What is Angular CLI

In this course we will learn

1. What is Angular CLI? Why should we use Angular CLI and the benefits it provide

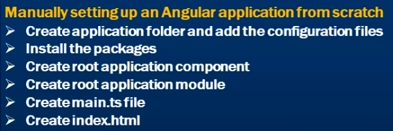
Before watching this course I strongly recomend to watch our Angular 2 tutorial for beginnners course in which we have discussed all the basics of Angular. Once you have a good understanding of the angular basics, it will be easy for you to get the most out of this Angular CLI course.

Here is the link for Angular 2 course

<https://www.youtube.com/watch?v=WWQZC>...

First let's understand why we should use Angular CLI and what problems it solves.

If you have any experience with Angular, then you already know manually setting up an Angular application from scratch is a laborious and time consuming process. We have to



1. Create a separate application folder and add the package definition file ( ie. package.json) and other configuration files. We will discuss what these configuration files are in detail in our upcoming videos in this series.

2. Install the packages using NPM

3. Create the root application component (i.e AppComponent) as every angular application should have atleast one component which is the root component. This root component bootstraps the angular application.

4. Create the root application module (AppModule) as every angular application should have atleast one module which is the root module

5. Create main.ts file which is the entry point to the application. The code in this file, loads the angular root module - AppModule

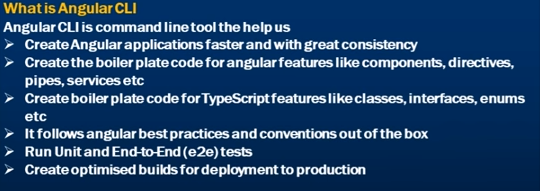
6. Create index.html which hosts our application

You have to manually write all the boilerplate code yourself, which is not only monotonous but also time consuming. If you have an inline view template and inline styles for your component, then it is enough if you just create the TypeScript component class file. But if you have lot of HTML and styles, then for maintainability and separation of concerns, you want your HTML to be in a separate template file and your styles in a separate stylesheet. In this case you will have to manually create the HMTL template and CSS files as well in addition to the component class file.

In a real world application, we will have many components. Just imagine the amount of time we have to spend to create these different component files and the same boilerplate code. In an Angular application, in addition to components, we also have directives, pipes, services etc. Again imagine the amount of time it takes to create that same boilerplate code for all these.

In a real world, we usually have more than one developer working on a given angular project. While all these developers are creating these different files and writing the required boiler plate code, are they following the angular teams best practices and conventions. What if the developers are not following them. How do we enforce them. Well, one way to enforce these is by manual code reviews. Code reviews are not only time consuming but also error prone.

The other option is to have some tooling in place to address this. Angular CLI is such a tool. It help us create angular applications, components, modules, pipes, directives, services and much more with great speed and consistency while still following the angular teams best practices and conventions.



What is Angular CLI

CLI stands for Command Line Interface. So Angular CLI is command line tool the help us

1. Create Angular applications faster and with great consistency

2. Create the boiler plate code for angular features like components, directives, pipes, services etc.

3. Create boiler plate code for TypeScript features like classes, interfaces, enums etc.

4. It follows angular best practices and conventions out of the box

5. Run Unit and End-to-End (e2e) tests

6. Create optimised builds for deployment to production

We will see all these in action in our upcoming videos.

In our next video, we will discuss installing Angular CLI

Text version of the video

<http://csharp-video-tutorials.blogspo>...

Slides

<http://csharp-video-tutorials.blogspo>...

Angular CLI Tutorial

<https://www.youtube.com/watch?v=rJ9o4>...

Angular CLI Text articles & Slides

<http://csharp-video-tutorials.blogspo>...

All Dot Net and SQL Server Tutorials in English

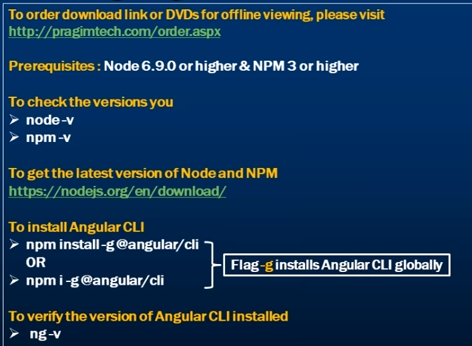
<https://www.youtube.com/user/kudvenka>...

All Dot Net and SQL Server Tutorials in Arabic

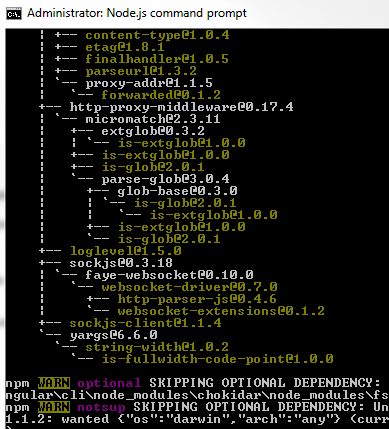
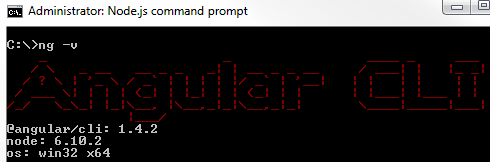
<https://www.youtube.com/c/KudvenkatAr>...

2. Installing Angular CLI

**Suggested Videos**  
[Part 1 - What is Angular CLI](https://www.youtube.com/watch?v=rJ9o4TyhSuo) | [Text](http://csharp-video-tutorials.blogspot.com/2017/09/what-is-angular-cli.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/09/what-is-angular-cli-slides.html)  
  
In this video we will discuss **how to install Angular CLI**

  
  
**Prerequisites for installing Angular CLI :** To install Angular CLI you should have installed Node 6.9.0 or higher, and NPM 3 or higher   
  
  
To check the versions that you have on your machine type the following commands in a command window. 

* node -v
* npm -v

You can get the latest version of Node and NPM from the following website. Click on the correct download link depending on the Operating System you have.  
<https://nodejs.org/en/download/>   
  
Once you have Node and NPM installed. Run Command Prompt as an administrator and execute the following command. Flag -g installs Angular CLI globally on your machine.  
npm install -g @angular/cli   
  
You can also use i as shortcut for install. So the above command can also be rewritten as shown below  
npm i -g @angular/cli   
  
If you see a tree structure as shown below, you have Angular CLI installed successfully.    
   
  
To verify the version of Angular CLI installed, execute the following command  
ng -v   
  
At the time of this recording, I have Angular CLI version 1.4.2 installed on my machine as you can see from the screenshot below.   
   
  
If you run into any problems installing Angular CLI, follow these steps and hopefully Angular CLI will be installed successfully.

  
  
**Step 1 :** Delete "npm" folder from the following path  
C:\Users\Your\_UserName\AppData\Roaming   
  
**Please note :** If you cannot find "AppData" folder, make sure in your windows operating system, you have "Show hidden files, folders, and drives" option is turned on. "AppData" is a hidden folder.   
  
**Step 2 :** Once you have the "npm" folder deleted, uninstall node.js. On a windows machine you can uninstall node.js from Control Panel\All Control Panel Items\Programs and Features. Right click on "Node.js" and select "uninstall" from the context menu.   
  
**Step 3 :** Reinstall Node.js by downloading the appropriate installer for your operating system from the following link.  
<https://nodejs.org/en/download/>   
  
**Step 4 :** Run Command Prompt as an Administrator and try to install Angular CLI again using the following command. Hopefully this time it installs successfully. If not, please leave the problem you are facing as a comment on this video and we will try to help as soon as we can. Also, if you had a problem and you solved it yourself, please let us know what the problem is and how you solved it by leaving a comment, so it could help others with a similar problem. After all, it's all about sharing and learning from each other.  
npm install -g @angular/cli

2. Angular CLI Create new project

**In this video we will discuss**

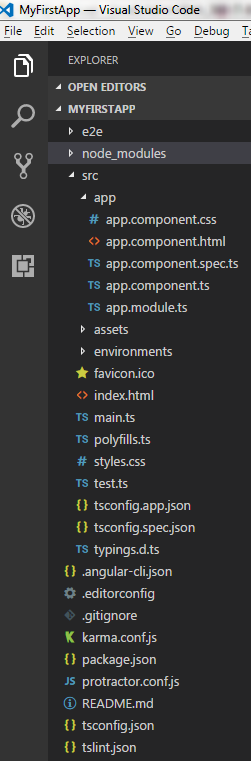
1. How to create a new angular project from scratch using Angular CLI
2. Run the app
3. Run unit and end-to-end test

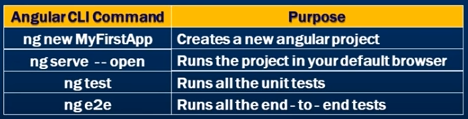
In this course we will use Visual Studio Code as the editor. Visual Studio Code is free and you can use it on any platform - Windows, Mac or Linux. If you have not installed it already, please install it by downloading from the following link.  
<https://code.visualstudio.com/download>   
  
**To create a new Angular Project**, open Command Prompt as an Administrator and execute the following command. This command creates all the required files and also installs all the required packages. Depending on your computer and internet connection speed, this command takes a few minutes to complete.   
  
c:\>ng new MyFirstApp

* ng is the Angular CLI
* new for creating a new application
* MyFirstApp is the name of your angular application

There are several options that we can use with "ng new". We will discuss all these options in our next video.   
  
Once the above command has completed successfully you will see the following messages.  
Installed packages for tooling via npm.  
Project 'MyFirstApp' successfully created.   
  
**So what did this "ng new" command do**

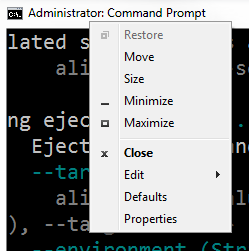
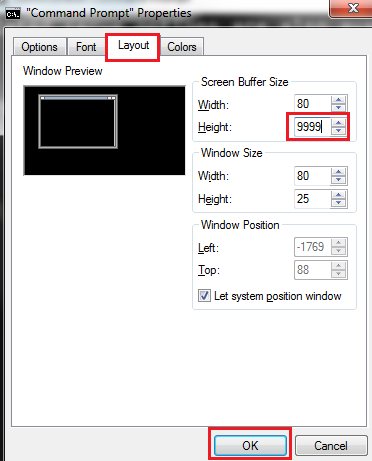
* A new folder with name MyFirstApp is created
* All the required configuration and source files are created.
* All the npm dependencies are installed in node\_modules folder
* Unit and end-to-end tests are created
* The Karma unit test runner is configured
* The Protractor end-to-end test framework is configured

We will discuss unit tests, end-to-end tests, Karma and Protractor in our upcoming videos.  
  
Please note that all these code and configuration files are created by the Angular CLI out of the box while still following the angular teams best practices and conventions.  
  
Now, go to the folder (MyFirstApp) that contains our angular project, by executing the following command. cd stands for change directory  
cd MyFirstApp  
  
Now execute the following command at the command prompt to open the project folder in Visual Studio Code. Notice there is a single space and a DOT after the word code.  
code .  
  
At this point in Visual Studio Code you will see all the Angular project files. Also notice node\_modules folder, that conatins all the installed packages.   
   
  
We will discuss, what all these files are and their purpose in our upcoming videos.  
  
To run the project using Angular CLI, type the following command at the command prompt. This command builds the application and opens it in our default browser. The flag --open, launches our default browser and runs the application. By default the application runs on port 4200. We can change this port number if required. We will discuss how to do that in our upcoming videos.



ng serve --open  
  
At the moment, the angular development server is running in watch mode, meaning when a file changes, those changes are automatically detected, compiled and the browser reloads to reflect the changes. This is called live reload. We can turn this live reload functionality off, if required. Again we will discuss how to do this in our upcoming videos.  
  
To stop the server, press CTRL + C while you are on the command prompt and then "Y" and ENTER key. This will stop the server.  
  
To run all the unit tests, use the following command  
ng test  
  
To run all the end-to-end tests, use the following command  
ng e2e  
  
We will discuss Unit tests, end-to-end tests and all the options we can use to run them using Angular CLI in our upcoming videos.

1. Customize Command Prompt

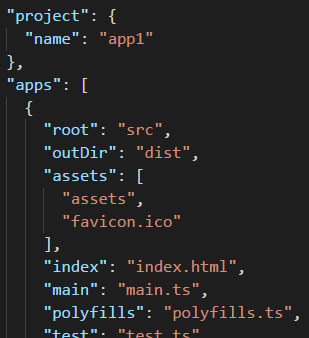
In this video we will discuss **how to customise command prompt window**. You might be wondering why we are talking about customising Command Window in an Angular CLI course. As we know, Angular CLI is a command line tool. We run it using the command prompt window. If we know how to customise the command prompt window to suit our needs, then we have better experience using the Angular CLI.    
  
If you already know, how to customise the command prompt window, you may skip this video.   
  
As far as Angular CLI is concerned, the most useful command I think is, ng --help. When we type this command in the command prompt window and hit enter key, it displays all the Angular CLI commands and the options we can use with these commands.  
  
As we scroll up in the command prompt window, notice we are not able to see all the commands and their associated options. To fix this we have to increase the buffer size. To increase the buffer size there are 2 simple steps  
  
**Step 1 :** Right click on the "title bar" of the command window and select "Properties"from the context menu   
   
  
**Step 2 :** Click on the "Layout" tab and set the "Height" property to 9999 under "Screen Buffer Size" section, and click "OK"   
   
  
With this change in place, execute that same command ng --help. Notice now we can see all the commands and their associated options.  
  
There is lot of help text displayed on the screen. If you want to find a specific command you can use the search feature of the command window. To use the search feature, right click on the "title bar" of the command window and select "Edit" from the context menu. You can then use the "Find" window to search for the command you are looking for.  
  
To redirect the output of ng --help command to the windows clipboard, use the CLIP command as shown below.  
ng --help | clip  
  
Once you have the output copied in the clipboard you can paste it anywhere you want it. For example in a notepad, word document etc.  
  
You can also redirect the output directly to a text document using the following command. This command creates a text document with name MyTextDoc.txt  in the folder where you have executed the command. This text documents will have the output of the command ng --help.  
ng --help >MyTextDoc.txt  
  
Similarly you can also redirect the output to a word document.  
ng --help >MyWordDoc.doc  
  
From the properties window of the command prompt you can also change the cursor size, font size, font colour and background colour.  
  
In our next video we will discuss all the options we can use with the Angular CLI's "ng new" command

1. Angular CLI ng new options

In this video we will discuss **some of the common options that we can use with ng new command**.   
  
The table below shows the common options, their data types, default values, alias and a short description of what they do.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Flag** | **Type** | **Default** | **Alias** | **Purpose** |
| --dry-run | Boolean | false | -d | Run through without making any changes. Just reports the files that will be created |
| --skip-install | Boolean | false | -si | Skip installing packages |
| --skip-tests | Boolean | false | -st | Skip creating tests |
| --inline-style | Boolean | false | -is | Use inline styles when generating the new application |
| --inline-template | Boolean | false | -it | Use inline templates when generating the new project |

1. Angular CLI configuration file

In this video we will discuss the significance of the **Angular CLI configuration file**(.angular-cli.json)   
  
This is the configuration file that the Angular CLI uses. As you can see from the image, it has several settings in it.   
   
  
  
**The settings from this file are used when we** 

* Generate angular features likes components, pipes, services etc
* Run unit and end-to-end tests
* Build the application etc.

We will be revisiting this file many times as we progress through this Angular CLI course.   
  
The table below shows some of the settings and their purpose. We will discuss the other settings and their purpose as we progress through the course.

|  |  |
| --- | --- |
| **Setting** | **Purpose** |
| project : name | Name of the project |
| apps: root | The root directory of the application. Default is src. We can change this using the "source-dir" option when generating a new angular project using the "ng new" command |
| apps: outDir | The output directory for build results. Default is dist |
| apps: assets | List of application assets that you want to copy when building your project. By default, the src/assets/ folder and src/favicon.ico are copied over |
| apps: index | The name of the start HTML file which is index.html by default |
| apps: main | The name of the main entry-point file. main.ts by default |
| apps: polyfills | The name of the polyfills file. Angular is built on the latest standards of the web platform. Targeting such a wide range of browsers is challenging because not all browsers support all features of modern browsers. This can be compensated by using polyfill scripts that implement missing features in JavaScript |
| apps: styles | Global styles to be included in the build. Default is styles.css. We can also use less or scss. To change to less or scss, use the "style" option when generating a new angular project using the "ng new" command |
| apps: prefix | The selector prefix to apply for the generated components. Default is "app". This can be changed by using the "prefix" option when generating a new angular project using the "ng new" command |

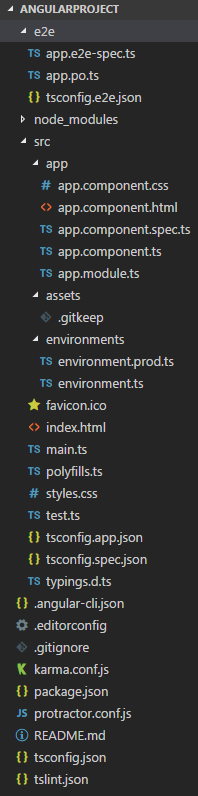
**The important point** to take away is that the values in the Angular CLI configuration file depends on the options that you have used with the "ng new" command when generating a new angular project. For example, if you do not use the --prefix option with the "ng new" command, then the default value "app" is stored in the configuration file for "prefix" setting. So the root component (AppComponent) that is created at the application generation time has "app" as the selector prefix.

  
  
**Instead if you want "pragim" as the prefix, use --prefix flag along with "ng new"command. When you do this several things happen** 

1. "pragim" is stored as the "prefix" setting value in .angular-cli.json configuration file
2. "pragim" is used as the selector prefix for the root component that the "ng new"command generates
3. Any new component that you generate in the future using the following command will also have "pragim" as the selector prefix  
   ng generate component componentName
4. If you want to override the prefix setting in the angular cli configuration file, you can use --prefix option with the generate command as shown below. This will generate the component "xyz" with the prefix "tech" instead of "pragim"ng generate component xyz --prefix tech
5. Some of the options like --prefix can be used with several commands like ng new and ng generate

**Please note :** If you generate a project with --skip-install flag and when you try to generate a new component using "ng new" command before executing the "npm install"command you will get the following error  
node\_modules appears empty you may need to run npm install   
  
To fix this, please first execute "npm install" to install the required npm packages and then generate the component.

7. Angular CLI project structure 1

In this video we will discuss the **Angular project structure**. I have split this into 2 videos. In this video, we will discuss all the files and folders in the Angular project, except the**"src"** folder and it's contents. In our next video we will discuss the **"src"**folder and it's contents.   
  
  
One of the easiest ways to create a working angular project, is by using the Angular CLI. The following Angular CLI command creates a working Angular Project out of the box  
ng new AngularProject   
  
  
The image below shows the Angular project in Visual Studio Code.   
   
  
As you can see there are several files in the project. The table below shows the purpose of each file/folder.

|  |  |
| --- | --- |
| **File / Folder** | **Purpose** |
| package.json | This file contains the packages to build and run our application. It contains two sets of packages, dependencies and devDependencies. The dependencies are essential for running the application. The devDependencies are only required to develop the application. These packages are installed into the node\_modules folder by the Node Package Manager (npm), when npm install commaned is excuted. You can also add your own custom scripts here.   "scripts" property in package.json file contains the useful npm commands. Notice we have "start": "ng serve". This means when we execute npm start it executes ng serve which builds and starts the server. In addition if you also want to launch the browser and open the application  CHANGE "start": "ng serve" TO "start": "ng serve --open" |
| node\_modules | The packages specified in package.json file are installed into this folder when we run npm install command |
| e2e Folder | Contains end-to-end tests and their configuration files. We will discuss end-to-end tests in our upcoming videos |
| .angular-cli.json | This is the Angular CLI configuration file. We discussed the use of this file in our previous video. |
| .editorconfig | Configuration file for Visual Studio Code. The settings in this file let you set certain code style guidelines. For example what indent\_style do you want - spaces or tabs and what should be the indent size etc. You can share this editorconfig file with other developers to maintain consistent coding styles. To read more about editor configuration please visit http://editorconfig.org |
| .gitignore | This file is used to determine files and folders you don't want to check in to source control. For example one of the folders we do not want to check in to source control is /dist folder which is auto generated when we build the application. So this folder is listed in this file. So, all the files and folders listed in this file are ignored, when a change set is checked in to source control. |
| karma.conf.js | Karma is the unit test runner for angular applications. As the name implies, karma.conf.js is the configuration file for Karma. |
| protractor.conf.js | Protractor is an end-to-end test framework for Angular applications. As the name implies, protractor.conf.js is the configuration file for Protractor. |
| README.md | This is a README file which contains the commonly used Angular CLI commands out of the box. You may enhance it with your own project documentation so that anyone checking out the repo knows the commands to use to build, run and test your app. |
| tsconfig.json | This is the TypeScript compiler configuration file. This file has several TypeScript compiler configuration settings. For example, to compile TypeScript to JavaScript on saving a TypeScript file set compileOnSave setting to true. If you do not want .map files to be generated, set sourceMap to false. .map files are used for debugging your application. |
| tslint.json | Angular has a linting tool that checks our TypeScript code for programmatic and stylistic errors as well as non-adherence to coding standards and conventions. tslint.json is the configuration file for linting. We will discuss the settings in this file when we discuss linting in our upcoming videos. |

In our next video we will discuss the "src" folder and it's contents.

8. Angular CLI project structure 2

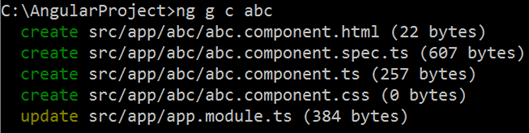
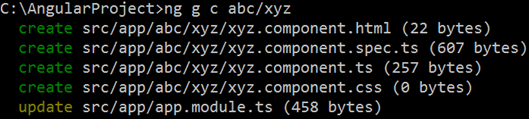
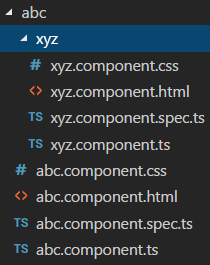
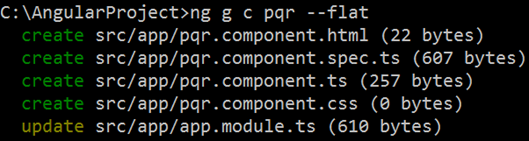
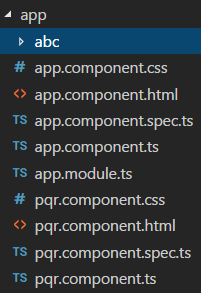
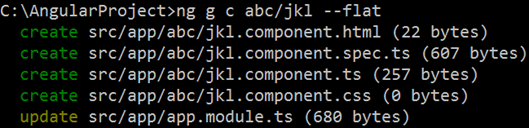
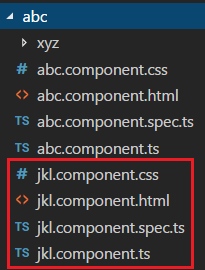
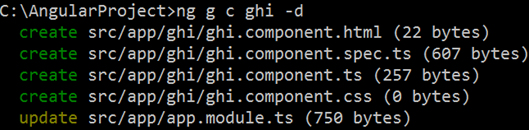
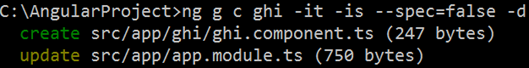
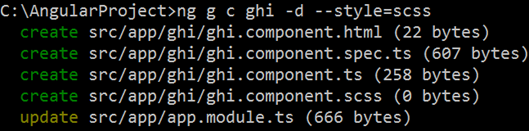
This is continuation to [Part 7](https://www.youtube.com/watch?v=54lvXWA3U-g). In [Part 7](https://www.youtube.com/watch?v=54lvXWA3U-g) we discussed all the files and folders in angular project except the "src" folder and it's contents. In this video we will discuss the "src"folder and it's contents. 

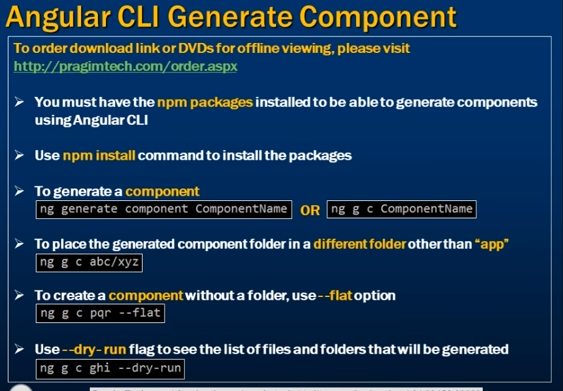
|  |  |
| --- | --- |
| **File / Folder** | **Purpose** |
| src folder | As the name implies, this folder contains all our angular project source code. Components, templates, pipes, services, images, styles etc that our angular application needs are present in this folder. The rest of the files and folders that are present outside this folder, are there to support building our angular application |
| assets | As the name implies, the assets folder contains the assets of your application like images and anything else to be copied when you build your application |
| environments | This folder contains the environment files. By default we have 2 environment files. environment.ts is for for development environment. Notice production property in this file is set to false. environment.prod.ts is for production. Notice in this file production property is set to true as expected. The build system defaults to the dev environment which uses `environment.ts`, but if we do a production build environment.prod.ts will be used. The file and environment mapping is in Angular CLI configuration file (.angular-cli.json) |
| favicon.ico | This is the favorite icon for your application which is typically displayed in the browser address bar and next to the page name in a list of bookmarks. Angular CLI provides this favorite icon out of the box. You may replace this favicon with your own company favicon |
| index.html | The main HTML page that is served when someone visits your site |
| main.ts | The main entry point for the application. This file contains the code to bootstrap the application root module (AppModule) |
| polyfills.ts | This is the polyfills file. Angular is built on the latest standards of the web platform. Targeting such a wide range of browsers is challenging because not all browsers support all features of modern browsers. This can be compensated by using polyfill scripts as they implement the missing features in JavaScript. So these polyfills allow us to use an API regardless of whether it is supported by a browser or not |
| styles.css | This file contains the global styles of our application. Styles that are local and specific to a component are often defined with in the component itself for easier maintenance |
| test.ts | This file is the main entry point for unit tests and loads all the .spec and framework files |
| tsconfig.app.json | TypeScript compiler configuration for the Angular app |
| tsconfig.spec.json | TypeScript compiler configuration for the unit tests |
| typings.d.ts | This is the TypeScript typings file. Many JavaScript libraries, such as jQuery, Angular etc extend the JavaScript environment with features and syntax that the TypeScript compiler doesn't recognize natively. When the typeScript compiler doesn't recognize something, it throws an error. So, we use TypeScript type definition files to tell the compiler about those libraries. These TypeScript type definition files have the extension d.ts. TypeScript editors leverage these type definition files to display type information   Many libraries include type definition files in their npm packages. Angular is one such library. For example, if you look inside node\_modules/@angular/core/ folder in an Angular application, it already contains the type definition files. All the files that have the extenstion d.ts are the type definition files. We will discuss more about these type definition files in our upcoming videos |
| app.component. {ts,html,css,spec.ts} | The root component (AppComponent) TypeScript, HTML template, StyleSheet and Spec files |
| app.module.ts | This is the root application module (AppModule) |

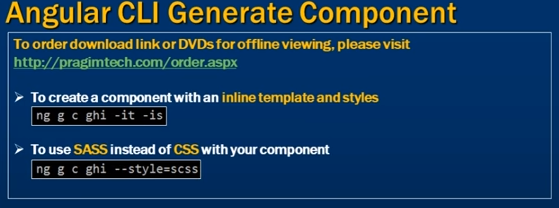
9. Angular cli generate component

In this video we will discuss **generating components using Angular CLI**.   
  
You must have the **npm packages** installed to be able to generate components using Angular CLI. Otherwise when we try to generate components using the ng generate command we will get the following error.  
node\_modules appears empty, you may need to run 'npm install'   
  
  
The following command creates a new Angular project with name "myProject" but it does not install the npm packages as we have used -si flag. The -si flag as we know skips installing the npm packages.  
ng new myProject -si   
  
  
At this point if we try to generate a new component using the following ng generate command, it reports an error - node\_modules appears empty, you may need to run 'npm install'  
ng generate component abc  
  
We will have to first execute npm install command to install the required packages. Once this is done we will be able to generate components.  
  
To generate a component use the following Angular CLI command  
ng generate component ComponentName  
  
OR the shortcut as shown below. In the following command the letter g stands for generate and the letter c stands for component  
ng g c ComponentName  
  
**When we execute this command (ng g c abc) , several things happen**

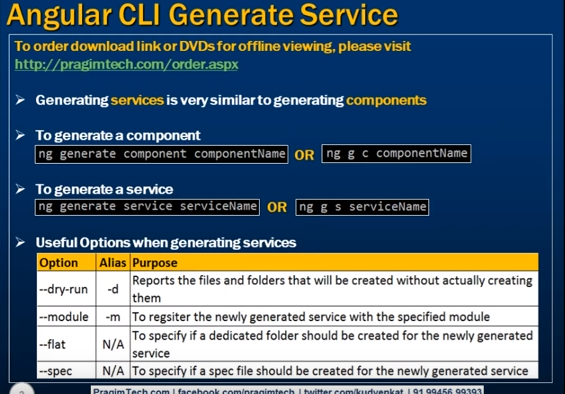
* A folder with name abc is created
* The component files (Component class, View template, CSS file and the spec file ) are created and placed inside the folder "abc"
* The root module file (app.module.ts) is also updated with our new component i.e the required import statement to import the abc component from the component file is included and the component is also declared in the declarations array of the @NgModule() decorator

   
  
**Placing the generated component folder in a different folder :** By default a separate folder is created for every new component that we generate, and the component files (.ts, .css, .html & .spec) are placed in this folder. This newly created folder is placed in the app folder by default. If you want the newly created folder to be placed in a different folder other than the app folder, simply include the folder name in the ng generate command as shown below.   
   
  
Notice now, the newly created "xyz" component folder is placed inside "abc" folder instead of the "app" folder   
   
  
**Generating a new component without a folder :** To create a component without a folder, use --flat option with the ng generate command   
   
  
Notice for the newly generated "pqr" component a separate folder is not created. The component files are placed in the "app" folder.   
   
  
**Placing the flat component files in a different folder other than app :** A flat component is a component that is created with --flat option. This component does not have it's own folder. By default the flat component files are placed in the "app" folder. If you want to place them in a different folder instead, specify the folder name along with the ng generate command.   
   
  
Notice, the newly generated "jkl" component files are placed in "abc" folder instead of the "app" folder.   
   
  
**Using --dry-run flag with component generation :** Just like how we can use the --dry-run flag with "ng new" command, we can also use it with ng generate command. The --dry-run flag reports the files and folders that will be generated, without actually generating them. Once you are happy with what it is going to generate, you can remove the --dry-run flag and execute the command.   
  
For example, the following ng generate command reports that it creates an external template and stylesheet for the component. It also generates a spec file (unit test file). Notice we have run the command with -d flag, so it only reports the files it is going to generate, without actually generating them.   
   
  
If you want an inline template and styles instead of an external template and stylesheet, use -it flag for inline template and -is flag for inline styles. Along the same lines, if you do not want a spec file use --spec=false. Notice we are also using the -d flag.   
   
  
To use sass instead of CSS with your component, use the --style=scss flag with ng generate command. If you want less use --style=less   






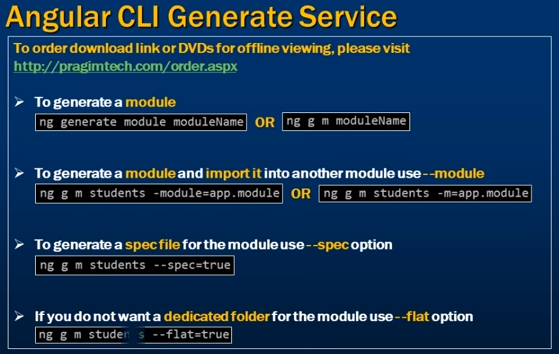
10. Angular cli generate service



**To generate a component we use**  
ng generate component componentName **OR** ng g c componentName   
  
  
**Similarly to generate a service we use**  
ng generate service serviceName **OR** ng g s serviceName   
  
**For example,** to generate a customer service we use the following command.   
ng generate service customer   
  
The above command generates the service and the spec file. What it does not do is register the service. Remember for us to be able to use the service, we must register the service.    
  
We can do it manually after creating the service or we can tell Angular CLI to register our service with a module, using --module option. We can also use it's alias -m   
  
**The following command not only generates employee service**, it also registers our service witht the AppModule  
ng generate service employee -module=app.module   
  
The above command can also be rewritten using aliases  
ng g s employee -m=app.module   
  
**We can also use the --dry-run flag** or it's alias -d to see what Angular CLI generates. Notice in the following command we are using -d option, so Angular CLI simply report the files it is going to generate   
  
ng g s student -d   
  
**The above command generates the service** and the spec file. If you do not want the spec file, simply set --spec=false   
  
ng g s student -d --spec=false   
  
**When generating a component**, Angular CLI by default creates a folder for the component and places all the component files in that folder. A service on the other hand will not have it's own folder. If you want a folder of it's own for a service that the Angular CLI is generating, set --flat option to false as shown below.   
  
ng g s student -d --spec=false --flat=false

11. Angular CLI tutorial for beginners

In this video we will discuss **generating modules using the Angular CLI.**  

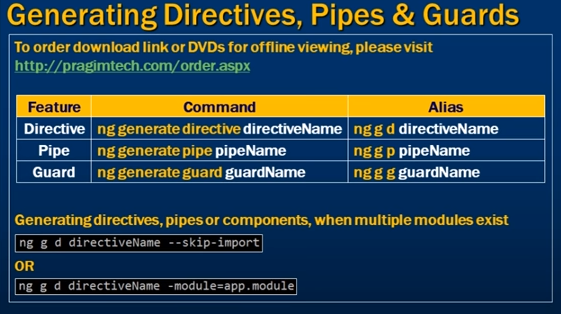


**To generate a module use**  
ng generate module moduleName **OR** ng g m moduleName   
  
  
For example to generate a students module we could use   
**ng generate module students -d OR ng g m students -d**   
  
  
**Please note :** Since we are using the --dry-run flag, the module file and folder is not actually created. We only get the report of the files and folders that will be created.  
  
The above command generates the students module inside students folder. Remember for us to be able to use this newly generated module, we must import it in the root module.  
  
We can do it manually after creating the module or we can tell Angular CLI to import our newly generated module into the root module using --module option. We can also use it's alias -m  
  
The following command not only generates students module, it also imports it into the root module (AppModule)  
**ng g m students -d -m=app.module**  
  
By default a spec file is not generated. If you also want a spec file to be generated use the --spec option  
**ng g m students -d -m=app.module --spec=true**  
  
When generating a module, Angular CLI by default creates a folder for the module and places the module files in that folder. If you do not want a dedicated folder for the module you are generating, use --flat option.  
**ng g m students -d -m=app.module --spec=true --flat=true**  
  
Unitil now, we have been using the --dry-run option. Now let's remove the -d option and execute the command so the module is actually created.  
**ng g m students -m=app.module --spec=true --flat=true**  
  
The above command not only creates the students module, it also imports it into the root module (AppModule). If we look inside app.module.ts file, notice

1. The required import statement to import students module is included
2. The students module is also included in the "imports" array

12. Angular cli generate directives, pipes and routing guards

Generating directives, pipes, routing guards and other angular features is very similar to generating component and services. We discussed generating components and services using the Angular CLI in our previous videos in this course. 





|  |  |  |
| --- | --- | --- |
| **Angular Feature** | **Complete Command** | **Alias** |
| Directive | ng generate directive directiveName | ng g d directiveName |
| Pipe | ng generate pipe pipeName | ng g p pipeName |
| Routing Guard | ng generate guard guardName | ng g g guardName |

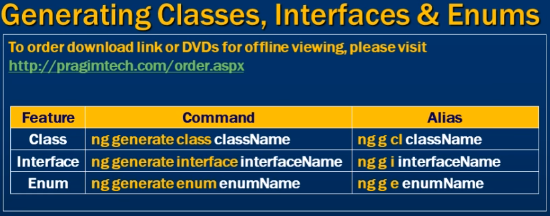
**Please note :** When you try to generate a directive, pipe or a component, and if you have multiple modules in your angular project you may get the following error  
More than one module matches. Use skip-import option to skip importing the component into the closest module.   
  
The reason we are getting this error is we have more than one module in our angular project, so Angular CLI does not know with which module the newly generated directive, pipe or component should be registered. So we have 2 options here.

1. Use --skip-import option to tell Angular CLI not to import and register the generated component, directive or pipe      
   ng g d directiveName --skip-import -d
2. Use --module option or it's alias -m to tell Angular CLI the module with which we want our newly generated component, directive or pipe should be registered.      
   ng g d directiveName -m=app.module -d

If you have just one module in your Angular project, then you wouldn't get this error, as the angular cli will automatically import and register the newly generated component, directive or pipe with that one existing module.  
  
When genearting certain angular features like services or routing guards, you will not get this error, even when you have multiple modules in your project, because by default, Angular CLI does not try to import and register these features.   
  
Please note that we can always use the following options along with ng generate command to customise the generation of directives, pipes and routing guards using the Angular CLI. 

|  |  |
| --- | --- |
| **Option** | **Purpose** |
| flat | Specifies if a dedicated folder should be created |
| module | Specifies the module with which the newly generated angular feature should be registered |
| spec | Specifies if a spec file should be generated |

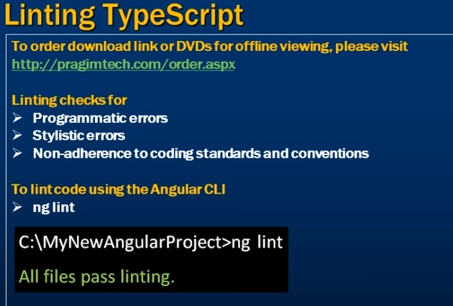
13. Angular cli generate class, interface and enum



So far we have discussed generating angular features like components, services, directives etc. We can use the Angular CLI to generate TypeScript features as well. In this video we will discuss generating TypeScript features like classes, interfaces and enumerations using the Angular CLI.   
  
  
As you have seen throughout this course, Angular CLI provides consistent set of commands for generating features.    
  
  
**To generate a class use**  
ng generate class className or ng g cl className   
  
**For example, to generate an employee class use**  
ng g cl employee   
  
The above command places the employee class directly in the "app" folder. Instead if you want the employee class in a different folder, simply prefix the name of the folder. The command below creates a folder with name "employee" and then creates the "employee" class in it.  
ng g cl employee/employee   
  
By default, a spec file is not created for the class. If you want a spec file to be generated set --spec option to true.  
ng g cl employee/employee --spec=true   
  
**To generate an interface use**  
ng generate interface interfaceName or ng g i interfaceName   
  
**To generate an enum use**  
ng generate enum enumName or ng g e enumName

14. Linting TypeScript

**(ng lint typeof-compare is deprecated. Starting from TypeScript 2.2 the compiler includes this check which makes this rule redundant.)**



In this video we will discuss **Linting TypeScript Code**   
  
Angular has a linting tool that checks our TypeScript code for programmatic and stylistic errors as well as non-adherence to coding standards and conventions. **tslint.json**is the configuration file for linting. This file contains all the default rules for linting our code.   
  
  
For the purpose of this demo I have created a brand new Angular project using the following command.  
ng new AngularProject  
  
Use the following command to lint the code  
ng lint   
  
  
Since we have just generated a new angular project and all the code in the project is auto-generated, we do not have any linting errors and we get the message - All files pass linting.  
  
We also see the following warning  
Warning: The 'no-use-before-declare' rule requires type checking  
  
Basically this warning is saying, if 'no-use-before-declare' rule is enabled we need to use --type-check option with the ng lint command  
ng lint --type-check  
  
'no-use-before-declare' rule is enabled out of the box and it disallows usage of variables before their declaration. To understand what this means, place the following **sayHello()**function in AppComponent class in app.component.ts file. 

sayHello() {

  var message = 'Hello';

  message = message + ' Pragim';

  console.log(message);

}

At this point, execute ng lint command again with --type-check option.    
  
ERROR: C:/AngularProject/src/app/app.component.ts[12, 17]: variable 'message' used before declaration  
ERROR: C:/AngularProject/src/app/app.component.ts[13, 5]: Forbidden 'var' keyword, use 'let' or 'const' instead  
  
Lint errors found in the listed files.  
  
Out of the box, "no-var-keyword" rule is also enabled by default. Turn this rule off by setting it's value to false in tslint.json  
"no-var-keyword": true   
  
Run ng lint command again with --type-check option   
  
Notice, now we only get 1 linting error  
variable 'message' used before declaration  
  
Now modify the code in sayHello() function as shown below. 

sayHello() {

  var message = 'Hello';

  message = message + ' Pragim';

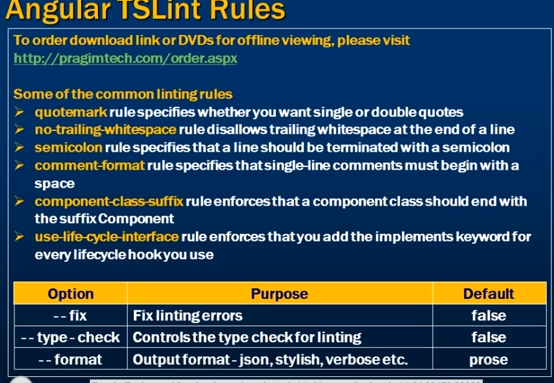
  console.log(message);

}

Run ng lint command again with --type-check option. Notice now we do not get any linting errors.   
  
Variables declared with let keyword are not accessible before they are declared. So this rule 'no-use-before-declare' can be safely disabled, if you have 'no-var-keyword' rule enabled.    
  
When 'no-use-before-declare' rule is disabled and when we run ng lint command without --type-check option, we will no longer get the below warning   
The 'no-use-before-declare' rule requires type checking

15. Angular tslint rules

In this video we will discuss **some of the common angular linting rules in tslint.json file**. You may modify these rules depending on your project requirements.

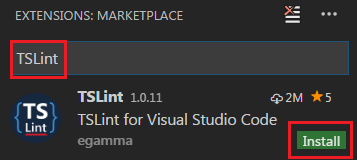
  
  
**Here are some of the common linting rules** 

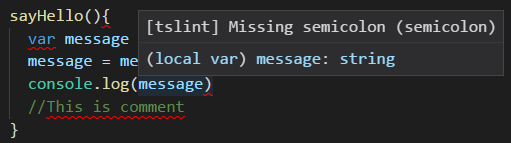
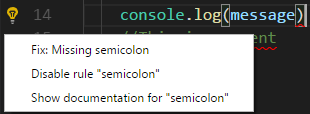
* quotemark rule specifies whether you want single or double quotes
* no-trailing-whitespace rule disallows trailing whitespace at the end of a line
* semicolon rule specifies that a line should be terminated with a semicolon
* comment-format rule specifies that all single-line comments must begin with a space
* component-class-suffix rule enforces that a component class should end with the suffix Component
* use-life-cycle-interface rule enforces that you add the implements keyword for every lifecycle hook you use

Some of the linting errors support automatic fix. To have these linting errors fixed automatically, run ng lint command with the --fix option.  
ng lint --fix  
  
To see the options that can be used with ng lint command, use  
ng lint --help  
  
At the moment, Visual Studio Code is not able to show any linting rule violations. In our next video, we will discuss how to display linting errors in Visual Studio Code so we can fix them as we are writing code.

16. TSLint in Visual Code Studio

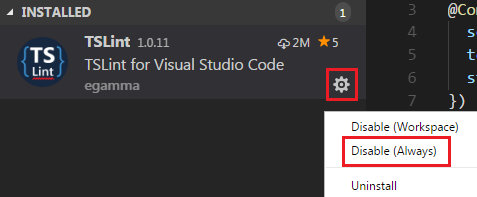
In this video we will discuss, **How to display linting errors in Visual Studio Code**  
  
  
At the moment, our editor Visual Studio Code does not show linting errors. It would be nice if Visual Studio Code can display these linting errors so we can fix them as we are writing code. To achieve this install Visual Studio Code extension - TSLint.   
  
  
**To install this extension** 

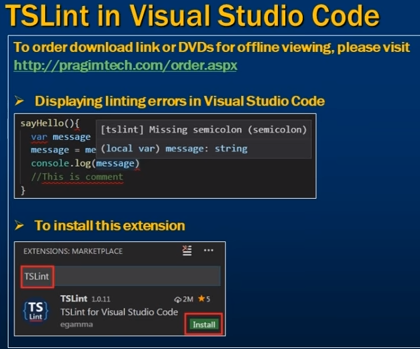
1. Click on the **"View"** menu in **"Visual Studi Code"** and select **"Extensions"** from the context menu
2. In the **"Search Extensions in Marketplace"** textbox type TSLint
3. Click the **"install"** button
4. Once installed, restart Visual Stduio Code to activate TSLint

**At this point**, in Visual Studio Code we will be able to see linting errors and we have the opportunity to fix them as we are developing our application.   
   
  
Once you click on the line where you see a linting error, a light bulb appears on the left margin and when you click on the light bulb you will see 1 to 3 options   
   
  
**You can click on the respective options** 

1. To have the linting errors fixed automatically depending on whether the issue supports automatic fix or not
2. To disable that specific rule
3. To get documentation of the rule

**To disable linting in VS code** 

1. Click on the **"View"** menu in **"Visual Studi Code"** and select **"Extensions"** from the context menu
2. In the **"EXTENSIONS"** window, expand **"INSTALLED"** section
3. Click the **"SETTINGS"** icon against TSLint extension
4. Select **"Disable (Always)"** option
5. Restart Visual Studio Code



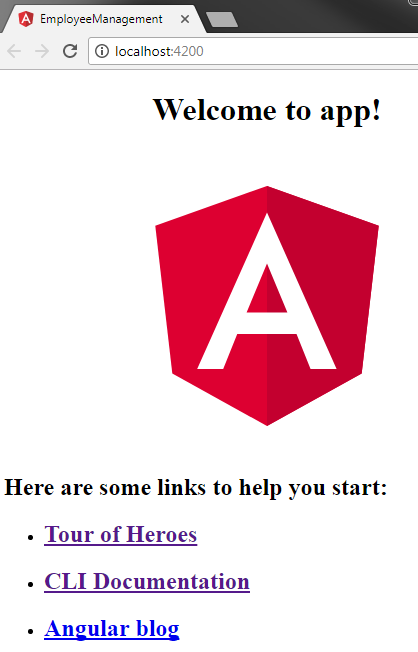
17. Angular Routing

**Implementing routing in an Angular application** involves many small steps. Angular CLI does a pretty good job in having some of these routing steps implemented out of the box by just using --routing option.   
  
  
Before we discuss, **how we can use Angular CLI to implement routing** let's setup routing manually so we understand all the moving parts as far as implementing routing is concerned.   
  
  
Using the following command, first create a brand new Angular project using the Angular CLI.   
ng new employeeManagement   
  
We named it employeeManagement. Let's assume we are using this application to manage employees. Out of the box, Angular CLI has created the root component - **AppComponent**. In addition let's create the components in the table below. The table shows the component name, it's purpose and the routes we are going to use to get to these components.

|  |  |  |
| --- | --- | --- |
| **Component** | **Purpose** | **Route** |
| home | This is the home component | /home |
| employees | This component displays the list of employees | /employees |
| pageNotFound | This component is used when a user tries to navigate to a route that does not exist | /nonExistingRoute |

Let's generate the above components using Angular CLI. Use the following commands to generate the components. Angular CLI not only generates these components, it also imports and registers the components in the root module.

|  |  |
| --- | --- |
| **Component** | **Command** |
| home | ng g c home |
| employees | ng g c employees |
| pageNotFound | ng g c pageNotFound |

At this point execute the following Angular CLI command  
ng serve --open  
  
This builds and launches the application and you will see the following page.   
   
  
**Here are the steps to implement routing in Angular**  
**Step 1 :** Set <base href> in the application host page which is index.html. The <base href> tells the angular router how to compose navigation URLs. This is already done for us by the Angular CLI, when we created this project.

<base href="/">

**Step 2 :** Import the RouterModule into the application root module AppModule. The Router Module contains the Router service and Router directives such as (RouterLink, RouterLinkActive, RouterOutlet etc). So for us to be able to implement routing, we first need to import the Router Module in our AppModule. So in app.module.ts make the following 2 changes

// Import RouterModule

import { RouterModule } from '@angular/router';

// Include RouterModule in the "imports" array of the @NgModule() decorator

@NgModule({

  declarations: [...

  ],

  imports: [

    BrowserModule,

    RouterModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

**Step 3 :** Configure the application routes.   
  
To configure routes, we first need to import Routes type from '@angular/router'. If you look at the definition of Routes type, it is actually an array of Route objects. This Routes type is not required for the application to work. However, using it provides us intellisense and compile time checking. For example, mis-spelled properties of the Route object will be reported as errors.

import { RouterModule, Routes } from '@angular/router';

// Each route maps a URL path to a component

// The 3rd route specifies the route to redirect to if the path

// is empty. In our case we are redirecting to /home

// The 4th route (\*\*) is the wildcard route. This route is used

// if the requested URL doesn't match any other routes already defined

const appRoutes: Routes = [

  { path: 'home', component: HomeComponent },

  { path: 'employees', component: EmployeesComponent },

  { path: '', redirectTo: '/home', pathMatch: 'full' },

  { path: '\*\*', component: PageNotFoundComponent }

];

// To let the router know about the routes configured above,

// pass "appRoutes" constant to forRoot(appRoutes) method

// We also have forChild() method. We will discuss the difference

// and when to use one over the other in our upcoming videos

@NgModule({

declarations: [...

],

imports: [

  BrowserModule,

  RouterModule.forRoot(appRoutes)

],

providers: [],

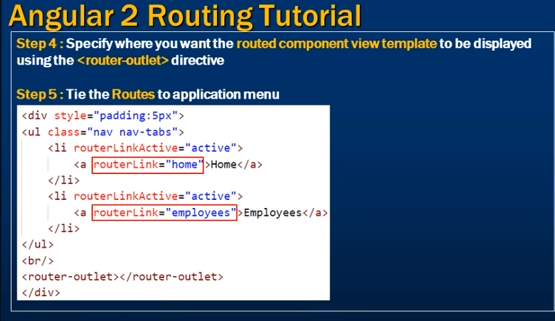
bootstrap: [AppComponent]

})

export class AppModule { }

**Step 4 :** Specify where you want the routed component view template to be displayed using the <router-outlet> directive. In our case we want them to be displayed in the root component AppComponent. So in the root component view template i.e in app.component.html, replace all the existing HTML with <router-outlet></router-outlet>  
  
At this point we should have routing working in our application. Build and run the application using the Angular CLI command : ng serve --open

* Notice the URL in the browser. We have /home in the URL and we see 'home works'! displayed.
* Now in the URL remove '/home', and type '/employees' and press the enter key. Notice the message 'employees works!' is displayed.
* Now in the URL remove '/employees', and type '/abc' and press the enter key. '/abc' is not a valid route so the message 'page-not-found works!' is displayed.
* Now remove '/abc' from the URL and press the enter key. Notice we are redirected to '/home' (the default route) and 'home works'! is displayed.



**Step 5 :** Tie the routes to application menu. Routing is working as expected, but we have to manually type the URL in the address bar. Instead let's include links to our home and employees routes. Let's do this in the Root component(AppComponent). Include the following HTML in app.component.html.  
  
The routerLink directive tells the router where to navigate when the user clicks the link.  
The routerLinkActive directive is used to add the active bootstrap class to the HTML navigation element whose route matches the active route.

<div style="padding:5px">

    <ul class="nav nav-tabs">

        <li routerLinkActive="active">

            <a routerLink="home">Home</a>

        </li>

        <li routerLinkActive="active">

            <a routerLink="employees">Employees</a>

        </li>

    </ul>

    <br/>

    <router-outlet></router-outlet>

</div>

We are using Bootstrap nav component to create the menu. We discussed Bootstrap nav component in [Part 27](https://www.youtube.com/watch?v=vom5dZKCBeM&index=27&list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V) of [Bootstrap tutorial](https://www.youtube.com/watch?v=314m7YBRFvQ&list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V). To install bootstrap execute the following npm command  
npm install bootstrap@3 --save  
  
Once Bootstrap is installed, open .angular-cli.json file and specify the path to the Bootstrap stylesheet in the styles property as shown below.

"styles": [

  "../node\_modules/bootstrap/dist/css/bootstrap.min.css",

  "styles.css"]

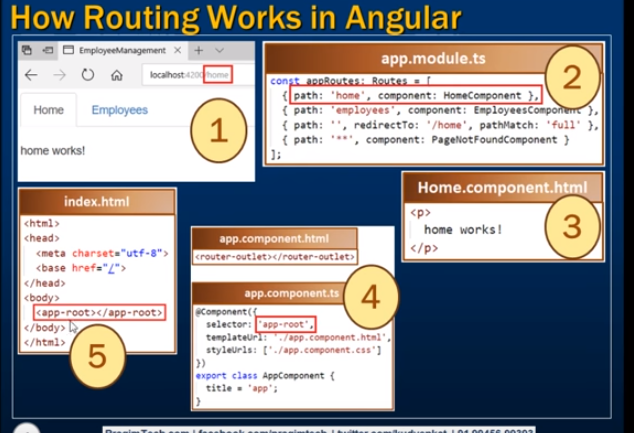
At this point build and run the application using the following Angular CLI command. Routing should be working as expected.  
ng serve --open  
  
**The following are the directives provided by the RouterModule**  
  
**routerLink**  
Tells the router where to navigate when the user clicks the navigation link  
  
**routerLinkActive**

* When a route is active the routerLinkActive directive adds the active CSS class. When a route becomes inactive, the routerLinkActive directive removes the active CSS class.
* The routerLinkActive directive can be applied on the link element itself or it's parent. In this example, for the active route styling to work correctly, routerLinkActive directive must be applied on the <li> element and not the <a> element.

**router-outlet**  
Specifies the location at which the routed component view template should be displayed   
  
At the moment routing is implemented in the root module - **AppModule**. However, for separation of concerns and maintainability, it is better to implement routing in a separate Routing module and then import that routing module in the **AppModule**. In a later video, we will discuss how to move routing into it's own routing module.    
  
As we have seen throughout this video, there are many moving parts that we have to remember, to implement routing correctly in an angular application. In our next video, we will discuss, the routing workflow and how routing actually works in Angular, by connecting all these little moving parts.

18. How routing works in angular

In this video we will discuss, **How routing works in an angular application**. This is continuation to [Part 17](https://www.youtube.com/watch?v=xsothrcm9YE). Please watch [Part 17](https://www.youtube.com/watch?v=xsothrcm9YE) from [Angular CLI tutorial](https://www.youtube.com/watch?v=rJ9o4TyhSuo&list=PL6n9fhu94yhWUcq5Pc16uf8YKXoZ87Vh_) before proceeding. 



As you have seen in our previous video, there are many small steps that you have to remember, to implement routing correctly in an angular application. Let's quickly recap those steps.   
  
**Step 1 :** Set <base href> in index.html.   
  
**Step 2 :** Import the RouterModule into the application root module AppModule.  
  
**Step 3 :**Configure the application routes.   
  
**Step 4 :** Specify where you want the routed component view template to be displayed using the <router-outlet> directive  
  
**Step 5 :** Create a navigation menu and tie the configured routes with the menu using the routerLink directive. Optionally, use the routerLinkActive directive to style the current route that is active, so the user knows the page that he is on, in the application.  
  
Now, let's connect all these small steps and see how routing actually works.  
  
1. We have built the "Home" and "Employees" links using the RouterLink directive. The RouterLink directive  tells the angular router where to navigate when the respective links are clicked. So for example, when we click on the "Home" link, the angular Router includes '/home' in the URL.  
  
2. When the URL changes the angular router looks for the corresponding route in the route configuration. In this case the URL changed to /home, so the router looks for the home route. We have the 'home' route already configured. In the route configuration, we have specified to use the HomeComponent.  
  
3. So the angular router knows to display the HomeComponent view template, but the question is where should the HomeComponent view template be displayed.  
  
4. At this point, the Angular router looks for the <router-outlet> directive. The home component view template is then displayed at the location where we have the <router-outlet> directive. In our case, we placed the <router-outlet> directive in the root component (AppComponent) because that is the top level component where we want our routed component templates to be displayed.  
  
5. We specified 'app-root' as the selector for the root component (AppComponent). This selector (app-root) is used as a directive in the application host page i.e index.html. So along with the navigation menu HTML that we already have in the root component, the HomeComponent view template is also display in index.html page.  
  
6 . Now when we click on the "Employees" link, **Steps 1 to 5** happen in the order specified and the HomeComponent view template is replaced with the EmployeesComponent view template.  
  
Hope you are now able to connect all the dots and have a good understanding of all the small steps of implementing routing in an angular application.  
  
**Please note :**When configuring routes in our previous video, we imported Routes type from '@angular/router'. If you look at the definition of Routes type, it is actually an array of Route objects. This Routes type is not required for the application to work. Even if we remove the Routes type declaration from appRoutes as shown below, the application routing works exactly the same way as before. However, using it provides us compile time checking if we mis-spell the properties of the Route object.   
  
Notice the type declaration **: Routes** is removed from appRoutes constant

const appRoutes = [

  { path: 'home', component: HomeComponent },

  { path: 'employees', component: EmployeesComponent },

  { path: '', redirectTo: '/home', pathMatch: 'full' },

  { path: '\*\*', component: PageNotFoundComponent }

];

At the moment routing is implemented in the root AppModule. However, for separation of concerns and maintainability, it is better to implement routing in a separate Routing module and then import that routing module in the AppModule. In our next video, we will discuss how to move routing into a separate routing module.

19. Implementing routing in separate module in angular

In this video we will discuss, **implementing routing in a separate routing module.**  
  
  
In [Part 17](https://www.youtube.com/watch?v=xsothrcm9YE) of [Angular CLI tutorial](https://www.youtube.com/watch?v=rJ9o4TyhSuo&list=PL6n9fhu94yhWUcq5Pc16uf8YKXoZ87Vh_), we implemented routing. At the moment, all the routing code is implemented in the root AppModule. However, for separation of concerns and maintainability, it is better to implement routing in a separate module and then import that routing module in the AppModule. If routing is in it's own module, it is easier to find and change routing code if required.   
  
  
**Moving routing code into it's own module is easy and straight forward**  
**Step 1 :** Create a new file in the 'app' folder. Name it app-routing.module.ts   
  
**Step 2 :** Copy and paste the following code in it. The code is commented and self-explanatory. 

// Import NgModule decorator to decorate AppRoutingModule class

import { NgModule } from '@angular/core';

// Import RouterModule and Routes type from angular router library

import { RouterModule, Routes } from '@angular/router';

// Import the following 3 components as we will reference

// them in the route definitions below

import { HomeComponent } from './home/home.component';

import { EmployeesComponent } from './employees/employees.component';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

// Configure the routes. The Routes type and the

// referenced components are imported above

const appRoutes: Routes = [

  { path: 'home', component: HomeComponent },

  { path: 'employees', component: EmployeesComponent },

  { path: '', redirectTo: '/home', pathMatch: 'full' },

  { path: '\*\*', component: PageNotFoundComponent }

];

// The NgModule decorator is imported above

// Pass the configured routes to the forRoot() method

// to let the angular router know about our routes

// Export the imported RouterModule so it is available

// to the module that imports this AppRoutingModule

@NgModule({

  imports: [RouterModule.forRoot(appRoutes)],

  exports: [RouterModule],

})

export class AppRoutingModule { }

**Step 3 :** Modify the code in the root AppModule in app.module.ts. The code is commented and self-explanatory. 

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

import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';

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

import { HomeComponent } from './home/home.component';

import { EmployeesComponent } from './employees/employees.component';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

@NgModule({

  declarations: [

    AppComponent,

    HomeComponent,

    EmployeesComponent,

    PageNotFoundComponent

  ],

  // Import AppRoutingModule which contains our routing code

  // AppRoutingModule has also exported angular RouterModule, so

  // all the RouterModule features are also availble to this module

  // including the <router-outlet> directive used in the AppComponent

  // If AppRoutingModule module did not export RouterModule we get

  // 'router-outlet' is not a known element error

  imports: [BrowserModule, AppRoutingModule],

  providers: [],

  bootstrap: [AppComponent]

})

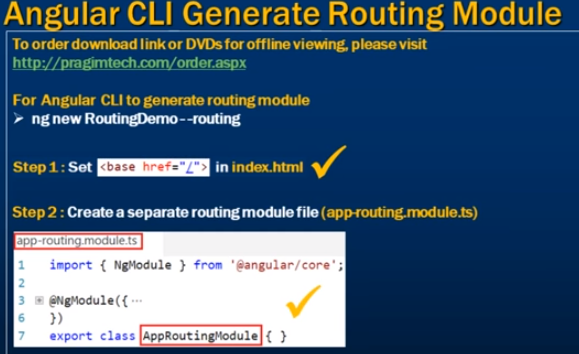
export class AppModule { }

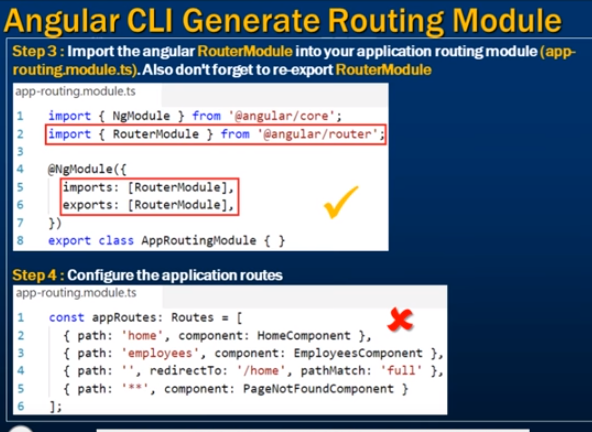
Save all changes and run the project using the following command  
ng serve --open   
  
Notice routing in our angular application works exactly the same way as before. We now have routing implemented in it's own module.   
  
To quickly recap, here are the steps to implement routing in a separate module.   
  
**Step 1 :** Set <base href> in index.html.   
  
**Step 2 :** Create a separate routing module file. You can name it anything you want. I named it app-routing.module.ts.  
  
**Step 3 :** Import the angular RouterModule into your application routing module (app-routing.module.ts). Also don't forget to re-export RouterModule.  
  
**Step 4 :** Configure the application routes.   
  
**Step 5 :** Import the application routing module (app-routing.module.ts) in the root AppModule.  
  
**Step 6 :** Specify where you want the routed component view template to be displayed using the <router-outlet> directive  
  
**Step 7 :** Create a navigation menu and tie the configured routes with the menu using the routerLink directive. Optionally, use the routerLinkActive directive to style the current route that is active, so the user knows the page that he is on, in the application.  
  
In our next video, we will discuss, how Angular CLI generates most of this routing code out of the box.

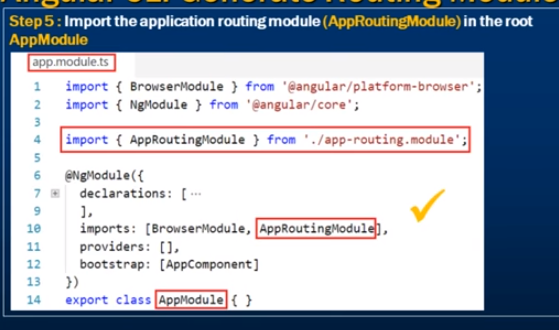
20. Angular CLI generate routing module

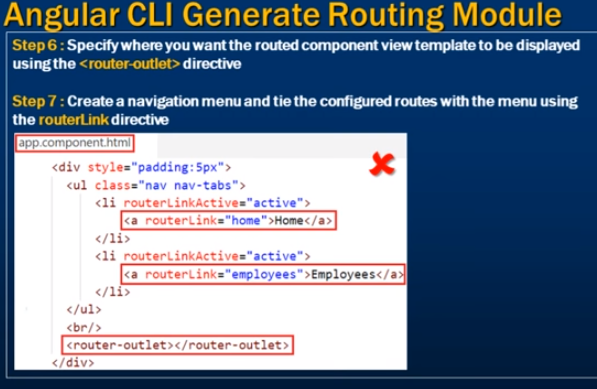
To make Angular CLI generate a routing module, all you have to do is use --routing option along with the ng new command when generating a new Angular project.  
ng new RoutingDemo --routing   
  
In our previous video, we discussed the steps to implement routing in a separate module, and them import that routing module in the application root module AppModule. Here are those steps. 

**Step 1 :** Set <base href> in index.html.   
  
**Step 2 :** Create a separate routing module file. You can name it anything you want. I named it app-routing.module.ts.   
  
**Step 3 :**Import the angular RouterModule into your application routing module (app-routing.module.ts). Also don't forget to re-export RouterModule.   
  
**Step 4 :**Configure the application routes.   
  
**Step 5 :** Import the application routing module (app-routing.module.ts) in the root AppModule.   
  
**Step 6 :** Specify where you want the routed component view template to be displayed using the <router-outlet> directive   
  
**Step 7 :** Create a navigation menu and tie the configured routes with the menu using the routerLink directive. Optionally, use the routerLinkActive directive to style the current route that is active, so the user knows the page that he is on, in the application.









Out of the above 7 steps, we only need to implement steps 4 & 7. The rest of the steps are implemented by the Angular CLI out of the box. Just imagine the amount of time we save.   
  
Before we can implement steps 4 & 7. Let's generate the following 3 components.

|  |  |
| --- | --- |
| **Component** | **Angular CLI Command** |
| HomeComponent | ng g c home |
| EmployeesComponent | ng g c employees |
| PageNotFoundComponent | ng g c pageNotFound |

Now let's implement Step 4. In app-routing.module.ts file specify the application routes. Copy and paste the following code. In addition to the routes, notice we are also importing HomeComponent, EmployeesComponent & PageNotFoundComponent as we are referencing these components in the route configuration. 

import { HomeComponent } from './home/home.component';

import { EmployeesComponent } from './employees/employees.component';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

const routes: Routes = [

  { path: 'home', component: HomeComponent },

  { path: 'employees', component: EmployeesComponent },

  { path: '', redirectTo: '/home', pathMatch: 'full' },

  { path: '\*\*', component: PageNotFoundComponent }

];

Now let's implement Step 4. In app.component.html copy and paste the following code.

<div style="padding:5px">

    <ul class="nav nav-tabs">

        <li routerLinkActive="active">

            <a routerLink="home">Home</a>

        </li>

        <li routerLinkActive="active">

            <a routerLink="employees">Employees</a>

        </li>

    </ul>

    <br/>

    <router-outlet></router-outlet>

</div>

Finally we need to install and reference Bootstrap, to style the navigation menu. To install bootstrap execute the following npm command. We can execute this command in the command prompt window or in the integrated terminal window in Visual Studio Code.  
npm install bootstrap@3 --save   
  
Once Bootstrap is installed, open .angular-cli.json file and specify the path to the Bootstrap stylesheet in the styles property as shown below.

"styles": [

  "../node\_modules/bootstrap/dist/css/bootstrap.min.css

",

  "styles.css"

]

At this point stop the server. Build and run the application again using the following Angular CLI command. Routing should be working as expected.  
ng serve --open

21. Running angular app locally

**In this video we will discuss** 

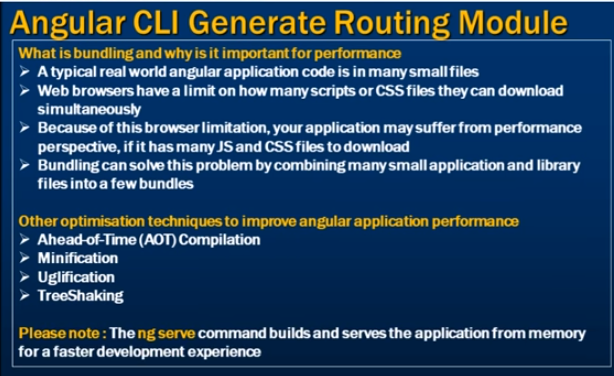
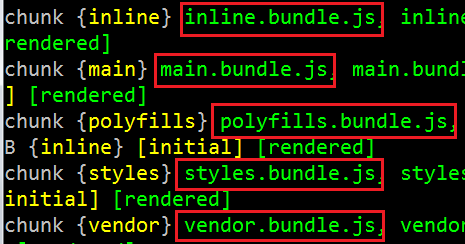
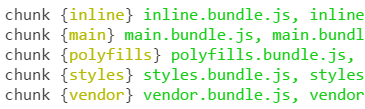
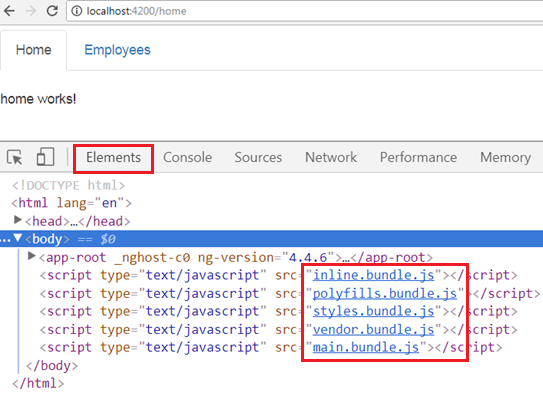
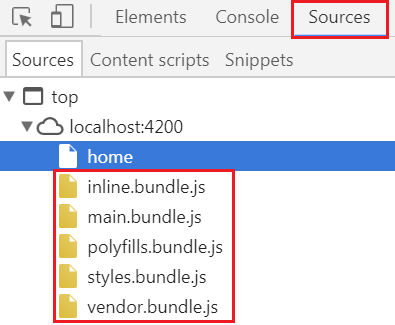
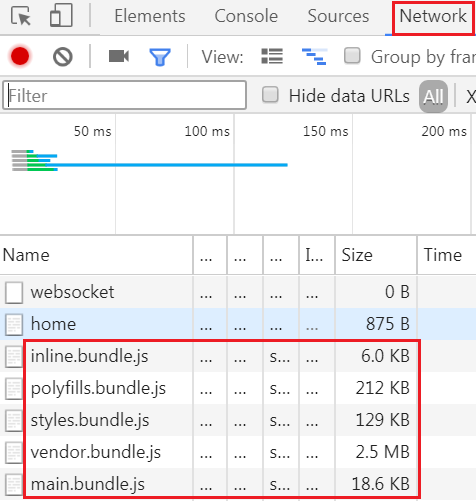
* How to compile and run an angular application locally on your development machine
* What happens behind the scenes when we compile and run an angular application
* What is bundling and why is it important for performance

So far in this video series we have been using the following command to build and run our angular application.  
ng serve --open  
  
  
Have you ever thought about what happens behind the scenes when we execute this command. Behind the scenes, Angular CLI runs Webpack to build and bundle all JavaScript and CSS code. The following are the bundles.

|  |  |
| --- | --- |
| **Bundle File** | **What it contains** |
| inline.bundle.js | WebPack runtime. Required for WebPack to do it's job |
| main.bundle.js | All our application code that we write |
| polyfills.bundle.js | Browser Polyfills |
| styles.bundle.js | Styles used by the application |
| vendor.bundle.js | Angular and 3rd party vendor files |



**What is bundling and why is it important for performance**  
A typical real world angular application is made up of many components. Each component code is in it's own .ts file which gets transpiled to JavaScript i.e to a .js file. Along the same lines, a component may also have it's own .css file for styles. So our angular application code is in many small files. In addition to our application code files, we also have vendor code files like Angular, jQuery etc.    
  
Web browsers have a limit on how many scripts or CSS files they can download simultaneously.   
  
Because of this browser limitation, your application may suffer from performance perspective, if it has many JavaScript and CSS files to download.

  
  
Bundling can solve this problem by combining many small application and library files into a few bundles. As mentioned before, Angular CLI runs WebPack for building and bundling angular applications.  
  
**There are several ways to see these generated bundles.**  
1. If you have executed the "ng serve --open" command in a command prompt window, upon build completion you can see the generated bundles in the command prompt window as shown in the image below.   
   
  
2. If you have executed the "ng serve --open" command in Visual Studio Code Integrated Terminal, upon build completion you can see the generated bundles in the integrated terminal window as shown in the image below.   
   
  
3. "ng serve --open" command builds and runs the application. By default the application runs at port number 4200. You can change this default port number if you want to. We will discuss how to do that in our upcoming videos. When the application is served in the browser you can see the generated bundles on the "Elements" tab in Browser Developer Tools.   
   
  
4. You can also see these bundles on the "Sources" tab in Browser Developer Tools.   
   
  
5. To see the bundles along with their sizes click on the Network tab. If you don't see the bundles, refresh the browser window by pressing F5.   
   
  
In addition to bundling, we can also use other optimisation techniques like Ahead-of-Time (AOT) Compilation, Minification, Uglification and TreeShaking to improve performance. We will discuss all these techniques and how to implement them in our upcoming videos.  
  
The ng serve command builds and serves the application from memory for a faster development experience. It does not write the build artefacts to the disk, so we cannot use this command if you want to deploy the build to another server. For example, if you want to deploy your angular application to a test server for testing, or to your production server we cannot use ng serve. We instead use ng build. This command writes the build artefacts to the specified output folder, so the application can be deployed elsewhere. We will discuss ng build in our upcoming videos.  
  
To customise the in-memory builds that the "ng serve" command produces, there are several options that we can use along with this command. We will discuss these options in our next video.

22. Angular CLI ng serve options

In this video we will discuss some of the common options that we can use with ng servecommand.   
  
To see the list of all options that we can use with "ng serve" command use --help option  
ng serve --help   
  
The following page also shows all the options that can be used with ng serve  
<https://github.com/angular/angular-cli/wiki/serve>   
  
The following command, builds and launches the application in your default browser.  
 ng serve --open  
  
Many of our channel subscribers have sent me emails saying their application is using Internet Explorer, but they want to use Google chrome instead. So thier question is how to change my default browser. Well that's simple and it really depends on the operating system you have. For example, on a Windows 7 operatin system here are the steps to change your default browser. 

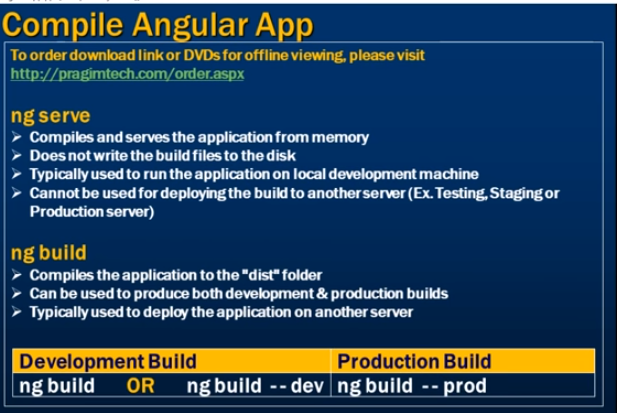
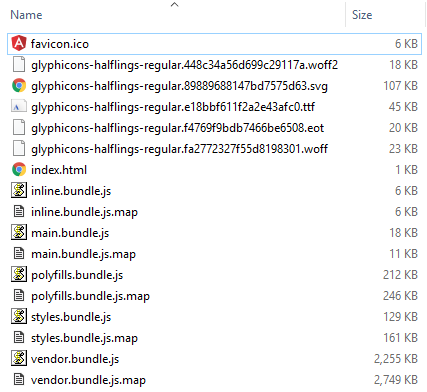
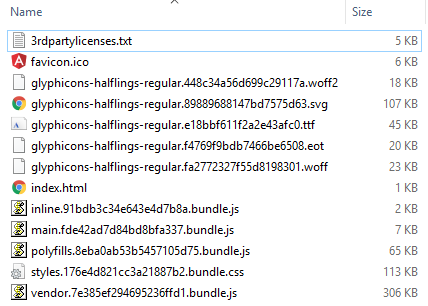
1. Click on the Windows Start Button and in the **"Search programs and files"** text box type: Control
2. Control Panel would appear in the list. Click on it.
3. In the **"Control Panel"** window, click on **"Default Programs"**
4. In the **"Default Programs"** window, click on **"Set your default programs"**
5. In the list of programs that appear, select the **"Browser"** that you want to be the default browser and then click on the link that says **"Set this program as default"**

That's it. At this point, execute "ng serve --open" command and you will have your application launched in your specified default browser.  
  
Instead of using the full option name --open, you can also use it's alias -o   
  
The following table shows the **common options, alias, default value** & their **purpose**

|  |  |  |  |
| --- | --- | --- | --- |
| **Option** | **Alias** | **Default** | **Purpose** |
| --watch | -w | true | Run build when files change |
| --live-reload | -lr | true | Whether to reload the page on change |
| --open | -o | false | Opens the url in default browser |
| --port | -p | 4200 | The port on which the server is listening |
| --extract-css | -ec |  | Extract css from global styles onto css files instead of js ones |

23. Compile angular app

In this video we will discuss **compiling angular applications**. Along the way we will discuss producing builds both for development and production use. We will also discuss the differences between ng serve and ng build commands.   
  
  
In [Parts 21](https://www.youtube.com/watch?v=cINAwGVbubg) and [22](https://www.youtube.com/watch?v=m-MzAA0TOvA) of [Angular CLI tutorial](https://www.youtube.com/watch?v=rJ9o4TyhSuo&list=PL6n9fhu94yhWUcq5Pc16uf8YKXoZ87Vh_) we discussed ng serve command. This command builds and serves the application from memory for faster development experience. ng serve command does not write the build files to the disk, so we cannot use it for deploying our application on a different server. For example, if we want to deploy our application to a test, staging or production server we cannot use ng servecommand. For this we use a different command and that is ng build command. 

  
  
When we execute ng build command it creates a folder with name "dist" and copies all the build files into that folder. Now the question that comes to our mind is, why is the folder named "dist". The folder is named "dist" because that is what is specified as the output directory for the build in the Angular CLI configuration file. Notice the "outDir"property is set to "dist".  
  
By default the ng build command does a development build, not a production build. The development build is not optimised for production use. The development build is typically used for testing. With a development build it is easier to debug as the development build contains source map files.  
  
ng build command on my machine produced the following files in the "dist" folder   
   
  
**As you can see in the "dist" folder we have**  
1. The favicon  
2. Glyphicon files  
3. Our host page index.html  
4. Bundle files and their corresponding source map files  
  
**Please note :** Both the following commands are equivalent and does the same thing, i.e they produce a development build  
ng build or ng build --dev  
  
If you want to deploy the application to a server, copy the contents of the "dist" folder to a folder on the server. We will discuss deployment in detail in a later video.  
  
The bundle files (inline, main, polyfills,styles, & vendor) generated by the development build are not optimised, meaning the bundles are not minified or treeshaked to remove the code that is not being used. A production build on the other hand will have all the performance optimisation techniques like Ahead-of-time (AOT) compilation, minification, uglification and treeshaking implemented. So the sizes of the bundles that the production build produces will be significantly less than the sizes of the bundles that a dev build produces.   
  
To do a production build use --prod option with the ng build command. ng build command with --prod option on my machine produced the following files in the "dist"folder.    
 

1. Notice the file sizes in the production build are significantly less than the file sizes in the development build.
2. With the production build, by default, we do not get the source map files because we usually do not need them on a production server.
3. Also notice, Production build extracts css from global styles into a css file instead of js ones.

In addition to these 3 differences between a dev build and a production build, there are several other differences as well. We will discuss them in detail in our next video.  
  
**ng serve vs ng build**   
  
ng serve 

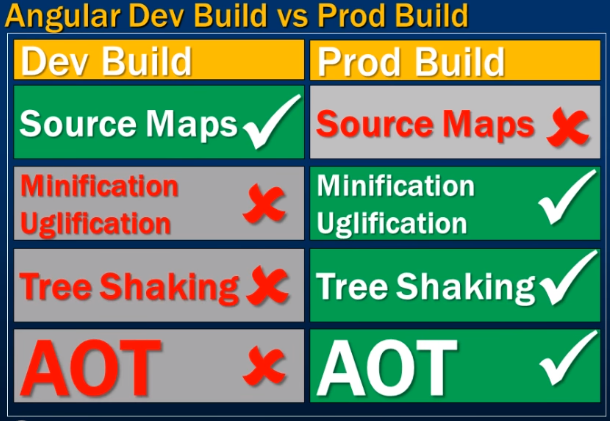
* Compiles and serves the application from memory
* Does not write the build files to the disk
* Typically used to run the application on local development machine
* Cannot be used for deploying the build to another server (Ex. Testing, Staging or Production server)

ng build 

* Compiles the application to the "dist" folder
* Can be used to produce both development & production builds
* Typically used to deploy the application on another server

24. Angular dev build vs prod build

In this video we will discuss the **differences between a development build and a production build in angular**.   
  
**To generate a development build we can use either**  
ng build   
OR  
ng build --dev   
  
**To generate a production build we use**  
ng build --prod   
  
  
Here are some of the differences between a development build and a production build in angular.  
  
**Source Maps :** Development build generate Source Maps where as production build does not.   
  
**What are Source Maps**  
To improve the performance, the application's JavaScript and CSS files are combined and compressed. It is extremely difficult to debug those compressed files. A source map holds information about the original files and can be used to map the code within a compressed file back to it’s original position in a source file. So with the help of these source maps we can easily debug our applications even after the files are compressed and combined.  
  
By default, a development build produce source maps where as a production build does not. However, we can change this default behaviour by using --sourcemaps option along with the ng build command. It's alias is -sm.  
  
The following command produces a development build without source maps as we have set -sm option to false  
ng build --dev -sm false  
  
On the other hand, if you want source maps along with your production build set -sm option to true as shown below.  
ng build --prod -sm true  
  
**Extracts CSS :** With the development build global styles are extracted to .js files where as with the production build they are extracted to .css files. To change this default behaviour use --extract-css option or it's alias -ec with the ng build command.  
  
The following command produces a development build with global styles extracted to .css file(s) instead of .js ones.  
ng build --dev -ec true  
  
**Minification & Uglification :** A Prod Build is both minified and uglified, where as a Dev Build is not.  
  
**What is Minification**  
The process of removing excess whitespace, comments, and optional tokens like curly brackets and semicolons is called Minification.   
  
**What is Uglification**  
The process of transforming code to use short variable and function names is called uglification.  
  
The minified and uglified version of the file is smaller in size than the full version, resulting in faster response times and lower bandwidth costs.  
  
If you look at the bundles generated by the prod build, you will notice that they are minified and uglified. Notice, extra whitespaces, comments, and optional tokens like curly brackets and semicolons are removed. Also notice, the code is transformed by using short variable and function names. On the other hand, the bundles generated by the dev build, are not minified and uglified.  
  
**Tree Shaking :**A Prod build is Tree Shaked, where as a Dev build is not.  
  
**What is Tree Shaking**  
Tree shaking is the process of removing any code that we are not actually using in our application from the final bundle. It's one of the most effective techniques to reduce the application size.  
  
If you look at the bundles generated by the production build, they are significantly less in size compared with the bundles generated by the development build. This is beacause with the production build the code is tree shaked to remove dead code i.e the code that is not referenced by the application.



**Ahead-of-Time (AOT) Compilation :**With a production build we get AOT (Ahead-of-Time) compilation, i.e the Angular component templates are pre-compiled, where as with a development build they are not. We will discuss Ahead-of-Time compilation in detail in our next video.  
  
The following table summarises the differences between a **development build** and a **production build**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Development Build** | **Production Build** |
| Source Maps | Yes | No |
| Extracts CSS | .js file | .css file |
| Minifaction | No | Yes |
| Uglification | No | Yes |
| Tree Shaking | No | Yes |
| AOT | No | Yes |

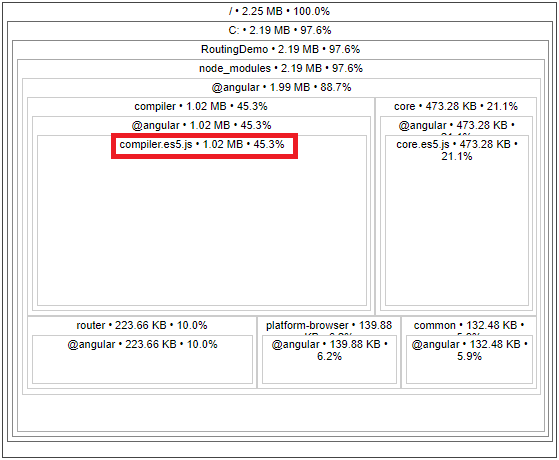
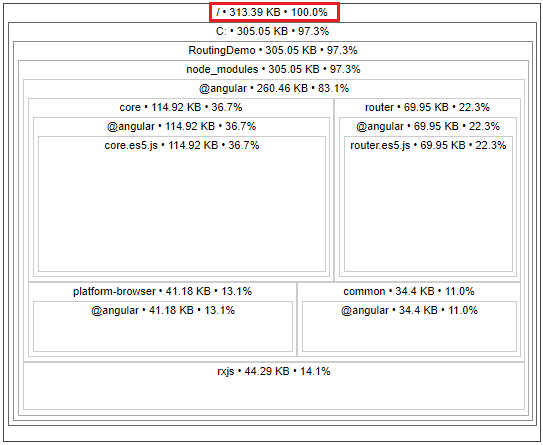
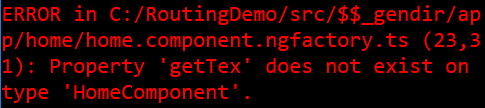
25. Angular AOT vs JIT

In this video we will discuss **Ahead-of-Time compilation** and **Just-in-Time compilation** in Angular.   
  
  
**In Angular we have 2 models of compilation** 

* **JIT** - Just-in-Time Compilation : JIT compilation as the name implies, compiles the application Just-in-Time in the browser at runtime.
* **AOT** - Ahead-of-Time Compilation : AOT compilation compiles the application at build time.

By default, with the development build we get **JIT compilation**. This is how it works. The application code along with the angular compiler is downloaded by the browser. At run-time, when a request is issued to the application, the **JIT-compiler** in the browser compiles the application code before it is executed. This means our user who made that first request has to wait for the application to compile first.   
  
In our previous videos we have seen that, when we build our angular application, the following JavaScript bundles are generated. 

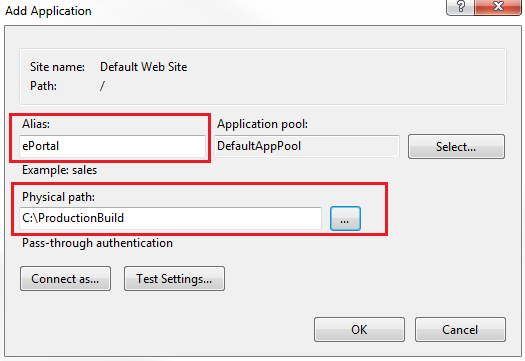
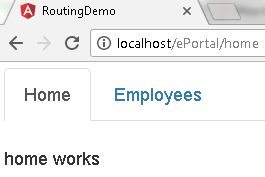
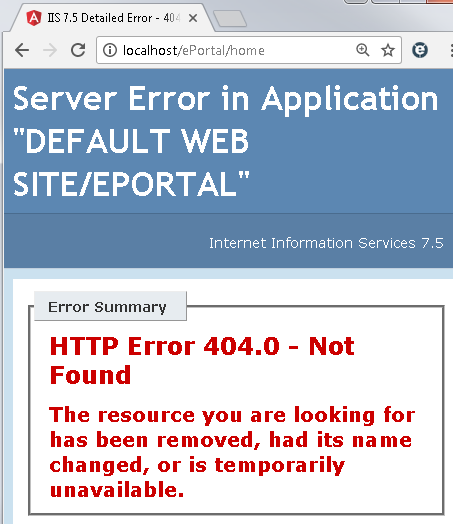
1. Inline
2. Main
3. Polyfills
4. Styles
5. Vendor

The vendor bundle contains the compiler along with the angular framework. **The compiler code is roughly half of the Angular framework.**  
  
There is a tool called **source-map-explorer** that we can use to inspect the JavaScript bundles. This tool analyzes the source map generated with the bundle and draws a map of all dependencies.  
  
To be able to use this tool we have to install it first. To install this tool, execute the following command  
npm install source-map-explorer --save-dev  
  
Once we have the tool installed, if you have not done the development build yet, do the development build using the following command.  
ng build  
  
Once the build is complete, you will have the JavaScript bundles along with the source map files. Now execute the following command.   
node\_modules\.bin\source-map-explorer dist\vendor.bundle.js  
  
The above command runs the **source-map-explorer**against the vendor bundle and we see the graph of it as shown below. Notice the angular compiler is around 45% percent of the bundle size. As this is development build and not optimised, notice the total size of the bundle is 2.19 MB.   
   
  
**With AOT compilation the angular application is pre-compiled**. So this means the browser loads executable code so it can render the application immediately, without waiting to compile the application first.   
  
This also mean with AOT, as the application is already pre-compiled, there is also no need for the browser to download the Angular compiler. As we already know, the compiler code is roughly half of the Angular framework, so omitting it dramatically reduces the application size.  
  
**By default, the production build is Ahead-of-Time compiled.** So there is no need to bundle up the angular compiler code in the vendor bundle. This brings down the vendor bundle size by almost 50%. In addition it is also minified, uglified and tree-shaked to remove any code that we are not referencing in our application. So the bundler size is further reduced.  
  
Now, execute the following command to generate a production build. Notice I have also turned on sourcemap option. Without the sourcemap we will not be able to use the source-map-explorer tool.  
ng build --prod --sourcemap true  
  
Once the production build is complete, execute the following command. Vendor bundle name in your production build may be slightly different. Change it accordingly and execute the command.  
node\_modules\.bin\source-map-explorer dist\vendor.7e385ef294695236ffd1.bundle.js  
  
Here is the graph produced by the above command. Notice now, we do not have the compiler in the bundle. The bundle size is just 313 KB.   
   
  
The AOT compiler also detects and reports template binding errors at build time itself. Let us understand this with an example.  
  
Include the following function **HomeComponent**class in home.component.ts file  
getText(): string {  
  return 'Hello Pragim';  
}  
  
In home.component.html include the following <div> element. Notice I have deliberately mis-spelled the getText() function name.  
<div [innerText]='getTex()'>  
  
Save changes, and execute the following command. This command does a development build in-memory. At the moment we are not using AOT, so we will not know about the template binding error that is introduced above. Notice at build time we do not see any errors.  
ng serve  
  
Now launch your default browser and navigate to http://localhost:4200. We see the "home works" message as expected. However, we do not see the message returned by getText() function. This is because we deliberately mis-spelled the getText() function name. We will see this error in the browser developer tools. So this proves that, with JIT compilation we will only come to know about the template binding errors at runtime.  
  
**With AOT compilation,** template binding errors are detected and reported at build time itself as apposed to runtime. To prove this, execute the following command. Notice we are using --aot option for pre-compiling our in-memory build.  
ng serve --aot  
  
The build completes with the following error. So this proves that, with AOT compilation, template binding errors are detected and reported at build time itself as apposed to runtime   
   
  
By default, the following 2 commands use JIT compilation   
ng build  
ng serve  
  
With either of these command we can use --aot option to turn on Ahead-of-Time compilation  
ng build --aot  
ng serve --aot  
  
The production build uses AOT by default. If you want to turn off AOT for the production build, you can do so by setting --aot option to false as shown below.  
ng build --prod --aot false  
  
At this point, we cannot use **source-map-explorer** to check if the angular compiler is in the vendor bundle because we do not have sourcemap files. If you want to inspect the bundles make sure you also turn on sourcemaps.

26. Deploy angular app to IIS

In this video we will discuss **deploying angular application to IIS**.   
  
  
**Here are the steps**   
  
**Step 1 :** Build your angular application.   
  
If you want to deploy a development build do a development build using the following Angular CLI command. The base-href option on the build command sets the base-hrefelement in index.html to "/ePortal/" instaed of "/". In the IIS server, we will create an application with name "ePortal" in just a bit.  
ng build --base-href /ePortal/   
  
  
If you want to deploy a production build do a production build using the following Angular CLI command.   
ng build --prod --base-href /ePortal/  
  
In our case let's deploy a production build. After the build is complete, you will notice a folder with name **"dist"** in your Angular project folder. This folder contains all the build files. These build files need to be copied to a folder on the server where we have IIS installed.  
  
**Step 2 :** Create a folder on the server where you have IIS installed. You can name the folder anything you want. I am going to name the folder **"ProductionBuild"** and I am creating it in **C:\** drive.  
  
**Step 3 :** Now copy all the contents of the **"dist"** folder into **"ProductionBuild"** folder  
  
**Step 4 :** Open IIS. There are several ways to do this. One way is to type **"inetmgr"** in the **"Run"** window and click **"OK"**  
  
**Step 5 :** Create an application in IIS. Name it "ePortal". This name has to match the value we have specified for the --base-href option in Step 1.  

* Exapand the root IIS node
* Expand **Sites**
* Right click on **"Default Web Site"** and select **"Add Application"** from the context menu
* In the **"Alias"** textbox, type **"ePortal"**
* Set the **"Physical Path"** to folder that contains the build files. In our case it is **"ProductionBuild"** folder in **C:\** drive

   
  
At this point, if you launch the browser and navigate to http://localhost/ePortal/home, you will see the **"home works"** message as expected. When you click on the **"Employees"** tab it also works as expected.   
  
   
  
However, when you **"Refresh"** the page by pressing F5, you will see the following HTTP 404 error   
  
   
  
**Step 6 :**  To fix this Page Refresh issue in Angular, include the following URL rewrite rule in you web.config file. This web.config file should be in copied the **"ProductionBuild"** folder where we have the rest of the build files.

<?xml version="1.0" encoding="utf-8"?>

<configuration>

  <system.webServer>

    <rewrite>

      <rules>

        <rule name="AngularJS Routes" stopProcessing="true">

          <match url=".\*" />

          <conditions logicalGrouping="MatchAll">

            <add input="{REQUEST\_FILENAME}" matchType="IsFile" negate="true" />

            <add input="{REQUEST\_FILENAME}" matchType="IsDirectory" negate="true"/>

            <add input="{REQUEST\_URI}" pattern="^/(api)" negate="true" />

          </conditions>

          <action type="Rewrite" url="/ePortal" />

        </rule>

      </rules>

    </rewrite>

  </system.webServer>

</configuration>

**Please note :** You may also point the IIS application directly to the **"dist"** folder in **RoutingDemo**project folder. The downside of this is every time you rebuild your application, the **"dist"** folder is deleted and recreated.  This means you will loose the **web.config** file and you have to create it again.