Introduction to Angular
with a simple but complete project

Angular Introduction

- Angular is a framework for building client applications in HTML, CSS and TypeScript (that compiles to JavaScript).

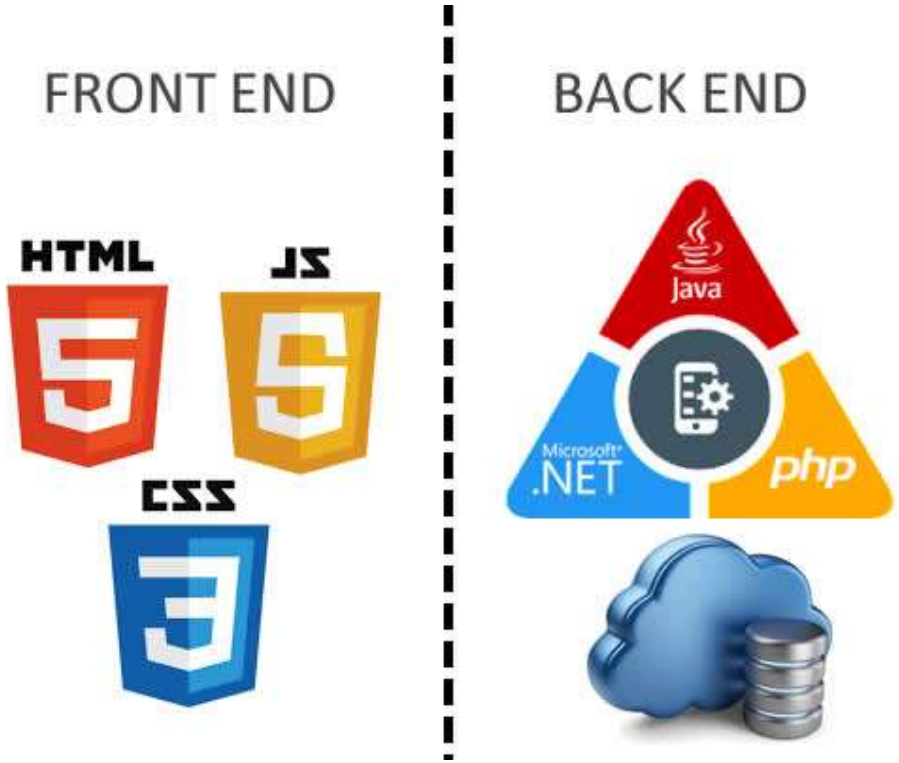
- It has changed the way we develop client side applications, by providing the possibilities to apply the best practices usually applied on server side like modular programming, separation of concerns, testability and many other, on **client side**.

Angular Introduction

- Evolution

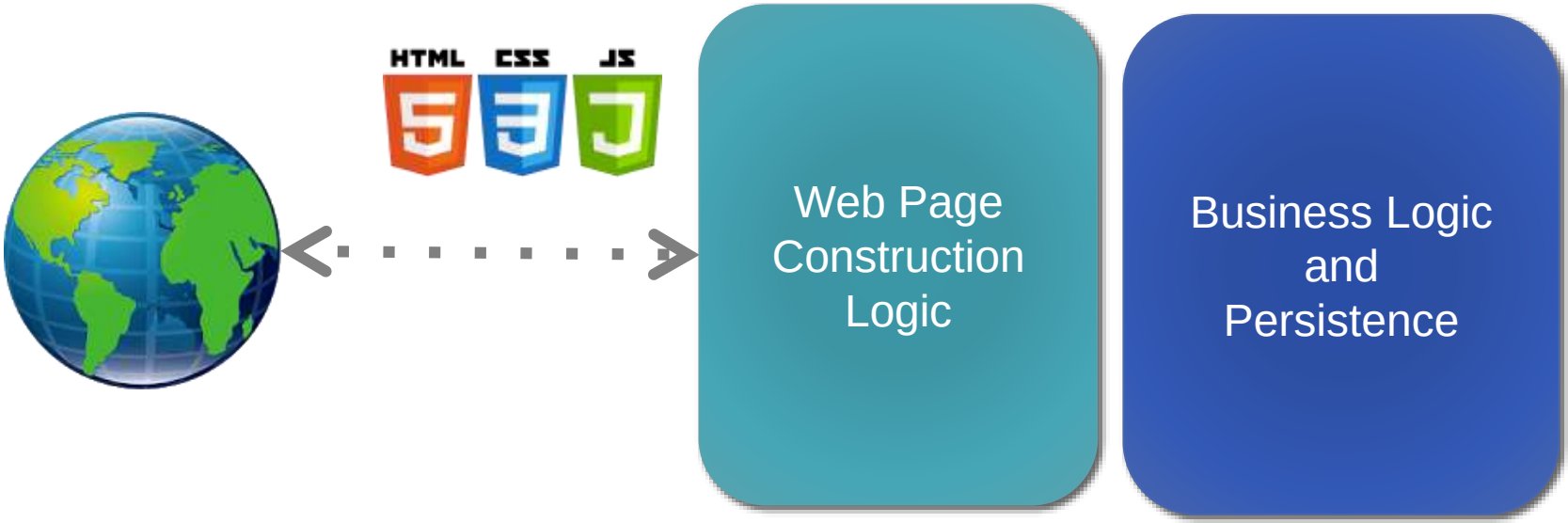
Angular Introduction

- Angular is oriented to develop the front end uncoupled of the back end



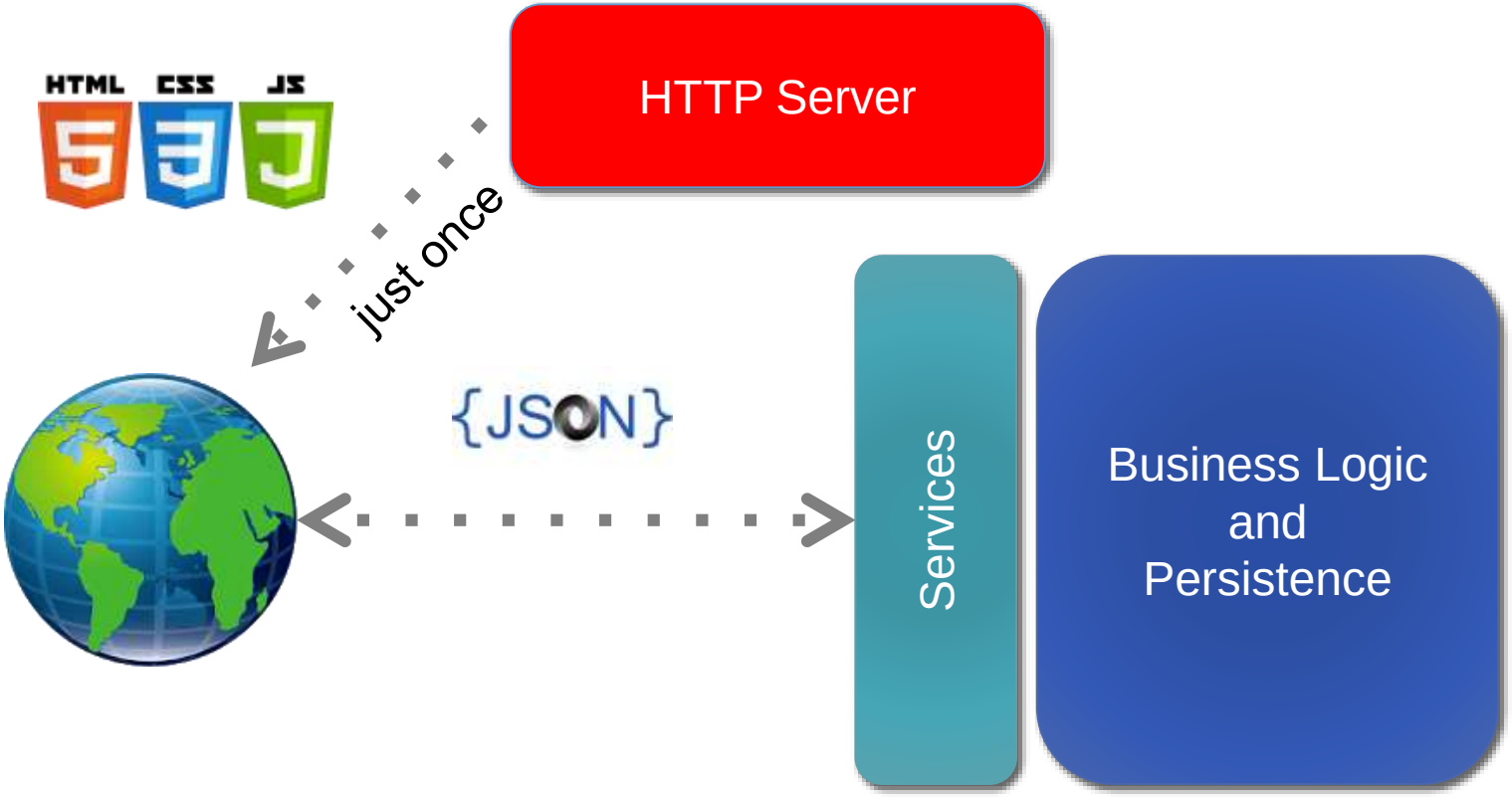
Angular Introduction

- Traditional WEB Architecture



Angular Introduction

- Service Oriented Front End Architecture - SOFEA



Angular Introduction

- SOFEA advantages
  - Scalability (processing, stateless, caching)
  - Interoperability (BaaS – Back-end as a Service, Web and Mobile)
  - Offline Applications
  - Asynchronous development (front-end x back-end)

Angular Introduction

- Angular uses the concept of Single Page Application (SPA)
  - SPA is not an application of a unique html file but a fully contained applications in the browser that do not need to make requests for new pages on the server.
  - Usually SPA makes request just of the data that will be show inside of the pages (accessing back end REST+JSON services)

Angular Introduction

- Single Page Application Advantages:
  - Faster, eliminate the download of html, js and css code in each request
  - Possibility to create off line applications

Angular Install the Environment

TypeScript

Angular Install the Environment

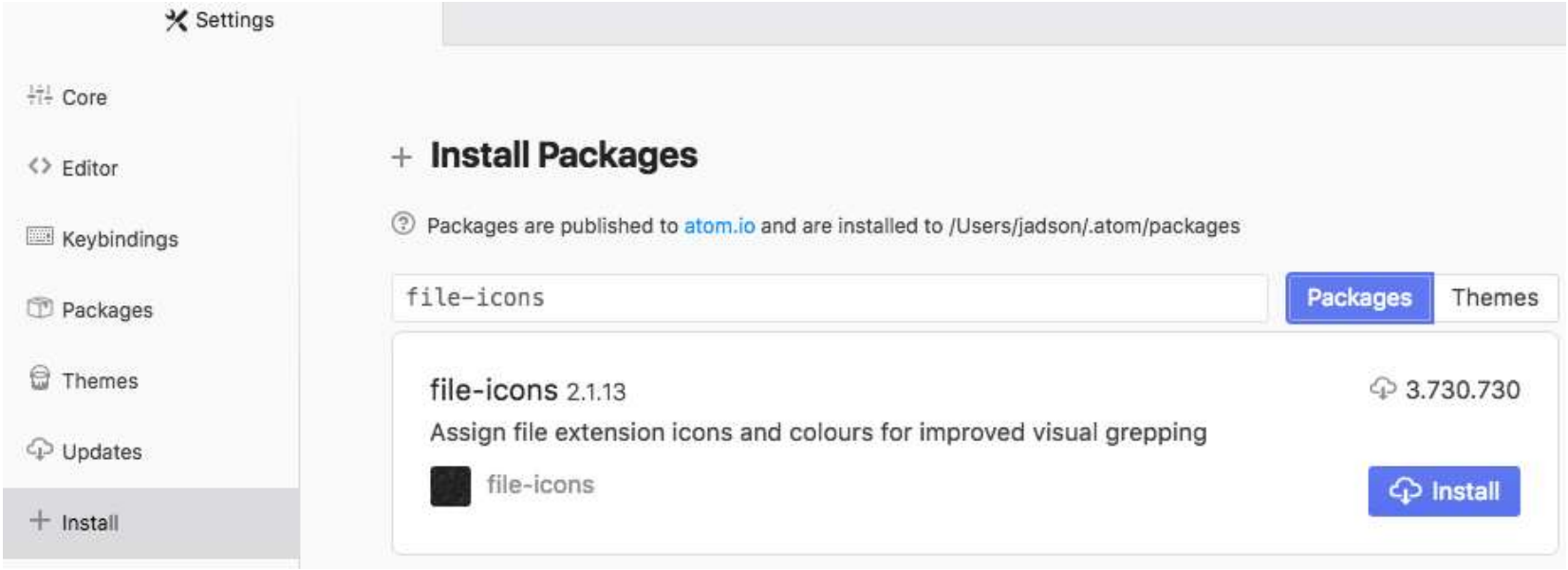
- **ATOM** : Text editor (or any other that you prefer)
- **Node.js + npm**: dependence management (npm ~= gradle/maven in java world)
- **Angular CLI**: Command Line Interfaces for angular
- **TypeScript**: The language of angular 2

Angular Install the Environment

- Download and install Aton (https://atom.io/)

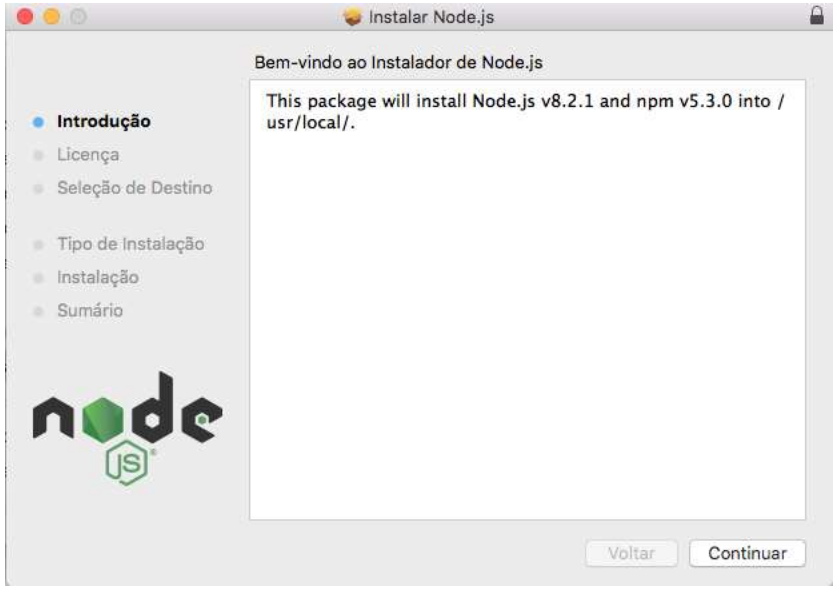
Angular Install the Environment

- Aton Plugins: Aton -> Settings -> Install
  - atom-typescript
  - file-icons
  - angular-2-typeScript-snippets



Angular Install the Environment

- Download and install Node.js (https://nodejs.org) to have access to **npm**



Angular Install the Environment

- After install **npm**, install typescript and angular cli using the npm of node.js

- sudo npm intall –g typescript

- sudo npm intall –g @angular/cli

Angular Install the Environment

- Checking

```
MacBook-Pro-de-Jadson:angular jadson$ ng -v
```

```
@angular/cli: 1.2.7
node: 8.2.1
os: darwin x64
```

Angular Create a new Project

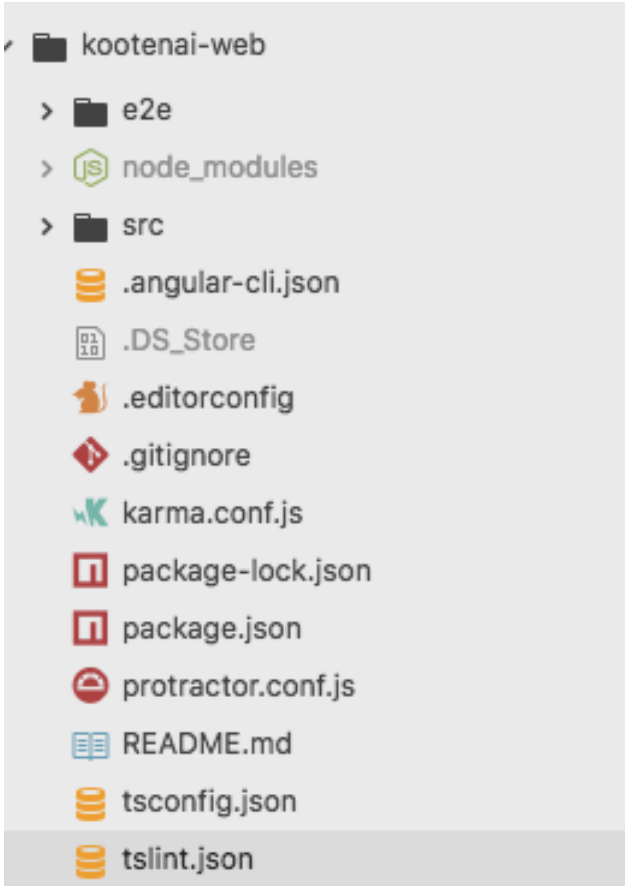
Angular Create a new Project

- Create a new angular project
  - **ng new *project_name***

```
MacBook-Pro-de-Jadson:angular jadson$ ng new kootenai-web
installing ng
  create .editorconfig
  create README.md
  create src/app/app.component.css
  create src/app/app.component.html
  create src/app/app.component.spec.ts
  create src/app/app.component.ts
  create src/app/app.module.ts
  create tsconfig.json
  create tslint.json
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Successfully initialized git.
Project 'kootenai-web' successfully created.
```

Angular Create a new Project

- Open angular project in Atom



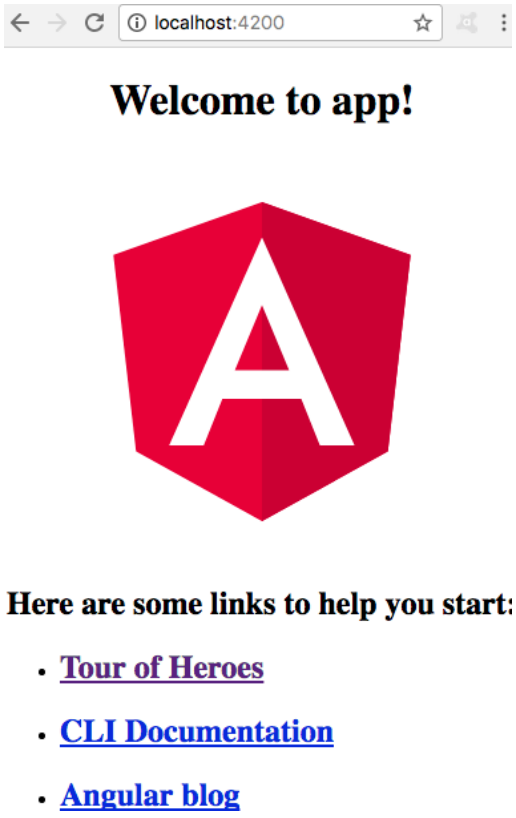
Angular Create the Project

- Running the project
  - **ng server** inside of project folder
  - open the browser on http://localhost:4200

```
MacBook-Pro-de-Jadson:angular jadson$ cd kootenai-web/
MacBook-Pro-de-Jadson:kootenai-web jadson$ ng server
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200 **
Hash: 2ff6e4c8f4ea5837cd76
Time: 15849ms
chunk    {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 191 kB {4} [initial] [rendered]
chunk    {1} main.bundle.js, main.bundle.js.map (main) 5.28 kB {3} [initial] [rendered]
chunk    {2} styles.bundle.js, styles.bundle.js.map (styles) 10.5 kB {4} [initial] [rendered]
chunk    {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.21 MB [initial] [rendered]
chunk    {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
webpack: Compiled successfully.
```

Angular Create a new Project

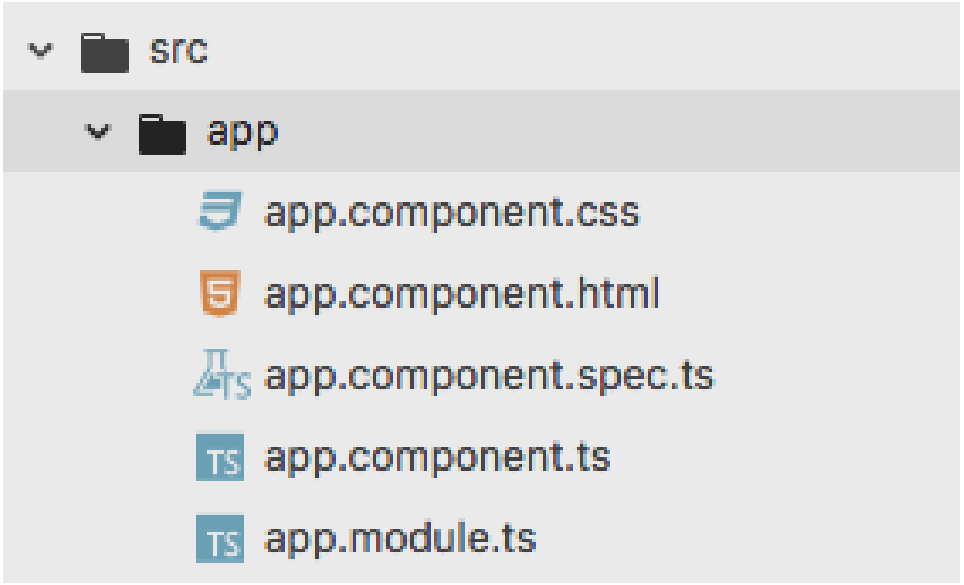
- Default Angular Page



Angular Creating Components

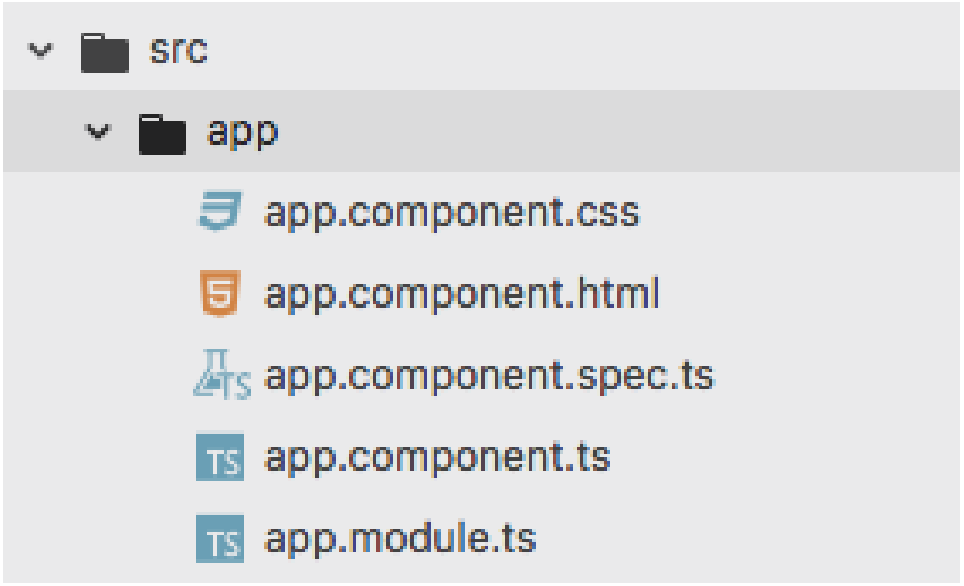
Angular Creating Components

- Angular is based on components.
  - There is already the main component called app.component that shows the “Wellcome to App” page when you access localhost:4200



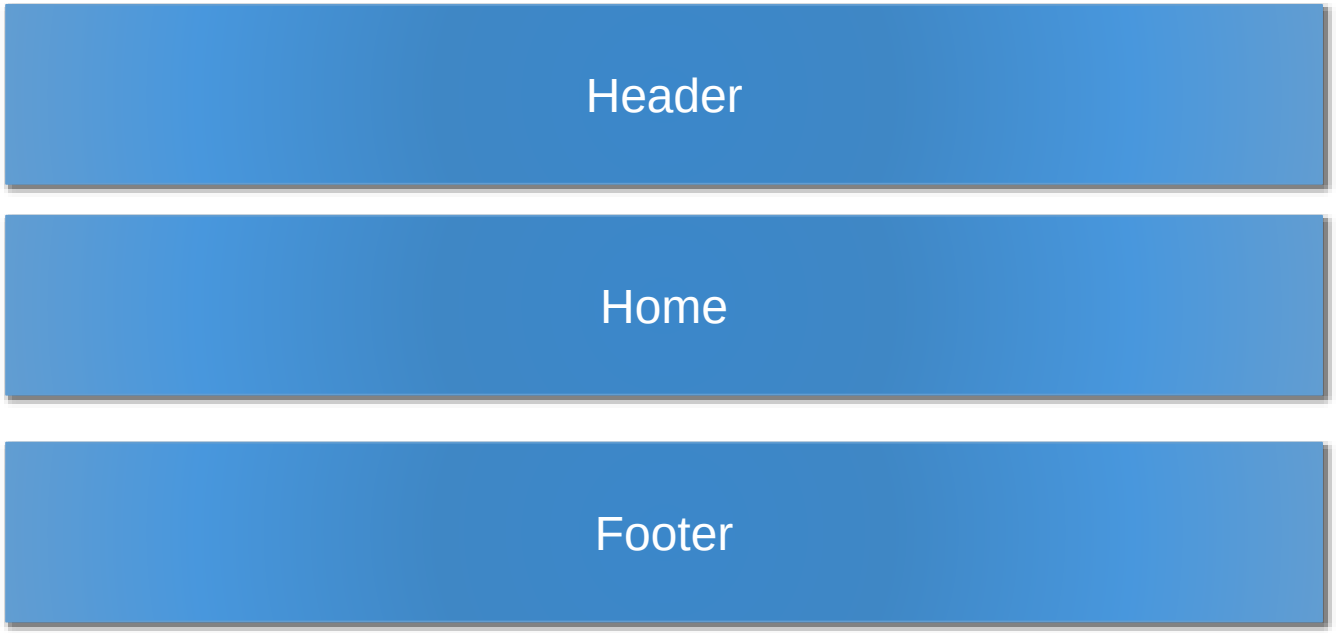
Angular Creating Components

- Angular component have 3 basics parts.
  - name-component.html (the html code of component)
  - name-component.css (css style of component)
  - name-component.ts (the typescritp of component)



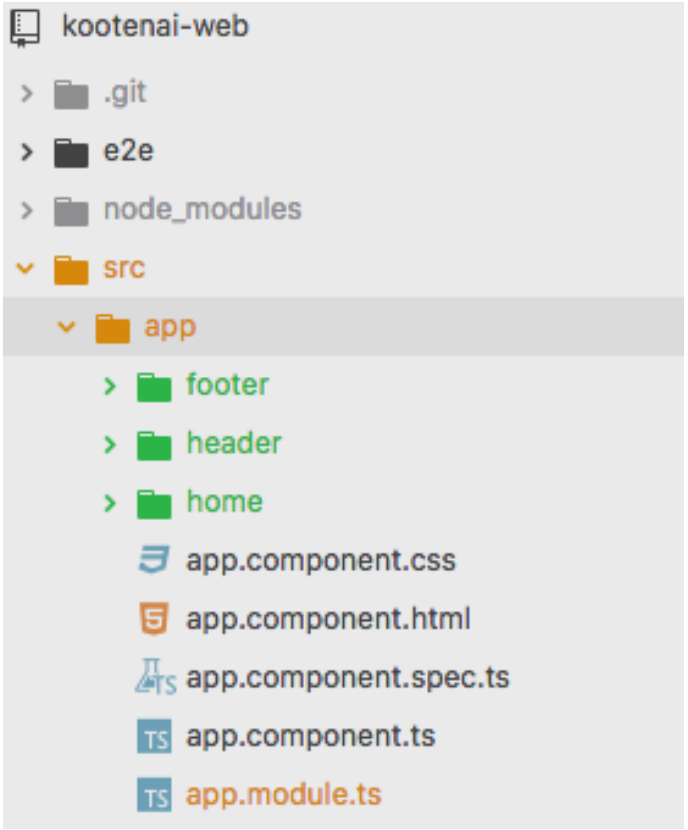
Angular Creating Components

- Our application will have 3 components
  - Let's create then with **ng g c *name*** Angular CLI command



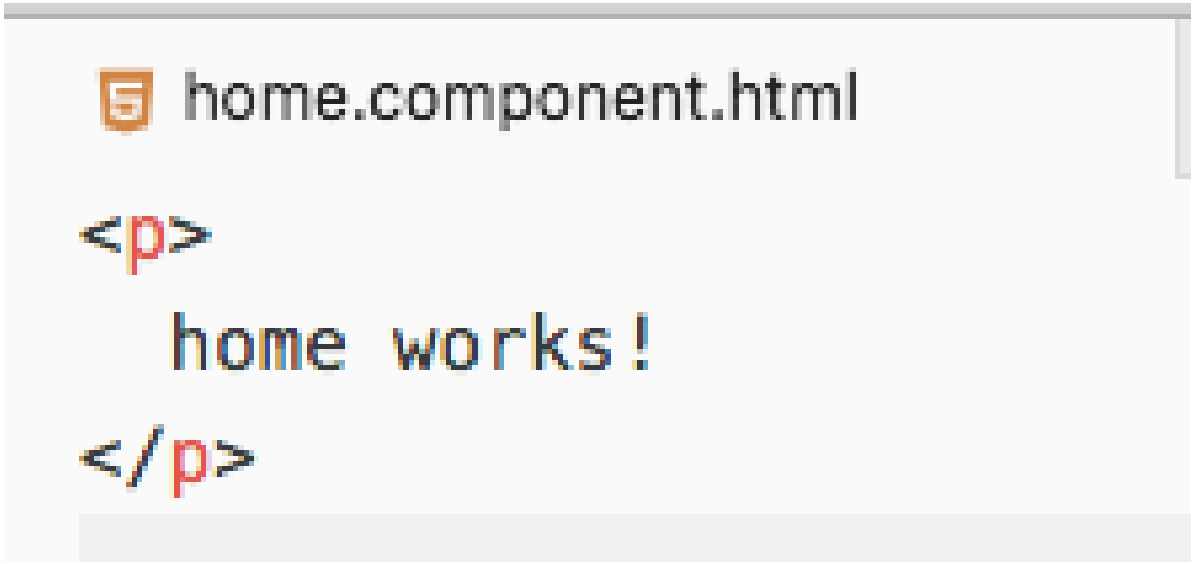
Angular Creating Components

- Create Angular components
  - ng g c header
  - ng g c home
  - ng g c footer



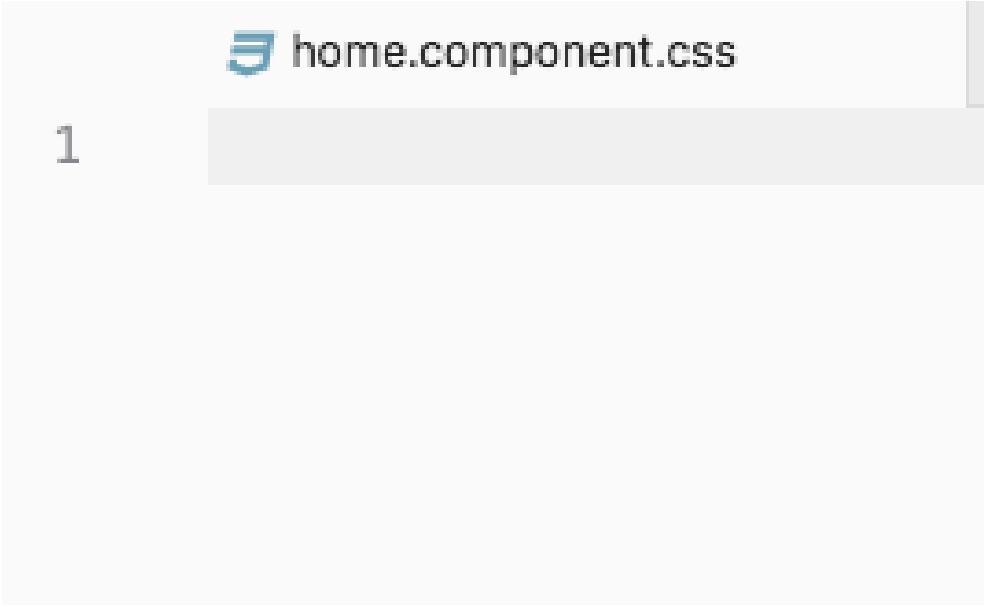
Angular Creating Components

- Each component has a simple html page



Angular Creating Components

- A empty css file



Angular Creating Components

- And a typescript class

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

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Angular Creating Components

- Each component has a **selector** in the typescript class that identify the component

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
```

Angular Creating Components

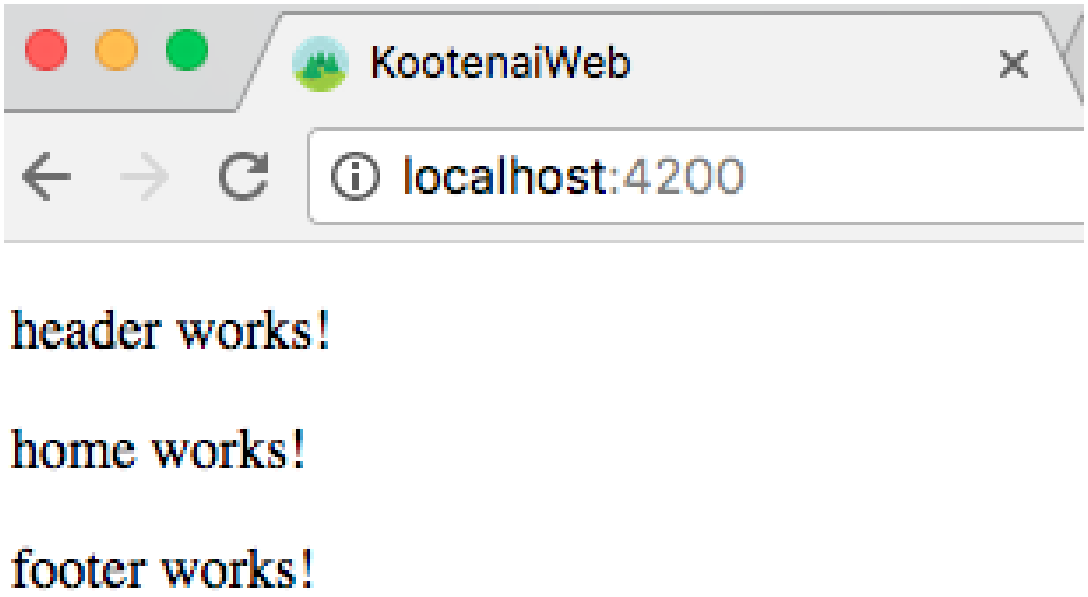
- So, let’s erase the content of the template app.component.html file and put our components selectors in the order of the components will be shown

app.component.html

```
<app-header> </app-header>

<app-home> </app-home>

<app-footer> </app-footer>
```

Angular Creating Components

- ng server to run the development angular server



Angular Project Look and Feel

Angular Project Look and Feel

- Now let's install bootstrap in our project to make view pretty
- To install new angular external modules use npm

npm install bootstrap@3 jquery --save

- This installs Bootstrap and jQuery into the **node_modules** folder within the project directory

Angular Project Look and Feel

- When we are development a web application with bootstrap and jquery we need to include its .css and .js files in our html pages.
- We can do this with angular, but usually angular has a file **.angular-cli.json** where we can include the .css and .javascript code that we will use in our project

Angular Project Look and Feel

- Open the **.angular-cli.json** file and add the css and js files of bootstrap and jQuery inside slyles and scripts arrays.

```
"styles": [
  "styles.css",
  "../node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
  "../node_modules/jquery/dist/jquery.min.js",
  "../node_modules/bootstrap/dist/js/bootstrap.min.js"
],
```

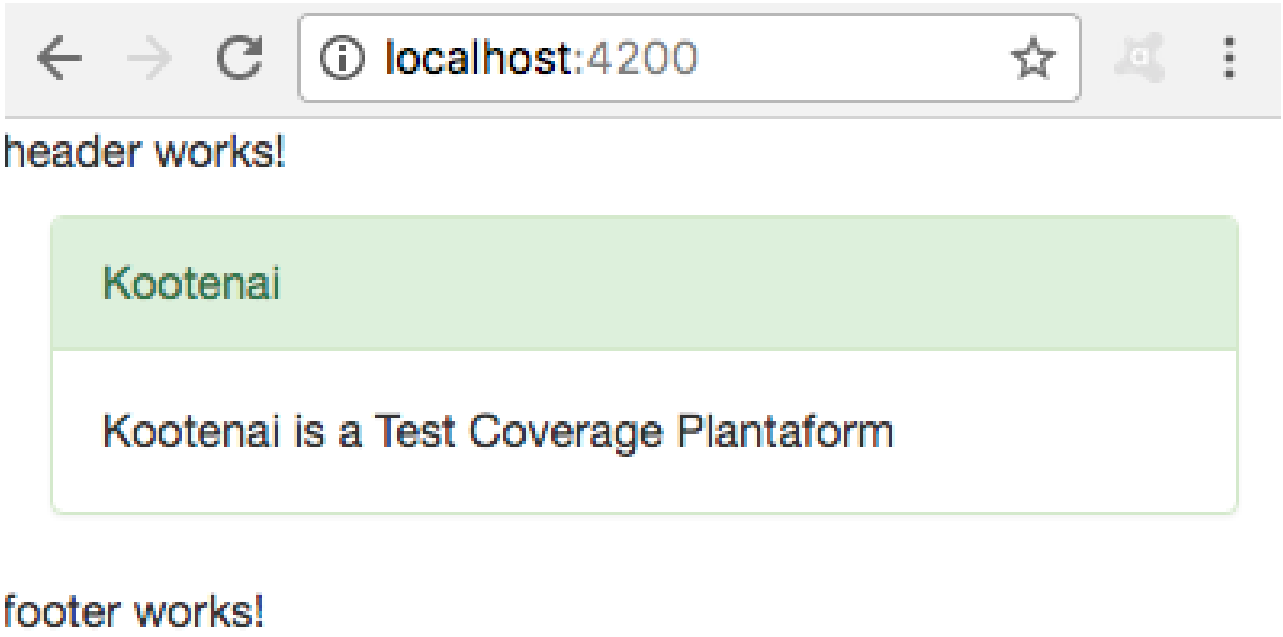
Angular Project Look and Feel

- Now we can open the home component template (html file) and use some bootstrap css class

```
<div class="container">
  <div class="panel panel-success">
    <div class="panel-heading">Kootenai</div>
    <div class="panel-body">Kootenai is a Test Coverage Plantaform</div>
  </div>
<div>
```

Angular Project Look and Feel

- The page will use bootstrap css style:



Angular Project Look and Feel

- We can also use bootstrap templates in our project

- Angular projects have a **assets** folder that we can use to put static files, like images, html templates, etc..

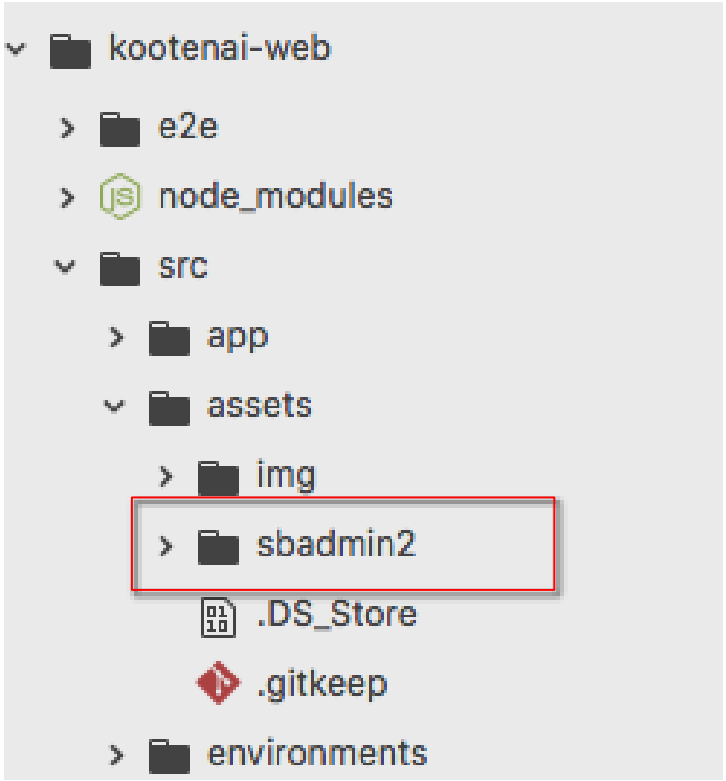
- Let’s use in our project the SB Admin 2 bootstrap theme (https://startbootstrap.com/template-overviews/sb-admin-2/)

Angular Project Look and Feel

- Adding the SB Admin 2 bootstrap theme
  - Download it

Angular Project Look and Feel

- Adding the SB Admin 2 bootstrap theme
  - Copy it content to the **assets** directory



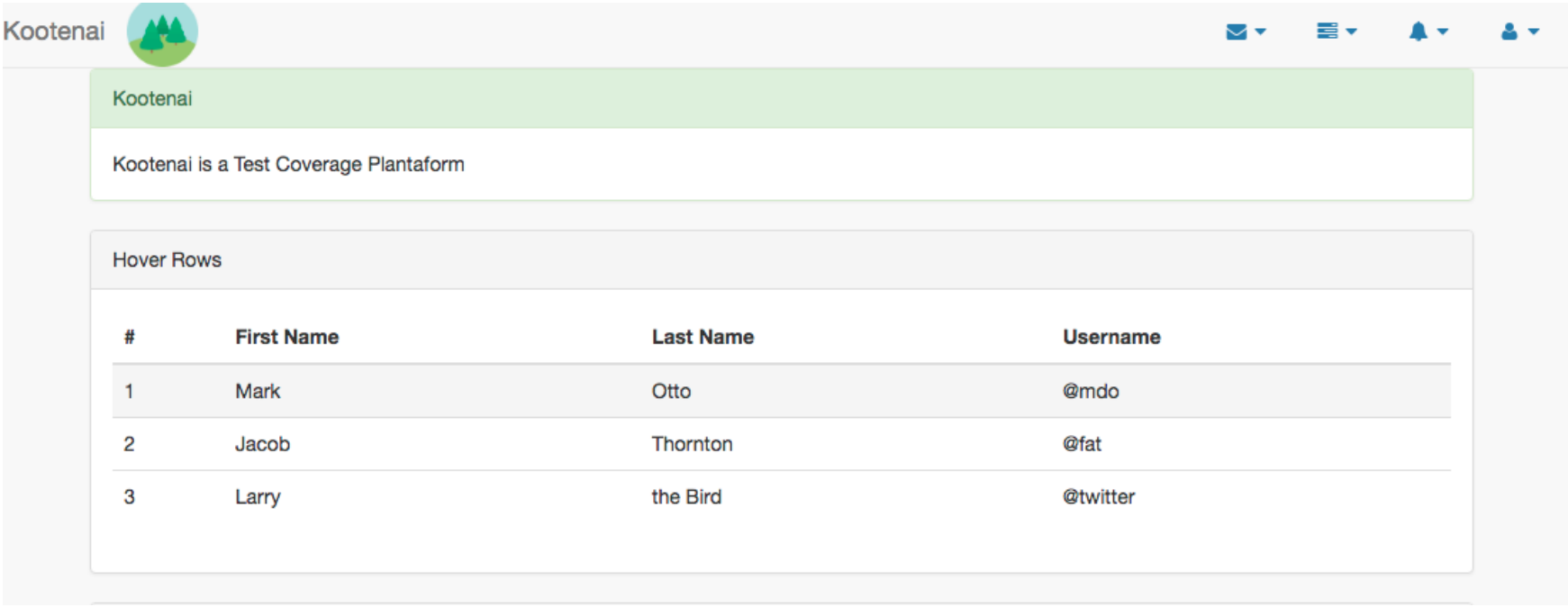
Angular Project Look and Feel

- Adding the SB Admin 2 bootstrap theme
  - Add its js and css files in the **.angular-cli.json** file

```
"styles": [
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",
  "../src/assets/sbadmin2/vendor/metisMenu/metisMenu.min.css",
  "../src/assets/sbadmin2/vendor/font-awesome/css/font-awesome.min.css",
  "../src/assets/sbadmin2/dist/css/sb-admin-2.css",
  "styles.css"
],
"scripts": [
  "../node_modules/jquery/dist/jquery.min.js",
  "../node_modules/bootstrap/dist/js/bootstrap.min.js",
  "../src/assets/sbadmin2/vendor/metisMenu/metisMenu.min.js",
  "../src/assets/sbadmin2/vendor/flot/excanvas.min.js",
  "../src/assets/sbadmin2/vendor/flot/jquery.flot.js",
  "../src/assets/sbadmin2/vendor/flot/jquery.flot.pie.js",
  "../src/assets/sbadmin2/vendor/flot/jquery.flot.resize.js",
  "../src/assets/sbadmin2/vendor/flot/jquery.flot.time.js",
  "../src/assets/sbadmin2/vendor/flot-tooltip/jquery.flot.tooltip.min.js",
  "../src/assets/sbadmin2/data/flot-data.js",
  "../src/assets/sbadmin2/dist/js/sb-admin-2.js"
```

Angular Project Look and Feel

- Adding the SB Admin 2 bootstrap theme
  - Now we can use SB Admin 2 elements in the angular components html files



Angular Data Binding

Angular Data Binding

- Interpolation
  - Allows us to read primitive or object values from component properties in the template (html file)

```
export class NavbarComponent implements OnInit {

  companyName = "MY COMPANY";

  constructor() { }

  ngOnInit() {
  }

}
```

```
<nav class="navbar">
  {{companyName}}
</nav>
```

Angular Data Binding

- Property Binding
  - Angular executes the expression and assigns it to a property of an HTML element, a component, or a directive.

```
export class NavbarComponent implements OnInit {

  companyName = "MY COMPANY";

  isReleased:boolean = true;

  constructor() { }

  ngOnInit() {
  }

}
```

```
<span [hidden]="! isReleased"> Release Information </span>
```

Angular Data Binding

- Event Binding
  - A component method responds to an event raised by an element, component, or directive.

```
// called by the form
addItem(f: FormControl){
   console.log(f.value);
   this.itemService.postItem(f.value).subscribe(
      () => { f.reset(); this.listar();}
   );
}
```

```
<form #frm="ngForm" (ngSubmit)="addItem(frm)">
```

Angular Data Binding

- Two-Way Data Binding
  - Its takes a combination of setting a specific element property and listening for an element change event.

```
export class NavbarComponent implements OnInit {

  companyName = "MY COMPANY";

  isReleased:boolean = true;

  name:string = "";

  constructor() { }

  ngOnInit() {
  }

}
```

Angular Data Binding

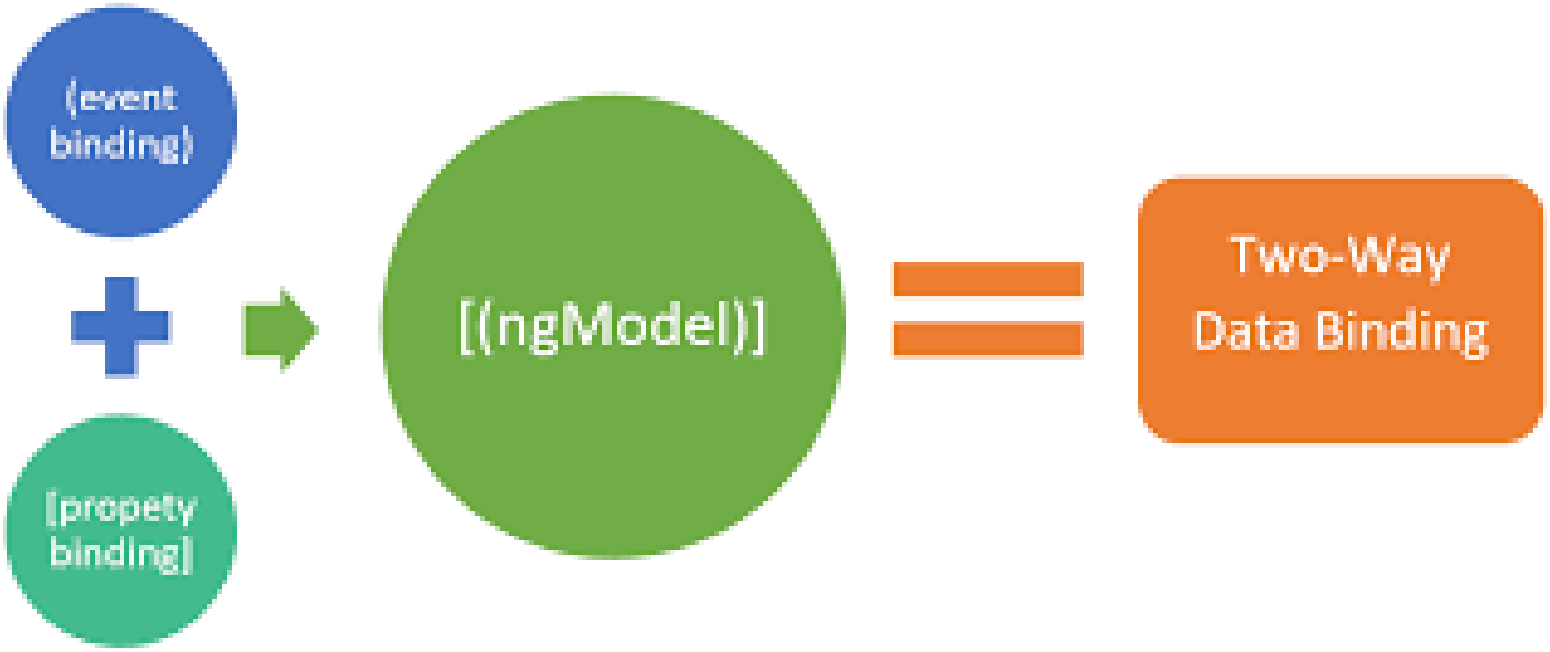
- Two-Way Data Binding
  - You can use a property + event binding

```
<input [value]="name" (input)="name=$event.target.value" >
```

  - Or **[( )]** syntax

```
<input [(ngModel)]="name">
```

Angular Data Binding



Angular Directives

Angular Directives

- Attribute Directives and Structural Directives

  - **Attribute Directives**: changes the appearance or behavior of a DOM element

  - **Structural Directives**: Change the DOM's structure, typically by adding, removing, or manipulating elements.

Angular Directives

- ngIf

```
export class ScenariosComponent implements OnInit {

   showScenario:boolean = true;

  constructor() { }

  ngOnInit() {
  }

}
```

```
<p *ngIf="showScenario">
   List of Scenarios
</p>
```

Angular Directives

- ngFor

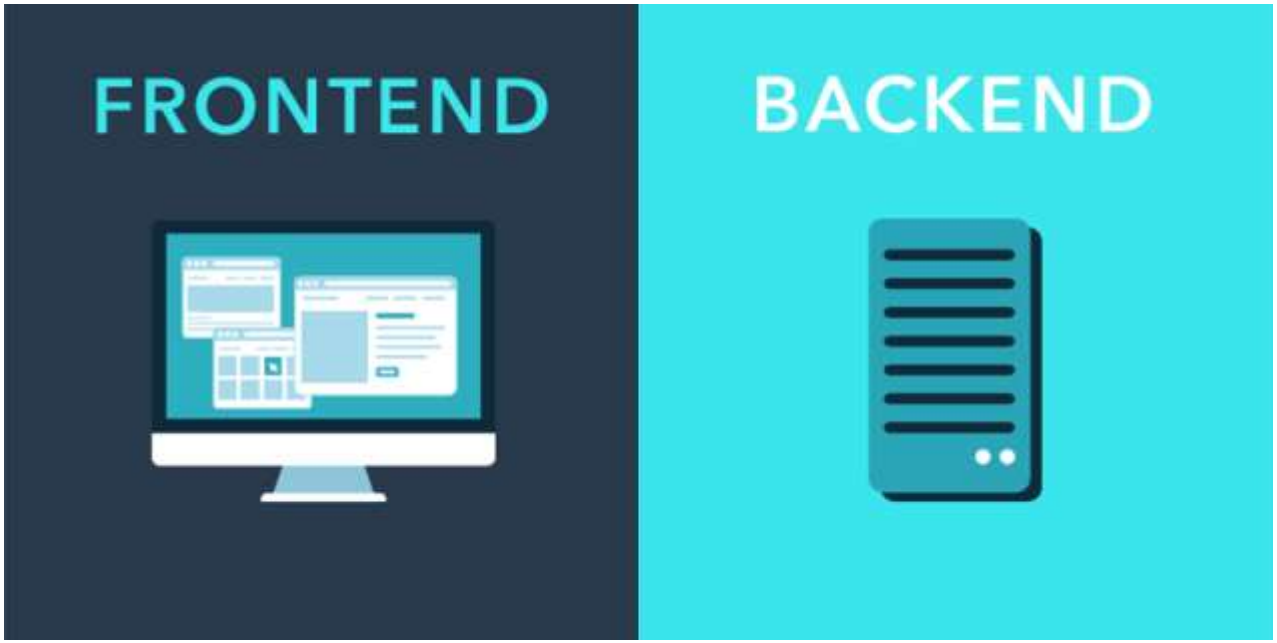
```
export class HomeComponent implements OnInit {

  // o array of scenarios
  scenarios = [];
```

```
<tbody>
    <tr *ngFor="let scenario of scenarios">
        <td></td>
        <td>{{scenario.name}}</td>
        <td>{{scenario.date}}</td>
        <td>{{scenario.requestUrl}}</td>
        <td>{{scenario.requestParameters}}</td>
    </tr>
</tbody>
```

Angular Directives

```
dy class="no-border-x">
  <tr *ngFor="let campo of objetoAcervo.campos">
    <td  *ngIf="campo.template.publico">{{campo.template.descricao}} </td>
    <td  *ngIf="campo.template.publico && campo.discriminatorColumn != 'AGRUPADOR' ">{{campo.valor}}
    <td  *ngIf="campo.template.publico && campo.discriminatorColumn == 'AGRUPADOR' ">
        <table style="width: 100%" class="table table-striped table-hover">
            <thead>
              <tr>
                <th width="30%">SubCampo</th>
                <th width="70%">Informações do SubCampo</th>
              </tr>
            </thead>
          <tbody class="no-border-x">
            <tr *ngFor="let subCampo of campo.subCampos">
               <td *ngIf="subCampo.template.publico">{{subCampo.template.descricao}} </td>
               <td *ngIf="subCampo.template.publico">{{subCampo.valor}}</td>
            </tr>
          </tbody>
        </table>
    </td>
  </tr>
dy>
```

Angular Communicate with back end

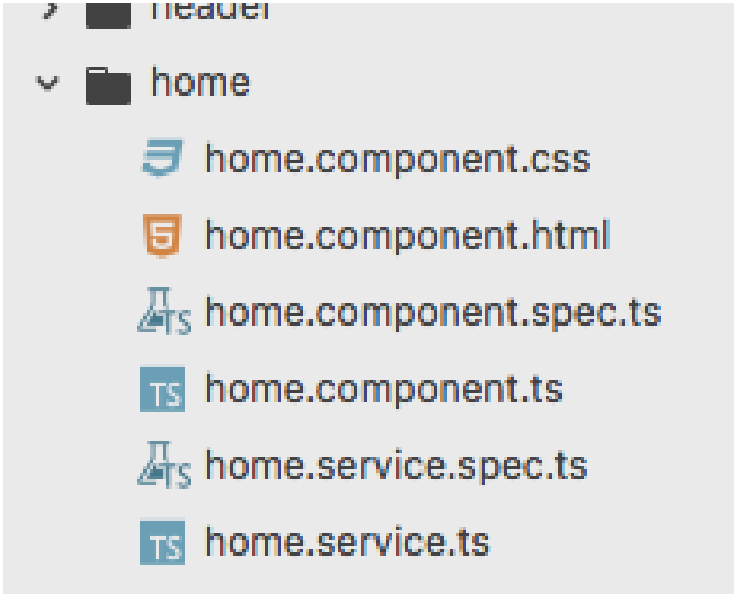


Angular Communicate with back end

- To communication with back end angular uses the concept of “services”

- Creating a new service
  - cd src/app/home
  - **ng g s home**

- This create inside of home
- folder the home.service.ts
- file



Angular Communicate with back end

- In the HomeService
  - import the HttpClient from **“@angular/commum/http”**
  - Inject it by constructor
  - Create a method ***getScearios()***
  - In the method getScenarios() call the ***get*** method passing the URL of the service
    - This call a REST service in the backend that will return a array of scenarios using json.

Angular Communicate with back end

- HomeService

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

import { HttpClient } from '@angular/common/http';

@Injectable()
export class HomeService {

  /** O URL do servico de unidades */
  urlGet = 'http://localhost:8080/kootenai-services/scenarios';

  constructor(private http: HttpClient) { }

  /** return scenarios */
  getScenarios(){
       // return a observable
       // any[] == return an array of any type
       return this.http.get<any[]>(this.urlGet);
  }

}
```

Angular Communicate with back end

- HomeService

```
getScenarios() {
      // return a observable
      // any[] == return an array of any type
      return this.http.get<any[]>(this.urlGet);
}
```

Angular Communicate with back end

- In the HomeComponent
  - import the HttpService from ./home.service
  - Inject it by constructor
  - Create a method ***scenariosList()*** that call the ***getScearios()*** from the service
  - On the ngOnInit() method call the ***scenariosList()***, when the home component is create (the html code is show)

  - ngOnInit() -> scenearios List() -> getScenarios()

Angular Communicate with back end

- HomeComponent

```
import { HomeService } from './home.service';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  // o array of scenarios
  scenarios = [];

  constructor(private homeService: HomeService) { }

  ngOnInit() {
    this.scenariosList();
  }

  /** Chama o servico que retorna as unidades ativas do sistema*/
  scenariosList(){
      // subscribe is a "observable" returned by the service
      // when the service return something, it will be notificated and to assign scenarios array
      this.homeService.getScenarios().subscribe(dados => this.scenarios = dados );
  }
}
```

Angular Communicate with back end

- HomeComponent

```
scenariosList() {
    // subscribe is a "observable" returned by the service
    // when the service return something,
    // it will be notificated and to assign scenarios array
    this.homeService.getScenarios().subscribe(
      dados => this.scenarios = dados
    );
}
```

Angular Communicate with back end

- In app module (app.module.ts file)
  - import the HttpClientModule and the HomeService
  - add HttpClientModule in imports[] arrays and HomeService in providers[] array

Angular Communicate with back end

- In the app module (app.module.ts file)

```
import { HttpClientModule } from '@angular/common/http';

import { HomeService } from './home/home.service';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    HomeComponent,
    FooterComponent
  ],
  imports: [
    HttpClientModule,  // to user services and call the backend
    BrowserModule
  ],
  providers: [HomeService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular Communicate with back end

- In the back end you can use any technology
- We create a RestFul Web Service using Spring that return a list of ScenarioDTO objects

```
@RestController
@CrossOrigin(origins = "http://localhost:4200")
public class ScenarioResource {
```

Angular Communicate with back end

- Back end

```
@GetMapping("/scenario")
public List<ScenarioDTO> getScenarios() {

    List<Scenario> scenarios = scanarioAutoRepository.findAll();

    List<ScenarioDTO> dtos = new ArrayList<>();

    for (Scenario scenario : scenarios) {
        ScenarioDTO dto = new ScenarioDTO();
        dto.setName(scenario.getName());
        dto.setRequestParameters(scenario.getRequestParameters());
        dto.setRequestUrl(scenario.getRequestURL());

        dtos.add(dto);
    }

    return dtos;
}
```

Angular Communicate with back end

- The Scenario[] arrays return by getScenarios() method of HomeService will have fields with the same name of ScenarioDTO return by back end.

```
public class ScenarioDTO {
    String name;
    Date date;
    String requestUrl;
    String requestParameters;
}
```

Angular Communicate with back end

- Now on the front end, you can iterate over the array on the home template (.html file) using **ngFor directive** and access the fields defined on back end.

```
export class HomeComponent implements OnInit {

  // o array of scenarios
  scenarios = [];
```

Angular Communicate with back end

- Now on the front end, you can iterate over the array on the home template (.html file) using **ngFor directive** and access the fields defined on back end.

```
<tr *ngFor="let scenario of scenarios">
    <td></td>
    <td>{{scenario.name}}</td>
    <td>{{scenario.date}}</td>
    <td>{{scenario.requestUrl}}</td>
    <td>{{scenario.requestParameters}}</td>
</tr>
```

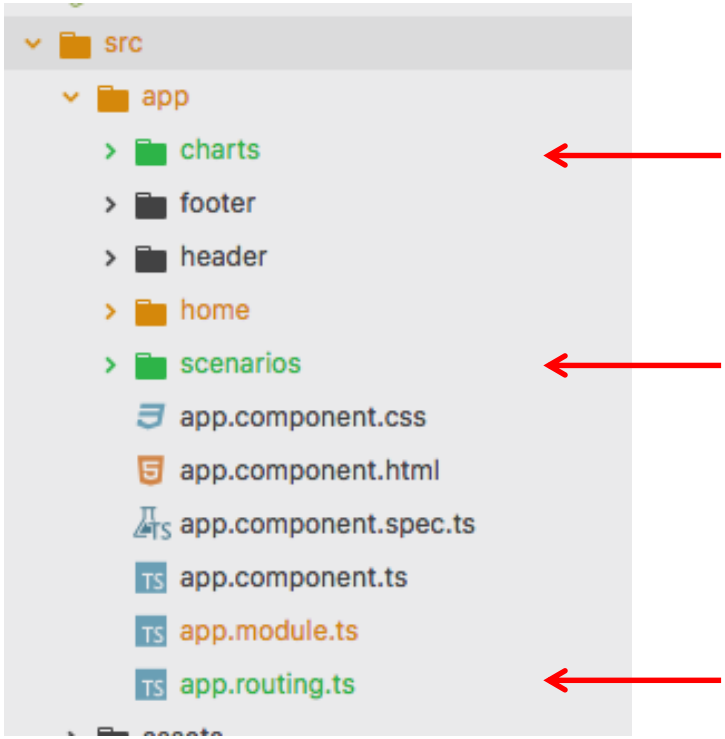
Angular Routes

Angular Routes

- Routes is a functionally that helps your application to become a Single Page Application
- It redirect the user to another component without reload the page or call the back end.

Angular Routes

- We create 2 new components charts and scenarios
- And a new file **app.routing.ts**



Angular Routes

- In the file app.routing.ts we will declare the root routes of our application

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

import { HomeComponent } from './home/home.component';
import { ChartsComponent } from './charts/charts.component';
import { ScenariosComponent } from './scenarios/scenarios.component';

/**
 * Define the routes
 */
const appRoutes: Routes = [
  { path: '',          component: HomeComponent },
  { path: 'scenarios', component: ScenariosComponent },
  { path: 'charts',    component: ChartsComponent },
];

/**
 * Define a variable of the root routes for Application
 */
export const routing: ModuleWithProviders
          = RouterModule.forRoot(appRoutes);
```

Angular Routes

- Declare a appRoutes variable of the type Routes that is a array with two fields: the path and the component
- When access one path the application will redirect for the component

```
/**
 * Define the routes
 */
const appRoutes: Routes = [
  { path: '',          component: HomeComponent },
  { path: 'scenarios', component: ScenariosComponent },
  { path: 'charts',    component: ChartsComponent },
];
```

Angular Routes

- Declare a const with the routes for root routes

```
/**
 * Define a variable of the root routes for Application
 */
export const routing: ModuleWithProviders
            = RouterModule.forRoot(appRoutes);
```

Angular Routes

- And import this const in the app.module.ts

```
import { routing } from './app.routing';

@NgModule({

  imports [
    HttpClientModule,  // to user servic
    BrowserModule,
    routing
  ],
```

Angular Routes

- Now we have to indicate where the html code of component will be drawn in our application
- We will indicate this with **router-outlet** tag.
- We put this tag on the app.component.html

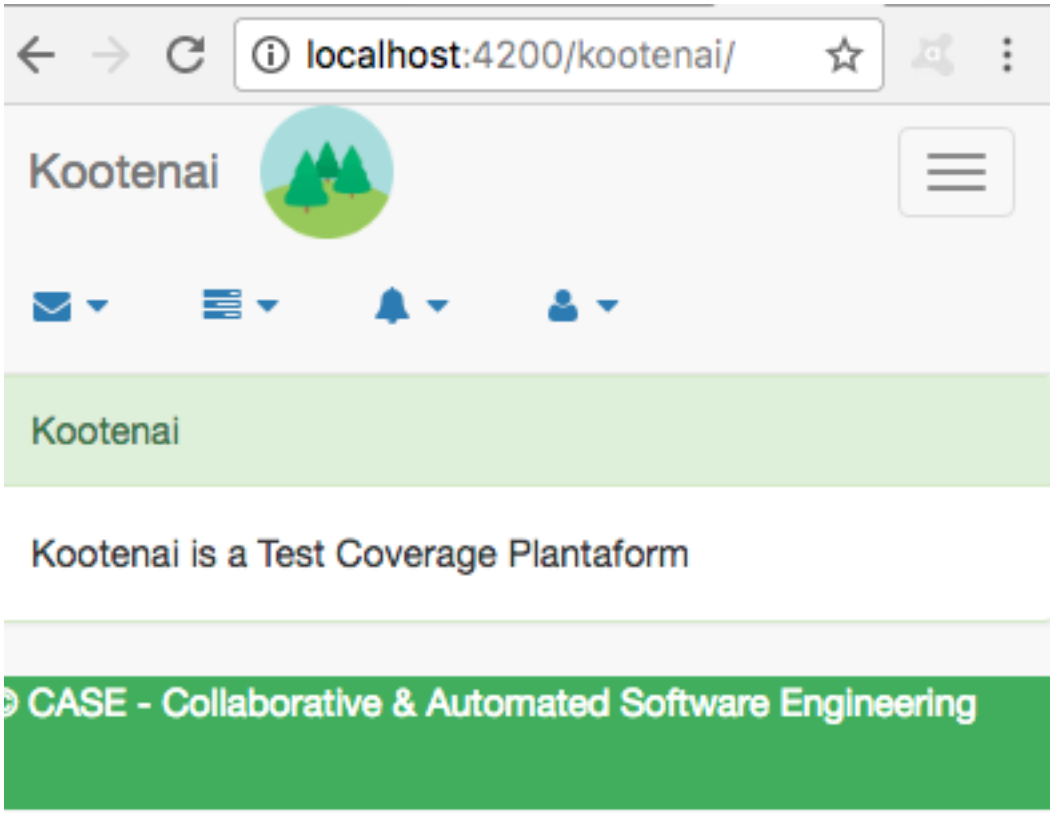
```
<app-header> </app-header>

<div class="container">
    <router-outlet>   </router-outlet>
</div>

<app-footer> </app-footer>
```

Angular Routes

- In the Home component we will let just the common code.
- When the user access the path **“/scenarios**”, the code of ScenariosComponent will be rendered in app.component.html
- When the user access the path **“/charts”**, the code of ChartsComponent will be rendered in app.component.html.

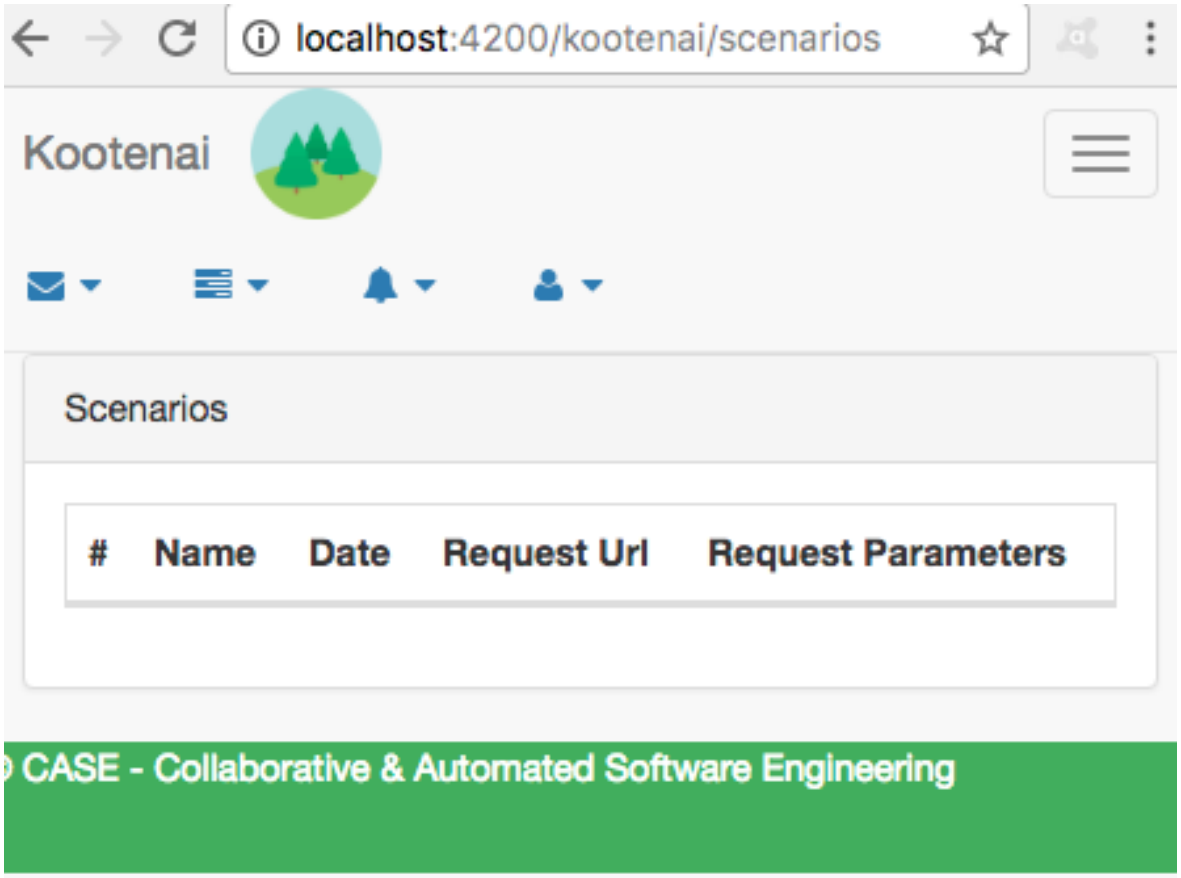
Angular Routes

- Accessing the path “”



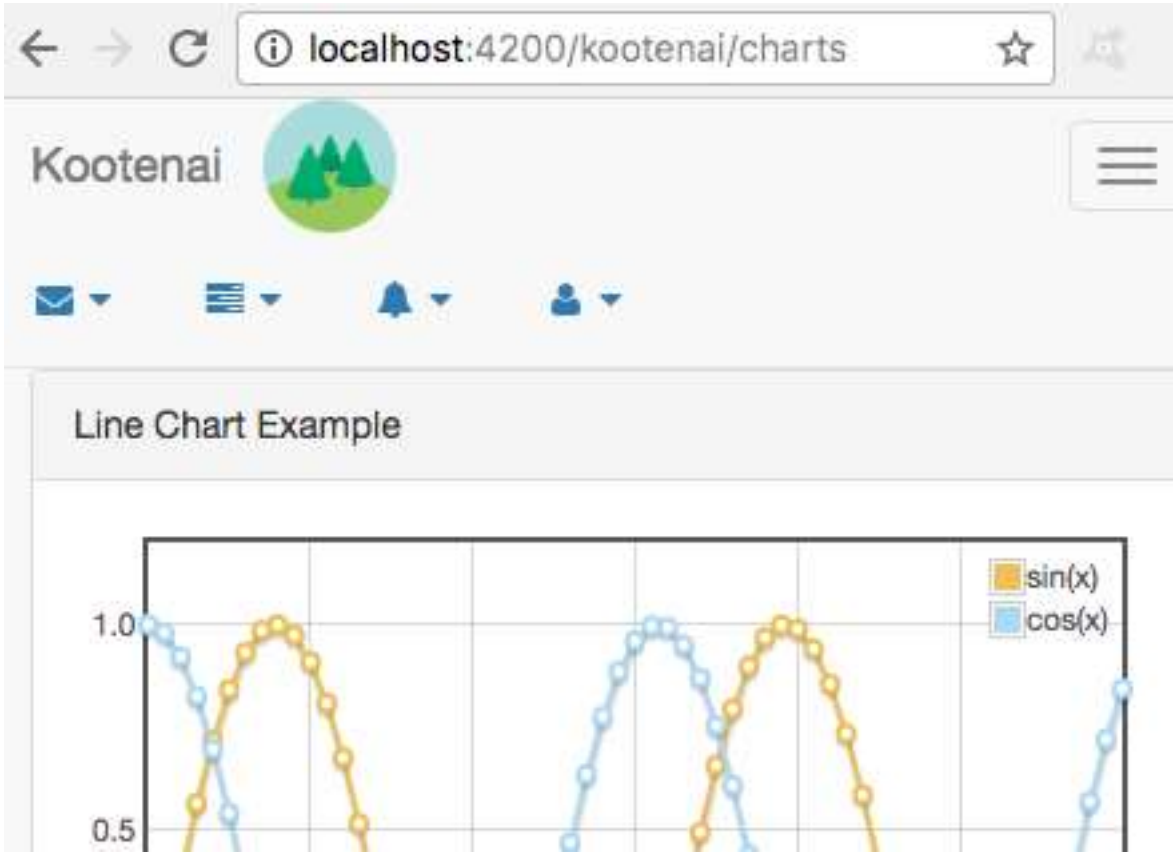
Angular Routes

- Accessing the path **“/scenarios”**



Angular Routes

- Accessing the path **“/charts”**



Angular Routes

- We can redirect from a link without reload the page using the directive **routerLink**

```
<li>
  <a routerLink="/charts"><i class="fa fa-area-chart fa-fw"></i> Charts</a>
</li>
```

Angular Versioning the Project

Angular Versioning the Project

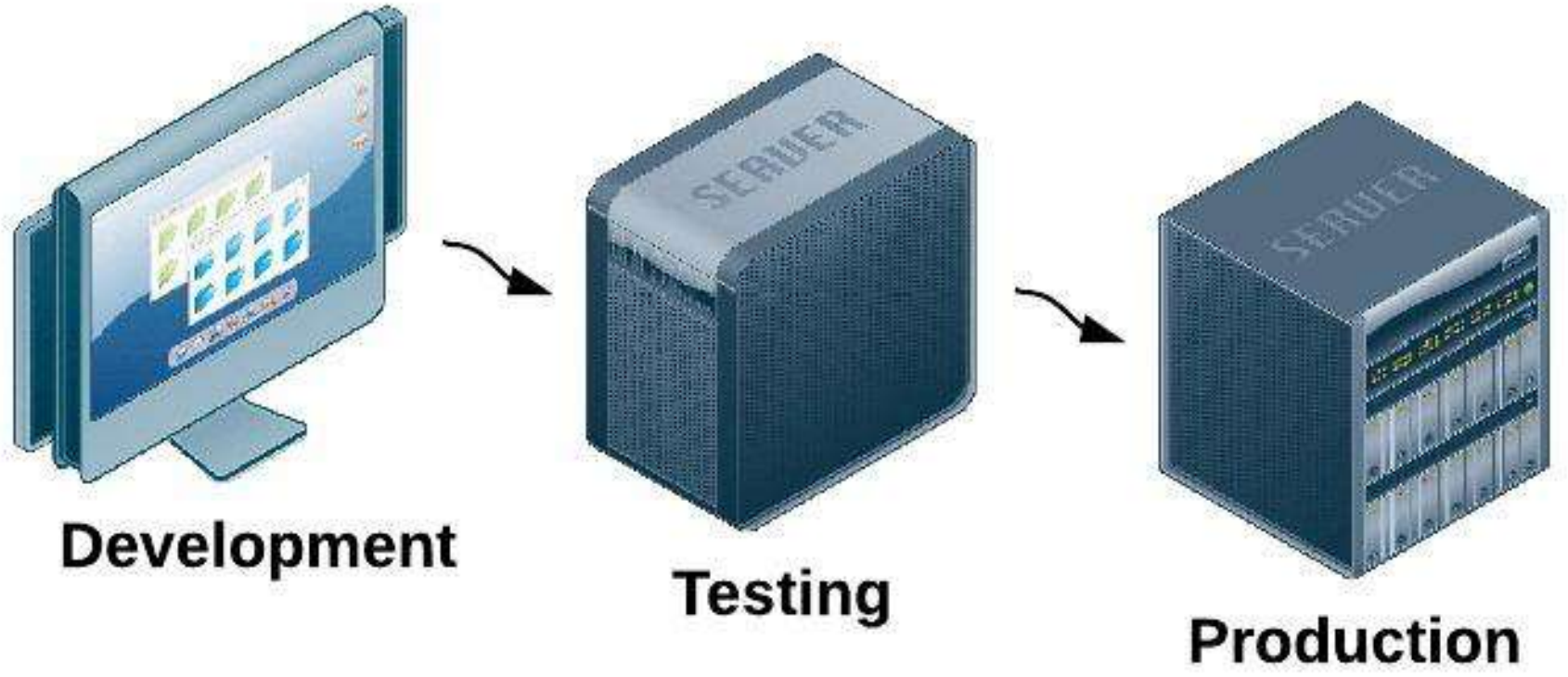
- Angular/cli automatically create a .gitignore file that ignore the **node_modules** directory

```
# See http://help.github.com/ignore-files/

# compiled output
/dist
/tmp
/out-tsc

# dependencies
/node_modules
```

Angular Versioning the Project

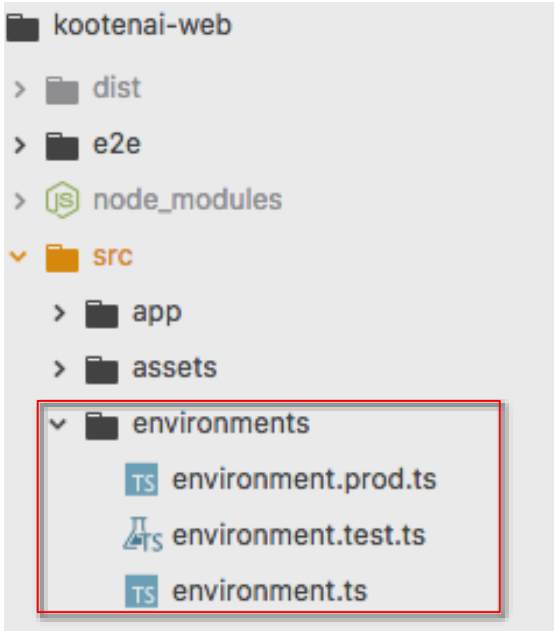
- This happens because this directory contains all dependence of project and is very big.
- When you clone a angular project (that should not contains the node_modules), you can restore it with the command **npm install**.
    - git clone url_to_my_project
    - cd project_directory
    - npm install
    - ng server

Angular Environment Variables



Angular Environment Variables

- You can manage different environments
- Angular create under src directory, a directory named environments, where you can configure global constants



Angular Environment Variables

- Define the environments that you will have in environments array in **.angular-cli.json** file:

```
"environments": {
  "dev": "environments/environment.ts",
  "test": "environments/environment.test.ts",
  "prod": "environments/environment.prod.ts"
}
```

Angular Environment Variables

- environment.ts is default environment
- You can specify the environment on the moment of the build
  - **ng build --env=test**
- This is very useful to define the api url.

```
export const environment = {
  production: false,
  apiUrl: 'http://localhost:8080'
};
```

Angular Environment Variables

- You can import env file in components, services, etc…, like this:

```
import { environment } from '../../environments/environment';

@Injectable()
export class HomeService {

  /** The URL of the scenerios service */
  urlGet = environment.apiUrl+'/scenario';

  constructor(private http: HttpClient) { }
```

Angular Build to Production

Angular Build to Production

- Edit the index.html to set the <base href> appropriately with the **context** of the application, ending with “/”.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Kootenai</title>
  <base href="/kootenai/">

  <meta name="viewport" content="width=device-
  <link rel="icon" type="image/x-icon" href="
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Angular Build to Production

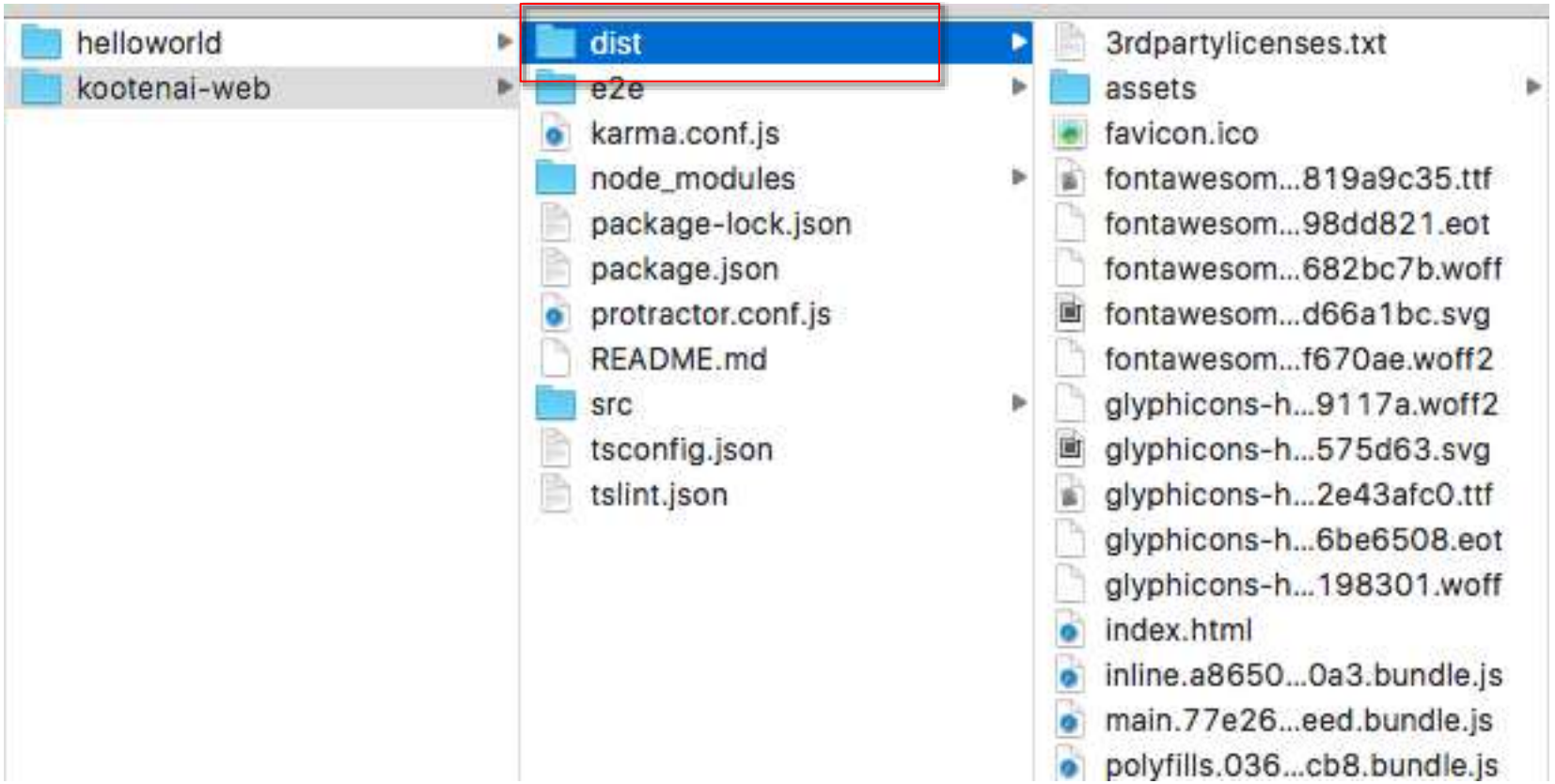
- To build the project to production, we use this command:

ng build --prod --env=prod

  - The **prod** option will minify all files and do another cool things to format the files do production.
  - The **env** option will build the correct environment file to use

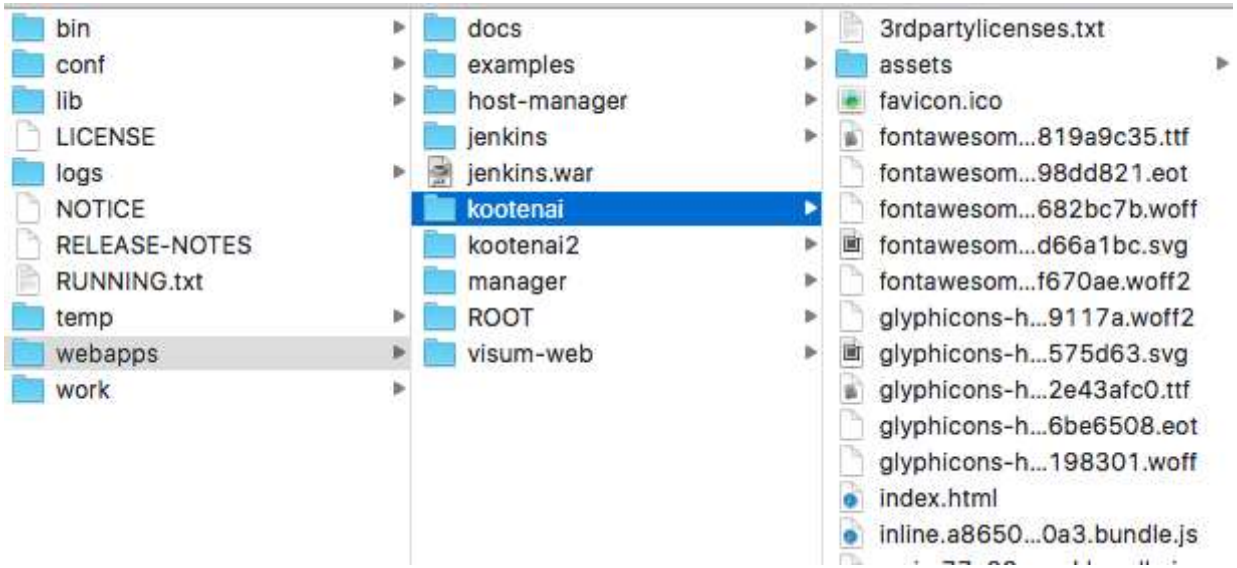
Angular Build to Production

- The build will generated a **dist** directory

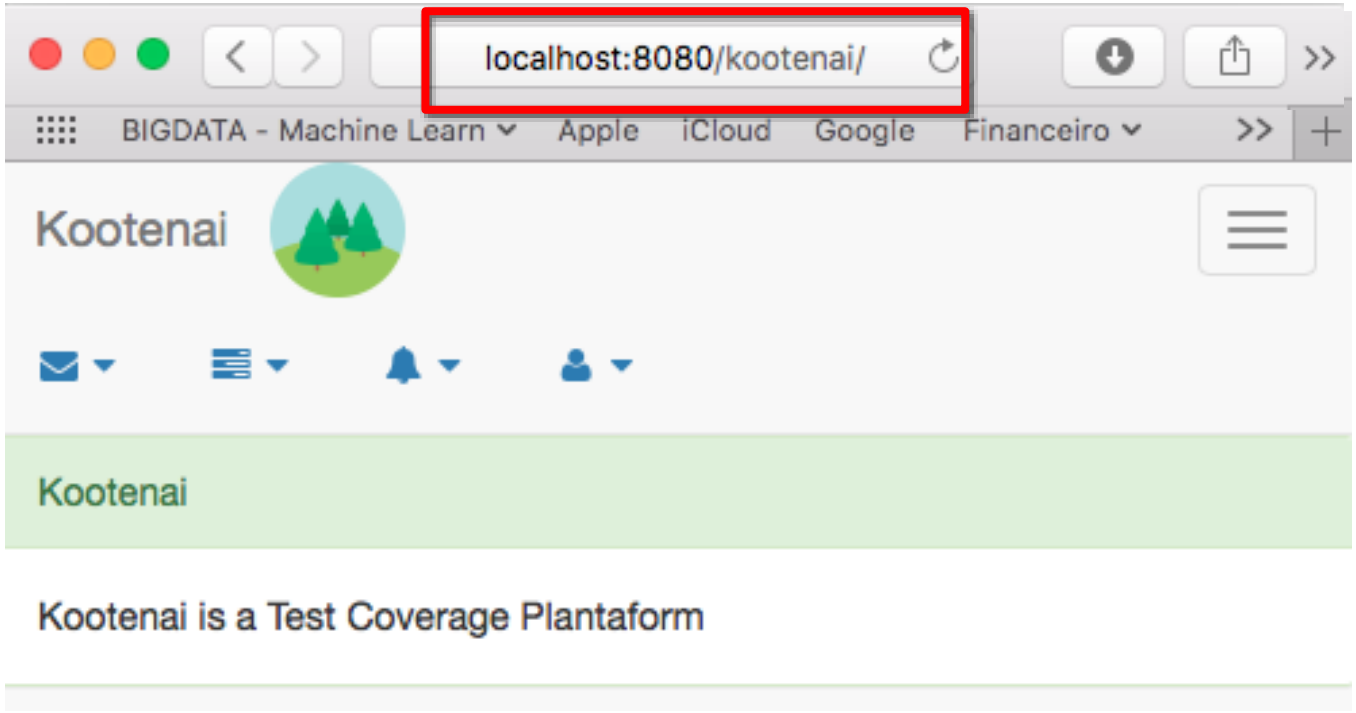


Angular Build to Production

- **Rename** the **dist** directory as application context name (**same name of base href in index.html**)
- Copy the directory put inside a HTTP Server (apache, tomcat, etc)



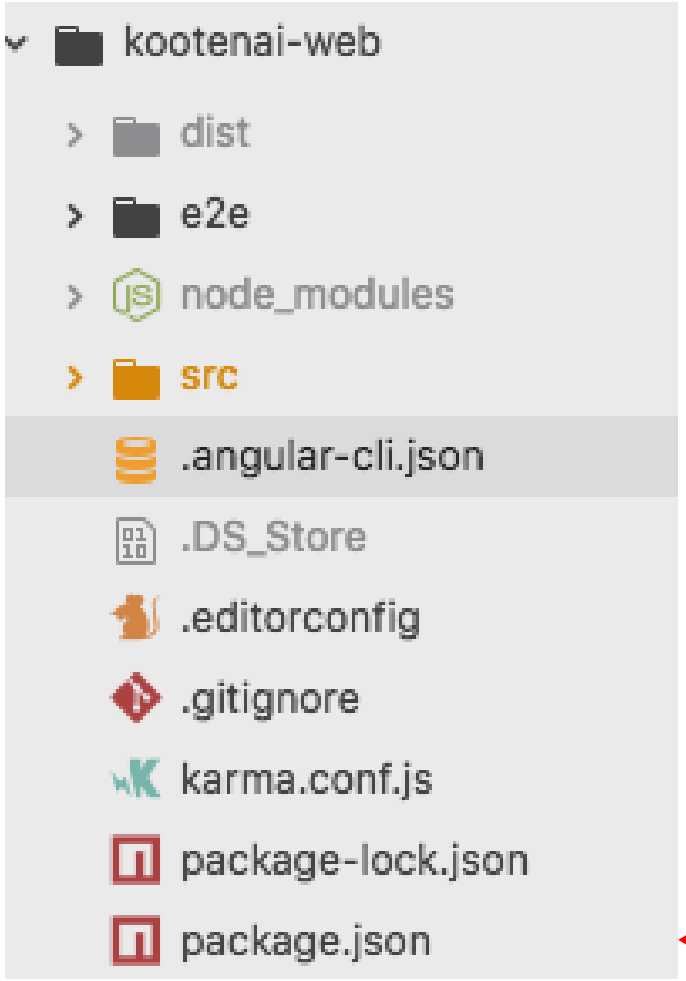
Angular Build to Production



Angular Commands Summary

- **ng new *name*** (create a new project)
- **npm install** (download all dependences and restore node_modules directory)
- **ng server** (run the application for development localhost:4200)
- **ng g m *name*** (generate a new module)
- **ng g c *name*** (generate a new component)
- **ng g s *name*** (generate a new server)
- **ng build --prod --env=prod** (build for production)

Angular Project Structure Overview



build output folder

end to end tests

Contains all dependences

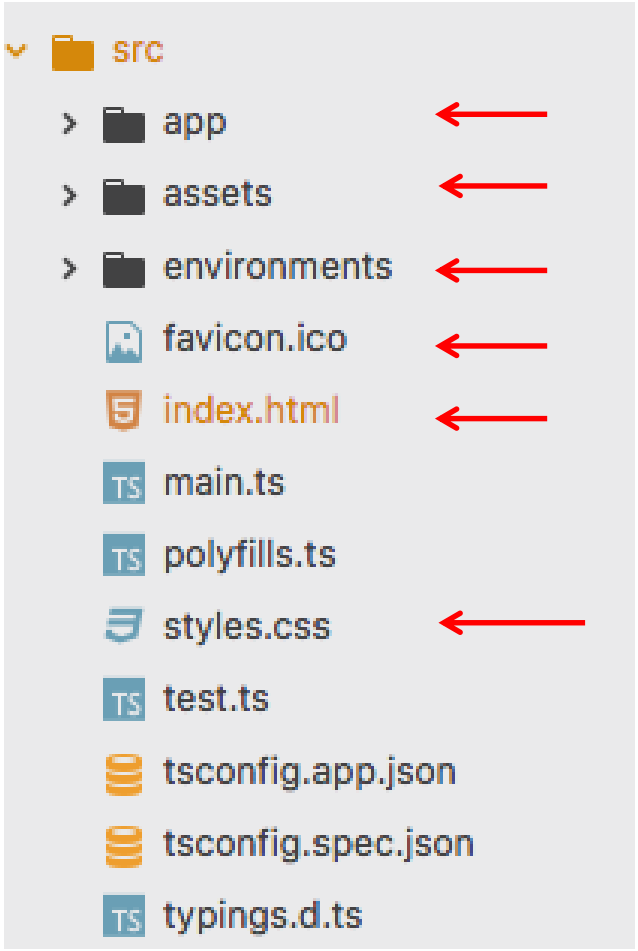
The code of our application

Configuration of your project

Git control Version

Declaration of angular dependences

Angular Project Structure (Inside src folder)



Our code

Images and other things can be put here

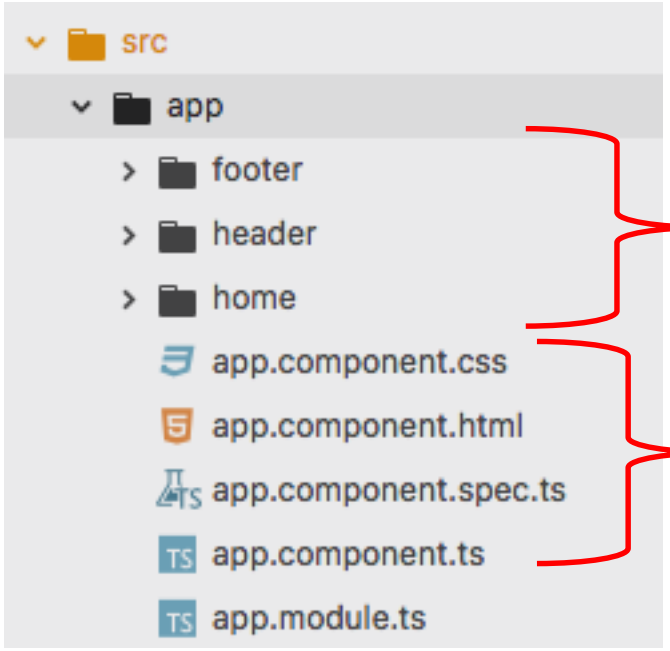
configuration to different environments (dev, test e prod)

The icon of our application

index.html of our application

Global CSS styles can be put here

Angular Project Structure (Inside app folder)



Our specific components

The main component (mandatory)

The main model (mandatory)