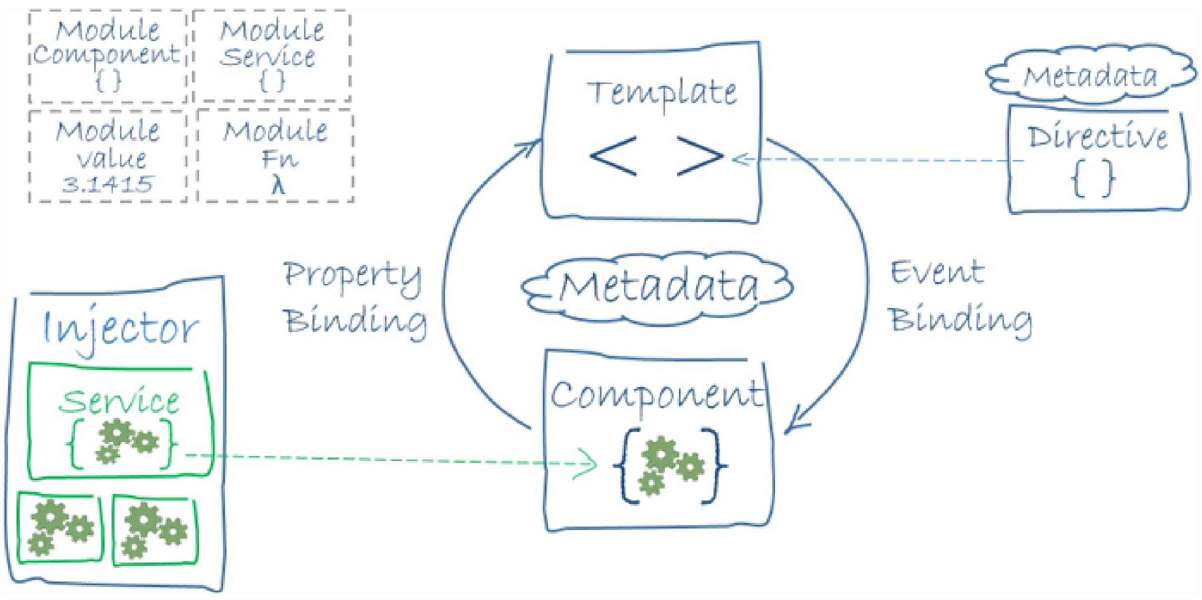
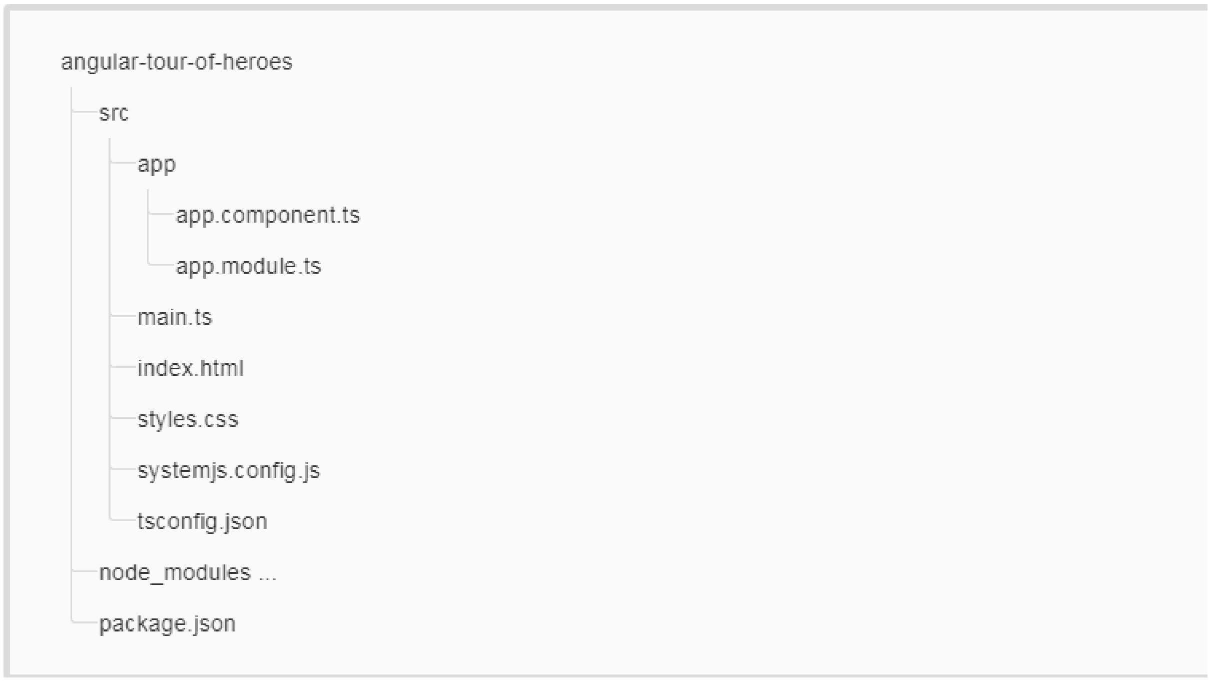
Angular 2 Architecture:



Angular CLI use full commands:

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`

Project folder structure:



```
angular-tour-of-heroes
  src
    app
      app.component.ts
      app.module.ts
    main.ts
    index.html
    styles.css
    systemjs.config.js
    tsconfig.json
  node_modules ...
  package.json
```

Custom Component:

Component = directive + template

Creating component manually

First create component file with ts extension.

@Component defines component

For a component you need two mandatory things

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

@Component({

        selector:'hello'

        template:`some thing`

})

export class HelloComponent{}
```

---------------------------------mycomponent.ts------------------------------

Now handover this component to module.

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

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


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


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

---module---

In html give the directive

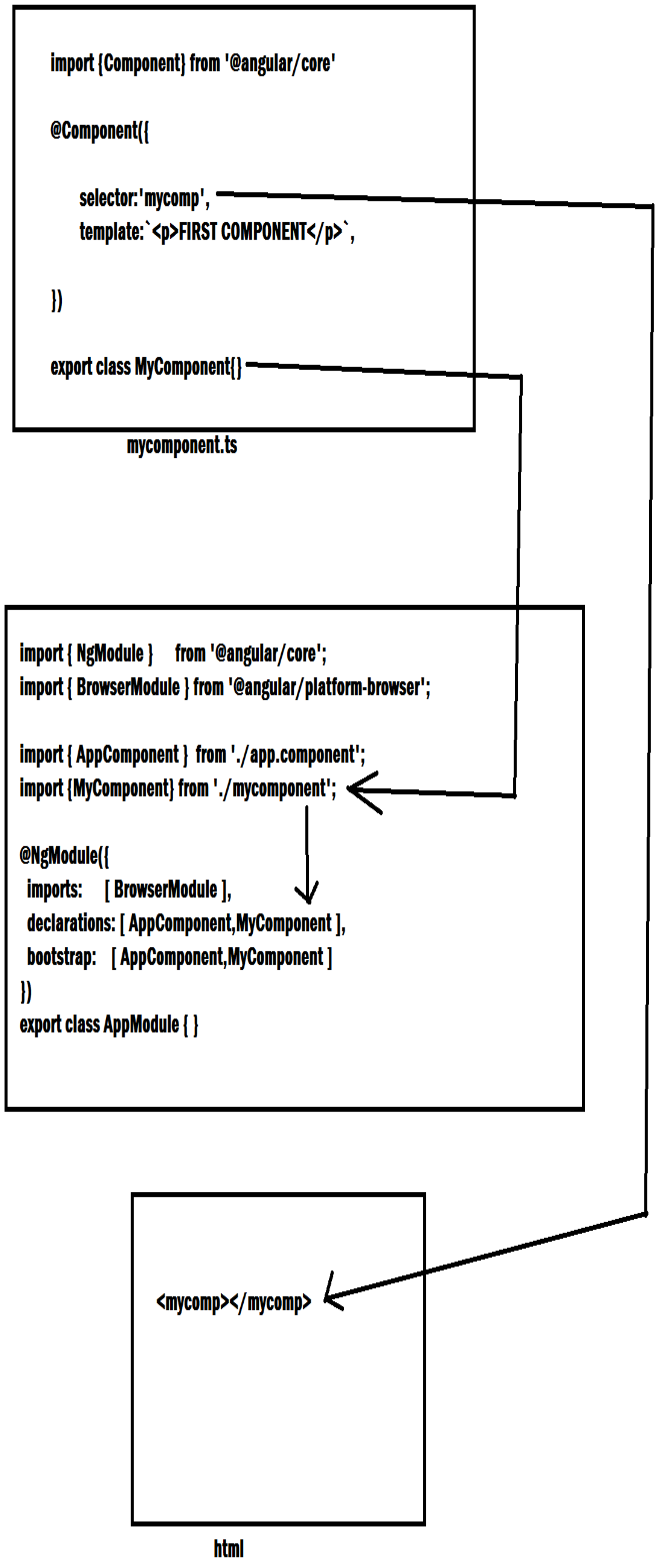
```
<first>HI</first>
```

-------------------------------index.html---

Follow below diagram for quick understanding.

But if your using angular client no need put this much effort

Just type ng generate component name



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

@Component({

    selector:'mycomp',
    template:`<p>FIRST COMPONENT</p>`,

})

export class MyComponent{}
```

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

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

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

```
<mycomp></mycomp>
```

Directives:

A Directive is set of functionalities add an additional behaviour to HTML elements in Angular applications

*ngIf:

The ngIf Directives is used to add or remove HTML Elements according to the expression. The expression must return a Boolean value. If the expression is false then the element is removed, otherwise element is inserted.

<div *ngIf=”condition”>

{{value}}

</div>

*NgFor :

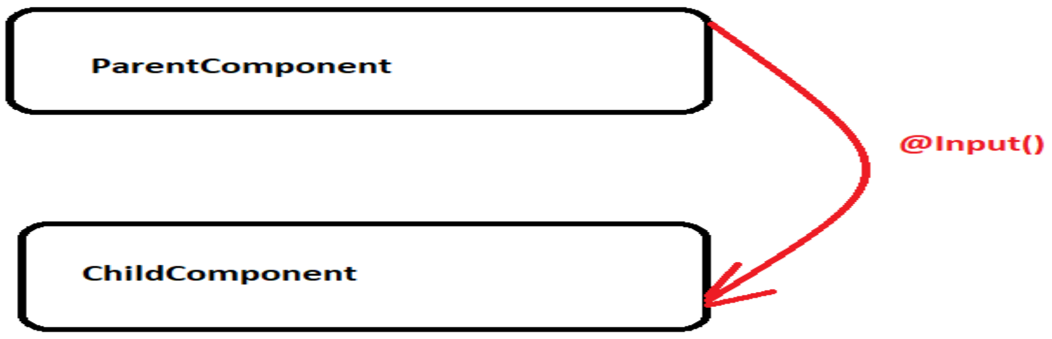
`NgFor` directive to iterate a list of items.

To iterate list of items following syntax **`let item of items`**.

<div *ngFor="let item of items">{{item.name}}</div>

@Inputs:

Whenever we need to transfer data from Parent component to Child components, we will use Inputs. Here we are taking username is input inputs are defined like [username].



Considering App Component as Parent Component & Demo Component as Child Component.

```
@Component({
    selector:'demo-component',
    templateUrl:'democomponent.html'
})
export class DemoComponent{

    @Input()
    username:string = '';

}
```

DemoComponent.ts

```
<h1>Demo Component - Child Compoent</h1>

<h1>{{username}}</h1>
```

democomponent.html

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

}
```

App.component.ts

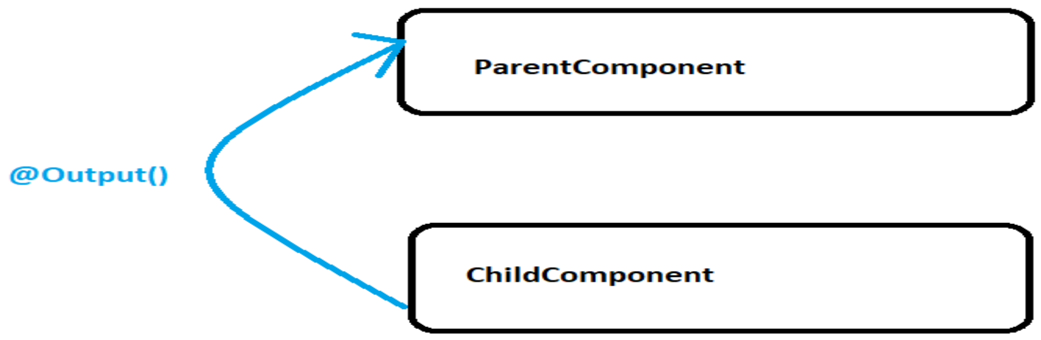
```
<h1>App Component - Parent Component</h1>
<input type="text" #usr (keyup)="0">
<demo-component [username]="usr.value"></demo-component>
```

App.component.html

Note: Don't bootstrap Child Component (Demo-Component) in app.module.ts.

@Output():

Angular outputs are applicable to transfer the data from child component to parent component.



Considering App Component as Parent Component & Demo Component as Child Component.

```
import { Component, EventEmitter, Input, Output } from "@angular/core";

@Component({
    selector:'demo-component',
    templateUrl:'democomponent.html'
})
export class DemoComponent{
    @Output()
    greet = new EventEmitter();

    saygreet(nm:string){
        this.greet.emit(nm);
    }
}
```

Democomponent.ts

```
<h1>Demo Component - Child Compoent</h1>

<input type="text"  #usnam (keyup)="saygreet(usnam.value)">
```

Democomponent.html

```
export class AppComponent {
  tmp:string = '';
  display(temp:string){
      this.tmp = temp;
  }
}
```

Appcomponent.ts

```
<h1>App Component - Parent Component</h1>
<h1>{{tmp}}</h1>
<demo-component (greet)="display($event)"></demo-component>
```

App.component.html

Greet is output which will emit the value from child component to parent component.

Services:

Angular Services are applicable to separate the business logic from component. Services avoid tight coupling of business logic with component.

To generate service use ng g s demo

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

@Injectable({
  providedIn:'root'
})
export class DemoService {
  greet(){
    return "Hello from service";
  }
}
```

Demoservice.ts

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers:[]
})
export class AppComponent {
  temp:any;
  constructor(private demoservice:DemoService){

  }
  display(){
    this.temp = this.demoservice.greet();
  }
}
```

Appcompoent.ts

```
<h1>App Component</h1>
{{temp}}

<input type="button" (click)="display()">
```

App.compoent.html

Service can be Injected in 3 ways.

1.Component Level

2.Module level

3.Global

Component Level Injection:

We will define service in providers of component decorator.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers:[DemoService]
})
```

Module Level Injection:

We will define service in providers of Module decorator. Then all the components declared in this module have access for Service.

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

app.module.ts

Global Injection:

We will define provided in root in Injectable decorator.

```
@Injectable({
  providedIn:'root'
})
export class DemoService {

  greet(){
    return "Hello from service";
  }
}
```

Demoservice.ts

Angular 2 Forms:

Angular supports two types of forms Template Drive & Reactive Forms.

Template Driven or Model Driven Forms

Here Iam discussing about template driven forms:

Here we are declaring a local variable called "form" and setting it to an instance of ngForm. This is very important. Now our local form variable becomes of type FormGroup allowing us access to the FormGroup API's on this local variable. We use this in the ngSubmit event where we send the value of the form via form.value

Since we are working with template driven forms, we can use the ngModel directive to capture the values of our forms. One thing to note if you are coming from Angular 1.x. Using ngModel as shown below creates a one-way data binding, so once we hit submit the data is only sent to the controller. If we wanted to use two-way data binding, we would have to wrap the ngModel in [()] and assign an attribute to it. Also the name of the field corresponds to the name attribute so our first input will be firstName.

Component:

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

import { HelloComponent } from './hello/hello.component';

@Component({

 selector: 'app-root',

 template: `<h1>FORMS DEMO</h1>

 <form #form="ngForm" (ngSubmit)="submitForm(form.value)">

   <div class="form-group">

    <label>First Name:</label>


    <input type="text" class="form-control"  name="firstName" ngModel required>

   </div>

   <div class="form-group">

    <label>Last Name</label>

    <input type="text" class="form-control" name="lastName" ngModel required>

   </div>

   <div class="form-group">

    <label>Gender</label>

   </div>
   <div class="radio">
     <label>
       <input type="radio" name="gender" value="Male" ngModel>
       Male
     </label>
   </div>
   <div class="radio">
     <label>
       <input type="radio" name="gender" value="Female" ngModel>
       Female
     </label>
   </div>
   <div class="form-group">
     <label>COURCES</label>
   </div>
   <label class="checkbox-inline">
     <input type="checkbox" value="angular" name="angular" ngModel> angular
   </label>
   <label class="checkbox-inline">
     <input type="checkbox" value="spring" name="spring" ngModel> spring
   </label>
   <label class="checkbox-inline">
     <input type="checkbox" value="phython" name="phython" ngModel> phython
   </label>
   <div class="form-group">
     <button type="submit" class="btn btn-default">Submit</button>
   </div>
 </form>
`,
          inputs:[`date`]
```

```
})
export class AppComponent {
            public date:Objec;t
 submitForm(form: any): void{
   console.log('Form Data: ');
   console.log(form);
 }
}
```

And now its time to go to module in module you need to import formsmodule from @angular/forms:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms'
import { AppComponent } from './app.component';
import { HelloComponent } from './hello/hello.component';
@NgModule({
 declarations: [
   AppComponent,
   HelloComponent
 ],
 imports: [
   BrowserModule,FormsModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

now your form was ready.

Angular 2 Form Example 2 with hands on implementation:

Here I am giving only component:

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

import { HelloComponent } from './hello/hello.component';

@Component({

  selector: 'app-root',

  template: `<h1>FORMS DEMO</h1>

  <form #demoForm="ngForm" (ngSubmit)="myfunction(demoForm.value)">

  <div>NAME<input type="text" name="name" ngModel></div>

  <div>ADDRESS<input type="text" name="addr" ngModel></div>

  <div>PHONE<input type="number" name="pno" ngModel></div>

  <div>EMAIL<input type="text" name="email" ngModel></div>

  <input type="submit">


  </form>

  `,


})

export class AppComponent {

        myfunction(result){

                console.log(result);


        }

}
```

If you observe above code snippet by comparing normal form you will get good understand in angular 2 form.

Two-way binding in angualar2:

You need to specify like [(ngModel)]=”name”

Angular2-states:

STATE	CLASS	CLASS
Controller has visited	ng-touched(visited)	ng-untouched (notvisited)
Value changed	ng-pristine(notchanged)	ng-dirty (changed)
Valid	ng-invalid (notvalid)	ng-valid (valid)

To see the state and applied class for each field put an id for field and getclass name like below:

```
<input type="text" #nameRef  name="name" ngModel>{{nameRef.className}}
```

Initially you can see these class:

ng-untouched ng-pristine ng-valid

once you changed the value now you are going to see:

ng-valid ng-dirty ng-touched

component with validation:

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

import { HelloComponent } from './hello/hello.component';

@Component({

  selector: 'app-root',

  template: `<h1>FORMS DEMO</h1>

  <form #demoForm="ngForm" (ngSubmit)="myfunction(demoForm.value)">

  <div>NAME<input type="text" #nameRef required name="name"
ngModel>{{nameRef.className}}</div>

  <div>ADDRESS<input type="text" required name="addr" ngModel></div>

  <div>PHONE<input type="number" required name="pno" ngModel></div>

  <div>EMAIL<input type="email" required name="email" ngModel></div>

  <input type="submit">


  </form>

  `,

        styles:[`input.ng-invalid{background:red;}`]
})
export class AppComponent {
public result1:string;

        myfunction(result){
                console.log(result);

        }
}
```

Applying two styles:

```
import { Component } from '@angular/core';
import { HelloComponent } from './hello/hello.component';
@Component({
  selector: 'app-root',
  template: `<h1>FORMS DEMO</h1>
  <form #demoForm="ngForm" (ngSubmit)="myfunction(demoForm.value)">
  <div>NAME<input type="text" #nameRef required name="name"
ngModel>{{nameRef.className}}</div>
  <div>ADDRESS<input type="text" required name="addr" ngModel></div>
  <div>PHONE<input type="number" required name="pno" ngModel></div>
  <div>EMAIL<input type="email" required name="email" ngModel></div>
  <input type="submit">

  </form>
  `,
        styles:[`
        input.ng-invalid{border-left:5px solid red;}.ng-valid{border-left:5px solid green;};


        `]
})

export class AppComponent {

public result1:string;


        myfunction(result){

                console.log(result);


        }

}
```

Reactive Forms:

Reactive forms provide a model-driven approach to handling form inputs whose values change over time

Reactive forms use an explicit and immutable approach to managing the state of a form at a given point in time. Each change to the form state returns a new state, which maintains the integrity of the model between changes. Reactive forms are built around observable streams, where form inputs and values are provided as streams of input values, which can be accessed synchronously.

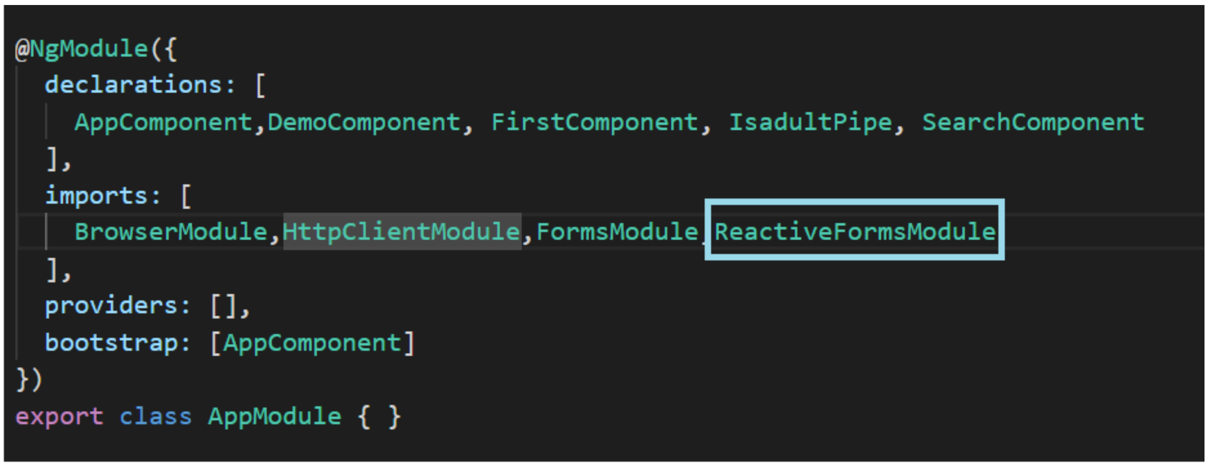
To use reactive form controls, import ReactiveFormsModule from the `@angular/forms` package and add it to your NgModule's `imports` array.

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{
  signupform:FormGroup = new FormGroup({});
  ngOnInit(){
    this.signupform = new FormGroup({
          'usr':new FormControl(null,Validators.required),
          'email':new
FormControl(null,[Validators.requiredTrue,Validators.email]),
          'pwd':new FormControl
    });
  }
  submitdata(){
    console.log(this.signupform);
  }

}
```

app.component.ts

```
<h1>App Component</h1>
<form>
    <div class="form-group">
    Name <input type="text" name="usr" class="form-control">
    </div>
    <div class="form-group">
    Email <input type="text" name="email" class="form-control">
    </div>
    <div class="form-group">
    password<input type="password" name="pwd" class="form-control">
    </div>
    <button type="submit" class="btn btn-success">Submit</button>
</form>
```

App.component.html



```
@NgModule({
  declarations: [
    AppComponent,DemoComponent, FirstComponent, IsadultPipe, SearchComponent
  ],
  imports: [
    BrowserModule,HttpClientModule,FormsModule,ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

App.module.ts

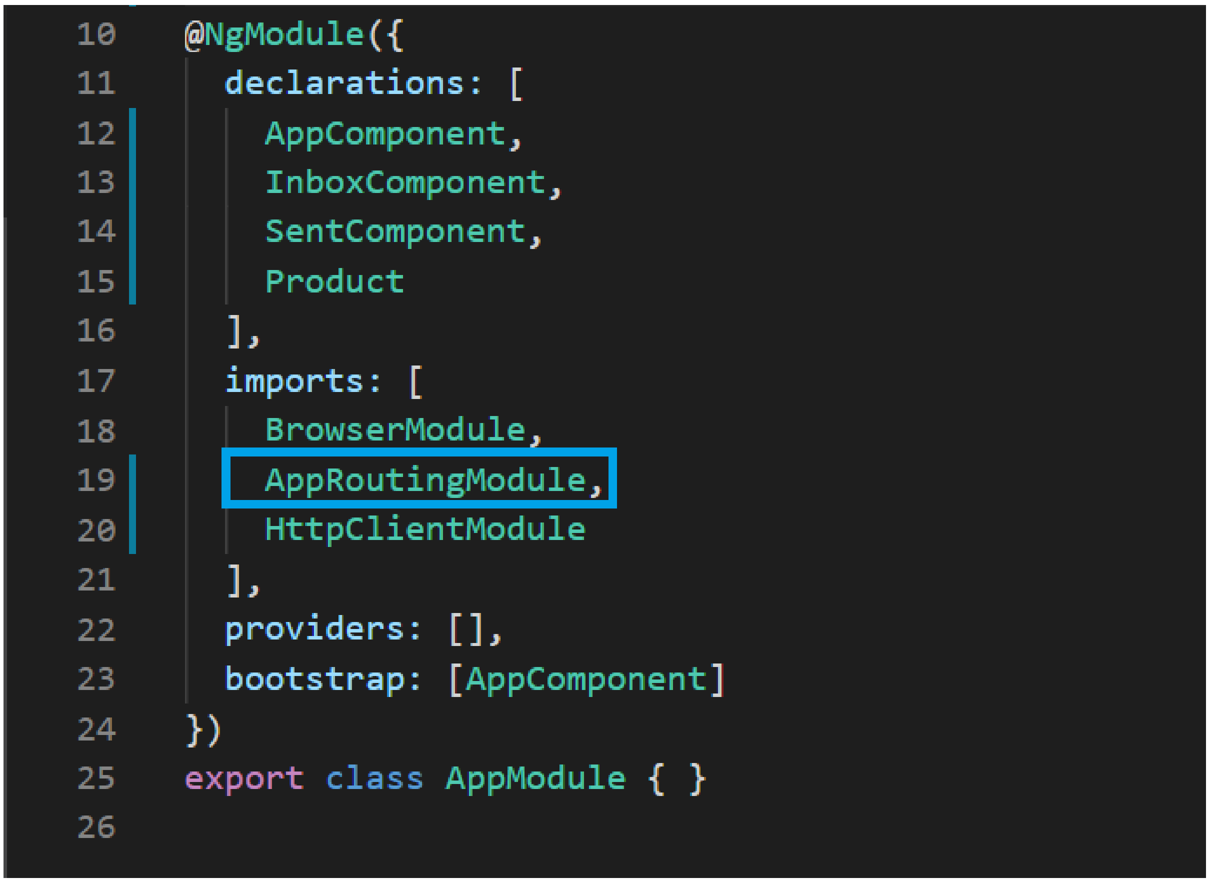
Routers:

Routers are applicable to implement navigation in Angular.

There are three fundamental building blocks to creating a route.

Import the `AppRoutingModule` into `AppModule` and add it to the `imports` array.

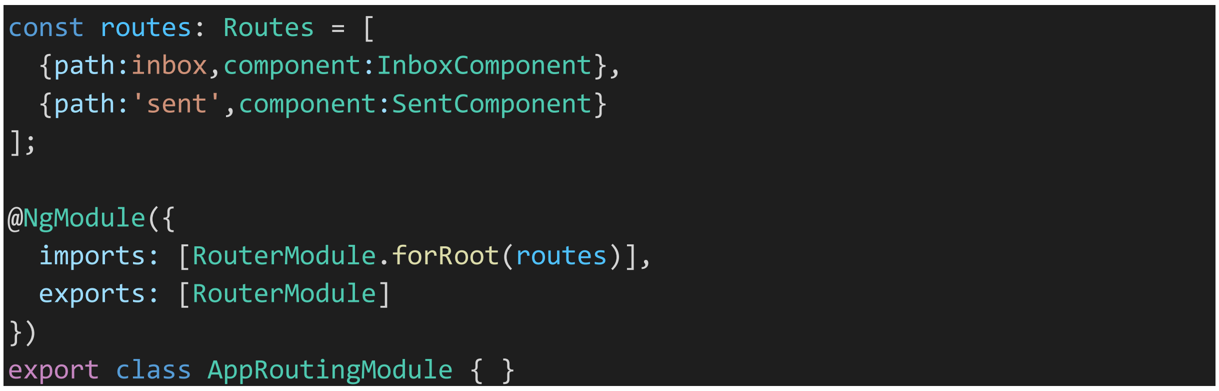
The Angular CLI performs this step for you. However, if you are creating an application manually or working with an existing, non-CLI application, verify that the imports and configuration are correct. The following is the default `AppModule` using the CLI with the `--routing` flag.



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

```

App-routing.module.ts

Define paths in app.routing.module.ts



```
const routes: Routes = [
  {path:inbox,component:InboxComponent},
  {path:'sent',component:SentComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

app.routing.module.ts

Now define router-outlet to display routers wherever we want to display our router output.

```html
<h1>Router Demo</h1>
<button><a routerLink="/inbox">INBOX</a></button>
<button><a routerLink="/sent">SENT</a></button>
<router-outlet></router-outlet>
```

app.component.html