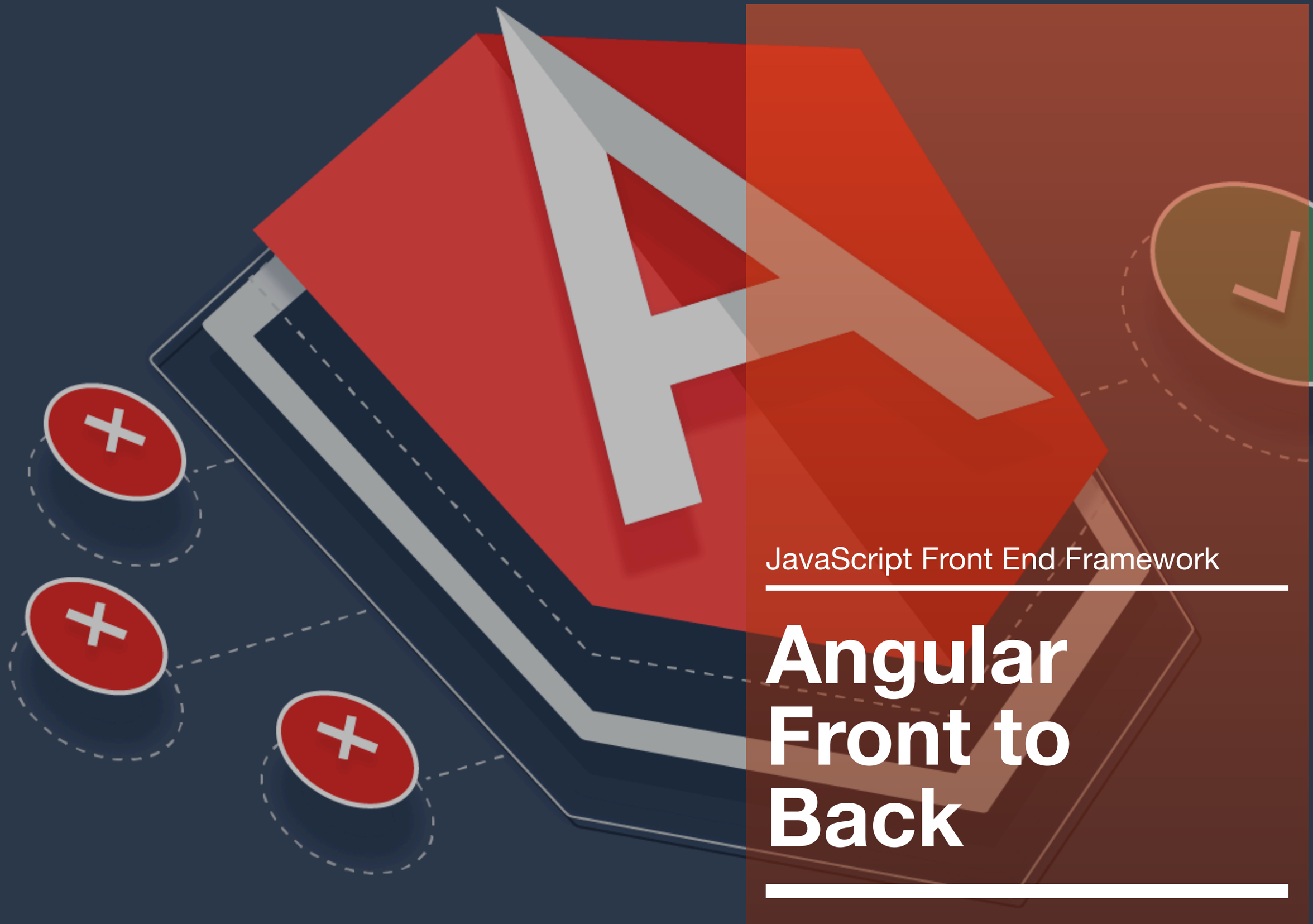


JavaScript Front End Framework

Angular Front to Back

Introduction

What is Angular? Why use Angular?

Angular is a front-end JavaScript framework that was created by and is maintained by Google. This is a framework used to create powerful front-end web applications (*front-end = running on the client side browser*). Note: it is possible to run Angular on the server side, however, at its core it is a front-end JavaScript framework. Angular is also part of a very popular full stack web development called MEAN - MogoDB, Express, Angular and NodeJs.

Angular and AngularJS are NOT the same and are two completely different frameworks.

Angular is great for rapid development & code generation compared to using just vanilla JavaScript (such as routing, HTTP request, form validations and many more). Angular also organises your code more neatly and breaks functionality up into individual components e.g. a navigation bar as one component and the search bar as another. Angular allows us to create dynamic content as opposed to static webpages. Angular is also cross platform and does not matter whether your application is viewed on a Windows, Mac or Linux or regardless of the browser viewed on such as Explorer, Safari, Chrome or Firefox. Finally, Angular has unit testing baked into the framework which makes it easy to create and run unit tests within your Angular applications.

Core Features:
Components, Services, Routing, Testing, Build Tools, Data Binding, Templating, HTTP Module, Observables, Forms Module, Directives, Pipes, Dependency Injection, Animation and TypeScript.

All of these things listed above come packaged into the core framework by default. Contrast this with the React framework where you would have to install libraries separately such as routing, http client, testing etc. However, Angular is going to be a bit more bloated compared to frameworks such as React due to all the core features.

However, with each new version of Angular, the file size are becoming more compact.

Ways to install Angular:

There are a few ways to get started with Angular and these are…

- Angular CLI
- Quickstart Seeds (*boilerplate applications - don't have all the tools the CLI provides*)
- Absolute Scratch (*not recommended*)

The Angular CLI (command line interface) requires Node.js and NPM (a JavaScript runtime and package manager) as a dependency in order to run. The CLI is the standard to quickly build an Angular project. The CLI creates a complete development environment with a dev server, build tools and everything else that you would need.

Version History:

- AngularJS / Angular 1 (*2010*)
- Angular 2 (*2015*)
- Angular 4 (*2017*)
- Angular 5 (*2017*)
- Angular 6 (*2018*)
- Angular 7 (*2018*)

Angular 3 was skipped due to the misalignment of the router package from the rest of the framework packages (there may have been some other issues as well for the skipping of version 3). AngularJS is different from Angular because AngularJS uses controllers & scopes whereas the later frameworks which now uses components. There are some developers that continue to use AngularJS (Angular 1) and this is completely different from Angular (i.e. angular 2 and above). Since Angular 4, there is now a 6 month release cycle for each new version of the framework.

Setup & File Structure

Environment Setup

NodeJS and Node Package Manager (NPM) can both be downloaded onto your machine by following the link below:
https://nodejs.org/en/

Once installed, if you are on a windows, you may wish to also install git bash (*this is not mandatory but is highly recommended terminal over the default windows powershell terminal*) from the link below:
https://git-scm.com/downloads

Finally, you would want to install a code editor such as Visual Studio Code, Atom or Sublime Editor (*the preference is yours*). Some useful extensions for Visual Studio Code to install are:

- Angular v7 Snippets by John Papa
- Bracket Pair Colorizer by CoenraadS

To toggle the display of the terminal within VS Code press both the **ctrl** + **~** (~ = *tilda*) keys on your keyboard.

We should now have the necessary tools installed in order to write code within our environment and we should be ready to start creating our very first Angular application projects using the Angular CLI tool.

Setup & File Structure

Angular CLI

Below is the webpage and GitHub page for the Angular CLI documentation on the things that you can do using the Angular CLI tool:

https://cli.angular.io/

https://github.com/angular/angular-cli

The first step is to install the Angular CLI tool globally within the terminal. To check that we have node and npm installed on our machines we can use the following commands:

```
$ node -v
$ npm -v
```

To install the Angular CLI tool globally run the following command within the terminal:

```
$ npm install -g @anglular/cli
```

Once installed we can now run the Angular CLI tool commands anywhere within the terminal as this should be now installed globally on your machine. You can run the following command to see if it installed correctly:

```
$ ng version
```

Navigate within the terminal to the folder/directory that you wish to create your Angular App using the Angular CLI tool we installed (alternatively create the folder and open it in terminal to quickly navigate to the folder/directory path). Run the following command replacing the AngularAppName with the name of your Angular App you wish to create:

```
$ ng new AngularAppName
```

The Angular CLI will ask a few questions before installing the various packages and setting up your angular application such as:

- Would you like to add Angular routing? (y/N) — answer y
- Which stylesheet format would you like to use? (use arrow keys) — choice is your's CSS, SCSS or SASS.

After answering these questions the CLI should install all the packages/dependencies required and setup your application files and folders/directories.

If we open this project folder in VS Code we can open up the integrated terminal and run some Angular CLI commands within our project directory such as:

- To view all the commands and flags in the CLI tool:

```
$ ng help
```

- To create the production/build assets of your application:

```
$ ng build
```

- Run end to end (e2e) testing:

```
$ ng e2e
```

- Builds your code on local server with auto-reload to view your application in the browser on localhost:4200:

```
$ ng serve --open
```

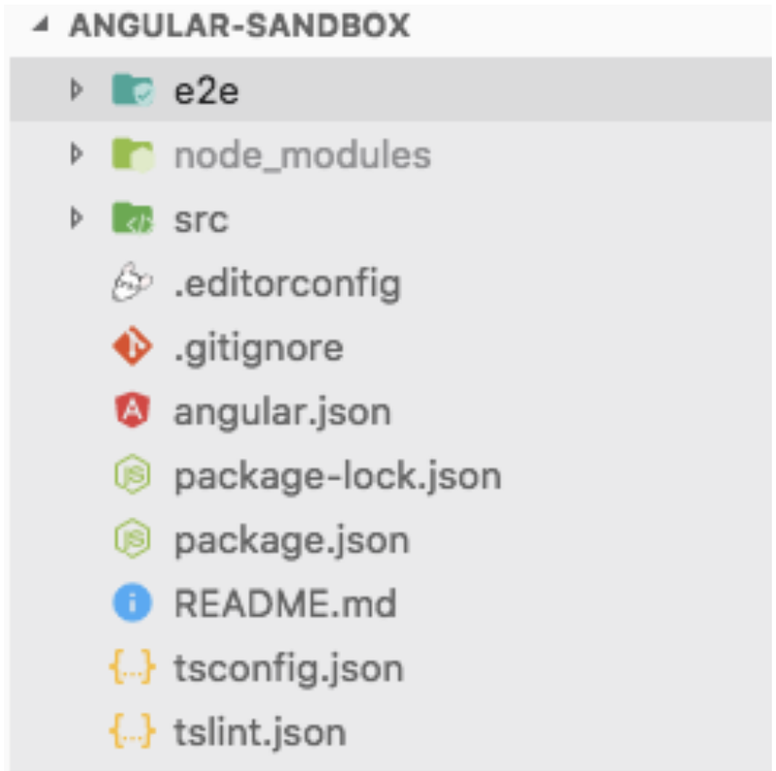
We now have a new boilerplate Angular Application installed on our machine which we can now edit and write our own Angular code to build up our own application.

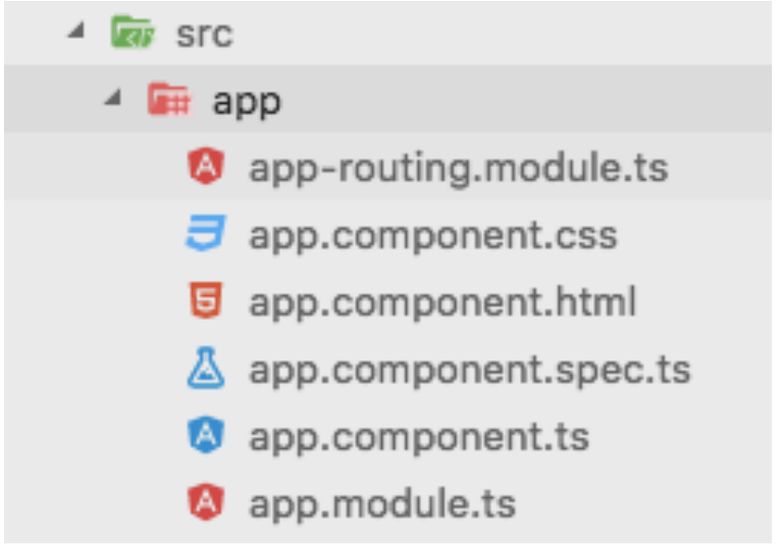
You may also wish to install Augury extension for your chrome browser to view your Angular App component state/ properties within your browser i.e. a dev tool similar to react dev tool but for Angular.
https://chrome.google.com/webstore/detail/augury/elgalmkoelokbchhkhacckoklkejnhcd?hl=en

Setup & File Structure

Angular File Structure

Now that we have created the base application file using the Angular CLI, we are now going to dive into the framework and directory structure of our Angular Application.





- The e2e folder contains the end to end testing files.
- The node_modules folder contains all the node packages/dependencies required by the Angular app.
- Within the src folder there are many subfolders. The app folder is basically your Angular application and will contain all the components, services, models and anything else to do for the application. The src folder also holds other files such as the index.html, styles.css, main.ts etc (*main.ts is the same as main.js but a TypeScript file*).
- Most of the other files are for development, testing and compiling etc. these are things that are not directly related to writing your Angular application. They are more related to your development environment in general.
- The package.json is the most important file which holds the manifest of all your application information and dependencies and npm scripts.
- The tsconfig.json files is the configuration for TypeScript which compiles our TypeScript down to ES5 JavaScript syntax which is supported by all browsers.
- The karma.config.js is the configuration file for testing.
- The .angular-cli.json file is the configuration file for the CLI.

Setup & File Structure

Setting up SSH & Github

SSH stands for Secure Shell and allows a secure way for two machines to communicate with each other. In order to make a secure connection, we would need to setup a SSH keys. In order to set this up, we would need to go into the terminal (*Windows would need to use git bash*) and enter the following commands. We can learn more on SSH Setup with GutHub on the link below:

https://help.github.com/enterprise/2.15/user/articles/connecting-to-github-with-ssh/

To check for existing keys on your machine enter the command:

$ ls -a ~/.ssh

This will check the /users/username/.ssh to find any such file or directory. If the terminal returns “No such file or directory” then this would mean that we do not have any keys on our machine and would need to create some. If we see id_rsa or id_rsa.pub in the terminal then this would mean we already have existing keys.

To create a SSH key we would use the following command:

$ ssh-keygen -t rsa -b 4096 -C “email@email.com”

This will create a private key which we will keep on our machine and a public key that we will give out to third party services such as GitHub. The rsa is the most popular SSH key and the bigger the bits the better the security (4096 is the recommended size for most services). The email is associated with the key pair. Once we run the command it is going to ask for some information such as:

- What we would like to call the file — it is recommended to stick with the default of id_rsa
- Enter passphrase & Confirm passphrase — optional.

Both the private and public keys will be saved within the directory /user/username/.ssh/ and we can start using them in a meaningful way.

If we were to run the previous command to check for existing keys on the machine, it should return the two files created i.e. the id_rsa (private key) and the id_rsa.pub (public key) as the output.

The private key should be kept private and never given out to any third party services. We should treat this file as a password. If someone was to access the private key, this would allow others to steal our machine identity and trick another machine into thinking that they were us.

The next command we would run is going to make sure that when we try to communicate with another service, such as GitHub, it will know which SSH key to use.

$ eval “$(ssh-agent -s)”

This will check whether ssh-agent is running, if it is not running, it will start things up and will display the Agent process id (pid).

The last command is to add the private key to the Agent:

$ ssh-add ~/.ssh/id_rsa

Once the identity has been added, we are now ready to actually take the public key file and provide that to the third party services such as GitHub. The following command will copy the public key to the clipboard (this is specific to the OS you are running - below is for mac):

$ pbcopy < ~/.ssh/id_rsa.pub

For the other operating system view the guide on:

https://help.github.com/enterprise/2.15/user/articles/adding-a-new-ssh-key-to-your-github-account/

Once copied you can go into your GitHub account into settings and within the SSH and GPG keys tab you can add a new SSH Key. You can add a title and the key and you should be able to use SSH with GitHub.

We can run the following command to check if the SSH key was set up correctly with GitHub:
$ ssh -T git@github.com
It will ask a question if we are sure that we want to continue connecting with the server and we can answer yes and let the command run. This will make a very basic ssh connection to the server and this is either going to work if the key was setup correctly or fail is something did not get setup correctly.

You should see the message if correctly setup, returned in the terminal:
Hi GitHubUsername! You've successfully authenticated, but GitHub does not provide shell access.

We can now push our files up to GitHub using a secure SSH connection compared to the standard http requests.