammerjs';  21.16 kB (gzip: 7.37 kB)
import 'rxjs';  50.5 kB (gzip: 12.1 kB)
import 'core-js';  90.51 kB (gzip: 30 kB)
import 'ngx-image-cropper';  516.56 kB (gzip: 128.66 kB)
import 'ngx-toastr';  522.77 kB (gzip: 128.74 kB)
import 'firebase';  827.21 kB (gzip: 223.94 kB)
import 'ngx-quill';  820.3 kB (gzip: 191.19 kB)
```

Bundle Phobia

rxjs@7.1.0

tree-shakeable | side-effect free | 1 dependency

BUNDLE SIZE

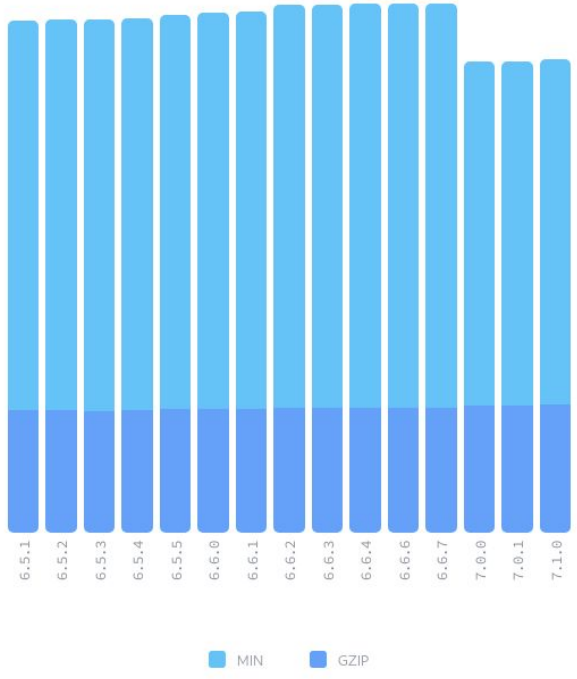
42 kB MINIFIED

11.4 kB MINIFIED + GZIPPED

DOWNLOAD TIME

28 ms SLOW 3G

2 ms EMERGING 4G

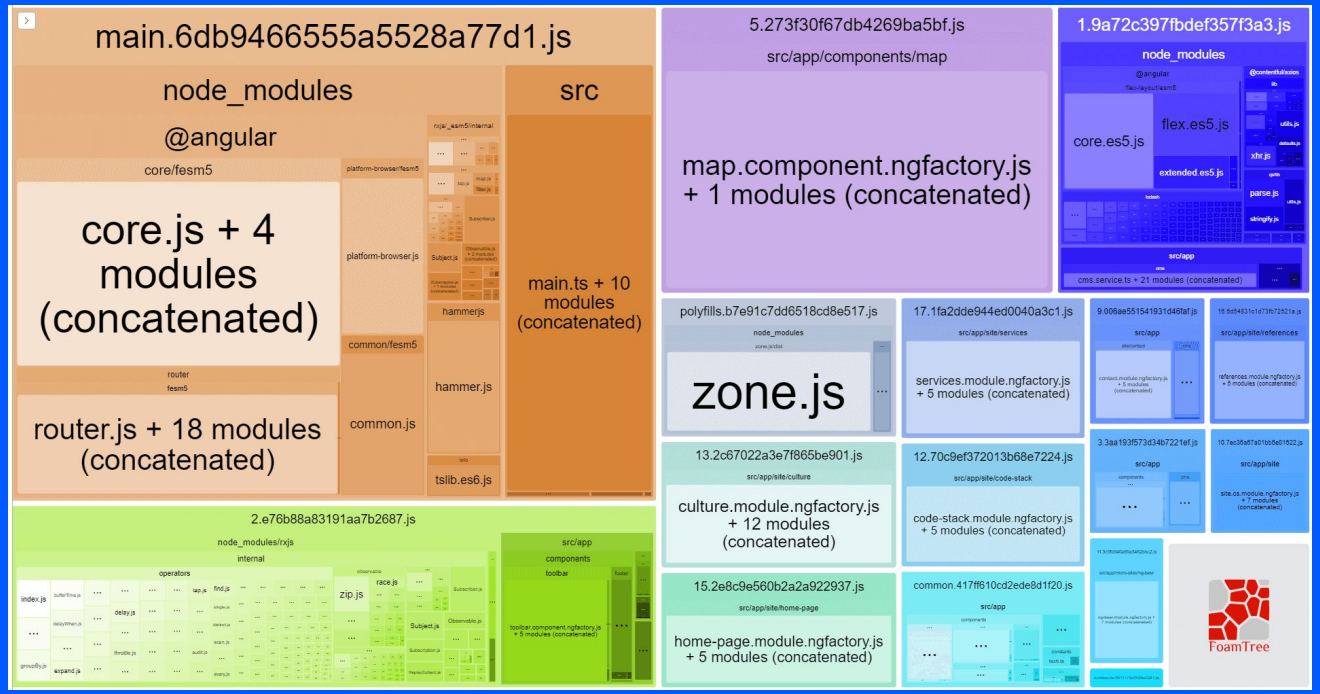


https://bundlephobia.com/result?p=rxjs

Bundle size analysis

flow^up

- Build your app `ng build --prod --stats-json`
  - This will add an extra *stats.json* with analysis of the application bundle (chunks)
- Run `npx webpack-bundle-analyzer dist/stats.json`



Lazy loading feature modules

flowup

- Application loads modules when they are needed
- Integrated solution for route level lazy loading and code splitting
  - Use `loadChildren` instead of directly referencing `children` in routing

```
const siteRoutes: Routes = [
  {
    path: WebsiteSection.HomePage,
    loadChildren: () => import('./home-page/home-page.module').then( onfulfilled: m => m.HomePageModule),
  },
  {
    path: WebsiteSection.Services,
    loadChildren: './services/services.module#ServicesModule',
  },
];
```

https://angular.io/guide/lazy-loading-ngmodules

Dynamic import

flowup

- ES2020 introduces function-like `import()` that returns a promise
- Use dynamic `import()` to lazily load code
  - Avoiding the load, parse, and compile cost until needed

```
// tsconfig.app.json
{
  "compilerOptions": {
    /* ... */
    "module": "ESNext", // or ⩾ ES2020
    /* ... */
  }
}
```

Browser compatibility is not affected because all module resolutions are solved by webpack behind of Angular CLI.

https://v8.dev/features/dynamic-import

Lazy loading in Angular pipe

flowup

Pipes are great at running code on demand.

Will execute only if the code or 3rd party library is actually needed.

MarkDown example

```
@Pipe({ name: 'markdownToHtml' })
export class MarkdownToHtmlPipe implements PipeTransform {
  transform(markdown: string): Observable<string> {
    return markdown
      ? from(import(/* webpackPrefetch: true */ 'marked'))
        .pipe(map(({ parse }) => parse(markdown)))
      : of('');
  }
}
```

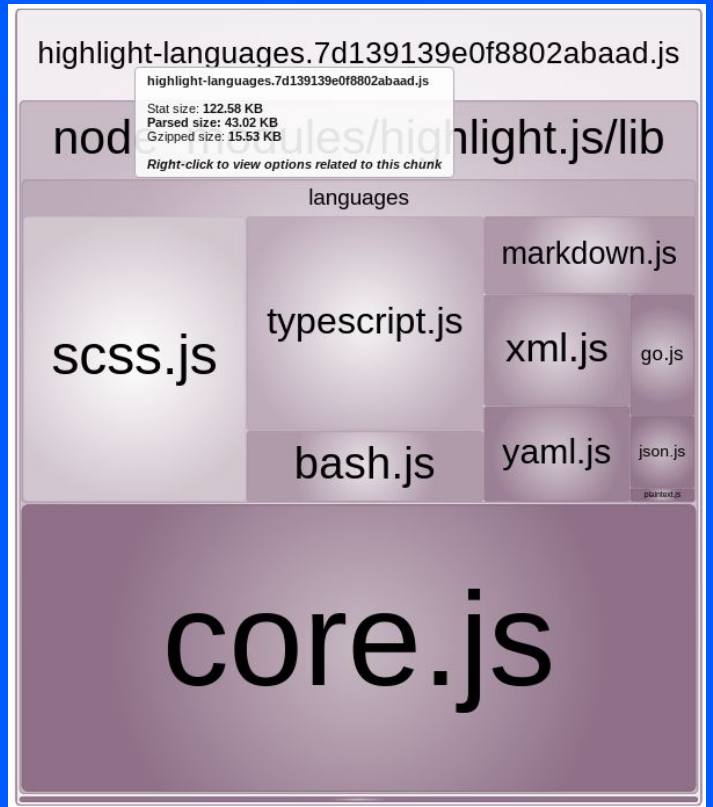
```
<div
  class="markdown-container"
  [innerHTML]="description | markdownToHtml | async | sanitize: 'html'"
></div>
```

Lazy loading in A pipe

flowup

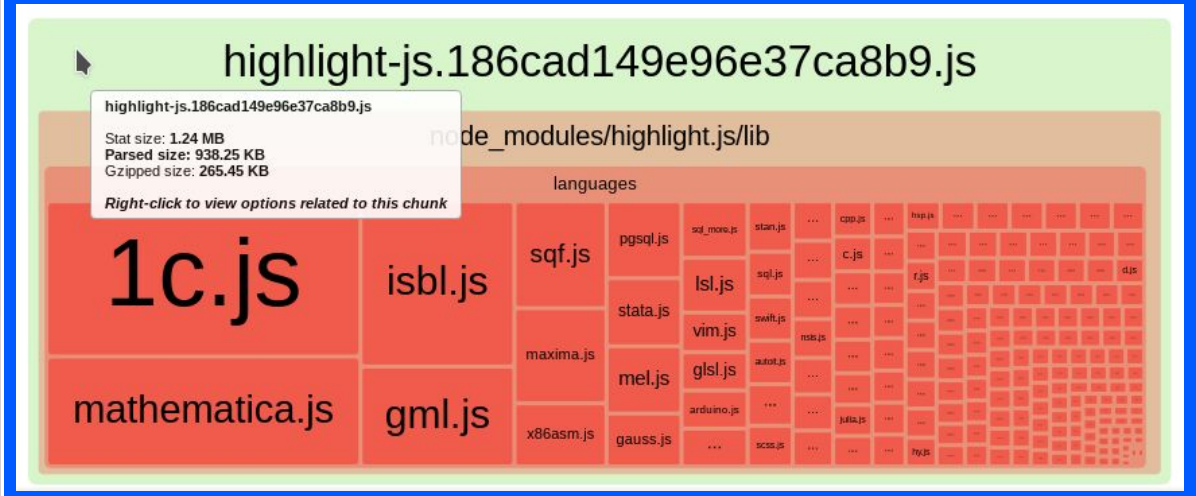
~95% reduction in size.

938 kB → 43 kB



HighlightJS example

- → Loaded only if there is code to highlight.
- → No unnecessary JS to load & parse



Lazy loading in pipe

flowup

HighlightJS example

```
<pre>
  <code
    class="hljs"
    [innerHTML]="code | highlight:code.lang | async | sanitize:'html'"
  ></code>
</pre>
```

```
@Pipe({ name: 'highlight' })
export class HighlightPipe implements PipeTransform {
  transform(value: string, lang?: string): Observable<string> {
    return value
      ? from(import(/* webpackPrefetch: true */ './highlight-languages')).pipe(
          map(({ highlightJS }) =>
            lang
              ? highlightJS.highlight(lang, value).value
              : highlightJS.highlightAuto(value).value,
          ),
        )
      : of('');
  }
}
```

```
import hljs from 'highlight.js/lib/core';
import bash from 'highlight.js/lib/languages/bash';
import go from 'highlight.js/lib/languages/go';
import xml from 'highlight.js/lib/languages/xml';
import json from 'highlight.js/lib/languages/json';
import markdown from 'highlight.js/lib/languages/markdown';
import plaintext from 'highlight.js/lib/languages/plaintext';
import scss from 'highlight.js/lib/languages/scss';
import typescript from 'highlight.js/lib/languages/typescript';
import yaml from 'highlight.js/lib/languages/yaml';

const langHighlighters = {
  bash,
  css: scss,
  go,
  xml,
  javascript: typescript,
  js: typescript,
  json,
  markdown,
  plaintext,
  scss,
  typescript,
  yaml,
};

function registerHighlightJS(): typeof hljs {
  Object.entries(langHighlighters).forEach(([langName, highlighter]) =>
    hljs.registerLanguage(langName, highlighter),
  );
  return hljs;
}
export const highlightJS = registerHighlightJS();
```

webpack & “dynamic” `import()`

flowup

- `import(foo)` is not supported because `foo` could potentially be any path to any file.
- At least some path must be specified.
- Bundling can be limited to a specific directory or set of files.
  - Every module that could potentially be requested is included.

```
1 const language = detectVisitorLanguage()
2
3 import(`./locale/${language}.json`).then((module) =>
4 { // do something with the translations
5 })
```

https://webpack.js.org/api/module-methods/#dynamic-expressions-in-import

webpack magic comments

flow^up

- Name chunks
- Preload and/or prefetch
- Select different modes
  - lazy (default)
  - lazy-once (bad name)
    - bundles files together in one bulk
  - eager
    - create no extra chunk and bundles in current chunk
  - weak
    - resolved only if chunk was already loaded (never performs a request)
- Include/exclude files
- Select only the specified exports of module

```
// Multiple possible targets
import(
  /* webpackInclude: /\.json$/ */
  /* webpackExclude: /\.skip-import\.json$/ */
  /* webpackPrefetch: true */
  `./locale/${language}`
);

// Single target
import(
  /* webpackChunkName: "my-chunk-name" */
  /* webpackMode: "lazy" */
  /* webpackExports: ["default", "foo"] */
  /* webpackPreload: true */
  'some-module'
);
```

https://webpack.js.org/api/module-methods/#magic-comments
https://webpack.js.org/guides/code-splitting/#prefetchingpreloading-modules

flow^up

Prefetch

Preload

- Hint to the browser that a resource might be needed
  - Leaves deciding whether and when loading it is a good idea or not to the browser.
- Useful if resource is likely to be used for future navigations or actions.
  - “One click away”

- Declarative fetch
  - Enables forcing the browser to make a request without blocking the onload event
- Use when you have high-confidence resource will be used in the current page.
  - key scripts
  - fonts
  - above fold images
- Easily controlled granularity with “as” specifying the resource type.

https://medium.com/reloading/preload-prefetch-and-priorities-in-chrome-776165961bbf
https://www.smashingmagazine.com/2016/02/preload-what-is-it-good-for/

webpack & code splitting

flow^up

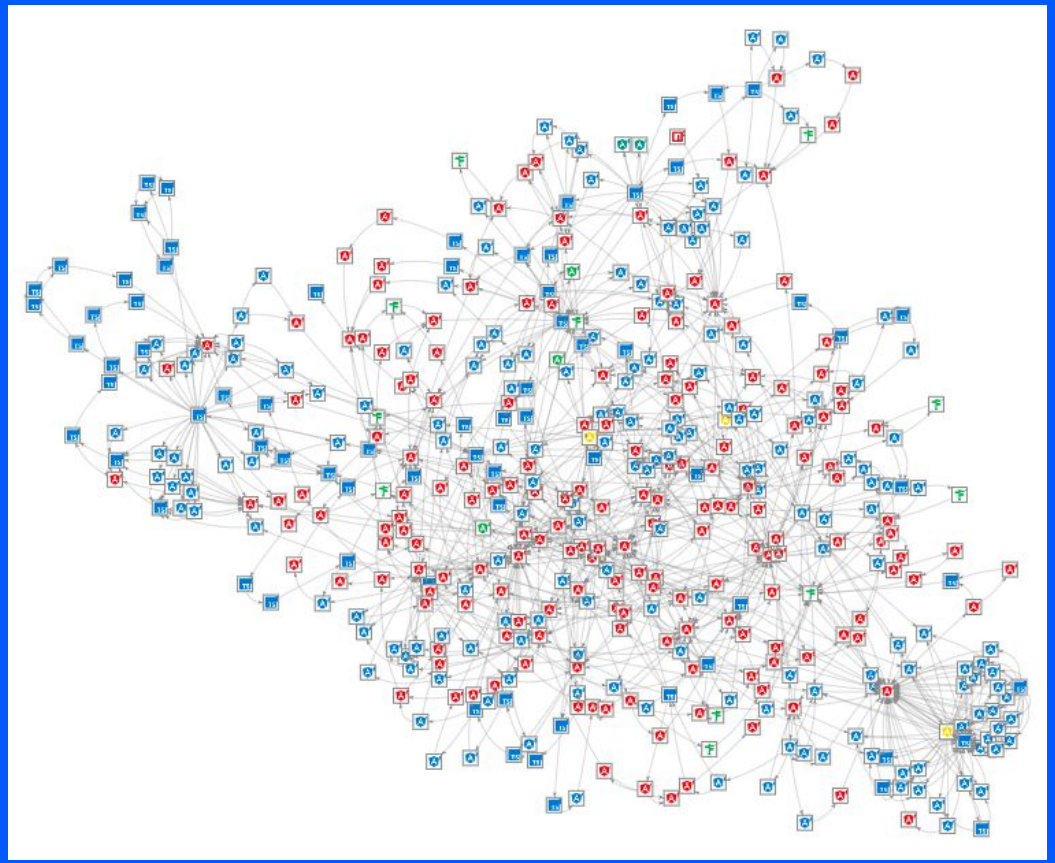
- Vendor chunk
  - 3rd party code usually coming from node_modules.
  - It will be loaded synchronously with main chunk.
- Async chunks:
  - Separate files for code which can be lazy-loaded.
  - For example a file for every Route loaded feature/module.
- Common chunk
  - Common code shared between different chunks.

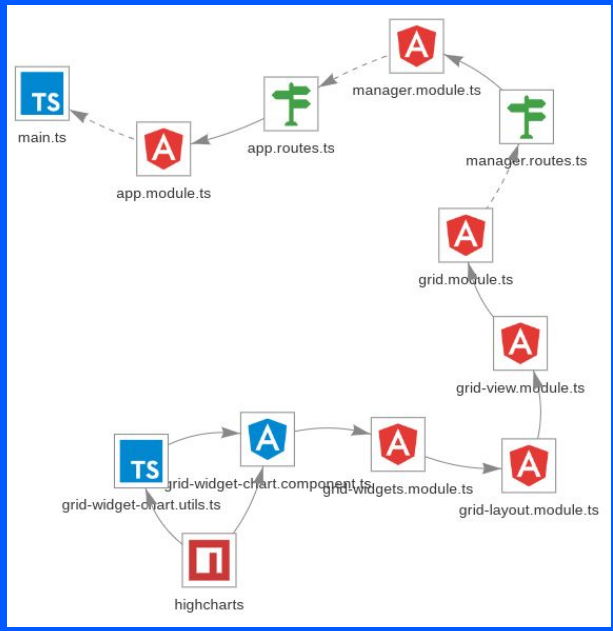
https://webpack.js.org/guides/code-splitting/

How big can share chunks be?

flow^up

Sometimes pretty big.





github.com/rx-angular/import-graph-visualizer

Loading components dynamically

flow^up

- Componen
  ngIf / ngSw
  when used
- Using librar
  achieve “la
  that will ac
  lazy-loadin


```
sync as c">
ntFactory.componentType"
.injector"
egy.OnPush,
= new EventEmitter();
y-chart.component'
22 }
```

https://www.npmjs.com/package/@ngxd/core

What we didn't cover

flowup

- Dynamic component loader service
  - Respect component module, its providers & context.
- Lazy loading using "defer" strategies
  - Wrap usages of components using "heavy" libraries with asynchronous conditions.
  - Useful when working with messy synchronous code.
- Performing lazy loading on router resolver level.
  - Handy way to lazy-load data or code if route plays big part in whether certain feature is needed.

Tips

flow^up

- Remember that any lazy loading strategy is most effective in combination with proper caching.
- Any resource that is not essential for the page and could be loaded lazily, should be loaded lazily.
- Do preload/prefetch code that is likely to be used.
- Write three-shakable code, it enables more efficient code-splitting & embraces good architecture practices.
- Observe your bundle size before you run into troubles.

Q&A

Vojtech Mašek

Head of engineering

@ flowup

/ vmasek

/ VojtechMasek

/ @vmasek