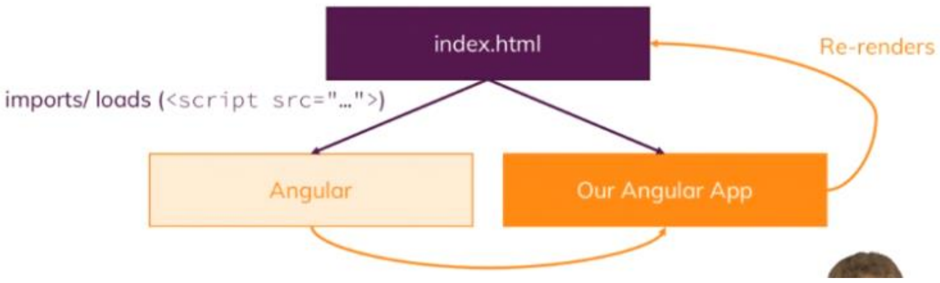
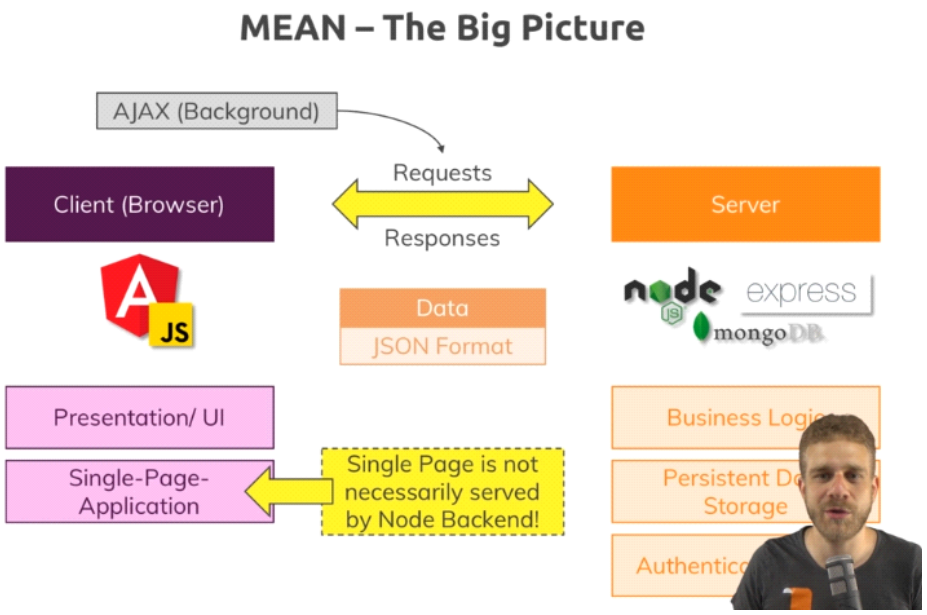
19 February 2022 14:42

Mean Stack Development

- MEAN Stack mean mango DB , Express JS, Angular and Node.js
- Angular is used to build a client side application with beautiful UI. At backend we use node and express and use mango DB for the database. Finally we can get a full stack development using Angular
- What is MEAN?
  - M=> Mango DB
  - E => Express JS (Only works with Nodejs)(Node JS is the language that is used on the server but Express JS is a framework for Node JS that makes the process easier)
  - A => Angular (Client Side technology)(Generate a nice dynamic UI)(Runs entirely on the browser but not on the server)(Angular is a framework of the JavaScript on the browser similar to Express JS for Node JS)
  - N => Node JS (We use this as we need to run some logic on the server)(Ex: Authentication)(Server side code)
- Angular?
  - Single Page applications are generated
  - It's job is to update the UI dynamically when we get a new update immediately (Ex: Facebook)
  - Render UI with Dynamic data
  - Handle USER input
  - Communicate with Backend Services
  - Provides a Mobile app like USER experience
- Node JS
  - Used in the backend service side
  - Server side library
  - JavaScript on the server side
  - JavaScript can run on the browser so node.js add some files that are required to run on the server and hence it runs on the server side also.
  - It Listen to Requests and Send Responses (Angular couldn't do it)
  - Execute Server Side logic (JavaScript can perform it. But as security reasons we should not provide such logic details on front end side. So we use Node JS to write that logic and hide it from the USER developer tools)
  - Interact with the databases and files (Same as the above for security reasons it was done)(It replaces PHP, ASPA.net, Ruby on rails etc.)
- Express JS
  - It is a node framework which simplifies writing Server Side Code and Logic
  - Basically uses the code of Node JS but it offers extra functionality
  - It is in the middleware based funnel requests through functions
  - Includes routing and view rendering
  - Similar to Larval For PHP
- Mango DB
  - It is a NoSQL database which stores "Documents" in "Collections" (Instead of "Records" in "Tables" as in SQL)
  - Store Application Data (Users, Products,…)
  - Enforces no Data Schema or Relations
  - Easily connected to Node/Express (Not to angular)
  - We won't connect to angular because we need to establish security
  - Here we are using Mango DB because of "MEAN" Stack, we can even use the SQL databases with the Node or Express.

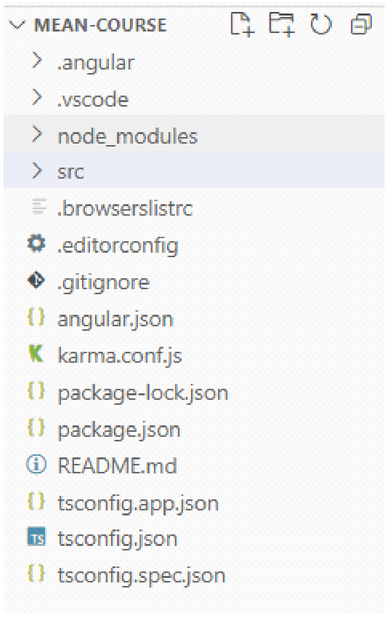


- Single Page Application
  - In angular app we have one root html file that is index.html which can be connected to our node server or other node server
  - We are having index.html which is our home page and according to the conditions our angular app re renders the index.html and generate a new HTML page for the viewer. In this way we no need to reload a page because everything is in the same page
  - This make us to generate applications which are highly responsive.
  - Angular : Front End
  - |
  - Node , Express , Mango DB : Backend
- On Node application we are having our business logic



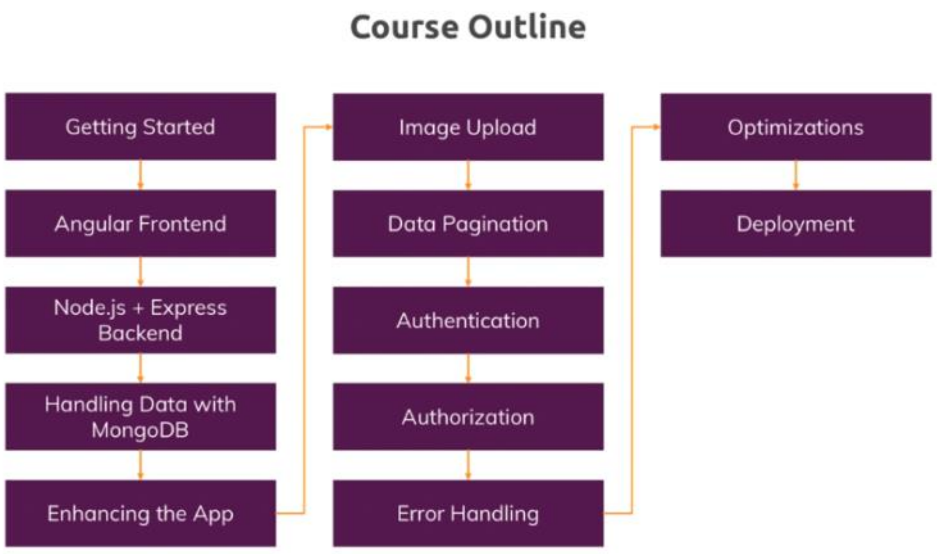
- How to install angular & Node (https://drive.google.com/file/d/1fAqGNm5H7I4A2fUte4muKYuLS4tzbfkq/view?usp=sharing)

- Screenshot (1308).png (NPM commands that I executed on my Command prompt to install angular , NPM)(You can check the current versions of angular and NPM that I have used here)
- (https://drive.google.com/file/d/1QLdTR7tZMqxQ4hZ-G4bDBd_JiCXPbFsV/view?usp=sharing) Installing the IDE

- 

```
MEAN-COURSE
  > .angular
  > .vscode
  > node_modules
  > src
  .browserslistrc
  .editorconfig
  .gitignore
  angular.json
  karma.conf.js
  package-lock.json
  package.json
  README.md
  tsconfig.app.json
  tsconfig.json
  tsconfig.spec.json
```

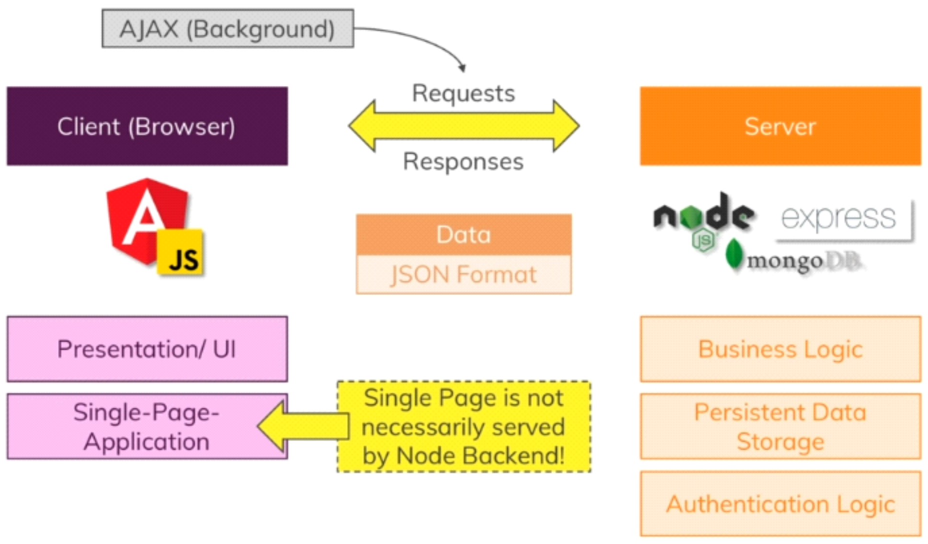
- In the above picture we can see package.json where we are having dependencies and development only dependencies.
- In source folder angular files reside in. and inside the src folder we have app folder that is the core files of the angular
- App.component.html is the mail html page that we are seeing on the browser
- We can add any number of components in src folder similar to app component
- Ng serve executes the index.html file of the project. And index file may contain any number of components which helps us to display the front end.
- The components are accessible within the <component-name>



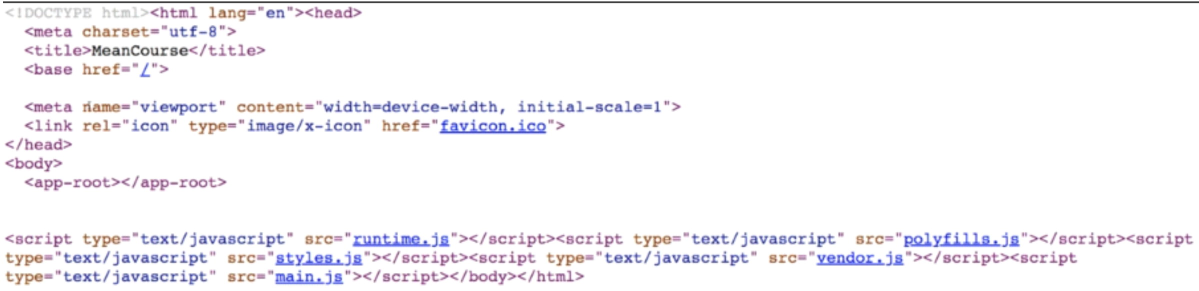
1. Introduction to Angular

19 February 2022 17:35

Introduction of Angular

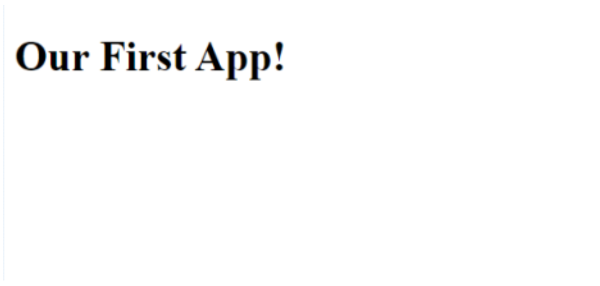


- The web page that we are able to see is written in app.component.html it is a component and that component is called in index.html and hence whatever we have modified inside the app component will be changed in the webpage.
- The page that we see is index.html
- The contents that we see is of different components written in index.html
- In our case it is written in app.component.html



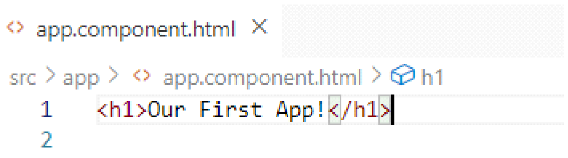
```
<!DOCTYPE html><html lang="en"><head>
  <meta charset="utf-8">
  <title>MeanCourse</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>


<script type="text/javascript" src="runtime.js"></script><script type="text/javascript" src="polyfills.js"></script><script
type="text/javascript" src="styles.js"></script><script type="text/javascript" src="vendor.js"></script><script
type="text/javascript" src="main.js"></script></body></html>
```

- So the following above picture is index.html and when we try to look at it in developer tools we are not able to see any code because we are using angular for to enhance security.
- As we are able to see app-root where it is called a component which is written in a separate file called app.component.html.
- Even the validation code is also absent in normal JavaScript because using express andnode.js we develop security for the logical code.
- In angular we create our own individual element as a component inform of tags



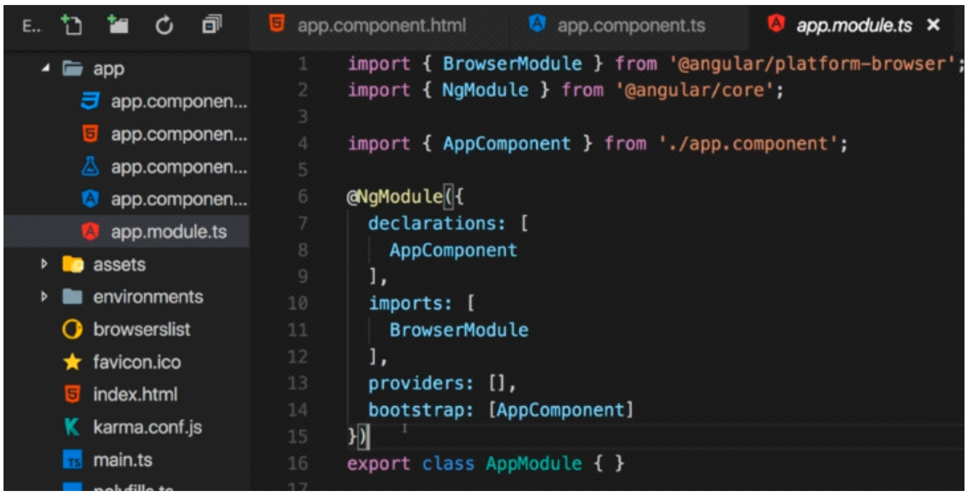
Our First App!

- The following output in the server came from index.html but in real it came from app-root tag which in turn calls the app.component.html because we have declared it's selector through app.component.ts and print it's html code in it dynamically
- This is same through out every component of angular .
- Through the app.component.ts we are calling its templateUrl (HTML page) and styleUrls (CSS page)

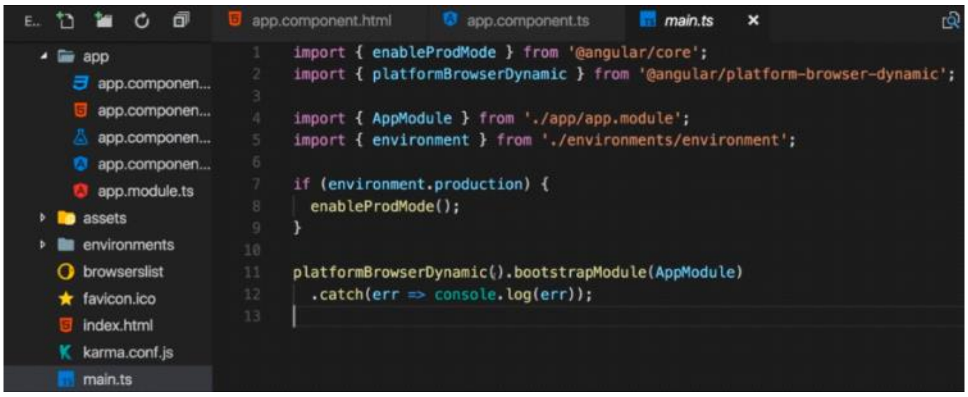


```html
<h1>Our First App!</h1>
```

- Here below we are having app.module.ts , where we declare all the components that we want to use in our current angular application. And in bootstrap array we are giving a hint to the index.html to call that component and start designing it's page as a first page



```typescript
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 { }
```



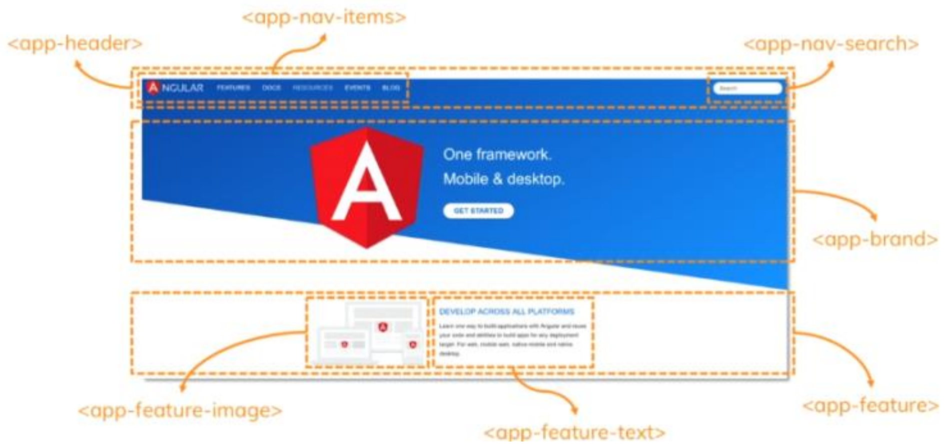
```typescript
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));

```

- First of all after angular starts executing it goes to main.ts first and then import all the modules that are required to start the application .
- Here we can see two modules platformBrowserDynamic and AppModule in the above picture , these two modules are responsible for following actions
- platformBrowserDynamic => this started the initiation to start the app module (remember app module means that is an component that we have created previously)
- platformBrowserDynamic with the help of bootstrapModule initiates AppModule
- First initiation starts from main.ts then goes to app.module.ts and check the bootstrap array and then it redirects to the selector that it requires in the following component and return the html and CSS page that index.html needs

2. Angular Components

20 February 2022 09:44

- Everything in angular works with components
- Express the entire page with components
- Advantage of components are we can build small building blocks of UI which you can use it anywhere and even reuse because some components appear more often. So we can reuse it without writing code again using this components.

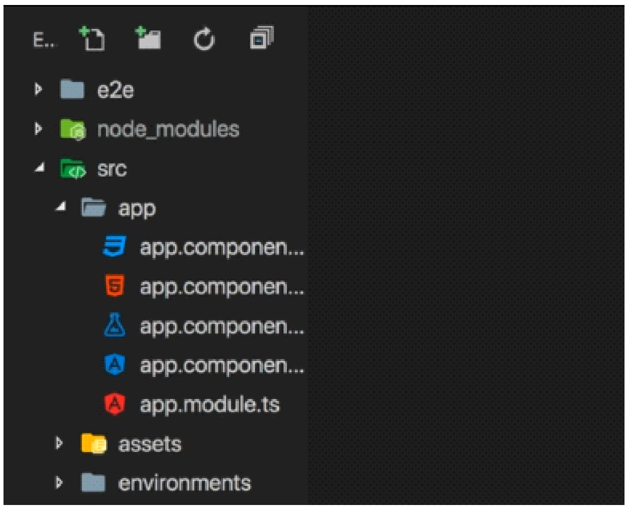


- Angular thinks everything in components. The above picture is the example for it. Each element of the page is a component. So we can easily modify it without effecting other components and even reuse it.

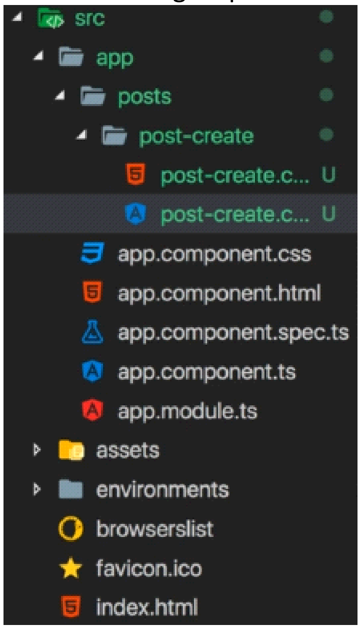
3. Adding our First Component

20 February 2022 09:57

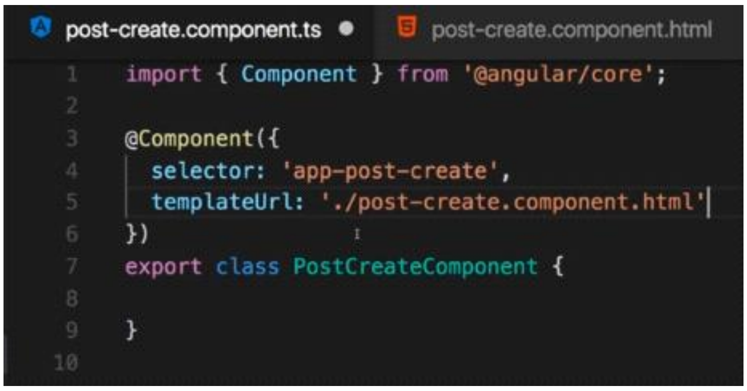
- Here in this course we want to do an application where we have an app and in that app, we have to edit , add , view the posts. And we want to do all those functions and logics in different components. Hence here is our first step we are adding our first component.
- Initially we are having the following default component named as app component in the angular



- Here if we want to create any new component in the base app component we have to do the following steps



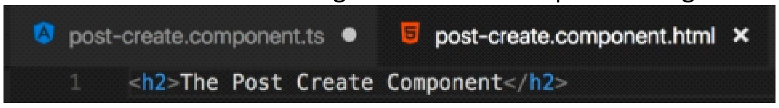
- First we have to create a sub folder to create a component. In our case we want to create a app functionality similar to Instagram or Facebook. So we are creating a sub folder posts which contain all the new component's necessary to create the app.
- One of the such sub-sub component we are defining now is post-create.
- Now we have to define files necessary to use that component.
- The files are post-create.component.ts and post-create.component.html.
- Where we can define html and CSS inside the typescript file , but for scalability we define it inside separate files.
- We define the global tag inside the selector of the typescript file and html inside the templateUrl
- After creating post-create.component.ts and post-create.component.html we have fill the code inside that.



post-create.component.ts ● | post-create.component.html

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

@Component({
  selector: 'app-post-create',
  templateUrl: './post-create.component.html'
})
export class PostCreateComponent {

}
```

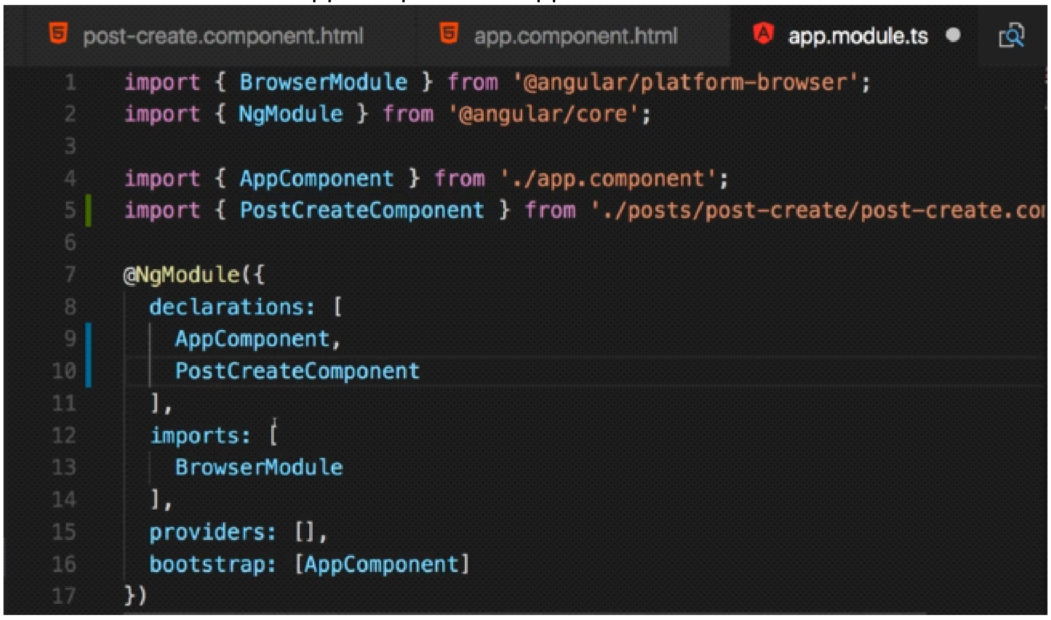
- The above picture is about post-create.component.ts.
- After creating a sub component under angular component we must define the type script file.
- First we have to create a class in the type script file. To use it's functionality in app component.
- The name of the current class is PostCreateComponent which is suggested name for post-create.component.ts.
- The nomenclature e for the naming is to create a class which starts with the name of the component name with suffix component "component_ name "+Component.
- After defining the class name we must export that class to the app module and define current class as component by using decorator (@Component).
- Under it we are having selector and templateUrl tag which is earlier mentioned



post-create.component.ts ● | post-create.component.html ×

```
<h2>The Post Create Component</h2>
```

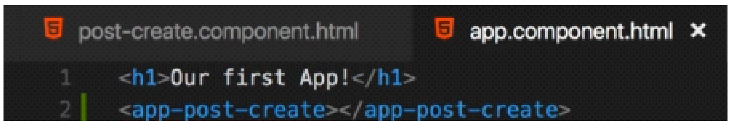
- We are defining the following HTML in post-create component.
- Remember we have called app-root inside the index.html , it mean we are calling app component inside index.html that app component take a reference of app.module.ts to call all it's component's and sub component's. Hence we have to define every component that we have created inside the app component in app.module.ts otherwise we will face the undefined error.



post-create.component.html | app.component.html | app.module.ts ●

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

import { AppComponent } from './app.component';
import { PostCreateComponent } from './posts/post-create/post-create.co

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

- After doing those operations we can add that component on the main app.component.html and hence it will also add in index.html



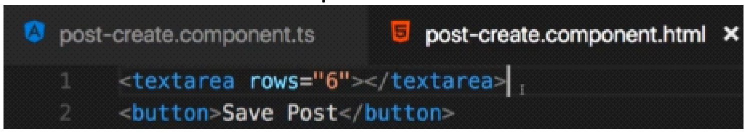
post-create.component.html | app.component.html ×

```
<h1>Our first App!</h1>
<app-post-create></app-post-create>
```

4. Listening to Events

20 February 2022 14:46

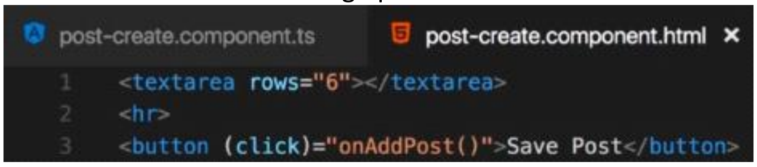
In the previous note , we have created post-create component and in that we are creating a text box which is useful to create a post and a button which is useful to submit the request.



post-create.component.ts | post-create.component.html

```html
<textarea rows="6"></textarea>
<button>Save Post</button>
```

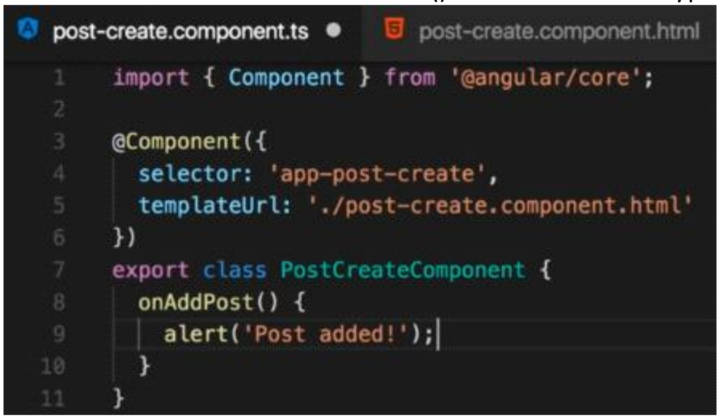
- But here we didn't give any functionality for these button and the following text box.
- Angular does the functionality at the back end.
- To add an listener to it , we are binding an event to it.
- Event binder is provided by angular
- Syntax: (eventWeWantToPerform)="ActionThatWeWantToPerform".
- Other wise we can add a method in place of action and that method must be defined in the typescript class of that component.
- These are the following operations that we have to do to add an event to the element



post-create.component.ts | post-create.component.html

```html
<textarea rows="6"></textarea>
<hr>
<button (click)="onAddPost()">Save Post</button>
```

- Here in post-create component we added an event to the button, click operation .
- That click calls the onAddPost() method that is in type script file of post-create component.



post-create.component.ts | post-create.component.html

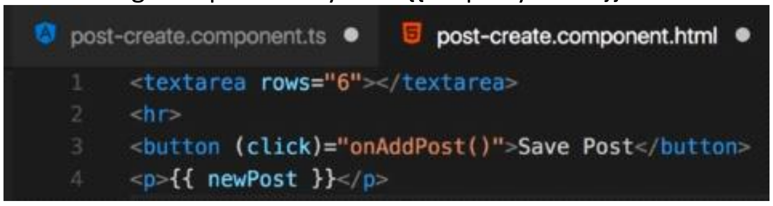
```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-post-create',
  templateUrl: './post-create.component.html'
})
export class PostCreateComponent {
  onAddPost() {
    alert('Post added!');
  }
}
```

- The button of the html file class the onAddPost() of the typescript file of the post-create component. And hence we get an output of an alert("Post added!")
- This is the sample of event triggering

5. Output the Content

20 February 2022 15:37

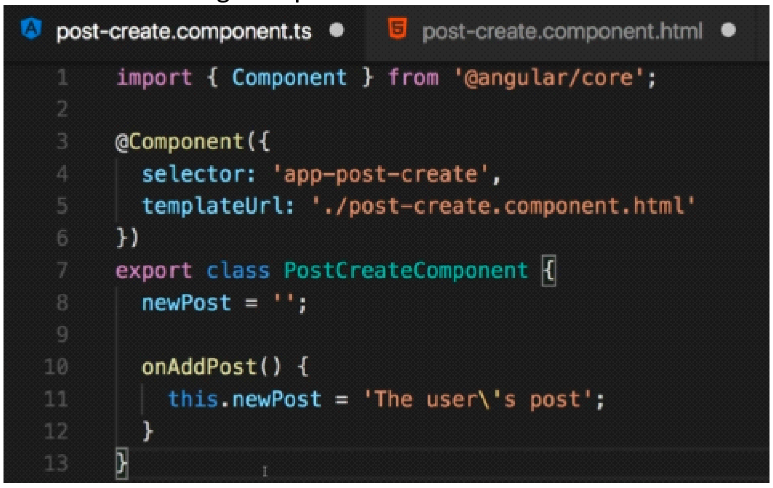
- Way 1: (We want to edit the whole text at the frontend using backend modification of the text)(Modifying occurs at backside)(We use curly brackets for binding)
- Previously we have created a textbox with and event enabled button , but now we want to output some text from the textbox to the page for example we want to update the paragraph tag. So first we need to get the text from the textbox to a variable inside the typescript class. In our case we use newPost as the variable. And update it in the function as we required , now at the typescript level a value is received from the html page to it. And we update it in the function down there. Now we have to return the content back to the html page. So we add hooks to receive at the html level from the typescript function.
- Html page -> sends request to typescript page using event binder -> value is updated in the function that html called -> the updated value is returned to html using hooks.
- Here we are using string interpolation to update at the html level.
- The value get's updated at typescript level and sends back to html level and print using string interpolation.
- String interpolation syntax: {{Property name}}



```
1  <textarea rows="6"></textarea>
2  <hr>
3  <button (click)="onAddPost()">Save Post</button>
4  <p>{{ newPost }}</p>
```

HTML file

- Line 3: Event binding
- Line 4: String interpolation

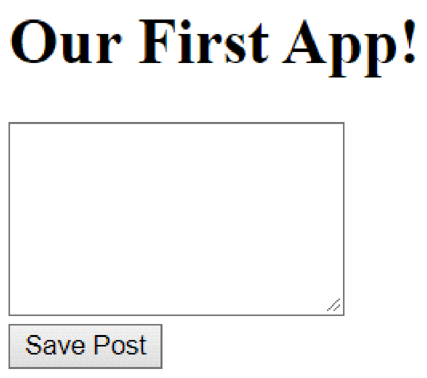


```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-post-create',
5     templateUrl: './post-create.component.html'
6   })
7   export class PostCreateComponent {
8     newPost = '';
9
10    onAddPost() {
11      this.newPost = 'The user\'s post';
12    }
13  }
```

TypeScript File

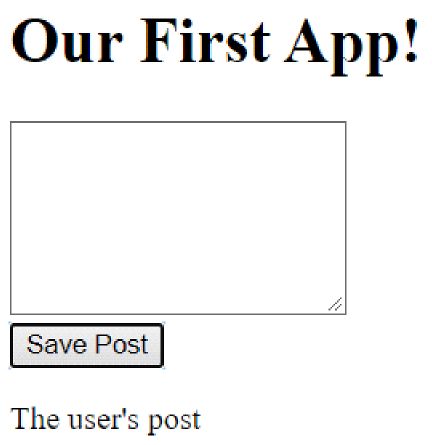
- Line 8: Defining the property
- Line 10: Defining the method

- Initially the output without any event's will be in this static way



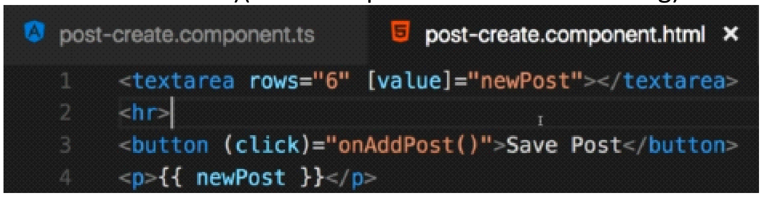
Our First App!

- After writing these event's and using string interpolation then after clicking this "Save Post" button we get a text by updating the html page.
- This looks like some dynamic nature came into our site



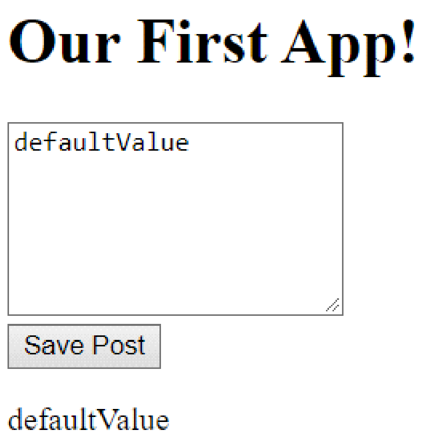
- This is the following output.

Way 2: (Modify the data at the backend parallel to the front end data modification)(Modifying occurs at front side)(We use square brackets for binding)

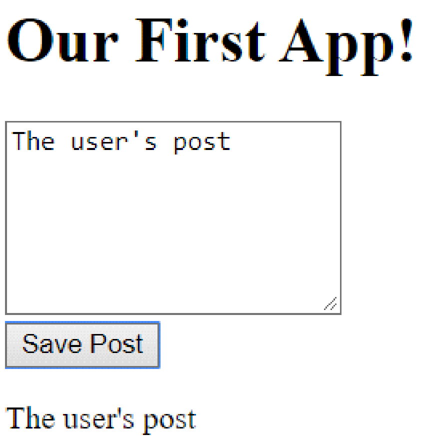


```
<textarea rows="6" [value]="newPost"></textarea>
<hr>
<button (click)="onAddPost()">Save Post</button>
<p>{{ newPost }}</p>
```

- Here we are using [method]="property" syntax where method is assigned to the html tag and the property is designed in the component typescript file. Hence we get the value of newPost which is assigned in the backend in the textbox of the front end.
- Initially the screen looks like this



- Here we can see "defaultValue" two times because the component is binded in other way (backend to the front end) and secondly the paragraph is interpolated.
- After clicking the save post button it changes in this way

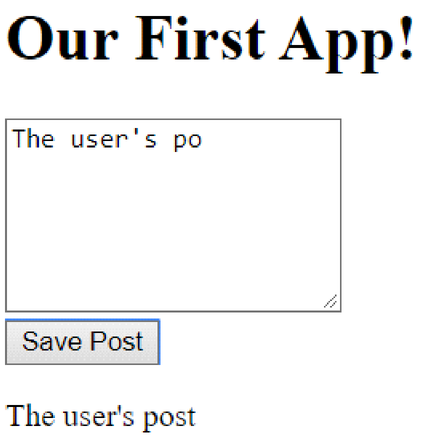

Our First App!

The user's post

Save Post

The user's post

- The first way that we have done is, changing the current value from the backend and the second method is importing the backend value to frontend side
- The backdrop of the method that we are using , it is not synchronized.


Our First App!

The user's po

Save Post

The user's post

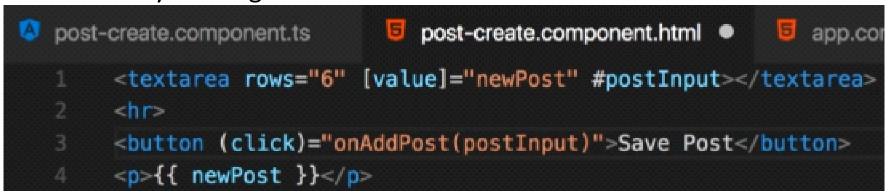
We can see the above picture is not synchronized..

6. Getting User Input and install angular material

20 February 2022 17:32

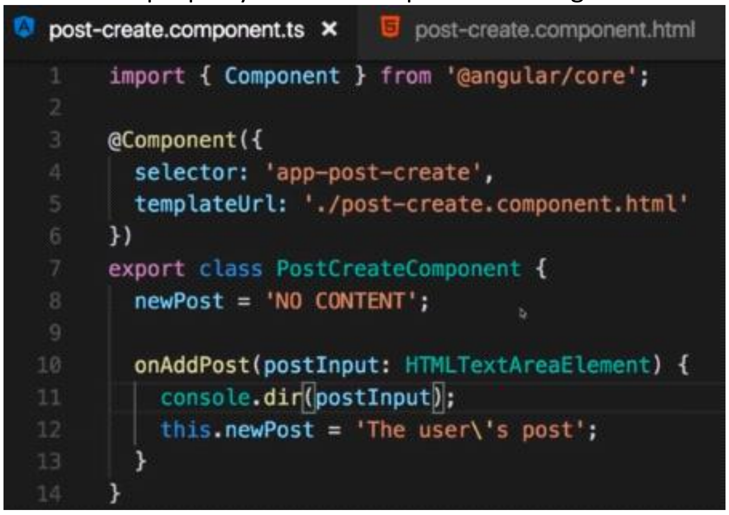
Way 1: Sync two sides only upon some event occurs

- In this page we are going to pass the value from the front end to the backend
- We are creating a local reference (#localref) and passing that inside the function of typescript class by binding that function to the button click



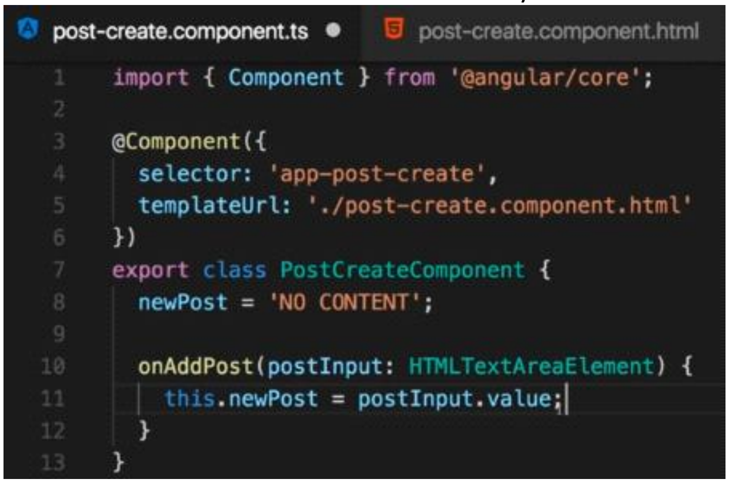
```html
<textarea rows="6" [value]="newPost" #postInput></textarea>
<hr>
<button (click)="onAddPost(postInput)">Save Post</button>
<p>{{ newPost }}</p>
```

- Here in the typescript file we are having a method which receives the postInput local reference variable which is of HTMLTextAreaElement.
- Here console.dir() prints all properties of the postInput (Text Area). There we could see the value property. Which is required to change the value of the backend.



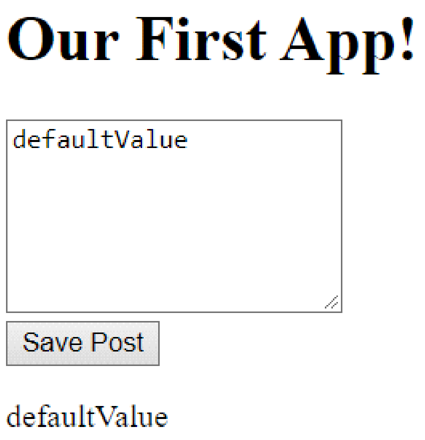
```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-post-create',
  templateUrl: './post-create.component.html'
})
export class PostCreateComponent {
  newPost = 'NO CONTENT';

  onAddPost(postInput: HTMLTextAreaElement) {
    console.dir(postInput);
    this.newPost = 'The user\'s post';
  }
}
```

- Now we are performing actions in such a way what ever value that we have entered in the textbox will be appeared below the button after we click the button. This signifies there established a sync between front end and backend.
- Initially backend communicate with the front end and print's a value and then after editing the text by clicking button the front end communicate with the backend and change the value of the backend. This establishes two way connection.

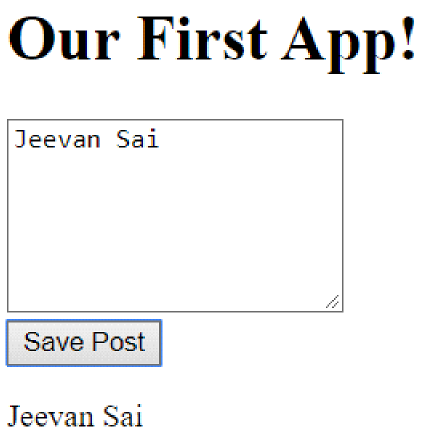


```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-post-create',
  templateUrl: './post-create.component.html'
})
export class PostCreateComponent {
  newPost = 'NO CONTENT';

  onAddPost(postInput: HTMLTextAreaElement) {
    this.newPost = postInput.value;
  }
}
```

- Finally two way connection established
- Initial output on the browser

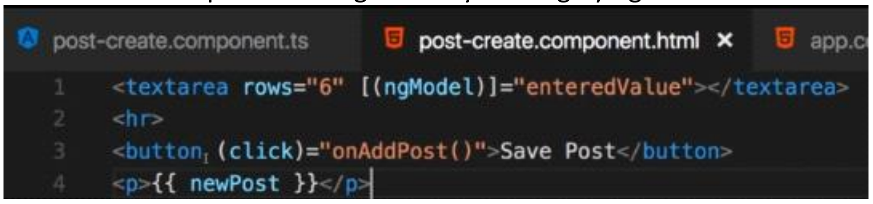


- If we try to change value in the Text Area and click the save post button the value get's updated in the backend and updated in front end automatically. Due to string interpolation at the front end.



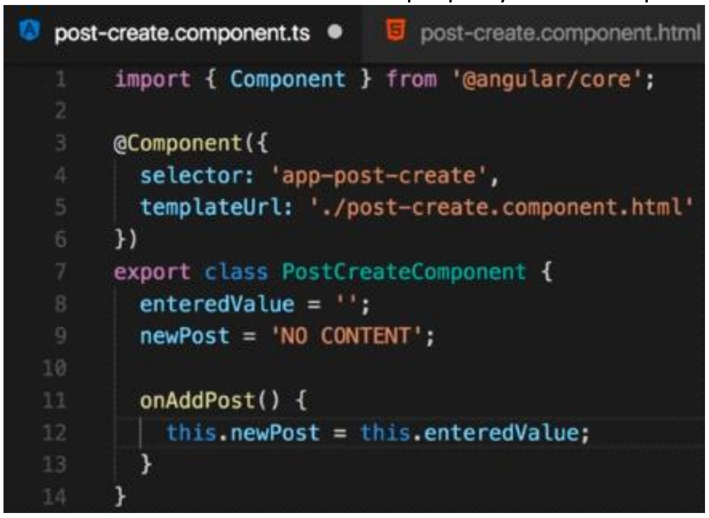
Way 2: Sync two sides with ngModel

- We do the process using two way binding by ngModel directive



```
<textarea rows="6" [(ngModel)]="enteredValue"></textarea>
<hr>
<button (click)="onAddPost()">Save Post</button>
<p>{{ newPost }}</p>
```

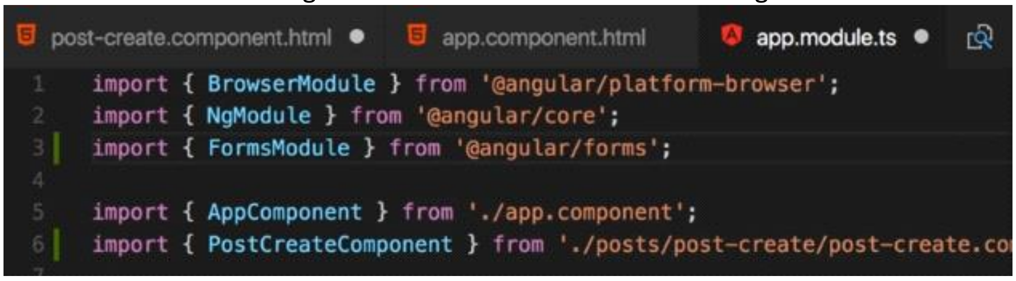
- Here we are using ngModel in post-create component for two way binding.
- ngModel must be surrounded by square and circular brackets which is used to receive and send the information, hence it is called two way binding.
- And here enteredValue is a property defined in post-create.component.ts.



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

@Component({
  selector: 'app-post-create',
  templateUrl: './post-create.component.html'
})
export class PostCreateComponent {
  enteredValue = '';
  newPost = 'NO CONTENT';

  onAddPost() {
    this.newPost = this.enteredValue;
  }
}
```

- The data that we sent through ngModel is received by this typescript file and when we click the button the default value of newPost will be modified by the enteredValue of the textbox. As we used string interpolation at the front end side newPost get's automatically updated.

- text Area text change+ button click => newPost text change at the front end



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

import { AppComponent } from './app.component';
import { PostCreateComponent } from './posts/post-create/post-create.co
```

- ***To use ngModel in the app component we must import FormsModule in app.module.ts. And add FormsModule in the imports***
- enteredValue is the identifier of the textbox's text at the backend side.
- We get the same output as the previous.

Don't forget to install angular material which is similar to bootstrap. But we must enter the logic behind those components.

The following command must be executed in command prompt to install angular material

```
ng add @angular/material
```

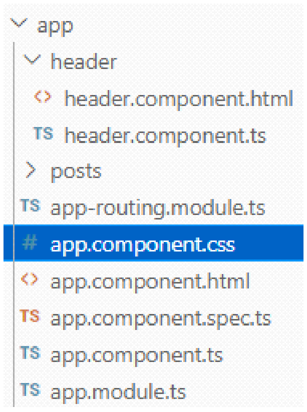
The following command make sure to add some dependencies in package.json they are angular material and angular cdk and in angular.json we add a default theme that is indigo pink. And in app.module.ts we add BrowserAnimationModule which helps in developing animations in the angular app. And in index.html we have added some font styles.

Visit Angular Material : https://material.angular.io

Here as we used textarea previously , once check about text area inside the documentation. We will find textarea is using matInput field . matInput is a form feature which is available under MatInputModule. And we have to add it under app.module.ts and imports area of app.module.ts

7. Adding Toolbar

21 February 2022 22:57



- Here we need to add a toolbar, in the sense it is called header. And header consists of two files one html and one typescript file.
- We are going to use inbuilt angular material to create the toolbar , hence we have to add related toolbar module of angular material into angular in app.module.ts

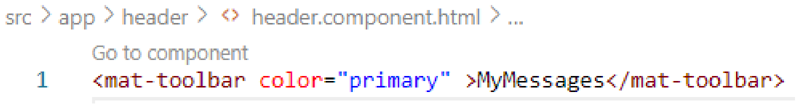
```
import { MatToolbarModule } from '@angular/material/toolbar';
import { AppRoutingModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { PostCreateComponent } from './posts/post-create/
post-create.component';
import { HeaderComponent } from './header/header.component';

import {} from '@angular/material';

@NgModule({
  declarations: [
    AppComponent,
    PostCreateComponent,
    HeaderComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MatInputModule,
    MatCardModule,
    MatButtonModule,
    MatToolbarModule
  ],
```

- Here we are importing MatToolbarModule to use in our html as a header file. And after importing as a import statements we need to import it under NgModule also. And we created a component called header to use it we have to declare it in angular under declarations "HeaderComponent". Now we can proceed with the code.

src > app > header > TS header.component.ts > HeaderComponent

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

@Component({
  selector:'app-header',
  templateUrl:'./header.component.html'
})
export class HeaderComponent{}

```

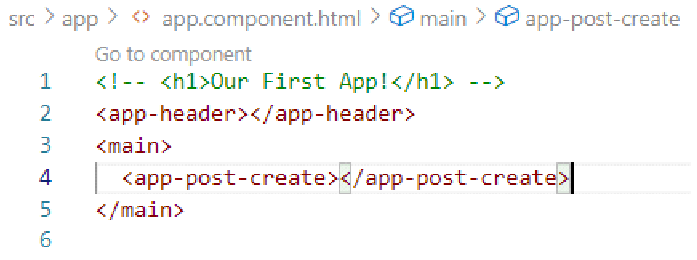
- The following is header.component.ts file where the html of that page is in header.component.html



src > app > header > <> header.component.html > ...

Go to component
```
<mat-toolbar color="primary" >MyMessages</mat-toolbar>
```

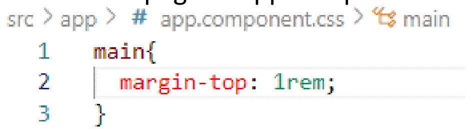
mat-toolbar is the tag that is provided by the AngularMaterial we imported it's file in the app.module.ts



src > app > <> app.component.html > main > app-post-create

Go to component
```
<!-- <h1>Our First App!</h1> -->
<app-header></app-header>
<main>
  <app-post-create></app-post-create>
</main>

```

And add that component to the app.component.html . So it in turn adds to the index.html file. And due to the spacing issue between app-header and app-post-create we are adding a margin between them in CSS page of app.component.css

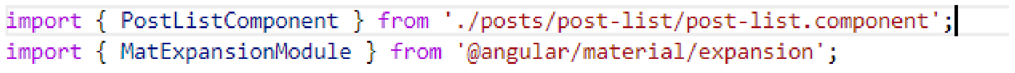


src > app > # app.component.css > main
```
main{
  margin-top: 1rem;
}
```

Now the tool bar part is done. We want to add new posts now for that we need expansion panels of angular material. So first import that in app.module.ts

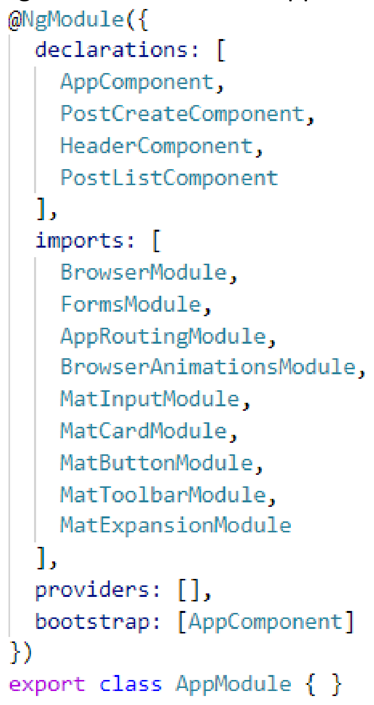
What ever module that we want to use in our angular app we have to import it in app.module.ts and whatever change we need to update on the front end we need to update in app.component.html(overview updating of components).

First import Expansion module in app.module.ts



```
import { PostListComponent } from './posts/post-list/post-list.component';
import { MatExpansionModule } from '@angular/material/expansion';
```

Import the MatExpansionModule and declare it in the declarations and import it in the imports of NgModule directive of app module class



```
@NgModule({
  declarations: [
    AppComponent,
    PostCreateComponent,
    HeaderComponent,
    PostListComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MatInputModule,
    MatCardModule,
    MatButtonModule,
    MatToolbarModule,
    MatExpansionModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

We imported the MatExpansionModule and now we have to use it. We are going to use it in post-list.component.ts so we need to create the html , TS and CSS files of that component. And import that component in app.module.ts and declare that component in the declarations of app.module.ts

Create new component post-list.component.ts

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

@Component({
  selector:'app-post-list',
  templateUrl:'./post-list.component.html',
  styleUrls:['./post-list.component.css']
})
export class PostListComponent{
  posts=[
    {title:'First Post',content:'This is the first post\'s content'},
    {title:'Second Post',content:'This is the second post\'s content'},
    {title:'Third Post',content:'This is the third post\'s content'},
  ];
}
```

We have defined selector as app-post-list

Now we have to define it's html and CSS files here.
The following is post-list.component.html

```
<mat-accordion>
  <mat-expansion-panel>
    <mat-expansion-panel-header>
      The expansion panel
    </mat-expansion-panel-header>
    <p>I'm in expansion panel</p>
  </mat-expansion-panel>
</mat-accordion>
```

- Mat accordion is the main tag that we must use while creating expansion panel
- Mat expansion panel is the main tag
- And to define the expansion panel header we use mat expansion panel header
- Under it we define the contents of the expansion.
- And to establish a margin between post-create or post-list in post-list.component.css we create a margin top and to consider whole expansion panel as a block we define display as block
- The margin top establishes the margin between the post-list and post-create panel

src > app > posts > post-list > # post-list.component.css > :host

```
:host{
  display:block;
  margin-top:1rem;
}
```

- And to establish the styling for the whole page we editing app.component.css
- The following app.component.css establishes the gap between header and the body

src > app > # app.component.css > main

```
main{
  margin-top: 1rem;
  width:80%;
  margin:auto;
}
```

- Add post-list in app.component.html
- If we want to set up margin for the whole app we must edit in app.component.css and if we want for only between components we must use that particular component . css

src > app > <> app.component.html > main > app-post-list

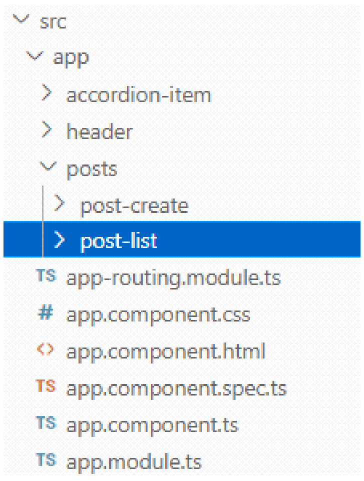
Go to component

```
<!-- <h1>Our First App!</h1> -->
<app-header></app-header>
<main>
  <app-post-create></app-post-create>
  <app-post-list></app-post-list>
</main>
```

8. Structural directive with mat-accordion

23 February 2022 17:16

Previously we have created a mat-accordion where we are getting group of drop downs. Now in this note we want to use the mat accordion conditionally using structural directives. For examples , if we have any posts to display mat accordion display it or else it display "No posts are available". Hence we are introducing if else and for statements in this notebook.



```
src
  app
    accordion-item
    header
    posts
      post-create
      post-list
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
```

- This is the hierarchy we are going to discuss overall
- First let's discuss about post-create component

```
<mat-card>
  <mat-form-field>
    <textarea matInput rows="6" [(ngModel)]="enteredValue"></
    textarea>
  </mat-form-field>
  <button
  mat-raised-button
  color="accent"
  (click)="onAddPost()">Save Post</button>
</mat-card>
```

- This is post-create.component.html here we use matcard to create a card around it. It belongs to the angular material module. And mat form filed is necessary to perform the form actions. matInput of text area creates a different look for the textArea box. And ngModel is used to two way connecting. And mat-raised-button gives a different css style for the button , it is of accent color. Where we given functionality for that button by calling add post.

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

@Component({
  selector:'app-post-create',
  templateUrl:'./post-create.component.html',
  styleUrls: ['./post-create.component.css']
})
export class PostCreateComponent{
  newPost="defaultValue";
  enteredValue="";
  onAddPost(){
    this.newPost=this.enteredValue;
  }
}
```

- This is post-create.component.ts
- Where the back end value can be modified in the front end and can be stored in the backend

it's two way binding.

```
mat-form-field,
textarea{
  width: 100%;
}
```

- This is post-create.component.css
- In post-list component we are having the mat accordion where we are having posts and their content's in the drop down. If there are no posts that component should print "No Posts Are available"

```
<mat-accordion multi="true" *ngIf="greaterThan(items.length)
else nopost">
  <app-accordion-item *ngFor="let item of items" [item]="item">
  </app-accordion-item>
</mat-accordion>
<ng-template #nopost><p class="info-text mat-body-1">No Posts
are available</p></ng-template>
```

- This is post-list.component.html
- Where in mat-accordion we are checking the backend by sending number of posts is greater than or less than 0 and if greater than 0, we need to print accordion else we need print "No posts are available".
- Ng-template is used to create a templates which may have constraint or no constraint
- We can see app-accordion here that is also a component which print's all the posts. So we are sending the necessary information to that component. App-accordion is similar to template to print a accordion. We are iterating item by item and passing it to the accordion-item.component.html .

```
<mat-expansion-panel>
  <mat-expansion-panel-header>
    <mat-panel-title>
      {{item.title}}
    </mat-panel-title>
  <!-- <mat-panel-description></mat-panel-description> -->
  </mat-expansion-panel-header>
  {{item.content}}
</mat-expansion-panel>
```

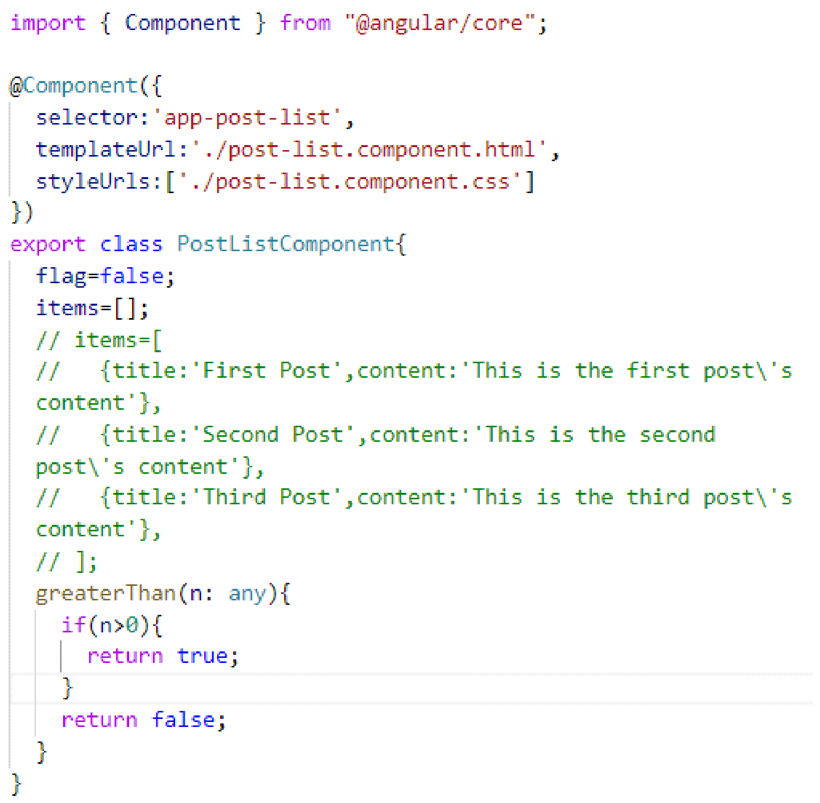
- It is accordion-item.component.html
- Where it input's each item from post-list.component.html and print it here in this component.
- It mean accordion-item is the sub component of post-list component.
- There are multiple accordion-item's inside a post-list.component.html

src > app > posts > post-list > # post-list.component.css > ...

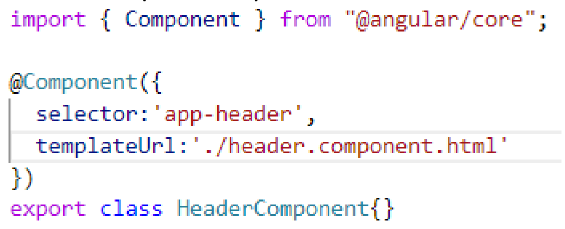
```
:host{
  display:block;
  margin-top:1rem;
}
.info-text{
  text-align:center;
}
```

- This is post-list.component.css

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

@Component({
  selector:'app-post-list',
  templateUrl:'./post-list.component.html',
  styleUrls:['./post-list.component.css']
})
export class PostListComponent{
  flag=false;
  items=[];
  // items=[
  //   {title:'First Post',content:'This is the first post\'s
  content'},
  //   {title:'Second Post',content:'This is the second
```


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

@Component({
  selector:'app-post-list',
  templateUrl:'./post-list.component.html',
  styleUrls:['./post-list.component.css']
})
export class PostListComponent{
  flag=false;
  items=[];
  // items=[
  //   {title:'First Post',content:'This is the first post\'s
  content'},
  //   {title:'Second Post',content:'This is the second
  post\'s content'},
  //   {title:'Third Post',content:'This is the third post\'s
  content'},
  // ];
  greaterThan(n: any){
    if(n>0){
      return true;
    }
    return false;
  }
}
```

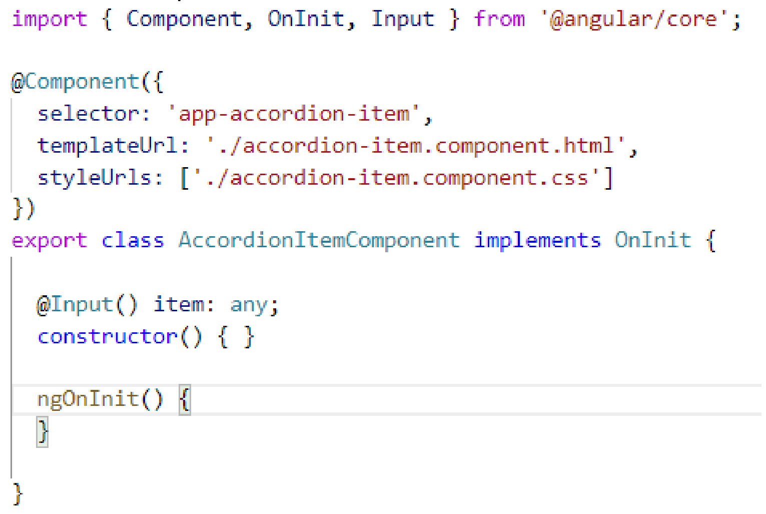
- This is post-list.component.ts which is called by post-list.component.html for greaterThan verification. If items are greater than 0 then print the posts or print "No Posts Are Available"
- This is previously discussed header module


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

@Component({
  selector:'app-header',
  templateUrl:'./header.component.html'
})
export class HeaderComponent{}
```

- This is header.component.ts

```
<mat-toolbar color="primary" >MyMessages</mat-toolbar>
```

- This is header.component.html
- Now we need to discuss the accordion-item component. Already accordion-item.component.html discussed above


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

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

  @Input() item: any;
  constructor() { }

  ngOnInit() {
  }

}
```

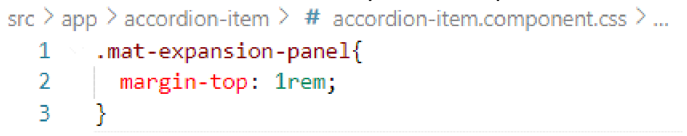
- This is accordion-item.component.ts
- Here this is used as sub accordions of original mat-accordion. For this , the following type of class is necessary or else we are getting an error
- As usual declare the new components in the app.module.ts

- Output is when ever we have posts in the item array of post-list.component.ts we display in accordion format and if we are not having any posts when the length of the array is less or equal to 0 then just print No Posts Available.

9. Creating Posts with Property and Event binding

23 February 2022 22:59

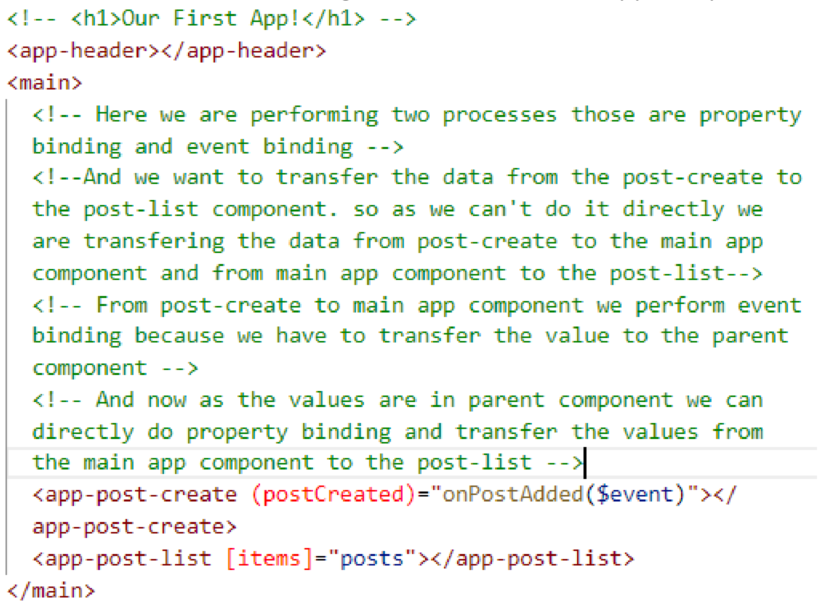
- The Operation that we want to do: We are having a two textboxes with one for title and other for the content. Where whenever we fill the following textboxes and click the save post button that is provided at the bottom. We are going to add a post and display below of the textboxes and button in form of accordion. Let's see the process that we have used here.
- Here we didn't create any new components so we no need to disturb the app.module.ts



src > app > accordion-item > # accordion-item.component.css > ...

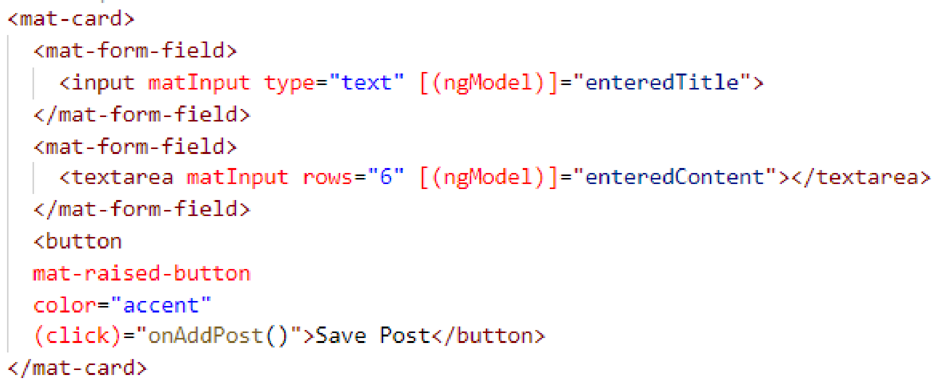
```css
.mat-expansion-panel{
  margin-top: 1rem;
}
```

- Accordion-item.component.css
- The following file is added because when we display the accordions as per the previous chapter there is no space between the posts. So now we are adding a margin between them.
- As per what we learnt in the chapter 5 the property and event binding. we do the same here. But with user defined components. We generally perform those event and property binding on the predefined html tags.
- [] => property binding
- () => event binding
- Based on this knowledge let's first check the app.component.html and see the code first



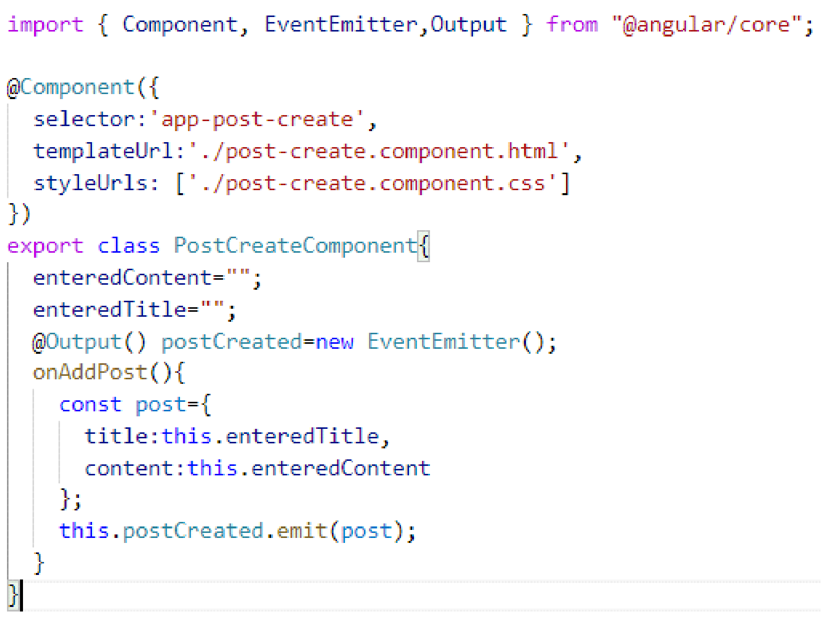
```html
<!-- <h1>Our First App!</h1> -->
<app-header></app-header>
<main>
  <!-- Here we are performing two processes those are property
  binding and event binding -->
  <!--And we want to transfer the data from the post-create to
  the post-list component. so as we can't do it directly we
  are transfering the data from post-create to the main app
  component and from main app component to the post-list-->
  <!-- From post-create to main app component we perform event
  binding because we have to transfer the value to the parent
  component -->
  <!-- And now as the values are in parent component we can
  directly do property binding and transfer the values from
  the main app component to the post-list -->
  <app-post-create (postCreated)="onPostAdded($event)"></
  app-post-create>
  <app-post-list [items]="posts"></app-post-list>
</main>
```

- The above image is app.component.html
- Now let's check what's happening in post-create component



```html
<mat-card>
  <mat-form-field>
    <input matInput type="text" [(ngModel)]="enteredTitle">
  </mat-form-field>
  <mat-form-field>
    <textarea matInput rows="6" [(ngModel)]="enteredContent"></textarea>
  </mat-form-field>
  <button
  mat-raised-button
  color="accent"
  (click)="onAddPost()">Save Post</button>
</mat-card>
```

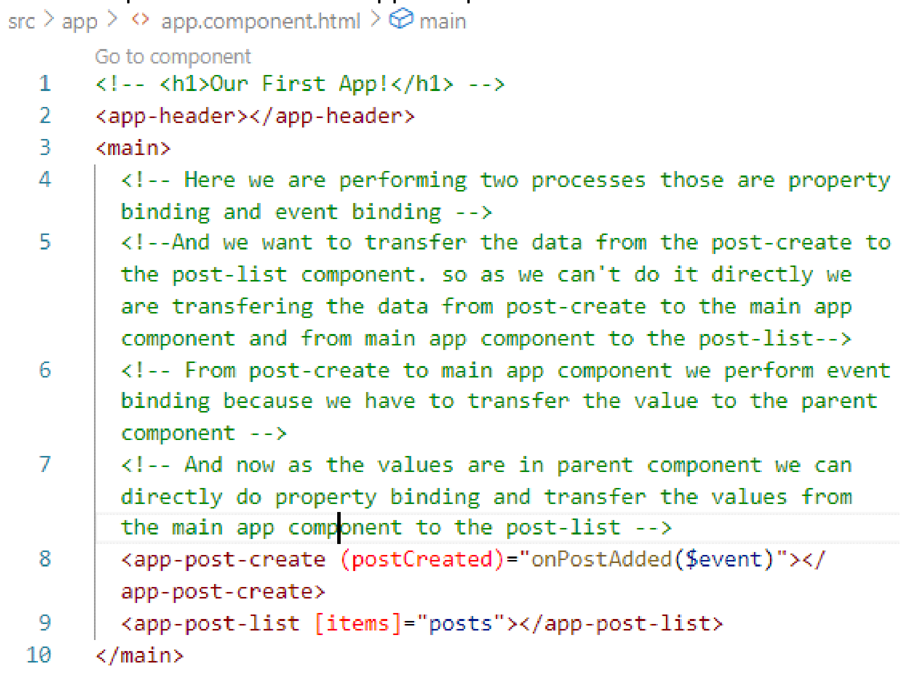
- This is post-create.component.html



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

@Component({
  selector:'app-post-create',
  templateUrl:'./post-create.component.html',
  styleUrls: ['./post-create.component.css']
})
export class PostCreateComponent{
  enteredContent="";
  enteredTitle="";
  @Output() postCreated=new EventEmitter();
  onAddPost(){
    const post={
      title:this.enteredTitle,
      content:this.enteredContent
    };
    this.postCreated.emit(post);
  }
}
```

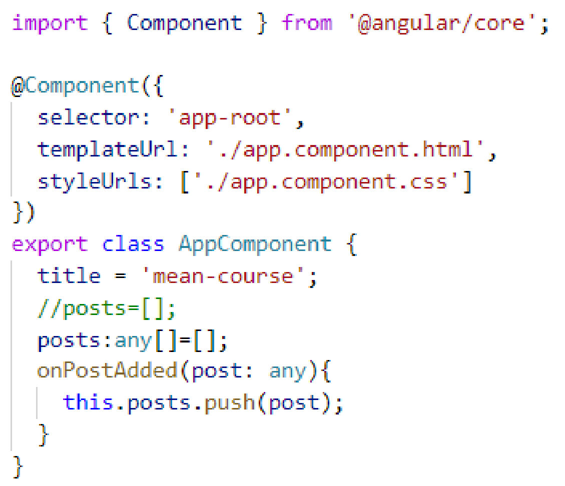
- This is post-create.component.ts

- Now when we see these two files we have included two way binding from the front end to the backend to the title and content field. And specially we can see @Output() it mean we are creating an emitter , so that we can emit the newly created post from the post-create component to the main app component.



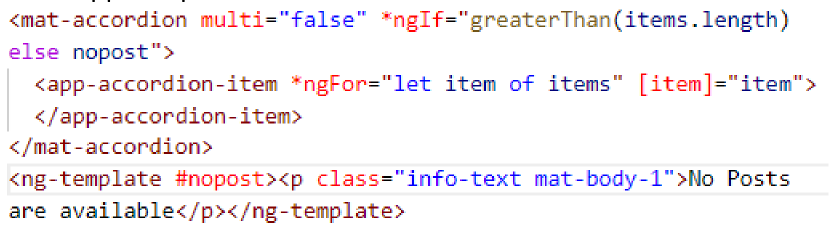
```
<!-- <h1>Our First App!</h1> -->
<app-header></app-header>
<main>
  <!-- Here we are performing two processes those are property binding and event binding -->
  <!--And we want to transfer the data from the post-create to the post-list component. so as we can't do it directly we are transfering the data from post-create to the main app component and from main app component to the post-list-->
  <!-- From post-create to main app component we perform event binding because we have to transfer the value to the parent component -->
  <!-- And now as the values are in parent component we can directly do property binding and transfer the values from the main app component to the post-list -->
  <app-post-create (postCreated)="onPostAdded($event)"></app-post-create>
  <app-post-list [items]="posts"></app-post-list>
</main>
```

- This is app.component.html
- Here as we know previously we have emitted an output named postCreated with a new post in it to the main app component. And here in app component it got considered using this step (postCreated) and after considering and accepting the postCreated. We are passing it to the onPostAdded which is in app.component.ts where this file consists data of all the posts. And here the new post going to add in it. All the posts are stored in variable posts . Here we must define the datatype of posts for sure or else we are getting an error. Now let's see the typescript of app component.



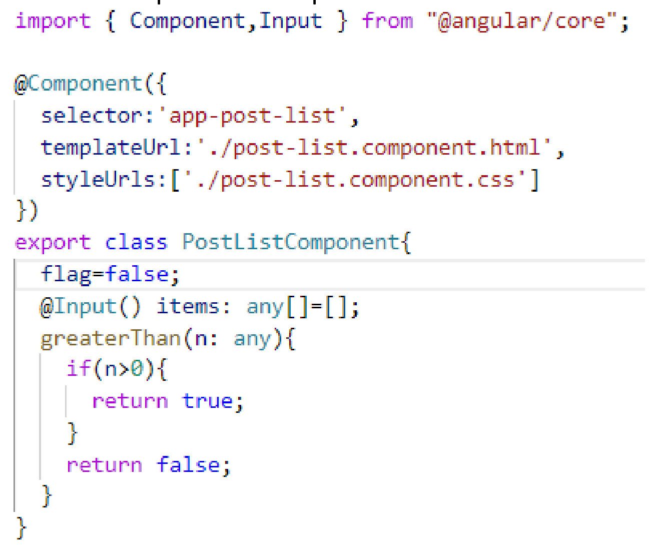
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'mean-course';
  //posts=[];
  posts:any[]=[];
  onPostAdded(post: any){
    this.posts.push(post);
  }
}
```

- This file is app.component.ts
- The variable that is declared is global to every component under it. Which can be retrieved through property binding.
- Here we can see, we declared an array "posts" of type any and we are collecting the data that is came through event binding by the post-create component here and store it in posts array. Which can be used globally.
- Now the data is ready , hence we must bind the property to the post-list to print the accordions
- And now in the above image of app.compoenent.html we can see the tag of app-post-list where we binded our items variable of post-list.component.ts to the posts variable of app.component.ts



```
<mat-accordion multi="false" *ngIf="greaterThan(items.length)
else nopost">
  <app-accordion-item *ngFor="let item of items" [item]="item">
  </app-accordion-item>
</mat-accordion>
<ng-template #nopost><p class="info-text mat-body-1">No Posts
are available</p></ng-template>
```

- This is post-list.component.html

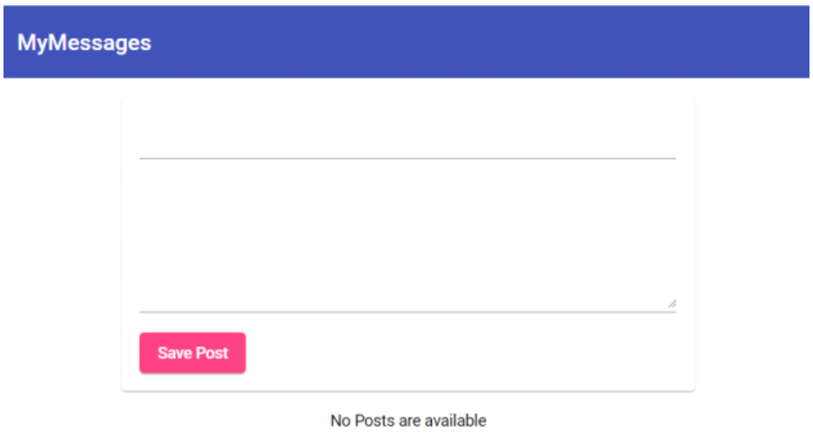


```
import { Component,Input } from "@angular/core";

@Component({
  selector:'app-post-list',
  templateUrl:'./post-list.component.html',
  styleUrls:['./post-list.component.css']
})
export class PostListComponent{
  flag=false;
  @Input() items: any[]=[];
  greaterThan(n: any){
    if(n>0){
      return true;
    }
    return false;
  }
}
```

- This is post-list.component.ts
- Here in post-list.component.ts , as we are retrieving the data from the app.component.(html or ts) to the post-list.component.ts. First we need to bind the property, and if we need to bind the property the sending and receiving variables must be of same types and to receive the data from outside the component we need to import input module as shown above.

The Output will be as following

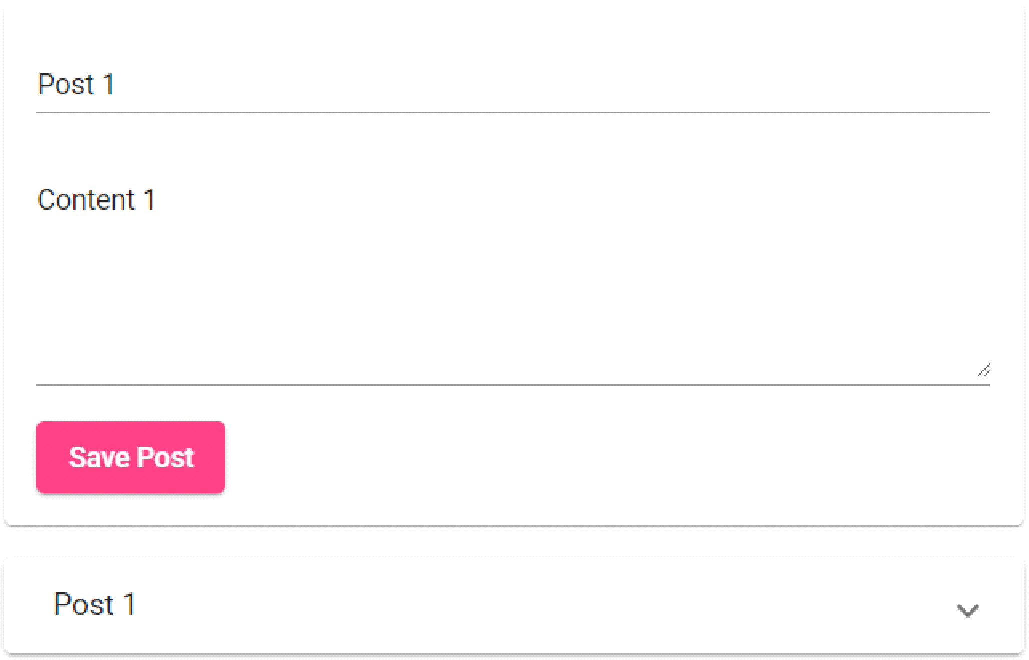
Intially :


MyMessages

Save Post

No Posts are available

- When we fill the title and content and save the post it changes as

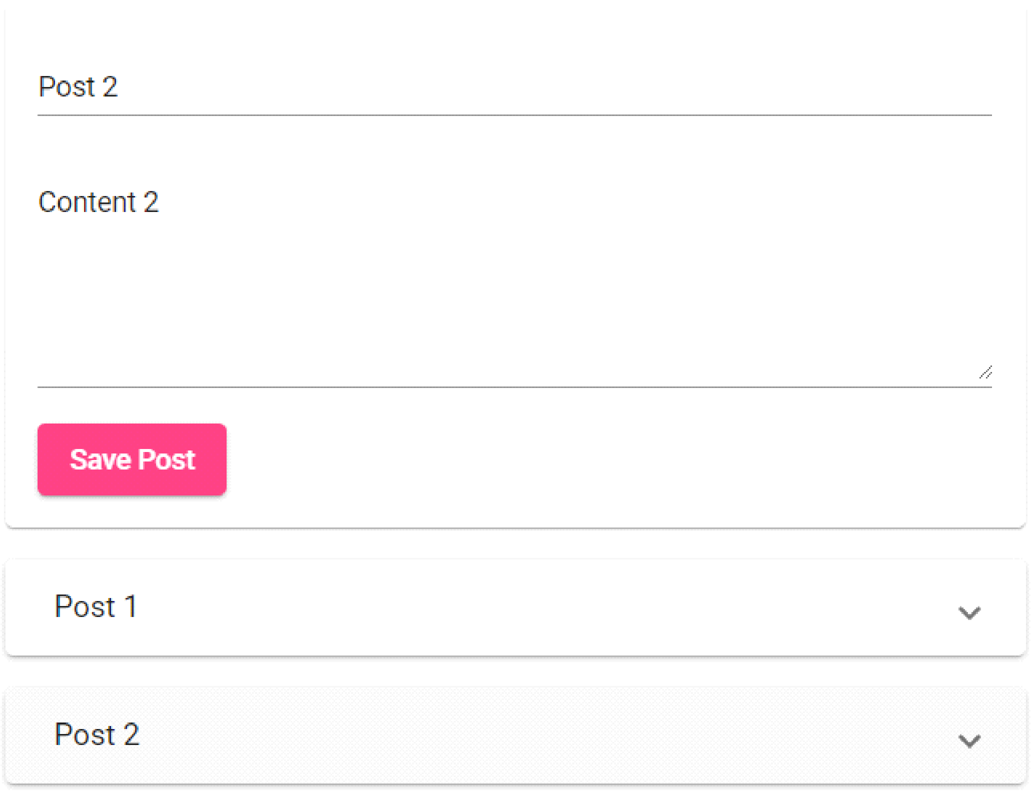

Post 1

Content 1

Save Post

Post 1

- And if we add another post it would be

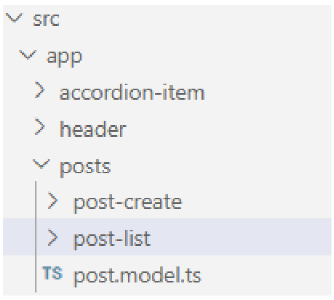


- In this way , as we add posts using this application the posts are auto generated. Dynamically in the page.

10. Post Model Interface

24 February 2022 00:06

Previously in chapter 9 we only utilized the data and pass the data without declaring the datatype or we have used "any". So to prevent attacks , we are creating our own datatype called Post and use it as datatype. This mainly used in logic creation and manipulation time mostly in typescript files. Let's change that from previous code.



```
src
  app
    accordion-item
    header
    posts
      post-create
      post-list
      post.model.ts
```

- Create post.model.ts and create a datatype using an interface so we can use it whenever we required in our project

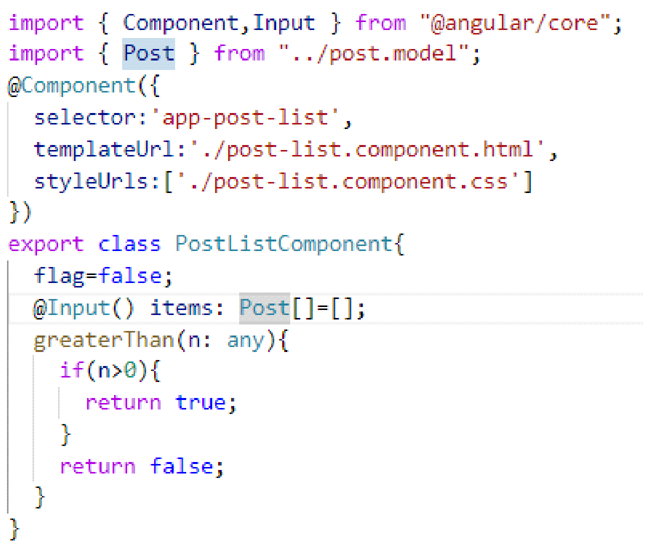
src > app > posts > TS post.model.ts > ...

```
export interface Post{
  title:string;
  content:string;
}
```

- This is post.model.ts
- Now we have defined our datatype and we have to use it in our code.
- The following changes needed in post-create.component.ts and post-list.component.ts and app.component.ts files

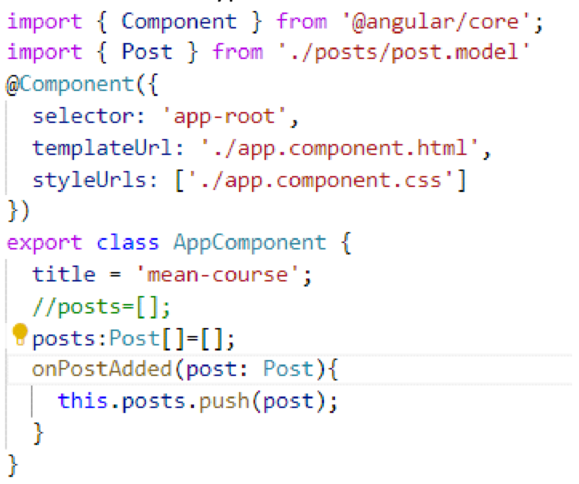
```
import { Component, EventEmitter,Output } from "@angular/core";
import { Post } from "../post.model";
@Component({
  selector:'app-post-create',
  templateUrl:'./post-create.component.html',
  styleUrls: ['./post-create.component.css']
})
export class PostCreateComponent{
  enteredContent="";
  enteredTitle="";
  @Output() postCreated=new EventEmitter<Post>();
  onAddPost(){
    const post:Post={
      title:this.enteredTitle,
      content:this.enteredContent
    };
    this.postCreated.emit(post);
  }
}
```

- This is post-create.component.ts where we are declaring emitter datatype as Post as it contain the post. Declare variable post datatype also as Post by using color beside the variable name
- Syntax=> Variable name: Datatype



```
import { Component,Input } from "@angular/core";
import { Post } from "../post.model";
@Component({
  selector:'app-post-list',
  templateUrl:'./post-list.component.html',
  styleUrls:['./post-list.component.css']
})
export class PostListComponent{
  flag=false;
  @Input() items: Post[]=[];
  greaterThan(n: any){
    if(n>0){
      return true;
    }
    return false;
  }
}
```

- This is post-list.component.ts
- Define datatype of the items variable and before using it don't forget to import that interface.



```
import { Component } from '@angular/core';
import { Post } from './posts/post.model'
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'mean-course';
  //posts=[];
  posts:Post[]=[];
  onPostAdded(post: Post){
    this.posts.push(post);
  }
}
```

- This is app.component.ts
- Declare the datatype of posts variable of app component

Output is same as previous chapter 9

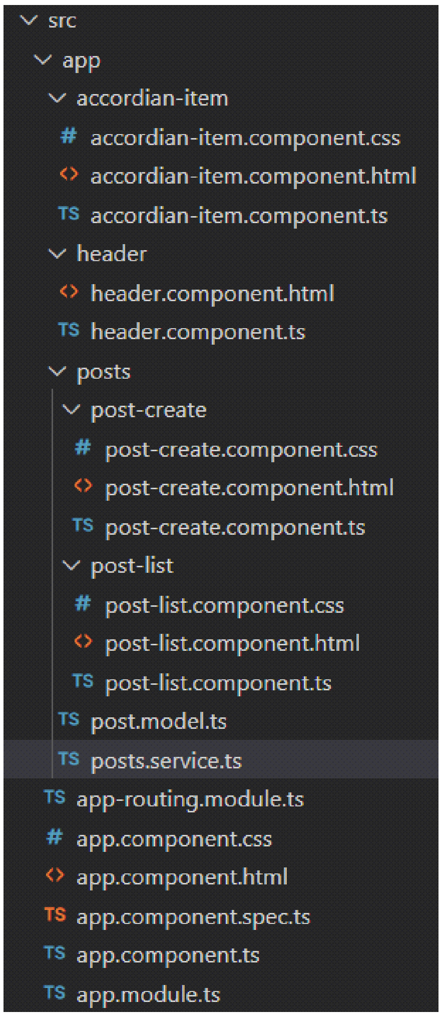
12. Service Injectable Method of Variable passing

14 April 2022 22:47

In general, If we want to establish a connection between two components where we need to access a variable using emitters, we need to establish a connection between them in /\ this shape.

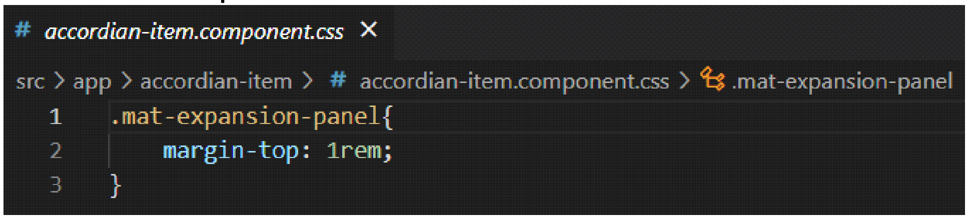
Suppose in current example we need to access a variable called posts from both the components, then we declare that variable in the common parent component of both. And then the component which sends data to the parent component sends using Output Event Emitter and if we want to access that variable from child component we read that variable using Input Event Emitter. In our example, posts-create sends data to app component using output emitter and post-list receives the data from app component using Input Event emitter.

So , to establish the direct connection between those post-create and post-list we use posts.service.ts instead of app component. Whole project is dumped below with the file names and folder structure.



```
src
  app
    accordian-item
      accordian-item.component.css
      accordian-item.component.html
      accordian-item.component.ts
    header
      header.component.html
      header.component.ts
    posts
      post-create
        post-create.component.css
        post-create.component.html
        post-create.component.ts
      post-list
        post-list.component.css
        post-list.component.html
        post-list.component.ts
      post.model.ts
      posts.service.ts
    app-routing.module.ts
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
```

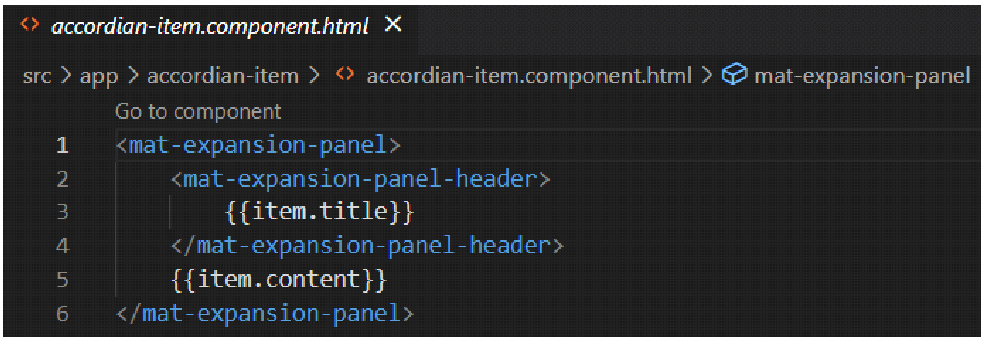
accordian-item.component.css



accordian-item.component.css
src > app > accordian-item > accordian-item.component.css > .mat-expansion-panel

```
.mat-expansion-panel{
    margin-top: 1rem;
}
```

accordian-item.component.html

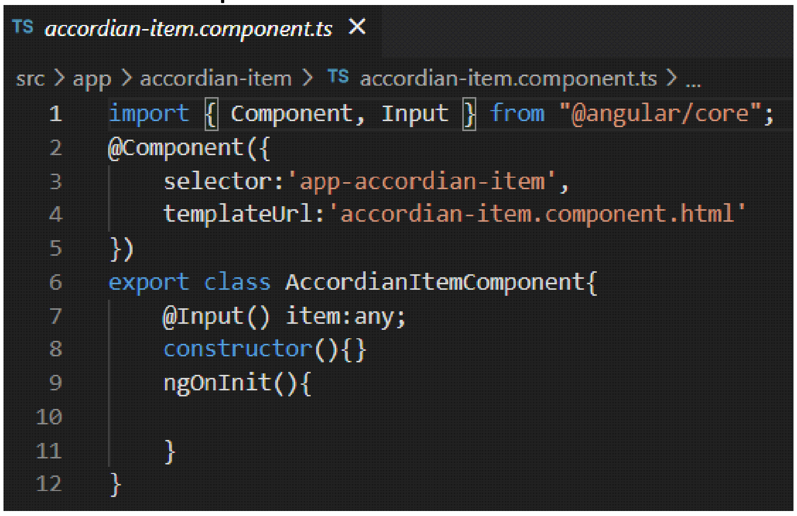

accordian-item.component.html ×

src > app > accordian-item > accordian-item.component.html > mat-expansion-panel

Go to component

```html
<mat-expansion-panel>
    <mat-expansion-panel-header>
        {{item.title}}
    </mat-expansion-panel-header>
    {{item.content}}
</mat-expansion-panel>
```

accordian-item.component.ts

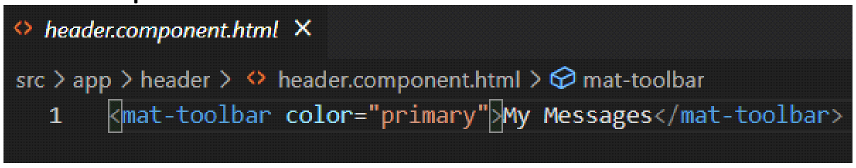

accordian-item.component.ts ×

src > app > accordian-item > accordian-item.component.ts > ...

```typescript
import { Component, Input } from "@angular/core";
@Component({
    selector:'app-accordian-item',
    templateUrl:'accordian-item.component.html'
})
export class AccordianItemComponent{
    @Input() item:any;
    constructor(){}
    ngOnInit(){

    }
}
```

In above accordian files we use accodian to print the UI output, we get the input from post-list.component and hence we established the Input Event Emmitter. Each Item of post-list is iterated over accordian to print the UI

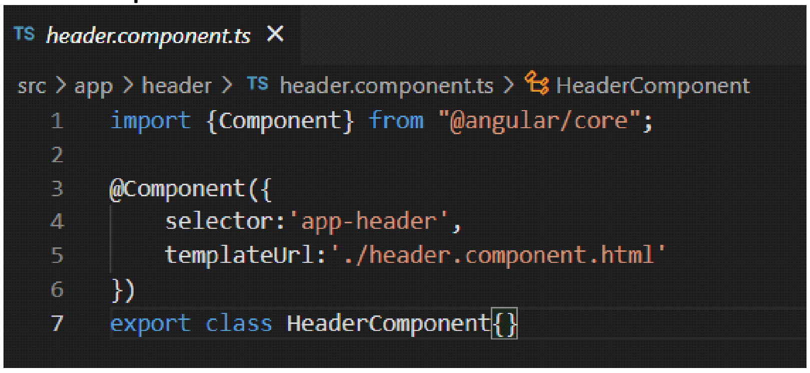
header.component.html


header.component.html ×

src > app > header > header.component.html > mat-toolbar

```html
<mat-toolbar color="primary">My Messages</mat-toolbar>
```

header.component.ts

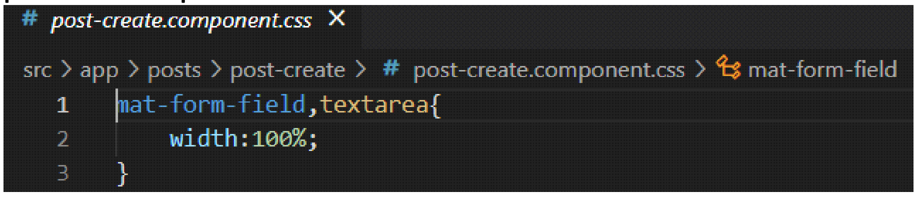

header.component.ts ×

src > app > header > header.component.ts > HeaderComponent

```typescript
import {Component} from "@angular/core";

@Component({
    selector:'app-header',
    templateUrl:'./header.component.html'
})
export class HeaderComponent{}
```

Header component is just to print the header of the application, we used bootstrap and angular material in this work.

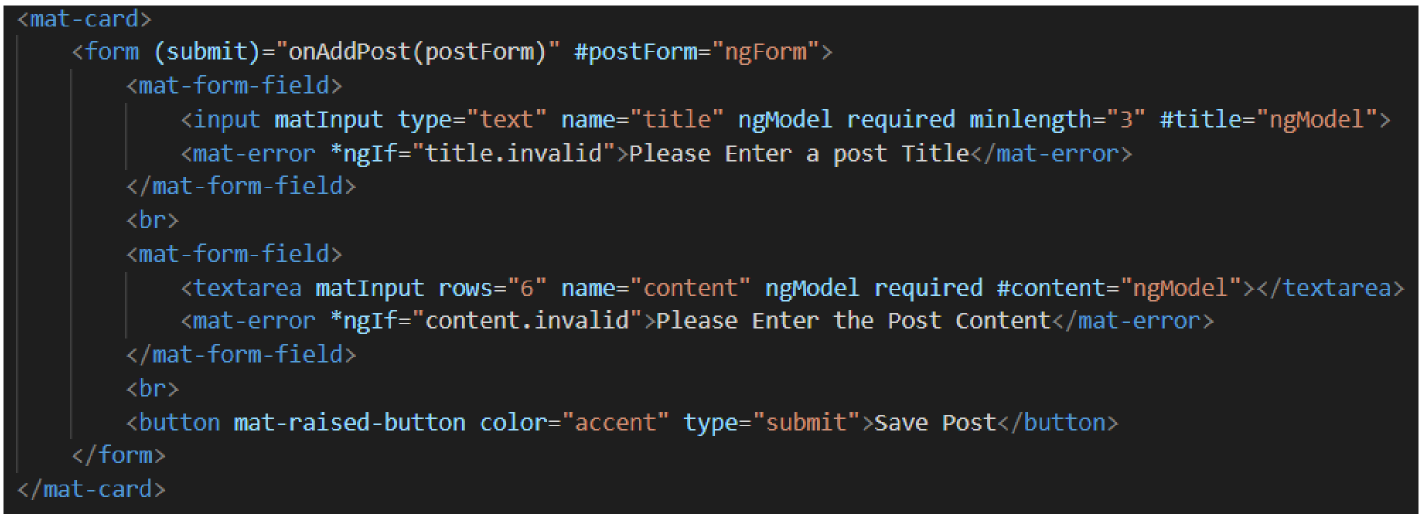
post-create.component.css


post-create.component.css ×

src > app > posts > post-create > post-create.component.css > mat-form-field

```css
mat-form-field,textarea{
    width:100%;
}
```

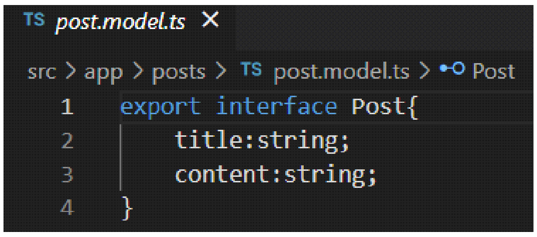
post-create.component.html



```html
<mat-card>
    <form (submit)="onAddPost(postForm)" #postForm="ngForm">
        <mat-form-field>
            <input matInput type="text" name="title" ngModel required minlength="3" #title="ngModel">
            <mat-error *ngIf="title.invalid">Please Enter a post Title</mat-error>
        </mat-form-field>
        <br>
        <mat-form-field>
            <textarea matInput rows="6" name="content" ngModel required #content="ngModel"></textarea>
            <mat-error *ngIf="content.invalid">Please Enter the Post Content</mat-error>
        </mat-form-field>
        <br>
        <button mat-raised-button color="accent" type="submit">Save Post</button>
    </form>
</mat-card>
```

- **This pattern of making form is preferable**
- **This form is referred as PostForm, the name is user defined. And similarly the input elements are also referred with their user defined names for ease of accessing. And used ngIf statements.**
- **On submitting the form it is redirected to onAddPost method of post-create typescript file.**
- **If the requirements that we mentioned in the input fields are violated then the mat-error will be called and error displayed. And halting the submissions was taken care under onAddPost of post-create typescript file.**

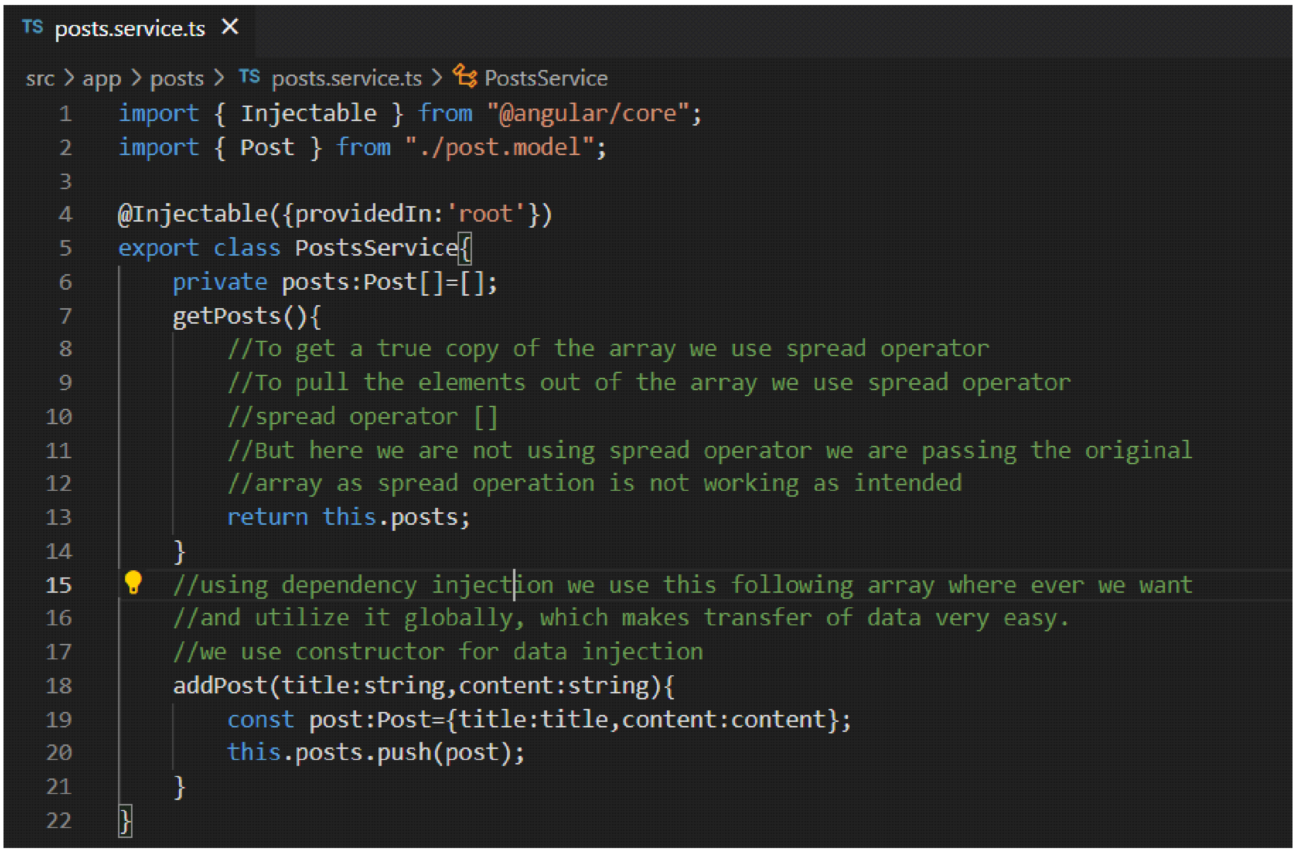
Post.model.ts



```typescript
export interface Post{
    title:string;
    content:string;
}
```

Here above we are creating our own datatype to refer inside the project.

Posts.service.ts



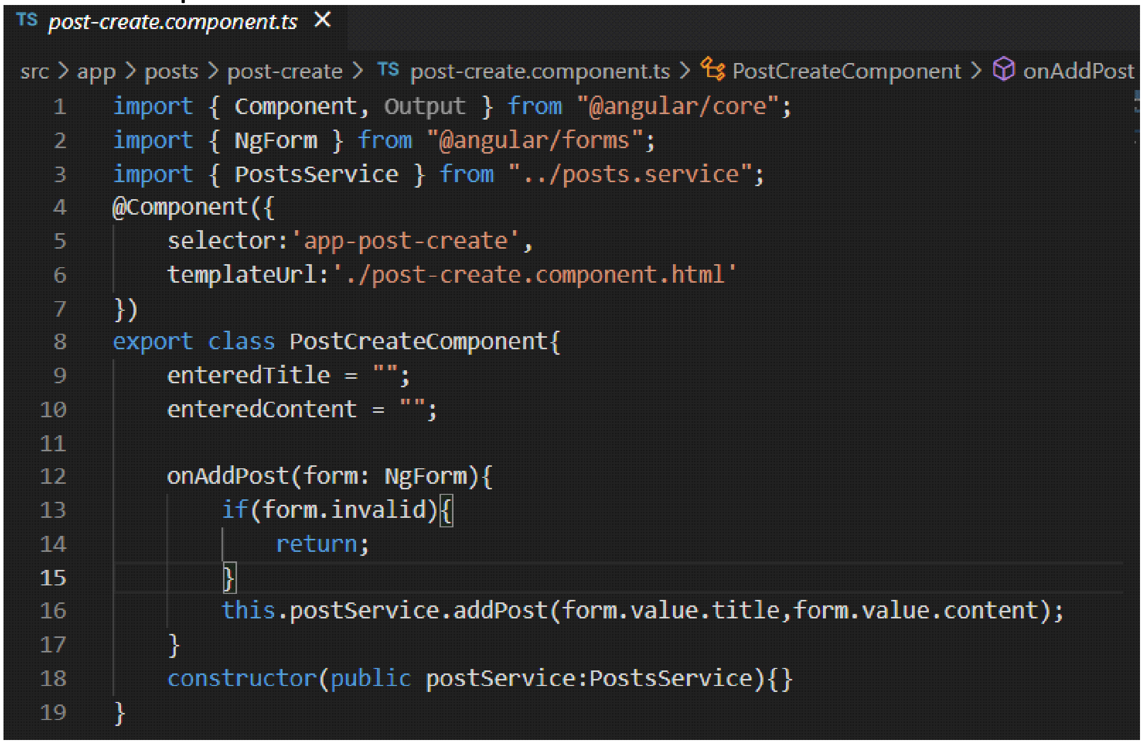
```typescript
import { Injectable } from "@angular/core";
import { Post } from "./post.model";

@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    getPosts(){
        //To get a true copy of the array we use spread operator
        //To pull the elements out of the array we use spread operator
        //spread operator []
        //But here we are not using spread operator we are passing the original
        //array as spread operation is not working as intended
        return this.posts;
    }
    //using dependency injection we use this following array where ever we want
    //and utilize it globally, which makes transfer of data very easy.
    //we use constructor for data injection
    addPost(title:string,content:string){
        const post:Post={title:title,content:content};
        this.posts.push(post);
    }
}
```

We are making this class as injectable , so we can use the variable posts and getPosts() function in post-list and addPost under post-create. If it is not injectable we could not do that

process.

We utilize the following variables and functions where ever we want by declaring postservice under class constructor of that component.

Post-create.component.ts



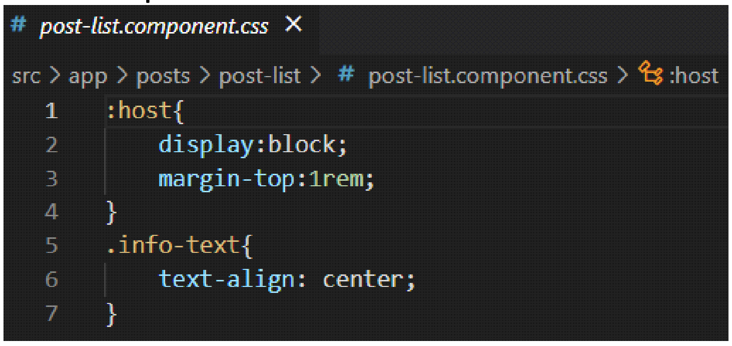
TS post-create.component.ts ×

src > app > posts > post-create > TS post-create.component.ts > PostCreateComponent > onAddPost

```
import { Component, Output } from "@angular/core";
import { NgForm } from "@angular/forms";
import { PostsService } from "../posts.service";
@Component({
    selector:'app-post-create',
    templateUrl:'./post-create.component.html'
})
export class PostCreateComponent{
    enteredTitle = "";
    enteredContent = "";

    onAddPost(form: NgForm){
        if(form.invalid){
            return;
        }
        this.postService.addPost(form.value.title,form.value.content);
    }
    constructor(public postService:PostsService){}
}
```

We are declaring the post service under the constructor and utilizing the addPost functionality and accessing the variable, which helps us in synchronization of print list using post-list.

Post-list.component.css

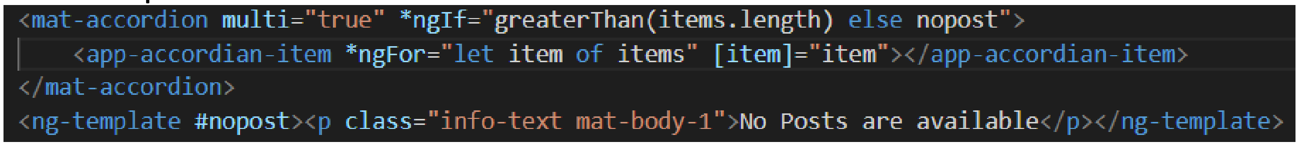


post-list.component.css ×

src > app > posts > post-list > # post-list.component.css > :host

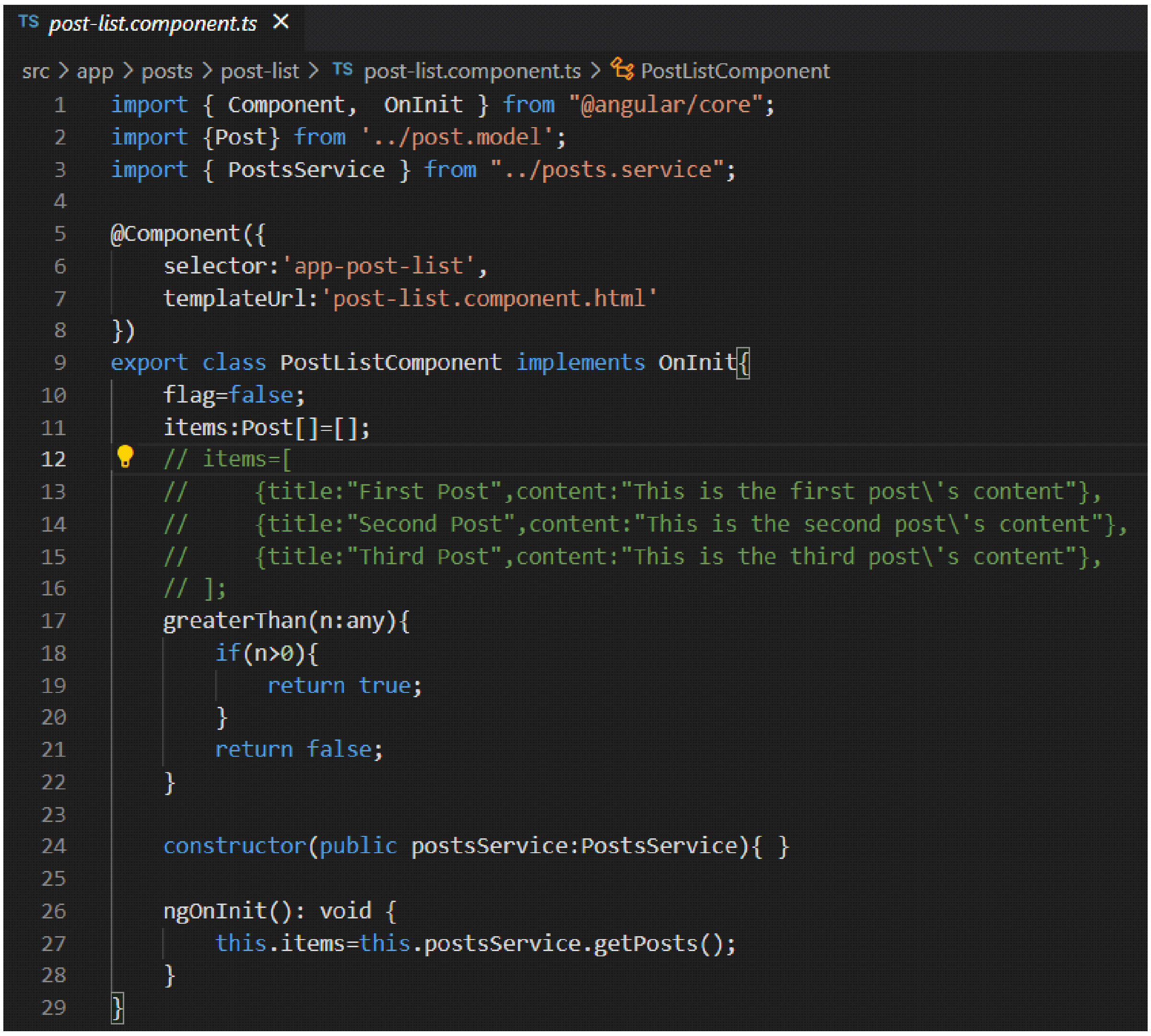
```
:host{
    display:block;
    margin-top:1rem;
}
.info-text{
    text-align: center;
}
```

Post-list.component.html



```
<mat-accordion multi="true" *ngIf="greaterThan(items.length) else nopost">
    <app-accordian-item *ngFor="let item of items" [item]="item"></app-accordian-item>
</mat-accordion>
<ng-template #nopost><p class="info-text mat-body-1">No Posts are available</p></ng-template>
```

Post-list.component.ts

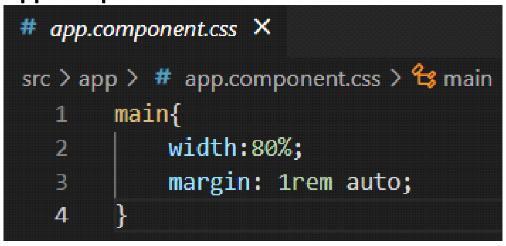

TS post-list.component.ts ×

src > app > posts > post-list > TS post-list.component.ts > PostListComponent

```
import { Component,  OnInit } from "@angular/core";
import {Post} from '../post.model';
import { PostsService } from "../posts.service";

@Component({
    selector:'app-post-list',
    templateUrl:'post-list.component.html'
})
export class PostListComponent implements OnInit{
    flag=false;
    items:Post[]=[];
    // items=[
    //     {title:"First Post",content:"This is the first post\'s content"},
    //     {title:"Second Post",content:"This is the second post\'s content"},
    //     {title:"Third Post",content:"This is the third post\'s content"},
    // ];
    greaterThan(n:any){
        if(n>0){
            return true;
        }
        return false;
    }

    constructor(public postsService:PostsService){ }

    ngOnInit(): void {
        this.items=this.postsService.getPosts();
    }
}
```

We are initializing the postService and using that we are retrieving all the posts through getPosts and storing it into the items and sending it item by item to accordian through databinding which is declared in post-list.component.html. Here we are executing getPosts when we initialize the component.

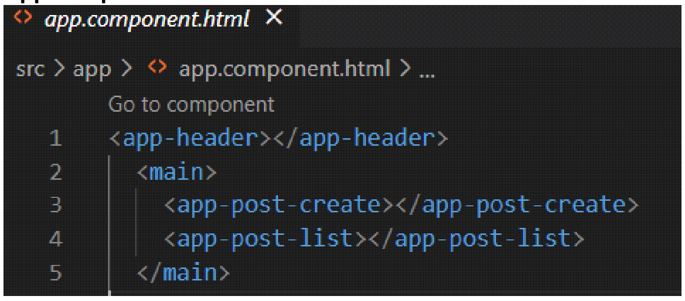
App.component.css


app.component.css ×

src > app > # app.component.css > main

```
main{
    width:80%;
    margin: 1rem auto;
}
```

App.component.html


app.component.html ×

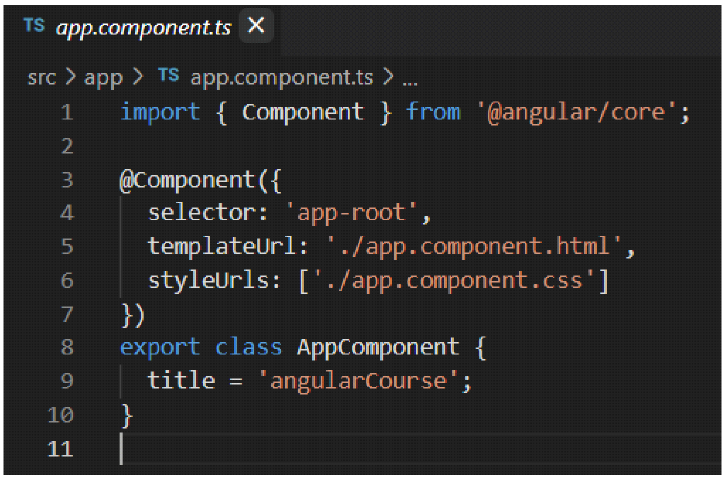
src > app > app.component.html > ...

Go to component

```
<app-header></app-header>
 <main>
   <app-post-create></app-post-create>
   <app-post-list></app-post-list>
 </main>
```

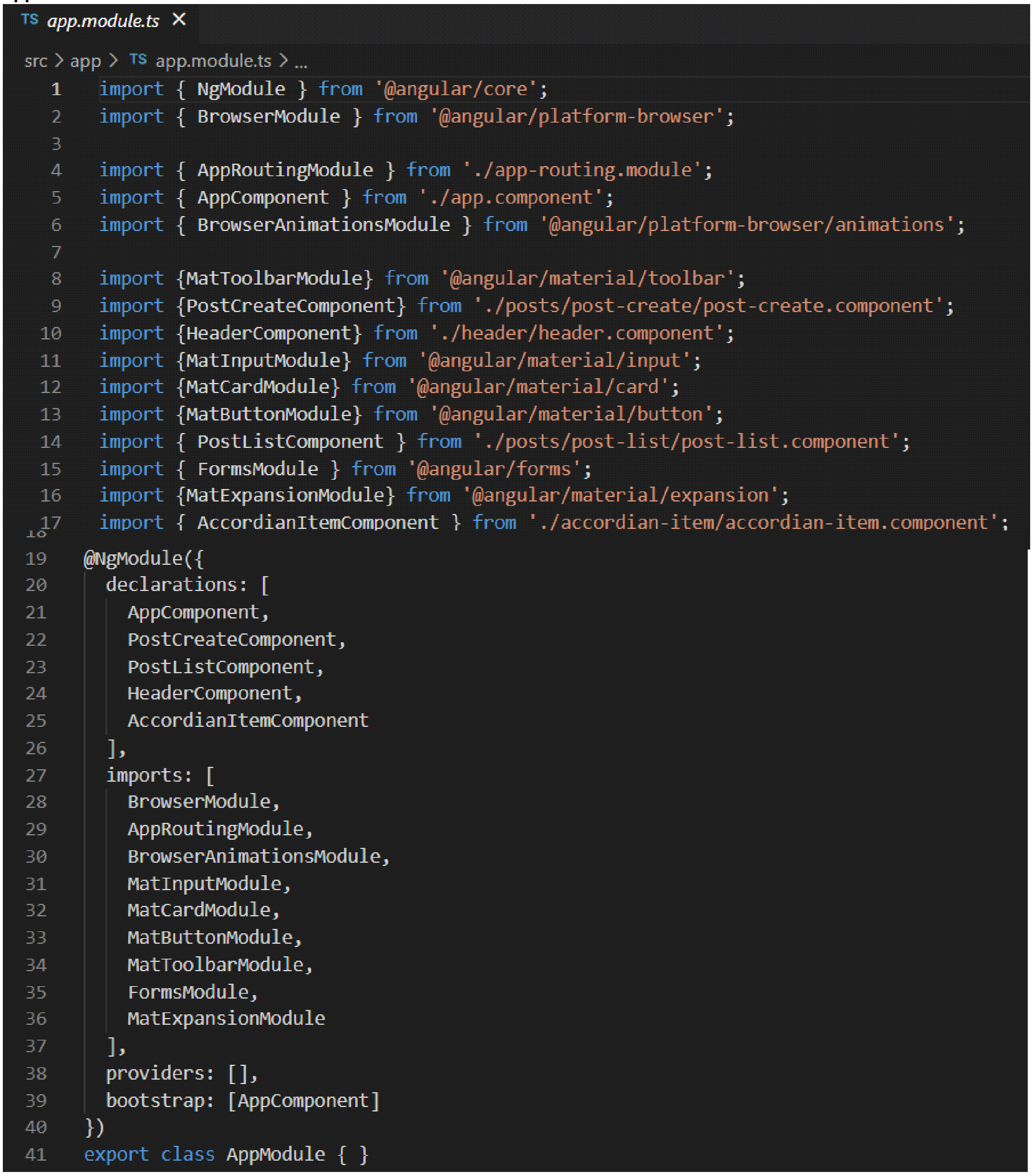
Once verify the image, here we removed all the data binding which is happened with the app component as now communication is happening through posts.service.ts .

app.component.ts



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

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

```

app.module.ts



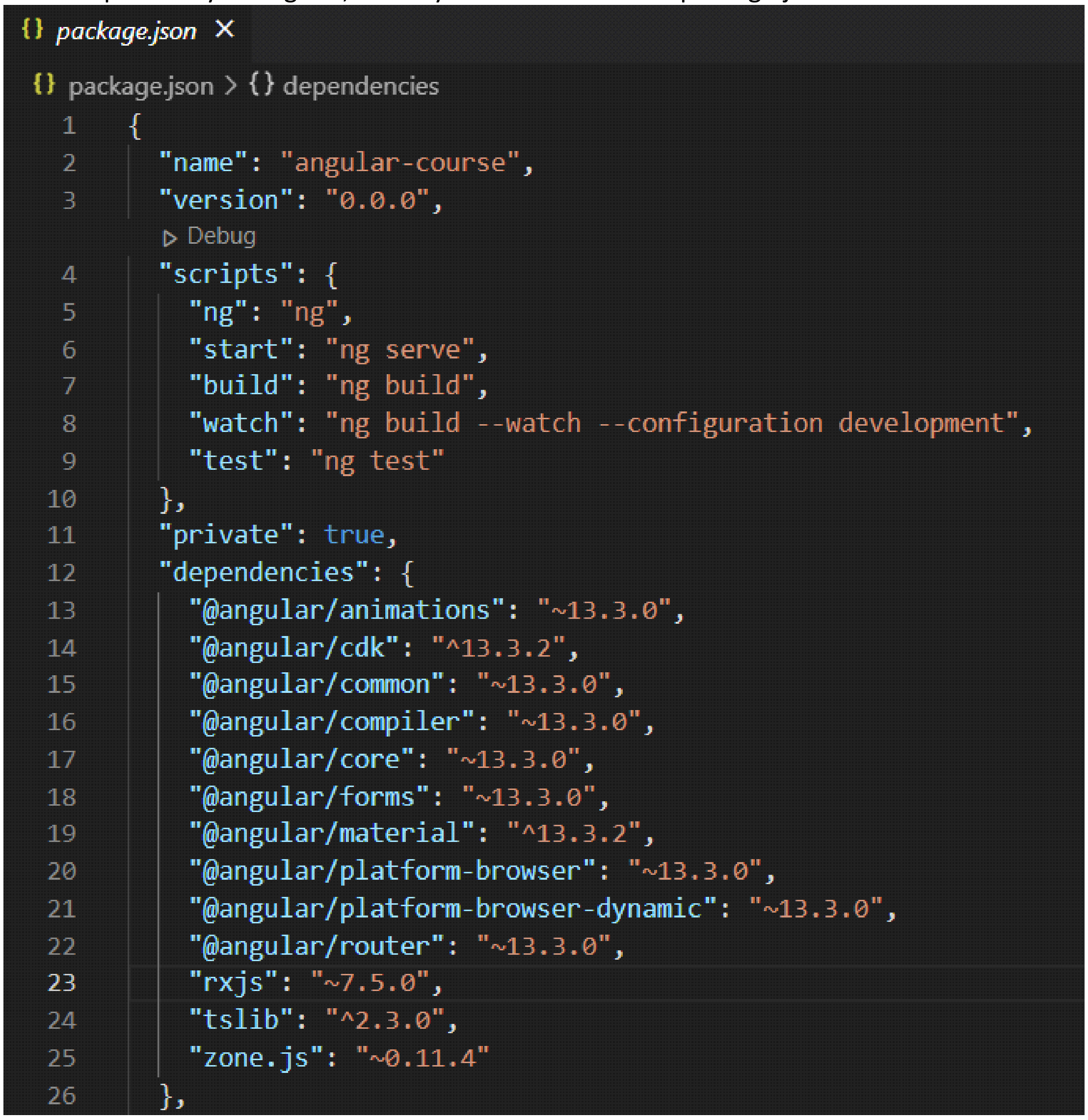
```typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import {MatToolbarModule} from '@angular/material/toolbar';
import {PostCreateComponent} from './posts/post-create/post-create.component';
import {HeaderComponent} from './header/header.component';
import {MatInputModule} from '@angular/material/input';
import {MatCardModule} from '@angular/material/card';
import {MatButtonModule} from '@angular/material/button';
import { PostListComponent } from './posts/post-list/post-list.component';
import { FormsModule } from '@angular/forms';
import {MatExpansionModule} from '@angular/material/expansion';
import { AccordianItemComponent } from './accordian-item/accordian-item.component';

@NgModule({
  declarations: [
    AppComponent,
    PostCreateComponent,
    PostListComponent,
    HeaderComponent,
    AccordianItemComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    MatInputModule,
    MatCardModule,
    MatButtonModule,
    MatToolbarModule,
    FormsModule,
    MatExpansionModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

If you want to use angular material or bootstrap inside the angular, you must execute the command whenever you initialize your project. And after installing you must restart your angular server localhost:4200

13. rxjs Observables

15 April 2022 12:39

- rxjs is also used for the input output operations that we done upto now, it is a core dependency of angular, which you can check under package.json



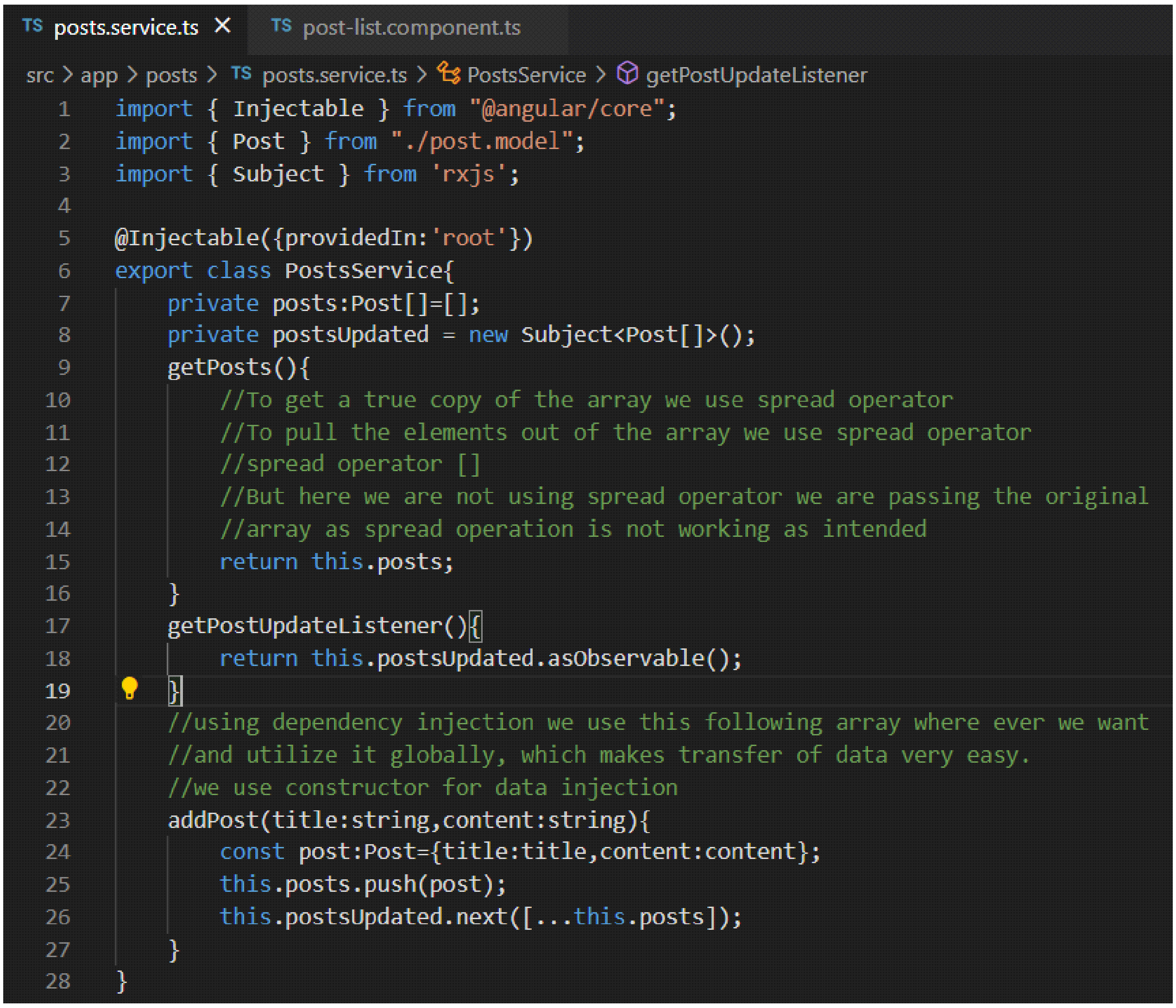
package.json ×

package.json > {} dependencies

```json
{
  "name": "angular-course",
  "version": "0.0.0",
  ▷ Debug
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~13.3.0",
    "@angular/cdk": "^13.3.2",
    "@angular/common": "~13.3.0",
    "@angular/compiler": "~13.3.0",
    "@angular/core": "~13.3.0",
    "@angular/forms": "~13.3.0",
    "@angular/material": "^13.3.2",
    "@angular/platform-browser": "~13.3.0",
    "@angular/platform-browser-dynamic": "~13.3.0",
    "@angular/router": "~13.3.0",
    "rxjs": "~7.5.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.11.4"
  },
```

- This provides us the objects that helps us to pass the data around components through subscription and dilute the subscription whenever component got destory and prevent memory leak (if the component diluted and component's variable is still occupying and having the memory it is memory leak, now using this subscription whenever component got destroyed the variable value's got diluted in the destroyed component and keep it empty).
- All the files are same as previous , only posts.service.ts and post-list.component.ts got changed.

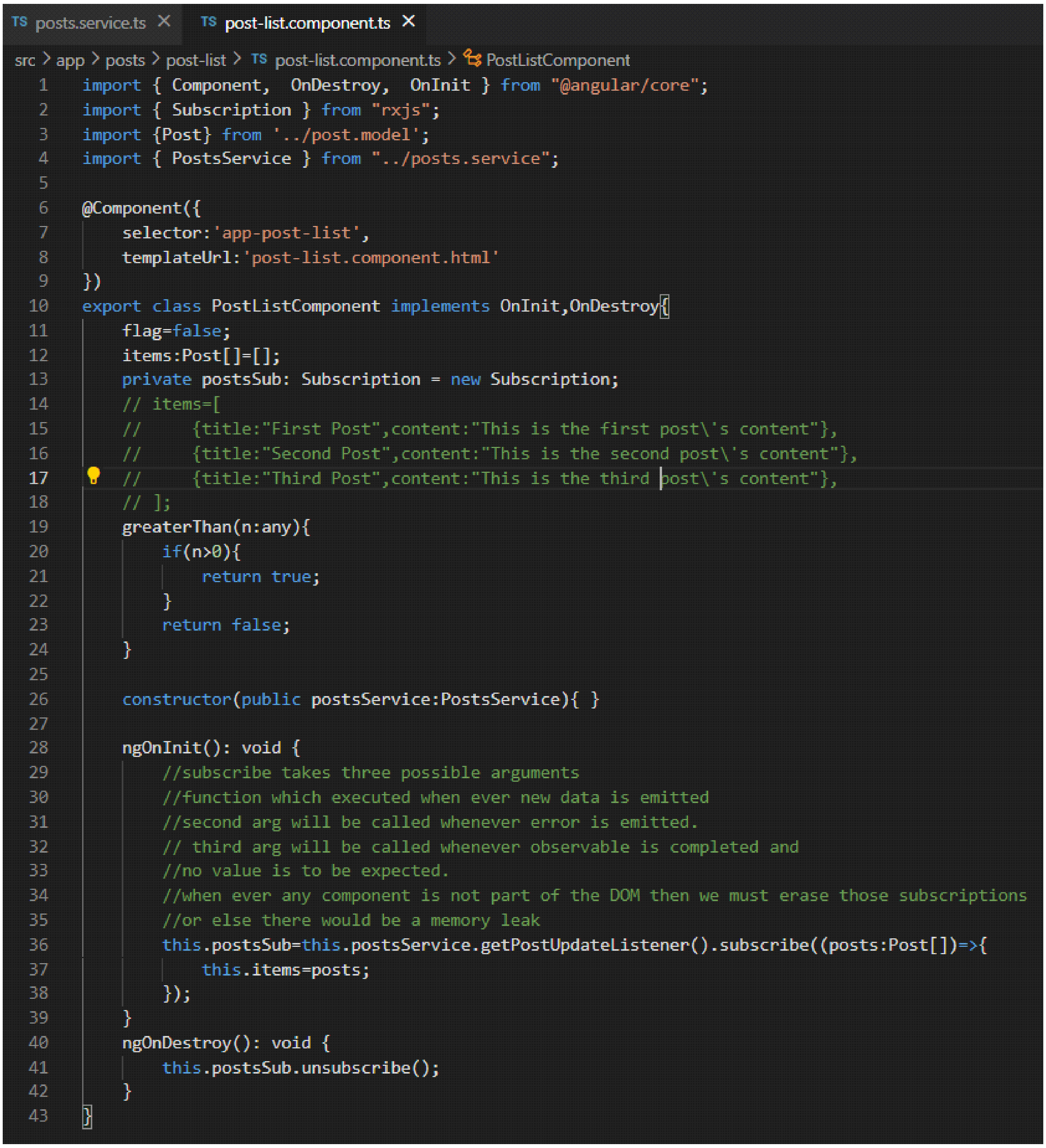
posts.service.ts



TS posts.service.ts ×
TS post-list.component.ts
src > app > posts > TS posts.service.ts > PostsService > getPostUpdateListener

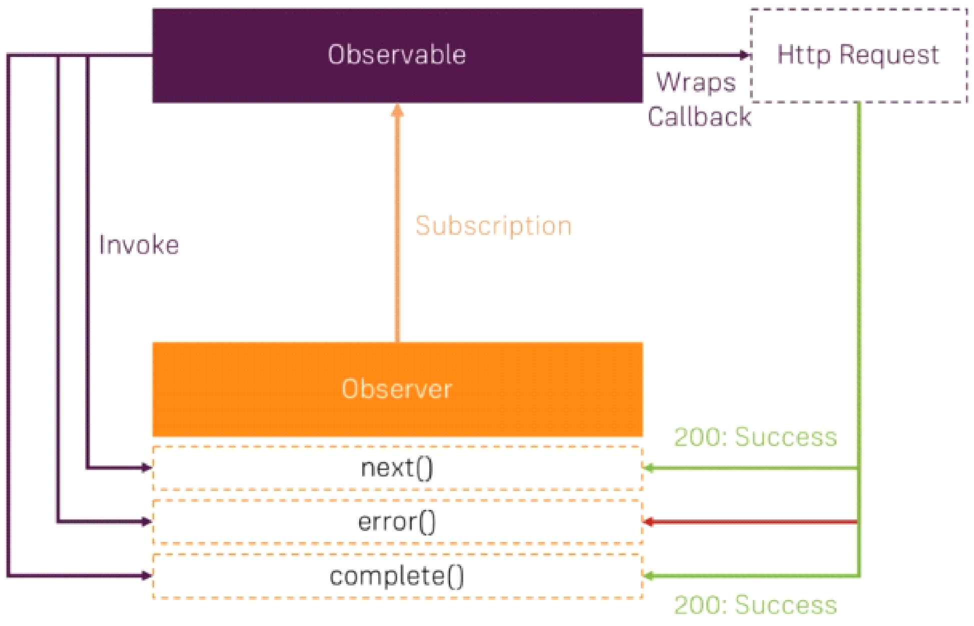
```
import { Injectable } from "@angular/core";
import { Post } from "./post.model";
import { Subject } from 'rxjs';

@Injectable({providedIn:'root'})
export class PostsService{
    private posts:Post[]=[];
    private postsUpdated = new Subject<Post[]>();
    getPosts(){
        //To get a true copy of the array we use spread operator
        //To pull the elements out of the array we use spread operator
        //spread operator []
        //But here we are not using spread operator we are passing the original
        //array as spread operation is not working as intended
        return this.posts;
    }
    getPostUpdateListener(){
        return this.postsUpdated.asObservable();
    }
    //using dependency injection we use this following array where ever we want
    //and utilize it globally, which makes transfer of data very easy.
    //we use constructor for data injection
    addPost(title:string,content:string){
        const post:Post={title:title,content:content};
        this.posts.push(post);
        this.postsUpdated.next([...this.posts]);
    }
}
```

Post-list.component.html

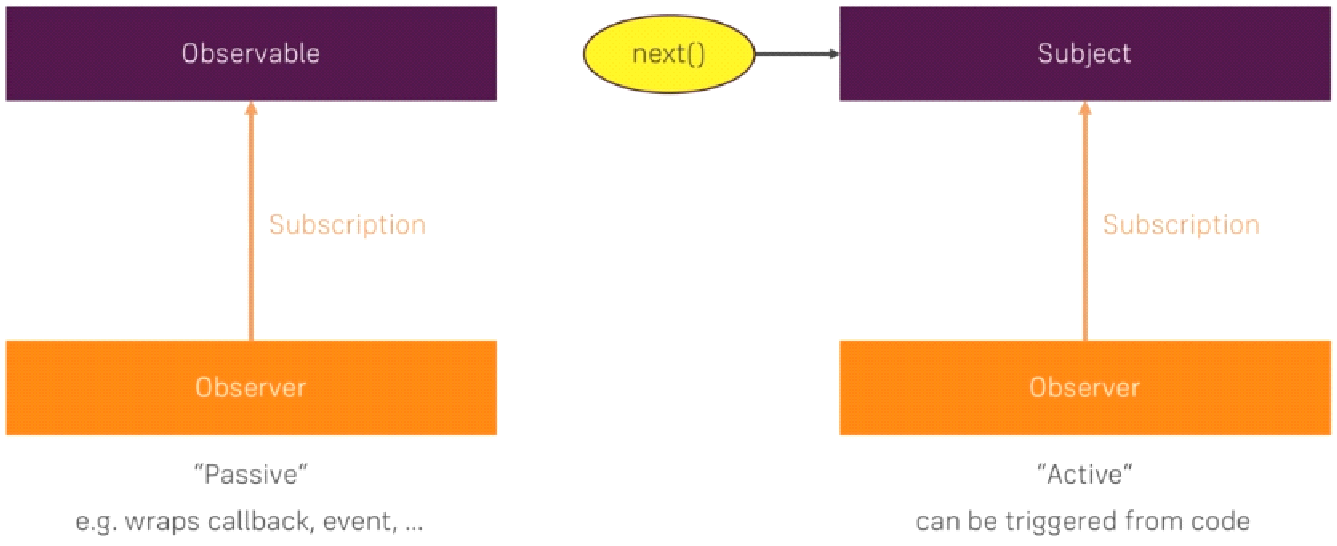


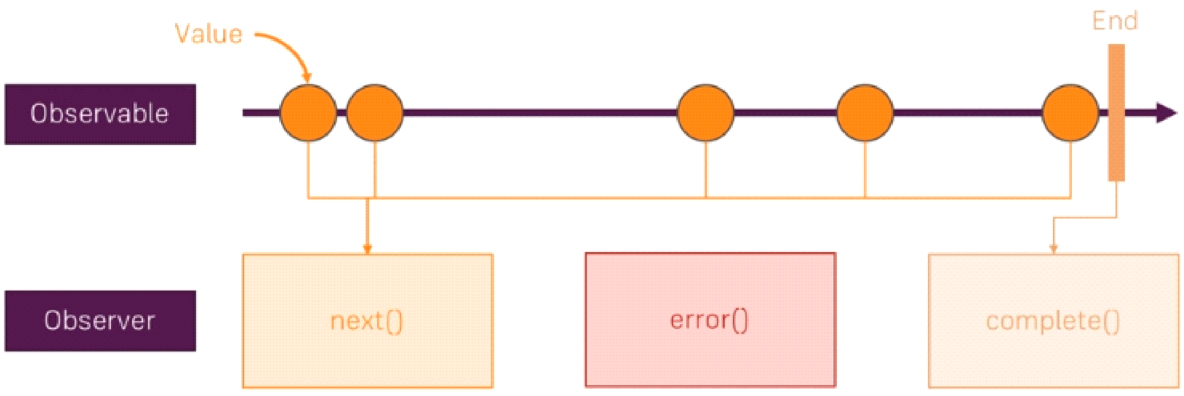
TS posts.service.ts ×
TS post-list.component.ts ×
src > app > posts > post-list > TS post-list.component.ts > PostListComponent

```typescript
import { Component,  OnDestroy,  OnInit } from "@angular/core";
import { Subscription } from "rxjs";
import {Post} from '../post.model';
import { PostsService } from "../posts.service";

@Component({
    selector:'app-post-list',
    templateUrl:'post-list.component.html'
})
export class PostListComponent implements OnInit,OnDestroy{
    flag=false;
    items:Post[]=[];
    private postsSub: Subscription = new Subscription;
    // items=[
    //     {title:"First Post",content:"This is the first post\'s content"},
    //     {title:"Second Post",content:"This is the second post\'s content"},
    //     {title:"Third Post",content:"This is the third post\'s content"},
    // ];
    greaterThan(n:any){
        if(n>0){
            return true;
        }
        return false;
    }

    constructor(public postsService:PostsService){ }

    ngOnInit(): void {
        //subscribe takes three possible arguments
        //function which executed when ever new data is emitted
        //second arg will be called whenever error is emitted.
        // third arg will be called whenever observable is completed and
        //no value is to be expected.
        //when ever any component is not part of the DOM then we must erase those subscriptions
        //or else there would be a memory leak
        this.postsSub=this.postsService.getPostUpdateListener().subscribe((posts:Post[])=>{
            this.items=posts;
        });
    }
    ngOnDestroy(): void {
        this.postsSub.unsubscribe();
    }
}
```



- This is the main picture , regarding what we have done until now. In our sense, post-list component is the observer which subscribes the Observable called posts.service.ts through subscription variable called postsSub. Here in our case the observable is Subject in the posts.service.ts. There the subject observable sends the data required by the observer (post-list.component.ts) through next function. And the memory location of the data is declared inside the post-list.component.ts under subscribe call.
- Post-list.component.ts asks the data changes to the posts.service.ts , which acts as a service between post-create and post-list. If there are any changes the data is passed using next function and store it in the variable which is declared inside the post-list.
- In our process, we did not used Http Request.
- Subject is a type of Observable




Value
End
Observable
Observer
next()
error()
complete()

14. Complete Code of Angular Front End

15 April 2022 13:55

The project files of angular are in this link (https://drive.google.com/file/d/1rfw3bXCGocRj2cKtgFXtZd8r-wV3GvGp/view?usp=sharing)

We are into the backend part , we process with the node JS and express JS.

11. Displaying error for invalid fields

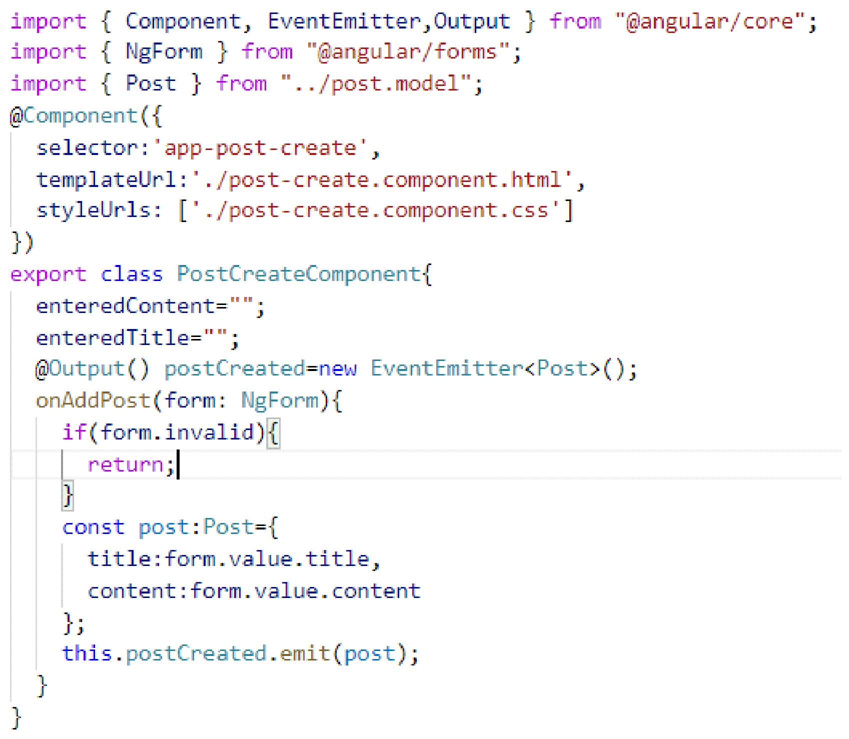
28 February 2022 22:04

Up to now , we are having two fields where even if we not enter any matter in the fields and click enter we get a post with empty title and empty content. But now what we want to do is, if we not enter anything in either of the fields then we display the corresponding error below that blank. And if the form is invalid then the onAddPost function must not execute which in turn prevent any post to create with empty title and content.

post-create.component.html

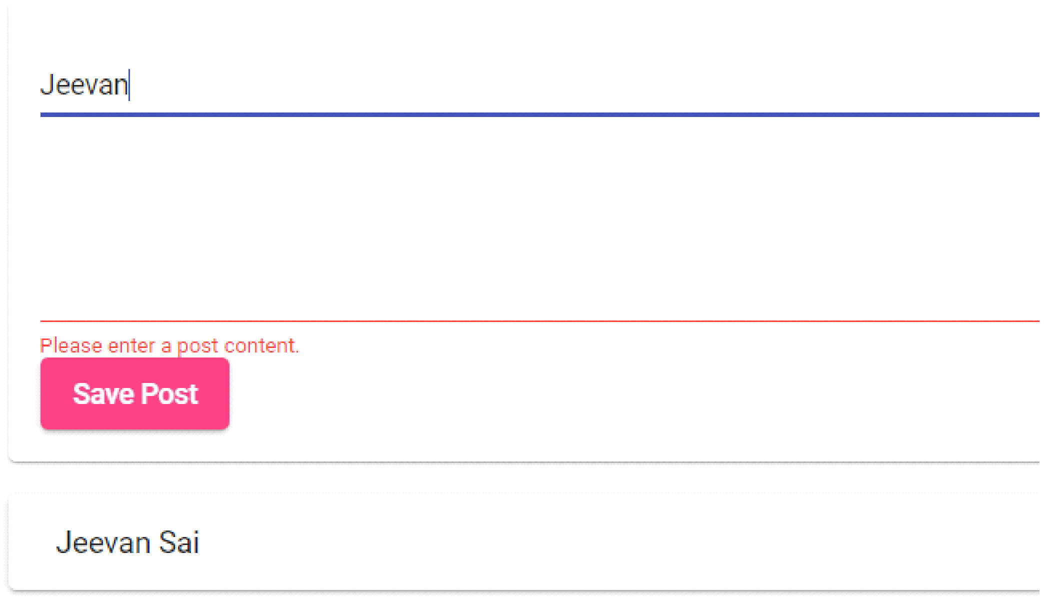
```
<mat-card>
  <form (submit)="onAddPost(postForm)" #postForm="ngForm">
    <mat-form-field>
      <!-- adding a directive without binding -->
      <input matInput type="text" name="title" ngModel required minlength="3"
      #title="ngModel">
      <mat-error *ngIf="title.invalid">Please enter a post title.</mat-error>
    </mat-form-field>
    <mat-form-field>
      <textarea matInput rows="4" name="content" ngModel required #content="ngModel"></
      textarea>
      <mat-error *ngIf="content.invalid">Please enter a post content.</mat-error>
    </mat-form-field>
    <button
    mat-raised-button
    color="accent"
    type="submit"
    >Save Post</button>
  </form>
</mat-card>
```

- Here we are having the form element so we need to give any separate action for save post button it will automatically trigger , submit attribute of the form tag and calls the onAddPost() function in post-create.component.ts where we are passing the form data by creating a local reference and storing ngForm in that.
- Here you can see the mat-error where we are checking the condition to print the error. We are retrieving the value of the input using the locally referenced name which is defined in that tag. We are doing the same for the content field also , and added some constraints for both the fields like required or min length or etc..



```
import { Component, EventEmitter,Output } from "@angular/core";
import { NgForm } from "@angular/forms";
import { Post } from "../post.model";
@Component({
  selector:'app-post-create',
  templateUrl:'./post-create.component.html',
  styleUrls: ['./post-create.component.css']
})
export class PostCreateComponent{
  enteredContent="";
  enteredTitle="";
  @Output() postCreated=new EventEmitter<Post>();
  onAddPost(form: NgForm){
    if(form.invalid){
      return;
    }
    const post:Post={
      title:form.value.title,
      content:form.value.content
    };
    this.postCreated.emit(post);
  }
}
```

- The onAddPost function is on the post-create.component.ts where it is having an form variable passing in it and accessing the variables of the form in it and creating post in it and emitting outside the component. See above where if form is invalid we are not creating any post and returning empty and if both fields are valid then we are creating a post.



Jeevan

Please enter a post content.

Save Post

Jeevan Sai

- This will be the output if we make any error in any of the field.

CIP

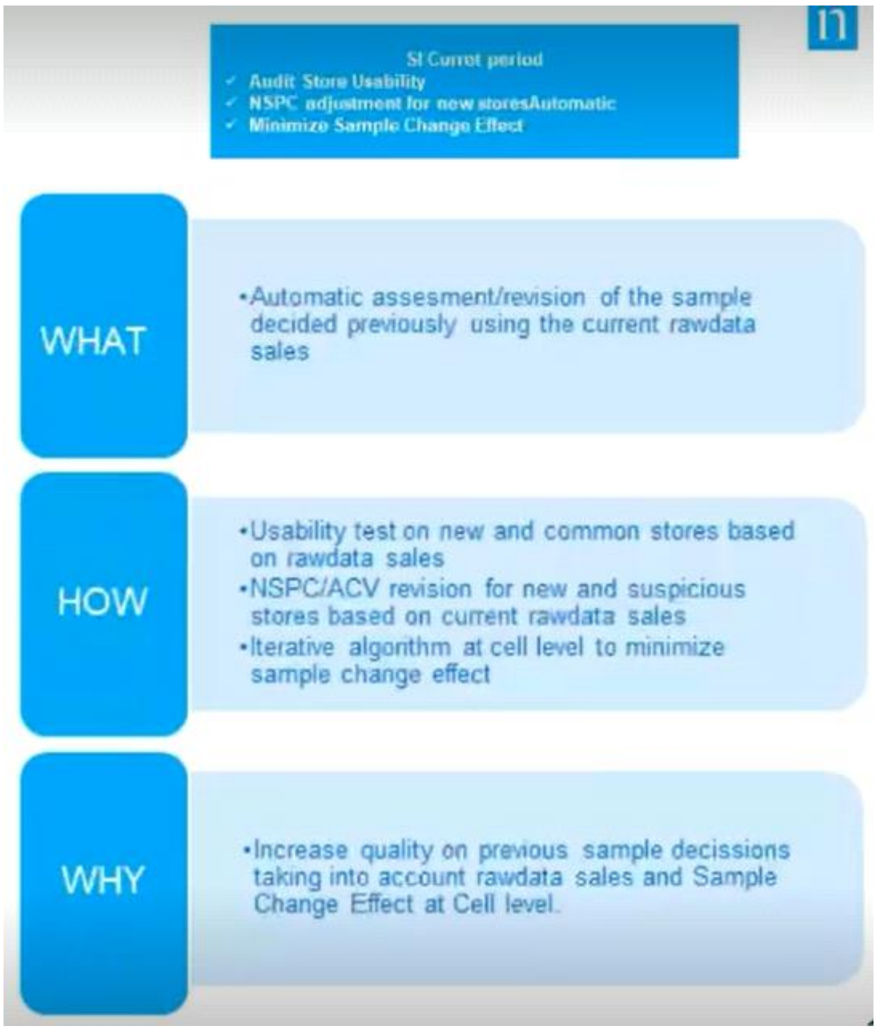
Wednesday, February 23, 2022 6:20 PM

MADRAS OVERVIEW (Market Definition For retail audit and scanning)

- What is channel (Global concept) : "general classification of shops" which can be shared across different countries. EX: shops , pharmacy
- Fun question:
- Which of the following is the example of channel in madras of CIP in the global concept?
  - NDTV
  - CNN
  - PHARMACY
  - INDIA TODAY
- But now channel in MADRAS - NRSP is different : it is related to "group of stores having similar characteristics" in terms of Homogeneity (type of shop) , data collecting methods (audit or scanning), time frame of collection (monthly, weekly)
- Sample is a sub part of the universe of data
- Industry - is used to find Control volume for the stores
- Shop - where retail goods sold
- Cell - group of similar shops defined by one or more stratification criteria
- Ratio's or post stratification - used to correct bias projection due to lack of or over sample
- Markets are defined geographic or demographic units for example market's include cities, regions, countries, shop types, shop sizes, type of market(traditional or supermarket) etc.
- Node of market hierarchy is market segment.
- Market data scope mean the data which will be produced in a group together by the elements of market hierarchy.

Sample Inspection

- The advantage of sample inspection is to effectively improve the quality. Because while making the sample we will not consider some data and we need to ensure that the data that we not consider is the data which is less important. And sample the data which provides good quality.
- If we not do this we may find data quality challenges from clients and costly re work must be done because we have re verify the sample to increase the accuracy and negative impact on the clients confidence on RMS service and they may cancel the contract.
- We can prevent this by following corrective measures, enabling and disabling stores, change store Control volume, change store cell allocation and decide on estimation of the stores
- SCC mean Sample Control change
- SI Current Period => Sample Inspection Current Period



SI Currot period

- ✓ Audit Store Usability
- ✓ NSPC adjustment for new storesAutomatic
- ✓ Minimize Sample Change Effect

WHAT

- Automatic assesment/revision of the sample decided previously using the current rawdata sales

HOW

- Usability test on new and common stores based on rawdata sales
- NSPC/ACV revision for new and suspicious stores based on current rawdata sales
- Iterative algorithm at cell level to minimize sample change effect

WHY

- Increase quality on previous sample decissions taking into account rawdata sales and Sample Change Effect at Cell level.

- Sample inspection data must be passed through sirval for sure for the validation.

Notebook

Monday, June 19, 2023 10:14 AM

Todo

1) ~~check all images~~

2) Azure learning

↳ Tomorrow

3) ~~chi-squared testing~~

Ci

X

4) Milkpacket

⑥ remove array in

build process

⑦ adapt a ~~us~~ with

apollo

Java Development

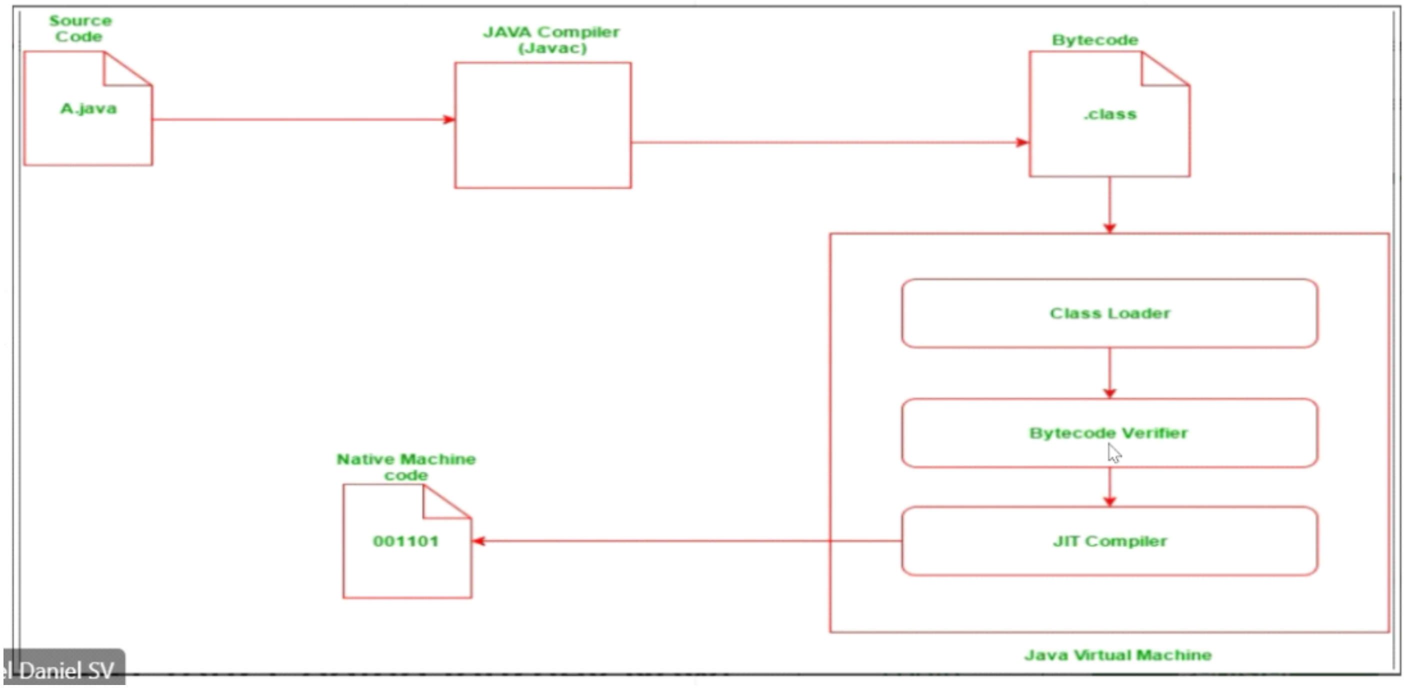
Monday, February 28, 2022 11:10 AM

Java is platform independent

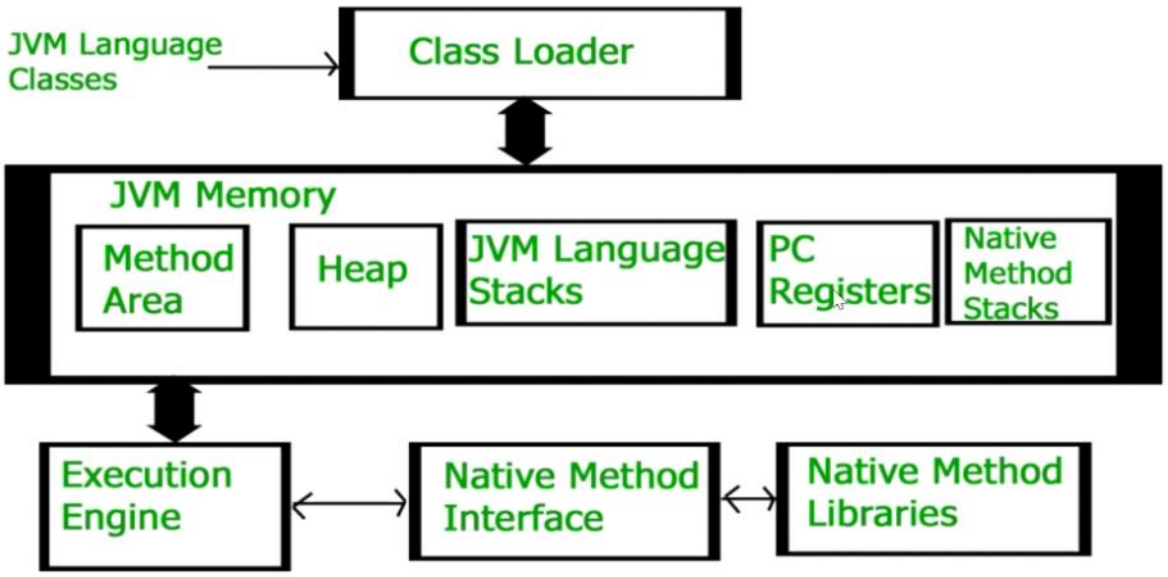
Java works by first compiling the source code into bytecode. Then, the bytecode can be compiled into machine code with the Java Virtual Machine (JVM). Java's bytecode can run on any device with the JVM which is why Java is known as a "*write once, run anywhere*" language.

JAVA is platform-independent language, the JVM is platform-dependent.

In java we generate a byte code it is platform independent (source code to bytecode)
Byte code is passed through JVM and here JVM is platform dependent



Class Loader loads the main class and byte code verifier verifies the byte code and JIT compiler compiles the



Native method stacks hold the native methods
To run a spring application we write a code in java
Springboot it'self have a inbuilt server (tomcat server)
File>New>Project>MavenProject

Maven => whenever we are creating a java project we need to have some dependencies, so if we are creating a maven project we can choose our dependencies manually that we need to install in our project automatically. Maven is also used to design applications, the jar or war file is created by it, to create an application.

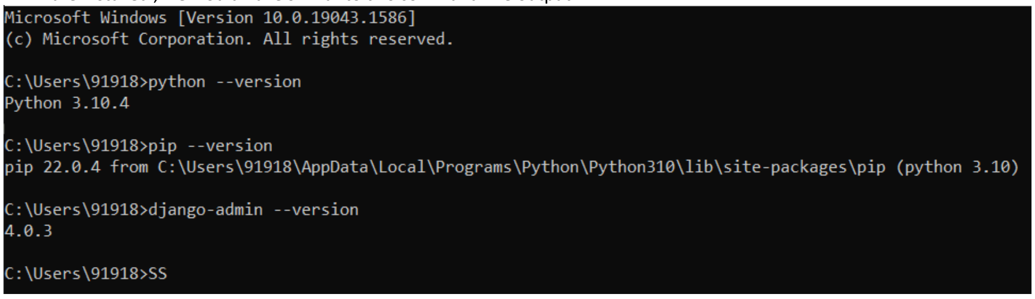
Maven Project

- Artificat Id

Django

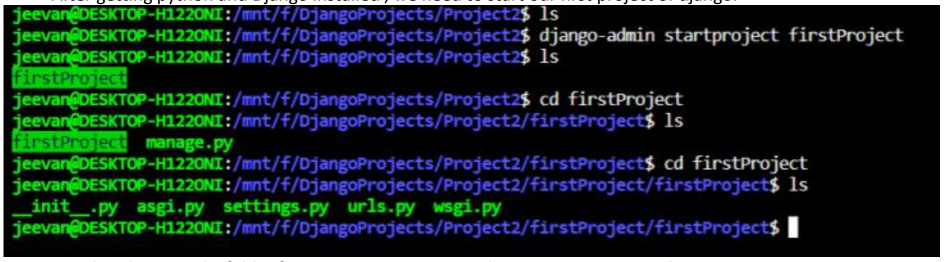
10 April 2022 17:17

- Framework is a combination of some components and packages, if we want to build any web application we need some components readymade because if we do it from scratch it is time consuming, so we utilize frameworks for this process. One of the web framework is Django.
- Generally we use HTML (for marking up), CSS(for designing) , JavaScript(For background logic) in our websites. We use HTML , CSS, JavaScript as the frontend part. At the backend part now a days we are using Django instead of servlets , JavaScript ,PHP, asp . Django is based on python. In java we are having model view container (MVC) , but in Django we are having model view template (MVT). Because Django is Template approach .
- Django helps us to build the application fast, and Django provides additional components like database connectivity automatically . Sometimes this may be the greater disadvantage because it can cause the overhead. Django ensures the security and scalability. Scalability in the sense it provides extension or broader space for the new users to access Django application.
- Django is a web framework for python.
- To check whether python , pip and Django are being installed or not on our system. If those are installed , we would have similar to this command line output.



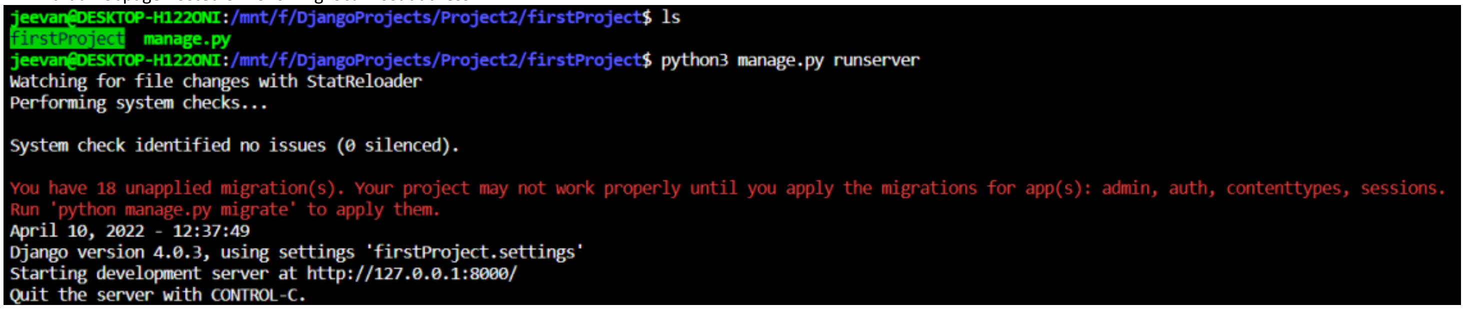
```
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\91918>python --version
Python 3.10.4

C:\Users\91918>pip --version
pip 22.0.4 from C:\Users\91918\AppData\Local\Programs\Python\Python310\lib\site-packages\pip (python 3.10)

C:\Users\91918>django-admin --version
4.0.3

C:\Users\91918>SS
```

- After getting python and Django installed , we need to start our first project of django.



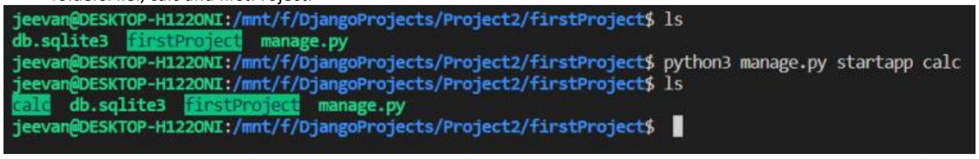
```
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2$ ls
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2$ django-admin startproject firstProject
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2$ ls
firstProject
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2$ cd firstProject
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$ ls
firstProject  manage.py
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$ cd firstProject
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject/firstProject$ ls
__init__.py  asgi.py  settings.py  urls.py  wsgi.py
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject/firstProject$
```

- We already created a folder for containing projects under Django Projects under Project2 we want to create our first project of Django named firstProject. For starting the Django project named firstProject we need to use "Django-admin startproject nameOfTheProject" , in our case it is django-admin startproject firstProject. We can see if the following command executed then all the following files are created automatically.
- We got all the basic files to run an application , but as we need a server to execute them on, Django provide a default server to execute.
- We need to run the manage.py under our project on the server, for that do the following.
- If we run the manage.py on the server, we get a local host address as the following. It tells us that webpage hosted on following local host address.

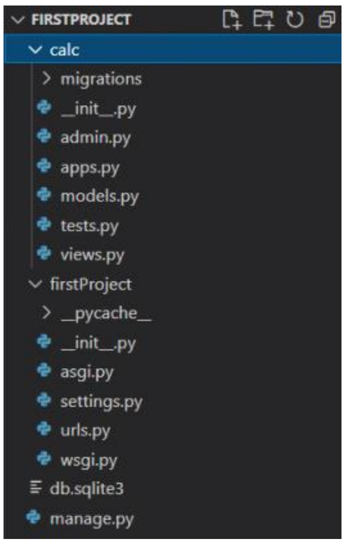


```
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$ ls
firstProject  manage.py
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 10, 2022 - 12:37:49
Django version 4.0.3, using settings 'firstProject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

- To quit the server use CTRL C
- To develop dynamic pages web use applications like Django. Dynamic mean each user finds a different interface according to his information.
- Now if we want to create a new django app , we need to create by following commads.
- The settings for the app is different from the original project settings , so we are having two folders. i.e., calc and firstProject.



```
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$ ls
db.sqlite3  firstProject  manage.py
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$ python3 manage.py startapp calc
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$ ls
calc  db.sqlite3  firstProject  manage.py
jeevan@DESKTOP-H122ONI:/mnt/f/DjangoProjects/Project2/firstProject$
```



- Now when we type localhost:8000 on the browser , we would get some page as the front page . Because we are mapping the page that we want to present under urls.py of project's section (but not under app section)

Django views are a key component of applications built with the framework. At their simplest they are a Python function or class that takes a web request and return a web response.

Views are used **to do things like fetch objects from the database, modify those objects if needed, render forms, return HTML, and much more**.

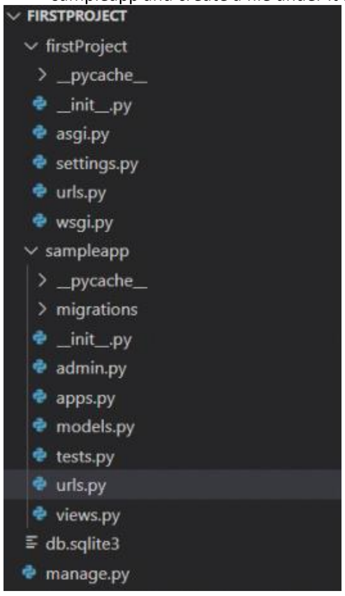
Django runs through each URL pattern, in order, and stops at the first one that matches the requested URL, matching against path_info . Once one of the URL patterns matches, Django imports and calls the given view, which is a Python function (or a class-based view).

In url.py, the most important thing is the "urlpatterns" tuple. It's **where you define the mapping between URLs and views**. A mapping is a tuple in URL patterns like − from django. conf. urls import patterns, include, url from django.

It is just that **django gives you the option to name your views in case you need to refer to them from your code, or your templates**. This is useful and good practice because you avoid hardcoding urls on your code or inside your templates. 10-Oct-2012

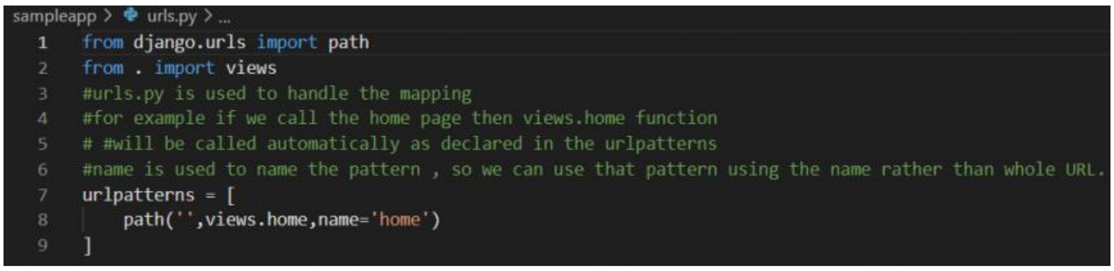
This **tells Django to search for URL patterns in the file books/urls.py** . For example, A URL request to /books/crime/ will match with the second URL pattern. As a result, Django will call the function views. 19-Jul-2019

- As we are getting error if we use app name as calc , we are creating another app named as sampleapp and create a file under it named as "urls.py".



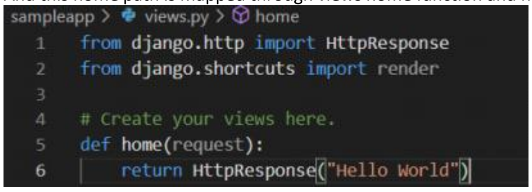
Now as we created new app named as sampleapp, we need to create content for it. To display on the main page. These apps are similar to the templates of angular.

- For this template, have urls.py as the following.
- Urls.py is used for URL mapping



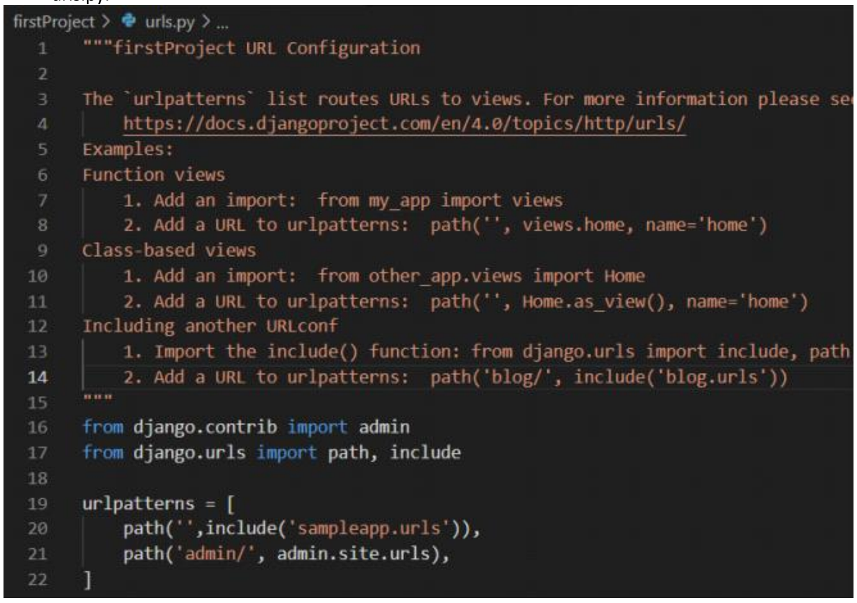
```
sampleapp > urls.py > ...
from django.urls import path
from . import views
#urls.py is used to handle the mapping
#for example if we call the home page then views.home function
# #will be called automatically as declared in the urlpatterns
#name is used to name the pattern , so we can use that pattern using the name rather than whole URL.
urlpatterns = [
    path('',views.home,name='home')
]
```

And this home path is mapped through views home function and name that path as home.



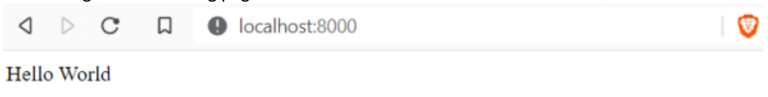
```
sampleapp > views.py > home
from django.http import HttpResponse
from django.shortcuts import render

# Create your views here.
def home(request):
    return HttpResponse("Hello World")
```

- The above urls.py is sample app 's app but we also need to declare it under main project urls.py.



```
firstProject > urls.py > ...
"""firstProject URL Configuration

The `urlpatterns` list routes URLs to views. For more information please se
    https://docs.djangoproject.com/en/4.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('',include('sampleapp.urls')),
    path('admin/', admin.site.urls),
]
```

- In the main url file we are declaring the path mapping for home to the sample app's URL file. So the following app's file that we declared will be executed further.
- We get the following page on the localhost:8000



localhost:8000

Hello World