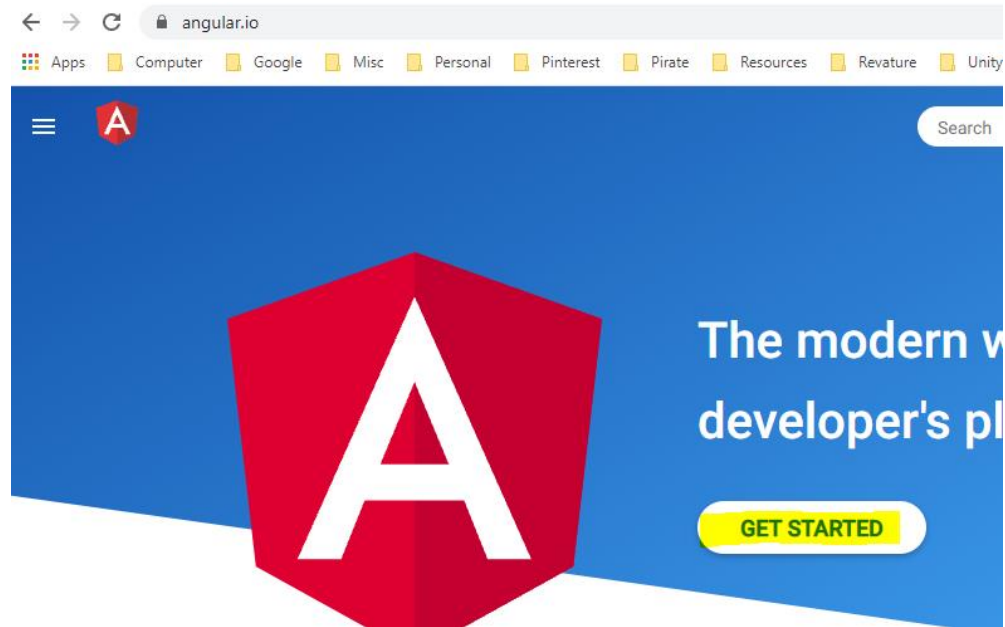


Contents

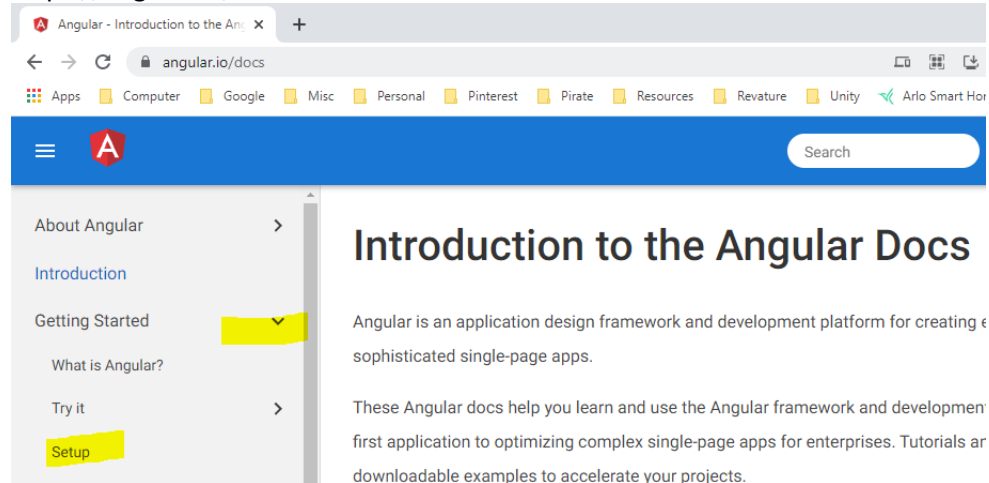
Angular Website.....	2
Install Angular	3
Create a VS Code Project my-app	5
Appendix: Describes Angular files from JwA calendar Aug 23 rd 2021.	19
Installing Angular CLI.....	19
Angular File Structure	19
References	20
Appendix: Aug 24 th 2021 angular-intro.md	21
Appendix: Aug 24th 2021 angular-components.md.....	23

Angular Website

https://angular.io/

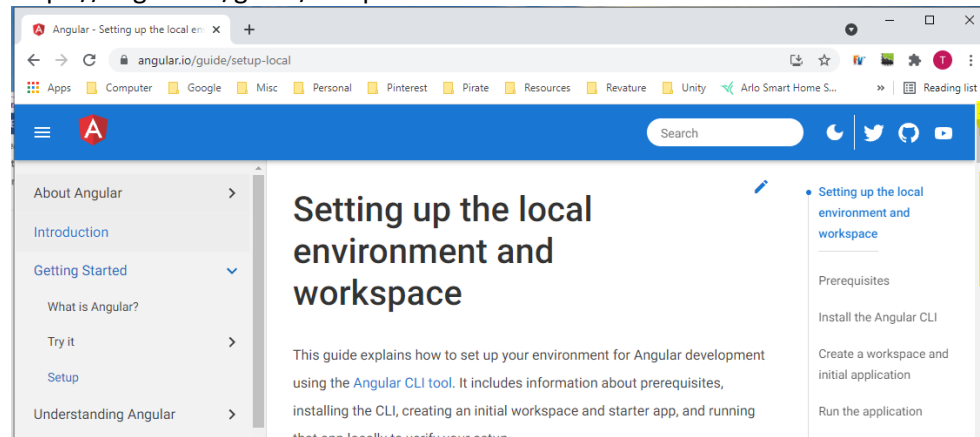


https://angular.io/docs

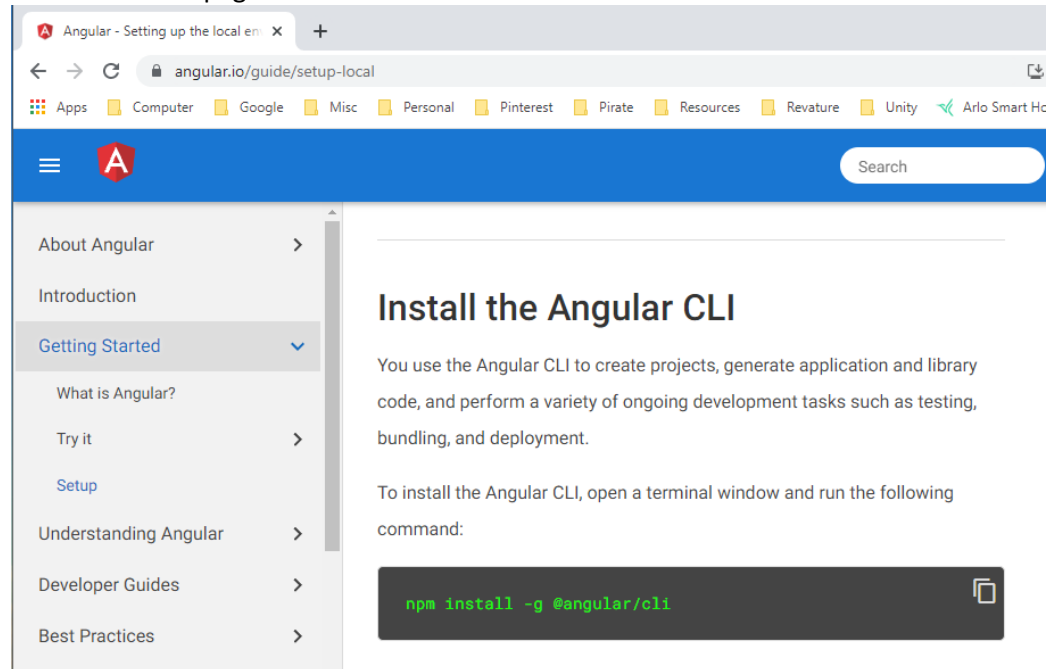


Install Angular

<https://angular.io/guide/setup-local>



Scroll down the page



Open Git Bash

From a previous lecture, we should have installed node.js
(if it is not installed go to <https://nodejs.org/en/> and download for your environment).

Verify node.js is on your system

node -v

MINGW64:/c/Users/tlw8748253

```
tlw8748253@TLW8748253-DL13 MINGW64 ~  
$ node -v  
v14.17.5
```

With node.js installed, it will come with npm (node package manager).

To install angular globally on your machine type the following in Git Bash

```
npm install -g @angular/cli
```

```
npm install -g @angular/cli
```

```
$ npm install -g @angular/cli  
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/  
request/request/issues/3142  
npm WARN deprecated har-validator@5.1.5: this library is no longer supported  
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions  
may use Math.random() in certain circumstances, which is known to be problematic. See  
https://v8.dev/blog/math-random for details.  
C:\Users\tlw8748253\AppData\Roaming\npm\ng -> C:\Users\tlw8748253\AppData\Roaming\npm\n  
ode_modules\@angular\cli\bin\ng  
+ @angular/cli@12.2.10 postinstall C:\Users\tlw8748253\AppData\Roaming\npm\node_modules  
\@angular\cli  
> node ./bin/postinstall/script.js  
+ @angular/cli@12.2.10  
added 1 package from 1 contributor, removed 4 packages and updated 22 packages in 35.01  
s
```

After installation proceed to next step.

Create a VS Code Project my-app

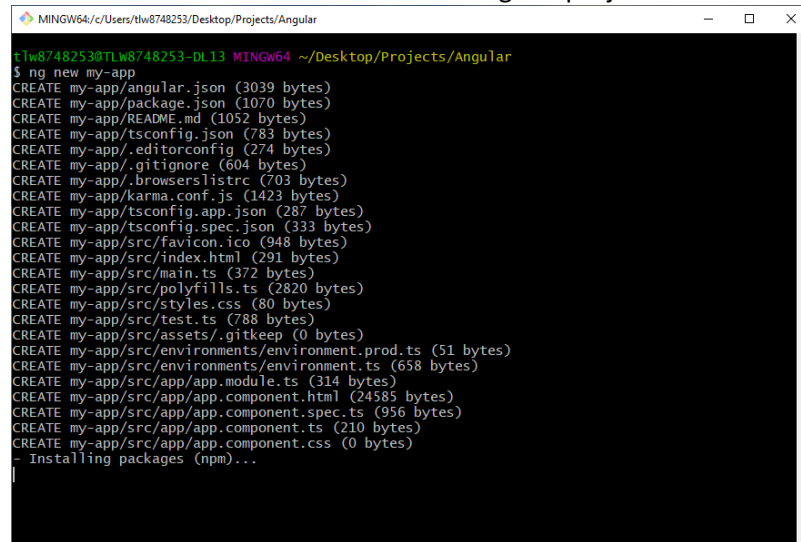
Create a file folder to hold your project, then reopen Git Bash in the folder you created, then:

Create a workspace and initial application

```
ng new my-app
```

Where my-app is the name of your project.

This will create a minimum shell of an Angular project.

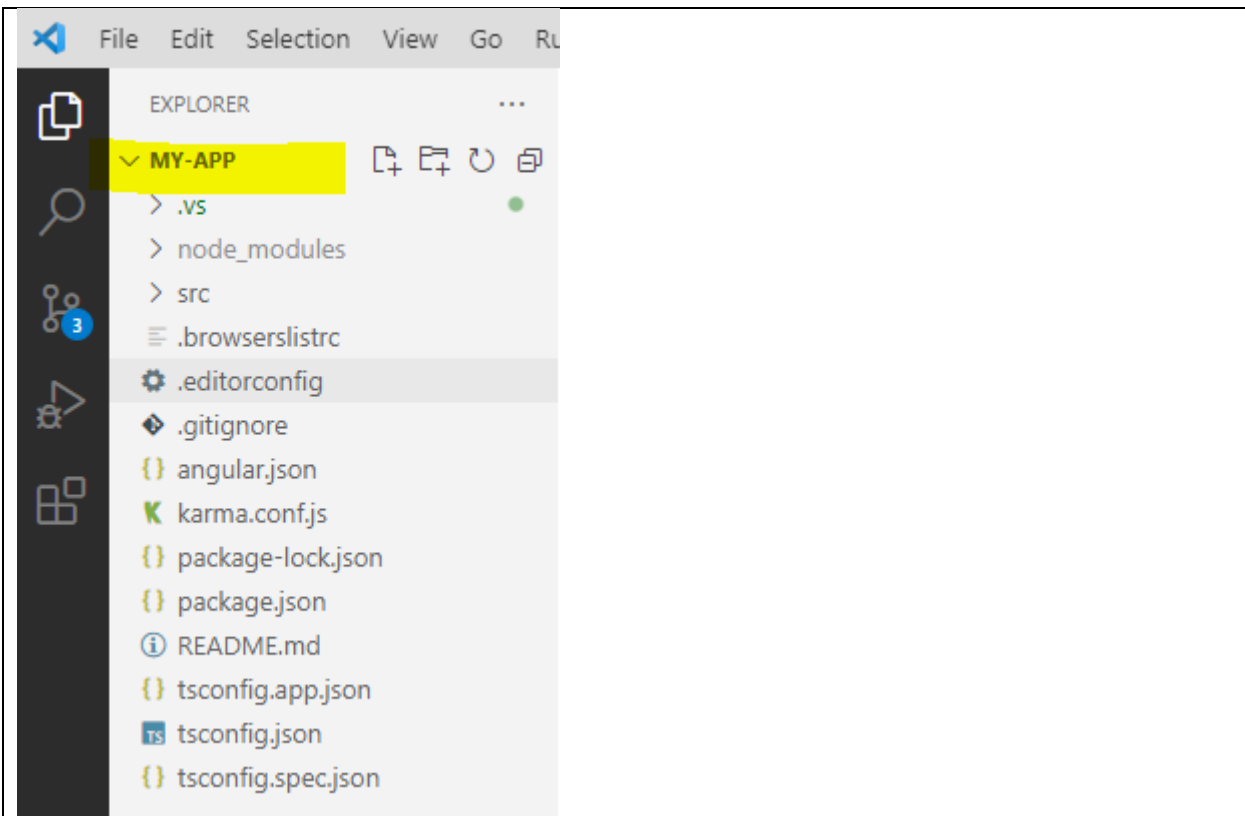


```
MINGW64~/Desktop/Projects/Angular
$ ng new my-app
CREATE my-app/angular.json (3039 bytes)
CREATE my-app/package.json (1070 bytes)
CREATE my-app/README.md (1052 bytes)
CREATE my-app/tsconfig.json (783 bytes)
CREATE my-app/.editorconfig (274 bytes)
CREATE my-app/.gitignore (604 bytes)
CREATE my-app/.browserslistrc (703 bytes)
CREATE my-app/karma.conf.js (1423 bytes)
CREATE my-app/tsconfig.app.json (287 bytes)
CREATE my-app/tsconfig.spec.json (333 bytes)
CREATE my-app/src/favicon.ico (948 bytes)
CREATE my-app/src/index.html (291 bytes)
CREATE my-app/src/main.ts (372 bytes)
CREATE my-app/src/polyfills.ts (2820 bytes)
CREATE my-app/src/styles.css (80 bytes)
CREATE my-app/src/test.ts (788 bytes)
CREATE my-app/src/assets/.gitkeep (0 bytes)
CREATE my-app/src/environments/environment.prod.ts (51 bytes)
CREATE my-app/src/environments/environment.ts (658 bytes)
CREATE my-app/src/app/app.module.ts (314 bytes)
CREATE my-app/src/app/app.component.html (24585 bytes)
CREATE my-app/src/app/app.component.spec.ts (956 bytes)
CREATE my-app/src/app/app.component.ts (210 bytes)
CREATE my-app/src/app/app.component.css (0 bytes)
- Installing packages (npm)...
```

```
warning: LF will be replaced by CRLF in tsconfig.app.json.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in tsconfig.json.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in tsconfig.spec.json.
The file will have its original line endings in your working directory
Successfully initialized git.
```

Wait for the project installation to complete

You can open the project with VS-Code



- **package.json** - used to configure npm package dependencies that are available to all projects in the workspace.

Is similar to the Project Object Model (POM) pom.xml in Maven Java projects.

package.json – [dependencies vs. devDependencies](#).

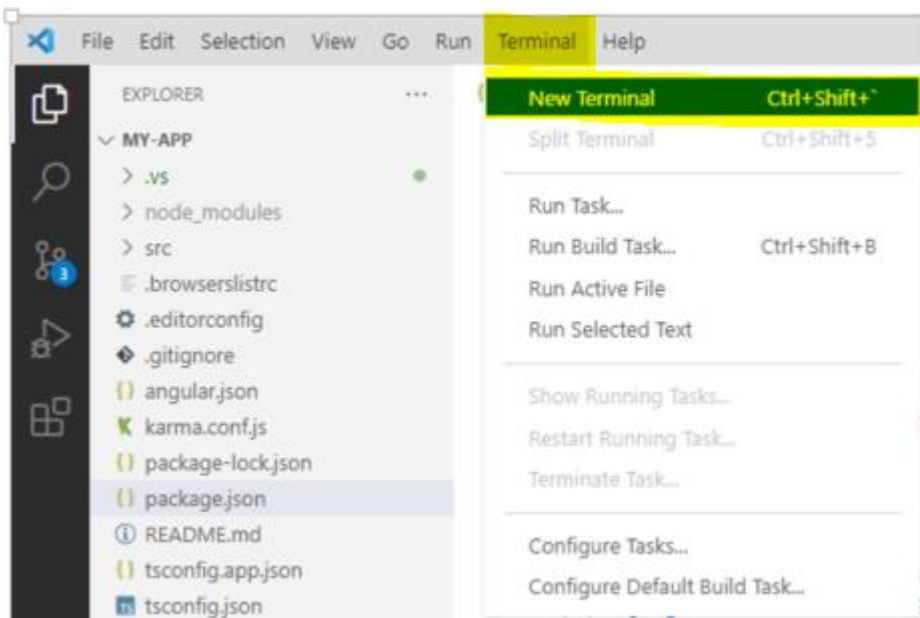
dependencies are what the final website needs where devDependencies are used during development.

Important scripts to start and build the Angular application:

```
"start": "ng serve",  
"build": "ng build",
```

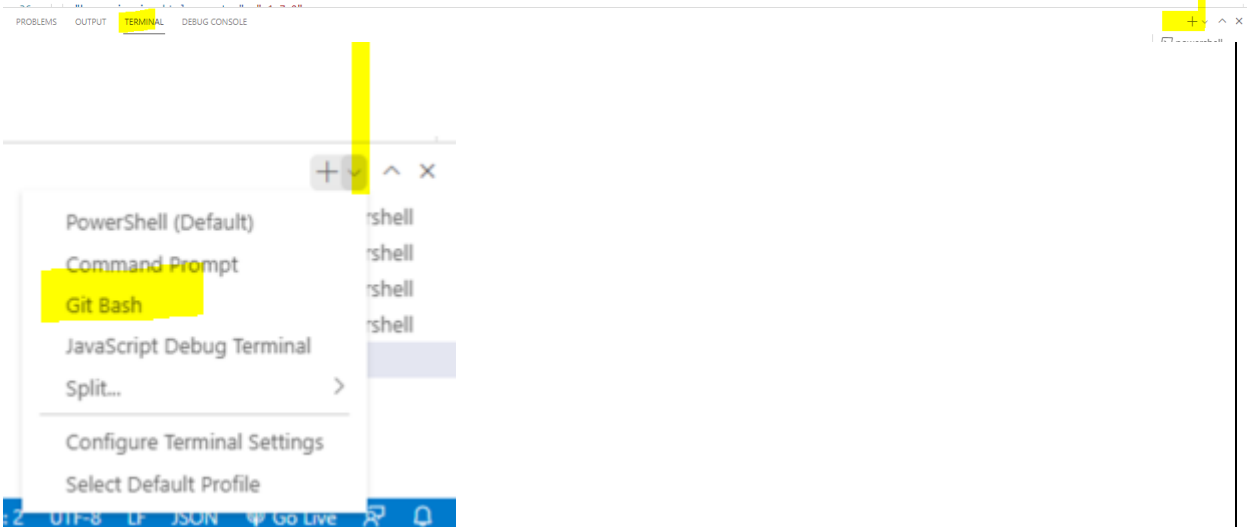
Additional information on files in an Angular project is found:

[Appendix: Describes Angular files from JwA calendar Aug 23rd 2021.](#)



Will open a Git Bash terminal or PowerShell depending on your VSCode setup

You can change the terminal to Git Bash in the terminal window by clicking the down arrow next to the plus sign.



Type

npm run start

Which runs the ng serve command defined in the package.json file.

```
tlw8748253@TLW8748253-DL13 MINGW64 ~/Desktop/Projects/Angular/my-app (master)
$ npm run start
```

```
> my-app@0.0.0 start C:\Users\tlw8748253\Desktop\Projects\Angular\my-app
> ng serve
```

✓ Browser application bundle generation complete.

Initial Chunk Files	Names	Size
vendor.js	vendor	2.09 MB
polyfills.js	polyfills	510.57 kB
styles.css, styles.js	styles	383.36 kB
main.js	main	55.02 kB
runtime.js	runtime	6.61 kB

| Initial Total | 3.02 MB

Build at: 2021-10-19T19:47:14.313Z - Hash: 3cf238b133240949fa8b - Time: 8155ms

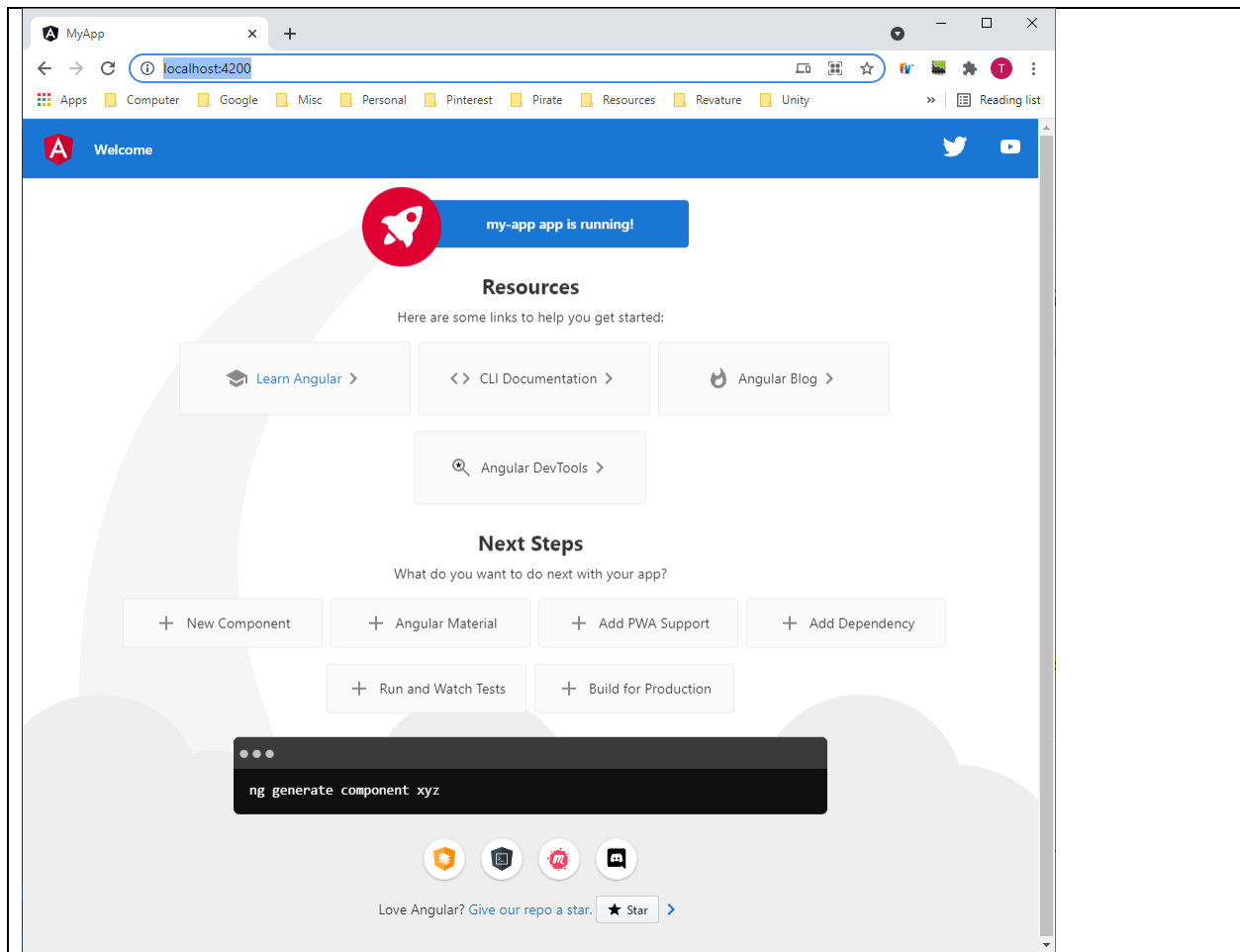
** Angular Live Development Server is listening on localhost:4200, open your browser on <http://localhost:4200/> **

✓ Compiled successfully.

Type in the address bar of a browser:

<http://localhost:4200/>

To open the application shell that was just compiled.



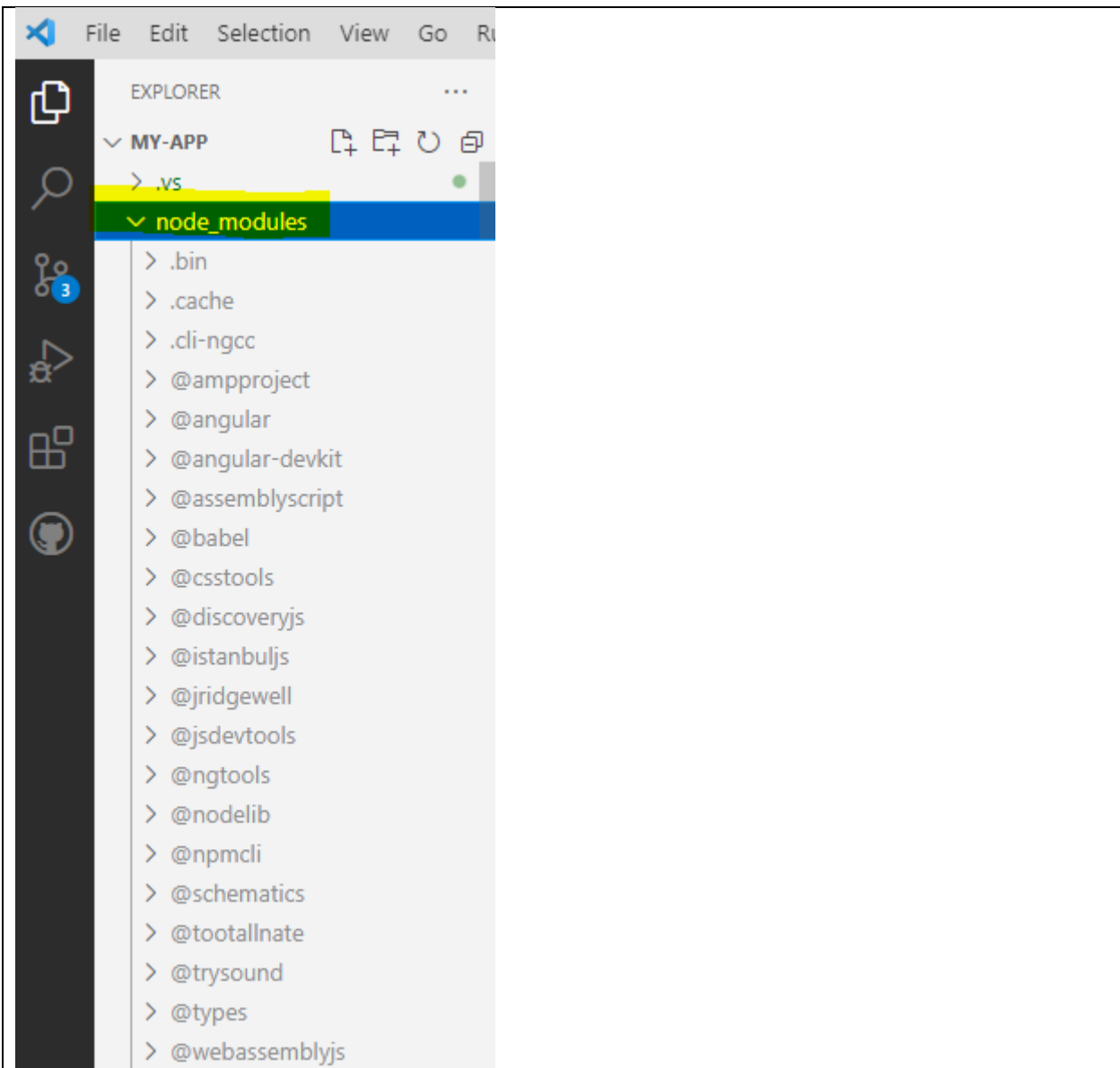
The window above shows your application is working.

Other Angular files:

karma.conf.js – configuration for unit testing similar to JUnit test.

tsconfig.json – type script configuration trans-compiler into JavaScript.

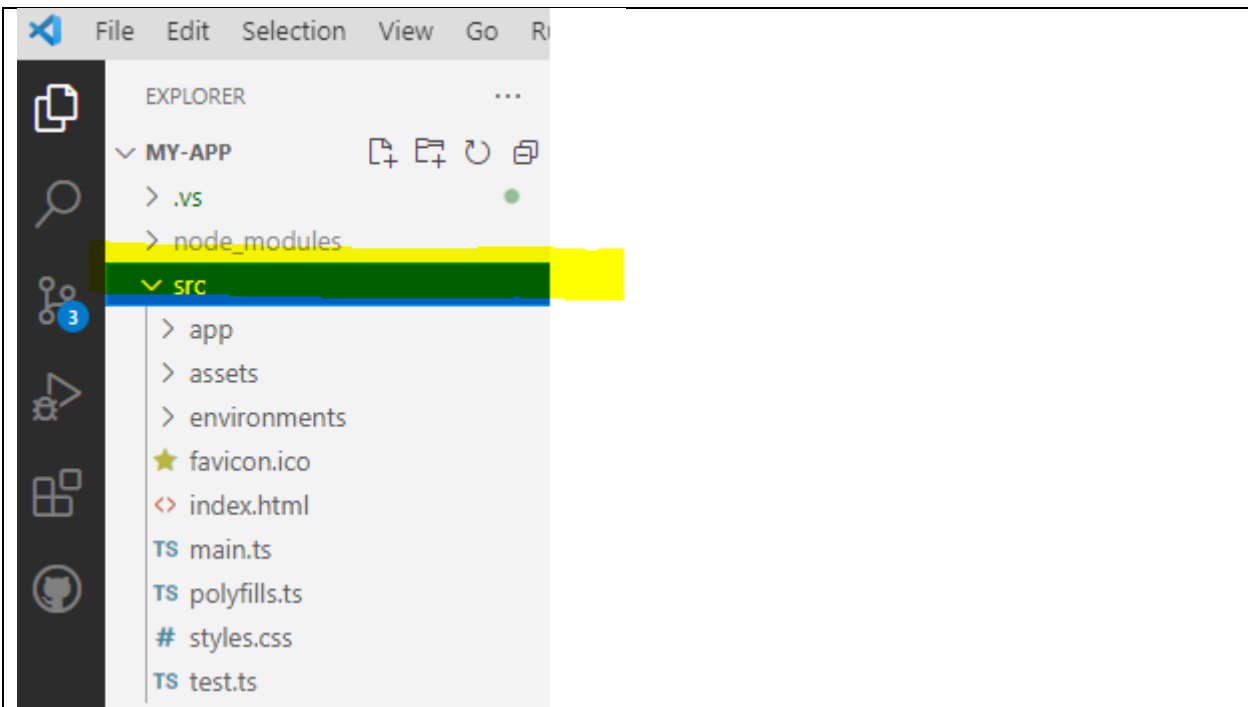
node_modules folder – contains all the dependencies for your project



<https://www.npmjs.com/>

Website to find information about the dependencies in the node_modules folder.
The site is where the dependencies are downloaded from.

Source folder:

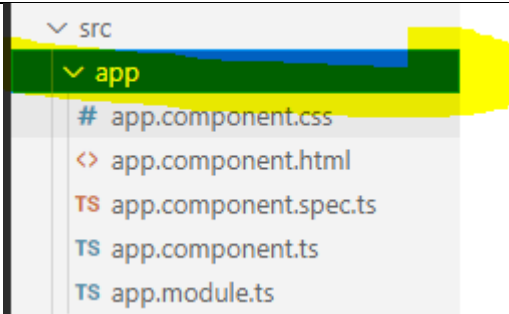


favicon.ico – icon seen on the tab of the browser window.

index.html – initial landing page.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

`<app-root></app-root>` refers to a component, in this case the app folder:



Which contains the html, css, and script for the project page.

Base model is the app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

bootstrap: [AppComponent] - important element inside @NgModule decorator. Startup process scanning for components that make up your program like <app-root> tag what it refers to.

Even though we see multiple .html source files it is still a single page application when package for runtime. Once the application is built it will reside in a dist\my-app folder for distribution to the runtime environment. Where you will see only the index.html and support file.

To build the file use the following command in the terminal window:

```
npm run build
```

```
$ npm run build
```

```
> my-app@0.0.0 build C:\Users\tlw8748253\Desktop\Projects\Angular\my-app
> ng build
```

```
✓ Browser application bundle generation complete.
✓ Copying assets complete.
```

✓ Index html generation complete.

Initial Chunk Files	Names	Size
main.e2ae13337439c05d49e0.js	main	134.15 kB
polyfills.fe861a01c9204748df8e.js	polyfills	36.19 kB
runtime.47c71d9e559014d0f763.js	runtime	1.02 kB
styles.31d6cfe0d16ae931b73c.css	styles	0 bytes

| Initial Total | 171.35 kB

Build at: 2021-10-19T20:39:58.118Z - Hash: 4bdf9965d757cc7f688f - Time: 23212ms

Then you can run the index.html with VSCode using live server.

Recommended to go through the tour of heroes tutorial for more information:

<https://angular.io/tutorial>

Aug 24th 2021 recording

Created angular-intro.md file:

[Appendix: Aug 24th 2021 angular-intro.md](#)

Aug 24th 2021 recording

Continue my-app project

Rebuild the application in Terminal window

npm run build

Creates deployment files in the "app-name"\dist\"app-name" directory:

C:\your-directory-structure\my-app\dist\my-app

Start the application after the new build

npm run start

However you do not have to rebuild the application every time.

You can just run the application with

npm run start

Which starts up a development server where coding changes can be seen in real-time.

Program structure

index.html

```
<body>
  <app-root></app-root>
</body>
```

<app-root></app-root> maps to the app.component.ts file selector: 'app-root':

```
@Component({
```

```
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
```

As such it then uses `templateUrl: './app.component.html'` to render the startup page from `index.html`.

The `templateUrl` file ('./app.component.html') is then imported between the `index.html` tags: `<app-root></app-root>`.

Any changes to the `templateUrl` file and `styleUrls` file will be seen in real-time on the development server as the changes are saved.

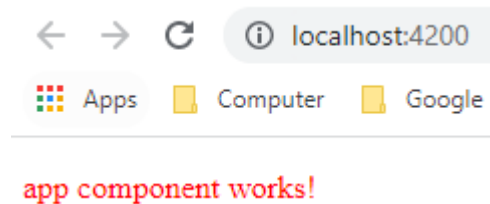
Modify the `app.component.html` file removing all the predefined code and replace with only:

```
<p>app component works!</p>
```

Add the following style to `app.component.css`

```
p {
  color: red;
}
```

Your page should now look like this:



Aug 24th 2021 recording

Create angular-components.md

[Appendix: Aug 24th 2021 angular-components.md](#)

@Component is a decorator similar to annotations in hibernate...

app.component.ts

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

Create new components

Shutdown development server

In VS Code terminal window do a ctrl+c

It could take a little while for it to terminate.

Create subdirectories to organize your components

In the VS Code Git Bash terminal window:

```
cd src/app
ng generate component example
```

The ng generate component command will create a new folder with the component name, in this case example. The command also create the files required for the new component and updates the app.module.ts file.

```
$ ng generate component example
CREATE src/app/example/example.component.html (22 bytes)
CREATE src/app/example/example.component.spec.ts (633 bytes)
CREATE src/app/example/example.component.ts (279 bytes)
CREATE src/app/example/example.component.css (0 bytes)
UPDATE src/app/app.module.ts (400 bytes)
```

app.module.ts

```
@NgModule({
  declarations: [
    AppComponent,
    ExampleComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

Now modify the app.component.html to display our new example component which was created with [**<p>example works!</p>**]:

Need to add the tags <app-example></app-example>. This is nesting of a component.

```
<p>app component works!</p>
<app-example></app-example>
```

Now run the program:

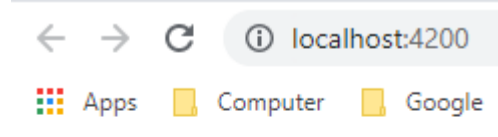
VS Code Git Bash terminal window:

```
npm run start
```

```
...
...
** Angular Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200/ **

√ Compiled successfully.
```

Your page should now look like this:



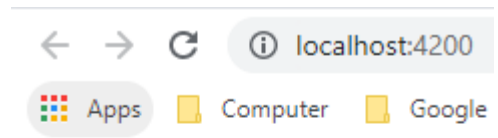
app component works!

example works!

Notice that the parent style in app.component.css will not apply to any child component. Each child component defines its own style in its <name>.component.css.

Modify the app.component.html with additional lines of `<app-example></app-example>` shows example of using a component multiple times:

```
<p>app component works!</p>
<app-example></app-example>
<app-example></app-example>
<app-example></app-example>
```



app component works!

example works!

example works!

example works!

Modify the example.component.html with an additional line will show on the web page the number of times is included.

```
<p>example works!</p>
<p>example added something else.</p>
```


←

→

↻

localhost:4200

Apps Computer Google

app component works!

example works!

example added something else.

example works!

example added something else.

example works!

example added something else.

Open a second VS Code Git Bash terminal to create another component.

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

tlm6748253@TLM6748253-DL13 MINGW64 ~/Desktop/Projects/Angular/my-app (master)
\$ cd src/app
tlm6748253@TLM6748253-DL13 MINGW64 ~/Desktop/Projects/Angular/my-app/src/app (master)
\$

PowerShell (Default)
Command Prompt
Git Bash
JavaScript Debug Terminal
Split...
Configure Terminal Settings
Select Default Profile

cd src/app
ng generate component another-component

\$ ng generate component another-component
CREATE src/app/another-component/another-component.component.html (32 bytes)
CREATE src/app/another-component/another-component.component.spec.ts (697 bytes)
CREATE src/app/another-component/another-component.component.ts (318 bytes)
CREATE src/app/another-component/another-component.component.css (0 bytes)
UPDATE src/app/app.module.ts (524 bytes)

Now add to the new component to example.component.html.

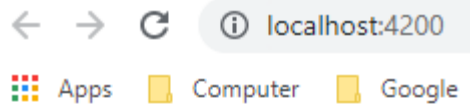
First look at another-component.component.ts to see the selector name:

@Component({
 selector: 'app-another-component',
 templateUrl: './another-component.component.html',
 styleUrls: ['./another-component.component.css']
})

Then add the tag <app-another-component></app-another-component> to example.component.html

<p>example works!</p>

```
<p>example added something else.</p>
<app-another-component></app-another-component>
<app-another-component></app-another-component>
<app-another-component></app-another-component>
```



app component works!

example works!

example added something else.

another-component works!

another-component works!

another-component works!

example works!

example added something else.

another-component works!

another-component works!

another-component works!

example works!

example added something else.

another-component works!

another-component works!

another-component works!

The rest of the Aug 24th 2021 recording is a crash course in TypeScript and is documented in:
TypeScript Aug 24th.docx
TypeScript Aug 24th.pdf

Appendix: Describes Angular files from JwA calendar Aug 23rd 2021.

<https://app.revature.com/curriculum/8105/batch/994/viewCalendar>

Angular CLI

Angular CLI

The [Angular CLI](#) is a command-line interface for Angular that helps us to get started with creating an Angular application. Angular CLI creates an Angular application and uses the [Typescript](#) programming language, [Webpack](#) for Module bundling, Karma for unit testing, and Protractor for end-to-end testing. The Angular CLI takes care of the configuration and initialization of various libraries. It also allows us to add components, directives, services, etc, to already existing Angular applications.

Installing Angular CLI

Before installing Angular CLI, make sure the development environment includes Node.js and an npm package manager.

- Run the command `npm install -g @angular/cli` on the terminal to install the Angular CLI using npm.
- Run the CLI command `ng new my-app` to create a new angular app with the `my-app` name.
- The Angular CLI includes a server so that we can easily build and serve your app locally. First, go to the `my-app` workspace folder and Launch the server by using the CLI command `ng serve --open`.

The `ng serve` command launches the server on HTTP port 4200, which watches our files and rebuilds the app as we make changes to those files. The `--open` (or just `-o`) option automatically opens the browser to <http://localhost:4200>.

After running the `ng server -o` command, we will see:

Angular File Structure

Generally, We use Visual Studio Code or Webstrom as a Code Editor for creating and editing Angular Applications. You can download and install Visual Studio Code from this website: <https://code.visualstudio.com/download>

The file structure of the Angular application described below:

The **e2e** folder at the top level contains source files for a set of end-to-end tests and test-specific configuration files. The **node_modules** folder provides npm packages to the entire workspace. The **src** folder contains the source files which give information about application logic, data, and assets.

- **app** - this folder contains the component files.
 - **app.component.ts** - used to define the logic for the app's root component (AppComponent).
 - **app.component.html** - used to define the HTML template associated with the root AppComponent.
 - **app.component.css** - used to define the base CSS stylesheet for the root AppComponent.
 - **app.component.spec.ts** - used to define the unit test for the root AppComponent.
 - **app.module.ts** - used to define the root module (AppModule) and helps the Angular to assemble the application. All components, including the AppComponent, must be declared inside the AppModule.
- **assets** - this folder contains image and other asset files.
- **environments** - this folder contains build configuration options for particular target environments.
- **favicon.ico** - An icon to used for an application in the bookmark bar.

- **index.html** - The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any `<script>` or `<link>` tags here manually.
- **main.ts** - The main entry point for an application. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser.
- **polyfills.ts** - Provides polyfill scripts for browser support.
- **styles.css** - Lists CSS files that applies the styles for a project.
- **test.ts** - The main entry point for unit tests used in the application.
- **.editorconfig** - this file contains configuration for code editors.
- **.gitignore** - it specifies untracked files that Git should ignore.
- **angular.json** - holds CLI configuration defaults for all projects in the workspace. It includes configuration options for the build, serve, and test tools.
- **browserslist** - used to configure the sharing of target browsers and Node.js versions among various front-end tools.
- **karma.conf.js** - it contains application-specific Karma configuration.
- **package-lock.json** - this provides version information for all packages installed into node_modules by the npm client.
- **package.json** - used to configure npm package dependencies that are available to all projects in the workspace.
- **README.md** - An introductory documentation for the root app.
- **tsconfig.app.json** - it holds application-specific TypeScript configuration, including TypeScript and Angular template compiler options.
- **tsconfig.json** - holds default TypeScript configuration for projects in the workspace.
- **tslint.json** - holds default TSLint configuration for projects in the workspace. TSLint is an extensible static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors.

References

- [Angular Docs - CLI Overview and Command Reference](#)
- [Angular Docs - Workspace and project file structure](#)

Appendix: Aug 24th 2021 angular-intro.md

Angular

Angular is an open-source framework developed by Google, intended to be used to create **single page applications** (SPAs) using TypeScript as the programming language.

TypeScript

TypeScript is a superset of JavaScript, meaning that all JavaScript is valid TypeScript. TypeScript adds additional functionality and features on top of JavaScript.

TypeScript is developed by Microsoft and supports

- Strong typing / static typing (unlike JavaScript, hence the word Type)
- Classes
- Interfaces
- Decorators

TypeScript is **transpiled** into JavaScript

- Transpilation v. Compilation
 - Transpilation: Transform source code from one language into another (with a similar level of abstraction)
 - Compilation: Transform source code into lower level code (binary, bytecode, assembly, etc.)
- TypeScript files are saved with a `.ts` extension instead of `.js`
- JavaScript is what the browser understands, so that is why we need to convert from TS to JS

Single Page Application (SPA)

A single page application is a frontend web application that operates from a single page. Whenever we visit a website that hosts a single page application, all of the associated code (HTML, CSS, JavaScript) are loaded all at once. Whenever we navigate to another "page", elements are added, removed, or changed through **DOM Manipulation**.

Single page application advantages

- Fast and responsive
- Caching capabilities

Single page application disadvantages

- Poor search engine optimization (SEO)
- Data is frontloaded (because all of the JavaScript, HTML, CSS is loaded at once)

Versions (History of Angular)

Angular has many different versions, the latest being Angular 12. Although Angular is open-source, Google has a dedicated team responsible for updating Angular on a regular basis, adding new features or creating bug fixes and enhancements.

It is important to distinguish between AngularJS and what is more commonly known as Angular 2. AngularJS is an entirely different framework from Angular 2+. AngularJS is an entirely different framework from Angular 2 and onwards. In summary, we have

- AngularJS / Angular 1 (October 2010)
 - Completely different framework than the newer version of Angular
 - So if you find tutorials and documentation on AngularJS, DON'T USE THEM
- Angular 2 (September 2016)
 - Newer framework
 - Up Angular 12 (May 2021)
 - This is the modern Angular

Whenever you look up documentation and tutorials on how to accomplish something in Angular, please distinguish between AngularJS and Angular 2+. Whatever tutorial you find for AngularJS WILL NOT work for Angular 2+.

Revisiting Node.js

Remember in week 3 that I said Node.js was a runtime environment for JavaScript, intended for backend applications, such as REST API servers written using JavaScript. Angular, however, is a frontend framework for developing single page applications which run in the browser.

So, why do we need Node.js? Node is required for obtaining and running all of the build and development tools that we need to use, such as

- npm (node package manager): very important in helping us manage dependencies, similar to one of the purposes of Maven
- npm provides us with access to Angular CLI (command line interface), which allows us to easily create new Angular projects, modules, components, pipes, services, etc. Angular CLI also helps us to run tests, to build our project, and also to start up a development server

- This development server runs within the node environment for us to host our application locally while developing it

Node Package Manager

Similar to how we can utilize Maven as a dependency manager, node package manager (npm) allows us to manage dependencies for a JavaScript application. Whenever we generate a new Angular project using `ng new <project name>`, it is also generating a new file within the root directory of the project called the `package.json` file.

The `package.json` file contains metadata to identify our project such as the project description, version, licensing information, etc. It contains a list of the required project dependencies as well. The dependencies are split into two different categories: `dependencies` and `devDependencies`.

- `dependencies` lists all of the dependencies necessary for running the final application
- `devDependencies` lists all of the dependencies that are only necessary during development. An example would be dependencies such as Karma, which is used for running tests

Using Angular CLI

Angular CLI is a command line interface that assists developers in getting started with creating Angular applications. Angular CLI generates a basic Angular project structure containing

- The default app module
- The default app component
- The basic files necessary for "bootstrapping" an Angular application
 - `index.html`
 - The file served whenever someone visits the website
 - When we build the Angular application using the CLI, it automatically adds the `<link>` and `<script>` tags (through Webpack)
 - `Main.ts`: contains the initial logic that starts up our Angular application
- TypeScript transpiler property files
- `package.json`
- etc.

In order to have access to Angular CLI, we need to install it using npm by running `npm install -g @angular/cli`.

Creating a project

Once Angular CLI is installed, we simply run `ng new <project name>` in order to create a new Angular app with the specified project name.

This will create a project that utilizes

- TypeScript for the programming language
- Webpack to bundle all of our code (HTML, CSS, JavaScript) together into a small number of files whenever we build our Application
- Karma for writing and running unit tests
- Protractor for end-to-end testing

Webpack

Webpack is a powerful module bundler that will bundle all of the JavaScript modules and required dependencies together into a single file to be executed by the browser. This is what Angular utilizes behind the scenes when we actually build our project. It will bundle together all of the component HTML files, CSS files, and JavaScript (TypeScript gets converted into JS) into a single HTML file, a single CSS file, and a small number of JS files that are linked to this HTML file through the `<script>` tag.

Appendix: Aug 24th 2021 angular-components.md

Angular Components

Components are the basic building blocks for any Angular application. They provide for a way to segment different parts of the UI (user interface) into separate files for easier organization as well as providing the structure for the single page application. Components also contain the programming logic (TypeScript) associated with a particular component.

Component Principles

- Angular apps are made up of multiple different components
- Components are intended to implement a single feature for the application that is visible on the screen
- Components wrap all of the HTML and TypeScript code to make the "widget" work correctly
- Components can be used multiple times
- Components can be nested inside of other components
- All Angular applications have an `app` component, which should be the most parent component
- Every component has the following files:
 - `app.component.html`: contains the HTML elements that belong to that particular component
 - `app.component.css`: contains the CSS styling for that particular component (very importantly, the styling does not cascade across to the nested components)
 - `app.component.ts`: contains the logic (behavior) of the component
 - `app.component.spec.ts`: contains the unit tests associated with this component

@Component decorator

Inside of the ``<component name>.component.ts`` file, we will find the usage of a decorator called the component decorator.

```
``typescript
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'my-app';
}
...

```

- selector: tells Angular that whenever it encounters the ``<app-root>`` tag, to render a component in place of that tag
- templateUrl: specifies the relative location of the HTML template associated with this component
- styleUrls: specifies all of the CSS files that contain the styling for the HTML template

Component Lifecycle (Lifecycle Hooks)

Whenever components are created and during the time of their operation, they go through various different phases. We have function that are known as `lifecycle hooks` that will execute whenever certain conditions are met. We can utilize this lifecycle hooks to potentially perform useful actions with our components.

These are the following lifecycle hooks to be aware of:

- constructor: Actually instantiates and populates the initial dependencies (through dependency injection, in the case of Angular)
- ngOnChanges(): whenever the input properties of a component change (properties decorated with the @Input() decorator), this method is called. Therefore, this method could be called multiple times during the lifetime of a component
- ngOnInit(): called ONE TIME when the component is first initialized (when it actually populates the DOM with that component)
- ngDoCheck(): called immediately after ngOnChanges() and ngOnInit() so that we can implement our own custom actions for change detection
- ngOnDestroy(): called before Angular destroys a component

