Components

Components are at the heart of Angular. The main idea is that you break down your application into different cohesive components and let the components handle the rest. Every component has a controller defined by a class and a template defined by html. In addition, a component's job is to enable the user experience and delegate everything non-trivial to services.

In this section we are going to write a simple `HelloAngular` component, compile it and run it in the browser. In addition, we will configure VSCode to build the TypeScript files as we go.

Note that there is a lot to talk about components. We are going dive into components a lot more in later chapters, but for now let's just keep things simple.

Project Files

The project files for this chapter are in **`angular2-intro/project-files/basic-component`**

You can either follow along or just look at the final result. As always, the `node_modules` folder is not included. You would have to install it with `npm i` in the project folder:

```
cd angular2-intro/project-files/basic-component && npm i
```

Getting Started

Make a folder on your desktop called `hello-angular` and navigate to it:

```
1  mkdir ~/Desktop/hello-angular && cd $_
```

Start npm in this folder with `npm init` and accept all the defaults.

After that, install the dependencies with:

```
1  npm i angular2 rxjs -S
```

Then install the "devDependencies":

```
npm i systemjs -D
```

After all the dependencies are installed, start VSCode in this folder with `code .`

Then create a `index.html` file in the root of the project and put in the following:

`index.html`

```
<html>
<head>
  <title>Hello Angular</title>

  <script src="/node_modules/angular2/bundles/angular2-polyfills.js"></script>
  <script src="/node_modules/systemjs/dist/system.src.js"></script>
  <script src="/node_modules/rxjs/bundles/Rx.js"></script>
  <script src="/node_modules/angular2/bundles/angular2.dev.js"></script>

  <!-- add systemjs settings later -->

</head>

<body>
  <!-- add app stuff later -->
</body>

</html>
```

This loads all the necessary scripts that we need to run Angular in the browser.

Note

If you need to support older browsers, you need to include the `es6-shims` before everything else:

```
<script src="/node_modules/es6-shim/es6-shim.js"></script>
```

Making the Component

Let's start by making the `main.ts` file in the root of the project. In this file we are going to define the main component called `HelloAngular` and then bootstrap the app with it:

`main.ts`

```
import {Component, OnInit } from 'angular2/core';
import {bootstrap} from 'angular2/platform/browser';

@Component({
  selector: 'app',
  template: `<h1> hello angular </h1> `
});

class HelloAngular implements OnInit  {
  constructor() { console.log('constructor called'); }
  ngOnInit() { console.log('component initialized'); }
}

bootstrap(HelloAngular, []);
```

- On line 1 we are importing the `component` meta data (annotation) and the `onInit` interface.
- On line 2 we are loading the `bootstrap` method that bootstraps the app given a component.
- On line 4, we are defining a component using the `component` annotation. The `@component` is technically a class decorator because it precedes the `HelloAngular` class definition.
- On line 5, we are telling angular to look out for the `app` tag. So when Angular looks at the html and comes across the `<app></app>` tag, it is going to load the template (on line 6) and instantiates the class for it (defined on line 9).
- On line 9, we are defining a class called `HelloAngular` that defines the logic of the component. And for fun, we are implementing the `OnInit` interface to log something to the

console when the component is ready with its data. We will learn more about the lifeCycle hooks later.

- Last but not least, we call the `bootstrap` method with the `HelloAngular` class as the first argument to bootstrap the app with the `HelloAngular` component.

Compiling the Component

Now we need to compile the file to JavaScript. We can do it from the terminal, but let's stick to VSCode. In order to that, we need to make two config files:

1. First is the standard `tsconfig.json` file

2. And the `tasks.json` file for VSCode to do the compiling

Create the `tsconfig.json` file in the root of the project and put in the following:

`tsconfig.json`

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false,
    "outDir": "output",
    "watch": true
  },
  "exclude": [
    "node_modules"
  ]
}
```

Then create the `tasks.json` in the `.vscode` folder in the root of the project and put in the

following:

`.vscode/tasks.json`

```
{
  "version": "0.1.0",
  "command": "tsc",
  "showOutput": "silent",
  "isShellCommand": true,
  "problemMatcher": "$tsc"
}
```

- Now we can build the TypeScript files as we work. We just need to start the build task with `command + shift + b` or using the prompt. If you want to use the prompt do the following:
  - Use `command + shift + p` to open the prompt
  - Then, type `> run build task` and hit enter to start the build task.
- After you run the build task, you should see an `output` file generated with `main.js` and the source maps in it.
- The task is watching the files and compiling as you go. To stop the task, open the prompt and type:

```
> terminate running task
```

Loading the Component

After compiling the component, we need to load it to the `index.html` file with `Systemjs`. Open the `index.html` file and replace `<!-- add systemjs settings later -->` with the following:

```
<script>
  System.config({
    packages: {
```

```
        output: {
          format: 'register',
          defaultExtension: 'js'
        }
      }
    });
    System.import('output/main')
    .then(null, console.error.bind(console));
  </script>
```

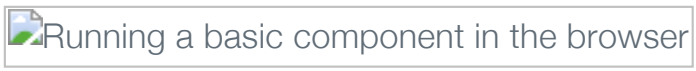
Now we can use our component in the body of the html:

```
<body>
  <app>Loading ...</app>
</body>
```

It is finally time to serve the app. You can serve the app in the current directory using the `live-server`:

```
live-server .
```

If everything is wired up correctly, you should be able to see the following:



Running a basic component in the browser