Installation

1. Install Angular CLI (A command-line interface (CLI) to automate your development workflow.)
2. npm install -g @angular/cli **or** npx -p @angular/cli ng version
3. $ ng –v (check version to see If installation is done)

Create your first angular project

1. ng new angularlms **or** npx ng new angularlms
   (This will create an angular project in the folder named angularlms)
   a. answer questions like; what name would you like to use for the project?
   b. Would you like to add angular routing? Y
   c. Which stylesheet format would you like to use? Css
2. cd angularlms
3. ng serve **or** npx ng serve --open
   (Your angular server is up and running on localhost:4200)

Cli structure

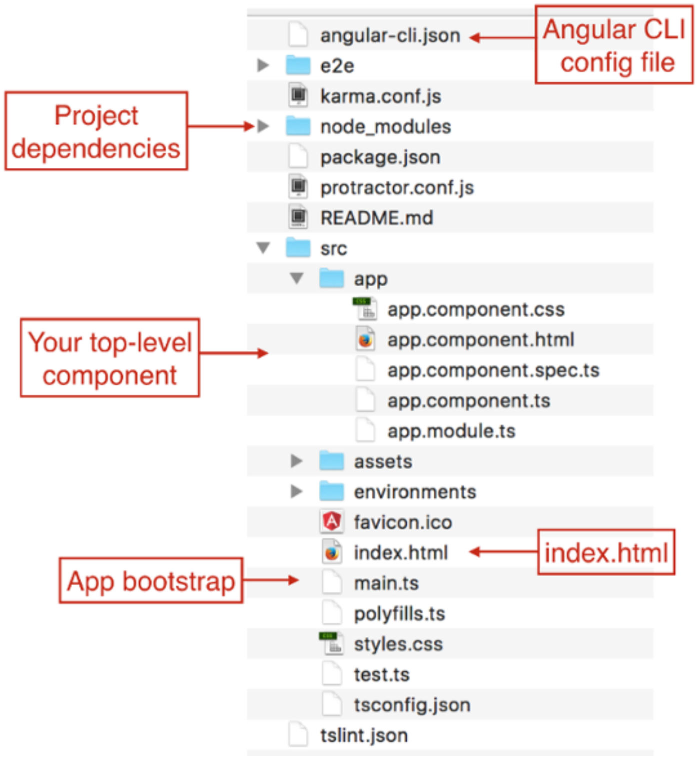


Figure 1

Angular Components

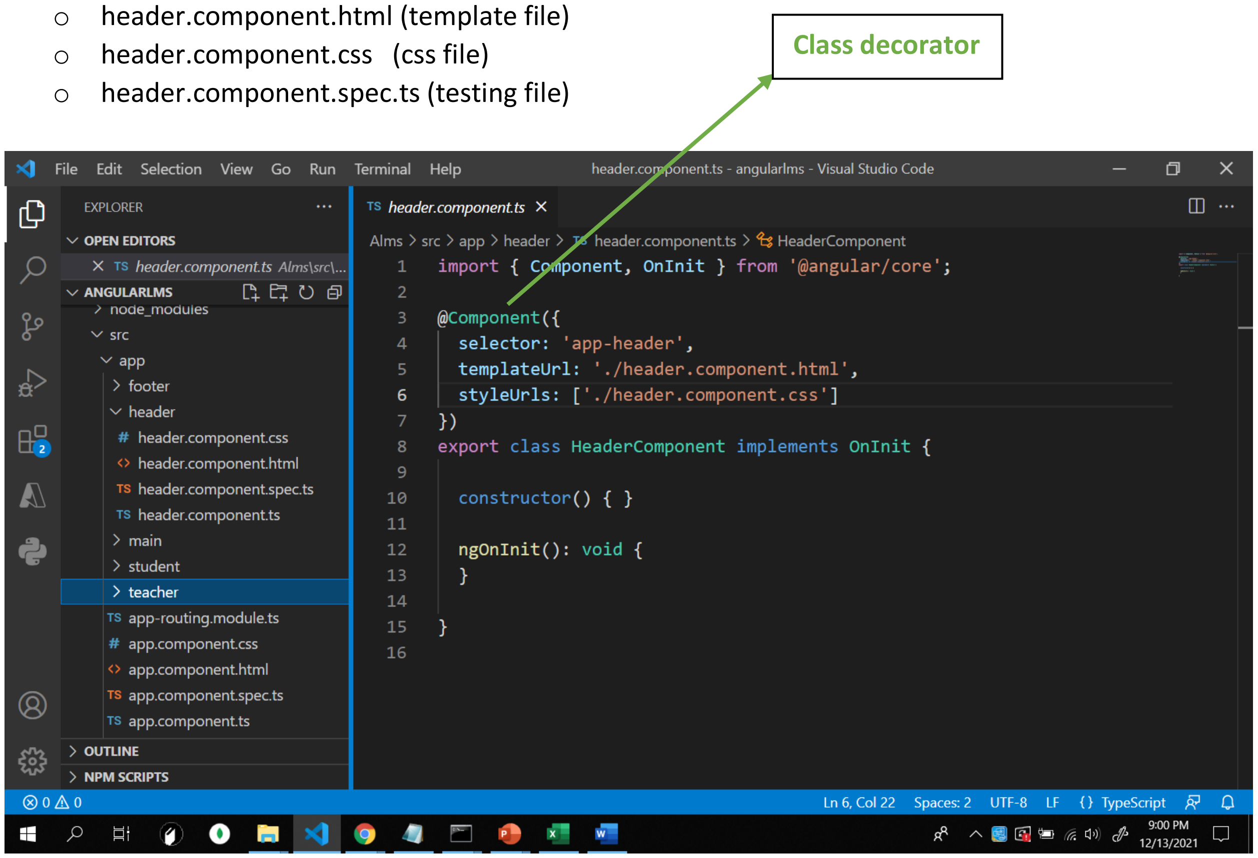
Components are the most basic UI building block of an Angular app. An Angular app contains a tree of Angular components. Each component controls a patch of screen called a view. Each component consists of:

- An HTML template that declares what renders on the page

- A Typescript class that defines behaviour
- A CSS selector that defines how the component is used in a template
- Optionally, CSS styles applied to the template

Create a component using angular CLI:

1. Ng generate component header **or** npx ng generate component headerIt will create a folder named header in your angular app. The folder will have:
    - header.component.ts (component file)
    - header.component.html (template file)
    - header.component.css (css file)
    - header.component.spec.ts (testing file)



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

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

  constructor() { }

  ngOnInit(): void {
  }

}
```

Figure 2

Decorators

In angular there are four main types of decorators, where each type has a unique role:

1. Class decorator e.g., @component and @NgModule
2. Property decorators for properties inside classes e.g., @Input and @Output
3. Method decorators for methods inside classes e.g., @HostListener
4. Parameter decorators for parameters inside class constructors e.g., @Inject

Class decorator

These are top level decorators that provide metadata about the classes. They tell that a particular class is a component or module, e.g. **@component** identifies the class immediately below it as a component class and specifies its metadata. In figure 2 HeaderComponent is a class with no special angular notation or syntax at all. Its not a component until you mark it as one with the @component decorator.

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

  constructor() { }

  ngOnInit(): void {
  }
```

@NgModule

```
@NgModule({
  imports: [],
  declarations: [],
})
export class ExampleModule {
  constructor() {
    console.log('Hey I am a module!');
  }
}
```

Make a simple SPA (Single Page Application)

Let's make a Simple angular single page application (SPA), in which there will be three links, Home, Teacher Panel and Student panel. On clicking home, it should display the main page, on clicking the teacher panel it should display teacher panel and same for the student panel link. Also, we want to structure our app in such a way that it should have a header, footer and the main body which will contain the contents of the page.

For this we need to create three components and apply basic routing (navigation links) to it.

1. Create Header, Footer, Main, Teacher and Student components
   a. ng generate component header
   b. ng generate component footer
   c. ng generate component main
   d. ng generate component teacher
   e. ng generate component student
2. Copy paste following in the index.html

```
<!doctype html>
```

```html
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Alms</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap
.min.css" integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">
</head>
<body>
  <app-root></app-root>
  <script src="../node_modules/jquery/dist/jquery.js"></script>
</html>
```

3. Copy paste the following html tags in the header.component.html

```html
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container">
        <ul class="navbar-nav mr-auto">
            <li class="navbar-brand mb-0 h1">Home<li>
            <li class="navbar-brand mb-0 h1"> Teacher Panel</li>
            <li class="navbar-brand mb-0 h1Student Panel</li>
        </ul>
    </div>
</nav>
```

4. Copy paste the following in the footer.component.html

```html
<!-- Footer -->
<footer class="page-footer font-small blue pt-4">

    <!-- Footer Links -->
    <div class="container-fluid text-center text-md-left">

      <!-- Grid row -->
      <div class="row">

        <!-- Grid column -->
        <div class="col-md-6 mt-md-0 mt-3">

          <!-- Content -->
          <h5 class="text-uppercase">Footer Content</h5>
          <p>Here you can use rows and columns to organize your footer
content.</p>

        </div>
        <!-- Grid column -->

        <hr class="clearfix w-100 d-md-none pb-3">
        <div class="col-md-3 mb-md-0 mb-3">
          <h5 class="text-uppercase">Links</h5>
          <ul class="list-unstyled">
            <li>
              <a href="#!">Link 1</a>
            </li>
          </ul>
        </div>
        <div class="col-md-3 mb-md-0 mb-3">
          <h5 class="text-uppercase">Links</h5>
          <ul class="list-unstyled">
            <li>
              <a href="#!">Link 1</a>
            </li>
          </ul>
        </div>
      </div>
    </div>
    <div class="footer-copyright text-center py-3">© 2020 Copyright:
      <a href="https://mdbootstrap.com/"> MDBootstrap.com</a>
    </div>
  </footer>
```

5. Copy paste the following code in the main.component.html

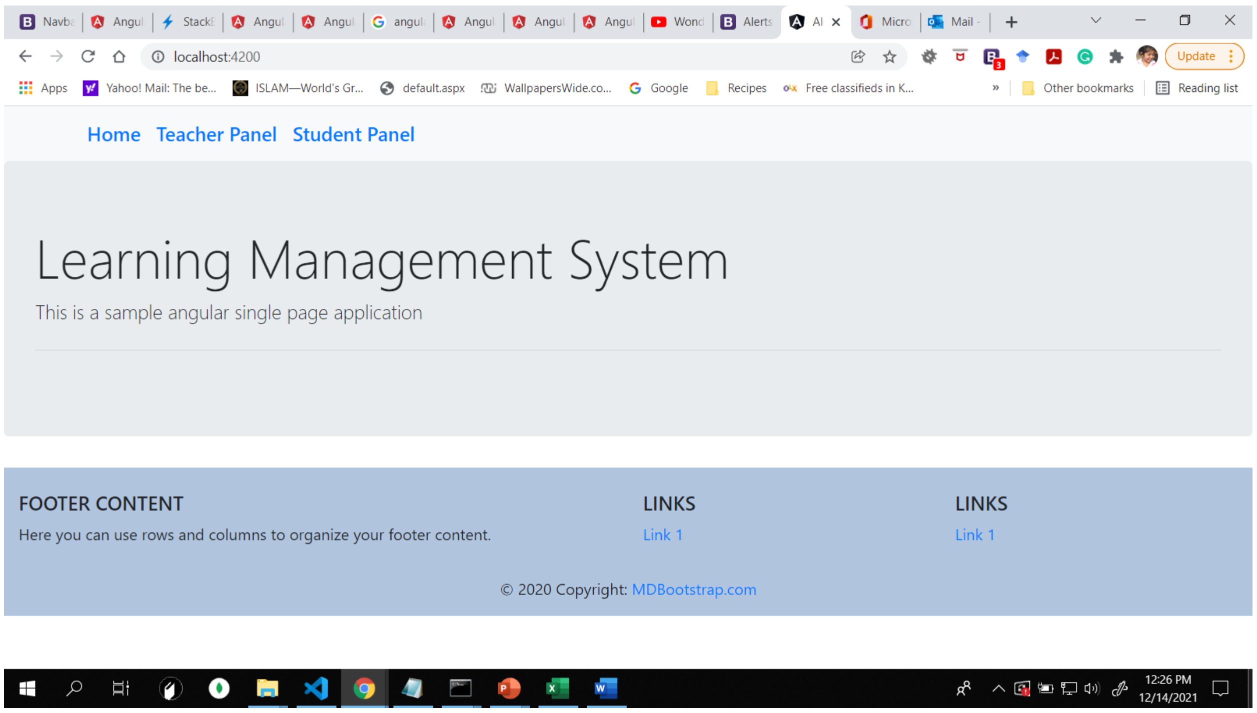
```
<div class="jumbotron">
```

```
    <h1 class="display-4">Learning Management System</h1>
    <p class="lead">This is a sample angular single page
application</p>
    <hr class="my-4">
</div>
```

6. Copy paste the following in the app.component.html

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

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

Your SPA will look like this.



Home Teacher Panel Student Panel

Learning Management System

This is a sample angular single page application

FOOTER CONTENT

Here you can use rows and columns to organize your footer content.

LINKS

Link 1

LINKS

Link 1

© 2020 Copyright: MDBootstrap.com

Adding Links to your SPA

Let's add links to Home, Teacher Panel and Student Panel.
For this we need to make some changes in the following files:

- app-routing.module.ts
- header.component.html (as this file holds the navigation bar)
- app.component.html (our app component holds the header component and should have a <router-outlet>)

So,

7. For routing (adding navigation links), open app-routing.module.ts file.

Routing lets you display specific views of your application depending on the URL path. App-routing.module.ts is generated automatically during installation if choosen. Otherwise, you need to manually create this file.
In the app-routing.module.ts we define the routes. Each route typically has two properties. The first is the **path** that specifies the URL path and the second is the **component**, that specifies what component your application should display for that path.

a. Import the components from the component folders. These components are defined in the routes defining that which component will load when a specific link path is clicked.
b. Add a JS object inside the Routes [] defining the path and the component which will be loaded after clicking a specific link.

```
import { TeacherComponent } from './teacher/teacher.component';
import { StudentComponent } from './student/student.component';
import { MainComponent } from './main/main.component';

const routes: Routes = [
  {
    path:'Teacher',
    component:TeacherComponent
  },
  {
    path:'Student',
    component:StudentComponent
  },
  {
    path:'',
    component:MainComponent
  }
];
```

c. Update header.component.html by adding <a> and routerLink attributes as shown in the code bellow.
RouterLink defines the path of the link which is defined in the app-routing.module.ts file.

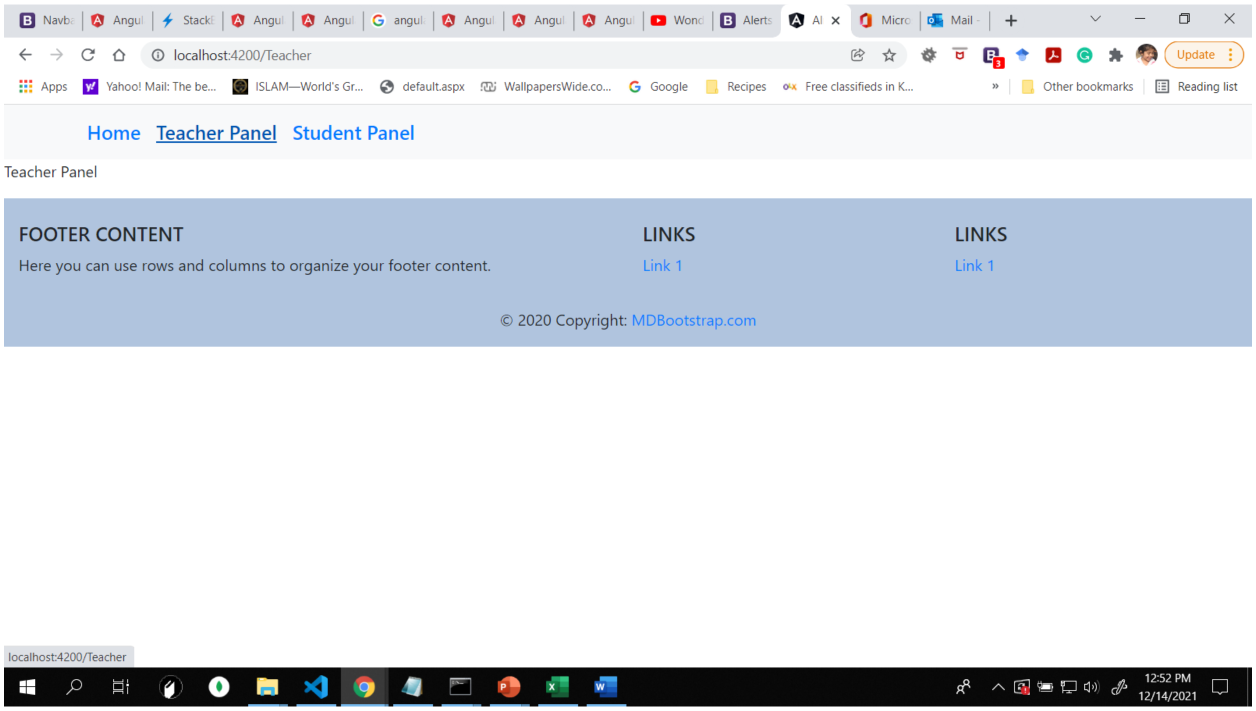
```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container">
        <ul class="navbar-nav mr-auto">
            <li class="navbar-brand mb-0 h1"><a routerLink =
"">Home</a></li>
```

```
            <li class="navbar-brand mb-0 h1"> <a routerLink =
"Teacher">Teacher Panel</a></li>
            <li class="navbar-brand mb-0 h1"><a routerLink =
"Student">Student Panel</a></li>
        </ul>
    </div>
</nav>
```

d. Update the app.component.html file by adding the **<router-outlet>** directive. The **router-outlet** is a directive that's available from the @angular/router package and is used by the router to mark where in a template, a matched component should be inserted.

```
<app-header></app-header>
 <router-outlet></router-outlet>
<app-footer></app-footer>
```

By now your SPA will have three working links: Home, Teacher Panel and Student Panel



Home Teacher Panel Student Panel

Teacher Panel

FOOTER CONTENT

Here you can use rows and columns to organize your footer content.

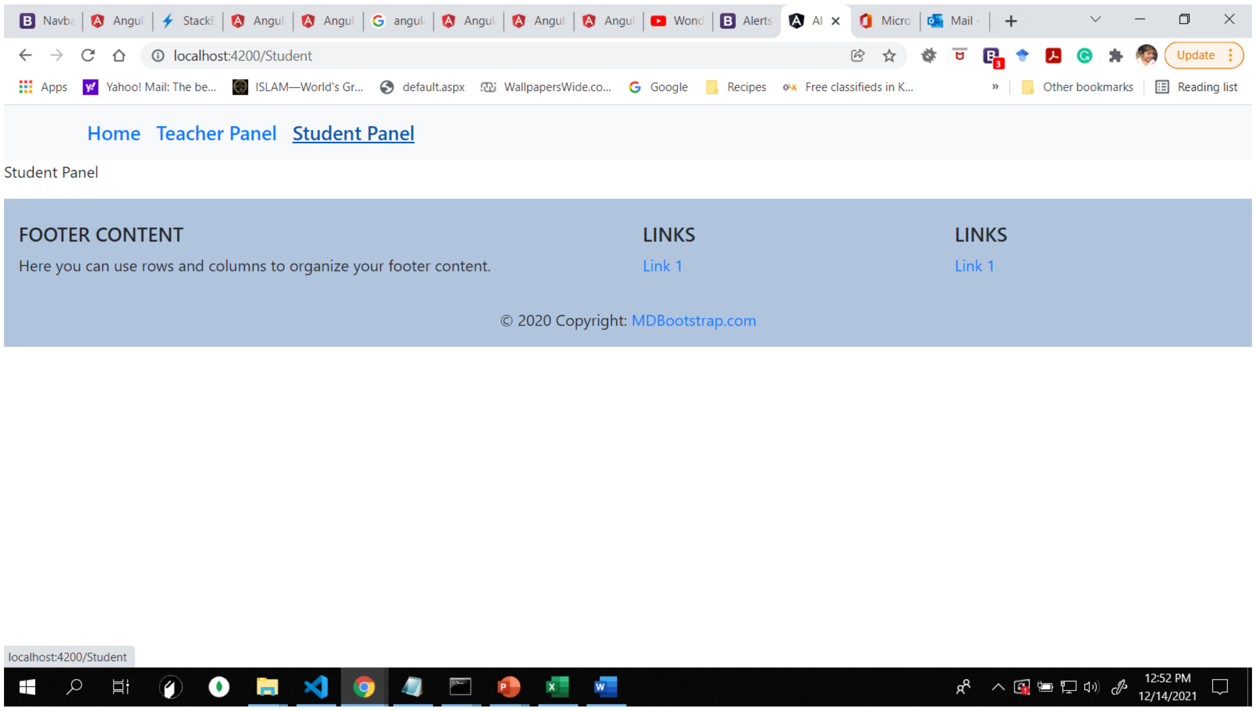
LINKS

Link 1

LINKS

Link 1

© 2020 Copyright: MDBootstrap.com



Student Panel

FOOTER CONTENT

Here you can use rows and columns to organize your footer content.

LINKS

Link 1

LINKS

Link 1

© 2020 Copyright: MDBootstrap.com

Nested Routing

What if we want multiple links inside each component i.e., Teacher Panel and Student Panel? We need to implement nested routing in our SPA. For this lets first add some buttons in the teacher component giving some options to be clicked on.

1. Open teacher.component.html and update the code as follows:

```
<button type="button" class="btn btn-primary>View Students</button>
<button type="button" class="btn btn-secondary">Add
Assignment</button>
<button type="button" class="btn btn-success">Add Quiz</button>
<button type="button" class="btn btn-danger">Teacher Profile</button>
<button type="button" class="btn btn-warning">Classes</button>
```

2. Add a new component viewstd by using the following command in the cmd
    a. ng generate new component viewstd
3. Put a link on the text **View Students** in the teacher.component.html file.

```
<button type="button" class="btn btn-primary><a
routerLink=”Viewstd”>View Students</a></button>
<button type="button" class="btn btn-secondary">Add
Assignment</button>
<button type="button" class="btn btn-success">Add Quiz</button>
```

```html
<button type="button" class="btn btn-danger">Teacher Profile</button>
<button type="button" class="btn btn-warning">Classes</button>
```

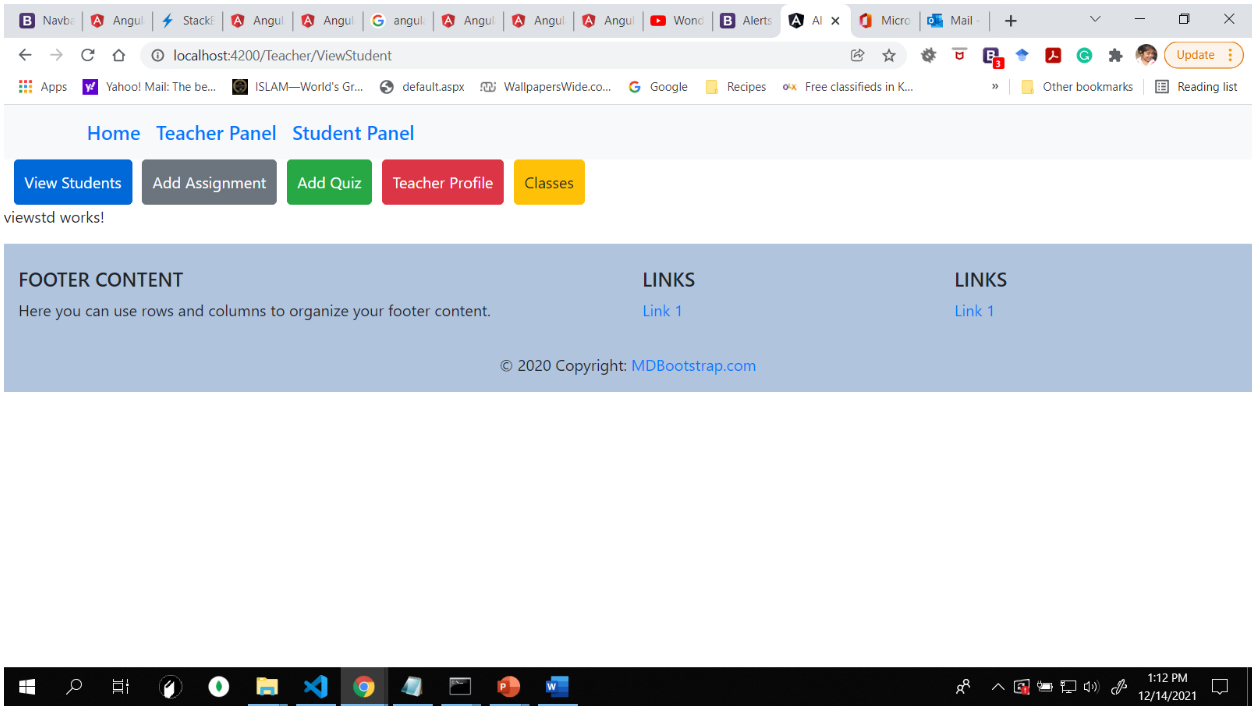
4. Update the app-routing.module.ts, and add a child to the teacher path as shown in the code below.

```typescript
const routes: Routes = [
  {
    path:'Teacher',
    component:TeacherComponent,
    children:[{
      path:'ViewStudent',
      component: ViewstdComponent
    }]
  },
  {
    path:'Student',
    component: StudentComponent
  },
  {
    path:'',
    component: MainComponent
  }
];
```

5. Update the teacher.component.html file by adding a <router-outlet> directive, as shown in the code below.

```html
<button type="button" class="btn btn-primary"><a routerLink =
"ViewStudent">View Students</a></button>
<button type="button" class="btn btn-secondary">Add
Assignment</button>
<button type="button" class="btn btn-success">Add Quiz</button>
<button type="button" class="btn btn-danger">Teacher Profile</button>
<button type="button" class="btn btn-warning">Classes</button>
<router-outlet></router-outlet>
```

6. Your SPA has now nested link and will look like this:



viewstd works!

FOOTER CONTENT

Here you can use rows and columns to organize your footer content.

LINKS

Link 1

LINKS

Link 1

© 2020 Copyright: MDBootstrap.com

Display teacher profile data.

Let's put some functionality in the component we created. Assume that a teacher wants to view his/her profile.

1. Create a component teacher-profile.
2. Add following code in the teacher-profile.html

```
<div>
    <h3>Name: </h3>
    <h3>Designation: </h3>
    <input id="class" placeholder="class">
    <span></span>
</div>
```

3. Create an interface declaring the properties of teacher. For this create a file teacher.ts in the teacher-profile folder, and write:

```
export interface Teacher {
    name: string;
    designation: string;
    class: string
  }
```

4. Import the interface file in the teacher-profile.component.ts

```
import {Teacher} from './teacher';
```

5. Implement the interface by defining the properties in the teacher-profile.component.ts class as follows:

```
export class TeacherProfileComponent implements OnInit {
  teacher: Teacher = {
    name: "Dr Gul Zahra",
    designation: 'Assitant Professor',
    class: "CSc 7"
  };
  constructor() { }

  ngOnInit(): void {
  }

}
```

The code above shows that the class `TeacherProfileComponent` has a teacher object of type Teacher with name, designation, and a class.

Now to show the data in the view, we need to do **interpolation**.
Interpolation refers to embedding expressions into marked up text. By default, interpolation uses the double curly braces {{ }} as delimiters.

6. To bind our data to the view we make use of interpolation and update the teacher-profile.component.html as follows.
7. To perform two-way data binding we use [(ngmodel)]. It creates two-way data bindings for reading and writing input-control values.

```
<ul class="list-group">
    <li class="list-group-item list-group-item-action list-group-item-dark">
        Name
    </li>
    <li class="list-group-item d-flex justify-content-between align-items-center">
        {{teacher.name}}
      <span class="badge badge-primary badge-pill">14</span>
    </li>
    <li class="list-group-item list-group-item-action list-group-item-
dark">
        Designation
    </li>
    <li class="list-group-item d-flex justify-content-between align-
items-center">
        {{teacher.designation}}
    </li>
    <li class="list-group-item d-flex justify-content-between align-
items-center">
        <input id="class" [(ngModel)]="teacher.class"
placeholder="class">
        <span class="badge badge-primary badge-
pill">{{teacher.class}}</span>
    </li>
</ul>
```

8. Update the teacher.component.html by adding a link on Teacher Profile

```
<button type="button" class="btn btn-danger"><a routerLink =
"ViewProfile">Teacher Profile</a></button>
```

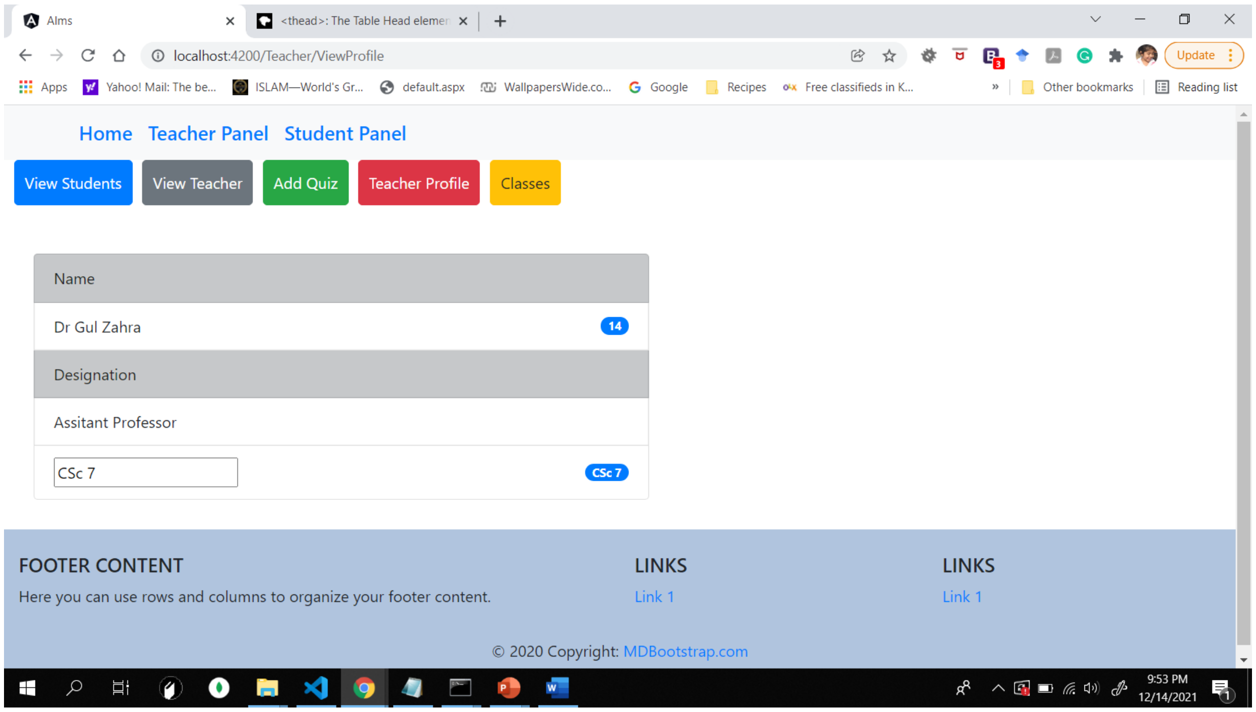
9. Update the app-routing.module.ts file

```
import {TeacherProfileComponent} from './teacher-profile/teacher-
profile.component';
```

```
{
    path:'Teacher',
    component:TeacherComponent,
    children:[{
      path:'ViewStudent',
      component: ViewstdComponent
    },
    {
      path:'ViewProfile',
      component: TeacherProfileComponent
```

```
        }
      ]
  }
```

10. After this your SPA will look like the following screen shot after clicking Teacher Profile



Name	
Dr Gul Zahra	14
Designation	
Assitant Professor	
CSc 7	CSc 7

FOOTER CONTENT

Here you can use rows and columns to organize your footer content.

LINKS

Link 1

LINKS

Link 1

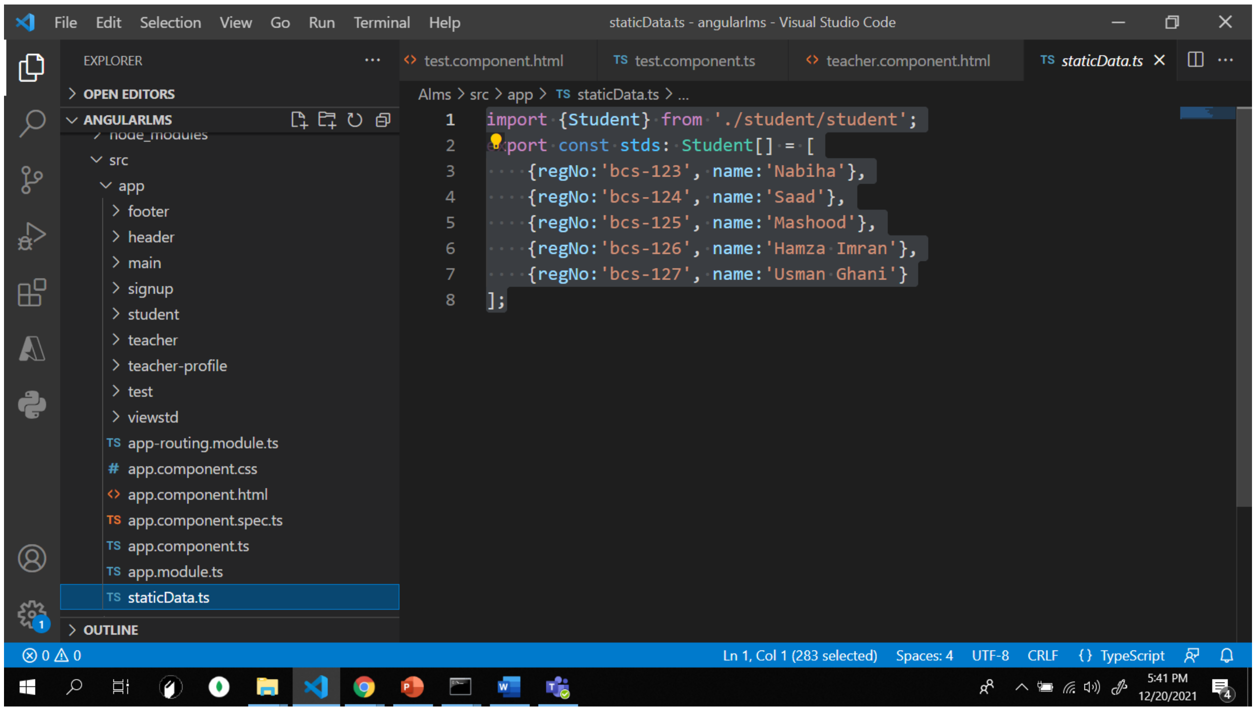
© 2020 Copyright: MDBootstrap.com

Display students' data

Now we want to display a list of students when we click on **view students** link. Assume that for the time being we have a list of students in an array. To achieve this,

1. Make a new file named staticData.ts in the app folder, and insert the following code:

```
import {Student} from './student/student';
export const stds: Student[] = [
    {regNo:'bcs-123', name:'Nabiha'},
    {regNo:'bcs-124', name:'Saad'},
    {regNo:'bcs-125', name:'Mashood'},
    {regNo:'bcs-126', name:'Hamza Imran'},
    {regNo:'bcs-127', name:'Usman Ghani'}
];
```


```
import {Student} from './student/student';
export const stds: Student[] = [
    {regNo:'bcs-123', name:'Nabiha'},
    {regNo:'bcs-124', name:'Saad'},
    {regNo:'bcs-125', name:'Mashood'},
    {regNo:'bcs-126', name:'Hamza Imran'},
    {regNo:'bcs-127', name:'Usman Ghani'}
];
```

2. Generate a new component viewstd using the command
   a. ng generate component viewstd
3. In the viewstd.component.ts file do the following:
   a. Import the staticData
   b. Define a data member of the class by assigning the array declared in the staticData file.

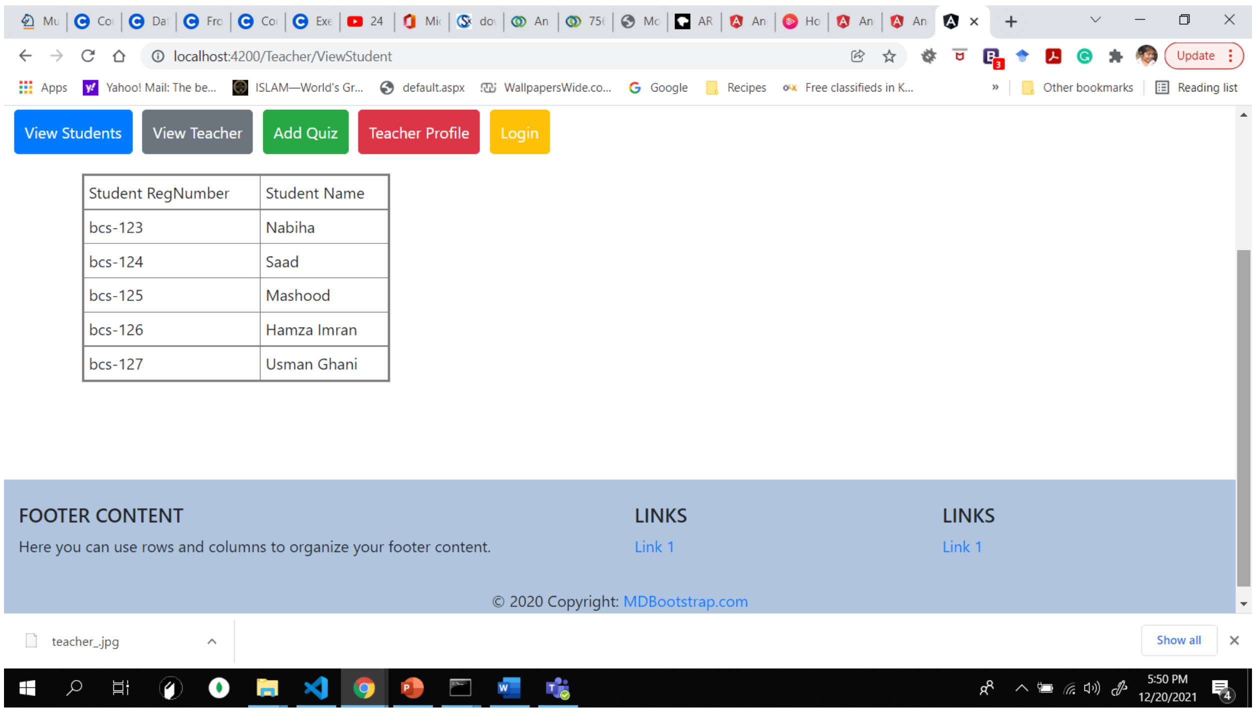
```
import {stds} from '../staticData';
```

```
export class ViewstdComponent implements OnInit {
  Class_stds = stds;
  constructor() { }
  ngOnInit(): void {
  }

}
```

4. In the view of the viewstd component create a table and make use of *ngFor directive to generate multiple rows of the table showing the students reg# and name.

```
<table>
    <tr>
        <td>Student RegNumber</td>
        <td>Student Name</td>
    </tr>
    <tr *ngFor="let student of Class_stds">
        <td>{{student.regNo}}</td>
        <td>{{student.name}}</td>
    </tr>

</table>
```

By now if you click on view Students link it should render as follows:



Student RegNumber	Student Name
bcs-123	Nabiha
bcs-124	Saad
bcs-125	Mashood
bcs-126	Hamza Imran
bcs-127	Usman Ghani

FOOTER CONTENT

Here you can use rows and columns to organize your footer content.

LINKS

Link 1

LINKS

Link 1

© 2020 Copyright: MDBootstrap.com

This example makes use of directive ***ngFor.**

In angular **directives** are components without a view/template. Directives are simply the instructions in DOM, which specifies how to place your components and business logic in angular. It is also a class, which is declared as **@directive**. We use Angular's built-in directives to manage forms, lists, styles, and what users see. There are different types of angular directives:

1. **Component directives**
   Component directive is used to specify the HTML templates. It has structure design and the working pattern of how the component should be processed, instantiated, and used at runtime. It is the most commonly used directive in any Angular project.
   sample-page.component.css: contains all the CSS styles for the component.
   sample-page.component.html: contains all the HTML code used by the component to display itself.
   sample-page.component.ts: contains all the code used by the component to control its behaviour.

2. Attribute directives

- Attribute directive alter the appearance or behavior of an existing element.
  The **ngModel** directive, which implements two-way data binding, is an example of an attribute directive.
  <input [(ngModel)] = "movie.name">
- **ngStyle** directive is used when we need to apply multiple inline styles dynamically to an element. E.g. in the view write
  <p [ngStyle]="getInlineStyles(framework)">{{framework}}</p>
  And in the component class:
  ```
          let styles = {
          'color': framework.length > 3 ? 'red' : 'green',
          'text-decoration': framework.length > 3 ? 'underline' : 'none'
          };
          return styles;
          }
  ```

- ngClass directive is used when we need to apply multiple classes dynamically. E.g. in the view/template write:
  <p [ngClass]="geClasses(framework)">{{framework}}</p>
  In the component class write:
  ```
  geClasses(framework) {
          let classes = {
          red: framework.length > 3,
          bolder: framework.length > 4
          };
          return classes;
  }
  ```
  In the component.css file write:
  ```
  .red {
  color: red;
  text-decoration: underline;
  }
  .bolder {
  font-weight: bold;
  }
  ```

3. Structural directives

- ***ngFor** is a repeater directive. It is used for displaying a list of items. (The above example shows its use).
  ```
  <tr *ngFor="let student of Class_stds">
  ```
- ***ngIf** is used for adding or removing elements from DOM dynamically e.g., <element *ngIf="condition"> content </element>

- ***ngSwitch** conditionally instantiate one template from a set of choices, depending on the value of a selection, e.g.

  <div [ngSwitch]="selectedCar">

  <template [ngSwitchCase]="'Bugatti'">I am Bugatti</template>

  <template [ngSwitchCase]="'Mustang'">I am Mustang</template>

  <template [ngSwitchCase]="'Ferrari'">I am Ferrari</template>

  <template ngSwitchDefault>I am somebody else</template>

  </div>

4. Custom directives

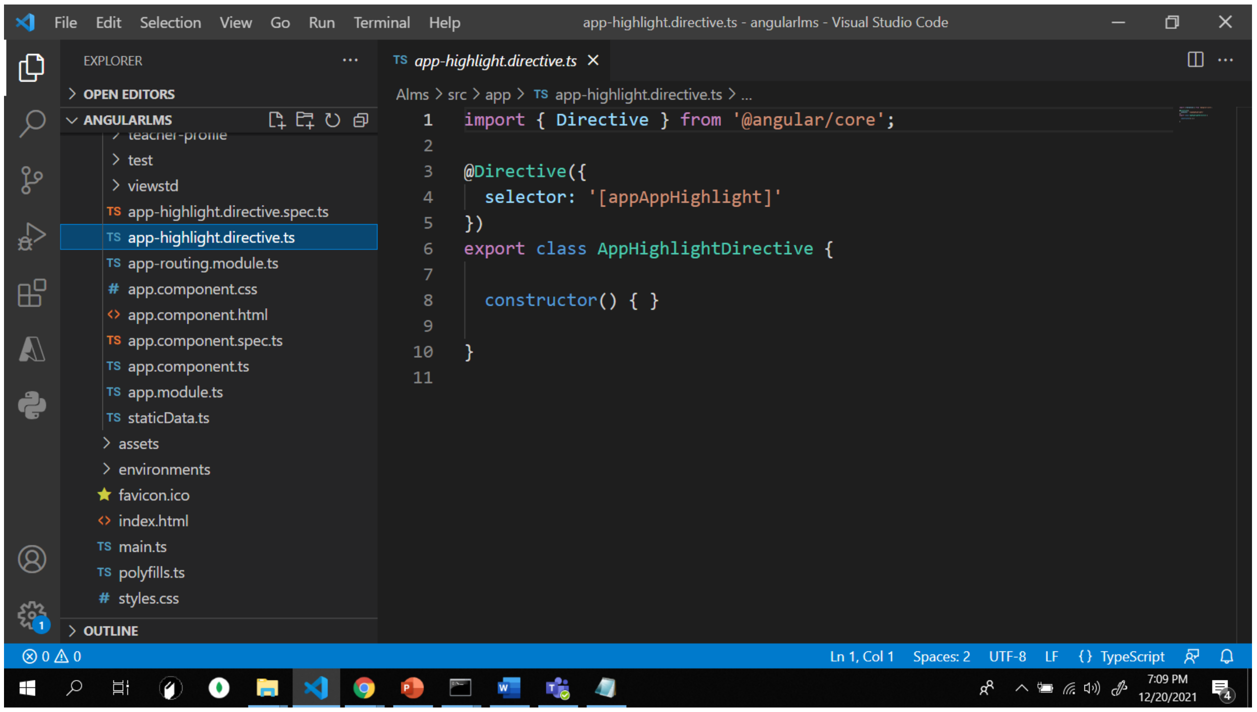
Creating a custom directive is just like creating an angular component. To create a custom directive, we have to replace **@componet** decorator with **@directive** decorator.

Example:

Consider we want to create a custom attribute directive, which will highlight the selected DOM element by setting an elements background color. To achieve this do the following:

a. Create an app-highlight directive by running the following command:
ng generate directive app-highlight.

b. The app-highlight.directive.ts file will have the following code.



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

@Directive({
  selector: '[appAppHighlight]'
})
export class AppHighlightDirective {

  constructor() { }

}
```

c. Now import ElementRef in the above file like:

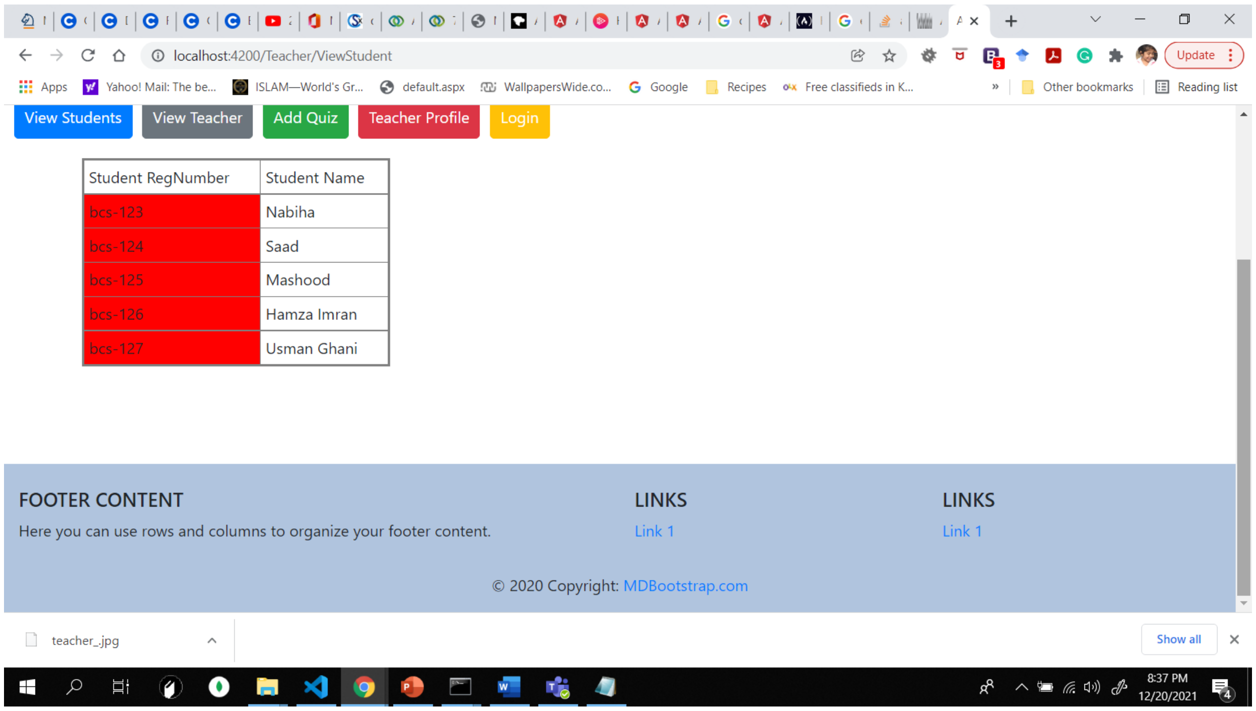
```
import { Directive, ElementRef } from '@angular/core';
```

ElementRef is a class that can hold a reference to a DOM element to be accessed. Create an ElementRef object inroder to access the DOM element using its nativeElement property. To do this amend the app-hightlight.directive.ts class as follows:

```
export class AppHighlightDirective {

  constructor(private eleRef: ElementRef) {
    eleRef.nativeElement.style.background = "red";
   }

}
```

d. I am going to use this custom directive in **viewstd** component. Use the custom attribute selector with any element in the view of viewstd component to set its background color red, as used in the viewstd.component.html file

```
<table>
    <tr>
        <td>Student RegNumber</td>
        <td>Student Name</td>
    </tr>
    <tr *ngFor="let student of Class_stds">
        <td appAppHighlight>{{student.regNo}}</td>
        <td>{{student.name}}</td>
    </tr>

</table>
```


View Students | View Teacher | Add Quiz | Teacher Profile | Login

Student RegNumber	Student Name
bcs-123	Nabiha
bcs-124	Saad
bcs-125	Mashood
bcs-126	Hamza Imran
bcs-127	Usman Ghani

FOOTER CONTENT

Here you can use rows and columns to organize your footer content.

LINKS

Link 1

LINKS

Link 1

© 2020 Copyright: MDBootstrap.com

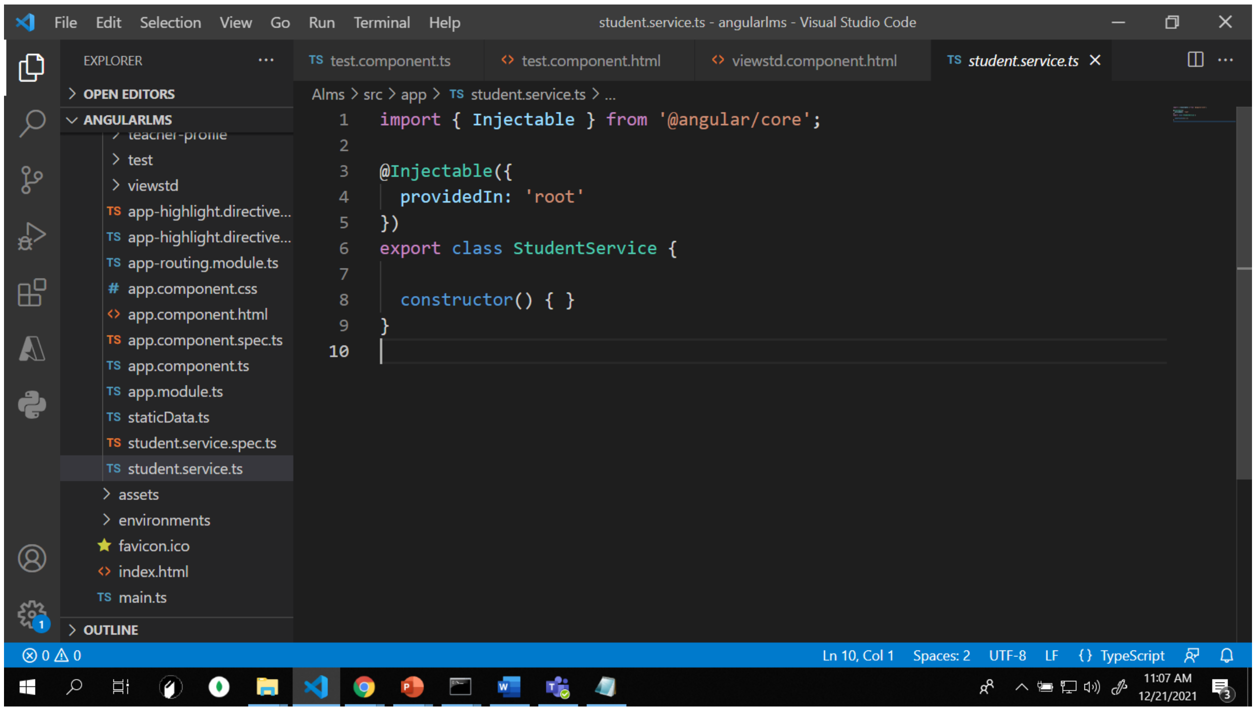
Services and their use

Services provides specific functionality in an Angular application. In an Angular application, one or more services can be used. Similarly, an Angular component may depend on one or more services.

Whenever we need to reuse the same data and logic across multiple components in the application, we should go for angular services. This means whenever you see the same logic or data-access code duplicated across multiple components, then you need to think about refactoring the same logic or data access code into a service.

Understanding angular service with an example:

1. To generate a service, use the following command:
   Ng generate service Student
   Two files will be generated following this command i.e. (student.service.ts and student.service.spec.ts)



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

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

  constructor() { }
}

```

2. For the time being, we will add some data regarding students in this service and write a method getStudents() which will return an array of students.
3. Copy paste following code in the student.service.ts file:

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

@Injectable({
  providedIn: 'root'
})
export class StudentService {
  getStudents(): any[] {
    return [
      {
        ID: 'std101', FirstName: 'Nabiha', LastName: 'Tufail',
        Branch: 'CSE', DOB: '29/02/1988', Gender: 'Female'
    },
    {
        ID: 'std102', FirstName: 'Saad', LastName: 'Ali',
        Branch: 'ETC', DOB: '23/05/1989', Gender: 'Male'
    },
    {
        ID: 'std103', FirstName: 'Sana', LastName: 'Asghar',
        Branch: 'CSE', DOB: '24/07/1992', Gender: 'Female'
    },
    {
        ID: 'std104', FirstName: 'Hina', LastName: 'Nasir',
        Branch: 'ETC', DOB: '19/08/1990', Gender: 'Female'
    },
    {
        ID: 'std105', FirstName: 'Ali', LastName: 'Ahsan',
        Branch: 'CSE', DOB: '12/94/1991', Gender: 'Male'
    }
    ];
  }
  constructor() { }
}
```

@Injectable() decorator in angular is used to inject other dependencies into the service.

To use this service in a component we need to inject it. To do this we need to perform three steps:

- Import the service in the component we want it to be used
- Register the service by specifying the provider
- Use the service

Lets say we want to use it in our viewstdService component. Do the following

1. Import StudentService in the viewstdService component.

```
import { StudentService } from '../student.service';
```

2. Register the service in the component by declaring it the providers array.

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

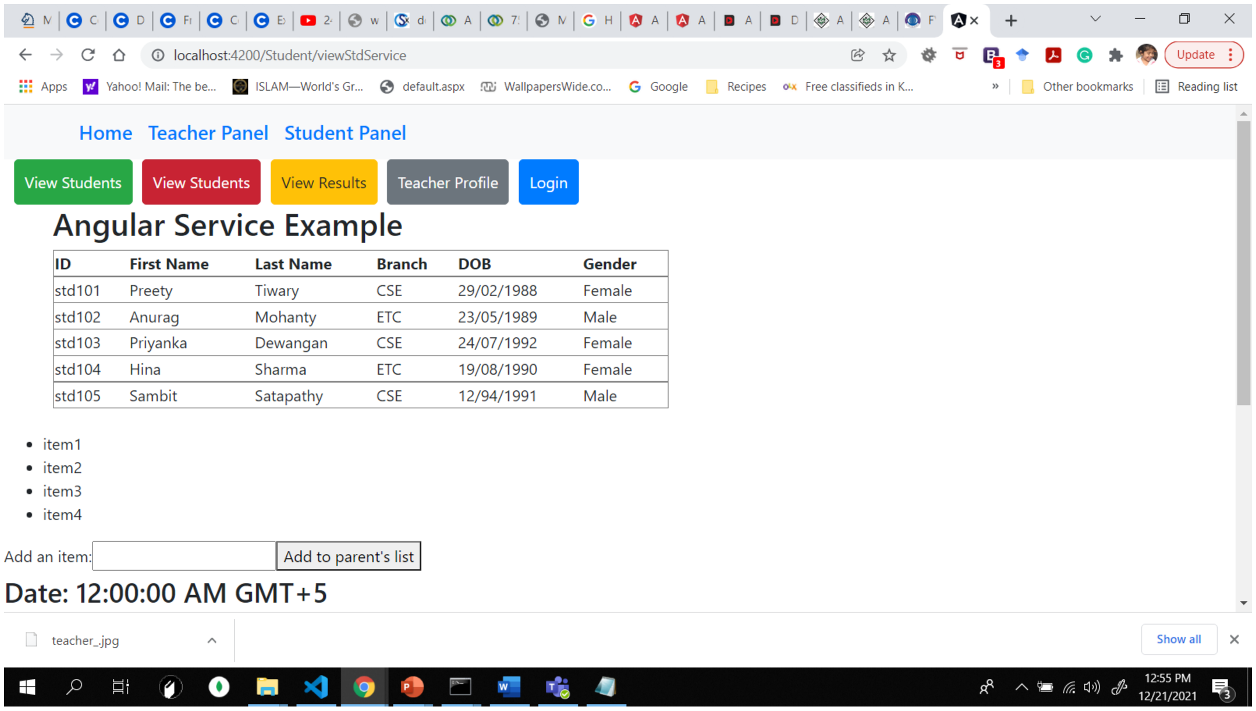
3. Inject the StudentService using the constructor. The private variable stdService points to StudentService instance which is then available throughout the class. The code will look like this:

```
export class ViewstdServiceComponent implements OnInit {
  students: any[];
  constructor(private stdService : StudentService) {
      this.students = this.stdService.getStudents();
  }
  ngOnInit(): void {
  }

}
```

As you can see in the above code, we declare one variable called students and then initialize this variable using the student service.

4. Write the following code in the components view.

```
<h2>Angular Service Example</h2>
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Branch</th>
            <th>DOB</th>
            <th>Gender</th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor='let student of students'>
            <td>{{student.ID}}</td>
            <td>{{student.FirstName}}</td>
            <td>{{student.LastName}}</td>
            <td>{{student.Branch}}</td>
            <td>{{student.DOB}}</td>
            <td>{{student.Gender}}</td>
        </tr>
    </tbody>
</table>
```



Home Teacher Panel Student Panel

View Students | View Students | View Results | Teacher Profile | Login

Angular Service Example

ID	First Name	Last Name	Branch	DOB	Gender
std101	Preety	Tiwary	CSE	29/02/1988	Female
std102	Anurag	Mohanty	ETC	23/05/1989	Male
std103	Priyanka	Dewangan	CSE	24/07/1992	Female
std104	Hina	Sharma	ETC	19/08/1990	Female
std105	Sambit	Satapathy	CSE	12/94/1991	Male

- item1
- item2
- item3
- item4

Add an item: [Add to parent's list]

Date: 12:00:00 AM GMT+5

Dependency Injection

In software engineering, dependency injection is a technique whereby one object supplies the dependencies of another object. In other words, we can say that it is a coding pattern in which classes receives their dependencies from external sources rather than creating them itself.

Dependency Injection is the heart of Angular Applications. The Dependency Injection in Angular is a combination of two terms i.e., Dependency and Injection.

Dependency: Dependency is an object or service that is going to be used by another object.
Injections: It is a process of passing the dependency object to the dependent object. It creates a new instance of the class along with its required dependencies.

In the above service example, we are using a studentservice in a component and have not created an instance of the service rather we declared a private field **`stdService`** of type studentservice.

Reactive Programming

It deals with a programming paradigm dealing with data streams and the propagation of changes. Data streams may be static or dynamic. Example of static data stream is an array or collection of data, with some initial quantity which will not change. Whereas dynamic data streams are event emitters. Event emitters emit the data whenever the event happens.

Reactive programming enables the data stream to be emitted from one source called **Observable** and the emitted data stream to be caught by other sources called **Observer** through a process called **subscription**. This pattern simplifies change detection and necessary updating in the context of the programming. **RxJs** is a JavaScript Library which enables reactive programming in JavaScript. Angular uses RxJs library extensively to transfer data between components, HTTP client, Router and Reactive forms.

Observables

Observables are data sources, and they may be static or dynamic. **Rxjs** provides lot of method to create Observable from common JavaScript Objects. E.g.

1. **of** Emit any number of values in a sequence and finally emit a complete notification. E.g.

   ```
   const numbers$ = of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
   ```

   numbers$ is an Observable object, which when subscribed will emit 1 to 10 in sequence.

2. **range** Emit a range f number in sequence.

   ```
   const numbers$ = range(1,10)
   ```

3. **from** Emit array, or iterable

   ```
   const numbers$ = from([1,2,3,4,5,6,7,8,9,10]);
   ```

4. **ajax** fetch a url through AJAX and then emit the response

   ```
   const api$ = ajax({ url: 'https://httpbin.org/delay/1',
   method: 'POST', headers: { 'Content-Type': 'application/text'
   }, body: "Hello" });
   ```

5. **fromEvent** Listen to an HTML element's event and then emit the event and its property whenever the listened event fires.

   ```
   const clickEvent$ =
   fromEvent(document.getElementById('counter'), 'click');
   ```

Subscribing to the observable

Every Observable object will have a method, subscribe for the subscription process.
Observer needs to implement three callback function to subscribe to the Observable object, which are:

- **next** – Receive and process the value emitted from the Observable
- **error** – Error handling callback
- **complete** – Callback function called when all data from Observable are emitted.

Once the three callback functions are defined, Observable's subscribe method has to be called:

```
const numbers$ = from([1,2,3,4,5,6,7,8,9,10]);
// observer
const observer = {
   next: (num: number) => {      this.numbers.push(num);
                              this.val1 += num },
      error: (err: any) => console.log(err),
      complete: () => console.log("Observation completed")
};
numbers$.subscribe(observer);
```

Operations

Rxjs library provides some of the operators to process the data stream. Some of the important operators are:

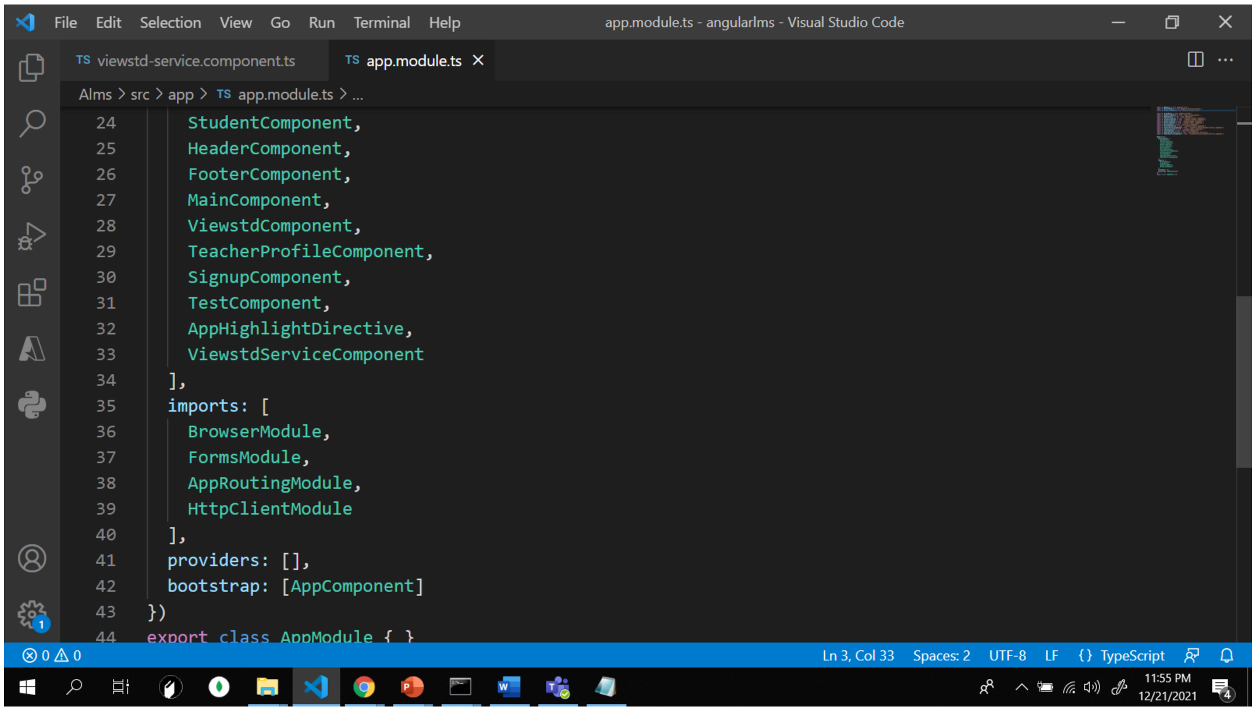
1. filter   Enable to filter the data stream using callback function.
2. map   Enables to map the data stream using callback function and to change the data stream itself.
3. Pipe

Communication with backend services using HTTP

Most front-end applications need to communicate with a server over the HTTP protocol, to download or upload data and access other back-end services. Angular provides a client HTTP API for Angular applications, the **HttpClient** service class in @angular/common/http.

Before you can use HttpClient, you need to import the Angular HttpClientModule in the app.module.ts. HttpClient service is available inside the HttpClientModule module, which is available inside the @angular/common/http package.

Include HttpClientModule in imports array of @NgModule decorator of app.module.ts file



File Edit Selection View Go Run Terminal Help
app.module.ts - angularlms - Visual Studio Code
viewstd-service.component.ts
app.module.ts
Alms > src > app > app.module.ts > ...
Ln 3, Col 33 Spaces: 2 UTF-8 LF {} TypeScript
11:55 PM
12/21/2021

```
    StudentComponent,
    HeaderComponent,
    FooterComponent,
    MainComponent,
    ViewstdComponent,
    TeacherProfileComponent,
    SignupComponent,
    TestComponent,
    AppHighlightDirective,
    ViewstdServiceComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```