Angular 4

Angular is **client side javascript framework** that helps build complicated web application with very few Lines Of Code that is easy to maintain and have potential to grow.

Before we learn about angular framework we should understand Framework

What is a Framework? Why do we need Frameworks?

This is my definition not a book definition.
For a small and simple application we don't need frameworks. We can just straight away write application in the given platform, environment or a language.

E.g. if our business requirement is to write a simple add/subtract calculator application that will run on console/terminal, then we can just choose a platform/language and code it without any third party dependencies.

Now let's say business asks us to add more advanced calculation capabilities to our calculator then we can choose a **third-party library** that already does that and delegate those calculation to it.

Now let's say business wants us to make our calculator go enterprise and requirements like

- store user information who uses our calculator
- store history to user calculation
- provide some analytics on the user's calculations
- create a Web UI for it
- And more...

To implement these requirements we will have to write a lot of boilerplate code and maybe a lot of duplicate code, depending on our design, that is not related to business requirements. Which makes it our application difficult to maintain. Some example of this boilerplate code is:

- CURD operation on entity/tables
- Manage transactions
- Database connection pooling
- Services classes initializing and composing
- MVC
- And more…

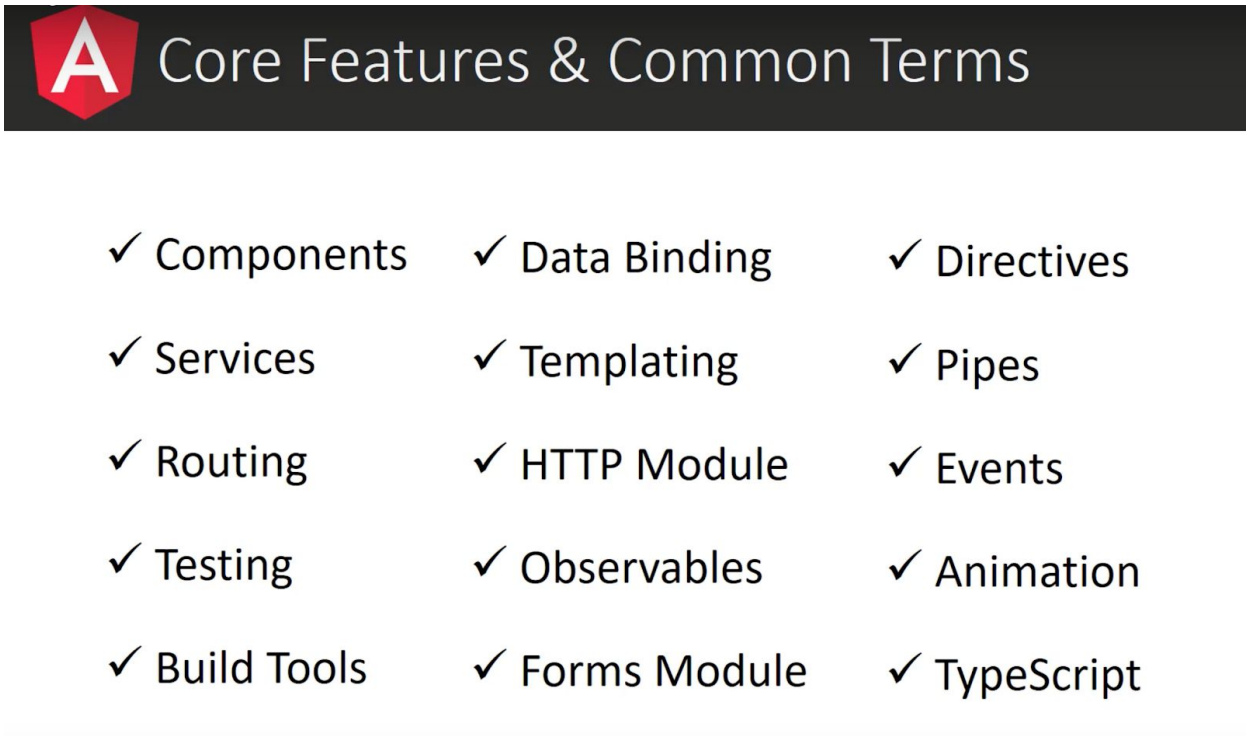
These concerns are common to most enterprise applications. A framework takes care of a lot of these common concerns that is unrelated to our business requirements. When we use a framework our application run within or on top of an abstract layer of framework. A framework could be made of a several libraries.

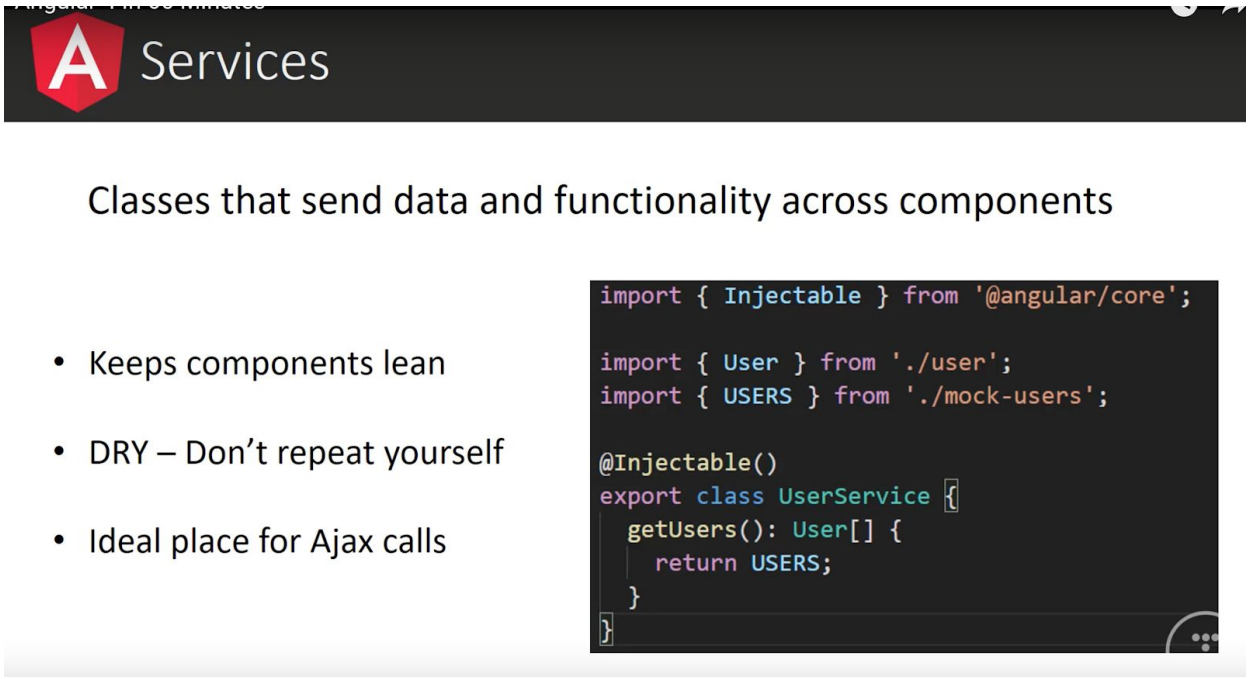
In addition to above features Framework could also provides additional benefits like:

- RAD - Rapid Application development
- Code Organization
- Cross Platform
- Unit Testing
- And more…

Why do we need angular?

- Create Single Page Application. Introduces all the benefits of SPA. e.g. smaller server request and response
- Create new tags (Components) and directives with richer functionality than existing HTML tags. E.g. <userProfile userId="123"></userProfile>
- After Imperatively writing components, Declaratively we use them. E.g. After coding all details of UserProfile component, we can use <userProfile></userProfile> wherever in the UI as many time as we want, even in loops.
- Separation of client side application concerns. UI, and Functionality.
- Modular application.
- And more...





Quick Start

https://github.com/angular/quickstart
Quick Start is a git project to create angular application. We can clone it to get a startup template.

Create Angular application using Quick Start

OSX:
git clone https://github.com/angular/quickstart app01_quick_start
cd app01_quick_start
rm -rf .git
xargs rm -rf < non-essential-files.osx.txt
rm src/app/*.spec*.ts
rm non-essential-files.osx.txt

npm install
npm start

Windows:
git clone https://github.com/angular/quickstart app01_quick_start
cd app01_quick_start
rd .git /S/Q
for /f %i in (non-essential-files.txt) do del %i /F /S /Q
rd .git /s /q
rd e2e /s /q
npm install
npm start

Create .gitignore and copy content from this file
https://github.com/angular/quickstart/blob/master/.gitignore

Angular CLI

https://cli.angular.io/
https://github.com/angular/angular-cli
Angular CLI generates angular application

Create Angular application using angular CLI

Install angular cli globally
$ npm install -g @angular/cli

Generate application and run server
$ ng new app02-angular-cli
$ cd app02-angular-cli
$ ng serve

NOTE: for some reason I haven't been able to use _ in project name using angular cli. It also gives error if I give "**app**" as project name.

Test generated application in browser
http://localhost:4200/

Generate other artifacts using angular cli

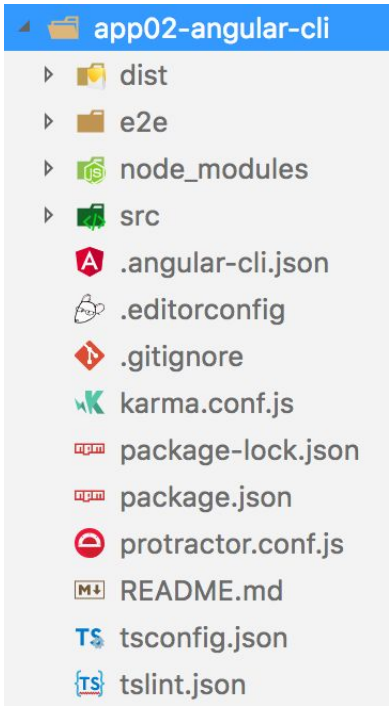
Scaffold	Usage

Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service
Class	ng g class my-new-class
Guard	ng g guard my-new-guard
Interface	ng g interface my-new-interface
Enum	ng g enum my-new-enum
Module	ng g module my-module

Angular application directory structure

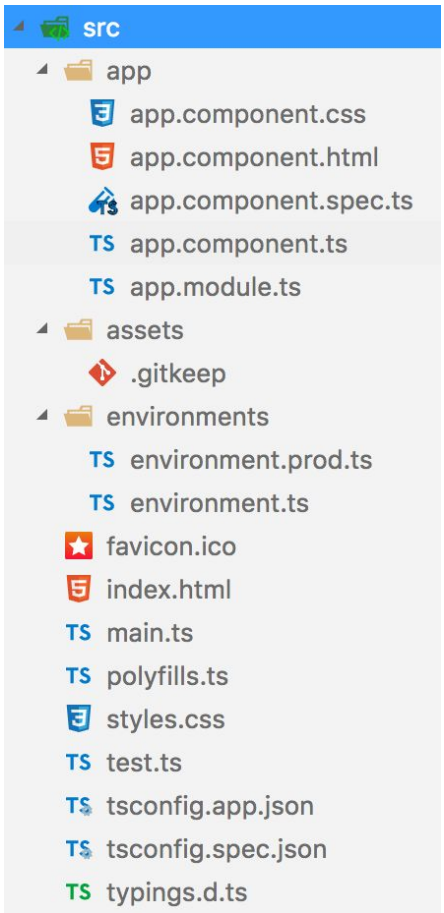
Application root folder

NOTE: for the most of our application development we will not modify root directory and files



dist/: Deployable code. This directory will be created after **ng build** command
e2e/: Application Unit Tests. End To End tests
node_modules/: npm dependencies
src/: Application code
.angular-cli.json: cli build configuration
.editorconfig: http://editorconfig.org/ IDE code style configuration
.gitignore: git ignore files patterns
karma.conf.js:Karma test runner
package-lock.json: Auto generated npm file. Describe dir structure of npm dependencies.
package.json: Project Manifest - describe node/npm project, dependencies and build script
Protractor.conf.js: http://www.protractortest.org/
tsconfig.json: TypeScript compiler/transpiler configuration
tslint.json: TypeScript lint configuration

Application src folder



src/app/: Actual angular application code.
src/assets/: Store assets. E.g. images
src/environments/

src/favicon.ico: Site's favicon
src/index.html: Our only application html that contains root component tag
src/main.ts: bootstraps angular application
src/polyfills.ts
src/styles.css
src/test.ts
src/tsconfig.app.json
src/tsconfig.spec.json
src/typings.d.ts

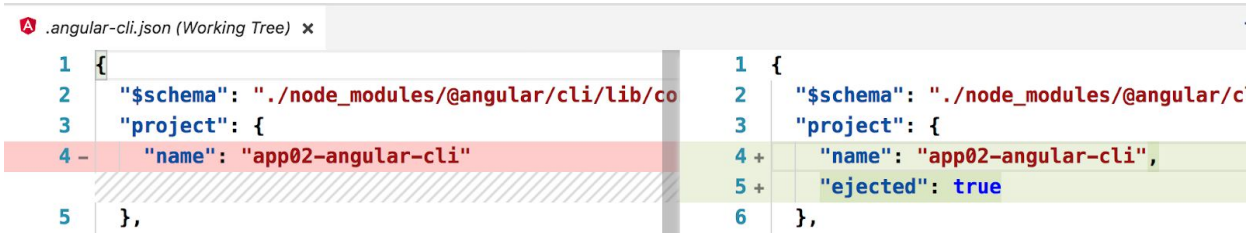
Convert Angular CLI build configuration to Webpack

https://github.com/angular/angular-cli/wiki/eject
By default **ng new <app-name>** command builds **.angular-cli.json** file. This file holds all angular application build configurations. Internally **ng** build commands use **webpack** to build angular application.

If we are more comfortable with configuring webpack's **webpack.config.js** instead of ng's **.angular-cli.json** then we use this command:

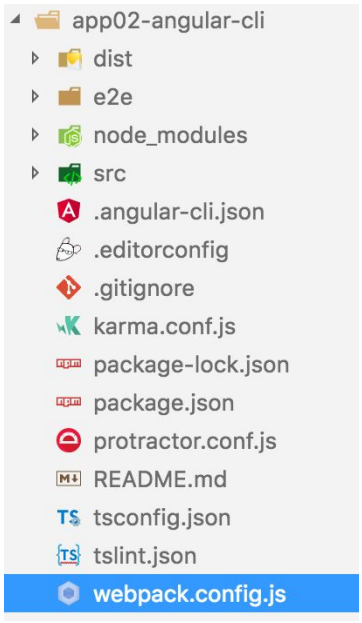
$ ng eject

Above command will add "ejected: true" to **.angular-cli.json**. Which means that all build configuration will be read from **webpack.config.js**.



In package.json all the build scripts will be changed to webpack build scripts

And you will find new **webpack.config.js**

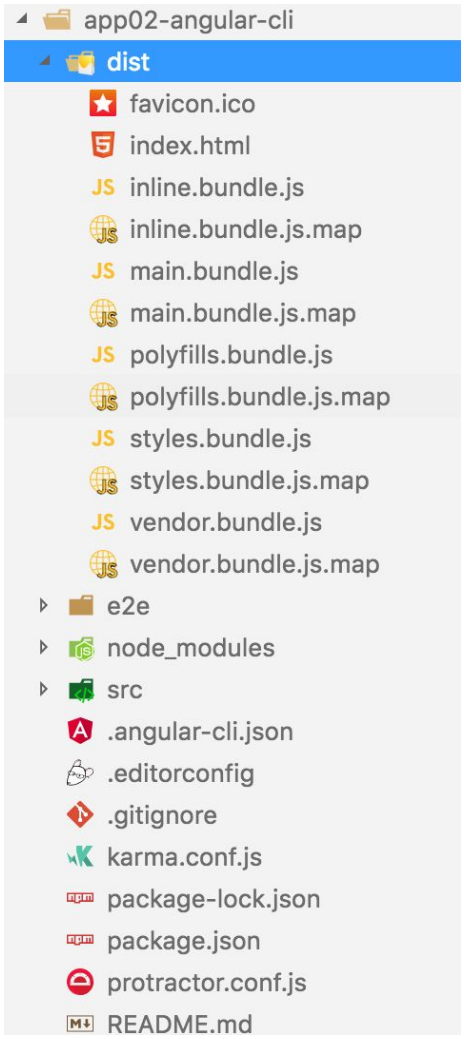


Build application deployment

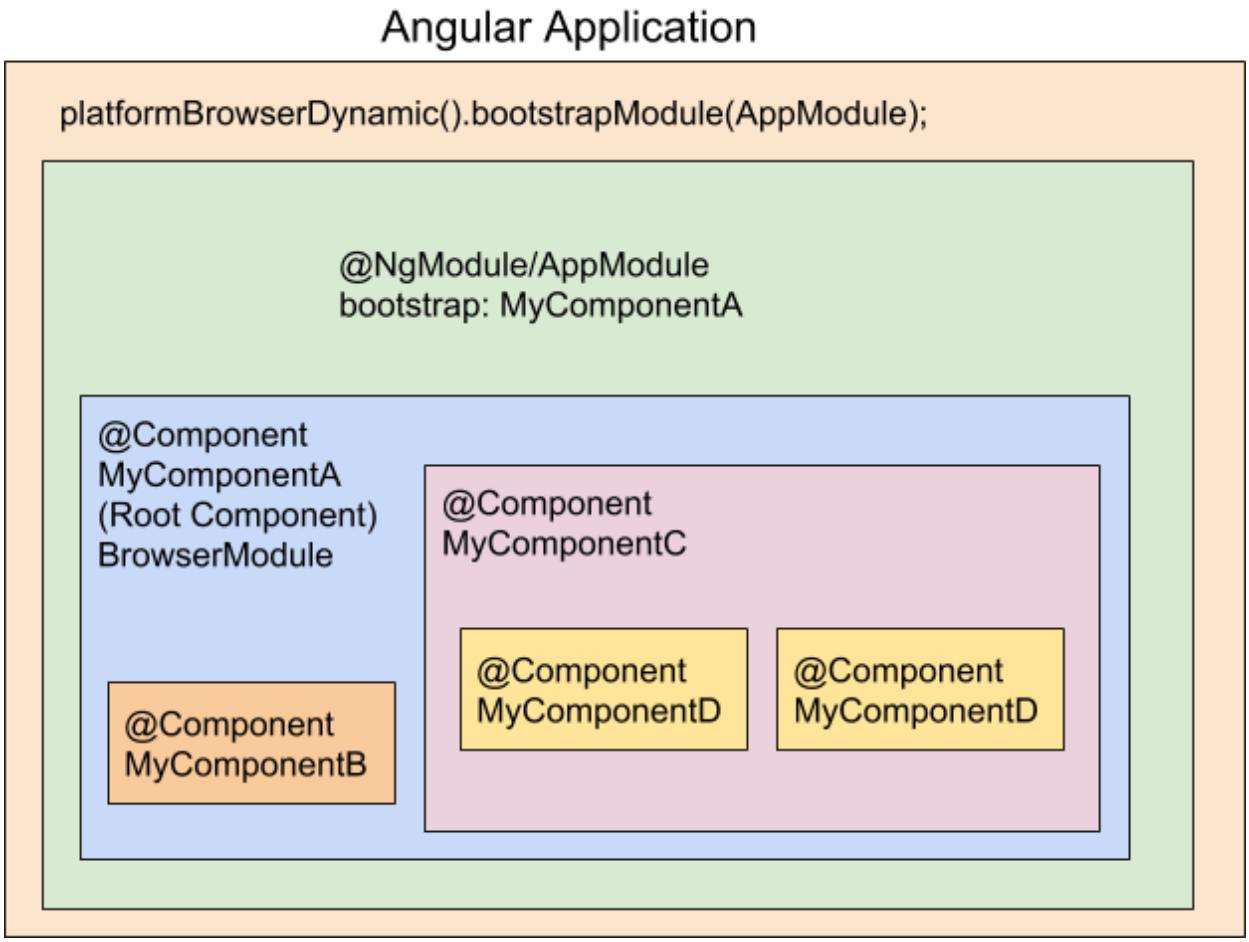
https://github.com/angular/angular-cli/wiki/build
Once we done writing our application and ready to make build application deployment we write this command:

$ npm run build
Or
$ ng build
Or to build production ready build and have compressed built with no sourcemap
$ ng build --target=production --environment=prod

This command will create **dist** folder. This folder or the files inside can be deployed



Angular Application

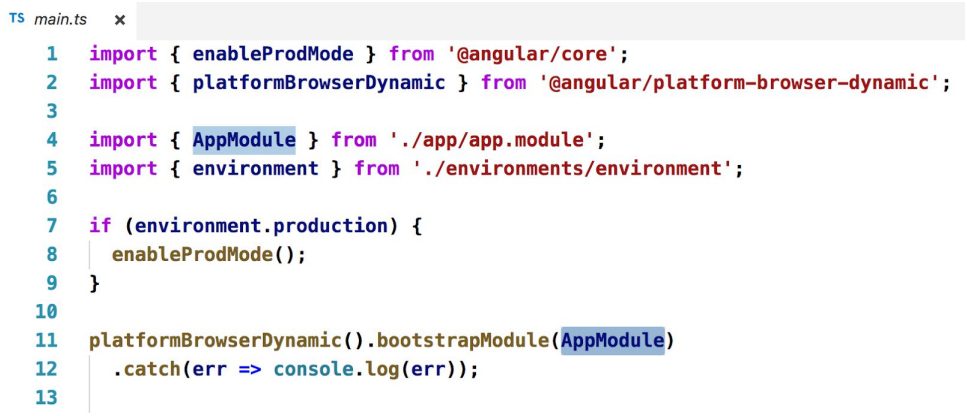


Start up Flow of Angular Framework and then Angular Application

.angular-cli.json -> main.ts -> AppModule/@NgModule -> Root Component -> Tree of other Components

Application Bootstrap - main.ts

https://angular.io/guide/bootstrapping#bootstrap-in-maints
Just like **main** class file Java or .Net, **main.ts** is the main entry point for Angular application. It's name is defined in **.angular-cli.json**.
As a convention main.ts is responsible to start Angular Framework and tell it about AppModule/@NgModule



AppModule - @NgModule

https://angular.io/guide/ngmodule
https://medium.com/@cyrilletuzi/understanding-angular-modules-ngmodule-and-their-scopes-81e4ed6f7407

```
@NgModule({
 declarations: [],
 imports: [],
 exports: [],
 providers: [],
 bootstrap: [],
})
```

Angular application is built of different building blocks. @NgModule in Angular application organizes all these building blocks. @NgModule have 5 groupings (Arrays) of these building blocks:

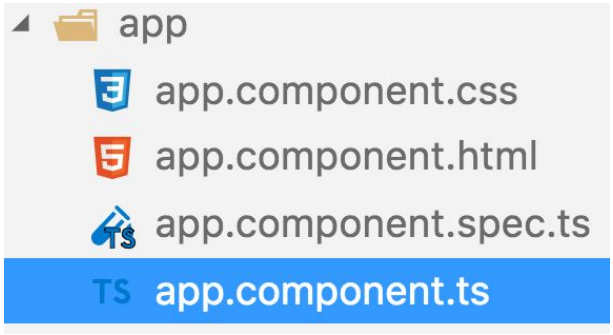
1. **`declarations`**: UI objects (components, directives, and pipes) that this module contain. UI objects are the ones that used in HTML template
2. **`imports`**: Other modules that this module depends on. Built-in modules E.g. Web browser capable, forms, http,... Or our custom modules
3. **`providers`**: Simple TypeScript classes. Array of Services that could be dependency injected
4. **`exports`**: UI objects or other module that will be used outside this modules.
5. **`bootstrap`**: Root Component. This is the component that we will include in index.html. index.html (static DOM, initial DOM) could have multiple root component tags. But ideally you will just define single root component.

@Component

https://angular.io/api/core/Component
Components are building UI blocks. A recommended UI component is composed of 4 files. E.g. the root **app.component** component built by **ng new <project-name>**, is composed of 4 files:

1. **app.component.css**: contains all the styles related to this component
2. **app.component.html**: contains HTML template
3. **app.component.spec.ts**: Unit test
4. **app.component.ts**: Component's controller

NOTE: the naming convention of component files



app.component.ts

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

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

```

A typical Component controller TypeScript file contains these elements:

1. `selector`: Tag that this component will create. We will use this tag in HTML where we want to place this component
2. `templateUrl`: Relative path to the HTML file that this component will show.
3. `styleUrls`: Relative path to the CSS file. These styles will be applied to this component.

NOTE: Any public property and function in @Component class is available in HTML template file.

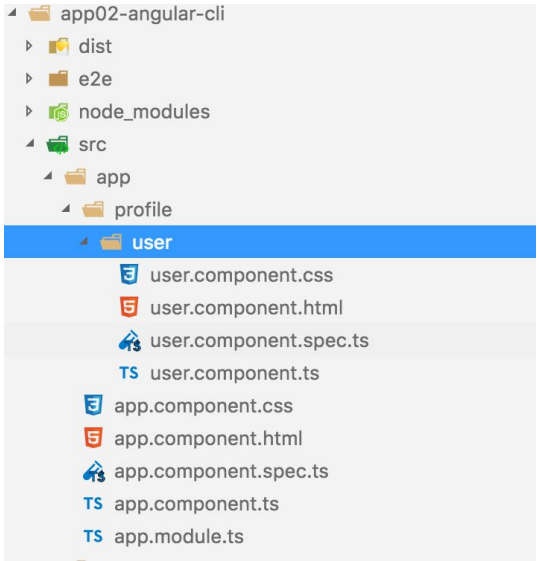
Generating new blank component

Instead of manually creating all 4 files mentioned above we can **ng** cli tool to generate new blank component
E.g. our application name is **app02-angular-cli** and we want to add a component **user** inside **profile** folder. In the terminal we need to be in **app02-angular-cli** folder (our application root folder) and give this command:

$ ng generate component profile/user

It will generate 4 component files and update app.module.ts

```
Sherazs-MacBook-Pro:app02-angular-cli sheraz$ ng g component profile/user
  create src/app/profile/user/user.component.css (0 bytes)
  create src/app/profile/user/user.component.html (23 bytes)
  create src/app/profile/user/user.component.spec.ts (614 bytes)
  create src/app/profile/user/user.component.ts (261 bytes)
  update src/app/app.module.ts (396 bytes)
```



New Component

A new generated component contains

- **app-user selector** (selector is the tag that we would use in UI)
- maps **templateUrl** and **styleUrls**
- Adds OnInit lifecycle hook. More on Component lifecycle hook will be explained in later section.

user.component.ts

```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-user',
5    templateUrl: './user.component.html',
6    styleUrls: ['./user.component.css']
7  })
8  export class UserComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit() {}
13 }
14
```

Hint: By default component selector is prefixed with "app-". It can be changed in **tslint.json**.

tslint.json

```
113       "check-operator",
114       "check-separator",
115       "check-type"
116     ],
117     "directive-selector": [
118       true,
119       "attribute",
120       "app",
121       "camelCase"
122     ],
123     "component-selector": [
124       true,
125       "element",
126       "app",
127       "kebab-case"
128     ],
```

Changes in app.module.ts

In **app.module.ts** it will add **UserComponent** in **declarations**.

app.module.ts (Working Tree)

```
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 { }
```

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

import { AppComponent } from './app.component';
import { UserComponent } from './profile/user/user.component';

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

Use component

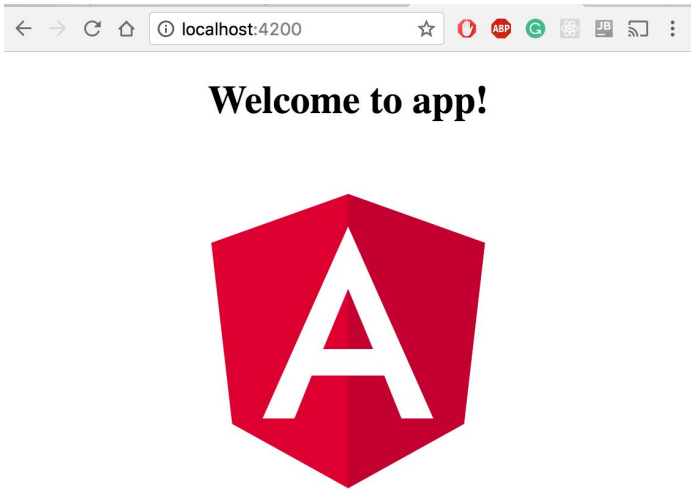
To use newly created component, we need to add <app-user></app-user> tag where we want to add place this component.

app.component.html (Working Tree)

```
  <img width="300" src="data:image/svg+xml;base64
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" href="https://angular.
  </li>
  <li>
    <h2><a target="_blank" href="https://github.c
  </li>
  <li>
    <h2><a target="_blank" href="https://blog.ang
  </li>
</ul>
```

```
  <img width="300" src="
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" href="https://angular.io
  </li>
  <li>
    <h2><a target="_blank" href="https://github.com
  </li>
  <li>
    <h2><a target="_blank" href="https://blog.angul
  </li>
</ul>
<app-user></app-user>
```

Test Component



Component Lifecycle Hooks

https://angular.io/guide/lifecycle-hooks

Event

List of all DOM Events
https://developer.mozilla.org/en-US/docs/Web/Events

We can call Component's method on any DOM event.

In the example below, on button's **click** event we call **changeMessage()** method.
DOM events are written in parenthesis
Method to be called, are written in in quotes as value

Component

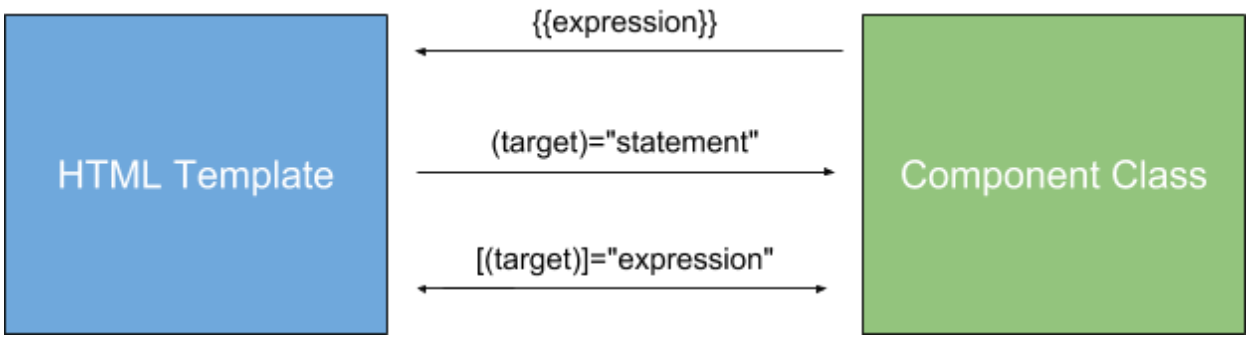
```
import { Component, OnInit } from '@angular/core';

@Component({
 selector: 'app-form01-button-click',
 templateUrl: './form01-button-click.component.html',
 styleUrls: ['./form01-button-click.component.css']
})
export class Form01ButtonClickComponent {
 message = 'Hello World';

 changeMessage() {
   this.message = `Message updated ${new Date()}`;
 }
}
```

UI

```
<button (click)="changeMessage()" class="btn btn-info">
 Update message
</button>
{{message}}
```

Binding Direction in HTML Templates

Data direction	Syntax	Can be used by
One-way from data source to view target	{{expression}} [target]="expression" bind-target="expression"	Interpolation Property Attribute Class Style
One-way from view target to data source	(target)="statement" on-target="statement"	Event
Two-way	[(target)]="expression" bindon-target="expression"	Two-way

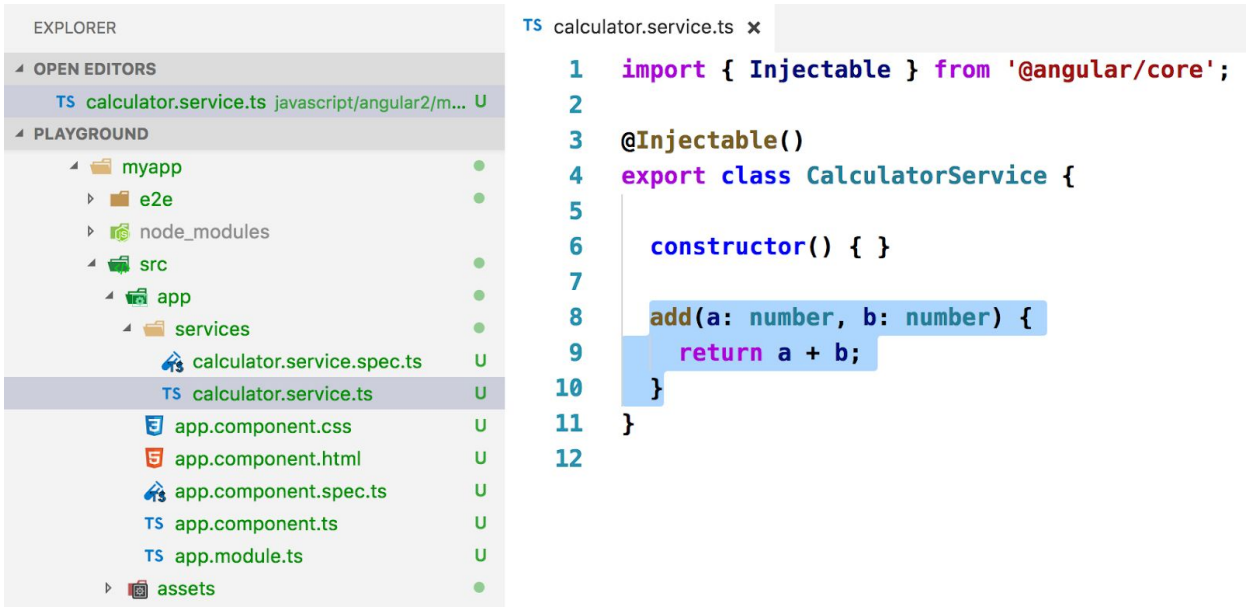


Services - Learn By Example

Create new application and add Calculator service in services folder
$ ng new myapp
$ ng generate service services/Calculator

Service

Add a add method to the service. @Injectable tells angular that this class can be used as a provider that can be dependency injected



Module

Add FormsModule in imports because we want to add HTML form elements.

Add CalculatorService in providers so that it will be available for dependency injection.

app.module.ts

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

import { AppComponent } from './app.component';
import { CalculatorService } from './services/calculator.service';


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

```

Component

Dependency Inject Calculator service in AppComponent

app.component.ts

```ts
import { Component } from '@angular/core';
import { CalculatorService } from './services/calculator.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  numA: number;
  numB: number;
  result: number;

  constructor(public calculatorService: CalculatorService) {
  }

  add() {
    this.result = this.calculatorService.add(this.numA, this.numB);
  }
}
```

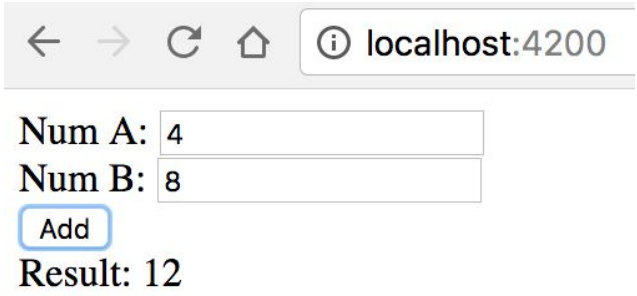
Template

Bind modal variable and call add() function

app.component.html

```html
Num A: <input type="number" [(ngModel)]="numA"><br/>
Num B: <input type="number" [(ngModel)]="numB"><br/>
<button (click)="add()">Add</button><br/>
Result: {{result}}
```

Working Example



Combining Modules - Learn by example

In this example there are 3 services.

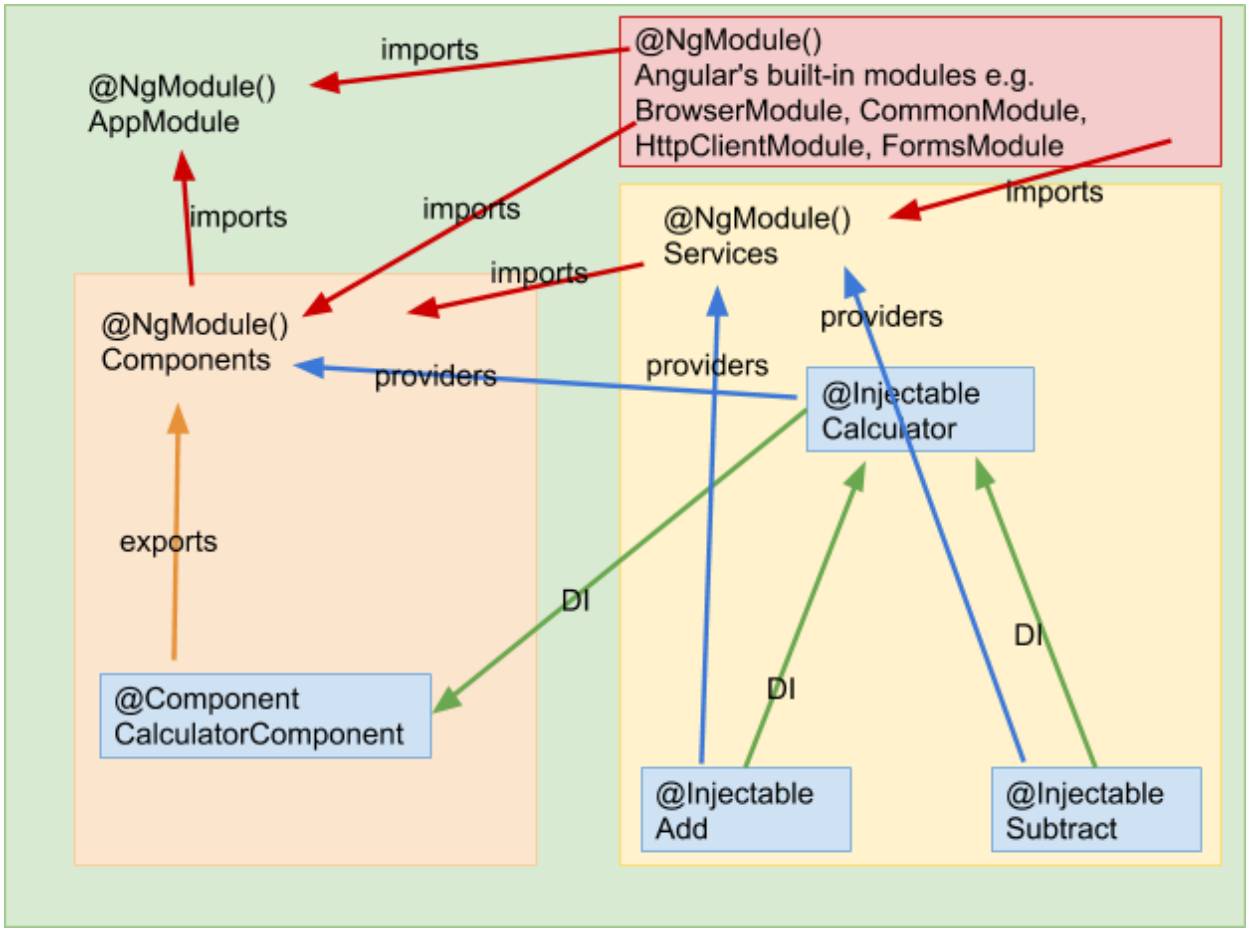
- Calculator
- Add
- Subtract

Calculator service depend on Add and Subtract
There is a CalculatorComponent it depends on Calculator Service
CalculatorComponent is used inside AppComponent

To organize our code we will create 3 @NgModule

- AppModule: contains app root component
- Components: Contains our custom components
- Services: Contains our custom services

The above example explanation will be achieved by doing below



Use ng CLI to build all Objects

App
$ ng new app04-combining-modules
$ cd app04-combining-modules

Modules
$ ng generate module components

$ ng generate module services

ng will create module name folder and create **.module.ts** file in it

Components
$ ng generate component components/Calculator

- ng will append **Component** suffix to Calculator component (look in .component.ts file)
- We created **CalculatorComponent** inside **components** folder. **components** folder is **components** module folder
- ng will create **calculator** folder in **components** folder and create **.component.css, .component.html, .component.spec.ts, .component.ts** files.
- ng added **CalculatorComponent** in **declaration** array of in **components.module.ts**

Services
$ ng generate service services/Calculator
$ ng generate service services/Add
$ ng generate service services/Subtract

- ng will append **Service** suffix to Calculator, Add and Subtract service (look in .service.ts file)
- We created **CalculatorService, AddService and SubtractService** inside **services** folder. **services** folder is **services** module folder
- ng will not create service name folder
- ng will not add services in module's providers array. We will have to do it manually.

Add service methods and dependency injection

This is to make 2 DI green arrows in design diagram inside @NgModule Services box

calculator.service.ts

```ts
import { Injectable } from '@angular/core';
import { SubtractService } from './subtract.service';
import { AddService } from './add.service';

@Injectable()
export class CalculatorService {

  constructor(private addService: AddService,
    private subtractService: SubtractService) {}

  add(a: number, b: number) {
    return this.addService.add(a, b);
  }

  subtract(a: number, b: number) {
    return this.subtractService.subtract(a, b);
  }
}
```

add.service.ts

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

@Injectable()
export class AddService {
  add(a: number, b: number) {
    return a + b;
  }
}
```

subtract.service.ts

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

@Injectable()
export class SubtractService {
  subtract(a: number, b: number) {
    return a - b;
  }
}
```

Providers in services modules

To have angular initialize AddService and SubtractService class and inject its instance in CalculatorService class we will have to add AddService and SubtractService as providers in services.module.ts

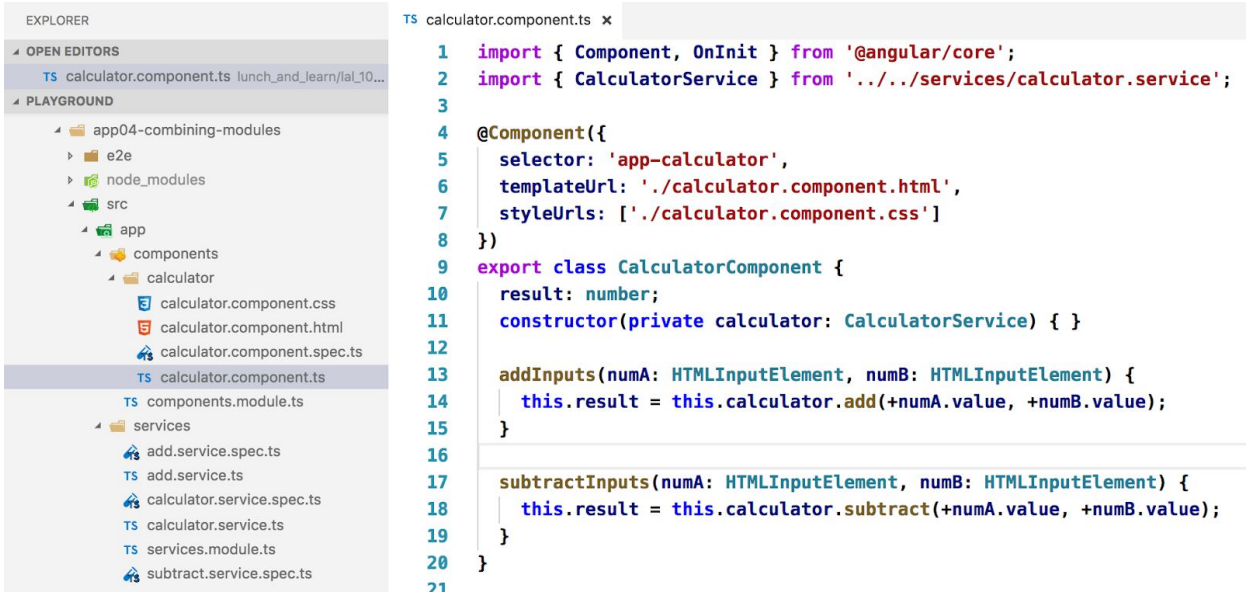
This will make 2 provider blue arrows from @Injectable Add and @Injectable Subtract to @NgModule Services

Also note angular's CommonModule being imported

CalculatorComponent event methods and CalculatorService dependency injection

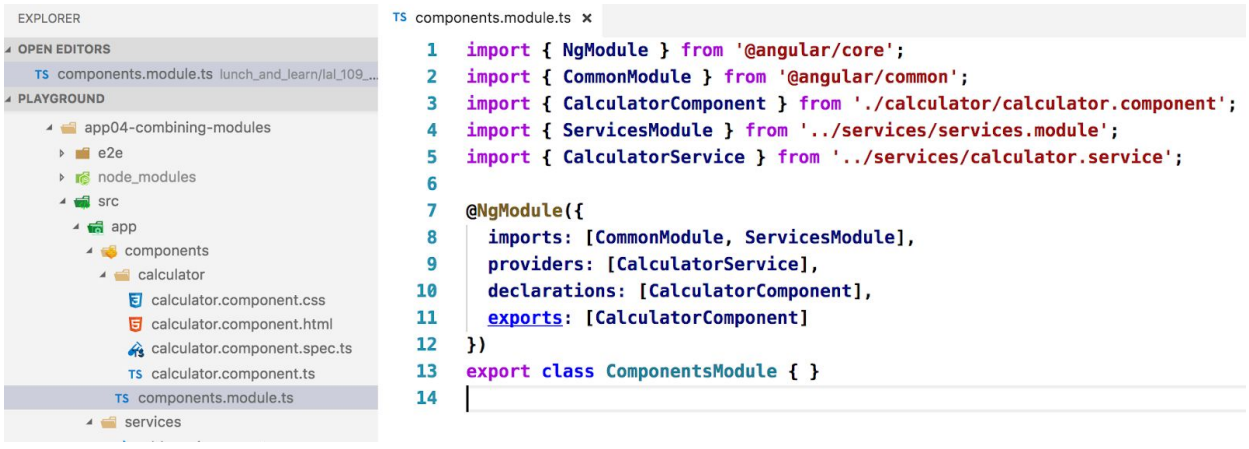
Dependency Inject CalculatorService in CalculatorComponent and using CalculatorService methods in UI event methods.

This will make the green arrow from @NgModule Service to @NgModule Components



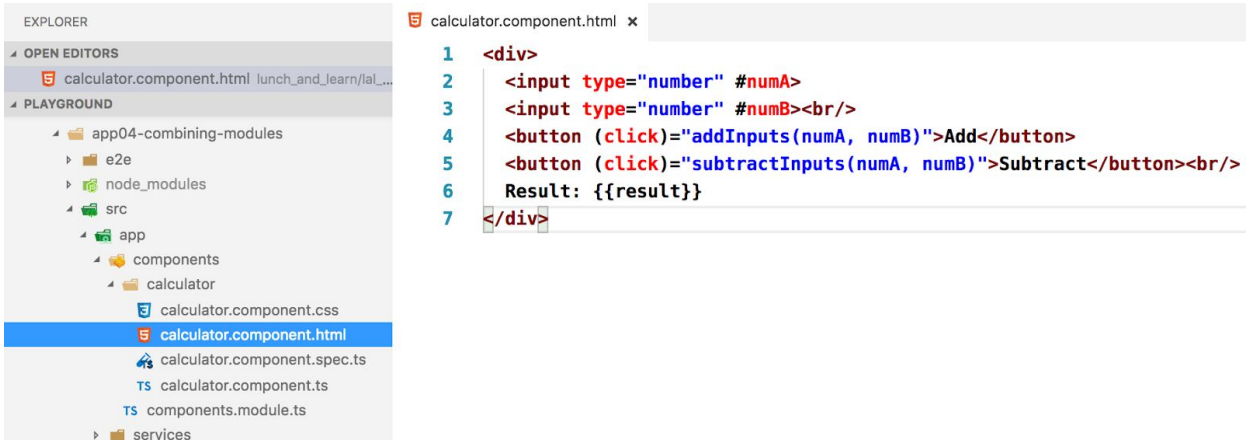
ComponentsModule changes

Since ComponentsModule depends on ServiceModule we need to add it to imports. This will make red arrow from @NgModule Services to @NgModule Components
Angular CLI added angular's CommonModule to import as well. This will make red arrow from angular built-in modules to @NgModule Components
To have angular initialize CalculatorService and dependency inject in CalculatorComponent we need to add CalculatorService in providers. This will make blue arrow from @Injectable CalculatorService to @NgModule Components
Since we need to use CalculatorComponent outside this module so we need to add it to exports. This will make orange arrow from @Component CalculatorComponent to @NgModule Components



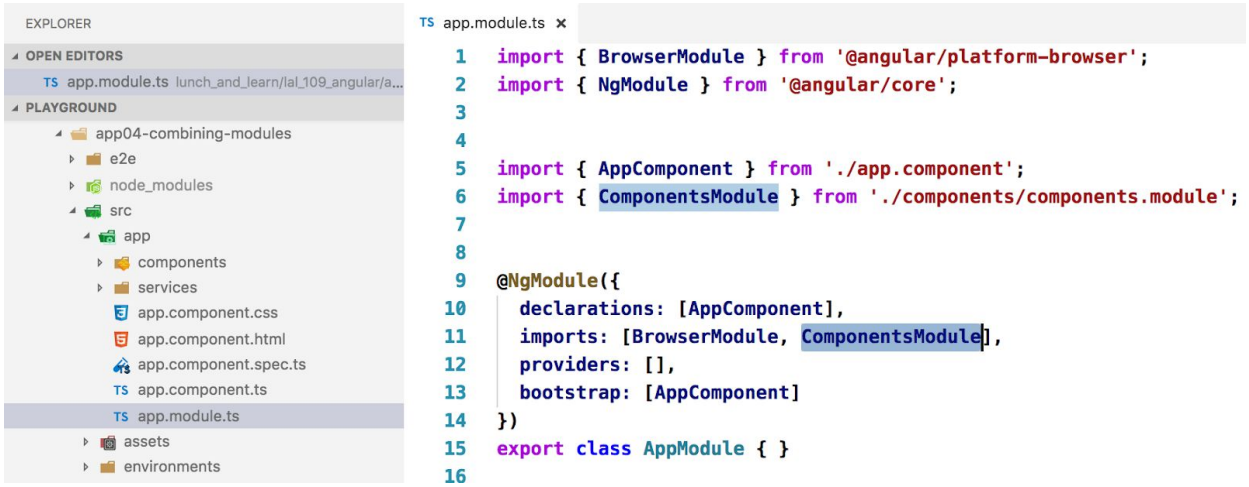
Calculator component template

- Instead of binding inputs to model, we have added #numA and #numB. Think this as HTML's id="" attribute. By doing that we can reference that element inside template.
- On button click events we are passing input elements to addInputs() and subtractInputs() methods.
- In the end we are showing CalculatorComponent result property.

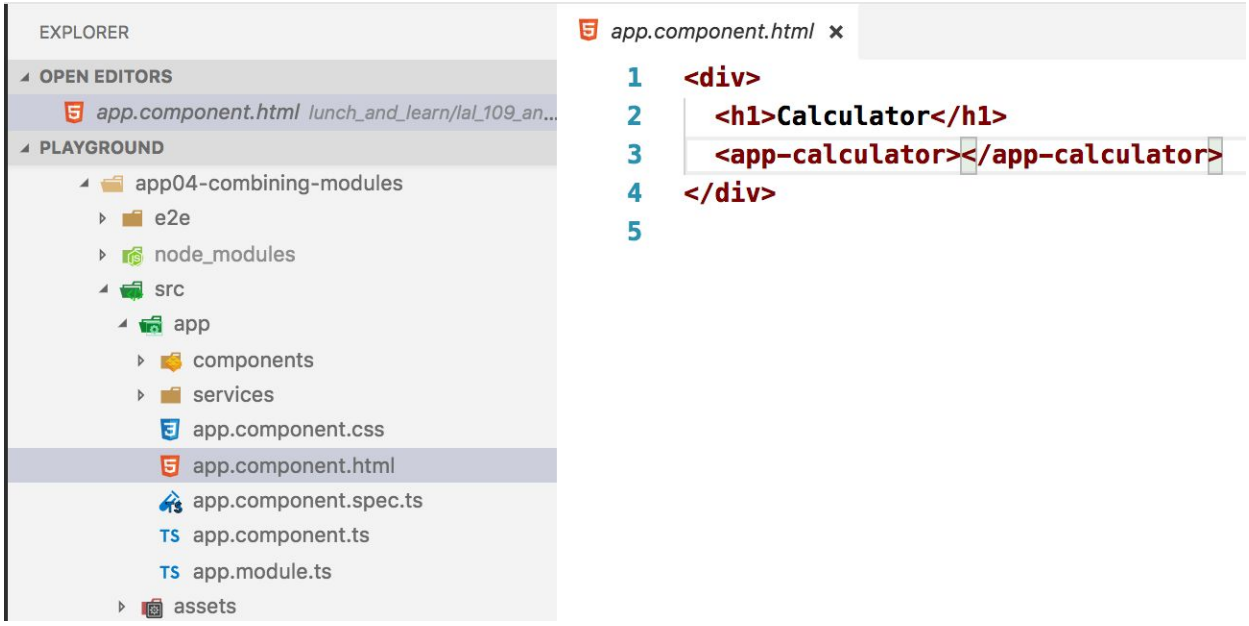


Add ComponentsModule in AppModule

By importing ComponentsModule in AppModule makes <app-calculator/> element available in AppComponent because CalculatorComponent was exported from ComponentsModule.
This will make red arrow from @NgModule Components to @NgModule App



Using <app-calculator/> in app.component.html template

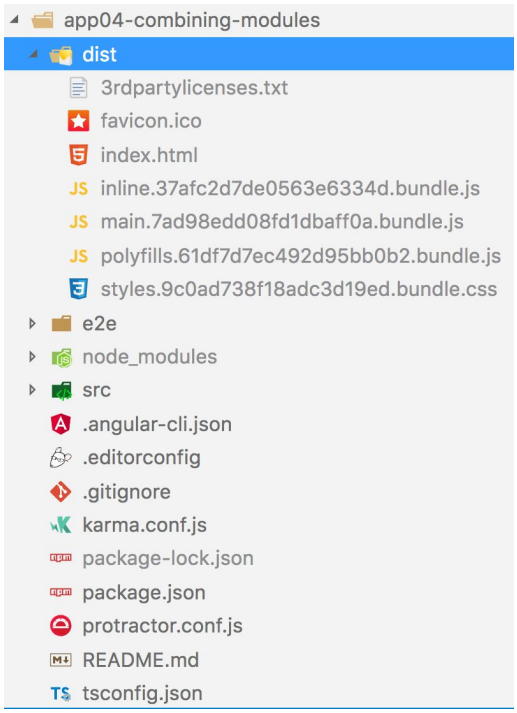


Production build

To create a production build run the command below

```
$ ng build --target=production --base-href '/'
```

This command will create production build under **dist** folder.

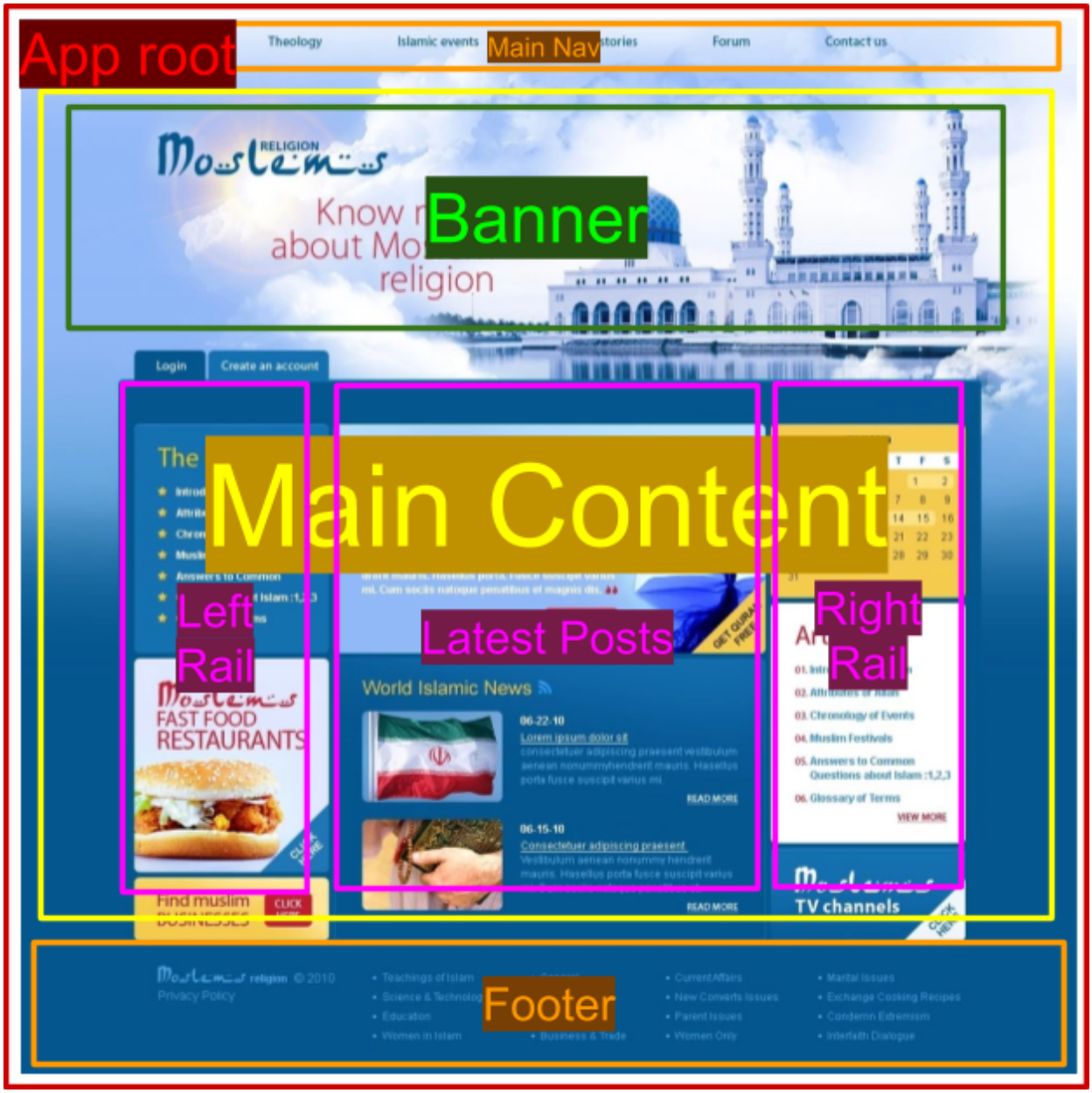


What is Router? and Why do we need it?

Let's say webdesigner gave us the below HTML template and we want to develop this webpage in angular. The requirements are:

- main nav and footer are displayed on all pages
- main nav and footer are main drivers of our website, available at all times.
- main content will be **replaced** for each main nav link
- main nav links should be bookmarkable

We will first divide each section into components and identify the Main Content section that will be different for each Main Nav link.



If we were to create it with plain Angular (without routers). Then our AppRoot template will look like this:

```
<div class="container">
 <main-nav></main-nav>
 <div class="content">
   <home *ngIf="showHomeCondition">
     <banner/>
     <left-rail></left-rail>
     <latest-ports></latest-ports>
     <right-rail></right-rail>
   </home>
   <theology *ngIf="showTheologyCondition">
```

```
        <banner/>
        <quran></quran>
        <study-matirial></study-matirial>
    </theology>
    ...
  </div>
  <main-footer></main-footer>
</div>
```

For each <main-nav> and <main-footer> link's click event we will have to set the conditions like showHomeCondition, showTheologyCondition … . Using this condition we will show or hide different main content sections. We will also not be able to bookmark.

To overcome these issue we use router


```
<div class="container">
 <main-nav></main-nav>
 <div class="content">
   <router-outlet></router-outlet>
 </div>
 <main-footer></main-footer>
</div>
```

- We give, replacing main content control to router.

- Place router links like `<a [routerLink]="['home']">home</a>` wherever in our application

Router

Routers are used for:

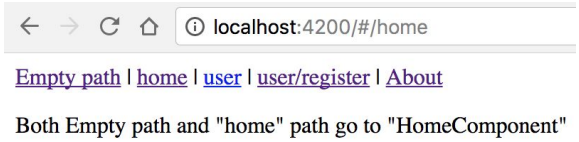
- “Single Page Apps” (SPA)
- hash-based routing or HTML5 history links
- Create dynamic links in SPA application

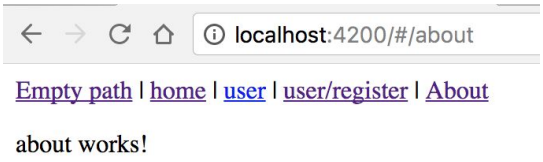
Components of Angular Router

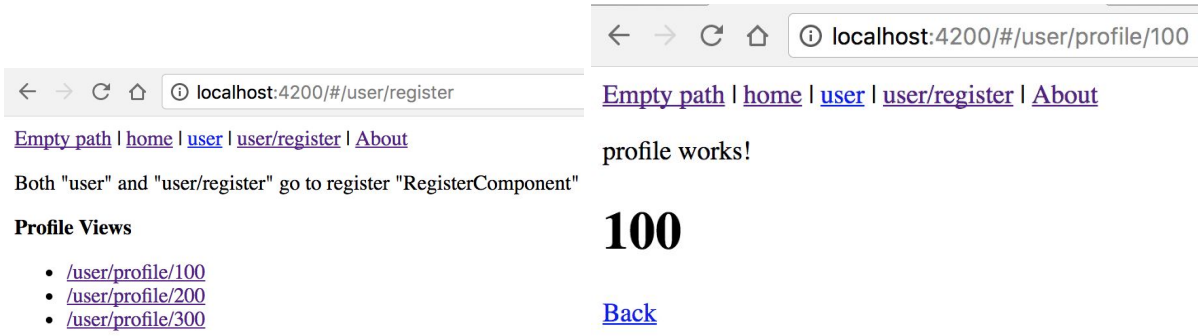
- **Routes** describes the routes our application supports. https://angular.io/api/router/Routes
- **RouterOutlet** is a “placeholder” component that shows Angular where to put the content of each route. https://angular.io/api/router/RouterOutlet
- **RouterLink** directive is used to link to routes. https://angular.io/api/router/RouterLink
- **ActivatedRoute** to get values from dynamic links. Values like path parameters and query parameters. https://angular.io/api/router/ActivatedRoute

Router - Learn by example

- Create main nav that stays on all pages
- Create 4 pages. Home, Register, Profile, About
- / and /home Should both go to Home page
- /about should go to About page
- /user and /user/register should go to Register page.
- Register page should display dynamically generated list of profile links
- Profile page should show profile of the ID passed to it.







Generate App

$ ng new app05-router-parameter
$ cd app05-router-parameter
$ ng generate component component/home
$ ng generate component component/about
$ ng generate component component/register
$ ng generate component component/profile

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { RouterModule, Routes } from '@angular/router';
4. import { LocationStrategy, HashLocationStrategy, APP_BASE_HREF
   } from '@angular/common';
5.
6. import { AppComponent } from './app.component';
7. import { HomeComponent } from
   './components/home/home.component';
8. import { AboutComponent } from
   './components/about/about.component';
9. import { RegisterComponent } from
   './components/register/register.component';
10.  import { ProfileComponent } from
   './components/profile/profile.component';
11.
12.  const childRoutes: Routes = [
13.    {path: '', redirectTo: 'register', pathMatch: 'full'},
14.    {path: 'register', component: RegisterComponent},
15.    {path: 'profile/:id',
16.      component: ProfileComponent}, // Path parameter :id
17.  ];
18.
```

```
19.  const routes: Routes = [
20.   {path: '', redirectTo: 'home',
21.     pathMatch: 'full'}, // if no path then redirect to /home
22.   {path: 'home', component: HomeComponent},
23.   {path: 'about', component: AboutComponent},
24.   {path: 'user',
25.     children: childRoutes} // child routes. all have "/user"
  prefix
26.  ];
27.
28.  @NgModule({
29.   declarations: [
30.     AppComponent, // AppComponent root component contains
  main nav
31.     ProfileComponent,
32.     AboutComponent,
33.     RegisterComponent,
34.     HomeComponent
35.   ],
36.   imports: [
37.     BrowserModule,
38.     RouterModule.forRoot(routes)
39.   ],
40.   providers: [ // 2 lines below will make Hash base URL. If
  Commented then it use HTML5 URL
41.     { provide: LocationStrategy, useClass:
  HashLocationStrategy },
42.     { provide: APP_BASE_HREF, useValue: '/' }
43.   ],
44.   bootstrap: [AppComponent]
45.  })
46.  export class AppModule { }
```

Line 19 and 12: Define Routes.
Line 25: Added **childredRoutes** as **children** of **routes**. All children routes will have prefix of **user** which will be **/user** in URL
Line 15: Added path parameter :**id**
Line 13 and 20: redirecting in case path is empty
Line 40-42: Added HashLocationStrategy. This done for the browser that do not support HTML history paths https://developer.mozilla.org/en-US/docs/Web/API/History_API
Default HTML5 path strategy is PathLocationStrategy.
https://angular.io/api/common/PathLocationStrategy
https://angular.io/api/common/HashLocationStrategy

Angular 1 Note: HTML5 routing works in Angular 1, but it needs to be explicitly enabled using $locationProvider.html5Mode(true).

app.component.html

```
<div class="container">
 <header>
   <a [routerLink]="['']">Empty path</a> |
   <a [routerLink]="['home']">home</a> |
   <a [routerLink]="['user']">user</a> |
   <a [routerLink]="['user/register']">user/register</a> |
   <a [routerLink]="['about']">About</a>
 </header>
 <div class="content">
   <router-outlet></router-outlet>
 </div>
</div>
```

Added **routerLink** and **<router-outlet>**. This will main nav and section where main content will be placed.

register.component.ts

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

@Component({
 selector: 'app-register',
 templateUrl: './register.component.html',
 styleUrls: ['./register.component.css']
})
export class RegisterComponent {
 profileIds = [100, 200, 300];
}
```

register.component.html

```
<p>
 Both "user" and "user/register" go to register
"RegisterComponent"
</p>
<div>
<b>Profile Views</b>
<ul>
 <li *ngFor="let profileId of profileIds">
   <a [routerLink]="['/user', 'profile', profileId]">
```

```
      /user/profile/{{profileId}}
    </a>
  </li>
</ul>
</div>
```

Build links to /user/profile and pass {{profileId}} number in the path

profile.component.html

```
<p>
 profile works!
</p>
<h1>{{profileId}}</h1>
<a href="javascript: history.back();">Back</a>
```

profile.component.ts

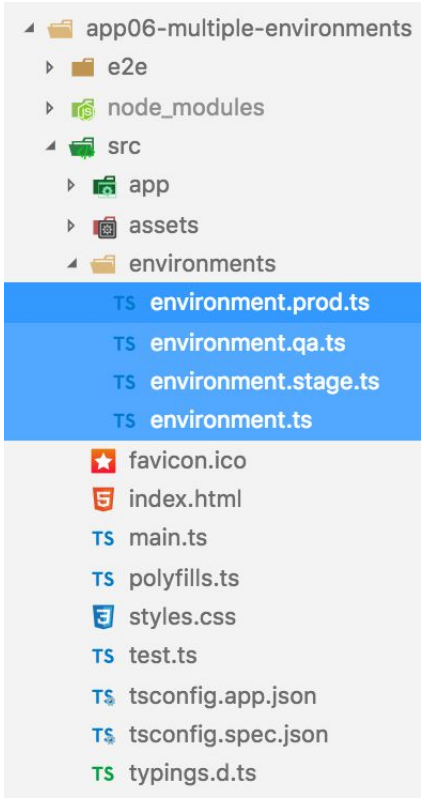
```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
@Component({
 selector: 'app-profile',
 templateUrl: './profile.component.html',
 styleUrls: ['./profile.component.css']
})
export class ProfileComponent {
 profileId: number;
 constructor(private activatedRoute: ActivatedRoute) {
   activatedRoute.params.subscribe(params => this.profileId =
params['id']);
 }
}
```

Obtaining path parameter **id** from the URL

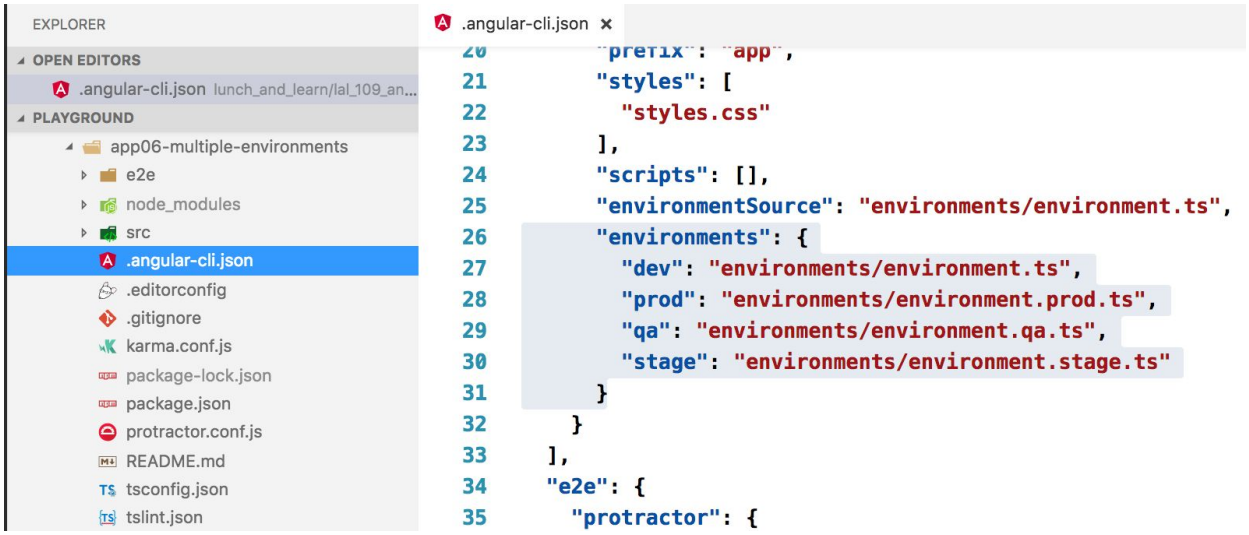
Angular Multiple Environments

https://blog.angulartraining.com/how-to-manage-different-environments-with-angular-cli-883c26e99d15

Step 1: Create multiple environment.<env name>.ts files under environment folder

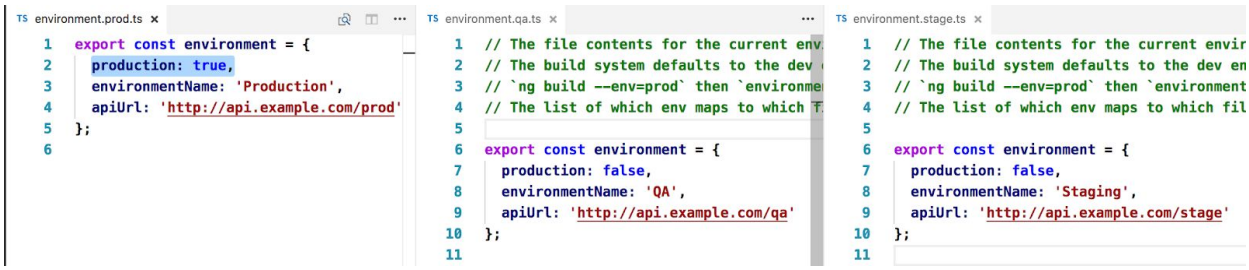


Step 2: Add environment.<env name>.ts in .angular-cli.json



Step 3: Set environment specific values in them

Set production:true for the environment that require extra minification



environment.prod.ts

```ts
export const environment = {
  production: true,
  environmentName: 'Production',
  apiUrl: 'http://api.example.com/prod'
};
```

environment.qa.ts

```ts
// The file contents for the current env
// The build system defaults to the dev
// `ng build --env=prod` then `environme
// The list of which env maps to which f

export const environment = {
  production: false,
  environmentName: 'QA',
  apiUrl: 'http://api.example.com/qa'
};
```

environment.stage.ts

```ts
// The file contents for the current envir
// The build system defaults to the dev en
// `ng build --env=prod` then `environment
// The list of which env maps to which fil

export const environment = {
  production: false,
  environmentName: 'Staging',
  apiUrl: 'http://api.example.com/stage'
};
```

Step 4: Use environment specific values in application

app.component.ts

```ts
import { Component } from '@angular/core';
import { environment } from '../environments/environment';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = `App ${environment.environmentName}`;
  serverUrl = environment.apiUrl;
}
```

app.component.html

```html
<h1>{{title}}</h1>
API URL: {{serverUrl}}
```

Step 5: Build or serve

To build environment specific build:
e.g. to build QA environment build
$ ng build --environment=qa

To serve environment specific app:
e.g. to build QA environment build
$ ng serve --environment=qa

Angular WAR

To create Java WAR and package angular application within it, we can use **frontend-maven-plugin** maven plugin.

https://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22com.github.eirslett%22%20AND%20a%3A%22frontend-maven-plugin%22

The steps below will create Java WAR + Angular

Step 1: create maven project

Create a java war project e.g. project_java using the command below

```
$ mvn archetype:generate
```

Step 2: Update java war project and pom.xml

- Convert maven's jar project to war project
- Update pom.xml to add frontend-maven-plugin and maven-resources-plugin plugins
- Add /src/main/webapp/WEB-INF/web.xml file
- Add index.html as welcome file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
version="2.5">
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

Step 3: create angular project inside maven project

```
$ cd project_java
$ ng new project_ui
```

Step 4: Add Hash bang strategy

In **app.module.ts** add HashLocationStrategy. This is done so that war paths don't conflict with angular url

```
providers: [{ provide: LocationStrategy, useClass:
HashLocationStrategy }],
```

Step 5: Create proxy for api calls

By doing this any request to http://localhost:4200/api will be handled by tomcat's http://localhost:8080/api
create proxy-config.json and add this content to it

```
{
    "/api": {
        "target": "http://localhost:8080",
        "secure": false
    }
}
```

Step 6: Add proxy in package.json

Update package.json and add proxy-config file in ng serve command.

```
"start": "ng serve --proxy-config proxy-config.json",
```

Step 7: Build and deploy java war

This will build war file deploy it for services. Deploy on tomcat that is running on port 8080

```
$ cd project_java
$ mvn clean install
```

Step 8: serve angular app

Serve angular application and start writing angular application. This will run angular application on port 4200

```
$ cd project_ui
$ npm run start
```

Angular WAR - Multiple Environment

Before doing this make angular UI of WAR file of multiple envi

DI Config Using InjectionToken

https://arturas.smorgun.com/2017/06/08/inject-environment-configuration-into-service-in-angular4.html

1. Define Service Token
2. Create Provider with the value in Module
3. Inject it in any Component or Service

Define Service Token

Create a InjectionToken in a place that is importable by module and any class that require dependency injection.
NOTE: Create it in separate Constants/Config file so that we don't get into Bi-directional or Circular import error. E.g.

// myConfig.js

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

export const SERVICE_ENDPOINT = new
InjectionToken<string>('SERVICE_ENDPOINT');
```

Create Provider with the value in Module

// app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { BlogService } from './services/blog.service';
import { SERVICE_ENDPOINT} from './myConfigs';

@NgModule({
 declarations: [AppComponent],
 imports: [BrowserModule, HttpClientModule],
 providers: [BlogService, {provide: SERVICE_ENDPOINT, useValue:
'https://jsonplaceholder.typicode.com/post'}],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

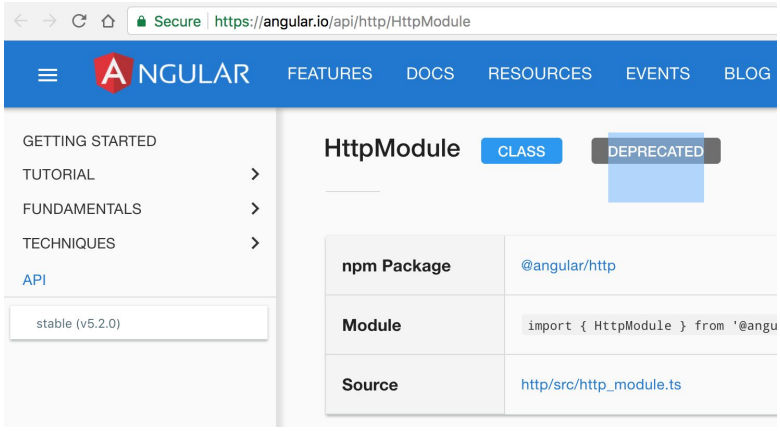
Inject it in any Component or Service

// app.component.ts

```
import { Component, Inject } from '@angular/core';
import { BlogService } from './services/blog.service';
import { SERVICE_ENDPOINT } from './myConfigs';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 posts: Array<any>;

 constructor(private blogService: BlogService,
@Inject(SERVICE_ENDPOINT) private serviceEndpoint: string) {
   console.log(serviceEndpoint);
 }

 getAllBlogPosts() {
   this.blogService.getAllBlogPosts();
 }

 createNewBlogPost() {
   this.blogService.createNewBlogPost();
 }
}
```

~~HttpModule~~ or HttpClientModule

Angular 2 HttpModule is deprecated
https://angular.io/api/http/HttpModule



With Angular 4.3 > HttpClientModule is now recommended
https://angular.io/api/common/http/HttpClientModule
it has advance feature :

- Interceptors allow middleware logic to be inserted into the pipeline
- Immutable request/response objects
- Progress events for both request upload and response download

http://brianflove.com/2017/07/21/migrating-to-http-client/
https://stackoverflow.com/questions/45129790/difference-between-http-and-httpclient-in-angular-4
https://blog.angularindepth.com/insiders-guide-into-interceptors-and-httpclient-mechanics-in-angular-103fbdb397bf

Angular + Bootstrap 4

Step 1: Install npm dependencies

Install these 2 components and npm dependencies

```
"@ng-bootstrap/ng-bootstrap": "^1.0.1",
"bootstrap": "^4.0.0",
```

Step 2: Import NgbModule

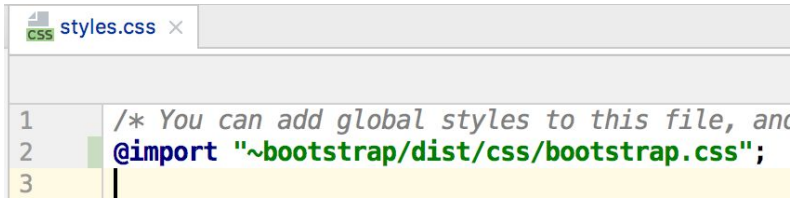
Add NgbModule.*forRoot*() module in app.module.ts. If any other child module require bootstrap 4 then just import NgbModule module.

```
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';
import {LocationStrategy, HashLocationStrategy} from "@angular/common";
```

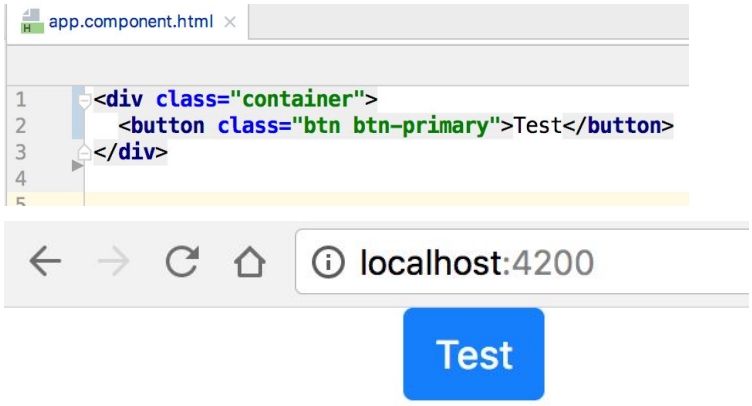
```
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';

import {AppComponent} from './app.component';
import {HttpClientModule} from "@angular/common/http";


@NgModule({
 declarations: [AppComponent],
 imports: [BrowserModule, HttpClientModule, NgbModule.forRoot()],
 providers: [{provide: LocationStrategy, useClass: HashLocationStrategy}],
 bootstrap: [AppComponent]
})
export class AppModule {
}
```

Step 3: Import bootstrap css

```
@import "~bootstrap/dist/css/bootstrap.css";
```



Step 4: Start using bootstrap classes



Router, sub Router, Login, Guard, Roles

ng generate module components

ng generate component components/public-menu
ng generate component components/admin-menu
ng generate component components/customer-menu
ng generate component components/about
ng generate component components/login
ng generate component components/view-items
ng generate component components/add-item
ng generate component components/edit-item

ng generate module layouts
ng generate component layouts/main
ng generate component layouts/sub-login

ng generate module services
ng generate service services/authentication/authentication
ng generate guard services/guard/authentication
ng generate class services/modal/SessionUser

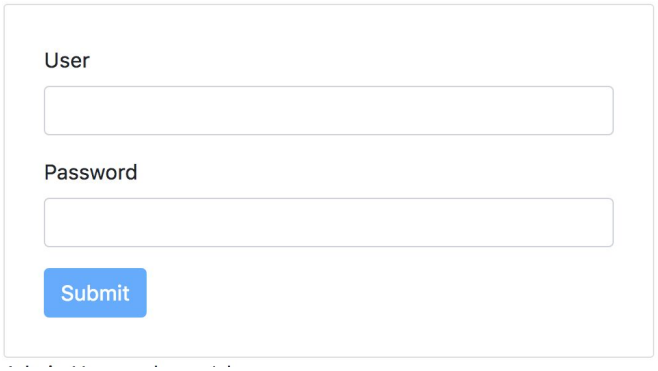
$ cd src/app/services
$ mkdir authentication
$ mkdir guard
$ mkdir model

Template (HTML) Driven Form

Template driven form are used for creating simple form. Below are all defined in the template (HTML)

- Form submission
- Input grabbing
- Input and form validation
- Display field and form errors

Example below creates this form:



login-component.html

```
<form #loginForm="ngForm" (submit)="login(loginForm)">
 <div style="border: 1px solid #dedede; padding: 30px; width: 500px; border-radius: 3px; margin-top: 40px">
   <div class="text-danger" *ngIf="loginFailed">
    Login Failed!
   </div>
   <div class="form-group">
    <label for="userId">User</label>
    <input ngModel required
           [ngClass]="{'is-invalid': userId.invalid && userId.touched}"
           name="userId"
           #userId="ngModel"
           id="userId" type="text" class="form-control">
    <div class="invalid-feedback">
     Please enter User Id
    </div>
   </div>
   <div class="form-group">
    <label for="password">Password</label>
    <input ngModel required
           [ngClass]="{'is-invalid': password.invalid && password.touched}"
           name="password"
           #password="ngModel"
           id="password" type="password" class="form-control">
    <div class="invalid-feedback">
     Please enter password
    </div>
   </div>
   <div>
    <button class="btn btn-primary" [disabled]="loginForm.invalid">Submit</button>
   </div>
{{loginForm.errors}}
```

```
  <div class="text-danger" style="margin-top: 30px;" *ngIf="loginForm.touched &&
loginForm.invalid">
    Please enter User Id and password
  </div>
 </div>
</form>
```

Angular automatically adds ngForm directive to any <form> tag it sees. We can use it by adding it to #template variable #loginForm="ngForm"

To all the inputs we need to add ngModel and can add it to #template variable #userId="ngModel"

ngModel validates against all HTML5 input validation rules.
https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation

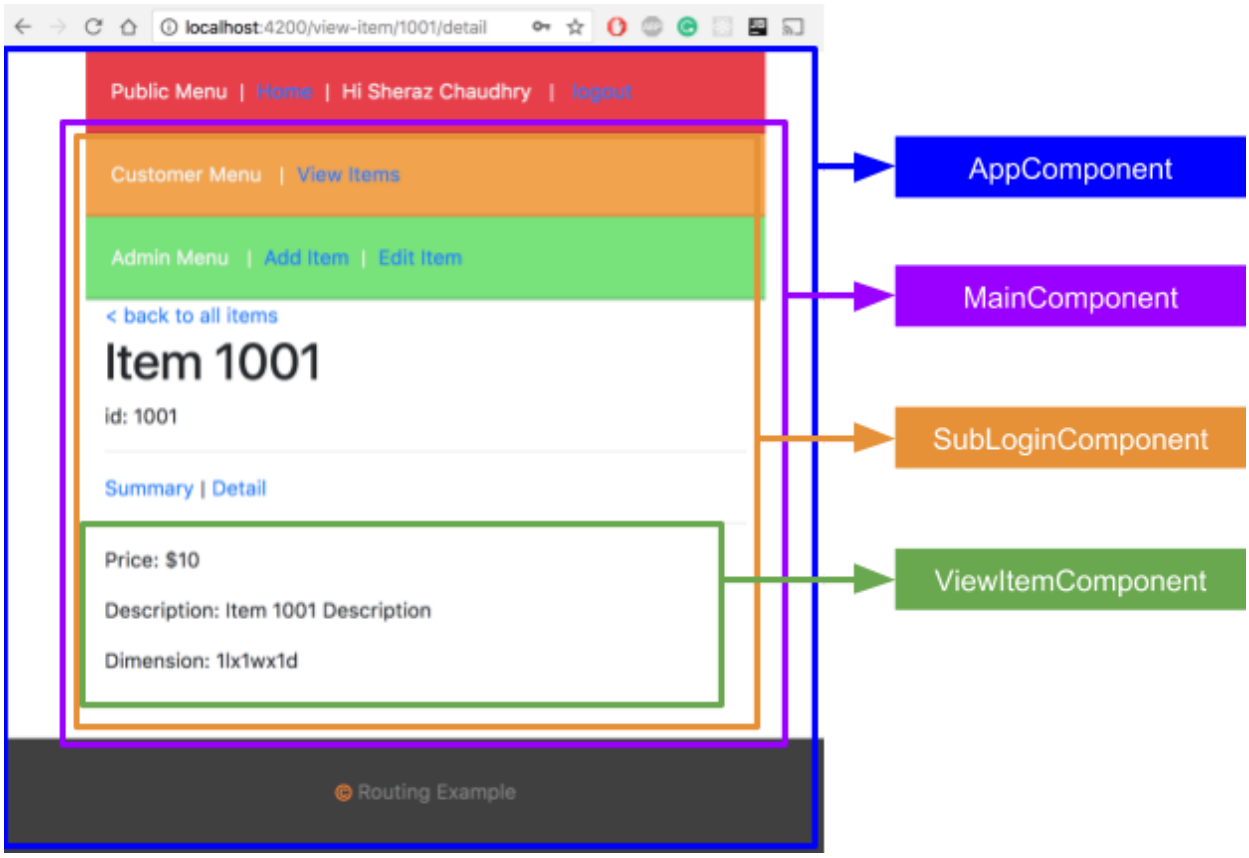
login-component.ts

```
import {Component} from '@angular/core';
import {AuthenticationService} from '../../services/authentication/authentication.service';
import {NgForm} from '@angular/forms';
import {SessionUser} from '../../services/modal/session-user';
import {Router} from '@angular/router';

@Component({
 selector: 'login',
 templateUrl: './login.component.html',
 styleUrls: ['./login.component.css']
})
export class LoginComponent {

 loginFailed: boolean;

 constructor(private authenticationService: AuthenticationService, private route: Router) {}

 login(loginForm: NgForm) {
  this.loginFailed = false;
  const sessionUser: SessionUser = this.authenticationService.login(loginForm.value.userId,
loginForm.value.password);
  if (sessionUser) {
   this.route.navigate(['/view-items']);
  } else {
   this.loginFailed = true;
  }
```

```
}
}
```

Sub Routes

Sub routes are created to create multiple levels of replaceable sections and map it to a URL. In **Routes** definition to create sub replaceable sections we have to add **children** routes. Any component that contains children routes should have <**router-outlet**></**router-outlet**>.



app-routes.ts

```
import {RouterModule, Routes} from '@angular/router';
import {MainComponent} from './layouts/main/main.component';
import {LoginComponent} from './components/login/login.component';
import {SubLoginComponent} from './layouts/sub-login/sub-login.component';
import {ViewItemsComponent} from './components/view-items/view-items.component';
import {AddItemComponent} from './components/add-item/add-item.component';
import {EditItemComponent} from './components/edit-item/edit-item.component';
import {ViewItemComponent} from './components/view-item/view-item.component';
```

```
import {ViewItemSummaryComponent} from
'./components/view-item-summary/view-item-summary.component';
import {ViewItemDetailComponent} from
'./components/view-item-detail/view-item-detail.component';
import {AuthenticationGuard} from './services/guard/authentication.guard';
import {UnAuthorizedComponent} from
'./components/un-authorized/un-authorized.component';
import {NotFoundComponent} from './components/not-found/not-found.component';
import {AuthorizationAdminGuard} from './services/guard/authorization-admin.guard';

const routes: Routes = [
 {
   path: '', component: MainComponent,
   children: [
    {
      path: '', component: SubLoginComponent, canActivate: [AuthenticationGuard],
      children: [
       {path: '', component: ViewItemsComponent},
       {path: 'view-items', component: ViewItemsComponent},
       {path: 'view-item/:itemId', component: ViewItemComponent,
         children: [
           {path: '', redirectTo: 'summary', pathMatch: 'full'},
           {path: 'summary', component: ViewItemSummaryComponent},
           {path: 'detail', component: ViewItemDetailComponent}
         ]
       },
       {path: 'add-item', component: AddItemComponent, canActivate:
[AuthorizationAdminGuard]},
       {path: 'edit-item/:itemId', component: EditItemComponent, canActivate:
[AuthorizationAdminGuard]}
      ]
    },
    {path: 'login', component: LoginComponent},
    {path: 'un-authorized', component: UnAuthorizedComponent},
    {path: '**', component: NotFoundComponent}
   ]
 }
];

export const routing = RouterModule.forRoot(routes);
```

Guards

Guards are used

authentication.guard.ts

```
import { Injectable } from '@angular/core';
import {CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router} from
'@angular/router';
import { Observable } from 'rxjs/Observable';
import {AuthenticationService} from '../authentication/authentication.service';

@Injectable()
export class AuthenticationGuard implements CanActivate {

 constructor(private authenticationService: AuthenticationService, private route: Router) {}

 canActivate(
   next: ActivatedRouteSnapshot,
   state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {
   if (!this.authenticationService.currentUser()) {
     this.route.navigate(['login']);
     return false;
   } else {
     return true;
   }
 }
}
```