Part One

Section One

` 1.
- **Angular is a tool that you can use to build web applications.**

2.
- **Angular has 3 building block:(CSM)**
    - **Component :**
        - **a component is what a user can see and interacts with.**
        - **Data binding b/n inpute fields and the data that is stored.**
        - **we can have many component.**
        - **you create a component for functionality.**
    - **Services:**
        - **Services are here to handle data.**
        - **we use services to communicate with "Restapi" to load data via a data base and distribute that data to all of our components.**
    - **Modules:**
        - **we use modules for encapsulation so that our entire application is nice and structured.**
        - **we use modules to suture our angular application in a nice and easy way.**

3. Angular Application is a single page application.
- **when we first load the page we load it one time , the entire application, then afterwards when we got to another page in our application add something to our application only the changes are getting updated.**
- **this reduces the load times.**

4.Angular application runs on a webserver.
- **if we run it locally it is called a localhost.**

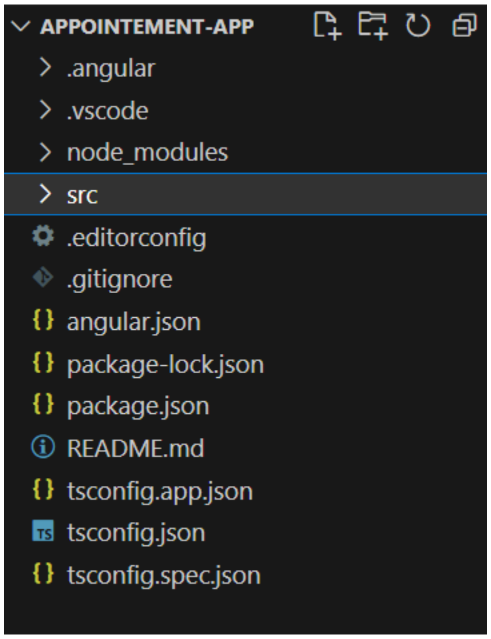
5. To get started with Angular we need the angular CLI(command lie interface)
- **The Agular CLI is a node package thus we need to install NodeJS(a way to run JavaScript and typescript on a webservice) and npm.**
- **once you downloaded Nodejs and installed it you can check if it is installed on the cmd using "node -v".**

Section Two

To create a new Angular project we can use the Angular CLI and use the command "ng new <name of project>". (Note: for your project name use lower case and connect every new word with a "-")
once you create the project go into it using the "cd" command
you can open the project from in visual studio using "code ." command
Then go to the terminal and go into the project folder and use the "ng serve -o" to start you project and also open your default browser to open the path.

Section Three

- **In this sections we will to at the files in the appointment-app Angular application.**

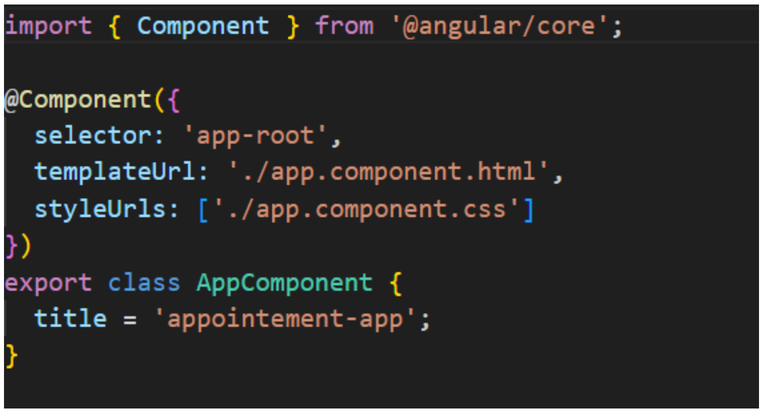


```
APPOINTEMENT-APP
  > .angular
  > .vscode
  > node_modules
  > src
    .editorconfig
    .gitignore
    angular.json
    package-lock.json
    package.json
    README.md
    tsconfig.app.json
    tsconfig.json
    tsconfig.spec.json
```

Src folder:

- **angular.json:**
    - **Here we cans et up some configuration for our Angular application.**
    - **One of the most important section is the "configuration" . For instance if we are serving for different environments for example like development environment, staging environment, and production environment we can specify specific environment for each of those environments.**
- **Package.json:**
    - **related NodeJS and and npm.**
    - **the most important par tis the "dependencies". if we hand over our application to another developer we will not send all the node modules extra, we wills send a very simple version of the application. He will then use NodeJs to write down "npm install " to install all of the dependencies that we see under the "dependencies" section.**
- **gitignore:**
    - **for version controll.**
    - **all the files that are getting ingored by the version controll system like "git".**

- **assets folder:**
    - **index.html:**
        - **the index.html in a web application is typically the main page.**
        - **in this html page under the <body></body> section we are injecting <app-route ></app-route>.**
        - **<app-root></app-root> is the main root where our application, where basically all of the action takes place.**
- **app folder:**
    - **Our Angluar CLI have automatically created one component .(components are one of the 3 building blocks which are what the uses sees and interacts with)**
    - **our application has one component by default ,our root component, which is "app" component.**
        - **app.component.ts files(ts is type script):**
            - **this is the first componet that authomathically got created.**



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

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

- **notice the decorateor @Componet.**
- **we also see that it is a type script class and its name is "AppComponet".**
- **the @Componet class turn the type script class AppComponet into a component.**
- **There are 3 important things that we see in the @Component decorator:**
    - **selector property: species the html tag you can use to render this component.**
      *** in our case the selector is name 'app-root'**
    - **templateUrl property: specifies the path to the HTML file associated with the component.**
      *** When you create a new angular project, angular automatically creates an HTML file called `app.component.html` in the app folder.**
    - **style Url properrty:**
      ***specifies the path to the CSS file associated with the component.**

- **app.componets.spec.ts:**
    - **this file is for unit testing we will get to this letter on.**

2. **.angular folder:**
    - **for configuration and meta files**
3. **.Vscode folder:**
    - **related to the code editor.**

4. **node_modules folder:**
   - **cotains all of our packages and dependencies.**

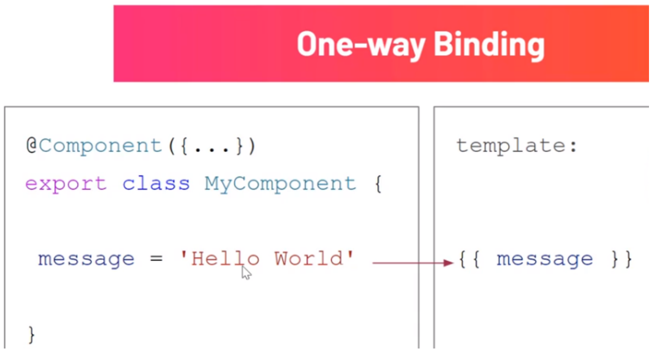
PART TWO

Section One

1.Components:

- components are one of the main building blocks of your Angular application.
- when ever you add a new functionality you will most likely create a component.
- the @Componet decorator is what turns the typescript class into a component.

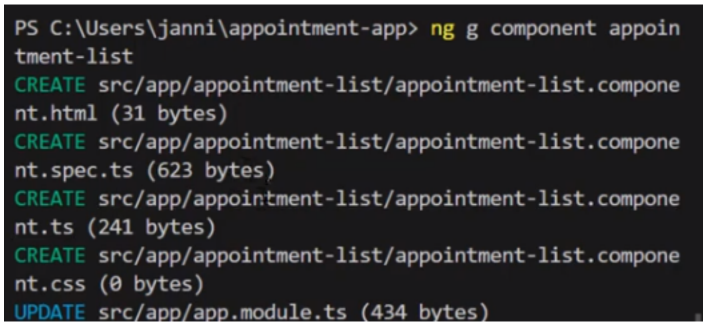
- Data Binding:
  - One way data binding:
    - **One-way data binding will bind the data from the component to the view (DOM) or from view to the component.**
    - allows us to display the content of the property in our template(html).
    - once we are in the template we use Two curly brackets (double curly brasses are the interpolation syntax for one way data binding)
    - **One-way data binding is unidirectional. You can only bind the data from component to the view or from view to the component.**



Section Two

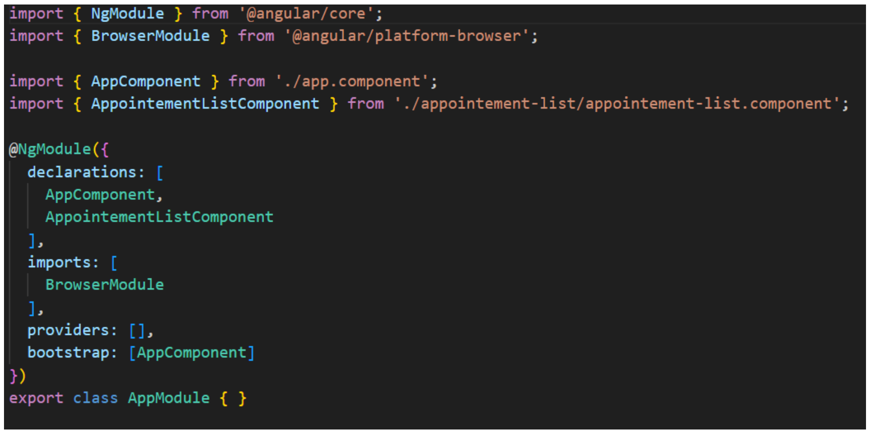
1 . Creating a new component:

- To create a new Component we will need to use the "ng command".
- since you are using the current terminal to the the Angular application crate a second terminal to execute the new command.(terminal -> split terminal).
- The command to create a new Component: "ng g component <name of component>".



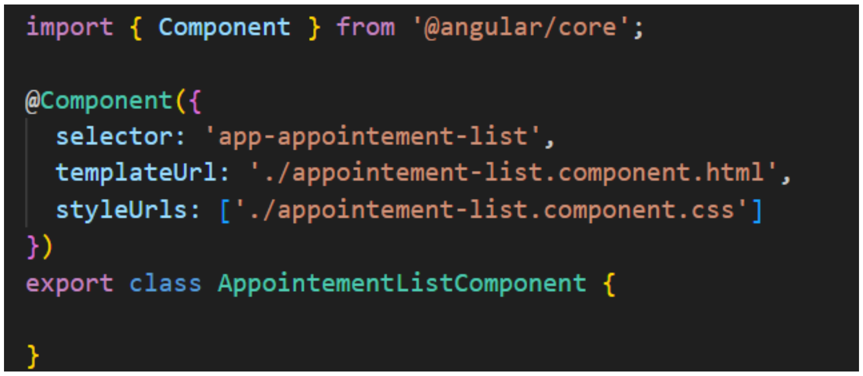
```
PS C:\Users\janni\appointment-app> ng g component appoin
tment-list
CREATE src/app/appointment-list/appointment-list.compone
nt.html (31 bytes)
CREATE src/app/appointment-list/appointment-list.compone
nt.spec.ts (623 bytes)
CREATE src/app/appointment-list/appointment-list.compone
nt.ts (241 bytes)
CREATE src/app/appointment-list/appointment-list.compone
nt.css (0 bytes)
UPDATE src/app/app.module.ts (434 bytes)
```

- **when we execute this command we see the following output:**
  - **As we can see the following are created:** appointemnet-list.component.html,appointemnet-list.component.spec.ts,appointemnet-list.component.ts,appointemnet-list.component.css
  - we also can see that the "app.module.ts" file , which is uder src/app directory, is updated.
  - if we open our app.module.ts file class we see the following.



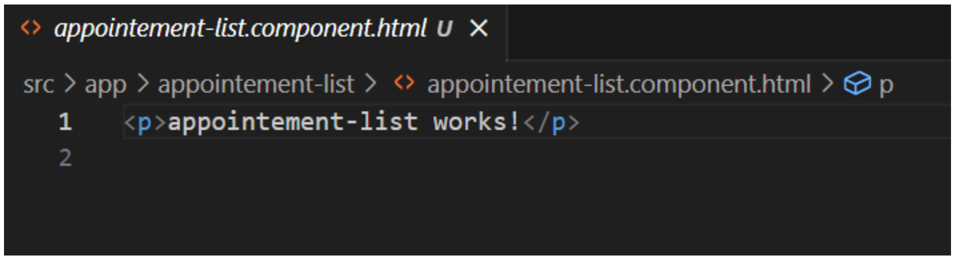
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { AppointementListComponent } from './appointement-list/appointement-list.component';

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

  - on the tope we see that the our "AppointementListComponet" class is being imported.
  - app.module.ts is the main module for our application.
  - we also see that the our "AppointementListCoponet" class is declared under the declaration section. on the top level this means that "AppointementListComponent" is declared/related to our app.module.
  - A component is delarcread inside a single module.
- Now if you go to "appointemnet-list.component.ts" we see the following:



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

@Component({
  selector: 'app-appointement-list',
  templateUrl: './appointement-list.component.html',
  styleUrls: ['./appointement-list.component.css']
})
export class AppointementListComponent {

}
```

- Thus we see that the selector(the html tag) for our new component is "app-appointment-list"
- now if we go to our appointemnet-list.component.html we see the following:

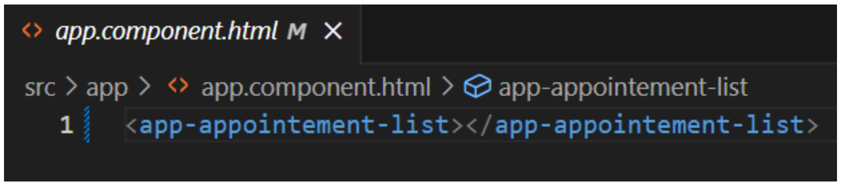


appointement-list.component.html U ×

src > app > appointement-list > appointement-list.component.html > p

```
<p>appointement-list works!</p>
```

- The question is how can we display this ?
- if you recall we said that the index,html is the home page of our application. and inside of out index.html we specified in the body the selector of the app component which is <app-root>.The app component is the main componet of our applications.Thus we go to "app.compont.html" files= and insert the selector of the appointemnet-list.component. as follows to view the appointemnet-list.component.



app.component.html M ×

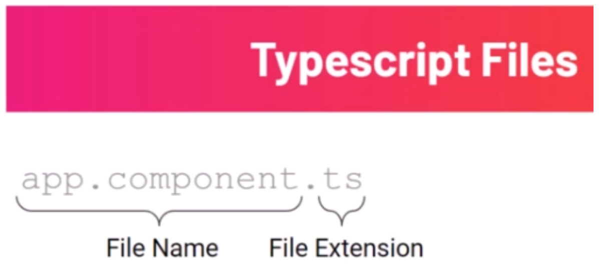
src > app > app.component.html > app-appointement-list

```
<app-appointement-list></app-appointement-list>
```

Section Three

Type Script:

1.file name



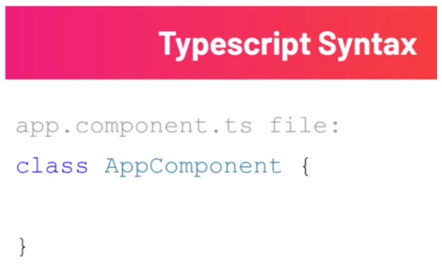
Typescript Files

$\underbrace{\texttt{app.component}}_{\text{File Name}}\underbrace{\texttt{.ts}}_{\text{File Extension}}$

2 Class:

- **in type script functions,variable ,propertires all go inside of the class(in between the curly braces).so code related to the class goes inside of the curly braces.**
- **type script uses PadcalCase and camelCase naming convention.**

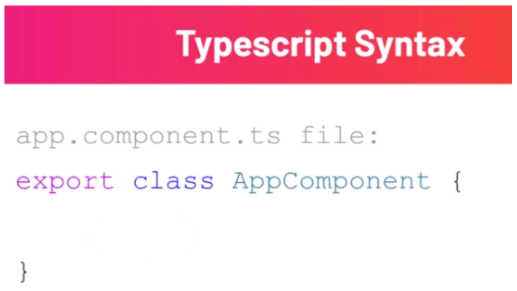
● **Pascal case (PascalCase) and Camel case (camelCase)**



Typescript Syntax

```
app.component.ts file:
class AppComponent {


}
```

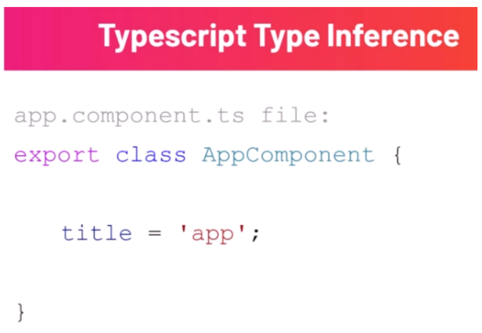
- **to make a class accessible outside of this file we export it.(Note. typescript is case sensitive so make sure to write export in lowercase letters)**



Typescript Syntax

```
app.component.ts file:
export class AppComponent {


}
```

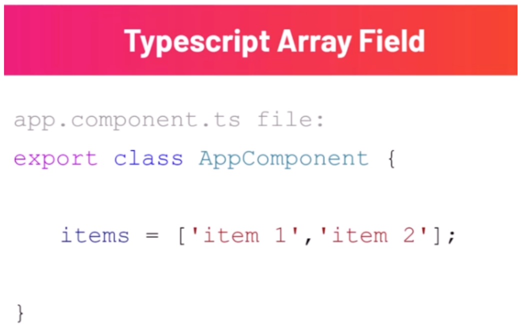
- **inside of a class you can define a filed and specify the type:**
  - **eg ->   title: string;**
  - **you can also set a default value and change as we need it.**
    - **title: string = 'app';**
  - **once you specify the default value type script refers to the default value type so there is no need to specify the type of a field if you set a default value to it.**



Typescript Type Inference

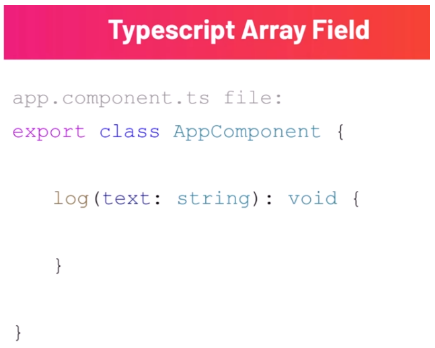
```
app.component.ts file:
export class AppComponent {

    title = 'app';

}
```

  - **we can also create craete arrays.**
    - **eg. items: string [ ] = ['item1','item 2'] ; however since the type is inferable for the value we can ignore the type.**


Typescript Array Field

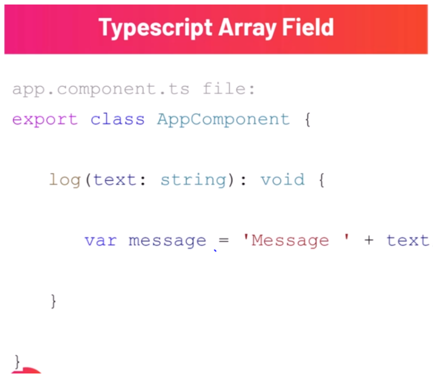
```
app.component.ts file:
export class AppComponent {

  items = ['item 1','item 2'];

}
```

- defining a method inside of a class:
  log(text:string):void{
  }


Typescript Array Field

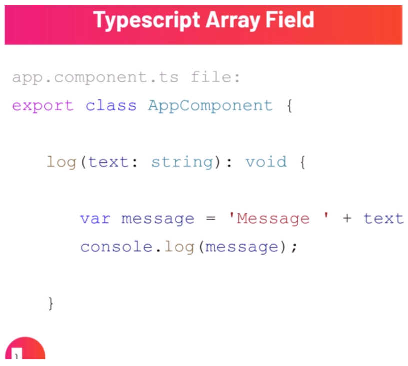
```
app.component.ts file:
export class AppComponent {

  log(text: string): void {

  }

}
```

- we can declare variable inside a method.
  - for this we use the key word "var" followed by the name of the varible.


Typescript Array Field

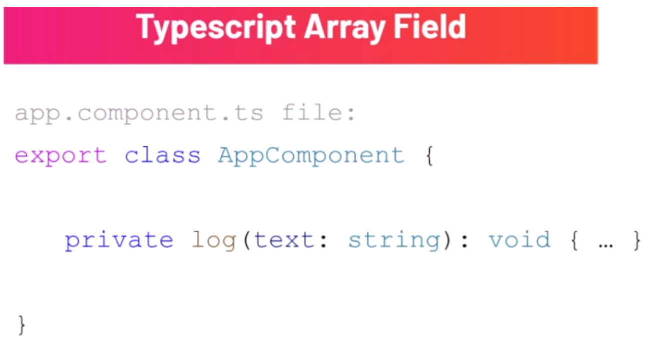
```
app.component.ts file:
export class AppComponent {

  log(text: string): void {

    var message = 'Message ' + text;

  }

}
```

- we can display the our variable using the log method of the "console" class.
  - for this we do "coonsole.log(<name of variable>)


Typescript Array Field

```
app.component.ts file:
export class AppComponent {

  log(text: string): void {

    var message = 'Message ' + text
    console.log(message);

  }
```

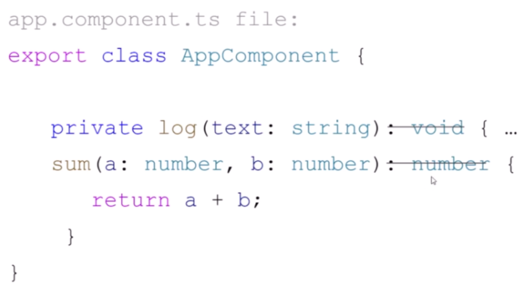
- methods are public in typescript by default.so if you want your method to be private you have to add the keyword private access modifier.



Typescript Array Field

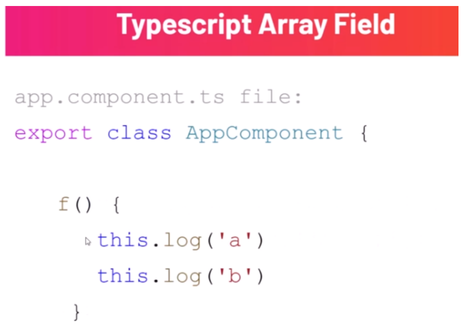
```
app.component.ts file:
export class AppComponent {

   private log(text: string): void { … }

}
```

- like fields and variables you can omit return types when they are inferable.



```
app.component.ts file:
export class AppComponent {

   private log(text: string)~~: void~~ { … }
   sum(a: number, b: number)~~: number~~ {
      return a + b;
    }
}
```

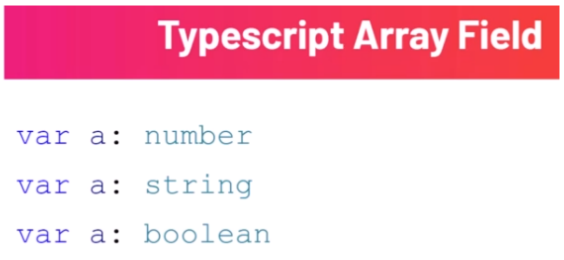
- The "this" key word is mandatory when you want access fields and methods inside the same class.



Typescript Array Field

```
app.component.ts file:
export class AppComponent {

   f() {
     this.log('a')
     this.log('b')
    }
```

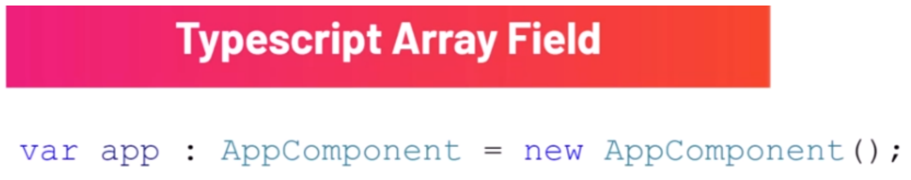
- the semicolon can also be omit semi colone if we are writing one commonet on one line.
- we have 3 primitive types in type script: number, string, boolean.



Typescript Array Field

```
var a: number
var a: string
var a: boolean
```

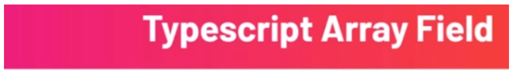
- you can also define variable of the class type.



Typescript Array Field

```
var app : AppComponent = new AppComponent();
```

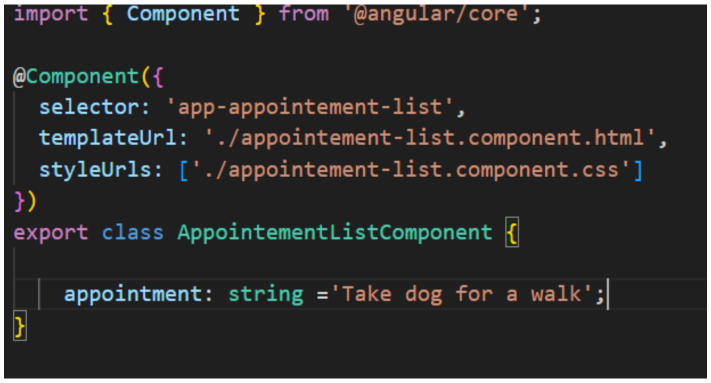
- Here is how you create arrays.


Typescript Array Field

```
var a: number[]
var a: string[]
var a: boolean[]
var app : AppComponent[]
```

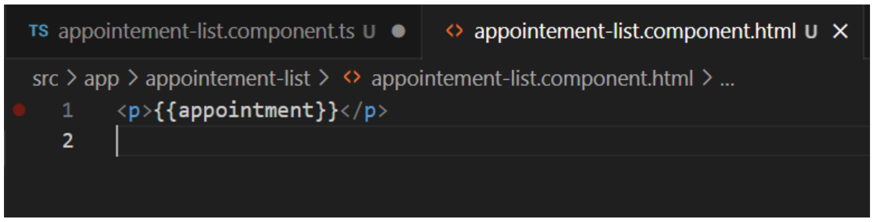
Section Four

- one way data binding.
  - create a fiedl in the appointement.componet.ts file:
    - appointement: string = 'take the dog for a walk.'



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

@Component({
  selector: 'app-appointement-list',
  templateUrl: './appointement-list.component.html',
  styleUrls: ['./appointement-list.component.css']
})
export class AppointementListComponent {

    appointment: string ='Take dog for a walk';
}
```

  - go to appointement.component.html file and use interpolation {{<name of field>}} and insert the value.


TS appointement-list.component.ts U
appointement-list.component.html U
src > app > appointement-list > appointement-list.component.html > ...

```
<p>{{appointment}}</p>
```

  - the reason why we can access the "appointment" field inside the appointement.component.html fil is beacuse the appointement.component.ts and appointement.component.html files are connected in the @Coponment decorator through the "templateUrl property".

Section Five

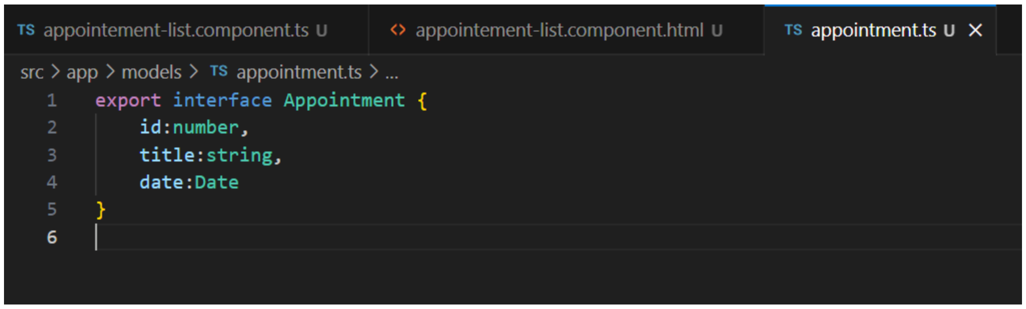
- typically we want a complex data type for our fields. Which is usually a custom datatype.
- for a custom data type we can do Two things:
    - create a class
    - create an interface.
- if you recall we can create an instance of a class in TypeScript so that we can have an object.
  - eg: appointment: <class name> = new <class name>;
- However for an interface things are a different.(it is more like a contact has to look like in the end)
- when ever you don't need to have method or anything like that it is better to create an interface.
  - How to create an interface:
    - go to the folder containing your coponemnt and use the command: "ng generate interface models/<name of comopnet>"
    - the above command creates an interface which is a model and puts it under the folder model which is under the app folder.

An interface is a syntactical contract that an entity should conform to. In other words, an interface defines the syntax that any entity must adhere to.

Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

for more info on interface in Type Script

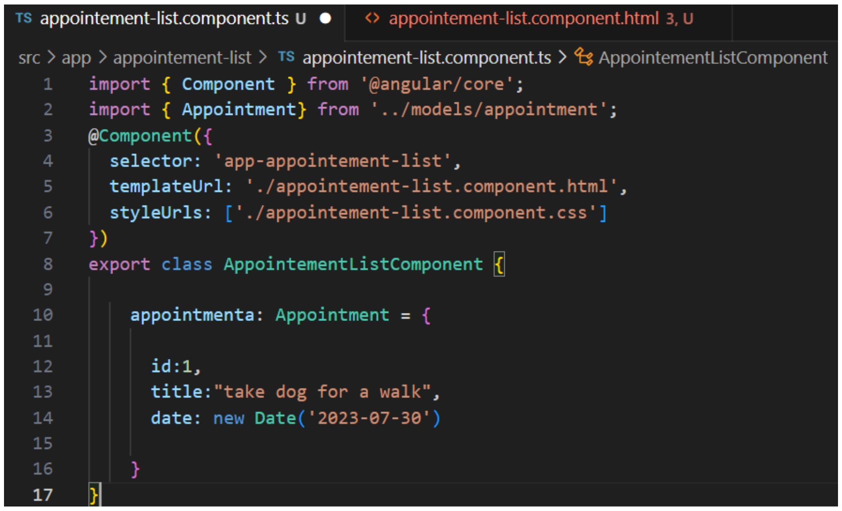
- we then added the following field as shown below.(the "Date" datatype is an interface)



TS appointement-list.component.ts U | <> appointement-list.component.html U | TS appointment.ts U ×

src > app > models > TS appointment.ts > ...

```typescript
export interface Appointment {
    id:number,
    title:string,
    date:Date
}

```



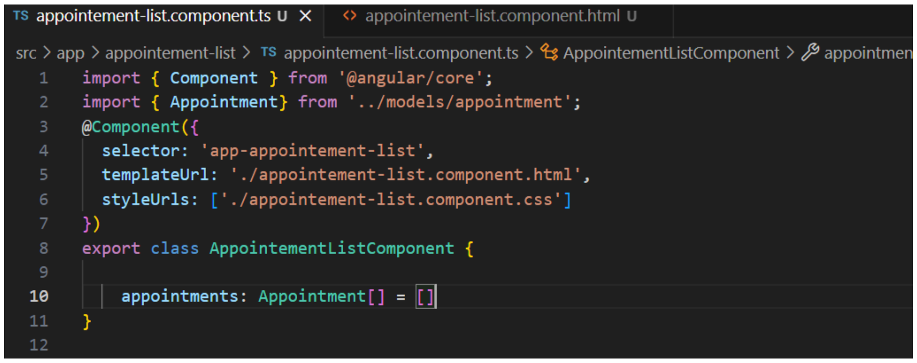
TS appointement-list.component.ts U ● | <> appointement-list.component.html 3, U

src > app > appointement-list > TS appointement-list.component.ts > AppointementListComponent

```typescript
import { Component } from '@angular/core';
import { Appointment} from '../models/appointment';
@Component({
  selector: 'app-appointement-list',
  templateUrl: './appointement-list.component.html',
  styleUrls: ['./appointement-list.component.css']
})
export class AppointementListComponent {

    appointmenta: Appointment = {

      id:1,
      title:"take dog for a walk",
      date: new Date('2023-07-30')

    }
}
```

Section Six

- we now change our appoint variable in to an array.

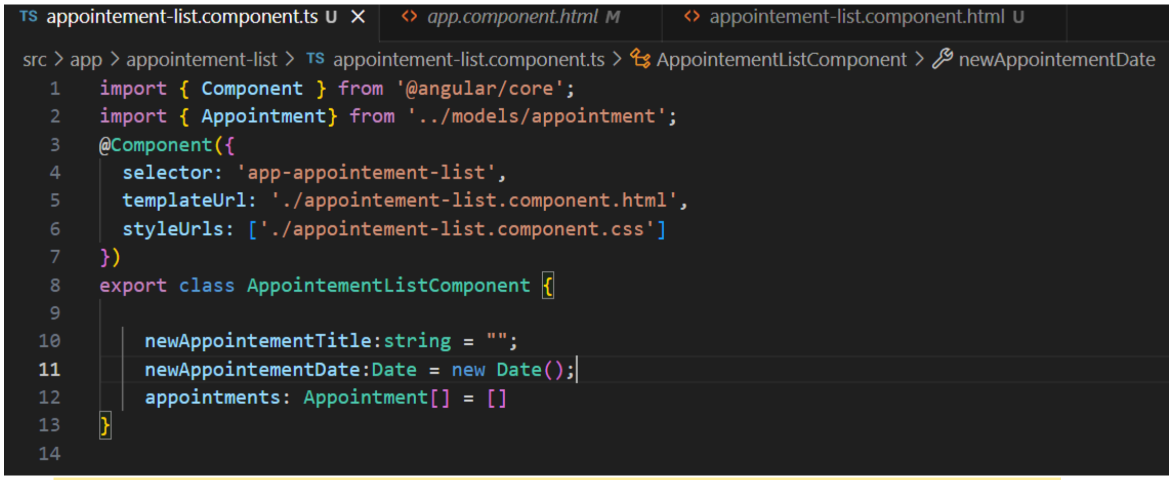


TS appointement-list.component.ts U × | <> appointement-list.component.html U

src > app > appointement-list > TS appointement-list.component.ts > AppointementListComponent > appointmen

```typescript
import { Component } from '@angular/core';
import { Appointment} from '../models/appointment';
@Component({
  selector: 'app-appointement-list',
  templateUrl: './appointement-list.component.html',
  styleUrls: ['./appointement-list.component.css']
})
export class AppointementListComponent {

    appointments: Appointment[] = []
}

```

-
- An appointment consists of three things: id, title, and date.
- for the html we will need a title and date fields. we will provide an id to our own appointments.
- it is up to you if you want to develop the Html first and then the typescript or the other way.
- Since we will be creating Two input fields: id and date , we know we have to bind those two fields to our component.
- For Two way data binding we need to create properties in our AppointementListComponet.

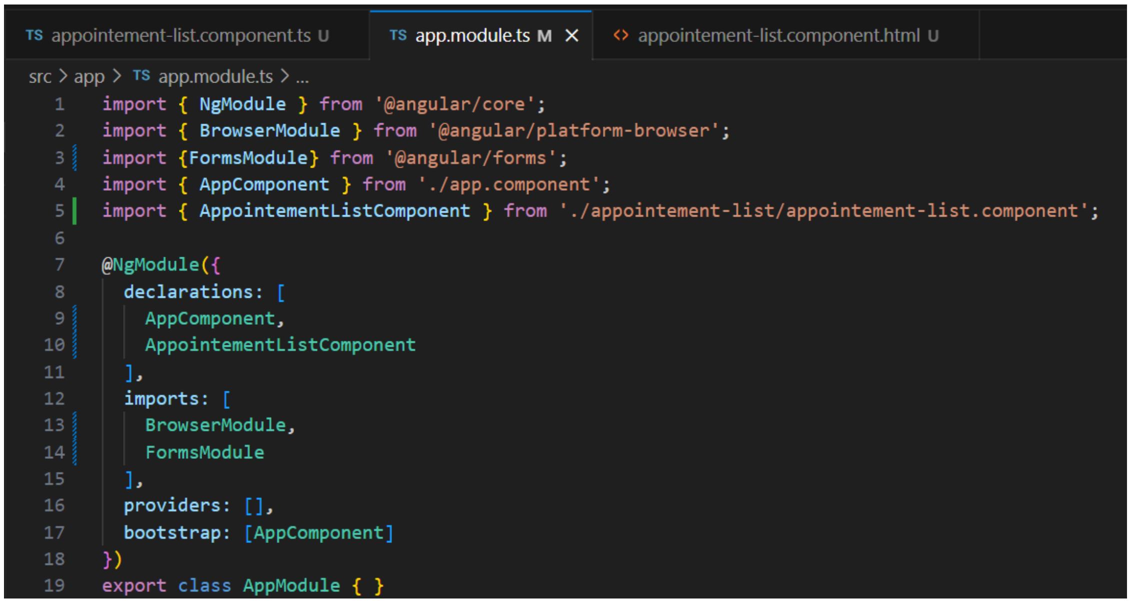


TS appointement-list.component.ts U × | app.component.html M | appointement-list.component.html U

src > app > appointement-list > TS appointement-list.component.ts > AppointementListComponent > newAppointementDate

```
import { Component } from '@angular/core';
import { Appointment} from '../models/appointment';
@Component({
  selector: 'app-appointement-list',
  templateUrl: './appointement-list.component.html',
  styleUrls: ['./appointement-list.component.css']
})
export class AppointementListComponent {

    newAppointementTitle:string = "";
    newAppointementDate:Date = new Date();
    appointments: Appointment[] = []
}
```

- For Two way data biniding we have to make use of another module - which is the "FormsModule" is imported.
- Thus,go to app.module.ts and import the "FormsModule" and also list it uder the "imports" property. Doing this allows us to use the Module in all of the componded declared under the "delcaration property"

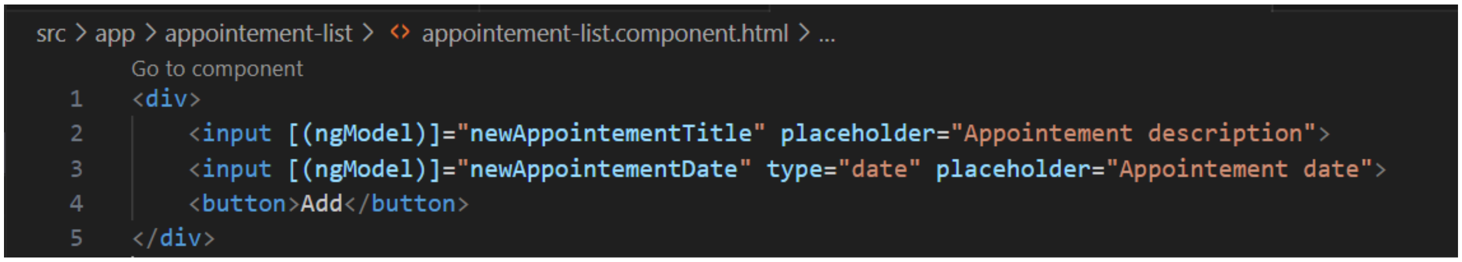


TS appointement-list.component.ts U | TS app.module.ts M × | appointement-list.component.html U

src > app > TS app.module.ts > ...

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import {FormsModule} from '@angular/forms';
import { AppComponent } from './app.component';
import { AppointementListComponent } from './appointement-list/appointement-list.component';

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

- We have now imported the FormsModule.
- Inside of the FormsModule we have what we need for our TwoWay data binding.
- under the input in your AppointementList.html filed add this [(ngModel)] = "name of the property you want to bind with".
- for the date inpute field since we dont want to write the date, we want a nice date picker, thus we specify an atribute "type = "date " "

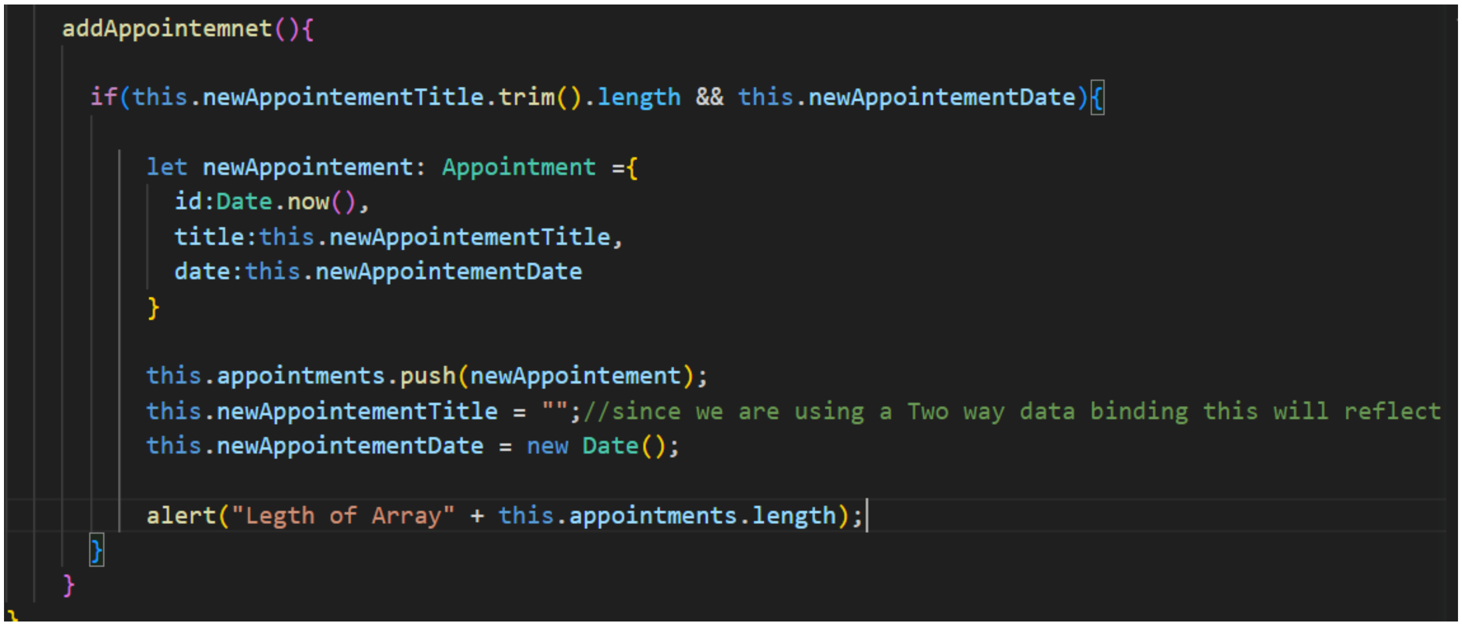


src > app > appointement-list > appointement-list.component.html > ...

Go to component

```
<div>
    <input [(ngModel)]="newAppointementTitle" placeholder="Appointement description">
    <input [(ngModel)]="newAppointementDate" type="date" placeholder="Appointement date">
    <button>Add</button>
</div>
```

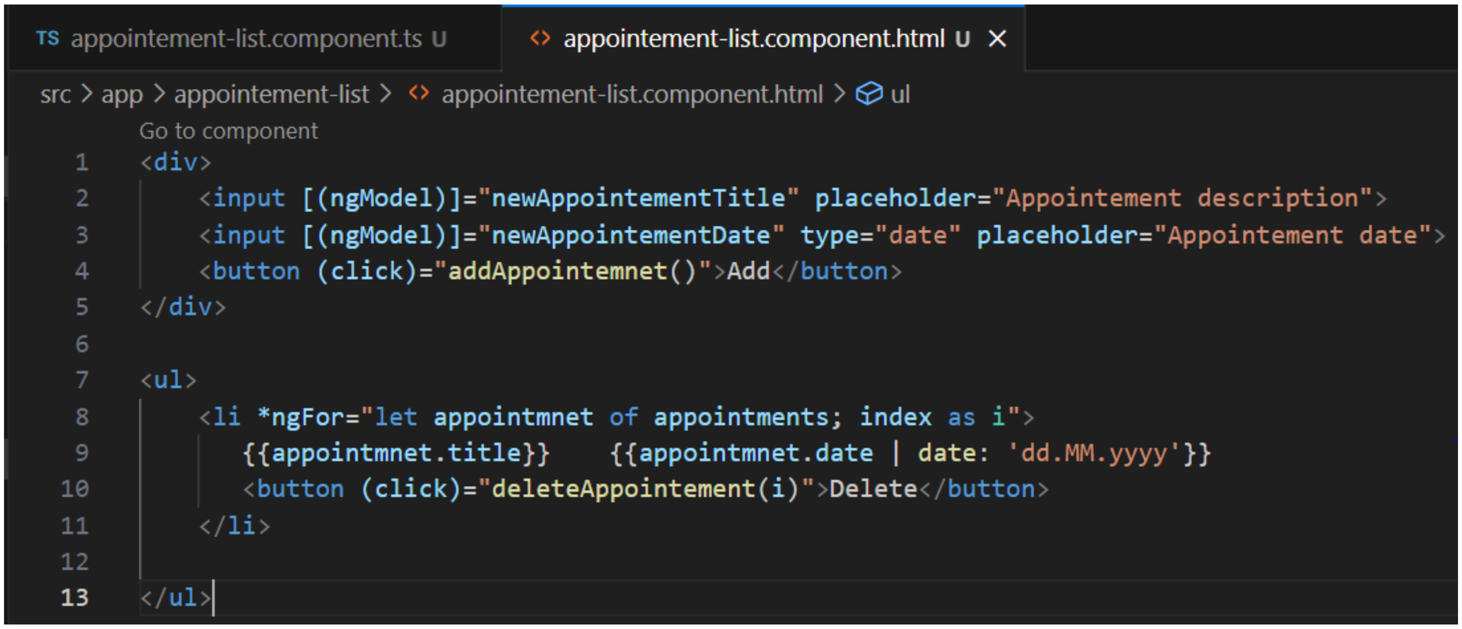
- whenever we change the value in the html field it will automatically change the value of the property to which we bind it with in the component.
- we now define a method called addAppointement() in the application-list.componet.ts class.
- we invoke this method whnever the Add button is clicked .we do that by adding the click even in the htmp button tage (click) = "addAppointement()".
- we then mofiy the methods as follows.



```
  addAppointemnet(){

    if(this.newAppointementTitle.trim().length && this.newAppointementDate){

      let newAppointement: Appointment ={
        id:Date.now(),
        title:this.newAppointementTitle,
        date:this.newAppointementDate
      }

      this.appointments.push(newAppointement);
      this.newAppointementTitle = "";//since we are using a Two way data binding this will reflect
      this.newAppointementDate = new Date();

      alert("Legth of Array" + this.appointments.length);
    }
  }
```

Section Seven

- Angular for loop.
  - we will now display the appointment list for the user.
  - <ul></ul> ul is the unordered list .
  - inside <ul><ul> we place <ls></ls>
  - to create a loop in angular we use a directive called ngFor.



TS appointement-list.component.ts U
appointement-list.component.html U
src > app > appointement-list > appointement-list.component.html > ul
Go to component

```
<div>
    <input [(ngModel)]="newAppointementTitle" placeholder="Appointement description">
    <input [(ngModel)]="newAppointementDate" type="date" placeholder="Appointement date">
    <button (click)="addAppointemnet()">Add</button>
</div>

<ul>
    <li *ngFor="let appointmnet of appointments; index as i">
      {{appointmnet.title}}    {{appointmnet.date | date: 'dd.MM.yyyy'}}
      <button (click)="deleteAppointement(i)">Delete</button>
    </li>

</ul>
```

- to delete an Appointenemtn we created a method called deleteAppointement(index: number) that's takes an index and provoked from the delete button in the html.
- to get the indesx we added "index as i "in our "**ngFor ="let appointment of Appointemes".

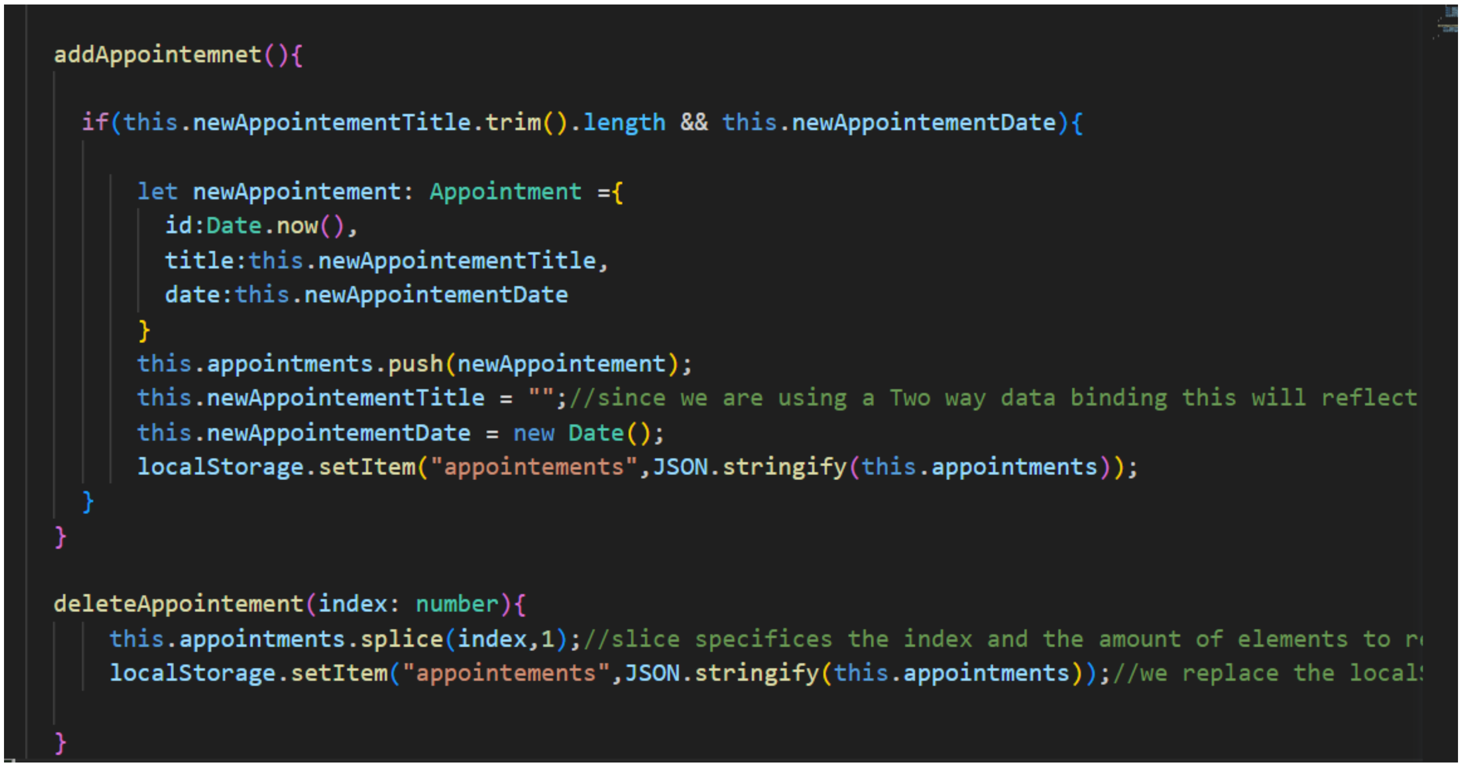
Section Eight

Storing an Appointment in the local storage:

- it is important to stor our data later on we will connect an SQL or noSql database.
- localstoge is a sloot in your browser.
- we first take the data from the typeScrip component class.For example we will take an apointment and converted it from the typeScript data type to "JSON"
- we do that using this command ":

```
localStorage.setItem("appointements",JSON.stringify(this.appointments));
```

- we add the above command to both our addAppointemnt and deleteAppointement methods.

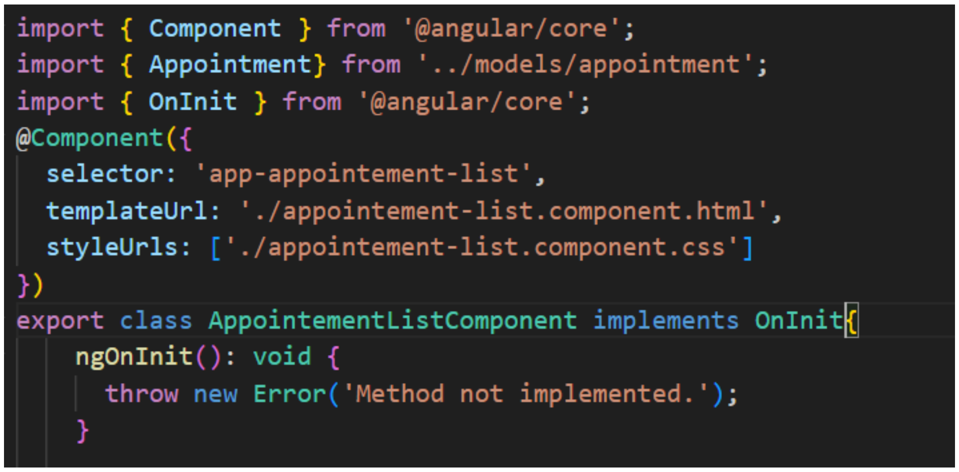


```
addAppointemnet(){

  if(this.newAppointementTitle.trim().length && this.newAppointementDate){

    let newAppointement: Appointment ={
      id:Date.now(),
      title:this.newAppointementTitle,
      date:this.newAppointementDate
    }
    this.appointments.push(newAppointement);
    this.newAppointementTitle = "";//since we are using a Two way data binding this will reflect
    this.newAppointementDate = new Date();
    localStorage.setItem("appointements",JSON.stringify(this.appointments));
  }
}

deleteAppointement(index: number){
   this.appointments.splice(index,1);//slice specifices the index and the amount of elements to r
   localStorage.setItem("appointements",JSON.stringify(this.appointments));//we replace the local

}
```

- notice that the the setItem method takes a key b and value.
- for the key we provided a string for the value we converted our Appointments array in to a JSO format string using "JSON.stringfy(this.appointemnets)" command.

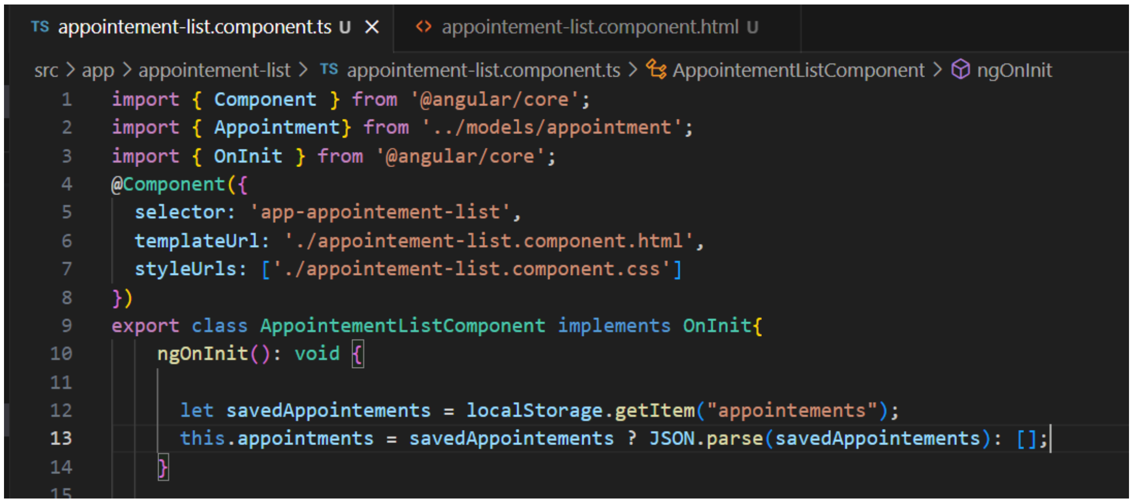
Section Nine

- using ngOnInit to load appointements from local storage.
  - we currently are saving our appointment array in to the localStorage whenever we add and delete an appointment.
  - However we need to load our appointments array that we saved to the local storage.
  - the best time to load the appoinmetns array is when we star the "AppointementListComponent " class.when this componet gets initialized.
  - for this we have someting called "lifeCycleHooks".
  - Angular calls lifecycle hook methods on directives and components as it creates, changes, and destroys them.for our project we will use the "ngOninit" lifecycle hook.
  - we import "OInit" from @angulat /Code and we specify on the class name that the class implents OnInit.However, since we declared that class will implement OnInit we have to implent the "ngOninit" method as shown below.



```
import { Component } from '@angular/core';
import { Appointment} from '../models/appointment';
import { OnInit } from '@angular/core';
@Component({
  selector: 'app-appointement-list',
  templateUrl: './appointement-list.component.html',
  styleUrls: ['./appointement-list.component.css']
})
export class AppointementListComponent implements OnInit{
    ngOnInit(): void {
      throw new Error('Method not implemented.');
    }
```

  - we then retrieve the stored Appointement array from the local Storage and assigne it to a variable.
  - if the variable is not null or emptry we deserialize what from the JSON format to typescript format using JSON.parse(this.<name of variable>) and assign the value to the Appointement array varibale.
  - if the variable is null we initialize the Appointments array with an empty array we do this with the ?:(ternary operator).


TS appointement-list.component.ts U ×
appointement-list.component.html U
src > app > appointement-list > TS appointement-list.component.ts > AppointementListComponent > ngOnInit

```
import { Component } from '@angular/core';
import { Appointment} from '../models/appointment';
import { OnInit } from '@angular/core';
@Component({
  selector: 'app-appointement-list',
  templateUrl: './appointement-list.component.html',
  styleUrls: ['./appointement-list.component.css']
})
export class AppointementListComponent implements OnInit{
    ngOnInit(): void {

      let savedAppointements = localStorage.getItem("appointements");
      this.appointments = savedAppointements ? JSON.parse(savedAppointements): [];
    }
```

Section Ten

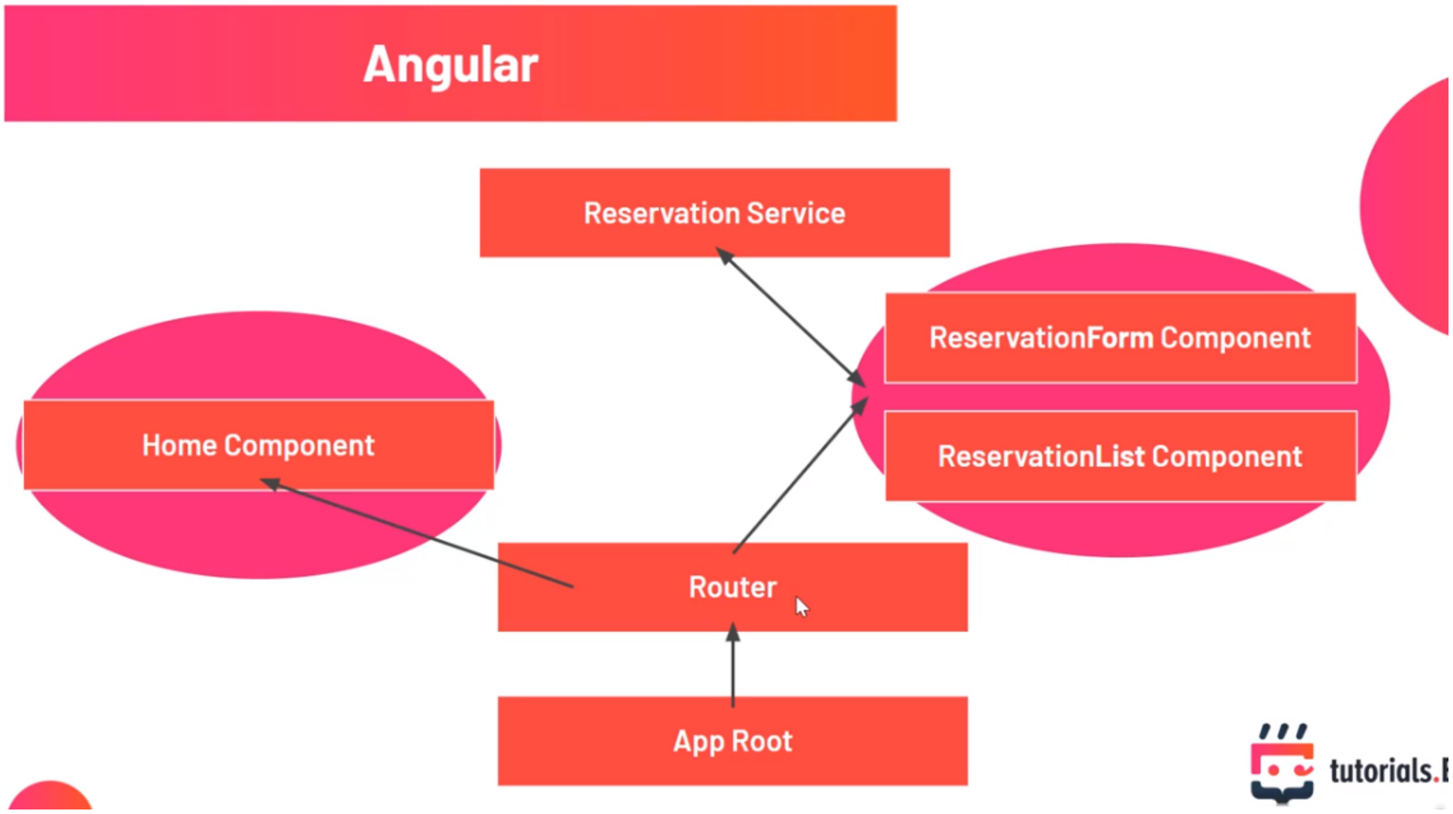
- **Adding Bootstrap.**
  - **is a styling library.**
  - **we install boot start using the command npm install bootstarp@5.3.**
  - **we then go to our "styles.css" file under src folder where we can specify the global style for our application.**
  - **in bootstrap we put most of the stuff inside a single container.**
  - **For instance if we have a new page everything in the page is inside a container.**
  - **in bootstarp we use a gird system which consistas of rows and columns.**
  -

--PART THREE--

Section Two

Project Structure

- **This application is about managing hotel reservations/bookings.**
- **Typically in web applications we have 4 types of operations(crud- create,read,update,delete):**
  - **create reservation**
  - **edit reservation**
  - **list reservation**
  - **delete reservation**
- **structure of our project**



- lets discus the structure of our project:

  1.we first start with our App-route(which is the selector tag for App-componet componet that is automatically created. thus our application always starts from app-root)

  2. in the app-root (app-component.hml we have <router-outlet></router-outlet>).with this we defined a root in our application. based on the current rooter path or root we can see different components.

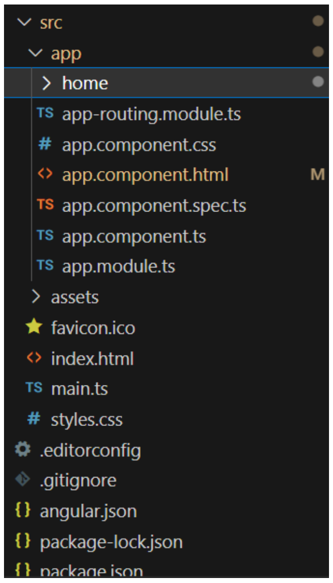
  3.From the root, if we get directed we can go to the Home Component. The circle that we see around the component is a module, which is the home moduel. The Home component is inside the Home module.

  4.From the root we can also got to the Reservationist Component and the ReservarionForm Componets(which are under the Reservation Module).

  5. In this project we will add a service to load and handle the data. The component is only for handling the user interface stuff and forwarding request to save and load data. Thus, the Reservation service will handle all the data. Since we have Two components which handle reservations it is best to add one service that we can all call for handling reservation data.

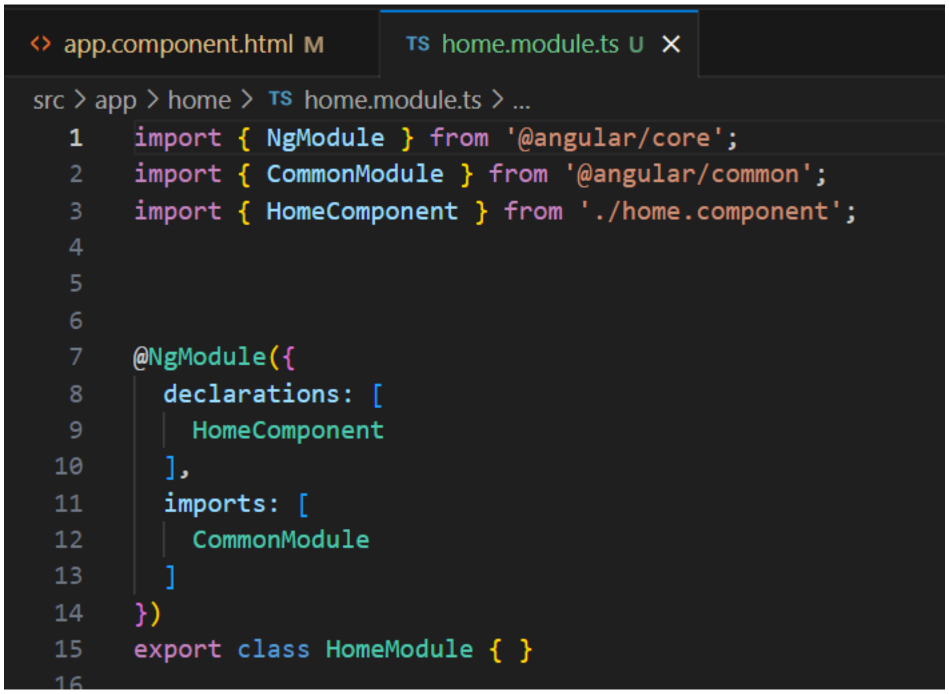
Section Three

- lets begin to set up the structure of our project:
  - we first create the "home" module using the command: ng g module <name of module>.
  - if we see under the "src/app" folder we see the home module.



```
src
  app
    home
    app-routing.module.ts
    app.component.css
    app.component.html        M
    app.component.spec.ts
    app.component.ts
    app.module.ts
  assets
  favicon.ico
  index.html
  main.ts
  styles.css
.editorconfig
.gitignore
angular.json
package-lock.json
package.json
```

- **we then create our home component and palce undet his specific module using the command: ng g component --module=home**
- **now if we go the "home.module.ts" file we see the component we just created listed under the declaration.**



app.component.html M | home.module.ts U

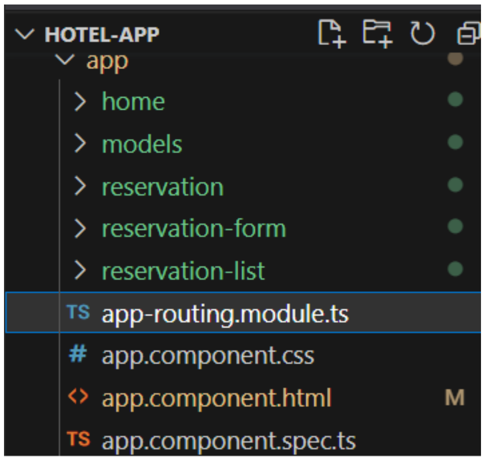
src > app > home > home.module.ts > ...

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HomeComponent } from './home.component';



@NgModule({
  declarations: [
    HomeComponent
  ],
  imports: [
    CommonModule
  ]
})
export class HomeModule { }
```

- **Note: a component can only be declared by a single module.Right now our home component is declared inside our home module and it can not be declared in another module.**
  - **we then create the reservation module and the two classes that are under this module ReservationModule and ReservationList.**
  - **we then create our service(reservation service) and put it under the reservation folder. we use the command: ng g service reservation/reservation**

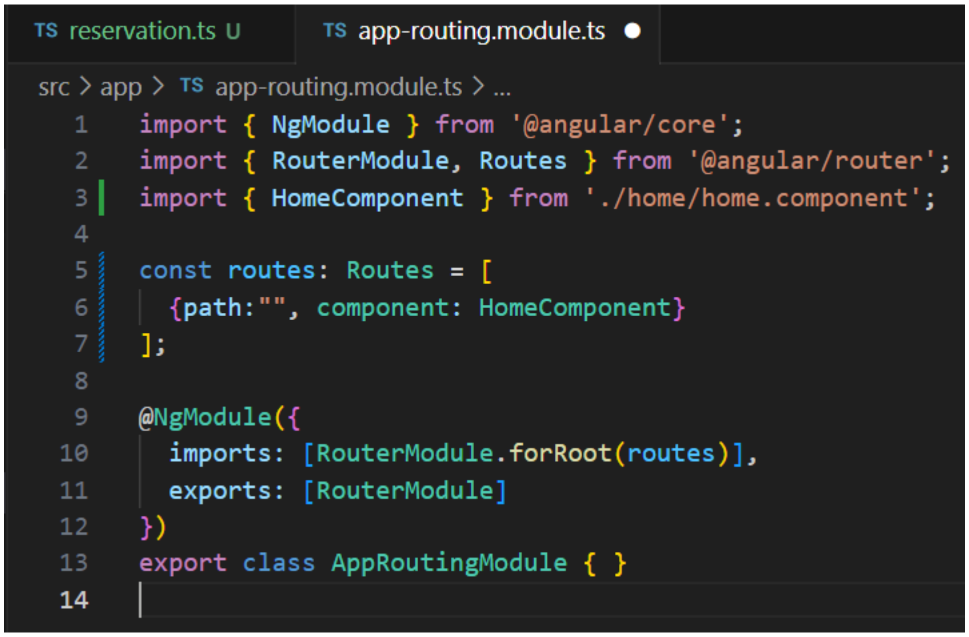
Section Four

- **if you got to home componet.html you will see the router-outlet.**
- **by using the router we can display diffrent compents by assigning roots to it.**
- **on the left side under the the "hotle app" folder we see a new file called the app-routing.module.ts.**



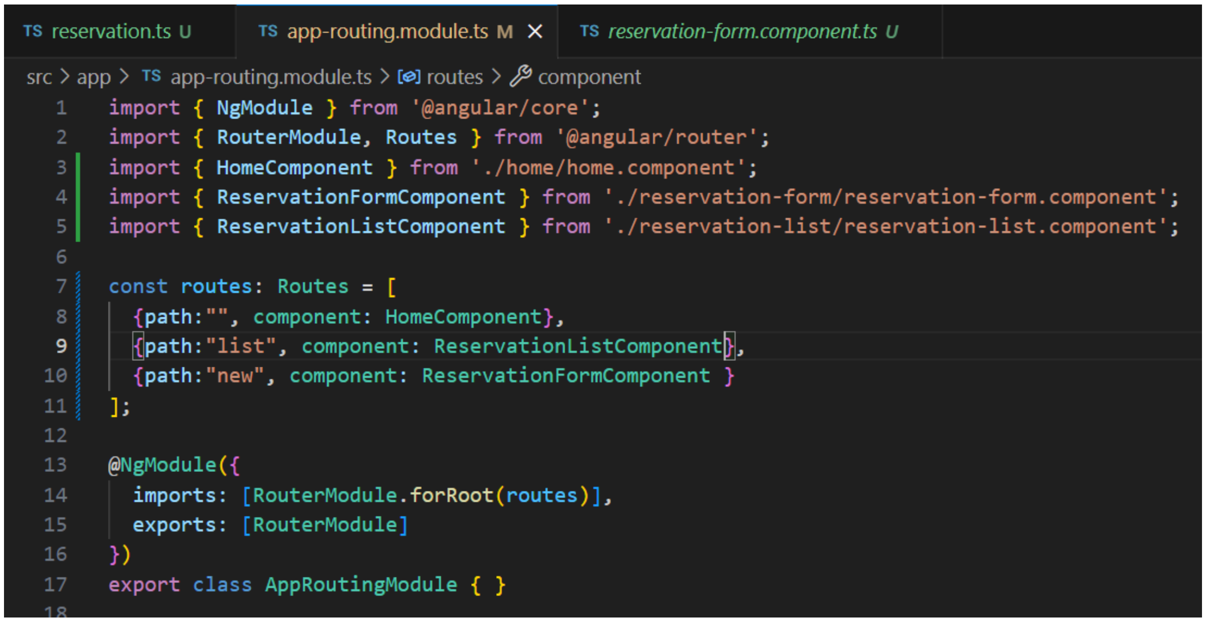
```
HOTEL-APP
  app
    home
    models
    reservation
    reservation-form
    reservation-list
    app-routing.module.ts
    app.component.css
    app.component.html   M
    app.component.spec.ts
```

- **inside of the app-routing.model.ts file we can define constant roots .These are the paths that a user can explore in our application.we do this by creating path objects in side the "cons routes" array.**
- **lets cretae a path object and attach a componet to it. we do this by this comamnd {path:"<ourpath>", component:<name of componet we wat to display>}.**



```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';

const routes: Routes = [
  {path:"", component: HomeComponent}
];

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

- since we didn't add anything in the path, when we visit our standar root(localhost:4200) we will display the home componet in our roorter outlet.
- we can add the path in simialt manner to add end points for ResevationList and ReservationForm.

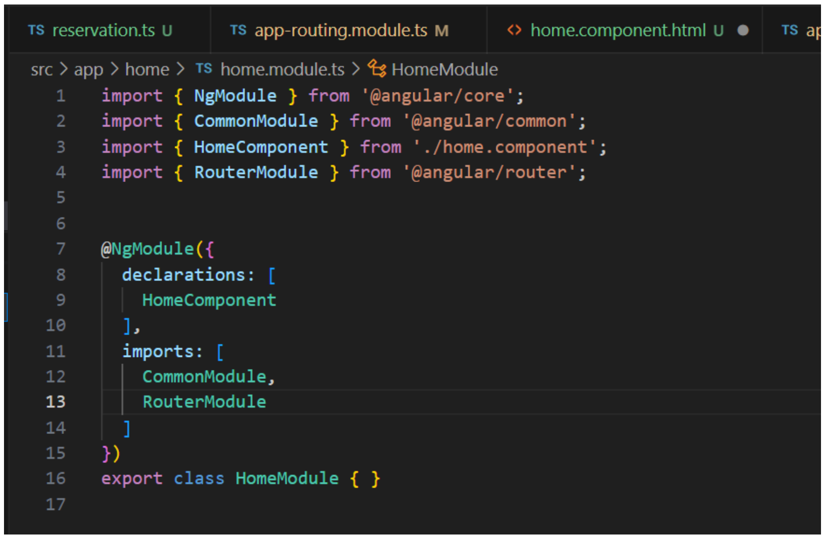

TS reservation.ts U
TS app-routing.module.ts M
TS reservation-form.component.ts U
src > app > TS app-routing.module.ts > routes > component

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ReservationFormComponent } from './reservation-form/reservation-form.component';
import { ReservationListComponent } from './reservation-list/reservation-list.component';

const routes: Routes = [
  {path:"", component: HomeComponent},
  {path:"list", component: ReservationListComponent},
  {path:"new", component: ReservationFormComponent }
];

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

Section Five

Building a Simple Navigation

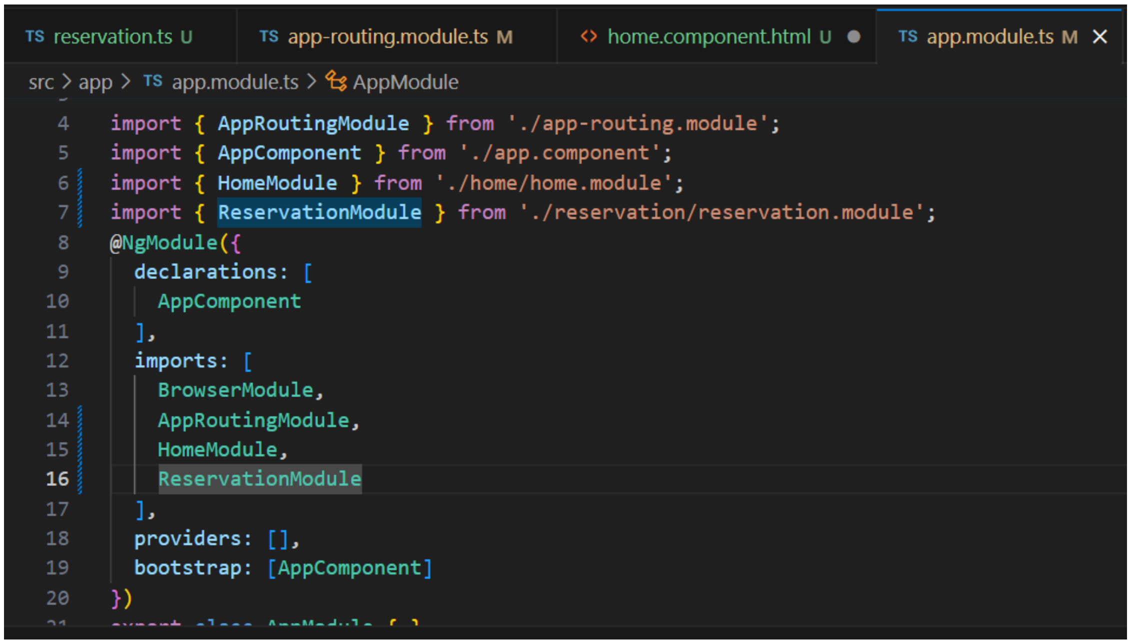
- lets build a simple navigation menu in our component by using the our rooter.
- we can make use of the rooter link attribute to link to other components.
- when we add [routerLink] attribute the button we created we get an error.This is beacuse we have not imported our RouterModuel to the HomeModule which contains our home component.
- to do this we go to home.module.ts and import the RouterModule and and add it to the imports array.when we do this the erroe we got goes away.


TS reservation.ts U | TS app-routing.module.ts M | home.component.html U | TS ap

src > app > home > TS home.module.ts > HomeModule

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HomeComponent } from './home.component';
import { RouterModule } from '@angular/router';


@NgModule({
  declarations: [
    HomeComponent
  ],
  imports: [
    CommonModule,
    RouterModule
  ]
})
export class HomeModule { }
```

- now when we add a lint to the routerLink attribute we get an errror.this is beacuse we have not added the two new modules that we create, the Home and Reservation Modules, to the AppModule.
- Thus we go to app.module.ts import those models and also add them to the imports array .


TS reservation.ts U | TS app-routing.module.ts M | home.component.html U | TS app.module.ts M ✕

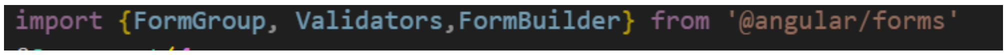
src > app > TS app.module.ts > AppModule

```
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeModule } from './home/home.module';
import { ReservationModule } from './reservation/reservation.module';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HomeModule,
    ReservationModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

Section Six

Creating a form group

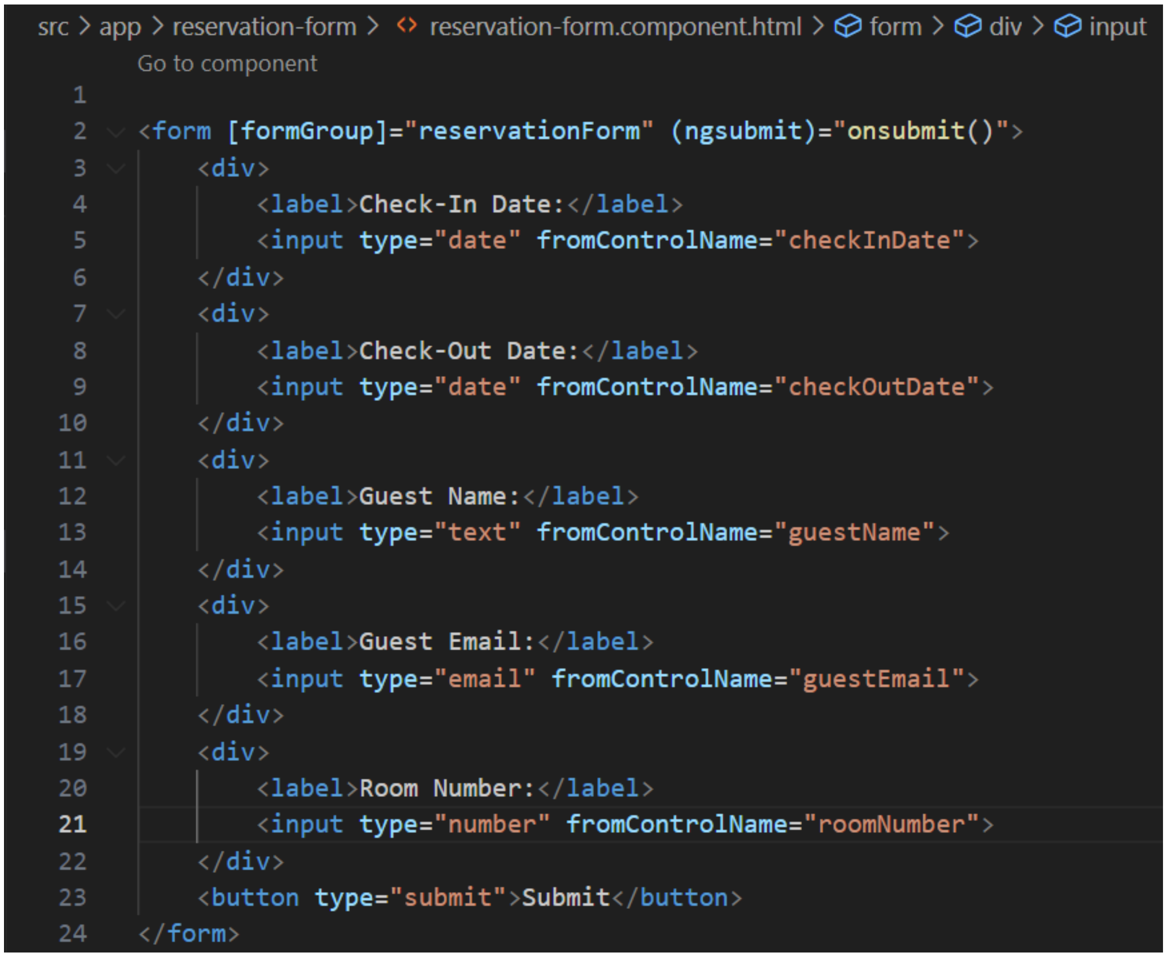
- There are Two ways of form validation in angular:
  - Reactive Form Validation: we validate our form in the attached component typescript class.
  - Template-Driven Form Validation: in side of the template (in our html)
- One we have this got to reservation-forms.compoenet.ts and import FormsGroup,Forms,Builder,Validators



```
import {FormGroup, Validators,FormBuilder} from '@angular/forms'
```

- now we need a from group that we can bind to our form. we created a formGroup field called reservation in the home.componet.ts : reservation: FormGroup = new FormGroup({});
- we then attache this group to our form:

<form [formGroup] = "<bname of fromgroup filed>" (ngsubmit)="<name of method>">

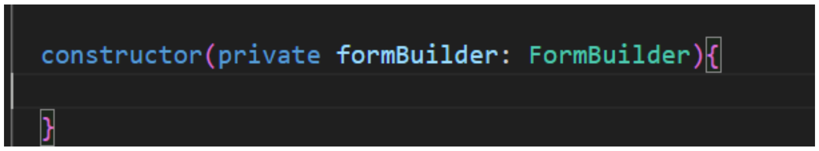

src > app > reservation-form > reservation-form.component.html > form > div > input
Go to component

```html

<form [formGroup]="reservationForm" (ngsubmit)="onsubmit()">
    <div>
        <label>Check-In Date:</label>
        <input type="date" fromControlName="checkInDate">
    </div>
    <div>
        <label>Check-Out Date:</label>
        <input type="date" fromControlName="checkOutDate">
    </div>
    <div>
        <label>Guest Name:</label>
        <input type="text" fromControlName="guestName">
    </div>
    <div>
        <label>Guest Email:</label>
        <input type="email" fromControlName="guestEmail">
    </div>
    <div>
        <label>Room Number:</label>
        <input type="number" fromControlName="roomNumber">
    </div>
    <button type="submit">Submit</button>
</form>
```

Section Seven

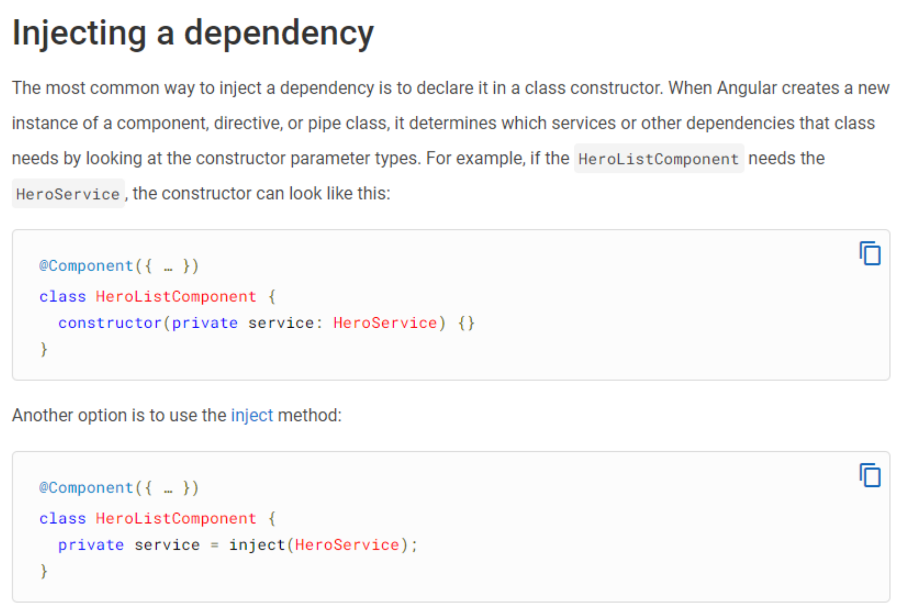
Validating a form using reactive form validation.

- **we now have created a form group called reservation and inside of this from group we have control where each control is an input field.**
- **we now have to pick all of our from controlls from our inpute form and put them inside our form group for this we will use FormBuilder.**
- **lets use the dependancy injection of angular,**
- **lets create a constructor() -> a constructor is a method that gets invoked the moment you create an instance of the class.**



```typescript
constructor(private formBuilder: FormBuilder){

}
```

- **here Angular will authomathicaly inject he FormBuilder in to our ReservationForm class.**



Injecting a dependency

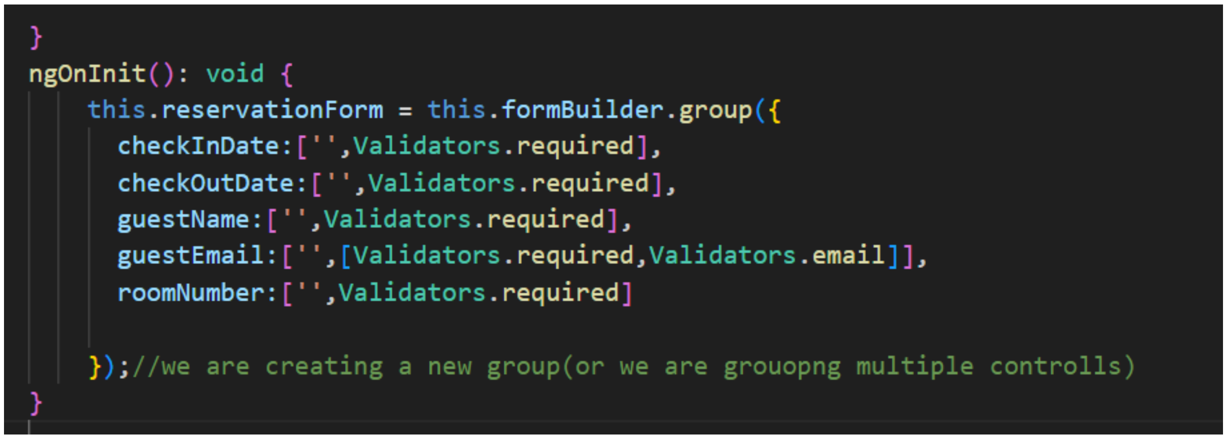
The most common way to inject a dependency is to declare it in a class constructor. When Angular creates a new instance of a component, directive, or pipe class, it determines which services or other dependencies that class needs by looking at the constructor parameter types. For example, if the `HeroListComponent` needs the `HeroService`, the constructor can look like this:

```
@Component({ … })
class HeroListComponent {
  constructor(private service: HeroService) {}
}
```

Another option is to use the inject method:

```
@Component({ … })
class HeroListComponent {
  private service = inject(HeroService);
}
```

- **we then create the form controll group using the [FromBuilder.group](#) method.**

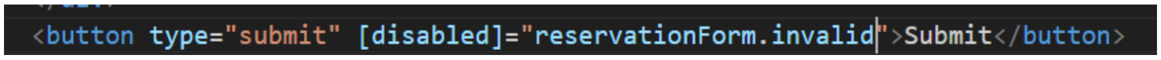


```
}
ngOnInit(): void {
    this.reservationForm = this.formBuilder.group({
      checkInDate:['',Validators.required],
      checkOutDate:['',Validators.required],
      guestName:['',Validators.required],
      guestEmail:['',[Validators.required,Validators.email]],
      roomNumber:['',Validators.required]

    });//we are creating a new group(or we are grouopng multiple controlls)
}
```

- **here we are adding all of our controls , provide their initial values and then validate them using the "Validator".**
- **if we want to implement multiple Validation on a from Control we create an array , such us in the case of the email form control.**

Section Eight

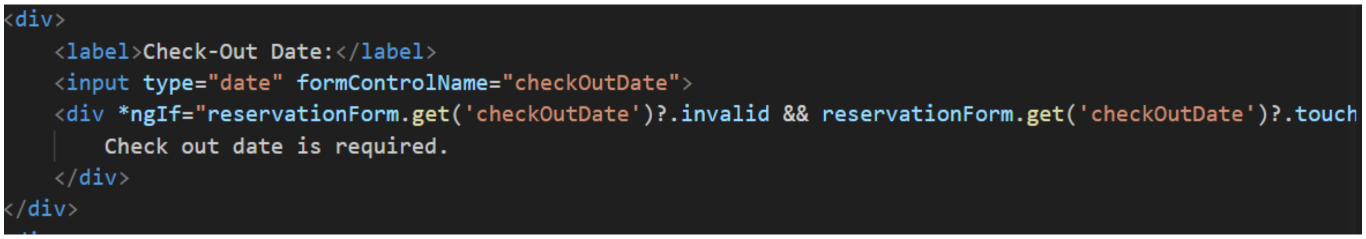
- **Dynamically toggle html attributes based on form state**
  - **we can disable our submit button if the reservationFrom is not valid we ca do this by using the adding the command to our submit button [disable] = "reservationFrom.invalid" .**



```
<button type="submit" [disabled]="reservationForm.invalid">Submit</button>
```

Section Nine:

Showing Validation messages

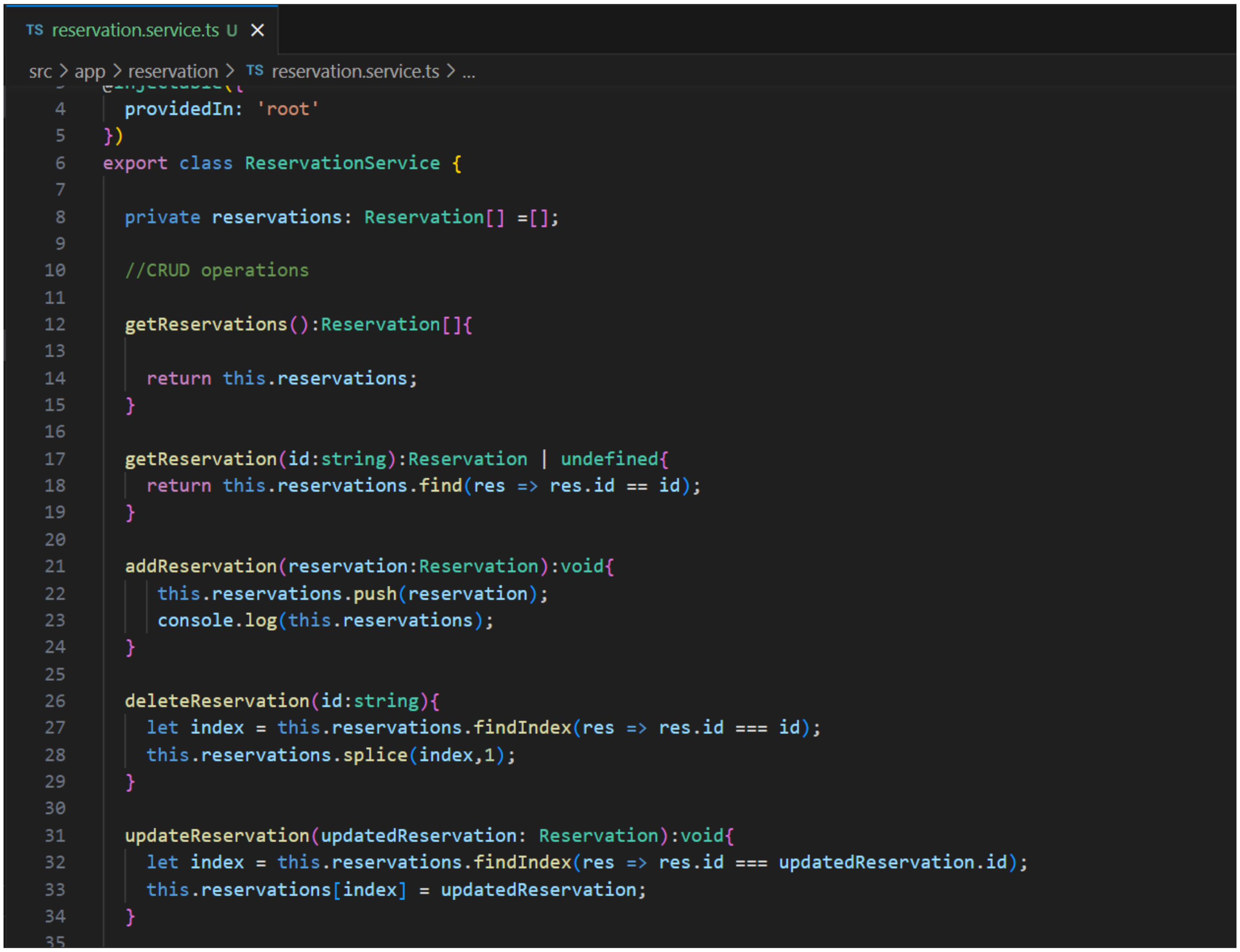


```
<div>
    <label>Check-Out Date:</label>
    <input type="date" formControlName="checkOutDate">
    <div *ngIf="reservationForm.get('checkOutDate')?.invalid && reservationForm.get('checkOutDate')?.touch
        Check out date is required.
    </div>
</div>
```

Section Ten

Creating the Reservation Service:

- **to load and store, update and delete our reservations we create a reservation service.**
- **the service is a place where we handle the data.**
- **for a service the decoretor is @Injectable => it makes the class to be able to be injected in a construcor.**
- **we then created the list that contins all of our reservations.**

- 

TS reservation.service.ts U ×

src > app > reservation > TS reservation.service.ts > ...

```
  providedIn: 'root'
})
export class ReservationService {

  private reservations: Reservation[] =[];

  //CRUD operations

  getReservations():Reservation[]{

    return this.reservations;
  }

  getReservation(id:string):Reservation | undefined{
    return this.reservations.find(res => res.id == id);
  }

  addReservation(reservation:Reservation):void{
     this.reservations.push(reservation);
     console.log(this.reservations);
  }

  deleteReservation(id:string){
    let index = this.reservations.findIndex(res => res.id === id);
    this.reservations.splice(index,1);
  }

  updateReservation(updatedReservation: Reservation):void{
    let index = this.reservations.findIndex(res => res.id === updatedReservation.id);
    this.reservations[index] = updatedReservation;
  }

```

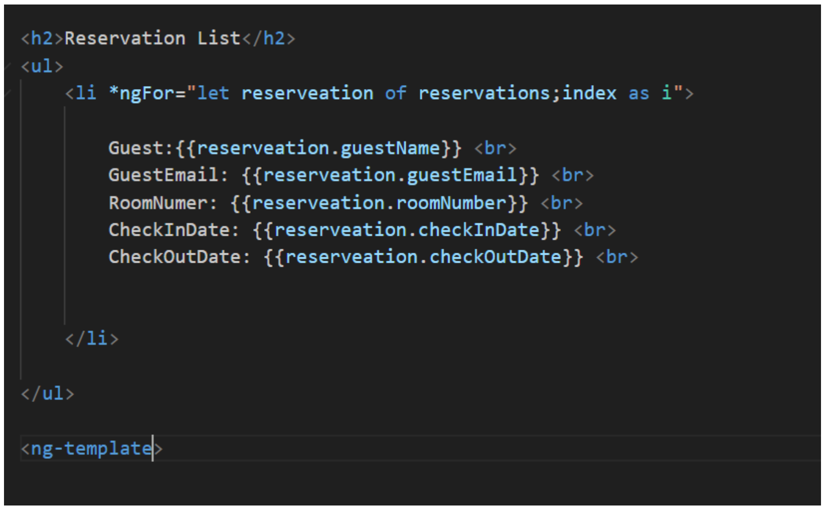
'==' equality Vs '===' strict equality operator: Please see here for more information

Section Eleven

using the localstorage in the reservation service.

Section Twelve

Showing all the reservation list

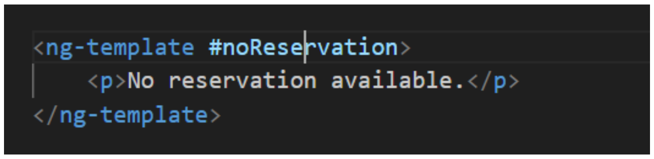


```
<h2>Reservation List</h2>
<ul>
    <li *ngFor="let reserveation of reservations;index as i">

        Guest:{{reserveation.guestName}} <br>
        GuestEmail: {{reserveation.guestEmail}} <br>
        RoomNumer: {{reserveation.roomNumber}} <br>
        CheckInDate: {{reserveation.checkInDate}} <br>
        CheckOutDate: {{reserveation.checkOutDate}} <br>


    </li>

</ul>

<ng-template>
```

Section 13

ng template and localreference.

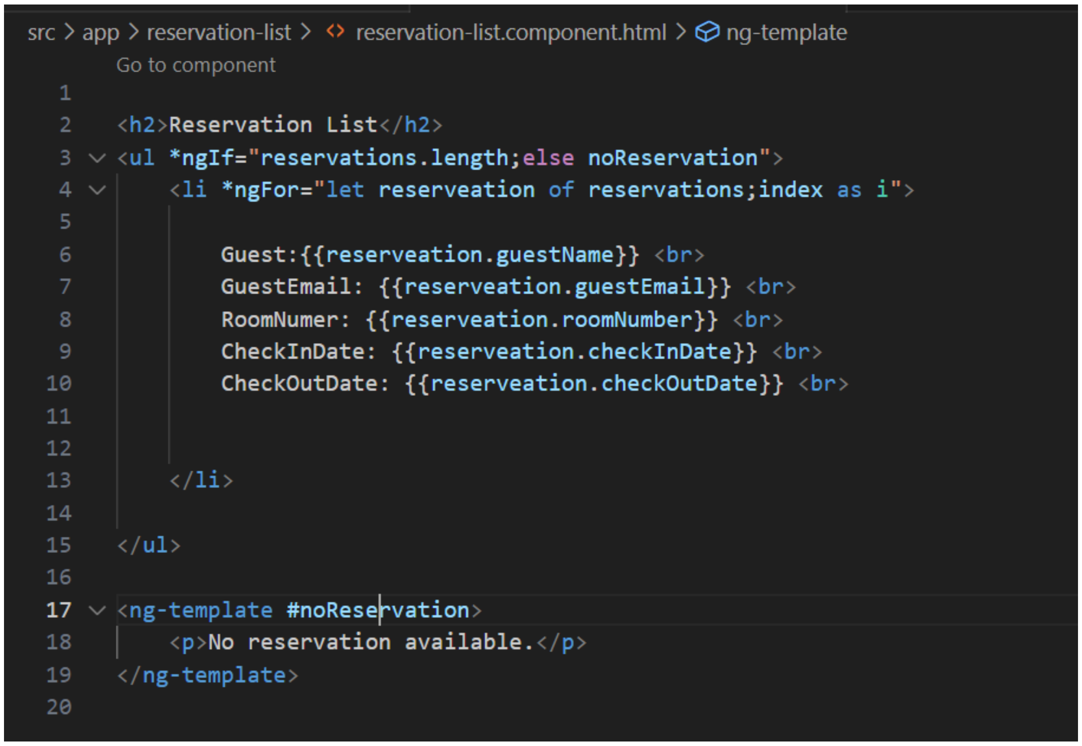
- **we want to show the list only if there are reservations.**
- **for this purpose we use *ng-template. and provide a name using the #.**



```
<ng-template #noReservation>
    <p>No reservation available.</p>
</ng-template>
```

- **now we go to our list and using *ngIf we set the condition. for this we do the following.**

<li *ngIf="reservations.length; else <name of ng template> ">



src > app > reservation-list > reservation-list.component.html > ng-template

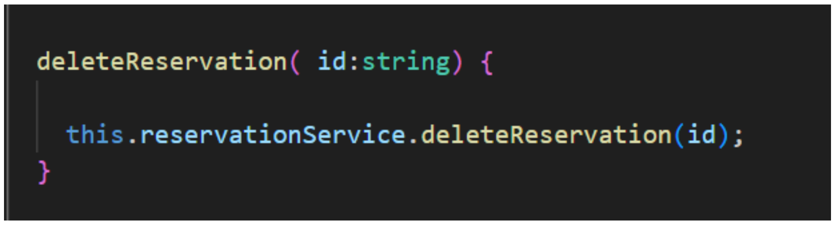
Go to component

```

<h2>Reservation List</h2>
<ul *ngIf="reservations.length;else noReservation">
    <li *ngFor="let reserveation of reservations;index as i">

        Guest:{{reserveation.guestName}} <br>
        GuestEmail: {{reserveation.guestEmail}} <br>
        RoomNumer: {{reserveation.roomNumber}} <br>
        CheckInDate: {{reserveation.checkInDate}} <br>
        CheckOutDate: {{reserveation.checkOutDate}} <br>


    </li>

</ul>

<ng-template #noReservation>
    <p>No reservation available.</p>
</ng-template>

```

Section 14

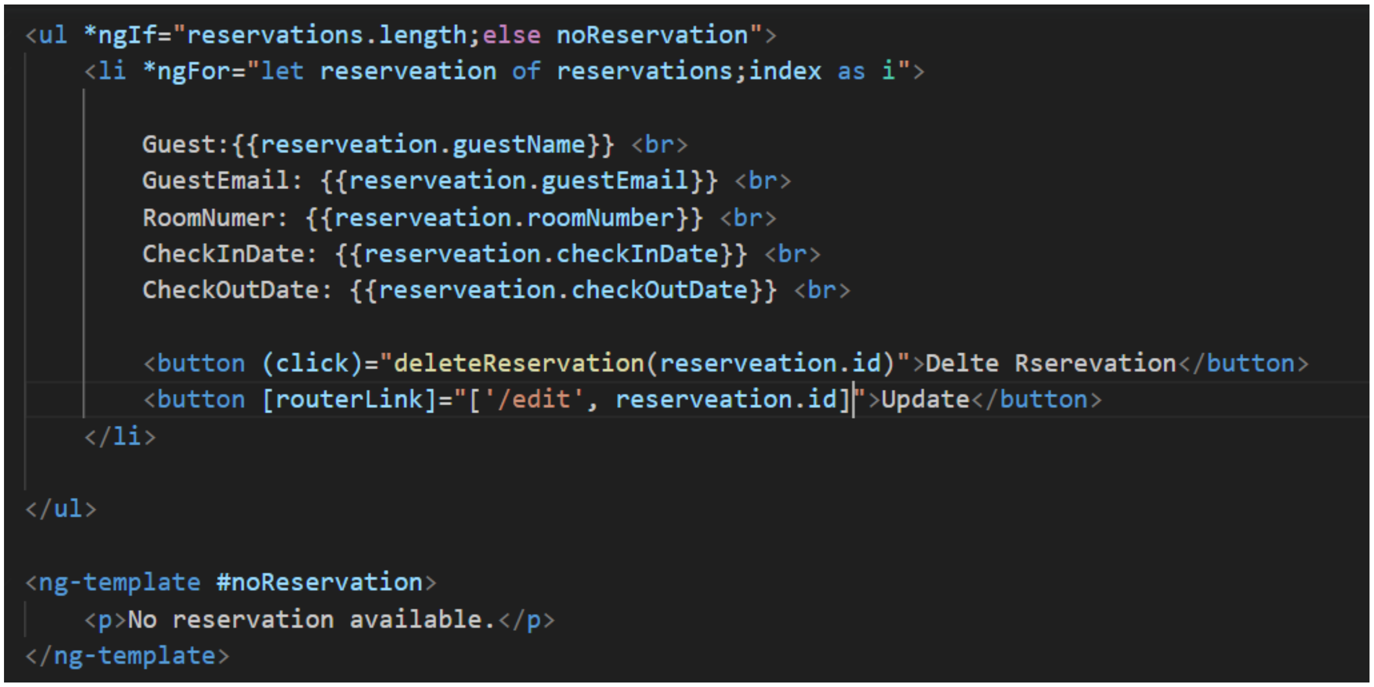
Deleting Reservations

- **we added a delete method on the reservaions list componet.**
- **in this new delete method we we called the deleteReservstion method from the reserevationsService class and passed in an id.**



```
deleteReservation( id:string) {

  this.reservationService.deleteReservation(id);
}
```

- **we then went to the reservationlistcompoet.html and added a delete button where we called the "delete" method we created.**

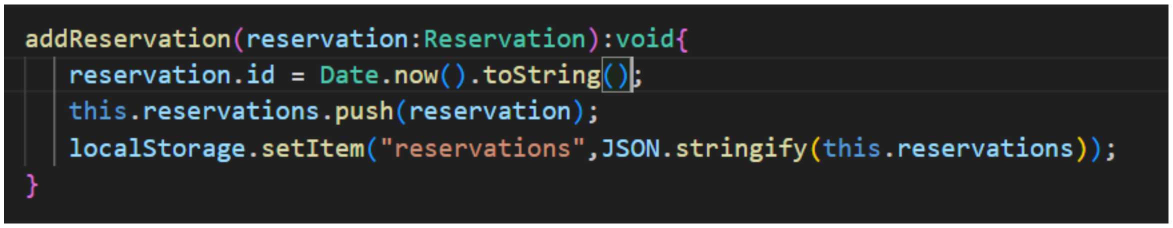


```
<ul *ngIf="reservations.length;else noReservation">
    <li *ngFor="let reserveation of reservations;index as i">

        Guest:{{reserveation.guestName}} <br>
        GuestEmail: {{reserveation.guestEmail}} <br>
        RoomNumer: {{reserveation.roomNumber}} <br>
        CheckInDate: {{reserveation.checkInDate}} <br>
        CheckOutDate: {{reserveation.checkOutDate}} <br>

        <button (click)="deleteReservation(reserveation.id)">Delte Rserevation</button>
        <button [routerLink]="['/edit', reserveation.id]">Update</button>
    </li>

</ul>

<ng-template #noReservation>
    <p>No reservation available.</p>
</ng-template>
```

Section 15

creating unique ids

- we added a unique id to the reseservation right before we added it to the Reservstions array in the reservation service componet.

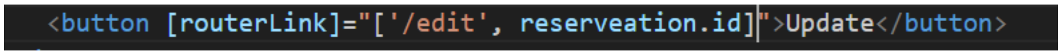


```
addReservation(reservation:Reservation):void{
   reservation.id = Date.now().toString();
   this.reservations.push(reservation);
   localStorage.setItem("reservations",JSON.stringify(this.reservations));
}
```

Section 17

Adding the edit route with parameter

- we now go to the app-rouing.module to add a new path to edit an existing reservation.
- now we created a new path for edit as follows:
- {path:"edit/:id", componet:ReservationFormComponent}
- then we added the link to button we created int he html as follows:



```
  <button [routerLink]="['/edit', reserveation.id]">Update</button>
```

- **to use routerLink we have to import a routerModuel from in the reservation module.**
- **we use the "routerLink" to navigate to a page and also pass in a value.**
- **we use the "router" to route to a diffrent link with out passing in a value.**

Section 18

editing forms

section 19

Combining components from different modules

- if we got our HomeComponet and copy the its selector tag which is <app-home></app-home> and add it to the reservationFormCompent.html and reservtonlistComponetn.html we get an error.
- the first thing that we should do is to got to the Reservations module and import the Home Module and also add it to the "imports" arrays.However we will still get the errors.
- IN order to resole this we have to got to the HomeCompomnet Module and and make "export" the home componet. we do this as shown below.
- **Because the HomeModule is declaring the HomeCompoent it is allowed to export it.**

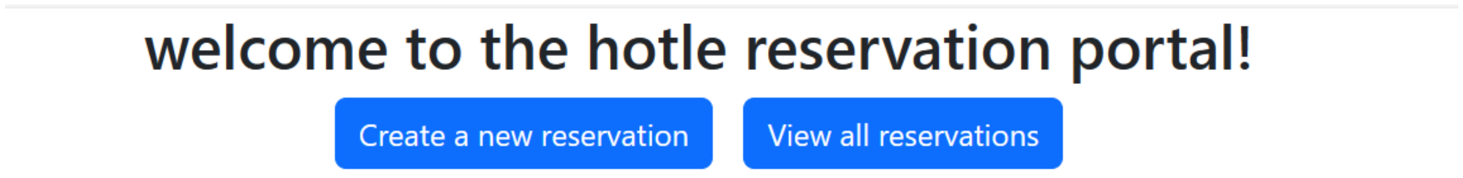
Section 20

Styling the home component

- **install bootstrap again in this application using the command nmp install bootstrap@5.3**
- **the go to "styles.css" and import the bootstrap**

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

- **you can find the path by going to your npm folder.**
- **we then got to out app.compoent.html.**
- **we add a <div class="cointaner"> and surround the router-outlet with it.**
- **we now got to homecompoent.html.**
- **for the button we use a class ="btn btn-primary" to bothn buttons.**
- **to center it we go to the top level div and in the class="text-center".**
- **we can add a margin to it by adding mx-2 to the button.**


welcome to the hotle reservation portal!

Section 21

Styling out Reservation Form Component

- **we go to the reservationForm component.**
- **we begin by adding classes to the inputes.**
- **we add class="for-control" to all the input fileds**

- **then for ever y div we add a class called "mb-3" to add some space to the bottom.**
- **we then add a class called "form-label " to the label.**
- **we apply a grid system by creating a <div class="row"> and placting other divs in side and turning their class to class="col"**

Section 22

styling the reservationlist component using table.

- **for his section we will use a table.**
- **we created a class**