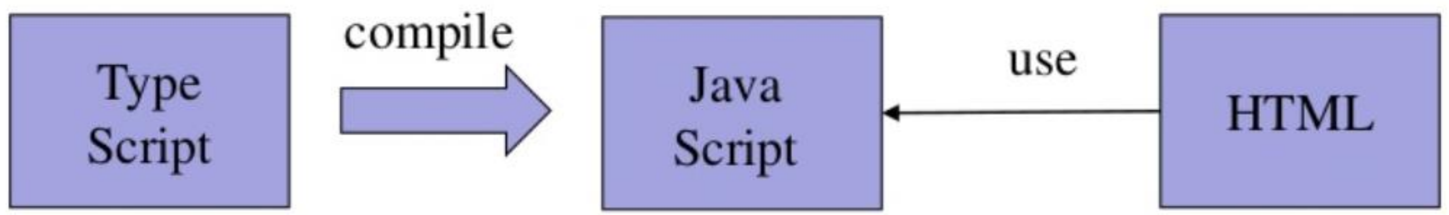
Angular Intro

What is Angular?

Angular is a framework that will provide us flexibility and power when building our apps.

- Takes advantage of ES6
- Web components
- Framework for all types of apps
- Speed improvements

- Angular code is written in TypeScript language
  - TypeScript is compiled into JavaScript
  - JavaScript is used in HTML Pages

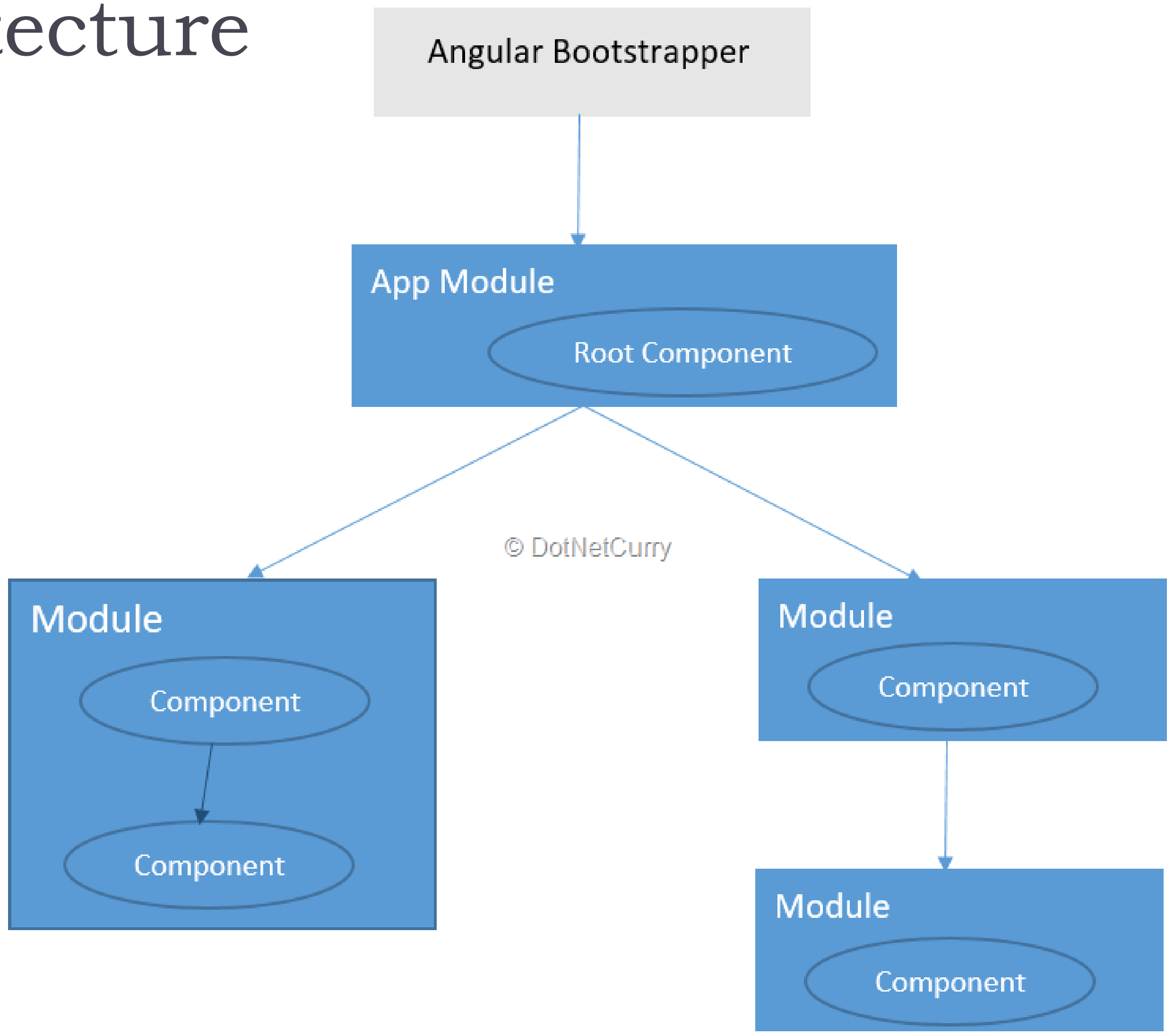


Angular enhances HTML

- Angular has set of directives to display dynamic contents at HTML page. Angular extends HTML node capabilities for a web applications.
- Angular provides data binding and dependency injection that reduce line of code.
- Angular extends HTML attributes with `Directives`, and binds data to HTML with `Expressions`.
- Angular follows MVC Architecture

Angular High Level Architecture

- An Angular application can be viewed as a tree of components.
- The application bootstraps using a component and rest of the application is then rendered with help of a number of sub-components.



Installation & Start up

Installation

- Install Node https://nodejs.org/en/

- Install the Angular CLI
  - `npm install -g @angular/cli`
- Check the version
  - `ng -version`

Create an Application

1. Run the CLI command `ng new` and provide the name `my-app`, as shown here:
   - `ng new my-app`
2. The `ng new` command prompts you for information about features to include in the initial app. Accept the defaults by pressing the Enter or Return key.
3. The Angular CLI installs the necessary Angular npm packages and other dependencies. This can take a few minutes.

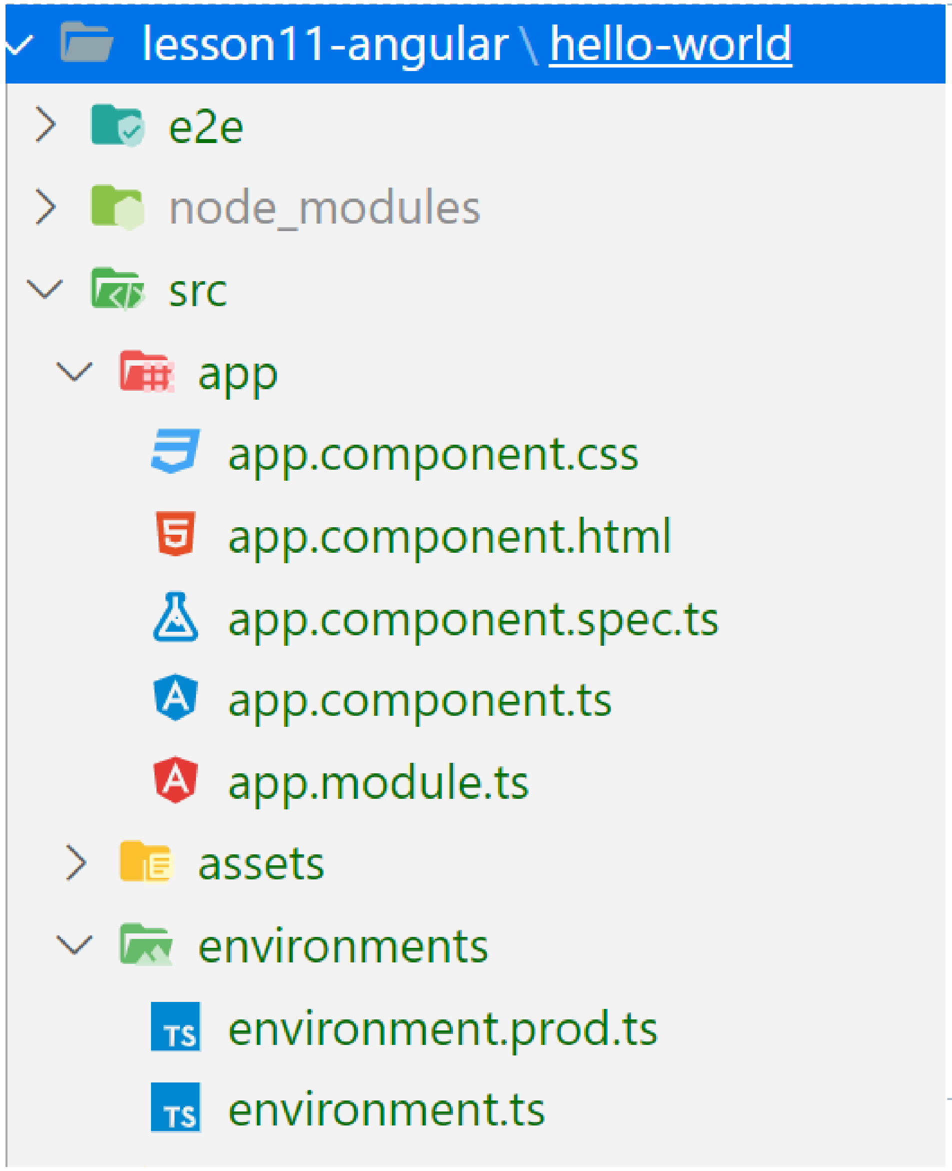
Run the application

- The Angular CLI includes a server, so that you can easily build and serve your app locally.
  1. Go to the workspace folder (`my-app`).
  2. Launch the server by using the CLI command `ng serve`, with the `--open` option.

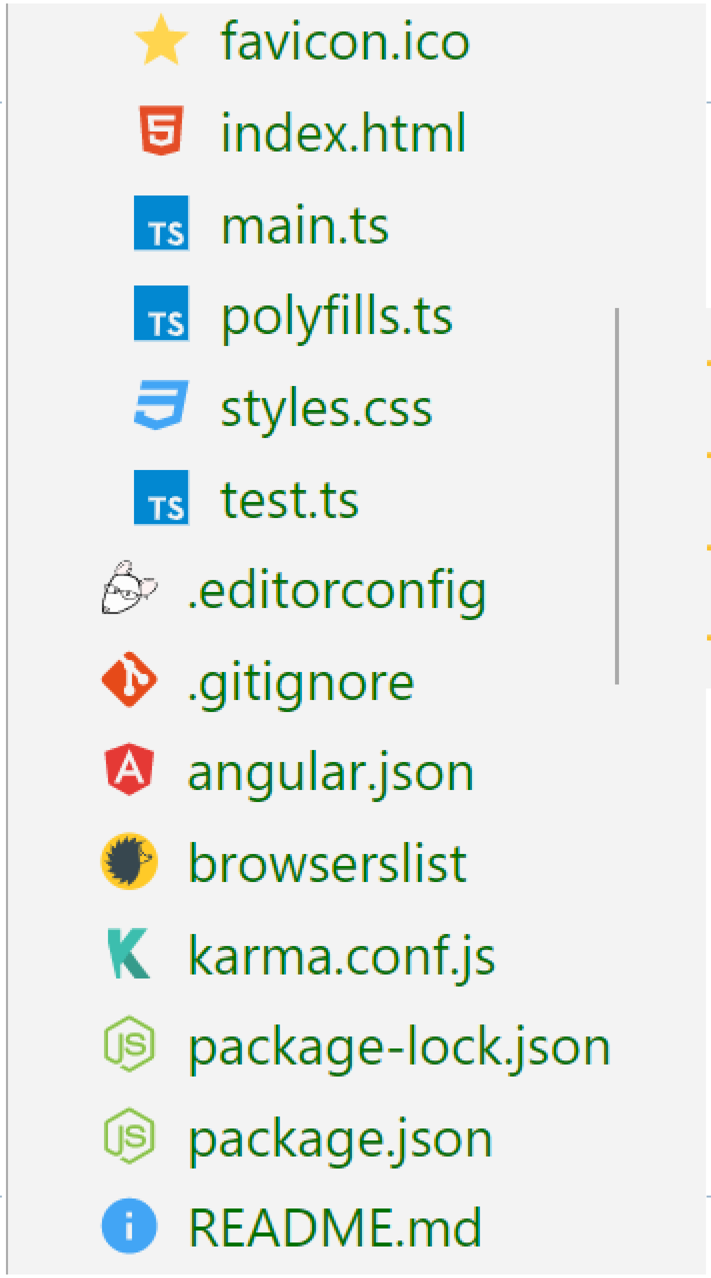
```
cd my-app
ng serve --open
```

- It will start angular server at default port number `4200` and access using `http://localhost:4200`

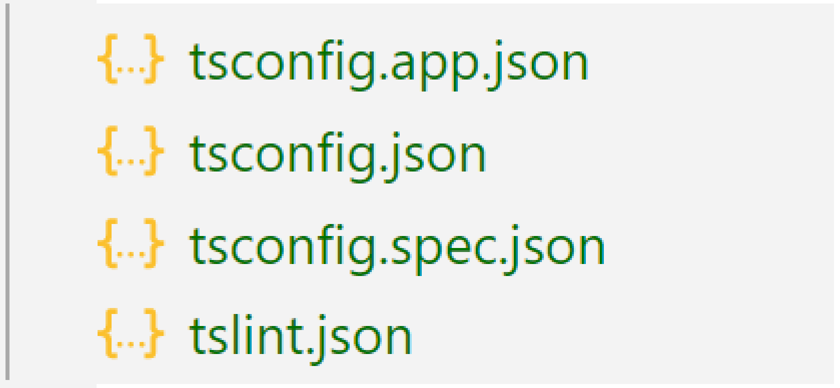
Application Structure



```
v lesson11-angular \ hello-world
  > e2e
  > node_modules
  v src
    v app
        app.component.css
        app.component.html
        app.component.spec.ts
        app.component.ts
        app.module.ts
    > assets
    v environments
        environment.prod.ts
        environment.ts
```



```
      favicon.ico
      index.html
      main.ts
      polyfills.ts
      styles.css
      test.ts
    .editorconfig
    .gitignore
    angular.json
    browserslist
    karma.conf.js
    package-lock.json
    package.json
    README.md
```



```
tsconfig.app.json
tsconfig.json
tsconfig.spec.json
tslint.json
```

Angular Module

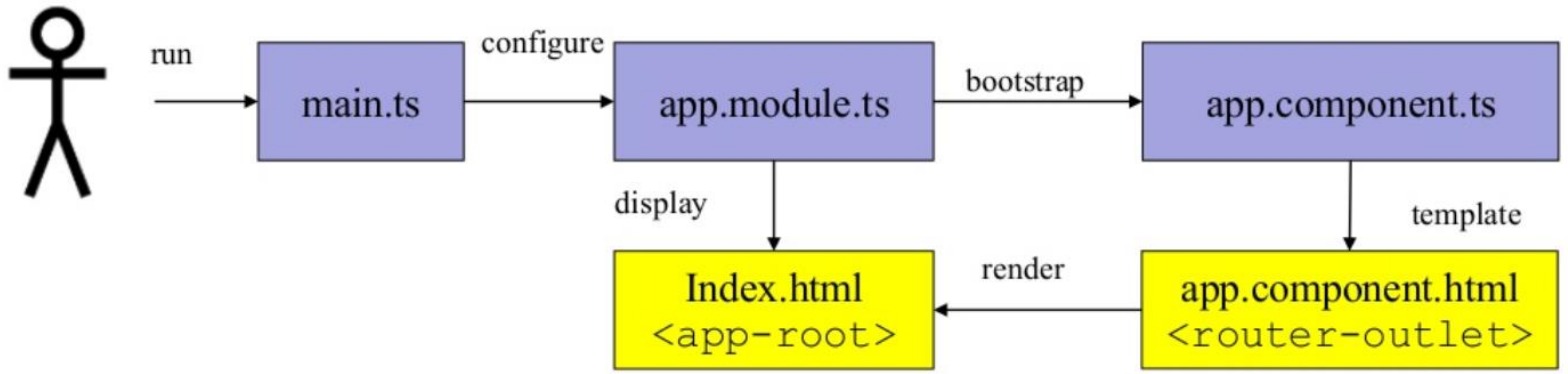
- Angular apps are modular and Angular has its own modularity system called *NgModules*.
- NgModules are containers for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities.

- Module key elements are:
  - `Components` for view and controller
  - `Directives` for data binding
  - `Pipes` for formatting
  - `Services` for reusable operations

- One module can use another module like FormModule, RouteModule, HttpClientModule, etc…

Angular Application Execution Flow

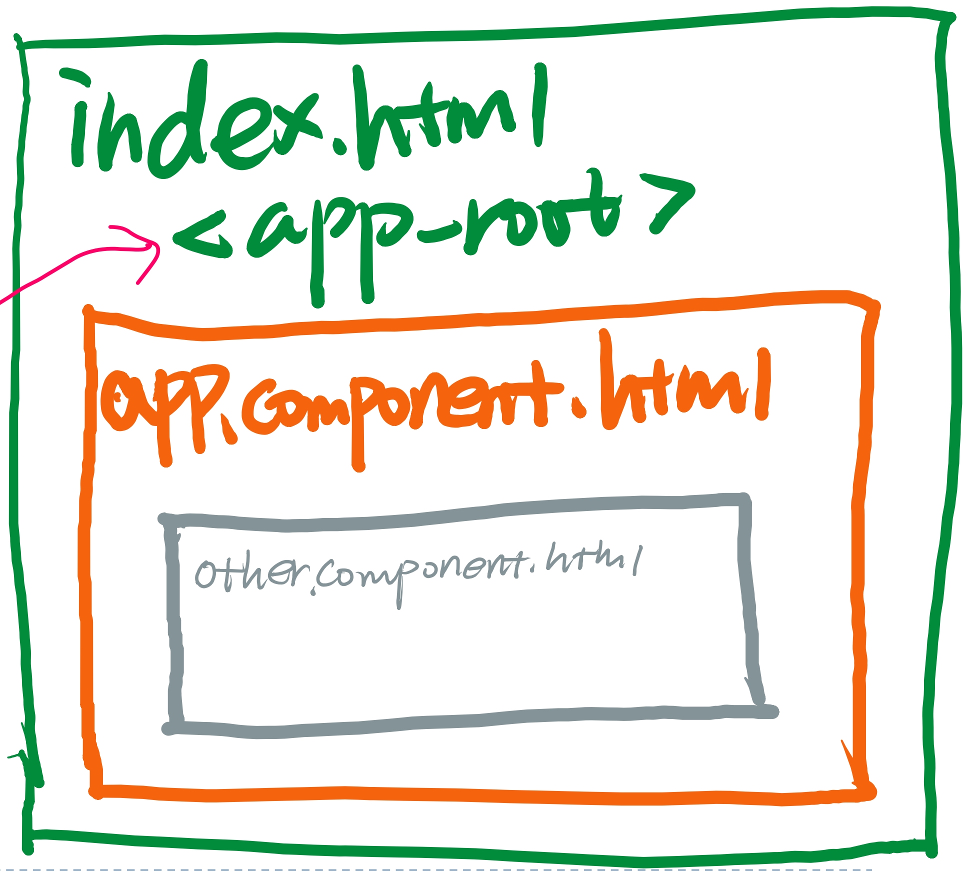
- Application executes `main.ts` file
- File `main.ts` configure app using `app.module.ts` file
- File `app.module.ts` defines application module
- Application displays `index.html` file
- File `index.html` bootstraps root component from `app.component.ts`



index.html

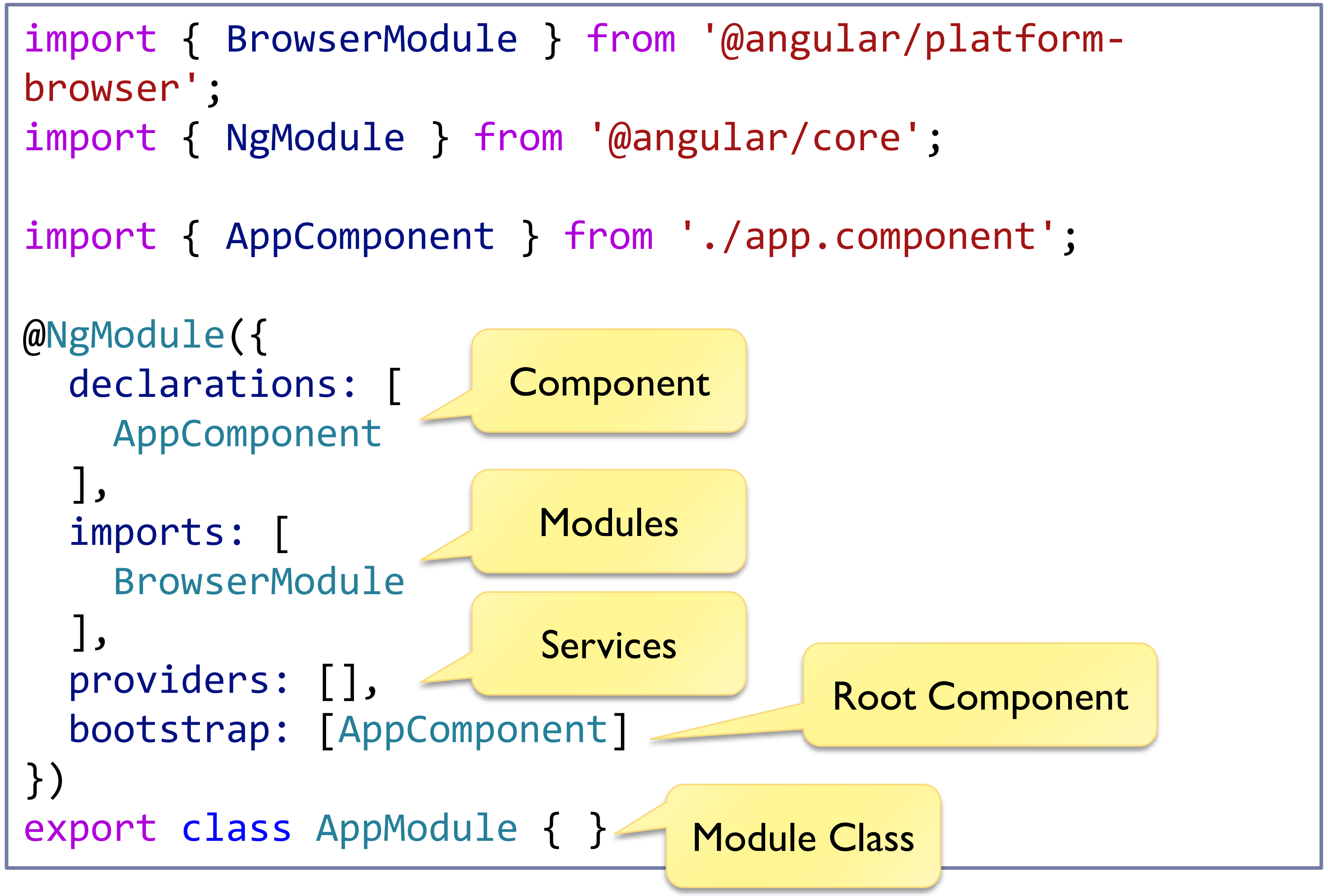
- index.html: The first file which executes alongside main.ts when page loads.

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



app.module.ts

- Define module using @NgModule decorator
- It contains mappings of application elements: component, service, pipe, etc. And other modules like ngRoute and FromModule, etc…



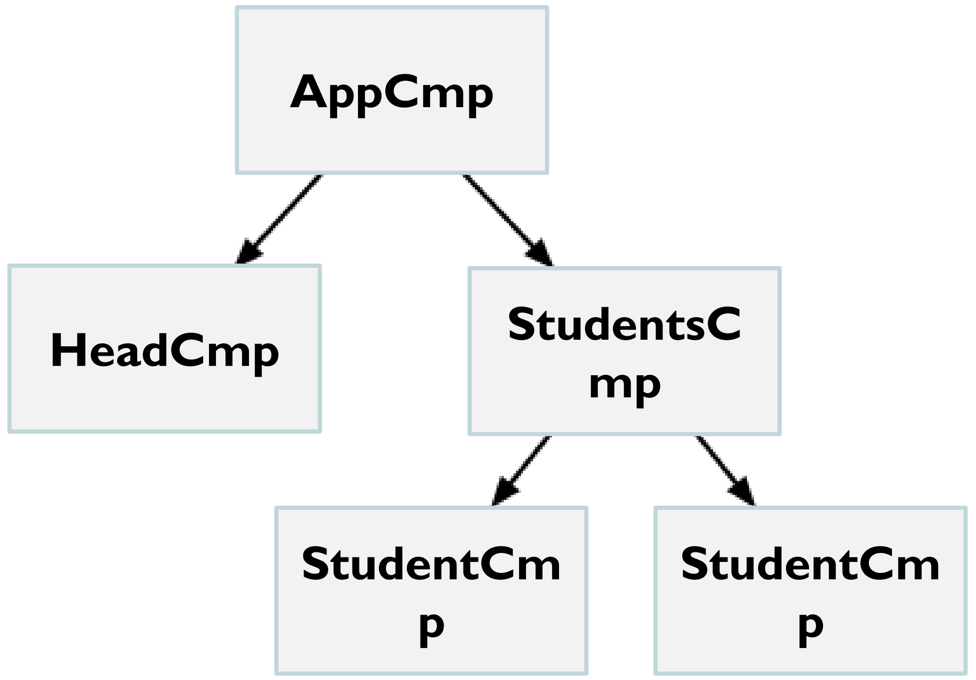
```
import { BrowserModule } from '@angular/platform-
browser';
import { NgModule } from '@angular/core';

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

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

Components

- One component is created for one View page
- To build an Angular application you define a set of components, for every UI element, screen, and route.
- An application will always have root components that contain all other components.



Root Component

- Application has one root component app.component.ts
- Root component is bootstrapped with index.html
- `@Component` decorator is used to define components.

app.component.ts

Import Component

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


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

Metadata

Template

Style

Class & Attributes

app.component.html

```
<h1>{{ title }} app is running!</h1>
```

Display Data and Handling Events

Data Binding

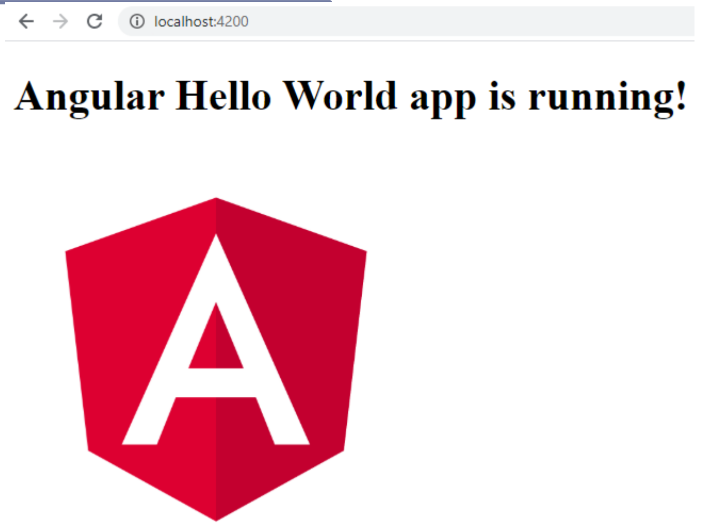
- Data Binding can be one-wary and two-way
- one-way: where data change in controller is reflected in view
- two-way: where data changes are reflected in both directions: controller and view

- Types of bindings are supported by Angular
  - One-way binding
    - Interpolation – `{{attribute-name}}`
    - Property binding – `[attribute-name]`
  - Event bBinding – `(event)`
  - Two-way binding – `[(attribute-name)]`

Interpolation

- One-way data binding id done by directive { { } }, called interpolation.
- Attributes defined in controller can be displayed in html using { { } }.

```
@Component({
  selector: 'app-root',
  // templateUrl: './app.component.html',
  template: `<h1>{{ title }} app is running!</h1>
            <img src="{{itemImageUrl}}">`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular Hello World';
  itemImageUrl = 'https://upload.wikimedia.org/Angular_logo.png';
}
```



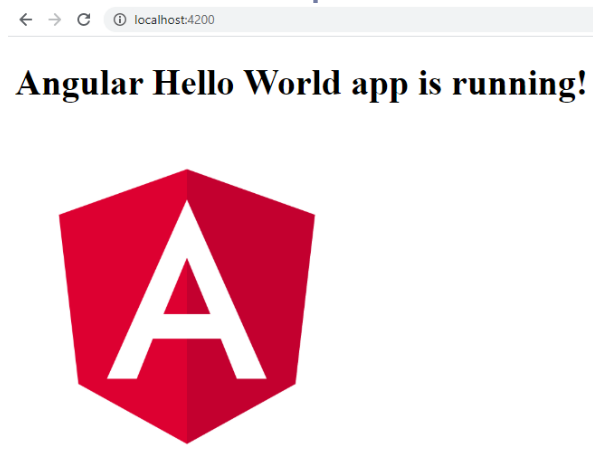
localhost:4200

Angular Hello World app is running!

Property Binding

- Property binding is a one-way mechanism that lets you set the property of a view element. It involves updating the value of a property in the component and binding it to an element in the view template.
- Property binding uses the [] syntax for data binding.
- It targets on Element property, Component property, Directive property

```
@Component({
  selector: 'app-root',
  template: `<h1 [textContent]="title"></h1>
             <img [src]="itemImageUrl">
             `
})
export class AppComponent {
  title = 'Angular Hello World';
  itemImageUrl = 'https://upload.wikimedia.org/Angular_logo.png';
}
```



Angular Hello World app is running!

Attribute Binding

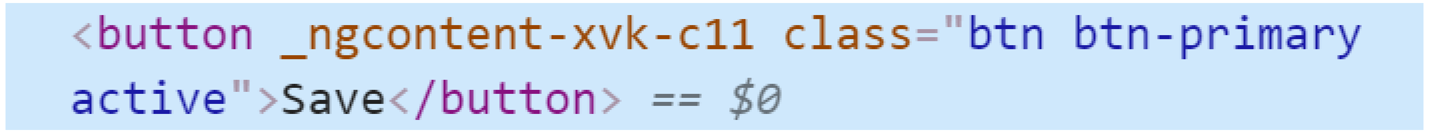
- Attribute binding is useful where we don’t have any property in DOM respected to an HTML element attribute.
- Most of the attributes of HTML elements have the one-to-one mapping with the properties of DOM objects, a few exceptions like `colspan`

```
@Component({
  selector: 'app-root',
  template: `
    <table>
      <tr>
        <td [attr.colspan]="colSpan"></td>
      </tr>
    <table>
    `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  colSpan = 2;
}
```

Class Binding

- Add and remove CSS class names from an element's class attribute with a class binding.
- To create a single class binding, start with the prefix class followed by a dot (.) and the name of the CSS class (for example, [class.foo]="hasFoo"). Angular adds the class when the bound expression is truthy, and it removes the class when the expression is falsy.

```
@Component({
  selector: 'app-root',
  template: `
       <button class="btn btn primary" [class.active]="isActive">Save</button>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  isActive = true;
}
```



```
<button _ngcontent-xvk-c11 class="btn btn-primary
active">Save</button> == $0
```

Style Binding

- Set styles dynamically with a style binding.
- To create a single style binding, start with the prefix style followed by a dot (.) and the name of the CSS style property (for example, [style.width]="width"). The property will be set to the value of the bound expression, which is normally a string. Optionally, you can add a unit extension like em or %, which requires a number type.

```
@Component({
  selector: 'app-root',
  template: `
    <img [style.width.px]="width" src="https://i.ytimg.com/vi/MPV2METPeJU/maxresdefault.jpg">
    <img [style]="styleExpr" src="https://i.ytimg.com/vi/MPV2METPeJU/maxresdefault.jpg">
    <button [style.backgroundColor]="isActive? 'blue': 'red'">Save</button>
  `
})
export class AppComponent {
  width = 100;
  styleExpr = { width: '100px', height: '100px' };
  isActive = false;
}
```

Event Binding

- Event binding allows you to listen for certain events such as keystrokes, mouse movements, clicks, and touches. The binding is from view target to data source
- The binding conveys information about the event. This information can include data values such as an event object, string, or number named `$event`.

```
@Component({
  selector: 'app-root',
  template: `
    <button (click)="onSave($event)">Save</button>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  onSave($event) {
    console.log('Button Clicked.', $event);
  }
}
```

Template Variables

- A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an Angular component or directive or a web component. You can easily access the variable anywhere in the template.

```
@Component({
  selector: 'app-root',
  template: `
  <p><input type="text" #usernameInput></p>
  <p><button (click)="show(usernameInput)">Show</button></p>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  show(username: HTMLInputElement) {
    console.log(username.value);
  }
}
```

Two-way Data Binding

- Two-way binding gives your app a way to share data between a component class and its template.
- Two-way binding does two things:
  1. Sets a specific element property.
  2. Listens for an element change event.
- Angular offers a special two-way data binding syntax for this purpose, [()]. The [()] syntax combines the brackets of property binding, [], with the parentheses of event binding, ().

```
@Component({
  selector: 'app-root',
  template: `
    <input [(ngModel)]="username" (keyup.enter)="onKeyup()">
    <p>Hello {{username}}!</p>
    <input [value]="password" (input)="password = $event.target.value">
    <p>Your password: {{password}}!</p>
  `,
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
  username = "Tina";
  password = "12345";

  onKeyup() {
    console.log(this.username);
  }
}
```

Pipes

Pipes

- A pipe takes in data as input and transforms it to a desired output.
- A way to write display-value transformations that you can declare in your HTML.
- Built-in Pipes: **@angular/common**
  - async
  - currency
  - date
  - decimal
  - json
  - lowercase
  - percent
  - slice
  - uppercase

Examples:

```
<p>{{ myValue | uppercase }}</p>
<p>{{ myDate | date:"MM/dd/yy" }}</p>
<p>{{ myValue | slice:3:7 | uppercase }}</p>
```

Built-in Pipes Example

```
@Component({
  selector: 'app-root',
  template: `
    {{course.title | uppercase | lowercase}}<br/>
    {{course.rating | number}}<br/>
    {{course.students | number: '2.2-2'}}<br/>
    {{course.price | currency: 'CAD'}}<br/>
    {{course.startDate | date: 'short'}}<br/>
  `
})
export class AppComponent {
  course = {
    title: "Angular",
    rating: '9.8989',
    students: '35',
    startDate: new Date(2020, 3, 23),
    price: 800.89
  }
}
```

Custom Pipes

- To create a custom pipe from CLI: `ng g p myPipe` or by adding the @Pipe decorator to a class.
- Custom pipes should be declared in the `declarations[]` array at `app.module.ts`

```
@Component({
  selector: 'app-root',
  template: `{{text | summary: 20}}`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  text: string = 'Angular is an app-
design framework and development platfo
g efficient and sophisticated single-pa
}
```

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

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
    name: 'summary'
})
export class SummaryPipe implements PipeTransform {
    transform(value: string, limit?: number) {
        if (!value) {
            return null;
        } else {
            let actualLimit = (limit) ? limit : 50;
            return value.substr(0, actualLimit) + '...';
        }
    }
}
```