Debug Angular apps in production without revealing source maps

Alternate approaches to handle source maps in production

Kevin Kreuzer Follow

Dec 18, 2018 · 12 min read

When our app gets deployed to production, we often encounter different code than we edited during development. Our code gets modified and optimized in various ways during the build process.

TypeScript gets transpiled, minified and uglifed. The resulting JavaScript bundle is as small as possible and able to run in the

Browser.

All those concepts are great because they improve the performance of our application. But we are looking at JavaScript that is hard to read and impedes debugging.

But there's a solution: **source maps**!

source maps—a short introduction 💁‍♀️

At its core, a source map is a JSON file that contains all the necessary information to map the transpiled code back to the original sources. Pretty cool!

Technically a source map is just a JSON file that contains the following fields:

- **version**: indicates the source map spec version
- **file**: the name of the transpiled file this source map belongs to
- **sourceRoot**: basePath—sources are located relative from here
- **sources**: the path to the original source files (for example TypeScript files)
- **sourcesContent**: optional property that can contain your whole source code. When the source code gets inlined in this property, sources do not need to be hosted to be retrieved.
- **names**: method or variable names found in the code
- **mappings**: this is where the whole magic happens. Technically the mappings property is a very big string that contains Base64 VLQ (Variable Length Quantity) values. Those values help to find the original position in the source files.

> ℹ *Originally source maps where about 10 times as large as the original code. By using VLQ they managed to decrease the source map size.*

How are source maps retrieved?

To retrieve source maps we need to tell the browser where they live. We can specify the sourceMappingURL at the end of our file by adding the following line:

```
//# sourceMappingURL=pathToSourceMaps
```

With this information, the browser can download the source map file and interpret its content to create the mappings.

> ☝️ *The browser only downloads the source maps when the developer tools are open. For normal users there's no performance impact.*

Instead of adding a comment to the end of your file you also can send the path as a value of the `SourceMap` HTTP header in the response that fetches the minified JavaScript file.

```
SourceMap: pathToSourceMap
```

The second possibility enables you to toggle source maps on the server side without touching your minified JavaScript files.

Source maps during development and production

Development and production builds differ.

During development, it makes sense to have full source maps since we focus on tooling, development experience or hot module replacement.

In production, on the other hand, we focus on performance—fast initial loads with small bundles.

Should I enable source maps during production?

The answer to this question strongly depends on your project. If you are working on an open source project then for sure it's yes.

But most of us don't work on OpenSource projects on our daily jobs. In enterprise projects, there are good reasons why you don't want to expose your source code.

- We do not want to expose the whole application code to be easily readable to the rest of the world.
- Faster builds
- **There are different approaches to have source maps without exposing them.**

> *Follow me on Twitter or medium to get notified about my newest blog posts!*🐤

Source maps in Angular—how to enable them in production without revealing them? 😎

Angular CLI lets us configure if we want source-maps or not. It then passes this information to the underlying Webpack.

To explore source maps in Angular let's start with a brand new Angular 7 project generated by Angular CLI.

```
ng new sourceMapInspector
```

Before we start, let's add a minimalistic feature to the app. We add a button that changes the title from 'sourceMapInspector' to 'awesome app'.

In our component we add a function to change the title:

```
public changeTitle(): void {
  this.title = 'awesome app';
}
```

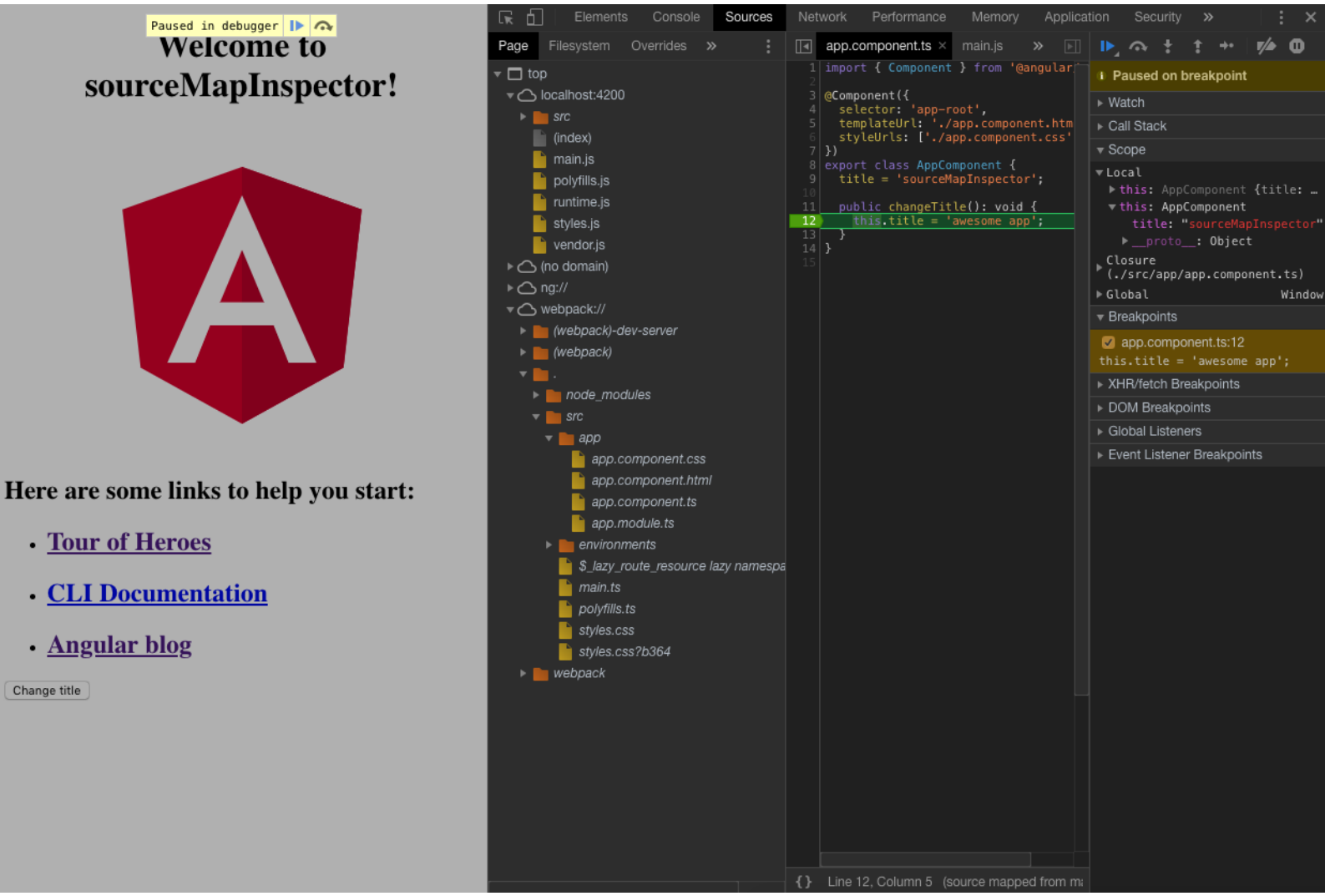
and call it inside our HTML:

```
<button (click)="changeTitle()">Change title</button>
```

source maps during development 🥷

To explore how source maps behave during development. Let's start our app and open the dev tools.

```
ng serve
```



Source maps help us display our original source under `webpack://` in the development tools.

We can now open `app.component.ts` and put a breakpoint inside the `changeTitle` function. By clicking on the “Change title” button we then hit our breakpoint.

Source maps are generated when we use `ng serve`. If we look at the last line inside our `main.js` we can see where the browser fetches the source maps.

```
//# sourceMappingURL=main.js.map
```

That’s great for development. We have full source maps and can easily debug our code.

Angular production builds and source maps 👨‍🔧

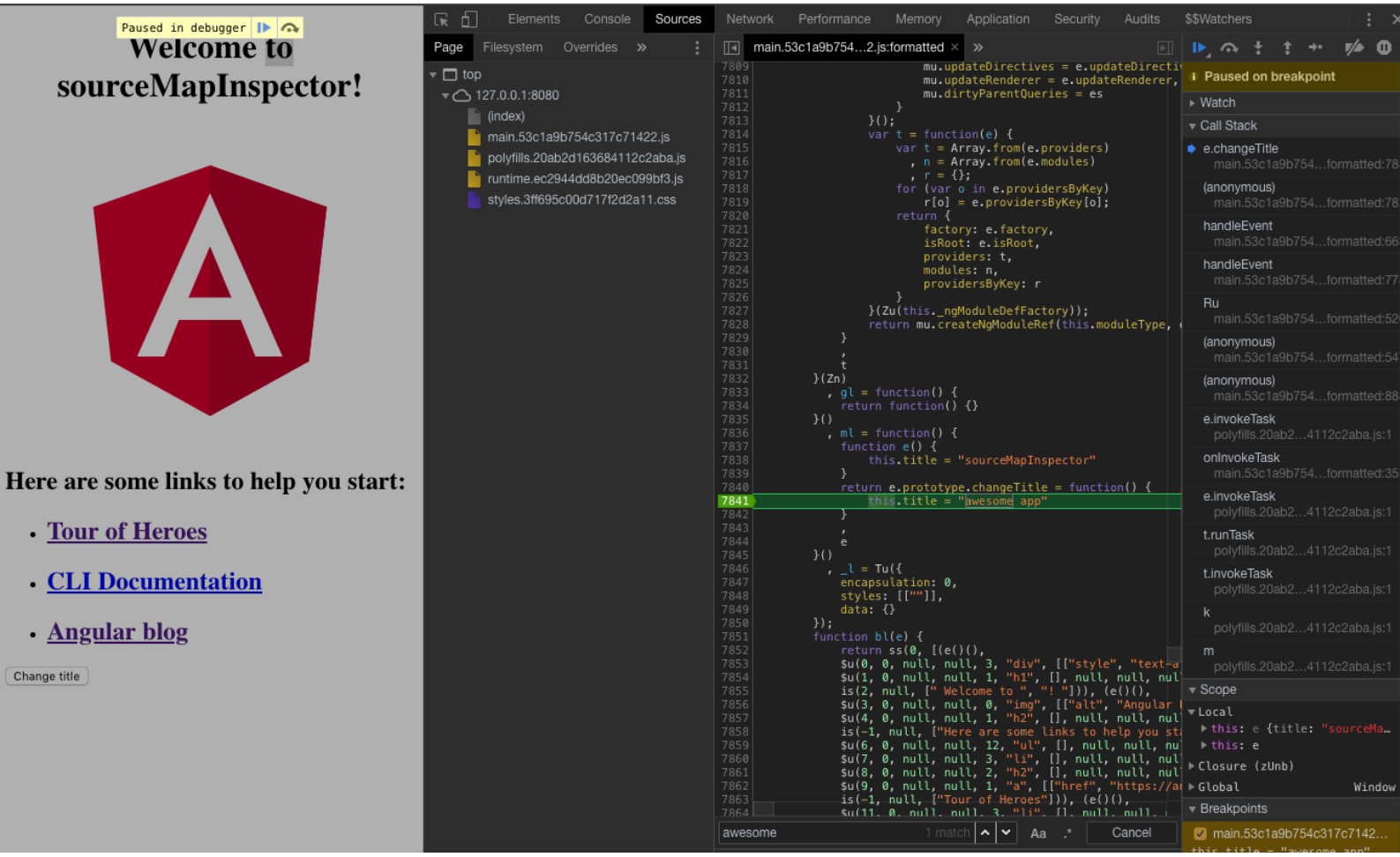
Let’s inspect how production builds in Angular behave in terms of source maps. We can run a prod build with the following command.

```
ng build --prod
```

The `dist` folder now contains the bundled files without maps. Let’s change into the `dist` folder and run the application on an HTTP server to get an impression on how it will look in production. I like to use the npm module `http-server` as local web server.

> `http-server` *is a simple, zero-configuration command-line http server that can be installed with* `npm i -g http-server`

So let’s run our production build and open the dev tools to debug our feature.



There's no sources and no Webpack menu item to click on. 🤔

Where do we set our breakpoint? We manually need to find our function in the transpiled JavaScript file which is cumbersome.

We set the breakpoint on line 7841 even though our app only contains a few lines of code. 😳

Identifying where to set the breakpoint in the transpiled code might be OK for such a small app. But trust me, it gets hairy when our app grows.

So how can we add source maps to production builds?

Adding source maps to an Angular production build

The `angular.json` file contains an architect property that allows us to specify if we want to use source maps for our production build.

```
"architect": {
  "build": {
    ...
    "configurations": {
        ...
        "sourceMap": false,
    }
  }
}
```

To enable source maps we either need to change the `sourceMap` attribute to `true` or we override it by passing `--source-map` to our `ng build` command.

This approach would add source maps to our production build and fetch them in production so that everybody can access our sources. Not so cool 😟

To prevent automatic fetching of source maps, we need a way to stop the creation of the comment at the end of the bundle.

For a long time, this was only possible with some shell script magic or by extracting the webpack config which is usually not recommended. But since 7.2, the Angular CLI offers us a fine grain control over source maps that allow us to handle such cases.

Fine grain control over source maps 🤩

Angular 7.2 gives us more fine grain control over source maps. Instead of a `boolean`, the `sourceMap` property now accepts an object with the following properties.

```
"sourceMap": {
  "hidden": true,
  "scripts": true,
  "styles": true
}
```

The `scripts` and `styles` property allows us to control if we want to generate source maps only for scripts or just for styles. The hidden property is the one which is important to us. It allows us to create hidden source maps. 😍

> *If you set the hidden property you explicitly need to set the scripts and styles property.*

Hidden source maps ❤️

Hidden source maps is a fancy term for a simple concept.

> *Generate source maps but do not add the comment to fetch them in your bundle.*

Source maps itself do not differ for regular builds or builds with hidden source maps. Only the generated bundle varies in one line—the comment added by web pack to retrieve source maps.

Let's have a look at a bundle generated with source maps.

Notice the comment at the end. When we open the dev tools, the browser will interpret this comment and try to fetch source maps. Now let's have a look at a bundle generated with hidden source maps.

We can see that no comment is added to the end of the file. Therefore the browser will not attempt to fetch source maps.

Hidden source maps are an essential feature that was missing for a long time. It is especially useful for dealing with source maps in production.

Prevent revealing source maps

On the internet, you can read about different approaches to ensure that your sourcs are not exposed.

- Deliver source maps over VPN. Source maps are generated on production builds but they are hosted on a VPN server and will therefore only be downloaded inside your company. This approach requires an adjustment of the URL in the comment. Currently, the Angular CLI does not offer a way to adjust this URL through the CLI.
- Use some toggling on the server to set the `SourceMap` header dynamically. This approach makes use of hidden source maps. Source maps can be generated and then be hosted on the server. Be aware that once toggled in; source maps will be fetched for all your clients.
- Use error tracking tools like Sentry. Error tracking tools allow you to upload your hidden source maps. With the help of your uploaded source maps, they are then able to display correct error messages. Those tools allow you to view error messages but do not allow you to debug your code.

Those are all great approaches. But they still require some extra effort or some special tooling. What if there's a way to deal with source maps in production with some simple npm scripts?

After some research and experiments, it turns out that there is! Thanks to Chrome's **"add source map"** feature!

Upload local source maps 🏋️

We can always regenerate source maps locally and upload them at a later point.

First, we will adjust the architecture property in our `angular.json`.

```
"sourceMap": {
  "hidden": true,
  "scripts": true,
  "styles": true
}
```

This change will generate hidden source maps on `ng build --prod`.

Next, we need to create a `postbuild` script which merely deletes the source maps we just generated.

```
"postbuild": "rm dist/sourceMapInspector/*.map"
```

The `postbuild` script is needed because otherwise, we would deliver source maps. Due to the missing comment, they would not be fetched. But they are still provided.

To debug in production, we manually upload the source maps. But from where do we get them? We just deleted them.

The answer is simple, we need to regenerate them.

```
"build:sourcemaps": "ng build --prod --output-
path=localSourceMaps"
```

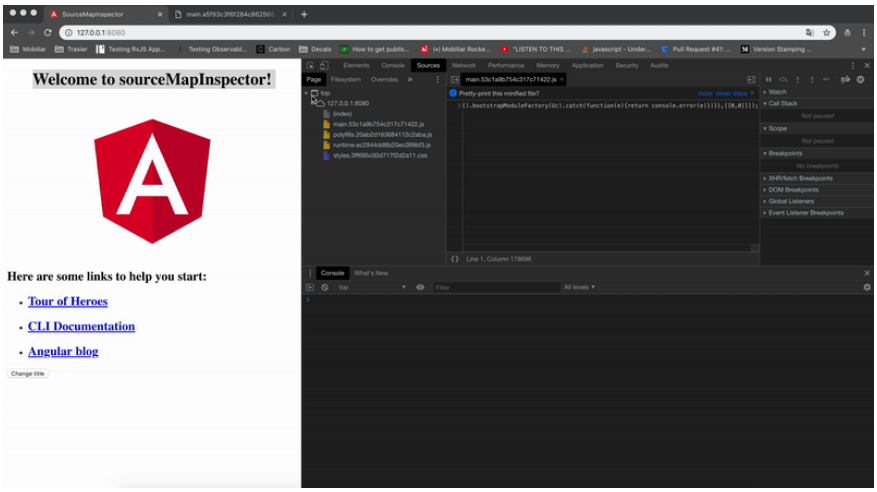
This script builds our project with hidden source maps into a folder called `localSourceMaps`.

> ☝️ *Be sure that you checked out the commit that matches the state of the app you want to debug.*

> *We build the source maps in a separate npm script to avoid the execution of the postbuild script. To avoid avoid possible confusion with the* `dist` *folder we also build to a new path called* `localSourceMaps`.

We can now use the Chrome dev tools to upload the source maps from our local file system. Therefore open the dev tools and right click in your `main[hash].js`. Now enter the path to your source maps generated by the script above.

The path must be added in the following way: `file///pathToFile`.



Uploading source maps from your local machine is quite nice. But still, it requires some manual steps that each developer has to execute. Finding the matching commit, regenerate source maps and upload them. Can we get rid of some of them? 🤔

hide uploaded source maps 👻

Not revealing source maps doesn't mean that we can't generate them and deploy them. What if we don't tell the browser yet where they are.

Again we start off by setting the hidden property in our `angular.json`.

```
"sourceMap": {
  "hidden": true,
  "scripts": true,
  "styles": true
}
```

Next up we need to hide our source maps. We merely create another folder in our `dist` and move the source maps to it.

```
"postbuild": "mkdir dist/sourceMapInspector/sourceMaps; mv
dist/sourceMapInspector/*.map
dist/sourceMapInspector/sourceMaps"
```

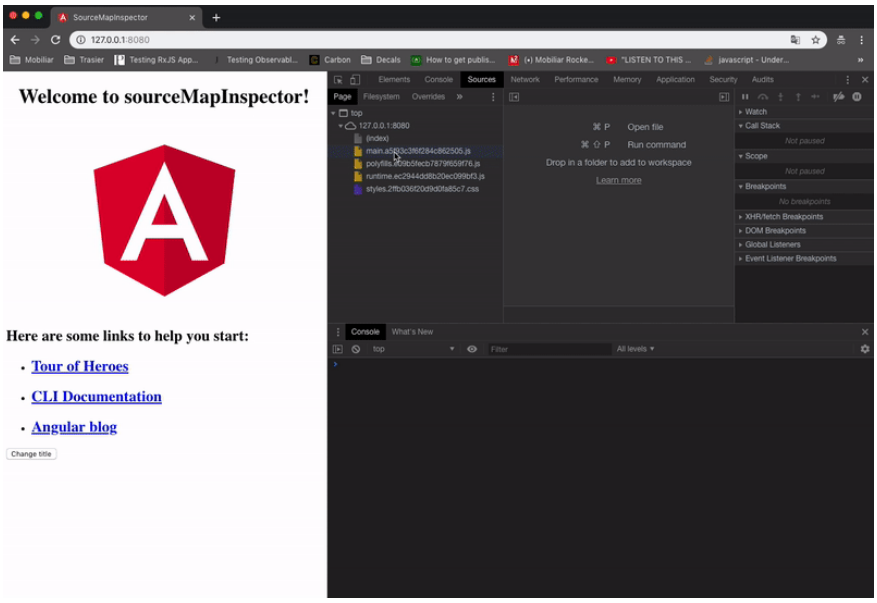
> *You are free to choose the directory name. Instead of sourceMaps you can also choose some name that is hard or impossible to guess.*

With this approach, the source maps get deployed to our server under the directory we moved them to.

To add them we again take advantage of the Chrome dev tools.

We open the `main[hash].js` file and add them by right clicking in our transpiled JavaScript file. This time though we enter the path to our sourceMap folder on the server.

`http://localhost:8080/sourceMaps/main.js.map` .



Nice but I am yet not on Angular 7.2, what to do?

Don't worry. All the approaches above are also possible on earlier Angular versions. There are just some things you need to be aware of.

The first difference is that you either set the architecture property to true or adjust your build command to `ng build --prod --sourceMap` .

If you choose to generate source maps at a later point, which is the case when you upload them from your local machine, you should also be aware of the following.

Webpack always uses the same hash for the bundle if the bundle content did not change. As soon as we do dynamic stuff like adding a timestamp to our bundle, a new hash will be generated.

You may wonder why this is important?

Remember the structure of a **source map**? The source map contains a `file` property which points to the original file the source map belongs to.

If we built our app in the first place without source maps, the comment at the end of the line would be missing, and therefore the file hash would differ from the hash in our source maps.

The name of the original file also contains a hash. If the hash of the property inside the source map doesn't match with the hash of the transpiled file we have generated a wrong source map which sometimes leads into troubles.

Therefore it is important that you build with source maps in the first place, delete them and upload them later.

Name your chunks 🖨

Adding source maps manually when we need them is quite awesome. Keep in mind that we can only add one source map as there's no option to add the source maps for all files.

This isn't too bad because we usually debug one file. In a lazy loading scenario, I would still recommend naming your chunk. Otherwise, you will find yourself having a hard time searching for the right bundle.

To name your chunks simply add `--named-chunks` to your build command.

Conclusion 🧐

Source maps are great! Not just during development but also in production.

For an open source project, I would always add them. For a closed source project we can have them in production without exposing them to the outside world.

We can take advantage of the fine grain control over source maps added in Angular CLI 7.2. Use hidden source maps for your productive app. They solve a lot of problems.

Hidden source maps and the Chrome dev tools offer us a simple way to add source maps as soon as we need them without much effort.

🙏 Please give some claps by clicking on the clap 👏🏻 button below if you enjoyed this post.🧞‍♂️

Claps help other people finding it and encourage me to write more posts

Feel free to check out some of my other articles about Front End development.

Angular: Refetch data on same URL navigation

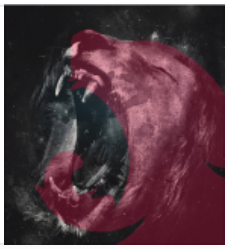
Different approaches with their pros and cons

medium.com

Step up your game, start using Nest!

Nest—one of the best things that happened to server-side JavaScript...

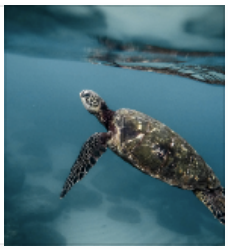
medium.com



Typescript inheritance deep dive 🐋

How inheritance in Typescript actually works behind the curtain?

hackernoon.com



Typescript method overloading

Method overloading is a familiar concept from traditional programming languages...

medium.com