**Tree Shaking**

* Tree shaking is a term commonly used in the JavaScript context for dead-code elimination.
* The new webpack 4 release expands on this capability with a way to provide hints to the compiler via the "side Effects" package.json flag to denote which files in your project are "pure" and therefore safe to prune if unused.

**Declare**

* we might want to use libraries/frameworks in our TypeScript files without getting compilation errors. What can we do? One solution is to use the **declare** keyword. The **declare** keyword is used for ambient declarations where you want to define a variable that may not have originated from a TypeScript file.
* For example, lets imagine that we have a library called myLibrary that doesn’t have a TypeScript declaration file and have a namespace called myLibrary in the global namespace. If you want to use that library in your TypeScript code, you can use the following code:

Ex. declare var myLibrary;

* <https://angular.io/guide/dynamic-component-loader>
* <https://angular.io/api/core/ViewChild>

it has now got two parameters from angular 8 version.

* **Browserlist, Differential loading in angular**

The browserlist is a config file in which you can define your target browsers. It is not something Angular-specific but a standard across many frontend related tools.

Angular uses it in it's build process to decide if differential loading should be used. Basically Angular is able to create a application bundle for modern browsers, which is using new features and can be smaller and more performant, as well as an application bundle for older browsers which don't support the new features and need polyfills to work properly.

With browserlist you can tell Angular which browsers your application will run on and Angular can then decide which bundles to create based on that.

* With angular 9 IVY is the default rendering engine.

entryComponents in your NgModules or had any uses of ANALYZE\_FOR\_ENTRY\_COMPONENTS, you can remove them. They are no longer required with the Ivy compiler and runtime.

* Appshell doesn’t work without routing enabled.
* Centralized error handler:   
  <https://angular.io/api/core/ErrorHandler>
* **Service Worker**

<https://angular.io/guide/service-worker-getting-started>

<https://academind.com/learn/angular/snippets/angular-pwa-service-worker-tutorial/>

* **Progressive Web App:**
  + <https://angular.io/guide/service-worker-getting-started>
  + <https://angular.io/guide/service-worker-communications>
  + <https://angular.io/guide/service-worker-devops>