

G. PULLA REDDY ENGINEERING COLLEGE (AUTONOMOUS): KURNOOL

DEPARTMENT: CSE

SUBJECT: ANGULAR LAB

CLASS: IV-VII SEM CSE-A, B, C, CST

FACULTIES: Mr. K. Bala Chowdappa, Mr. P. Rama Rao,

Mrs. D.L.N. Prasunna, Mrs. T. Swati

ANGULAR LAB (AR(P))								
VII Semester : Common for CSE, CST, CSE(AIML) & CSE(DS)					Scheme : 2020			
Course Code	Category	Hours/Week			Credits	Maximum Marks		
SCCS04	SC	L	T	P	C	Continuous Internal Assessment	End Exam	TOTAL
		0	0	4	2	40	60	100
Sessional Exam Duration 2 Hrs					End Exam Duration: 3 Hrs			
Course Outcomes : At the end of the course the student will be able to								
CO1: Understand the Angular and its working								
CO2: Implementing components and templates								
CO3: create single page and custom route applications								
CO4: Build applications that can get data from server								
CO5: Implement available and create user defined libraries								
List of Experiments								
1. Knowing the Editor								
2. Implementing components								
3. Implementing Templates								
4. Creating routing applications								
5. Displaying a list								
6. Adding Services								
7. Adding Navigation								
8. Getting data from a Server								
9. Using Published Libraries								
10. Creating User Defined Libraries								

EXPERIMENT 1: Knowing the Editor

AIM: Knowing about the Installation, Framework, Services and Libraries that are required for developing angular applications

PROCEDURE:

Introduction to the Angular

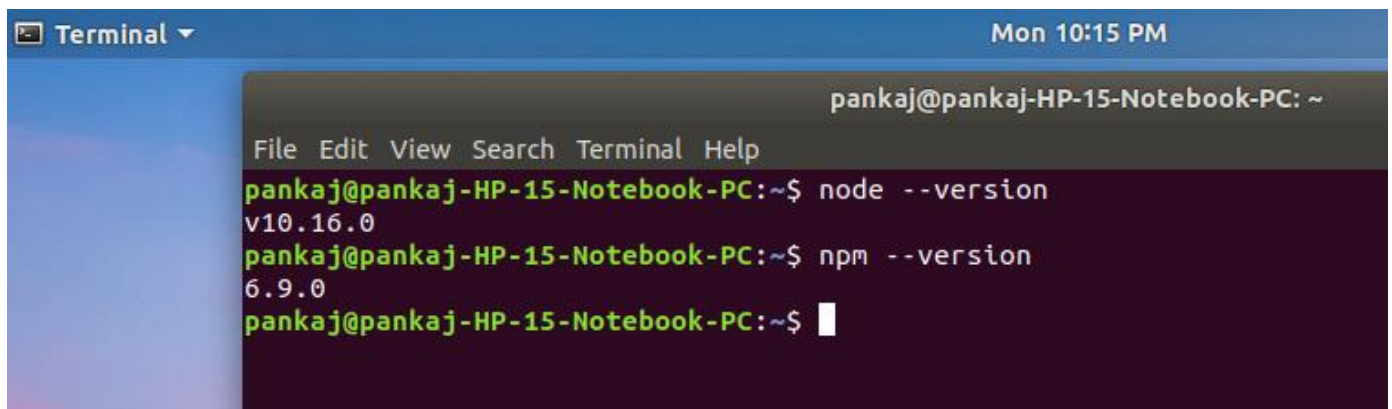
Angular is a front-end framework which is used to create web applications. It uses typescript by default for creating logics and methods for a class but the browser doesn't know typescript.

Angular CLI is a tool that does all these things for you in some simple commands. Angular CLI uses webpack behind to do all this process.

Note: Please make sure you have installed node and npm in your system. You can check your node version and npm version by using the following command:

node --version

npm --version

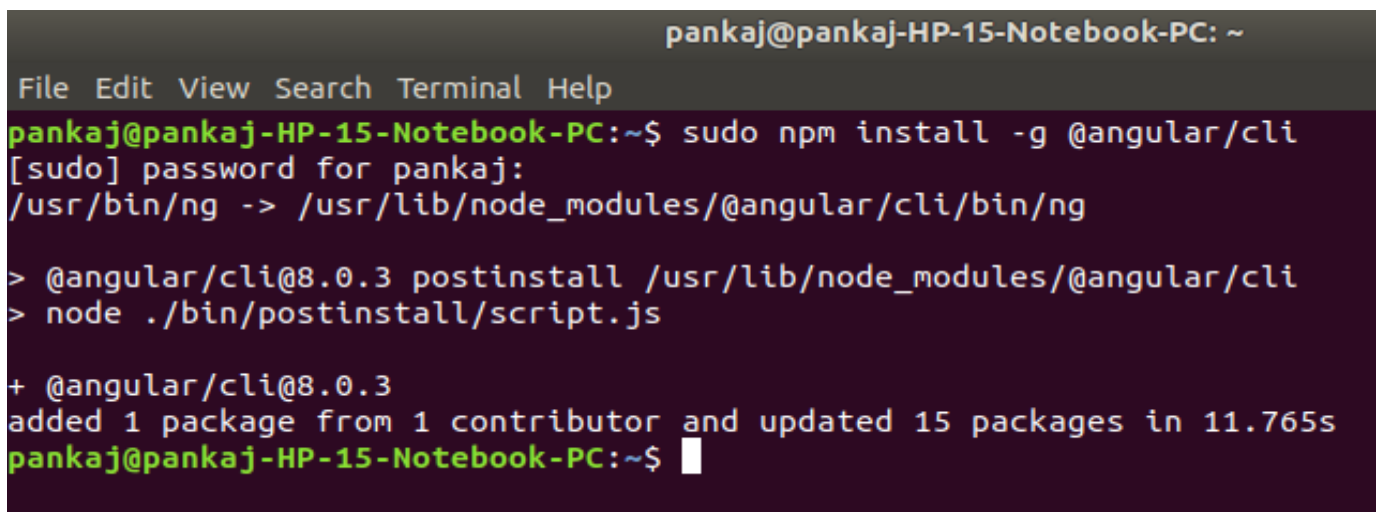
A terminal window titled 'Terminal' with a dropdown arrow and a clock showing 'Mon 10:15 PM'. The prompt is 'pankaj@pankaj-HP-15-Notebook-PC: ~'. The menu bar shows 'File Edit View Search Terminal Help'. The user enters 'node --version' and the output is 'v10.16.0'. Then the user enters 'npm --version' and the output is '6.9.0'. The prompt is now 'pankaj@pankaj-HP-15-Notebook-PC:~\$' with a cursor.

```
pankaj@pankaj-HP-15-Notebook-PC:~$ node --version
v10.16.0
pankaj@pankaj-HP-15-Notebook-PC:~$ npm --version
6.9.0
pankaj@pankaj-HP-15-Notebook-PC:~$
```

Steps to create your first application using angular CLI:

Step 1: Install angular cli

npm install -g @angular/cli

A terminal window with the same title and clock as the previous one. The prompt is 'pankaj@pankaj-HP-15-Notebook-PC: ~'. The menu bar shows 'File Edit View Search Terminal Help'. The user enters 'sudo npm install -g @angular/cli'. The terminal shows '[sudo] password for pankaj:' followed by the installation path '/usr/bin/ng -> /usr/lib/node_modules/@angular/cli/bin/ng'. Then it shows the postinstall script running: '> @angular/cli@8.0.3 postinstall /usr/lib/node_modules/@angular/cli' and '> node ./bin/postinstall/script.js'. Finally, it shows '+ @angular/cli@8.0.3' and 'added 1 package from 1 contributor and updated 15 packages in 11.765s'. The prompt is now 'pankaj@pankaj-HP-15-Notebook-PC:~\$' with a cursor.

```
pankaj@pankaj-HP-15-Notebook-PC:~$ sudo npm install -g @angular/cli
[sudo] password for pankaj:
/usr/bin/ng -> /usr/lib/node_modules/@angular/cli/bin/ng
> @angular/cli@8.0.3 postinstall /usr/lib/node_modules/@angular/cli
> node ./bin/postinstall/script.js
+ @angular/cli@8.0.3
added 1 package from 1 contributor and updated 15 packages in 11.765s
pankaj@pankaj-HP-15-Notebook-PC:~$
```

Step 2: Create new project by this command

Choose yes for routing option and, CSS or SCSS.

ng new myNewApp

```

pankaj@pankaj-HP-15-Notebook-PC: ~
File Edit View Search Terminal Help
pankaj@pankaj-HP-15-Notebook-PC:~$ ng new myNewApp
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS [ http://sass-lang.com/documentation/file
_SASS_REFERENCE.html#syntax ]
CREATE myNewApp/README.md (1025 bytes)
CREATE myNewApp/.editorconfig (246 bytes)
CREATE myNewApp/.gitignore (629 bytes)
CREATE myNewApp/angular.json (3529 bytes)
CREATE myNewApp/package.json (1283 bytes)
CREATE myNewApp/tsconfig.json (470 bytes)
CREATE myNewApp/tslint.json (1985 bytes)
CREATE myNewApp/browserslist (429 bytes)
CREATE myNewApp/karma.conf.js (1020 bytes)
CREATE myNewApp/tsconfig.app.json (210 bytes)
CREATE myNewApp/tsconfig.spec.json (270 bytes)
CREATE myNewApp/src/favicon.ico (5430 bytes)
CREATE myNewApp/src/index.html (295 bytes)
CREATE myNewApp/src/main.ts (372 bytes)
CREATE myNewApp/src/polyfills.ts (2838 bytes)
CREATE myNewApp/src/styles.scss (80 bytes)
CREATE myNewApp/src/test.ts (642 bytes)
CREATE myNewApp/src/assets/.gitkeep (0 bytes)
CREATE myNewApp/src/environments/environment.prod.ts (51 bytes)
CREATE myNewApp/src/environments/environment.ts (662 bytes)
CREATE myNewApp/src/app/app-routing.module.ts (245 bytes)
CREATE myNewApp/src/app/app.module.ts (393 bytes)
CREATE myNewApp/src/app/app.component.scss (0 bytes)
CREATE myNewApp/src/app/app.component.html (1152 bytes)
CREATE myNewApp/src/app/app.component.spec.ts (1101 bytes)
CREATE myNewApp/src/app/app.component.ts (213 bytes)
CREATE myNewApp/e2e/protractor.conf.js (810 bytes)
CREATE myNewApp/e2e/tsconfig.json (214 bytes)
CREATE myNewApp/e2e/src/app.e2e-spec.ts (637 bytes)
CREATE myNewApp/e2e/src/app.po.ts (251 bytes)
CREATE myNewApp/src/styles.scss (80 bytes)
CREATE myNewApp/src/test.ts (642 bytes)
CREATE myNewApp/src/assets/.gitkeep (0 bytes)
CREATE myNewApp/src/environments/environment.prod.ts (51 bytes)
CREATE myNewApp/src/environments/environment.ts (662 bytes)
CREATE myNewApp/src/app/app-routing.module.ts (245 bytes)
CREATE myNewApp/src/app/app.module.ts (393 bytes)
CREATE myNewApp/src/app/app.component.scss (0 bytes)
CREATE myNewApp/src/app/app.component.html (1152 bytes)
CREATE myNewApp/src/app/app.component.spec.ts (1101 bytes)
CREATE myNewApp/src/app/app.component.ts (213 bytes)
CREATE myNewApp/e2e/protractor.conf.js (810 bytes)
CREATE myNewApp/e2e/tsconfig.json (214 bytes)
CREATE myNewApp/e2e/src/app.e2e-spec.ts (637 bytes)
CREATE myNewApp/e2e/src/app.po.ts (251 bytes)

> core-js@2.6.9 postinstall /home/pankaj/myNewApp/node_modules/babel-runtime/node_modules/core-js
> node scripts/postinstall || echo "ignore"

> core-js@2.6.9 postinstall /home/pankaj/myNewApp/node_modules/karma/node_modules/core-js
> node scripts/postinstall || echo "ignore"

> @angular/cli@8.0.3 postinstall /home/pankaj/myNewApp/node_modules/@angular/cli
> node ./bin/postinstall/script.js

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

added 1015 packages from 1041 contributors and audited 19005 packages in 95.797s
found 0 vulnerabilities

Successfully initialized git.
pankaj@pankaj-HP-15-Notebook-PC:~$

```

Step 3: Go to your project directory

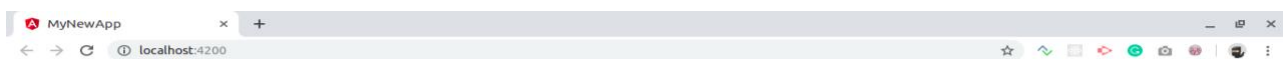
```
cd myNewApp
```

```
pankaj@pankaj-HP-15-Notebook-PC: ~/myNewApp
File Edit View Search Terminal Help
pankaj@pankaj-HP-15-Notebook-PC:~$ cd myNewApp/
pankaj@pankaj-HP-15-Notebook-PC:~/myNewApp$
```

Step 4: Run server and see your application in action

```
ng serve -o --poll=2000
```

```
pankaj@pankaj-HP-15-Notebook-PC: ~/myNewApp
File Edit View Search Terminal Help
pankaj@pankaj-HP-15-Notebook-PC:~/myNewApp$ ng serve -o --poll=2000
93% after chunk asset optimization SourceMapDevToolPlugin polyfills.js generate SourceMa
Date: 2019-06-24T16:58:12.903Z
Hash: 55a034f4077a23b171a3
Time: 24279ms
chunk {main} main.js, main.js.map (main) 11.4 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 248 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.6 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.94 MB [initial] [rendered]
** Angular Live Development Server is listening on localhost:4200, open your browser on h
ttp://localhost:4200/ **
i [wdm]: Compiled successfully.
```



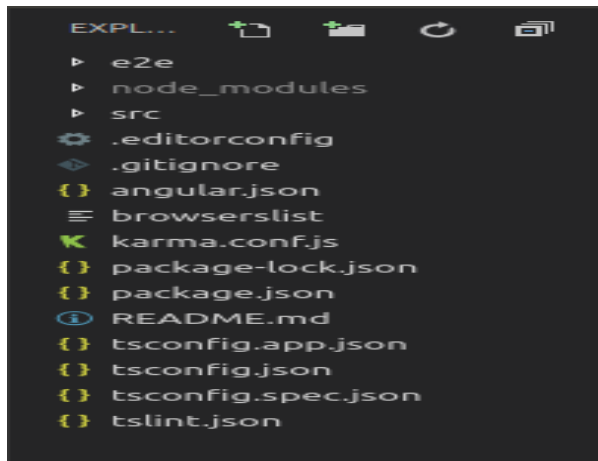
Welcome to myNewApp!



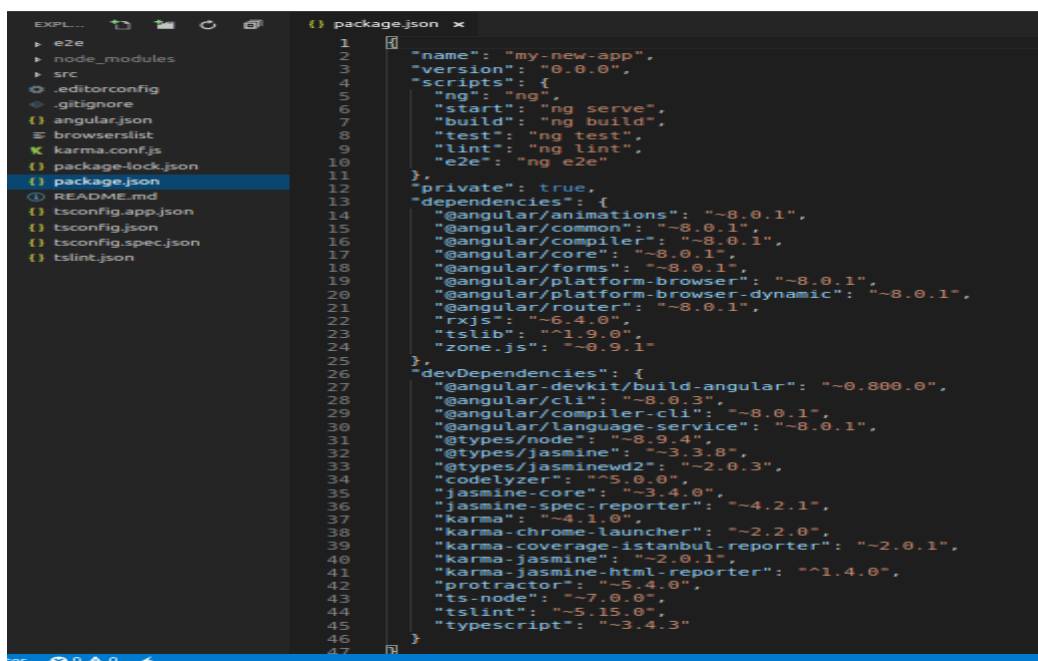
Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

Introduction to directory structure:



- **e2e** It contains the code related to automated testing purpose. For example, if on a certain page you are calling a REST API then what should be the return status code, whether it is acceptable or not etc.
- **node_modules** It saves all the dev dependencies (used only at development time) and dependencies (used for development as well as needed in production time), any new dependency when added to project it is automatically saved to this folder.
- **src** This directory contains all of our work related to project i.e. creating components, creating services, adding CSS to the respective page, etc.
- **package.json** This file stores the information about the libraries added and used in the project with their specified version installed. Whenever a new library is added to the project it's name and version is added to the dependencies in package.json.



Other files: As a beginner you don't need these files at this time, don't bother about that. These all are used for editor configurations and information needed at compile time. The built-in webpack in angular CLI manages all for you.

Inside src folder:

- **index.html** This is the entry point for the application, **app-root** tag is the entry point of the application on this single page application, From this page angular will add or remove the content from the DOM or will add new content to the DOM. Base **href="/"** is important for routing purposes.
- **html**

```
<!DOCTYPE HTML>

<html lang="en">

  <head>

    <meta charset="utf-8">

    <title>MyNewApp</title>

    <basehref="/">

    <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>

  </body>

</html>
```

- **style.scss** This file is the global stylesheet you can add that CSS classes or selectors which are common to many components, for example, you can import custom fonts, import bootstrap.css, etc.
- **assets** It contains the js images, fonts, icons and many other files for your project.

Inside app folder:

- **app.module.ts** An angular project is composite of so many other modules in order to create an application you have to create a root module for your application in the hierarchy. This app.module.ts file is that. If you want to add more modules at the root level, you can add.
- **declarations** It is the reference of the array to store its components. The app component is the default component that is generated when a project is created. You have to add all your component's reference to this array to make them available in the project.

- **imports** If you want to add any module whether angular or you have to add it to imports array to make them available in the whole project.
- **providers** If you will create any service for your application then you will inject it into your project through this provider array. Service injected to a module is available to it and it's child module in the project hierarchy.
- **bootstrap** This has reference to the default component created, i.e., AppComponent
- **app.component.html** Edit this file to make changes to the page. You can edit this file as an HTML file. Work directly with div or any other tag used inside body tags, these are components and do not add **html head body** tags.
 - html

```
<h1>
  Hello world
</h1>

<div>
  <p>
    This is my First Angular app.
  </p>
</div>
```

- **app.component.spec.ts** These are automatically generated files which contain unit tests for source component.
- **app.component.ts** You can do the processing of the HTML structure in the .ts file. The processing will include activities such as connecting to the database, interacting with other components, routing, services, etc.
- **app.component.scss** Here you can add CSS for your component. You can write scss which further compiled to CSS by a transpiler.

More commands that you will need while working on the project:

ng generate component component_name

ng generate service service_name

ng generate directive directive_name

EXPERIMENT 2: Implementing Components

AIM: Creating Components and implementing them using Angular

PROCEDURE:

Components:

Components are the most basic UI building block of an Angular app. An Angular app contains a tree of Angular components.

Angular components are a subset of directives, always associated with a template. Unlike other directives, only one component can be instantiated for a given element in a template.

A component must belong to an NgModule in order for it to be available to another component or application. To make it a member of an NgModule, list it in the declarations field of the NgModule metadata.

Step 1: Create a component with the name

- form

Step 2: insrc/app/app.component.html

```
<app-form></app-form>
```

Step 3: insrc/form/form.component.html

```
<!DOCTYPEhtml>
<html lang="en">
<head>
<title>Registration details</title>
</head>

<body background="blue">
<form>
<fieldset>
<legend>Personal Details</legend>
<p>
<label>
Salutation
<br/>
<select name="salutation">
<option>--None--</option>
<option>Mr.</option>
<option>Ms.</option>
<option>Mrs.</option>
```



```

<option>Dr.</option>
<option>Prof.</option>
</select>
</label>
</p>

<p><label>First name: <inputname="firstName"/></label></p>

<p><label>Last name: <inputname="lastName"/></label></p>

<p> Gender :
<label><inputtype="radio" name="gender" value="male"/> Male </label>
<label><inputtype="radio" name="gender" value="female"/> Female </label>
</p>

<p><label>Email:<inputtype="email" name="email"/></label></p>

<p><label>Date of Birth:<inputtype="date" name="birthDate"></label>
</p>

<p><label> Address : <br/><textarea name="address" cols="30" rows="3"></textarea></label></p>

<p><button type="submit">Submit</button></p>

</fieldset>
</form>
</body>
</html>

```

OUTPUT:

Personal Details

Salutation
Mrs.

First name: D

Last name: PRASUNNA

Gender : ☐ Male ☒ Female

Email: prasunna.cse@gprec.ac.in

Date of Birth: 14-06-1985

Address :
Hyderabad

Submit

EXPERIMENT 3: Implementing Templates

AIM: To implement templates using Angular CLI

PROCEDURE:

Templates:

A template is a form of HTML that tells Angular how to render the component.

Views are typically organized hierarchically, allowing you to modify or show and hide entire UI sections or pages as a unit. The template immediately associated with a component defines that component's *host view*. The component can also define a *view hierarchy*, which contains *embedded views*, hosted by other components

Step-1: app.component.html

```
<divclass="container-fluid">
  <h1> Registration Form</h1>
  <form[formGroup]="registrationform"(ngSubmit)="loginrequest()">
    <divclass="form-group">
      <label> Name</label>
      <inputformControlName="username"type="text"class="form-control">
    </div>
    <div>
      <label> Password</label>
      <inputformControlName="password"type="password"class="form-control">
    </div>
    <div>
      <label> Confirm Password</label>
      <inputformControlName="cfnpassword"type="password"class="form-control">
    </div>
    <buttonclass="btnbtn-primary"type="submit">Submit</button>
  </form>
</div>
```

Step 2: app.component.ts

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

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

```

}))
export class AppComponent {
  title = "exp3";
  registrationform = new FormGroup({
    username: new FormControl('Admin'),
    password: new FormControl(""),
    cfnpassword: new FormControl("")
  });
  loginrequest() {
    console.log(this.registrationform.value);
  }
}

```

Step 3: app.module.ts

```

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';

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

```

Step 4: app-routing.module.ts

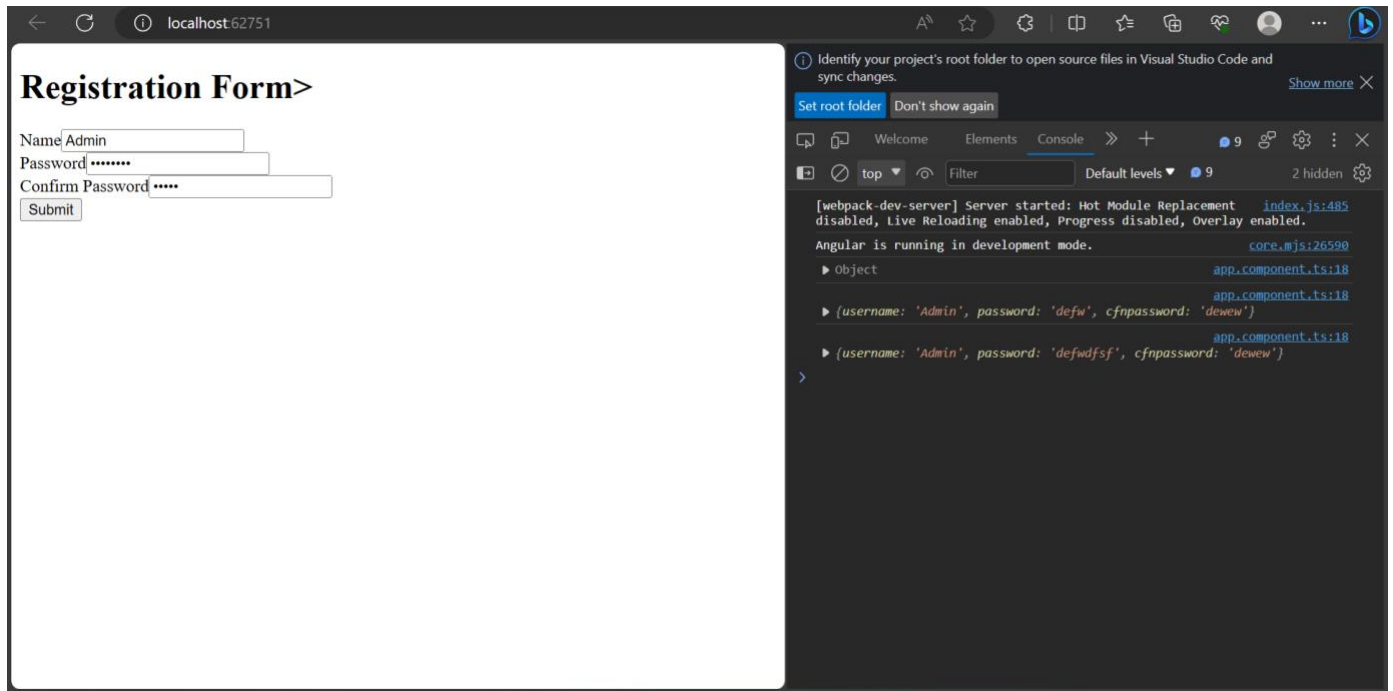
```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [];

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

```

OUTPUT:

EXPERIMENT 4: Implementing Routing

AIM: To implement Routing among components using Angular CLI

PROCEDURE:

Step 1: Let's create two components **Faculty** and **Student**

Step 2: Now, we need to do routing between Faculty and Student Components.

In app-routing.module.ts

Declare the path under an array constroutes:Route

```
const routes: Routes = [  
  {path:'s',component:StudentComponent},  
  {path:'f',component:FacultyComponent}  
];
```

There is a const routes defined which is of type Routes. It is an array which holds all the routes we need in our project. The const routes is given to the RouterModule as shown in @NgModule. To display the routing details to the user, we need to add <router-outlet> directive where we want the view to be displayed.

Step 3: app.module.ts

```
@NgModule({  
  declarations: [  
    AppComponent,  
  
    myroutes  
  ],
```

Step 4: app.component.html

```
<router-outlet></router-outlet>  
  
<nav>
```

```
<a routerLink="/f" style="padding: 10px;">FACULTY </a>
<a routerLink="/s" style="padding: 10px;">STUDENT</a>
</nav>
```

a single page application (SPA) does not have different pages to link to. Instead, it has different *views* to display to the user. To allow a user to navigate and change the view, you will want to use the RouterLink directive instead of href:

Step 5: app.component.css

```
h1{
  text-align: center;
  color: #ce2b2b;
}
nav a {
  padding: 15px 20px;
  text-decoration: none;
  margin-top: 10px;
  display: inline-block;
  background-color:aqua;
  border-radius: 4px;
  margin-left: 20px;
}

nav a:visited, a:link {
  color: #3643b6;
}
nav a:hover {
  color: #fff;
  background-color: #3643b6;
}
nav a.active {
  color: #ce2b2b;
  background-color: #49ce0c;
}
```

Step 6: Now, in **faculty.component.html** let's write a simple html code for faculty login page.similarly, in **student.component.html**

faculty.component.html

```
html>
<body>
```

```

<h1> Faculty Login Form </h1>
<form>
  <divclass="container">
    <label>Username : </label>
    <inputtype="text"placeholder="Enter Username"name="username"required>
    <label>Password : </label>
    <inputtype="password"placeholder="Enter Password"name="password"required>
    <buttontype="submit">Login</button>
    <inputtype="checkbox"checked="checked"> Remember me
    <buttontype="button"class="cancelbtn"> Cancel</button>
    Forgot <a href="#"> password? </a>
  </div>
</form>
</body>
</html>

```

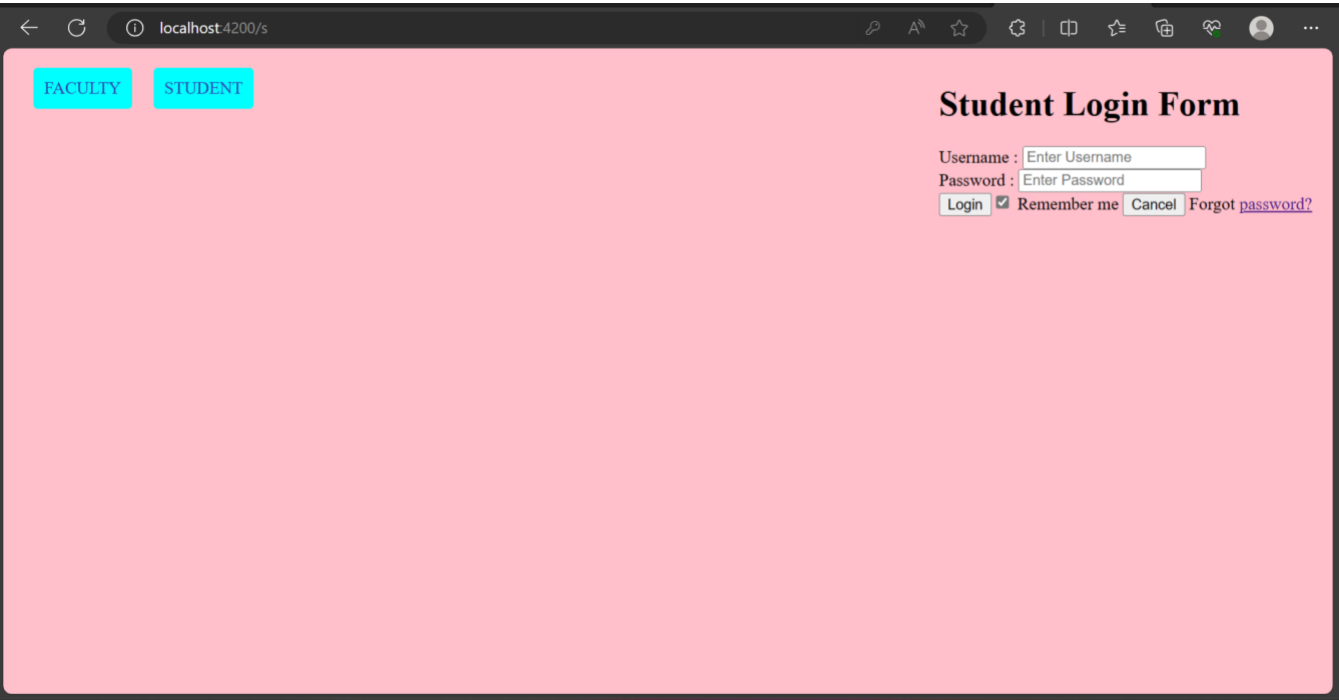
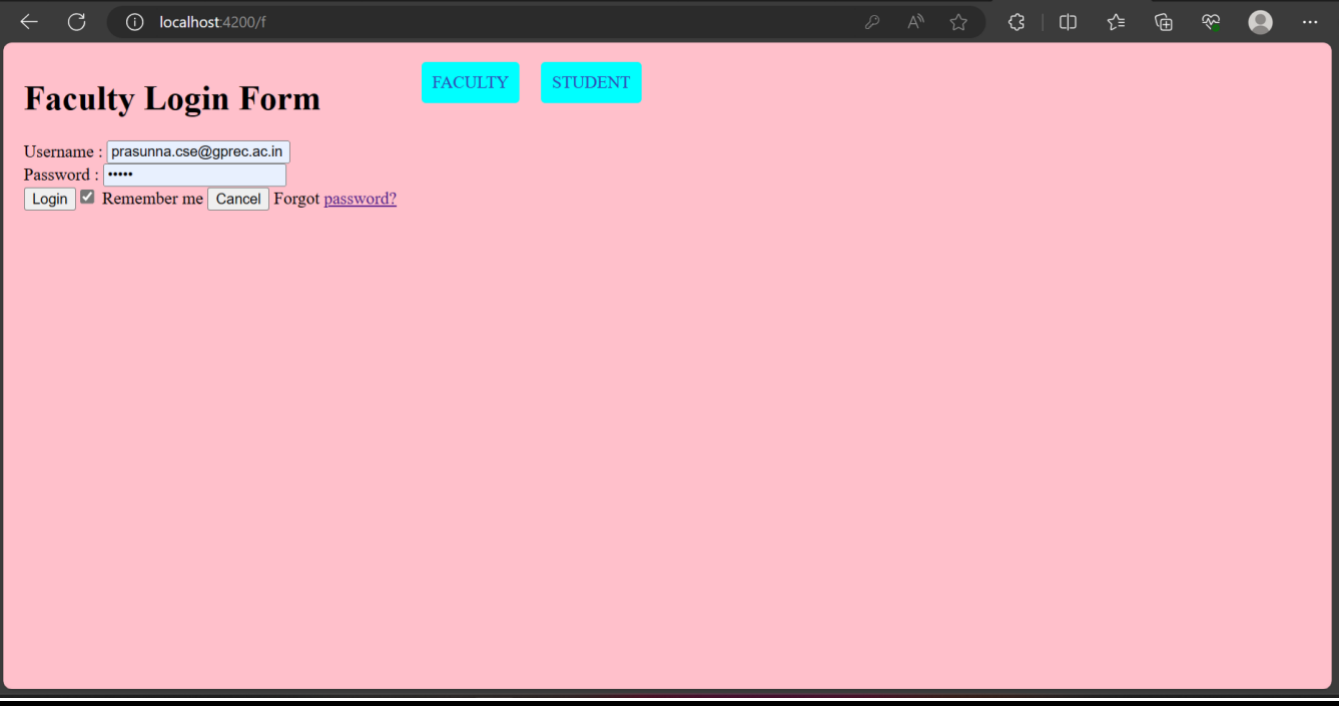
student.component.html

```

<html>
<body>
  <tablealign="right">
    <tr><td>
      <h1> Student Login Form </h1>
      <form>
        <divclass="container">
          <label>Username : </label>
          <inputtype="text"placeholder="Enter Username"name="username"required> <br>
          <label>Password : </label>
          <inputtype="password"placeholder="Enter
Password"name="password"required><br>
          <buttontype="submit">Login</button>
          <inputtype="checkbox"checked="checked"> Remember me
          <buttontype="button"class="cancelbtn"> Cancel</button>
          Forgot <a href="#"> password? </a>
        </div>
      </form>
    </td></tr></table>
</body>
</html>

```

OUTPUT:



EXPERIMENT 5: Angular Lists**AIM:** Implementing the list of Events/ Task in Angular**PROCEDURE:****Step 1:** Create app-routing.module.ts file

Src/app/ app-routing.module.ts

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

const routes: Routes = [];

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

Step 2: app.component.html

```
<h1>{{ title }}</h1>
<div><input type="text" placeholder="Enter New Task" #task/>
<br>
<br>
<button (click)="addtask(task.value)">Add New Task</button>
</div>

<ol *ngFor="let item of list" type="number">

  <li>{{ item.id+1 }} {{ item.name }} <button (click)="removeTask(item.id)">Remove</button></li>
</ol>
```

Step 3: app.component.css

```
h1 {
  font-size: 24px;
  color: #333;
  text-align: center;
}

/* Style the input and button container */
div {
```

```
text-align: center;
margin-top: 20px;
}

input[type="text"] {
width: 25%;
padding: 10px;
font-size: 16px;
border: 1px solid #ccc;
border-radius: 5px;
}

button {
padding: 10px 20px;
font-size: 16px;
background-color: #007bff;
color: #fff;
border: none;
border-radius: 5px;
cursor: pointer;
}

button:hover {
background-color: #0056b3;
}

/* Style the list of tasks */
ul {
list-style-type: none;
padding: 0;
}

li {
display: flex;
justify-content: space-between;
align-items: center;
padding: 10px 0;
}

/* Style the remove button */
libutton {
padding: 5px 10px;
font-size: 14px;
background-color: #ff0000;
color: #fff;
border: none;
border-radius: 5px;
cursor: pointer;
}

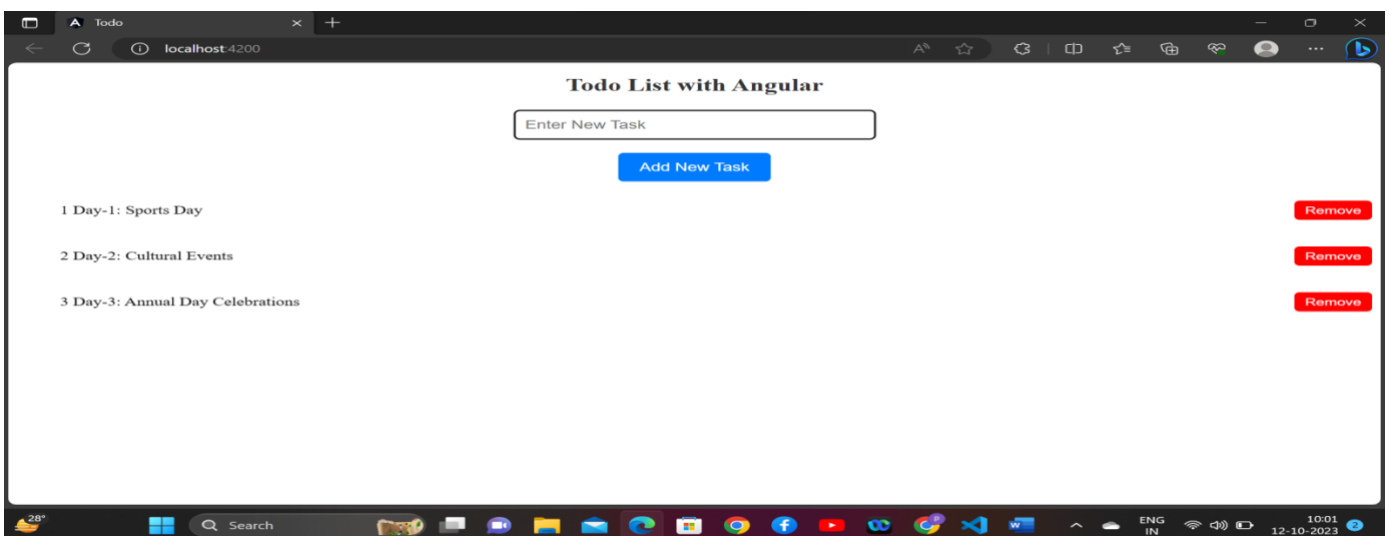
libutton:hover {
```

```
background-color: #cc0000;  
}
```

Step 4: app.component.ts

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title = 'Todo List with Angular';  
  list: any[] = [];  
  addtask(item: string)  
  {  
    this.list.push({ id: this.list.length, name: item })  
    console.warn(this.list);  
  }  
  removeTask(id: number)  
  {  
    console.warn(id)  
    this.list = this.list.filter(item => item.id !== id);  
  }  
}
```

OUTPUT:



EXPERIMENT 6: Angular Services

AIM: Implementing services using Dependency Injection in Angular.

PROCEDURE:

Step 1: Create two components with the names

- angular
- Java script

Step 2: Go to **src/app/angular/angular.component.html**

```
<divclass="container">

  <div><imgsrc="assets/angular.jpg"alt="Angular"style="width: 240px; height: 180px;"></div>
  <divstyle="text-align: left; padding: 2px 50px;">
    <h3>{{ title }} Course</h3>
  </div>

  <divstyle="text-align: left; padding: 2px 80px;">
    <button(click)="OnEnroll()"style="background-color: blue">Enroll</button>
  </div>
</div>
```

Step 3: Go to **src/app/angular/angular.component.ts**

```
import { Component } from '@angular/core';
import { EnrollService } from '../Services/enroll.service';

@Component({
  selector: 'app-angular',
  templateUrl: './angular.component.html',
  styleUrls: ['./angular.component.css'],
  providers: [EnrollService]
})
export class AngularComponent {
  title= "Angular"

  constructor(private enrollService: EnrollService)
  {
  }
}
```

```

OnEnroll()
{

    this.enrollService.OnEnrollClicked(this.title);
}

}

```

Step 4: Go to src/app/javascript/javascript.component.html

```

<div class="container">

    <div></div>
    <div style="text-align: left; padding: 2px 50px;">
        <h3>{{ title }} Course</h3>
    </div>

    <div style="text-align: left; padding: 2px 80px;">
        <button (click)="OnEnroll()" style="background-color: yellow">Enroll</button>
    </div>
</div>

```

Step 5: Go to src/app/javascript/javascript.component.ts

```

import { Component } from '@angular/core';
import { EnrollService } from '../Services/enroll.service';
@Component({
    selector: 'app-javascript',
    templateUrl: './javascript.component.html',
    styleUrls: ['./javascript.component.css'],
    providers: [EnrollService]
})
export class JavascriptComponent {
    title = "JavaScript";

    constructor(private enrollService: EnrollService)
    {

    }

    OnEnroll()
    {
        this.enrollService.OnEnrollClicked(this.title);
    }
}

```

Step 6: Create a folder with the name **Services** in **src/app/Services**

Step 7: Create a file with the name **enroll.service.ts** in **src/app/Services/enroll.service.ts**

```
export class EnrollService {
  OnEnrollClicked(title: string) {
    alert("Thanking for enrolling to '"+title+'course.');
  }
}
```

OUTPUT:

Personal Details

Salutation
Mrs.

First name: D

Last name: PRASUNNA

Gender : ☐ Male ☒ Female

Email: prasunna.cse@gprec.ac.in

Date of Birth: 14-06-1985

Address :
Hyderabad

Submit

localhost:4200 says
Thanking for enrolling to JavaScriptcourse.

OK

JavaScript Course

Enroll

Angular Course

Enroll

EXPERIMENT 7: Adding Navigations

AIM: Adding Navigation Bar and Links in Angular

PROCEDURE:

The `<mat-sidenav>`, an Angular Directive, is used to create a side navigation bar and main content panel with material design styling and animation capabilities.

- `<mat-sidenav-container>` - Represents the main container.
- `<mat-sidenav-content>` - Represents the content panel.
- `<mat-sidenav>` - Represents the side panel.
- Follow the following steps to update the Angular application

Step 1:

1. Create a project with a name *materialApp* as explained in the *Angular 6 - Project Setup* chapter.
2. Modify *app.module.ts*, *app.component.ts*, *app.component.css* and *app.component.html* as explained below. Keep rest of the files unchanged.
3. Compile and run the application to verify the result of the implemented logic.

app.module.ts.

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

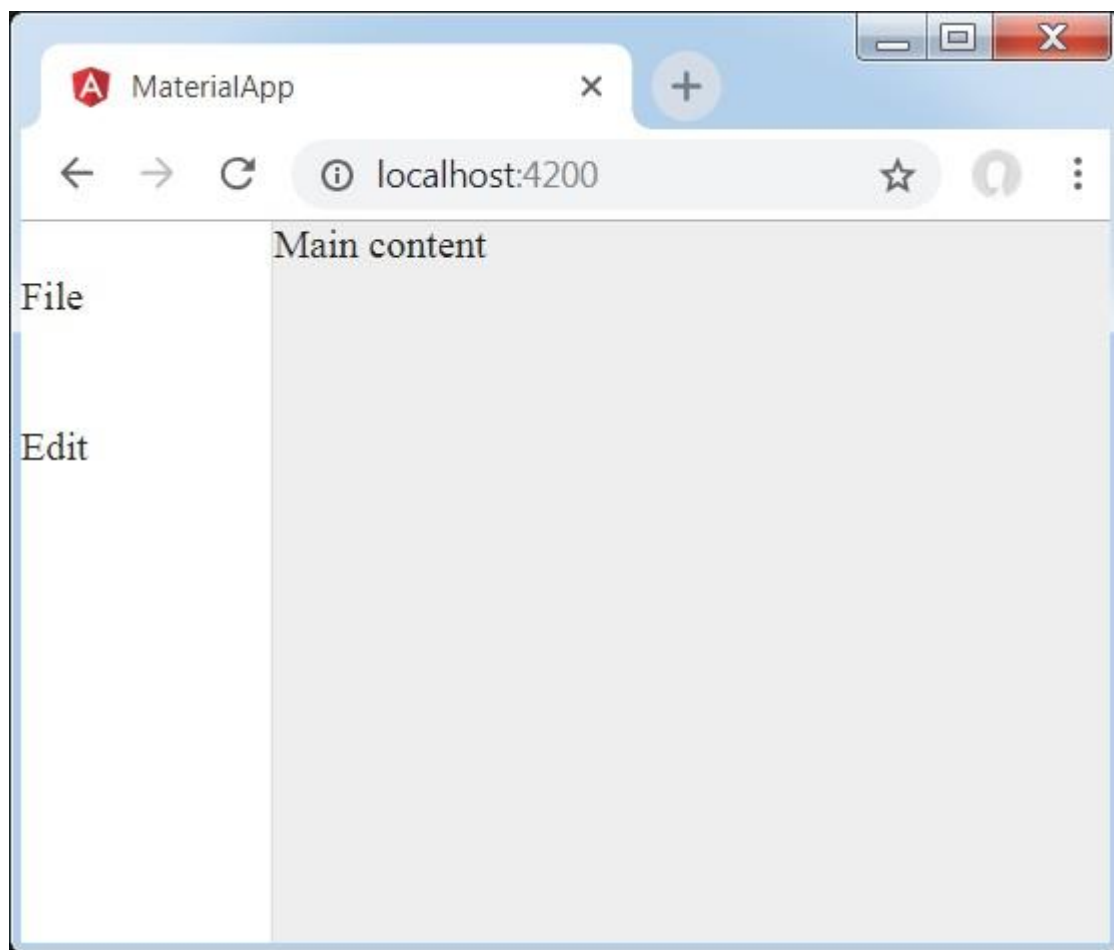
Step 2: app.component.css.

```
.tp-container {  
  position: absolute;  
  top:0;  
  bottom:0;  
  left:0;  
  right:0;  
  background:#eee;  
}  
.tp-section {  
  display: flex;  
  align-content:center;  
  align-items:center;  
  height:60px;  
  width:100px;  
}
```

Step 3: app.component.html.

```
<mat-sidenav-containerclass="tp-container">  
<mat-sidenavmode="side"opened>  
<sectionclass="tp-section">  
<span>File</span>  
</section>  
<sectionclass="tp-section">  
<span>Edit</span>  
</section>  
</mat-sidenav>  
<mat-sidenav-content>Main content</mat-sidenav-content>  
</mat-sidenav-container>
```


OUTPUT:



EXPERIMENT 8: Reading Data from Server

AIM: Accessing data from the server using HTTP in Angular

PROCEDURE: Using the HttpClient.get() method to fetch data from a server.

This asynchronous method sends an HTTP request, and returns an Observable that emits the requested data when the response is received.

The get(url, options) method takes two arguments; the string endpoint URL from which to fetch, and an *optional options* object to configure the request.

Important options include the observe and responseType properties.

The observe option specifies how much of the response to return

The responseType option specifies the desired format of the returned data

To better understand the observe and responseType option types, see below.

Use the options object to configure various other aspects of an outgoing request. In adding headers, for example, the service set the default headers using the headers option property.

Use the params property to configure a request with HTTP URL parameters, and the reportProgress option to listen for progress events when transferring large amounts of data.

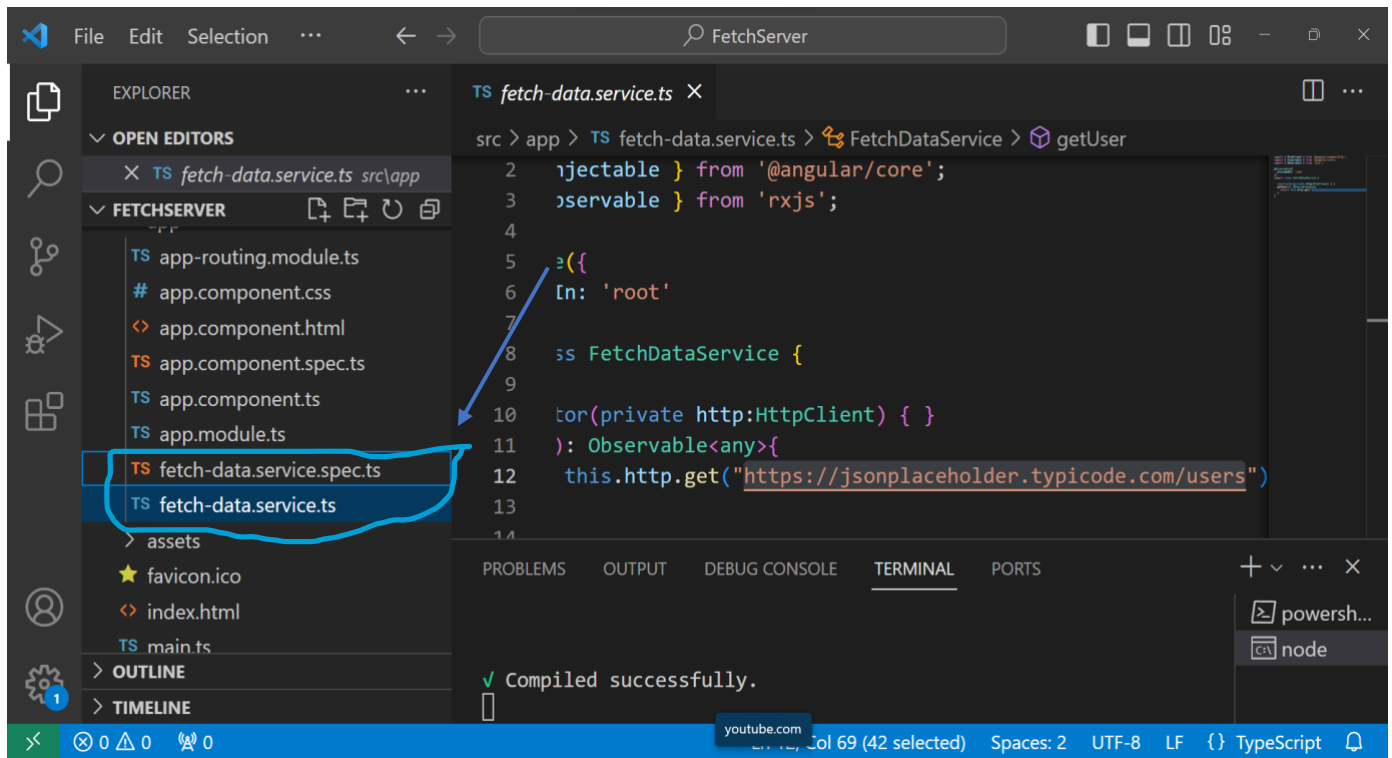
Applications often request JSON data from a server. In the ConfigService example, the app needs a configuration file on the server, config.json, that specifies resource URLs.
assets/config.json

```
content_copy{  
  "heroesUrl": "api/heroes",  
  "textfile": "assets/textfile.txt",  
  "date": "2020-01-29"  
}
```

To fetch this kind of data, the get() call needs the following options: {observe: 'body', responseType: 'json'}. *These are the default values for those options*, so most get() calls - and most of the following examples - do not pass the options object. Later sections show some of the additional option possibilities.

Step 1: Generate a service with the command `ng g s fetch-data`.

Then, two files will be created with `fetch-data.service.spec.ts` and `fetch-data.service.ts`



Step 2: fetch-data.service.ts

[this.http.get\("https://jsonplaceholder.typicode.com/users"\)](https://jsonplaceholder.typicode.com/users)

export class FetchDataService and get() request method. HttpClient. get() method is an asynchronous method that performs an HTTP get request in Angular applications and returns an Observable. And that Observable emits the requested data when the response is received from the server.

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

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

  constructor(private http:HttpClient) { }

  getUser(): Observable<any>{
    return this.http.get("https://jsonplaceholder.typicode.com/users");
  }
}
```

Step 3: app.component.ts

in app.component.ts call the method with **fetchService.getUser().subscribe()** method.

- The subscriber function defines how to obtain or generate values or messages to be published. To execute the observable you have created and begin receiving notifications, you call its subscribe()

method, passing an observer. This is a JavaScript object that defines the handlers for the notifications you receive.

```
import { Component } from '@angular/core';
import { FetchDataService } from './fetch-data.service';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Demo';
  userList:any[] = [];
  constructor(private fetchService: FetchDataService) {}
  ngOnInit(): void{
    this.fetchService.getUser().subscribe(user => this.userList = user);
  }
}
```

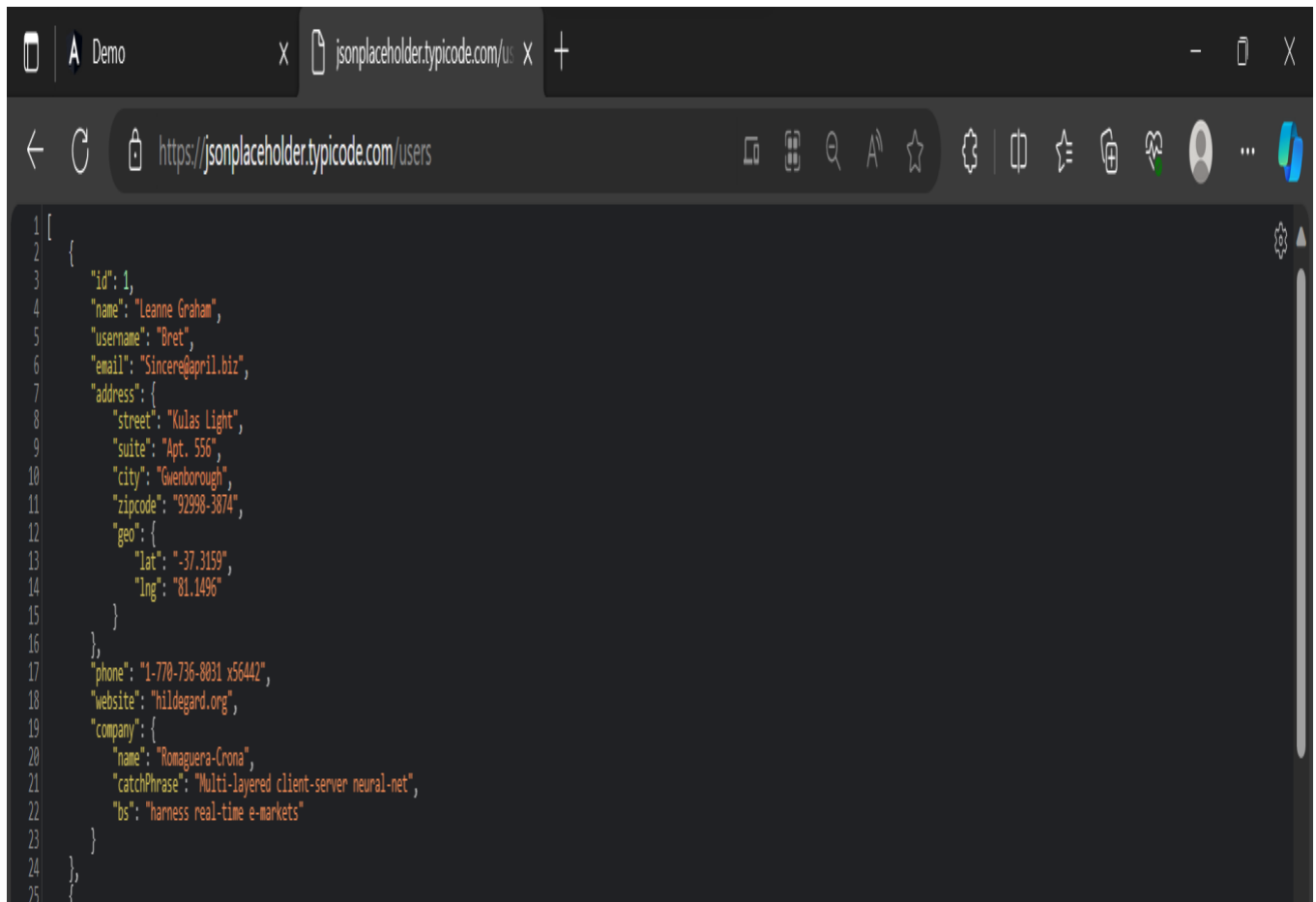
Step-4:

app.component.html

```
<table style="margin: 0 auto; border: 1px black solid;">
  <thead>
    <th style="border: 1px black solid; padding: 15px;">Name</th>
    <th style="border: 1px black solid; padding: 15px;">Id</th>
    <th style="border: 1px black solid; padding: 15px;">Email</th>
    <th style="border: 1px black solid; padding: 15px;">Street</th>
  </thead>
  <tr *ngFor="let user of userList">
    <td style="border: 1px black solid; padding: 15px;">{{ user.name }}</td>
    <td style="border: 1px black solid; padding: 15px;">{{ user.id }}</td>
    <td style="border: 1px black solid; padding: 15px;">{{ user.email }}</td>
    <td style="border: 1px black solid; padding: 25px;">{{ user.address.street }}</td>
  </tr>
</table>
```

<https://jsonplaceholder.typicode.com/users>

Though this link, we can access the data from the server.



```

1 [
2   {
3     "id": 1,
4     "name": "Leanne Graham",
5     "username": "Bret",
6     "email": "Sincere@april.biz",
7     "address": {
8       "street": "Kulas Light",
9       "suite": "Apt. 556",
10      "city": "Gwenborough",
11      "zipcode": "92998-3874",
12      "geo": {
13        "lat": "-37.3159",
14        "lng": "81.1496"
15      }
16    },
17    "phone": "1-770-736-8031 x56442",
18    "website": "hildegard.org",
19    "company": {
20      "name": "Romaguera-Crona",
21      "catchPhrase": "Multi-layered client-server neural-net",
22      "bs": "harness real-time e-markets"
23    }
24  },
25 ]

```

OUTPUT:



Name	Id	Email	Street
Leanne Graham	1	Sincere@april.biz	Kulas Light
Ervin Howell	2	Shanna@melissa.tv	Victor Plains
Clementine Bauch	3	Nathan@yesenia.net	Douglas Extension
Patricia Lebsack	4	Julianne.OConner@kory.org	Hoeger Mall
Chelsey Dietrich	5	Lucio_Hettinger@annie.ca	Skiles Walks
Mrs. Dennis Schulist	6	Karley_Dach@jasper.info	Norberto Crossing
Kurtis Weissnat	7	Telly.Hoeger@billy.biz	Rex Trail
Nicholas Runolfsson	8	Sherwood@rosamond.me	Ellsworth Summit
Glenna Reichert	9	Chaim_McDermott@dana.io	Dayna Park
Clementina DuBuque	10	Rey.Padberg@karina.biz	Kattie Turnpike

EXPERIMENT 9: Using Published Libraries

AIM: create an angular library and install the library in another project by npm install commands.

PROCEDURE:

Libraries in programming languages are collections of prewritten code that users can use to optimize tasks.

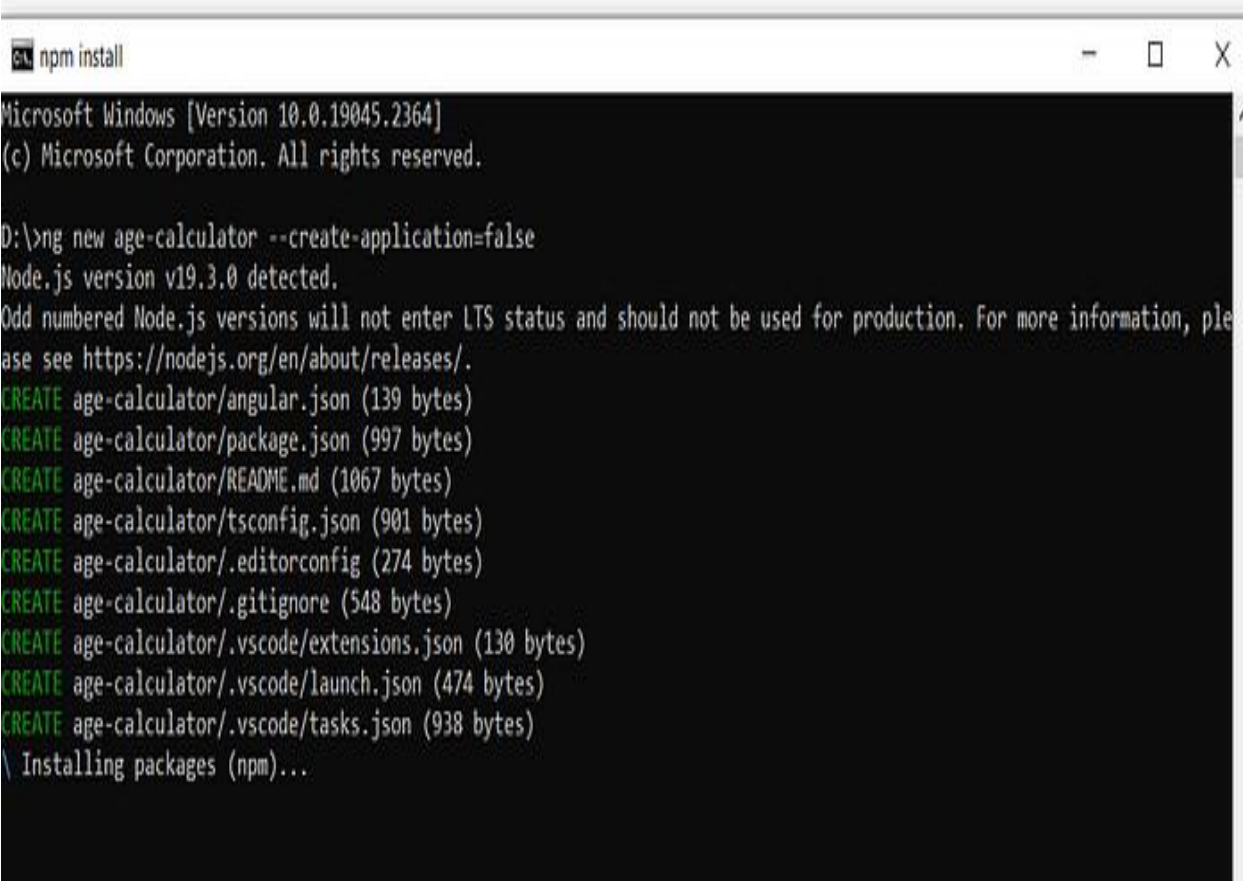
An Angular Library consists of

- Components,
- Modules
- Services

Steps involved in creating a library majorly are inlining all the templates, compiling it with ngc, and producing its build formats. However, this is a lot of manual tasks and ng-packagr has got us covered to do just that

Step 1:

ng new age-calculator --create-application=false



```
Microsoft Windows [Version 10.0.19045.2364]
(c) Microsoft Corporation. All rights reserved.

D:\>ng new age-calculator --create-application=false
Node.js version v19.3.0 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for production. For more information, please see https://nodejs.org/en/about/releases/.
CREATE age-calculator/angular.json (139 bytes)
CREATE age-calculator/package.json (997 bytes)
CREATE age-calculator/README.md (1067 bytes)
CREATE age-calculator/tsconfig.json (901 bytes)
CREATE age-calculator/.editorconfig (274 bytes)
CREATE age-calculator/.gitignore (548 bytes)
CREATE age-calculator/.vscode/extensions.json (130 bytes)
CREATE age-calculator/.vscode/launch.json (474 bytes)
CREATE age-calculator/.vscode/tasks.json (938 bytes)
\ Installing packages (npm)...
```

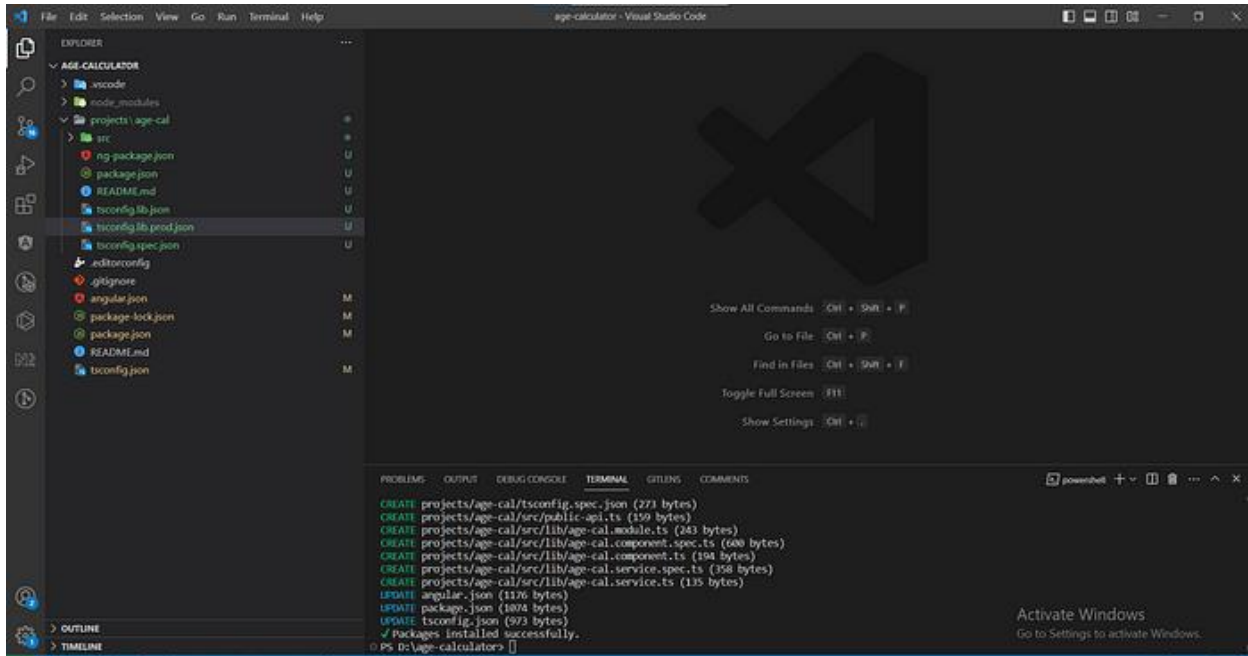
This would create a workspace for you with no src folder as you haven't created the application and just a workspace under which we can create our library using the command

Step 2:

```
ng generate library<library-name>
```

```
ng g library age-cal
```

This creates a projects folder with your lib folder inside it. Here you can start writing the feature you want for your library



Step 3:

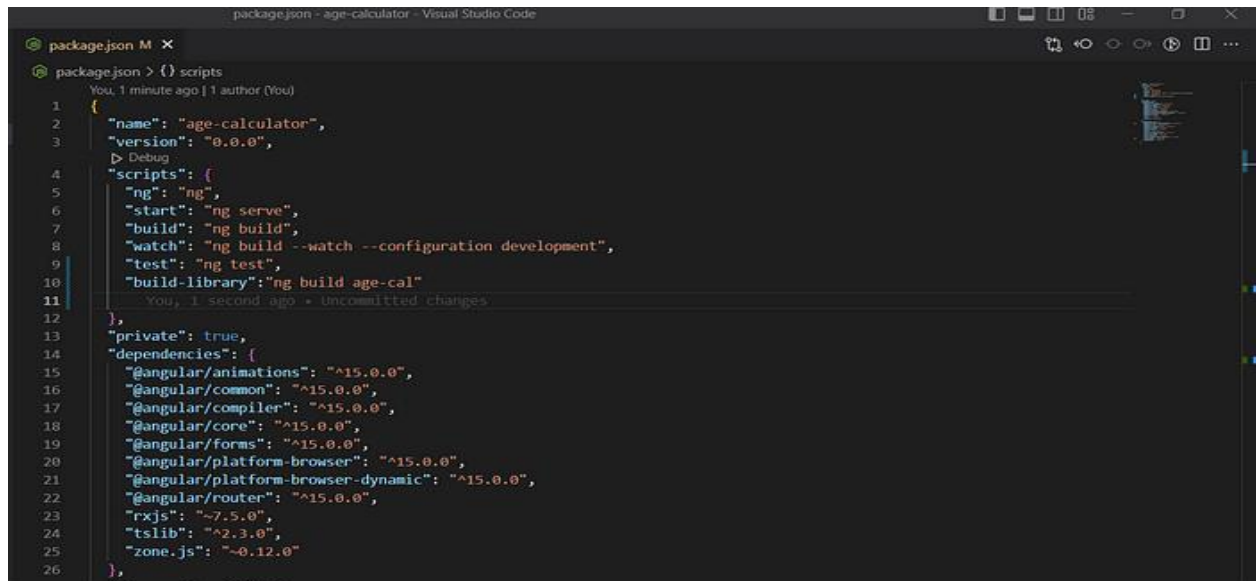
Now add our library to our root package.json as

```
"build-library": "ng build age-cal"
```

```
npm run build-library
```

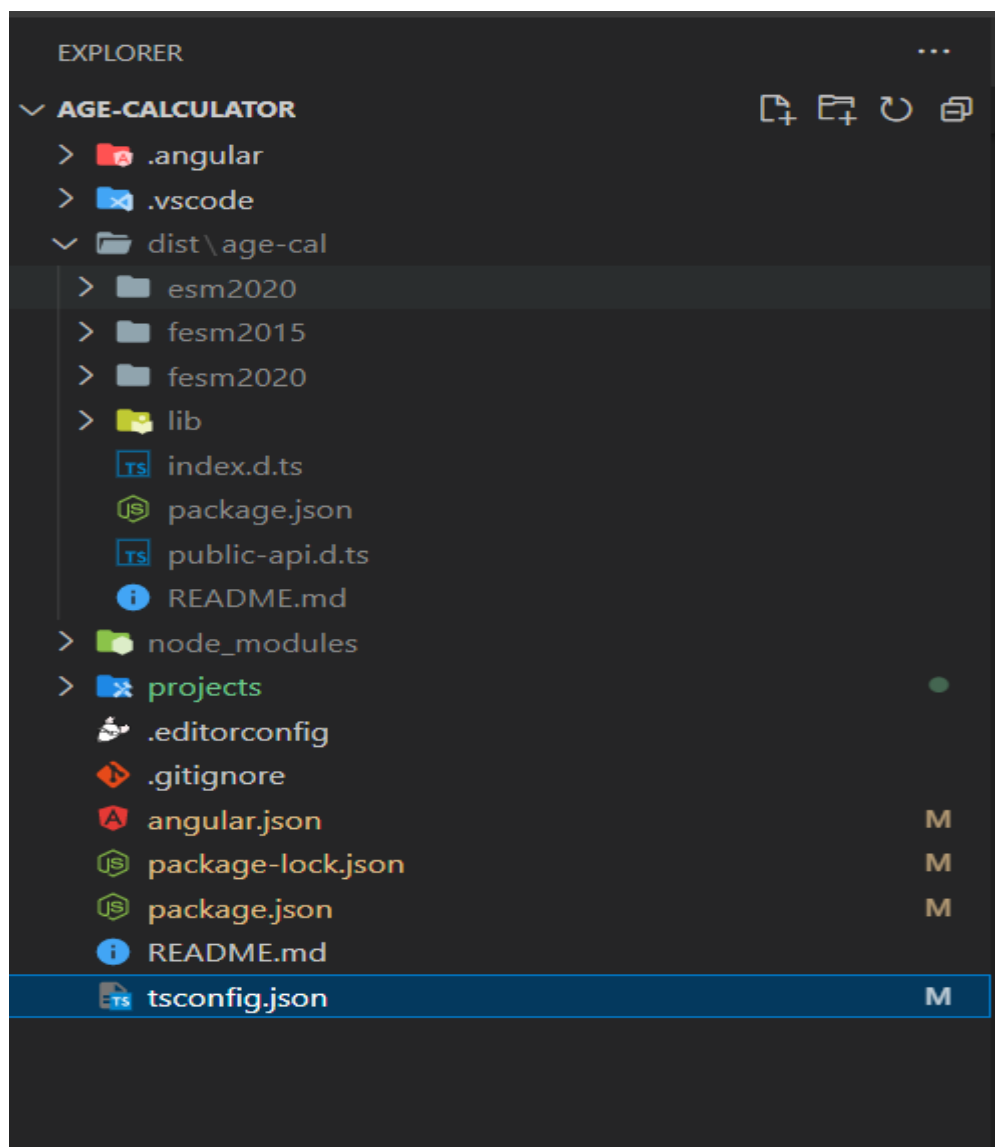
This would create a dist folder for your library.

Step 4:



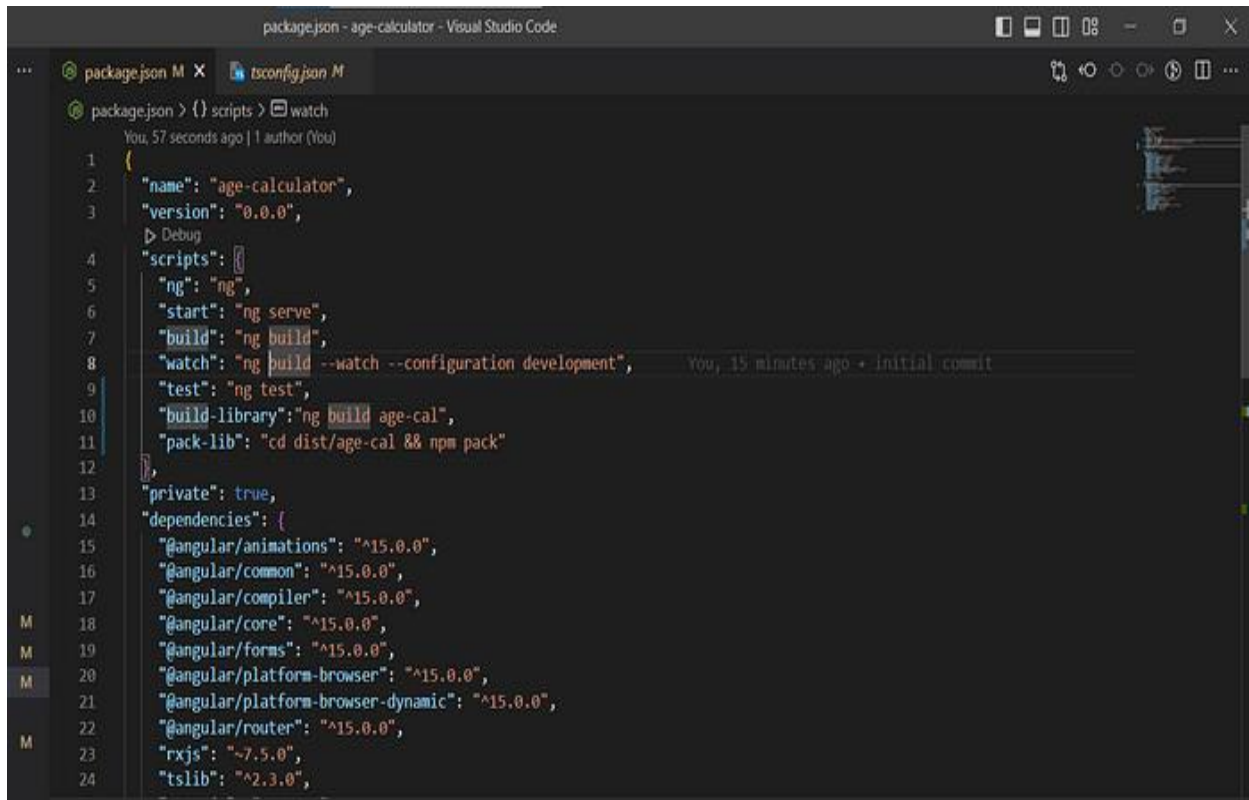
The screenshot shows the Visual Studio Code editor with the `package.json` file open. The file contains the following JSON structure:

```
{
  "name": "age-calculator",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "build-library": "ng build age-cal"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^15.0.0",
    "@angular/common": "^15.0.0",
    "@angular/compiler": "^15.0.0",
    "@angular/core": "^15.0.0",
    "@angular/forms": "^15.0.0",
    "@angular/platform-browser": "^15.0.0",
    "@angular/platform-browser-dynamic": "^15.0.0",
    "@angular/router": "^15.0.0",
    "rxjs": "~7.5.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.12.0"
  }
}
```



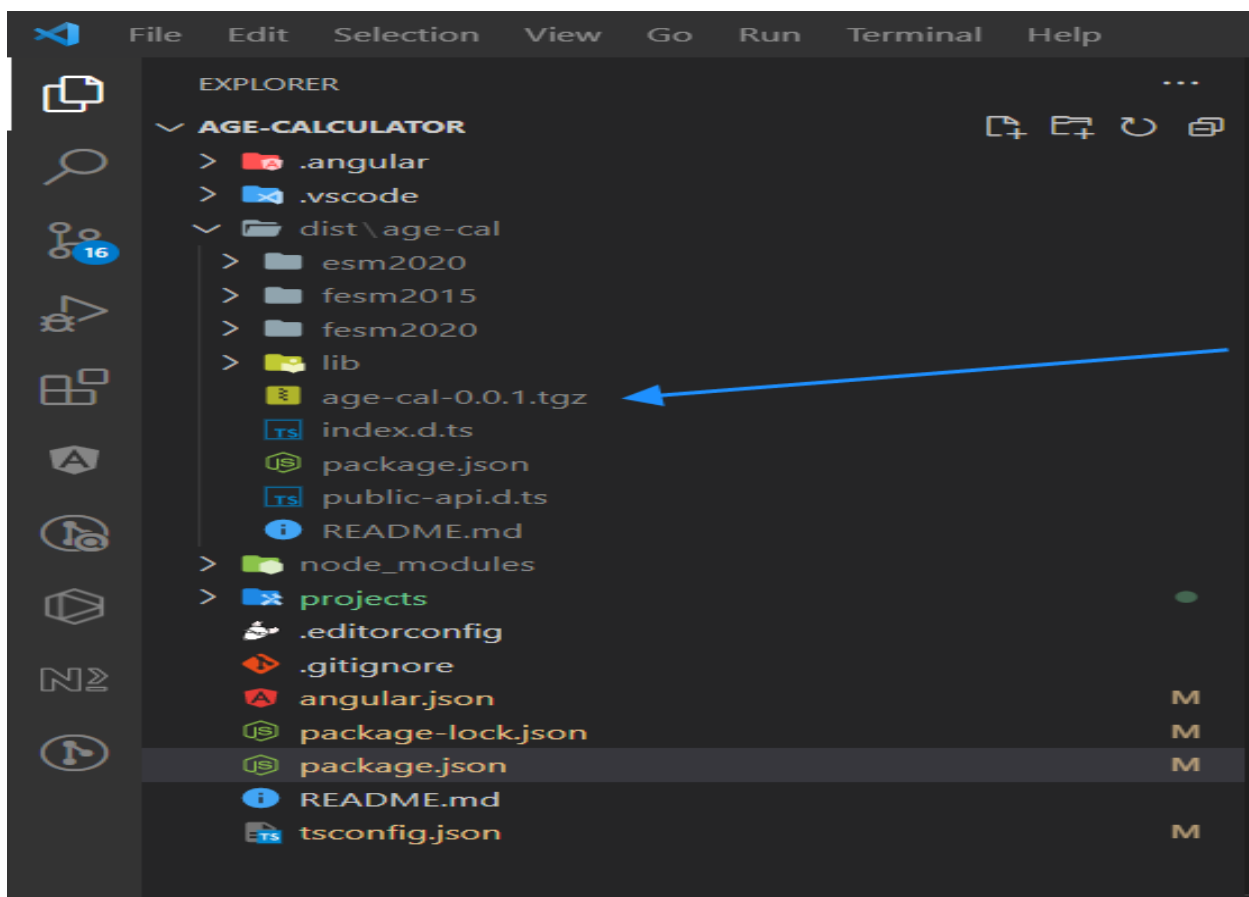
To pack our library, we go to the library distribution directory and run the npm pack command.

```
"pack-lib": "cd dist/age-cal&&npm pack"
```



The screenshot shows the Visual Studio Code editor with the `package.json` file open. The file contains the following configuration:

```
{
  "name": "age-calculator",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "build-library": "ng build age-cal",
    "pack-lib": "cd dist/age-cal && npm pack"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^15.0.0",
    "@angular/common": "^15.0.0",
    "@angular/compiler": "^15.0.0",
    "@angular/core": "^15.0.0",
    "@angular/forms": "^15.0.0",
    "@angular/platform-browser": "^15.0.0",
    "@angular/platform-browser-dynamic": "^15.0.0",
    "@angular/router": "^15.0.0",
    "rxjs": "^7.5.0",
    "tslib": "^2.3.0",
  }
}
```

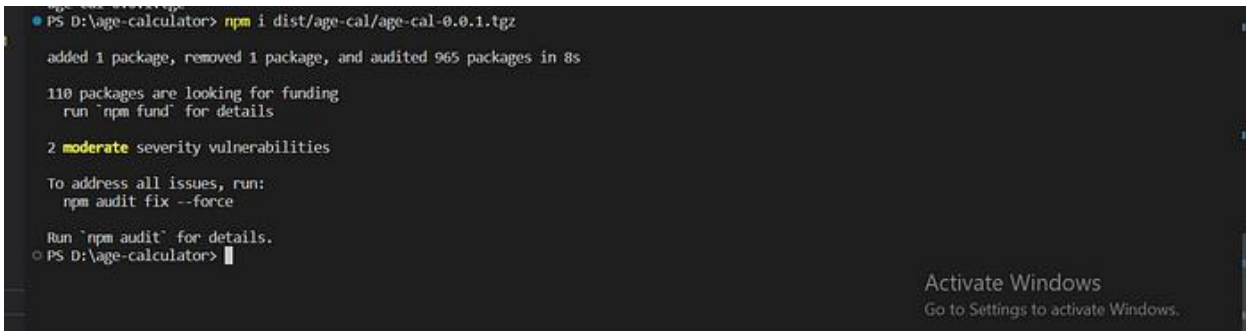


Step 5:

Create a new angular application and install your library inside this using:

```
npm install <path-to-tgz-file>
```

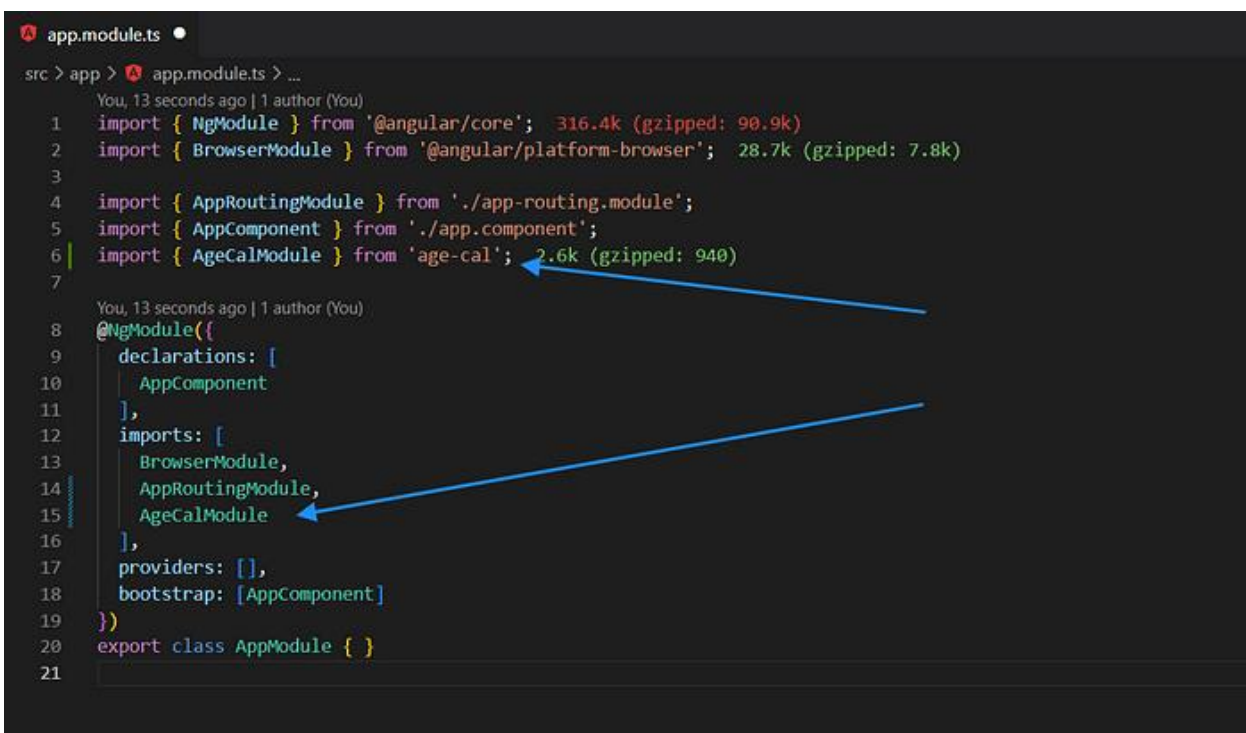
Check your package.json to see if it has been installed and is reflected inside the dependencies.



```
PS D:\age-calculator> npm i dist/age-cal/age-cal-0.0.1.tgz
added 1 package, removed 1 package, and audited 965 packages in 8s
110 packages are looking for funding
  run `npm fund` for details
2 moderate severity vulnerabilities
To address all issues, run:
  npm audit fix --force
Run `npm audit` for details.
PS D:\age-calculator>
```

Step 6: Import the module of your library and use the component.

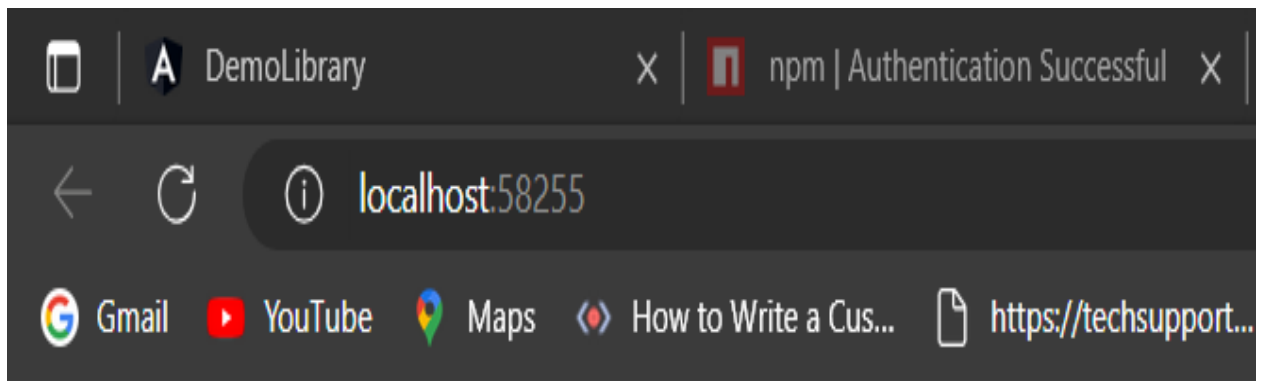
```
import { AgeCalModule } from 'age-cal';
```



```
app.module.ts
src > app > app.module.ts > ...
You, 13 seconds ago | 1 author (You)
1 import { NgModule } from '@angular/core'; 316.4k (gzipped: 90.9k)
2 import { BrowserModule } from '@angular/platform-browser'; 28.7k (gzipped: 7.8k)
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { AgeCalModule } from 'age-cal'; 2.6k (gzipped: 940)
7
8 @NgModule({
9   declarations: [
10     AppComponent
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule,
15     AgeCalModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

Add library component selector in app.component.html

```
<lib-age-cal></lib-age-cal>
```

OUTPUT:

age-calc works!

02/02/2015 your age is 8

EXPERIMENT 10: Creating user defined Libraries

AIM: Use the Angular CLI to generate a new library

PROCEDURE:

To use your own library in an application:

1. Build the library. you cannot use a library before it is built. `content_copy ng build my-lib`.
2. In your applications, import from the library by name: `content_copy import { myExport } from 'my-lib'`;

Step 1:

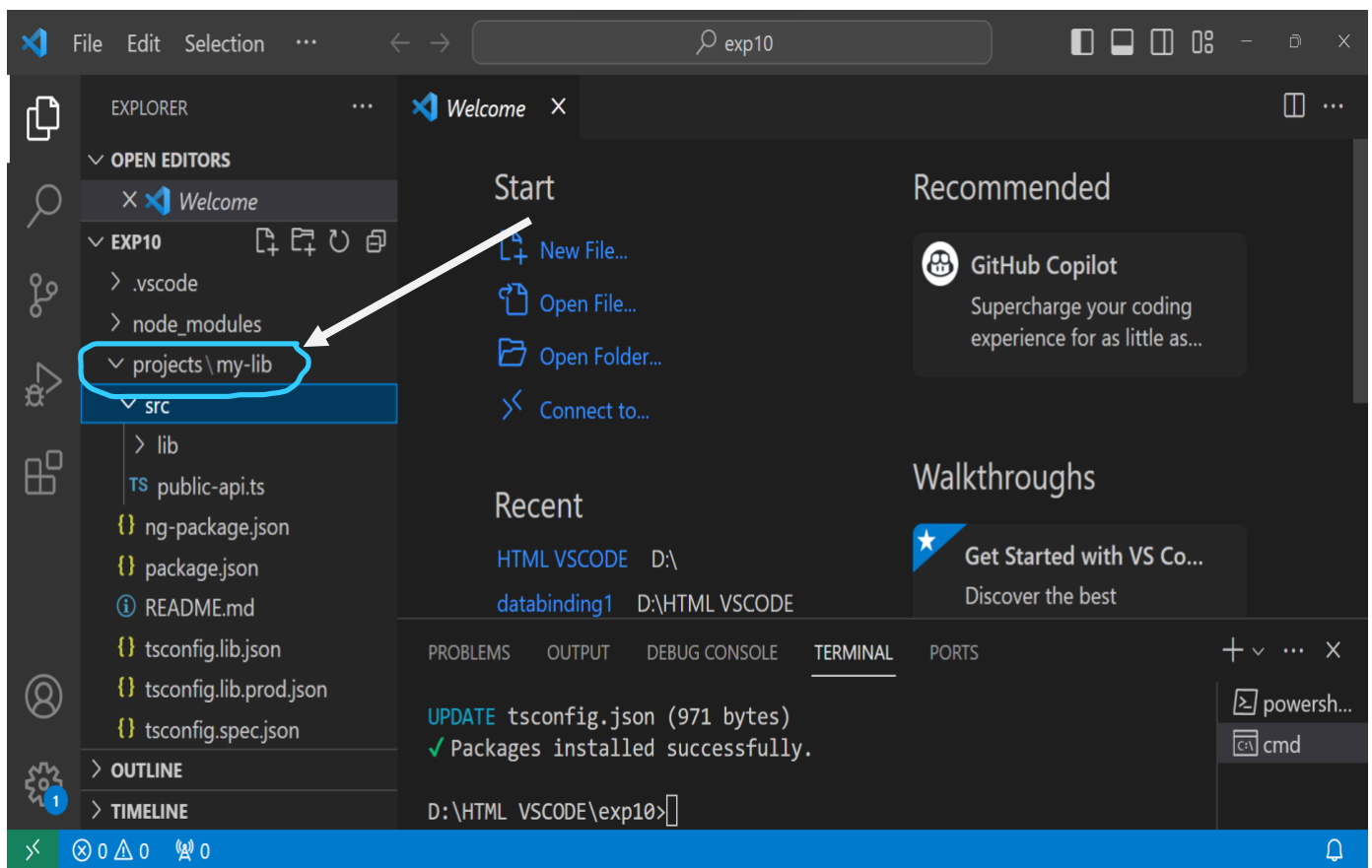
Use the Angular CLI to generate a custom library skeleton in a new workspace with the following commands.

```
ng new my-workspace --no-create-application
```

```
cd my-workspace
```

```
ng generate library my-lib
```

The **ng generate** command creates the **projects/my-lib** folder in your workspace, which contains a component and a service inside an NgModule.

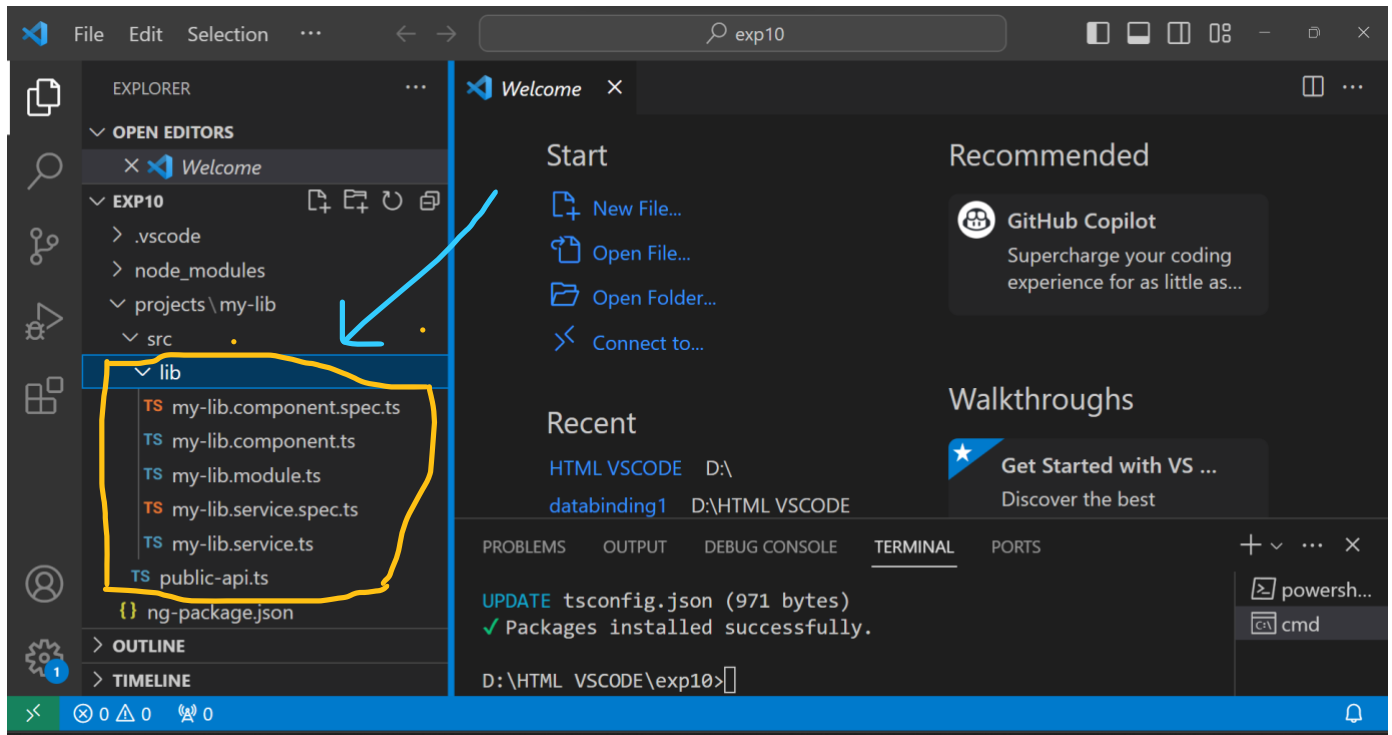


When you generate a new library, the workspace configuration file, `angular.json`, is updated with a project of type library.

Under **src** we have some files under **lib** folder which are default files.

Like:

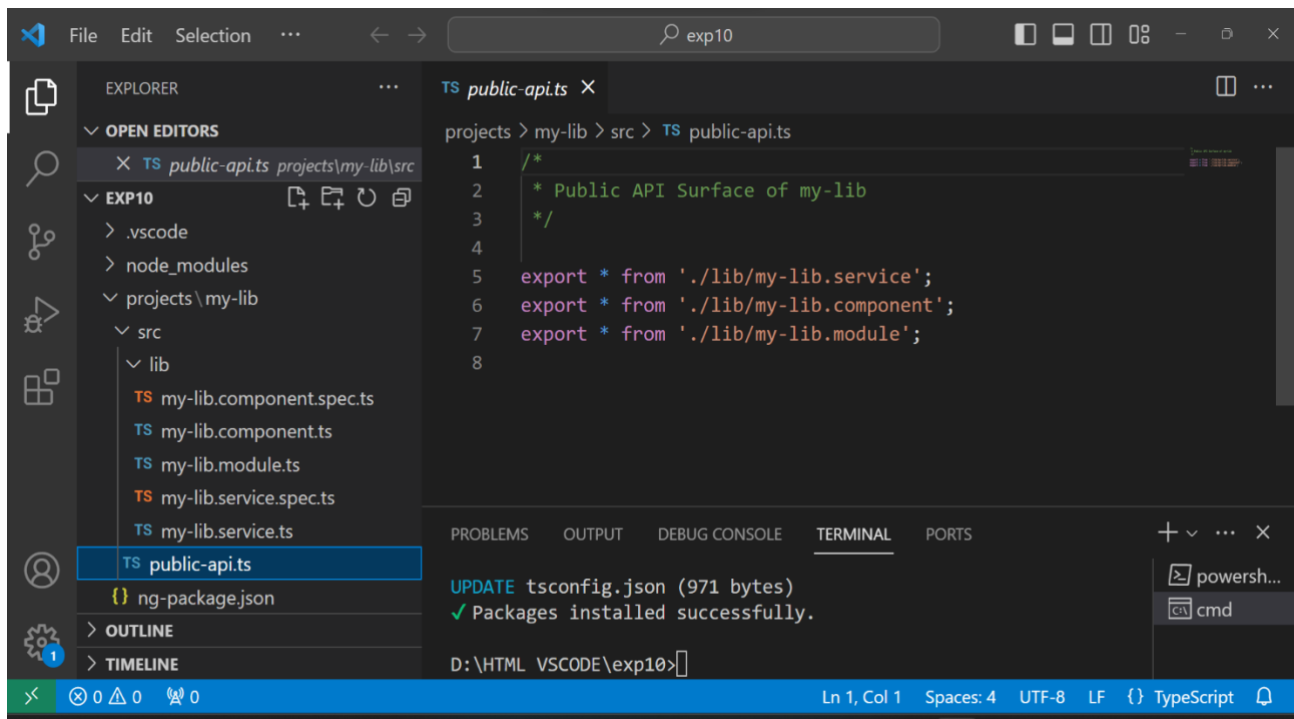
- my-lib.component.spec.ts
- my-lib.component.ts
- my-lib.module.ts
- my-lib.services.spec.ts
- my-lib.services.ts
- public-api.ts



Step 2: src/lib/public-api.ts

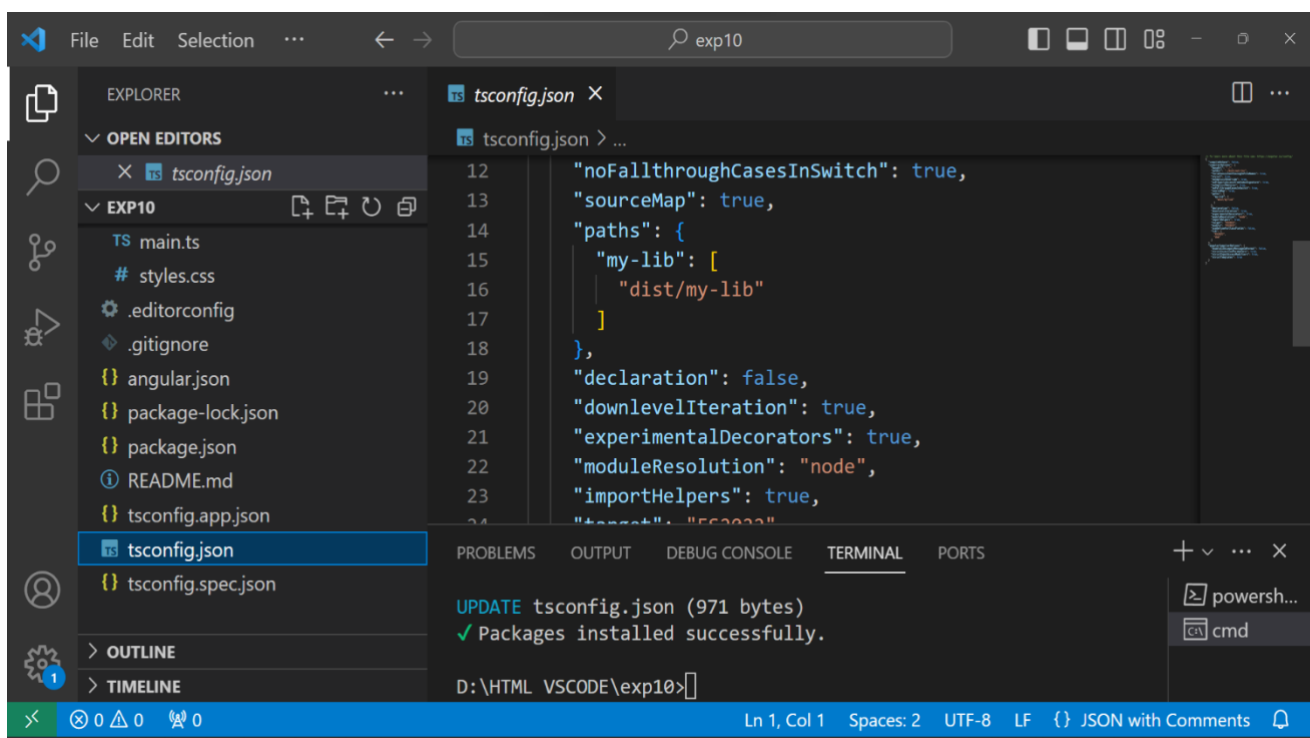
To make library code reusable you must define a **public API** for it. This "user layer" defines what is available to consumers of your library. A user of your library should be able to access public functionality (such as NgModules, service providers and general utility functions) through a single import path.

The **public API** for your library is maintained in the public-api.ts file in your library folder. Anything exported from this file is made public when your library is imported into an application. Use an NgModule to expose services and components.



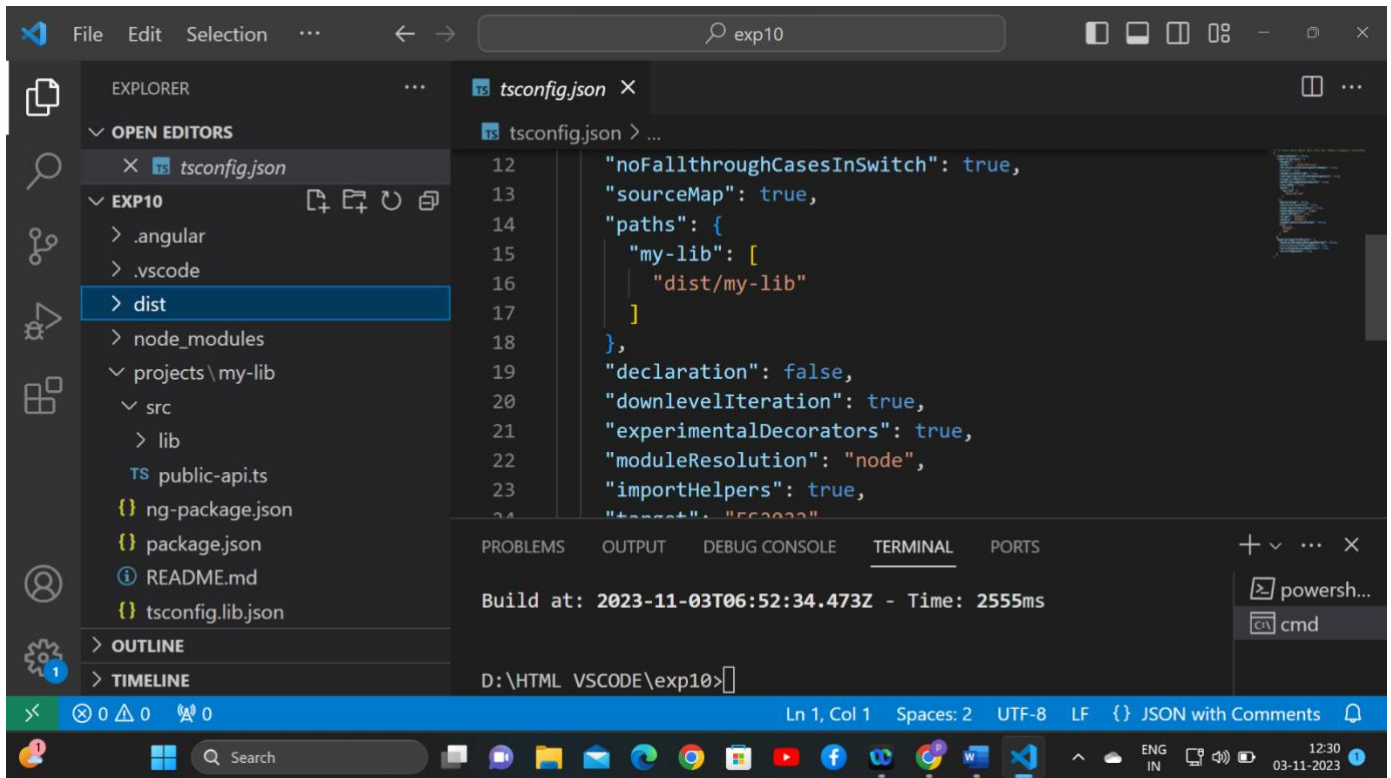
Step
3:

src/tsconfig.json



Step 4: >ng build my-lib

You should always build libraries for distribution using the production configuration. This ensures that generated output uses the appropriate optimizations and the correct package format for npm.

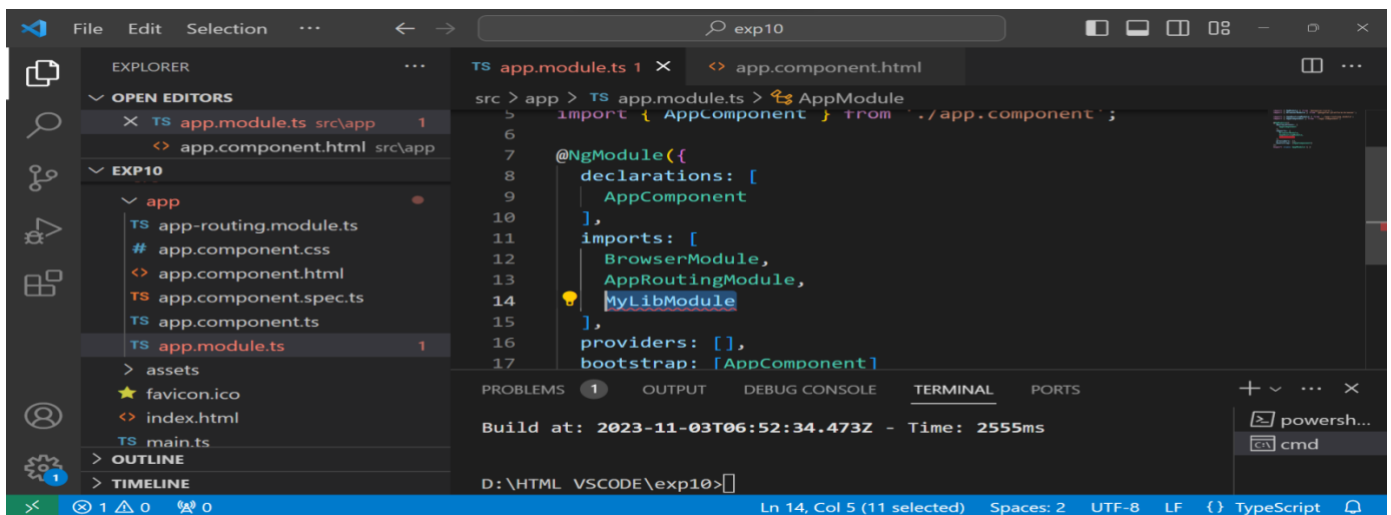


Step 5: app.component.html

```
<lib-my-lib></lib-my-lib>
```

Step 6: app.module.ts

in app.module.ts file import **MyLibModule**



OUTPUT:

