Angular: Quick Notes

Version 1

Kapil Sharma

2018-10-27

Contents

1 About these notes

These notes are distributed with ***Corporate Training: Angular*** meetup scheduled by ***Kapil Sharma*** and ***PHP Reboot*** on October 27th, 2018.

Please remember, these are not complete notes to teach you angular, but quick notes as most people make to remember a concept that they learned or understood earlier.

If you wish to learn Angular, ***Corporate Training: Angular*** meetup videos will be available on Kapil's `YouTube Channel`[1] after the meetup. In case of any doubt, you can get in touch with Kapil through `Twitter`[2] or `Linked In`[3]

1.1 Sponsors

Corporate Training: Angular meetup series by PHP Reboot is sponsored by `Ansh Systems`[4] and `Jet Brains`[5].

As Sponsors, ***Ansh Systems*** is providing arrangements and snacks for the meetup. Ansh Systems (Do Business As `Eastern Enterprise`[6] in Europe), is having 200+ employees development center in India, working on technologies like PHP, Node JS, Dot Net, Angular, React, Vue, Android and iOS native development etc. This whole corporate training meetups series are actually one of their many internal corporate trainings, provided to their new and existing employees. As sponsors, Ansh Systems also allowed PHP Reboot to publically share their internal corporate training of Angular.

Jet Brains is expert in providing great IDE like PHP Storm, Web Storm, Intelli J Idea etc. As sponsors, they are providing two Jet Brains license for any of their IDE to be raffled out during the meetup.

[1] Kapil's YouTube Channel http://bit.ly/kapilyoutube
[2] Kapil's Twitter https://twitter.com/kapilsharmainfo
[3] Kapil's Linked In https://www.linkedin.com/in/kapilsharmainfo
[4] Ansh Systems: http://www.ansh-systems.com
[5] Jet Brains: https://www.jetbrains.com
[6] Eastern Enterprise B.V.: https://www.easternenterprise.com

2 Install & First Application

2.1 Node JS

As prerequisite, Angular need Node JS. Angular application does not depends on Node JS but while development, development tools of angular like CLI, Development Server, installing dependencies etc take advantage of tools provided by Node JS. Node JS can be downloaded from `Official Website`[1]. It is recommended to install latest version, not LTS version.

It is recommended to install Node JS using `Node Version Manager (NVM)`[2]. Angular works best with Latest version. However, if you are a Node JS developer, or depends on Node JS for other tasks, you might want to stay with LTS version. NVM, not only allow us to install multiple Node JS versions, but also it is very easy to switch versions of Node JS with NVM.

Intalling Node JS directly or through NVM is straight forward. Please google the latest information on installing Node JS.

2.2 Angular CLI

Angular CLI makes it very easy to work with Angular. We mostly develop Angular Applications in `Type Script`[3], which is super set of Java Script. Don't worry about it, Type Script will covered during the Corporate Training Meetup. Type Script needs to be compiled to Java Script. Also, we have lot of dependencies for the project development and a local server to test the application. Angular CLI manages all that and makes developer's life very easy.

We can install Angular CLI globally with following command

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

[1]https://nodejs.org/en/
[2]https://github.com/creationix/nvm
[3]Type Script: https://www.typescriptlang.org

2.3 First Angular Application

Angular CLI, installed above, can be used with `ng` command. Run following command

```
ng new HelloWorldApp
```

The command will create a new folder `HelloWorldApp` with angular project. To run local server, go to new project folder and run command

```
ng serve
```

After that, open `localhost:4200` in the browser to check default application.

2.3.1 Directory Sctucture

At root level, in project, you will find 3 folders

- e2e
  - This folder stands for `End to End` testing and contains test cases. We will check testing later.
- node_modules
  - This folder is created by npm (Node Package Manager) and contains all the dependencies of the project.
- src
  - This is the folder which contains source code of our project and while developing angular application, we will be mostly working within this folder.
  - This folder also contains few files and folder. We will be mostly using `app` folder and know about other files and folder as and when needed while we advance with angular learning. For now, just do not touch any other file/folder.

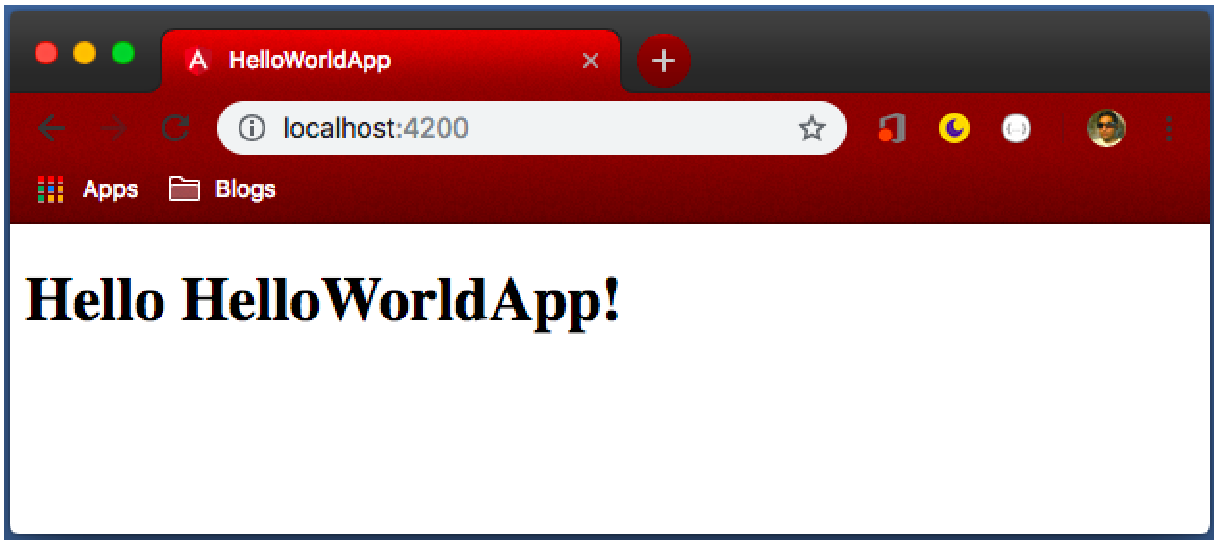
Other that these 3 folders, you will find few files as well at root level. They are mostly supporting files to configure our new Angular project.

2.3.2 Changing output for First Application

Let's develt everything in `app/app.component.html` and put simple code like

```
<h1>
  Hello {{ title }}!
</h1>
```

The output will be as follow



```
Hello HelloWorldApp!
```

Figure 2.1: Hellow World App

We can see `{{ title }}` has been replaced by `HelloWorldApp`. This is done in `app.component.ts` file.

```
 1 import { Component } from '@angular/core';
 2 
 3 @Component({
 4   selector: 'app-root',
 5   templateUrl: './app.component.html',
 6   styleUrls: ['./app.component.css']
 7 })
 8 export class AppComponent {
 9   title = 'HelloWorldApp';
10 }
```

Notice line 9 `title = 'HelloWorldApp';`. This is actually where we are assigning value to a variable `title`, which is later used in `app.component.html` file. This is actually called ***String Interpolation***. Thus, if there is a variable in Type Script class, we can display it in HTML with String Interpolation like `{{ variableName }}`. We will learn more about it in "Data Binding" section of next chapter "Understanding component; Basic building block of Angular".

2.4 How our application works?

We saw we can update “app.component.html” to update design and “app.component.ts” to change the variable, but how are they working?

To understand it in better way, let’s check the source code of application in browser (Right click and select “view source” in chrome). It is like

```html
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>HelloWorldApp</title>
6    <base href="/">
7
8    <meta name="viewport" content="width=device-width, initial-scale=1">
9    <link rel="icon" type="image/x-icon" href="favicon.ico">
10 </head>
11 <body>
12   <app-root></app-root>
13 <script type="text/javascript" src="runtime.js"></script><script type="
       text/javascript" src="polyfills.js"></script><script type="text/
       javascript" src="styles.js"></script><script type="text/javascript"
       src="vendor.js"></script><script type="text/javascript" src="main.js
       "></script></body>
14 </html>
```

What!! Our code from “app.component.html” (`<H1>` tag) or “app.component.ts” (`title` variable) is not there. How it is working then?

We write angular code in Type Script, which is super set of Java Script but the browser do not understands Type Script. Thus, to display the page, angular needs to compile our application into Java Script.

Notice Line 13, here we are including few Java Script files

- runtime.js
- polyfills.js
- styles.js
- vendor.js
- main.js

We didn’t wrote any of these JS file. They are actually dependencies of Angular application and in this simple Hello World Application, main.js contains our actual source code. We can also notice line 12

`<app-root></app-root>`, which is updated by generated JS files to display out contents. However, let's not try to understand generated Java Script; Angular CLI makes sure that generated JS files are optimized but not very convenient to read by humans. Instead, let's concentrate on how Angular CLI read our application to generate these JS files.

When we run `ng serve` (or `ng biuld`, as we will see later), Angular CLI first looks `angular.json` file to check the configurations. These configurations are well set and mostly we need not to touch them. From it, it identify main Type Script file is `src/main.ts` as follow:

```
 1 import { enableProdMode } from '@angular/core';
 2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
 3 
 4 import { AppModule } from './app/app.module';
 5 import { environment } from './environments/environment';
 6 
 7 if (environment.production) {
 8   enableProdMode();
 9 }
10 
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .catch(err => console.error(err));
```

Here, first few lines are **`import`**. It is a Type Script feature in include other files in the program. Afther them, it sets environment (development, testing, staging, production, or as you wish). Then at line 11, `bootstrapModule` inform Angular CLI that `AppModule` is the first module to be initialized and actually is the starting point of our application. All files of `XyzModule` are stored in folder `src/xyz`. Thus, files of AppModule are stored in folder `src/app`.

To load App module, Angular CLI will first check `src/app/app.module.ts` file, which, by default, looks as follow

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

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

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
12    ],
13    providers: [],
14    bootstrap: [AppComponent]
15  })
16  export class AppModule { }
```

Here we get first important lession about angular, Type Script part of this file contains import statements (first four line) and an empty class (last line).

Line 6-15 looks like doc-block, or metadata about the class. In Angular, it is called decorator. Module Decorator, to be exact as it starts with `@NgModule`, which is imported from angular core library in line 2. It takes a Java Script object as an argument. This Java Script object contains few few important properties, most important here is `bootstrap` (line 14). This tells Angular CLI which component should be loaded first. For now, our application contains just one module and one component but as we start making complex applications, we will have multiple modules and each module will contain several components.

Now angular knows it need to start with AppComponent, it will further check `app.component.ts` file.

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

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

Here, we can find Component Decorator. Most important to note is `selector`, which is `app-root`. If you remember, we saw `<app-root></app-root>` on line 11 of generated HTML. It tell angular, it need to replace `app-root` tag with `app component` and code of `app.component.html` will replace `app-root` on browser through generated java script.

This is just high level view of how angular applications work. We are not ready to learn more about components, which are basic building blocks of an angular application.

3 Components; Angular's basic building blocks

Components are the core of an Angular application. Every angular application have at least one component. Every component generally have 4 files:

- TypeScript file (name.component.ts)
  - This is a Type Script class with `@Component` decorator. It defines data and logic for the component
- HTML file (name.component.html)
  - Although having this it is not mandatory as we can define HTML template in component decorator as well, but generally component HTML have multiple lines, which we mostly prefer to define in separate HTML file.
- Styles file (name.component.css)
  - It is style sheet for component's HTML. Like HTML, can can define CSS also in component decorator, but in most cases, to separate different type of code in different files, we mostly define styles in separate css file.
- Tests (name.component.spec.ts)
  - This file contains test cases for the component. We will discuss how to write unit tests for angular application but for most of the training, we will concentrate to understand different angular features. Thus, until we discuss about unit tests, we will not use this file in any component.

3.1 Creating a component

There are two ways to create new components:

1. Manually
2. Using Angular CLI

Let's check both of them one by one.

3.1.1 Creating a component manually.

We already discussed that a component generally have 3 files; TypeScript class, HTML Template and CSS styles.

In out HelloWorldApp, lets create a new component to display "Manual compoent".

As we discussed in `Section 2.4 How our application works`?, a module bootstraps only one component and we already bootstraped "AppComponent". All other components, now should be called or loaded from AppComponent.

> Separate folder for each component: A complex angular application contains many components. Thus, as good a good coding practice, we should create separate folder for each component and they should follow hirarchical order. For example, if app component is going to load "Manual-Component", it must be in "app/manual" folder.

For this example, we are going to name our new componene as `ManualComponent`. Following coding best practices of angular, lets create folder `manual` in `app` folder. Then we need to create 3 files

src/app/manual/manual.component.ts

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-manual',
5    templateUrl: './manual.component.html',
6    styleUrls: ['./manual.component.css']
7  })
8  export class ManualComponent { }
```

As we discussed, Type Script is super set of Java Script. Like Java Script, if we include a TS file into another TS file, included TS file needs to export something. Thus, on line 8, we are exporting a class named `ManualComponent`. However, we need not to code anything right now so this class is empty.

From angular's point of view, to make this file as component, we need to define component decorator. Component is something provided by Angular, not Type Script. Thus, in first line, we are importing Component from angular core library. Once imported, we can use `@Component()` decorator on "ManualComponent" class to tell angular this is a component.

A Component decorator needs three things (as JS object)

- selector
  - It tells angular how other components will refer this component. In other words, it define the tag which we can use in HTML. To avoid confusion with other HTML tags, as a coding

best practice, we prefix our custom tags with something, mostly with `app-`. Thus, out selector is `app-manual`. However, you are free to choose any name, provided it is unique.

- templateUrl
  - It defines the template (HTML) for the component. As discussed earlier, in case we just have one or two line HTML, we can also define HTML directly here. In that case, we define template, instead of templateUrl. However, either "template" or "templateUrl" is mandatory.
- styleUrls
  - Please note, we have only one template but can have multiple style files for a component. Thus, it is styleUrls (Notice "s" at the end). Also, unlike template, it is and array, not string.
  - Like templateUrl, we can also replace styleUrls with styles. Any one of them is mandatory.

Lets also define manual.component.html, which is ideally just one line of code

```
<p>Loaded from Manual Component</p>
```

Similarly, manual.component.css is also very simple style sheet

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

We defined the component but we are still not using it anywhere. Lets use our new tag in app.component.html

```
<h1>
  Hello {{ title }}!
</h1>
<app-manual></app-manual>
```

Although it seems we are done, we have one final step remaining. Angular still do not know about our new component and will throw an errer as soon as it comes across `app-manual` tag in app component's html. To tell angular about our new component, we must update `app.module.ts` file.

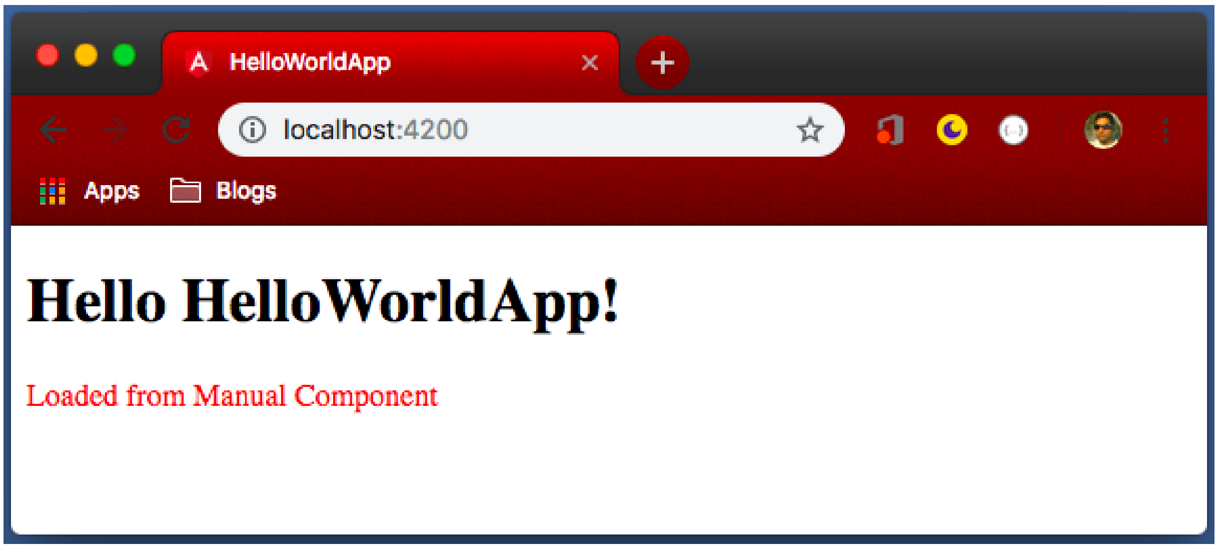
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

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

@NgModule({
  declarations: [
 9      AppComponent,
10      ManualComponent
11    ],
12    imports: [
13      BrowserModule
14    ],
15    providers: [],
16    bootstrap: [AppComponent]
17  })
18  export class AppModule { }
```

Here, we added line 5, where we are importing our new component. Still, just importing a Type Script file will not work. In line 10, we are actually declaring our new component. Declaration means, angular will not immediately load this new component but will read its component decorator and will know about new tag. As soon as we use new tag in any component (app.component.html in our case), angular will load (create instance of ManualCompoent) our component there.

With this, our application now loads our new component



Hello HelloWorldApp!

Loaded from Manual Component

Figure 3.1: Manual Component

3.1.2 Creating a component using Angular CLI.

It is good to understand how angular components works. However, once we understand it, instead of creating components manually, we can use angular CLI to actually create our components. This will fast track component creation, as single command can create all the required files as well as make necessary changes in module.ts file.

> Angular CLI starts with `ng` command. We already saw `ng new` and `ng serve` commands. They both were Angular CLI commands.

Command to generare new component is `ng generate component <name>`. Angular CLI also comes with lot of short-cut command. Short cut command to generate component is `ng g c <name>`. Let's run `ng generate component cli` command to create new component "CliComponent" through Angular CLI.

```
ng generate component cli
CREATE src/app/cli/cli.component.css (0 bytes)
CREATE src/app/cli/cli.component.html (22 bytes)
CREATE src/app/cli/cli.component.spec.ts (607 bytes)
CREATE src/app/cli/cli.component.ts (257 bytes)
UPDATE src/app/app.module.ts (466 bytes)
```

As we can see in output, it generated 4 files and updated app.module.ts file. Let's also update "cli.component.html" and "cli.component.css" to match our manual component style.

cli.component.html

```
<p>Loaded from CLI Component</p>
```

cli.component.css

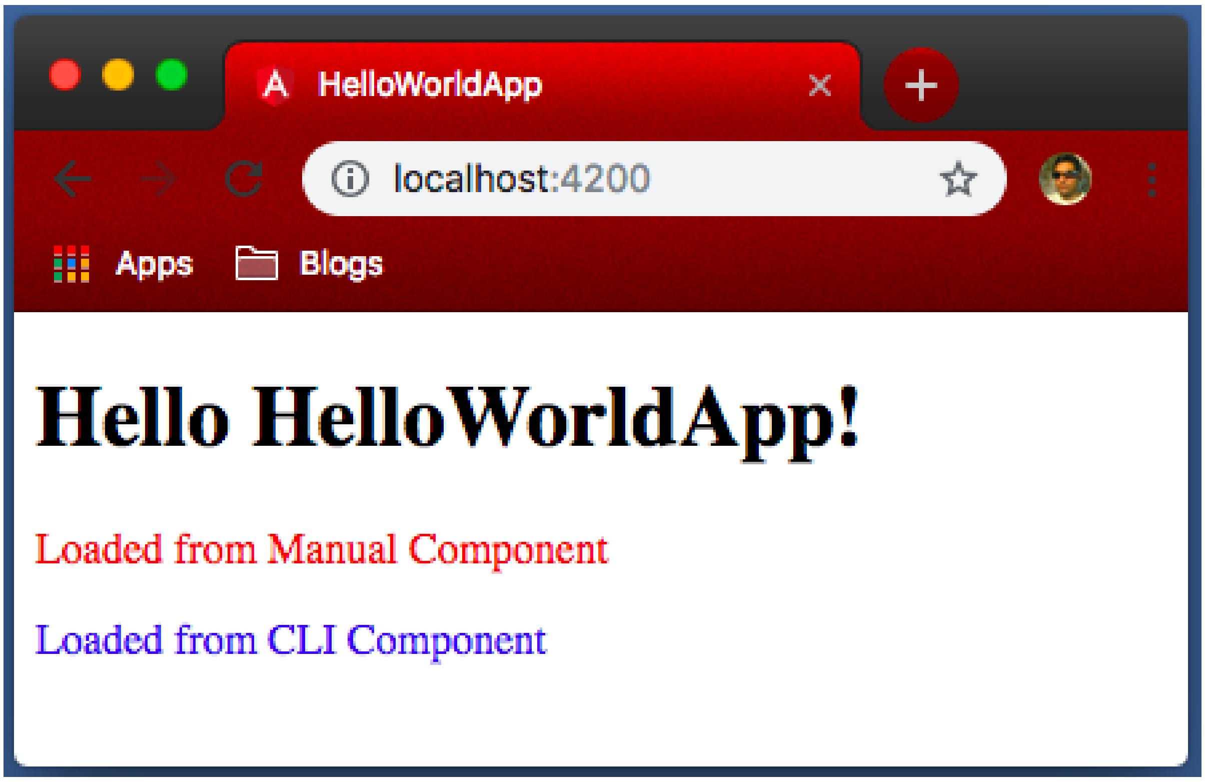
```
p {
  color: blue;
}
```

We also need to add our selector "app-cli" (auto generated, check in ts file) in "app.component.html"

app.component.css

```
<h1>
  Hello {{ title }}!
</h1>
<app-manual></app-manual>
<app-cli></app-cli>
```

Now our generated output is as follow



Hello HelloWorldApp!

Loaded from Manual Component

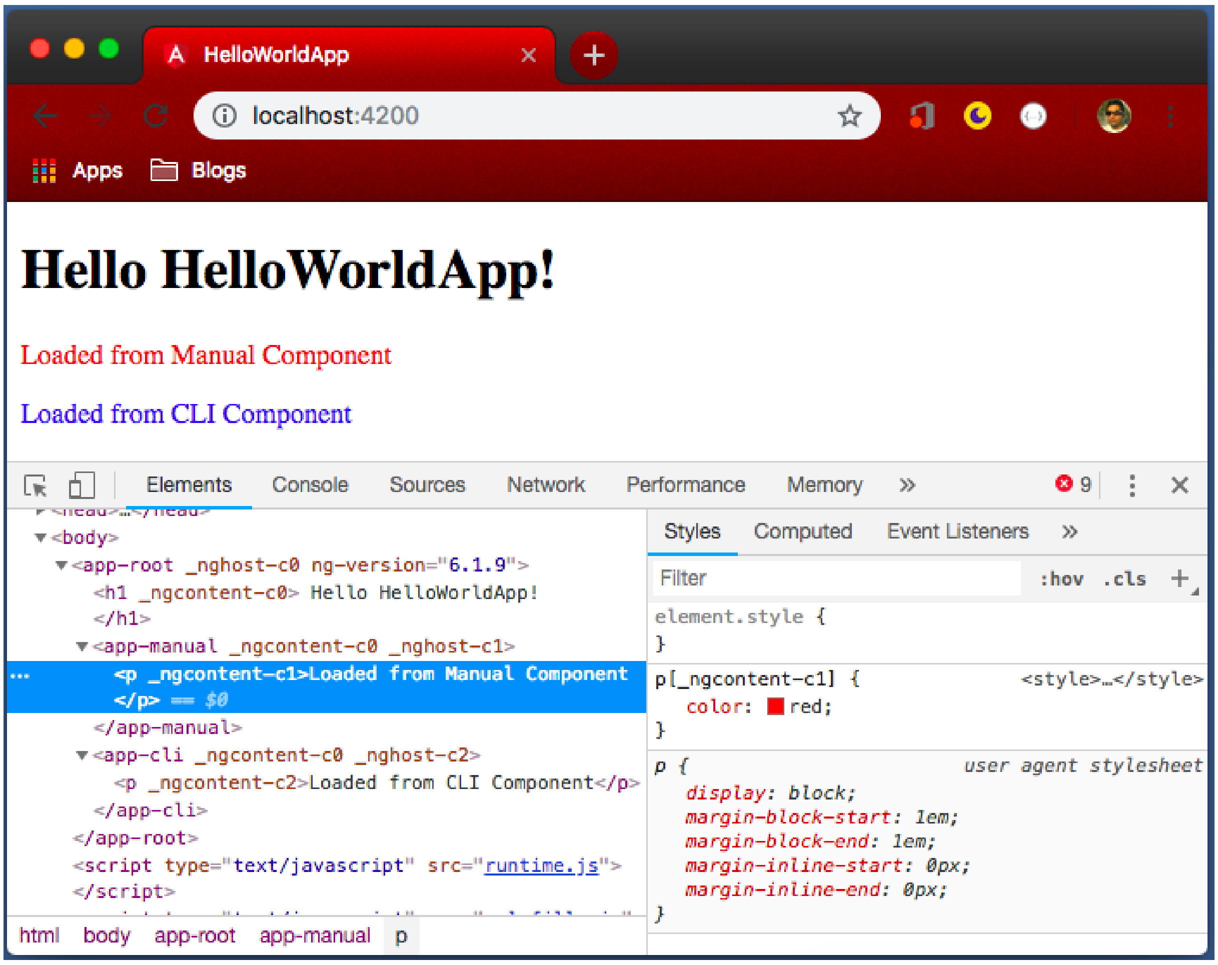
Loaded from CLI Component

Figure 3.2: Cli Component

3.2 Component stylesheet

In above image, we can see manual component is in red and cli component is in blue, exactly as we defined in our css.

However, if you were checking closely, you might be wondering that both components have `<p>` tag, then how angular manage to define separate styles for single tag. To understand this, lets use chrome's dev tools to check generated HTML code as shown in following image.



Hello HelloWorldApp!

Loaded from Manual Component

Loaded from CLI Component

```
<head>…</head>
▼<body>
  ▼<app-root _nghost-c0 ng-version="6.1.9">
     <h1 _ngcontent-c0> Hello HelloWorldApp!
     </h1>
   ▼<app-manual _ngcontent-c0 _nghost-c1>
... <p _ngcontent-c1>Loaded from Manual Component
      </p> == $0
     </app-manual>
   ▼<app-cli _ngcontent-c0 _nghost-c2>
      <p _ngcontent-c2>Loaded from CLI Component</p>
     </app-cli>
   </app-root>
   <script type="text/javascript" src="runtime.js">
   </script>
```

```
element.style {
}
p[_ngcontent-c1] {                    <style>…</style>
  color: red;
}
p {                          user agent stylesheet
  display: block;
  margin-block-start: 1em;
  margin-block-end: 1em;
  margin-inline-start: 0px;
  margin-inline-end: 0px;
}
```

Figure 3.3: Dev tools

As you might notice, app-root tag have a special attribute _nghost-c0 and all the elements within it, including app-manual and all-cli, have attribute _ngcontent-c0. Here, c0 is the name angular provided to app-root.

Similarly, app-manual and app-cli have attributes _nghost-c1 and nghost-c2 and all element within them (only "p" tag) have attributes _ngcontent-c1 and _ngcontent-c2 respectively. Thus, angular provided them name as c1 and c2.

This way, angular may identify different components. In the same image, "p" tag under app-manual is selected and its styles are loaded on right hand side. Please note, angular have not generated generic style for p tag but for p[_ngcontent-c1] tag. In this way, angular makes sure the style we defined for an individual components, affets only that component. We will later learn how to define generic style that may be applied to all elements, regardless of component.