Angular Material

Overview of Angular Material

Angular Material is a User Interface (UI) component library that developers can use in their Angular projects to speed up the development of elegant and consistent user interfaces. Angular Material offers you reusable and beautiful UI components like Cards, Inputs, Data Tables, Datepickers, and much more.

Install Angular Material

Use this command to set up the angular material for your project.

→ ng add @angular/material

It will ask some questions:

1. Choose the name of the theme.

```
✓ Package information loaded.

The package @angular/material@16.0.3 will be installed and executed.
Would you like to proceed? Yes
✓ Packages successfully installed.
? Choose a prebuilt theme name, or "custom" for a custom theme: (Use arrow keys)
> Indigo/Pink        [ Preview: https://material.angular.io?theme=indigo-pink ]
  Deep Purple/Amber  [ Preview: https://material.angular.io?theme=deeppurple-amber ]
  Pink/Blue Grey     [ Preview: https://material.angular.io?theme=pink-bluegrey ]
  Purple/Green       [ Preview: https://material.angular.io?theme=purple-green ]
  Custom
```

Install Angular Material

2. Set up global Angular Material typography styles.
3. Set up browser animations for Angular Material.

```
? Set up global Angular Material typography styles? Yes
? Include the Angular animations module? Include and enable animations
```

Use a component from Angular Material

In order to include any component from angular material in your angular project :

1. From the toolbar select the components.
2. Then select the name of the component you want to add to your project.
3. Go to API and copy the import statement and add it to the shared module.
4. From the Example choose the template.

API Calls (Delete)

How to deal with the Delete function

Step one: Add a new function in the home service that hits API to the Delete function

```
deleteCourse(id:number)
 {
   this.http.delete('https://localhost:44382/api/course/
   DeleteCourse/'+id).subscribe((resp:any)=>{
     console.log('Deleted');
     alert('The Course Deleted Successfully')
   },err=>{
     console.log('Something went wrong');
   })
 }
```

How to deal with the Delete function

Step two: Add a new button called Delete in the HTML file inside the table in the manage course component and send the id as a parameter.

```
 <button type="button"
(click)="openDeleteDailog(course.course_Id)"
class="btn btn-danger m-1">Delete</button>
```

How to deal with the Delete function

Step three: Add the template of the delete form in the HTML file of the manage course component using the dialog from Angular material.

1) From the Component go to the dialog.
2) Add the API and the name of the module in the shared module in array.

```
imports: [
  CommonModule,
  HttpClientModule,
  MatDialogModule
],
exports:[
  NavbarComponent,
  FooterComponent,
  HttpClientModule,
  MatDialogModule
]
```

How to deal with the Delete function

3) Import the shared module in the admin module.

```
imports: [
  CommonModule,
  AdminRoutingModule,
  SharedModule
]
```

How to deal with the Delete function

Step three: Add the template of the delete form in the HTML file of the manage course component.

Note: Use the ng template which is used to render HTML within Angular templates. Although, it is not rendered directly to the DOM.

The <ng-template> is an Angular element for rendering HTML. It is never displayed directly.

```
<ng-template #callDeleteDailog>
    <div style="padding:15px">
<h2>Are you sure you want to delete this item?</h2>
<button class="btn btn-primary m-1"  mat-dialog-close="yes">Yes</button>
<button class="btn btn-danger m-1"  mat-dialog-close="no">No</button>
</div>
</ng-template>
```

How to deal with the Delete function

@ViewChild: Property decorator for configuring the view query. Whenever a change is detected, the change detector looks for the first element or directive matching the selector in the view DOM. As soon as a new child matches the selector in the view DOM, the property is updated.

How to deal with the Delete function

And define ViewChild properties In the manageCourse.component.ts.

```
export class ManageCourseComponent implements OnInit {
@ViewChild('callDeleteDailog') callDeleteDailog!:
TemplateRef<any>
```

How to deal with the Delete function

Step four: Implement the openDeleteDailog function In the managecourse.component.ts

```
  constructor(public home: HomeService, public dialog:
MatDialog) { }
  openDeleteDailog(id: number) {
    const dialogRef = this.dialog.open(this.callDeleteDailog);
    dialogRef.afterClosed().subscribe((result) => {
      if (result != undefined) {
        if (result == 'yes')
          this.home.deleteCourse(id);
        else if (result == 'no')
          console.log('Thank you');     } }) }
```

Reactive Forms

Overview of Reactive Form

Many common applications handle input from users with forms.

Applications use forms to enable users to log in, update a profile, enter sensitive information, and perform many other tasks.

Overview of Reactive Form

Angular supports two ways of handling user input through forms: reactive and template-driven.

It captures user input events from the view, validates the input, creates a form model and data model, and tracks changes.

Overview of Reactive Form

To generate an internal representation of a template-driven form, template directives are used.

In reactive forms, you create your own form representation in the component class.

```
import { ReactiveFormsModule } from '@angular/forms';

imports: [
    CommonModule,
    AuthRoutingModule,
    ReactiveFormsModule
]
```

Advantages of Reactive Form

1. Using custom validators.
2. Changing validation dynamically.
3. Dynamically adding form fields

Form Control

Form Group

Form Control

Overview of Form Control

Form Control: Form input instance with multiple functionalities, ex: validation, flag, and data binding.

An Angular form control encapsulates both the value of data and the validation information for each form element.

In a reactive form, every input should be bound to a form control.

Syntax of Form Control

Use this syntax to create an instance of the form control class:

```
Instance_name = new FormControl(form state, Validation Option)
```

Form Group

Overview of Form Group

A form group encapsulates a collection of form controls.

Controls give you access to the state of elements, while groups give you access to the state of wrapped controls.

At initialization, every form control in a form group is identified by its name.

How to deal with a reactive form

To work with reactive forms, you will be using the ReactiveFormsModule instead of the FormsModule.

Step 1:
Open the Shared.module.ts and add ReactiveFormsModule in the import and export array.

How to deal with a reactive form

Step 2:
Adding a Form to the Component Template.

Step 3:
Building the Component Class.

Overview of Form Group

FormGroups are collections of Form controls that track the values and validity of control instances in the group.

One of the building blocks of angular forms is the FormGroup.

Overview of Form Group

FormGroups encapsulate all information related to a collection of Form Controls that addresses this problem.

Each of these controls has a value and a validation status.

Overview of Form Group

Angular forms can be used by importing the FormsModule (for template-driven forms) and ReactiveFormsModule (for reactive forms) from the @angular/forms package.

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms’;

  imports: [
    FormsModule,
    ReactiveFormsModule
  ],
```

Overview of Form Group

FormGroup Demo will be in the register component.

Register form template link:
https://mdbootstrap.com/docs/standard/extended/registration/

Angular material link:
https://material.angular.io/components/input/overview

API Calls (Create)

Overview Of Http Post

This service is provided as an injectable class that includes methods for making HTTP requests.

During form submission, apps often use POST requests to transmit data to the server.

Example HTTP Post

Inside the admin module, create a new component to contain the template of the create new course form.

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g c admin/createCourse
CREATE src/app/admin/create-course/create-course.component.html (28 bytes)
CREATE src/app/admin/create-course/create-course.component.spec.ts (602 bytes)
CREATE src/app/admin/create-course/create-course.component.ts (229 bytes)
CREATE src/app/admin/create-course/create-course.component.css (0 bytes)
UPDATE src/app/admin/admin.module.ts (770 bytes)
```

Example HTTP Post

In our project, we'll use an angular material dialog box in the create component.

So, add the API for the dialog in the Shared module.

```
import {MatDialogModule} from '@angular/material/dialog';
```

Example HTTP Post

Add the name of the dialog module in the import and export array.

```
imports: [
  CommonModule,
  RouterModule,
  FormsModule,
  ReactiveFormsModule,
  MatFormFieldModule,
  MatInputModule,
  MatDialogModule
],
```

```
exports:[
  FormsModule,
  ReactiveFormsModule,
  MatFormFieldModule,
  MatInputModule,
  NavbarComponent,
  FooterComponent,
  MatDialogModule
]
```

Example HTTP Post

Define a form group in the create component and use it to send data to the DB.

```
export class CreateCourseComponent {
createForm : FormGroup= new FormGroup({
  course_Name :new FormControl('',[Validators.required]),
  price :new FormControl('',[Validators.required]),
  startdate :new FormControl('',[Validators.required]),
  enddate :new FormControl('',[Validators.required]),
  imagename :new FormControl('',[Validators.required])
})
```

Example HTTP Post

Add the template form in the HTML file for Create component.

```
<h2 mat-dialog-title>Create New Course </h2>
<mat-dialog-content class="mat-typography">
    <form class="example-form" [formGroup]="createForm">
        <mat-form-field class="example-full-width" appearance="fill">
            <mat-label>Course Name </mat-label>
            <input type="text" matInput formControlName="course_Name">
            <mat-error *ngIf="createForm.controls['course_Name'].hasError('required')">
                Course Name is <strong>required</strong>
            </mat-error>
        </mat-form-field>
        <br>
        <mat-form-field class="example-full-width" appearance="fill">
            <mat-label>Price</mat-label>
            <input type="number" matInput formControlName="price">
            <mat-error *ngIf="createForm.controls['price'].hasError('required')">
                price is <strong>required</strong>
            </mat-error>
        </mat-form-field>
        <br>
```

Example HTTP Post

```
<mat-form-field class="example-full-width" appearance="fill">
    <mat-label>Start Date </mat-label>
    <input type="text" matInput formControlName="startdate">
    <mat-error *ngIf="createForm.controls['startdate'].hasError('required')">
        Start Date is <strong>required</strong>
    </mat-error>
</mat-form-field>
<br>
<mat-form-field class="example-full-width" appearance="fill">
    <mat-label>End Date </mat-label>
    <input type="text" matInput formControlName="enddate">
    <mat-error *ngIf="createForm.controls['enddate'].hasError('required')">
        End Date is <strong>required</strong>
    </mat-error>
</mat-form-field>
<br>
```

Example HTTP Post

Create a createCourse function in the home service which calls the API function using the post method.

```
    createCourse(body: any) {
       //hits Api (create function)
       debugger
       this.http.post('https://localhost:44382/api/course/'
   , body).subscribe((resp: any) =>
        {
         alert('Created Sucessfully');
       }, err => {
         alert('Something wont wrong');
       })
     }
```

Example HTTP Post

Create a saveCourse function in the create component to call createCourse function from home services.

```
saveCourse()
{
  this.home.createCourse(this.createForm.value)
;
}
```

Example HTTP Post

Add a button in the HTML file of the create component to call the SeveCourse function.

```
<mat-dialog-actions align="end">
    <button mat-button mat-dialog-close>Cancel</button>
    <button mat-button (click)="saveCourse()" [mat-dialog-close]="true"
     cdkFocusInitial>Save</button>
</mat-dialog-actions>
```

Example HTTP Post

Add a button in the HTML file of the Manage course component to open the create dialog .

```
<button type="button" (click)=“openCreateDialog()“
class="btn btn-outline-info m-1">Create Course
</button>
```

Example HTTP Post

In the typescript file for managecourse component create an object of MatDialog Service to use an open method from this service.

```
constructor(private dialog :MatDialog) { }

  openCreateDialog() {
    const dialogRef = this.dialog.open(CreateCourseComponent);
  }
```

Note: The open method takes the name of the component as a parameter.

Example HTTP Post