1. **Exercise1 : Create feature module and routing module**

**Step 1:** Type this command to generate a new module with routing

ng generate module --routing=true **users**

Change ‘users’ to your feature folder name under the app folder where you want modules to be generated.

The following files should be generated under app/users

**users-routing.module.ts**

**users.module.ts**

**Step 2:**

Open **users.module.ts** & add components which belong to this module into **declarations**

@NgModule({

declarations: [

NewUserComponent,

UserInfoComponent,

UserListComponent,

UsersComponent,

],

imports: [

CommonModule,

UsersRoutingModule

]

})

export class UsersModule { }

**Step 3:**

Open **app.module.ts** & remove these components from **declarations**

NewUserComponent,

UserInfoComponent,

UserListComponent,

UsersComponent,

**Step 4:**

Open **users-routing.module.ts** (generated by ng generate module --routing=true users) and add the following routing config

**const routes: Routes = [**

**{ path: 'users',**

**component: UsersComponent ,**

**children: [**

**{ path: '',**

**component: UserListComponent,**

**children:[**

**{ path:'',component:UserInfoComponent},**

**{ path: 'edit-user/:userName', component: NewUserComponent },**

**{ path: 'new-user',component: NewUserComponent}**

**]**

**},**

**]**

**},**

**];**

**Step 5:**

Update content of **users.component.html** looks like this

<router-outlet></router-outlet>

Update content of user-list.component.html looks like this

<router-outlet></router-outlet>

<h2>User List</h2>

<table class="table table-striped">

<tr>

<th>User Name</th>

<th>Email</th>

<th>Phone</th>

<th>Actions</th>

</tr>

<tr *\*ngFor*="let user of users" [class.table-active]="user === selectedUser"

(click)="onSelect(user)">

<td>{{user?.userName}}</td>

<td>{{user?.email}}</td>

<td>{{user?.numberPhone}}</td>

<td>

<a routerLink="/users/edit-user/{{user?.userName}}" routerLinkActive="active">Edit User</a>

</td>

</tr>

</table>

<button type="button" routerLink="/users/new-user" class="btn btn-link">New User</button>

NOTE:

**<router-outlet></router-outlet>** is a placeholder where you expect content of component is render

**Step 6:**

Open **app-routing.module** & update routing config ( Remove paths for users since we already move to users-routing.module.ts

const routes: Routes = [

{ path: '',

redirectTo: '/users',

pathMatch: 'full'

},

{ path: '\*\*', component: PageNotFoundComponent }

];

**Step 7:**

Open **app.module** & import **users.module** before **app.routing.module**

@NgModule({

declarations: [

AppComponent,

PageNotFoundComponent

],

imports: [

BrowserModule,

UsersModule,

AppRoutingModule,

],

providers: [],

bootstrap: [AppComponent]

})

**Step 8:**

Run ng serve & check navigations

Hint:

Please type code instead of copy & paste.

Ask question if there are somethings unclear

**Exercise 2 : Connect backend & login with user name & password**

1. Download Spring Boot app from here 

Note: please follow this link below if you have not configured maven setting

<https://inside-docupedia.bosch.com/confluence/display/igpmci/How+to+setup+Maven>

1. Start application with command line: mvn spring-boot:run
2. Create login form which has 2 input fields user name & password and one Login button
3. Generate a login service & inject into login component

ng generate service users/services/login

1. Inject HttpClient into login service
2. Create login function in LoginService & call post function from http client with body user & password

public login(userName:string,pass:string) {

let body = new LoginModel(userName,pass);

return this.httpClient.post('http://localhost:8080/token/generate-token',body);

}

1. Handle login button by submitting user & password to serve. After login successfull, save token into local storage. Check response status is 200, if yes redirect to user list

this.loginService.login(userName,pass).subscribe( (data) => {

if(data['status'] == 200) {

let token = data['token'];

localStorage.setItem('toke',token);

// use router.navigate to navigate to other page

}

**Exercise 3: Register new user**

1. Change login screen to add ‘Register’ link and add router link to new-user

<a [routerLink](https://angular.io/api/router/RouterLink)="/users/new-user" [routerLinkActive](https://angular.io/api/router/RouterLinkActive)="active">Register</a>

1. Change new user screen to add the following input fields :

user name, password, first name, last name, age, salary

1. Bind input fields with form using ReactiveForm ( see last exercise )
2. Add a save button on the new user screen & create saveUser function to handle save event for the button

<button class="btn btn-success" (click)="saveUser()">Add</button>

1. Change user.model to add the following attributes :

export class User {

id: number;

username: string;

password:string;

firstName:string;

lastName:string;

age:number;

salary:number

}

1. Create new user service & add createUser function

ng generate service user/services/user

createUser(user: User) {

return this.httpClient.post('http://localhost:8080/signup', user);

}

1. Inject user service into user.component.ts via constructor

constructor(private userService:UserService) { }

* 1. Change saveUser function in user.component.ts to call createUser function from user service

saveUser() {

this.userService.createUser(this.form.value)

.subscribe( data => {

this.router.navigate(['login']);

});

}

1. Note: change path ‘login’ to match with your routing config

Run & check the changes. You should be able to login with the newly created user.

**Exercise 4: Get list of user from server**

1. Add getUsers function in user service

getUsers() {

let token = window.localStorage.getItem('token');

let headers: HttpHeaders = new HttpHeaders();

headers.set('Authorization','Bearer ' + token);

return this.httpClient.get('http://localhost:8080/users',{headers:headers});

}

Hints:

let token = window.localStorage.getItem('token');

is to read the token from local storage. This token is save after login

let headers: HttpHeaders = new HttpHeaders();

headers.set('Authorization','Bearer ' + token);

is to create HttpHeaders object which is used for the 2nd parameter in get function of http client

This parameter is to by-pass authentication when sending a request to the server. This token contains user name & password which are in encrypted format

1. Inject user service in user list component

constructor(private userService:UserService ) { }

1. Update ngOnInit to call getUsers from user service & load data from server

users: User[];

ngOnInit() {

this.apiService.getUsers()

.subscribe( data => {

this.users = data.result;

});

}

1. Login again & check if we can see list of user after login

**Exercise 5: Practice with Angular Mat Table**

1. Add imports: [MatTableModule, HttpClientModule] in app.modules.ts
2. Create Members Component:

ng g component features/components/members

1. Create share/ member.model.ts

export class Member {  
 public login: string;  
 public id: number;  
 public node\_id: string;  
 public avatar\_url: string;  
 public gravatar\_id: string;  
 public url: string;  
 public html\_url: string;  
 public followers\_url: string;  
 public following\_url: string;  
 public gists\_url: string;  
 public starred\_url: string;  
 public subscriptions\_url: string;  
 public organizations\_url: string;  
 public repos\_url: string;  
 public events\_url: string;  
 public received\_events\_url: string;  
 public type: string;  
 public site\_admin: boolean;  
  
 @Deserializer()  
 public static deserialize(json: any | string): Member {  
 return undefined;  
 }  
}

1. create share/ member-data-table.model.ts

export class MemberDataTable {  
 public login: string;  
 public id: number;  
 public node\_id: string;  
 public type: string;  
  
 constructor(login: string, id: number, node\_id: string, type: string) {  
 this.login = login;  
 this.id = id;  
 this.node\_id = node\_id;  
 this.type = type;  
 }  
}

1. Create arrays hold dataSource: Arrays hold data for material table and tableColumns: variable that will be used to hold the name of the columns in members.component.ts.

dataSource: MemberDataTable[];  
  
tableColumns: string[] = ['login', 'id', 'node\_id', 'type'];

1. Open the members.component.html template and add the <mat-table> component to create a table:

<table mat-table>  
   
</table>

1. You also need to provide a data source from where the table can get the data to display. You can provide a data source by using the dataSource property:

<table mat-table [dataSource]="dataSource">  
   
</table>

1. Defining the Material Table’s Columns Templates For example, let's suppose you want to add the following columns to your table:

* Login
* Id
* Node Id
* Type

<ng-container matColumnDef="login">  
 <mat-header-cell \*matHeaderCellDef>Login</mat-header-cell>  
 <mat-cell \*matCellDef="let row">{{row.login}}</mat-cell>  
</ng-container>

<ng-container> component to create a column definition. Inside of it, you define the header and row cells using the <mat-header-cell> and <mat-cell> components.The matColumnDef property of defines the unique name of the column. In this case it's policyNumber.

1. Defining the Material Table’s Rows Templates. To define rows in your Material table, you need to include <mat-header-row> and <mat-row> components inside your table and provide the tableColumns array which holds the columns list.

matHeaderRowDef directive is used to provide a configuration object for the table header row. In this case, it's the name of the variable that provides the list of columns,

next, the matRowDef directive is used to provide a configuration for row cells,

The <mat-header-row> and <mat-row> provide Material styling for the row's header and cells.

<mat-header-row \*matHeaderRowDef="tableColumns"></mat-header-row>  
<mat-row \*matRowDef="let row; columns: tableColumns"></mat-row>

1. Create members services members.service.ts

constructor(private httpClient: HttpClient) {  
}  
public getMemebers(): Observable<Member[]> {  
 return this.httpClient.get<Member[]>('https://api.github.com/orgs/lemoncode/members');  
}

1. Inject user service into members.component.ts via constructor

constructor(private readonly membersService: MembersService)

1. Update ngOnInit to call getMembers from members service & load data from server and convert Member[] to MembersDataTable.

ngOnInit() {  
 this.membersService.getMemebers().subscribe((members: Member[]) => {  
 if (!!members) {  
 this.dataSource = members.map(value => new MemberDataTable(value.login, value.id, value.node\_id, value.type));  
 }  
 });  
}

**Exercise 6: Practice with Angular Material Design Components to enhance UI/UX**

1. Access this page <https://startangular.com/product/materialpro-angular-8/> and download sample source code
2. Change to angular-8 directory & run npm install
3. Modify template to suit your need
4. Run & check template
5. Customize the template to enhance login, user list, new user screen

**Exercise 7: Practice with Angular Feature Module**

1. Add Angular Material library

ng add @angular/material

1. Create new folder called “**features**” under **app** folder.
2. Generate feature module

ng g module features/car-management --routing=true

*Note: observe files generated from the command.*

1. Generate main component of feature module

ng g component **features/car-management**

1. Change routers config

const routes: Routes = [  
 {  
 path: '',  
 component: CarManagementComponent,  
 }  
];

1. Generate sub component

ng g component features/car-management/components/car-table

1. Add imports: [MatTableModule, HttpClientModule] in app.module.ts
2. Change root router config in app-routing.module.ts

const routes: Routes = [  
 {  
 path: '',  
 component: AppComponent,  
 children: [  
 {  
 path: 'cars',  
 loadChildren: () => import('./features/car-management/car-management.module').then(m => m.CarManagementModule)  
 },  
 ]  
 },  
];

Note: **app.component.html** should have **<router-outlet></router-outlet>**

1. Create new folder **models** under **features/car-management**
2. Create new file **car.model.ts** under features/car-management/models

export class CarModel {  
 car\_id: number;  
 name: string;  
 brand: string;  
 year\_release: string;  
}

1. Generate car.service.ts which gets data from server

ng g service features/car-management/services/cars

@Injectable({  
 providedIn: 'root'  
})  
export class CarsService {  
  
 constructor(private httpClient: HttpClient) {  
 }  
 public getCars(): Observable<CarModel[]> {  
 return this.httpClient.get<CarModel[]>('http://localhost:3050/api/cars');  
 }  
}

1. Edit CarTableComponent to get data from server

export class CarTableComponent implements OnInit {  
  
 constructor(private readonly carsService: CarsService) {  
 }  
  
 dataSource: CarModel[];  
  
 tableColumns: string[] = ['name', 'brand', 'year\_release', 'edit', 'delete'];  
  
 ngOnInit() {  
 this.carsService.getCars().subscribe((cars: CarModel[]) => {  
 this.dataSource = cars;  
 });  
 }  
  
}

1. Edit **car-table.component.html** to use **mat-table** directive

<mat-table style="width: 100%" [dataSource]="dataSource" class="mat-elevation-z8">  
 <ng-container matColumnDef="name">  
 <mat-header-cell *\*matHeaderCellDef*>Name</mat-header-cell>  
 <mat-cell *\*matCellDef*="let row">{{row.name}}</mat-cell>  
 </ng-container>  
 <ng-container matColumnDef="brand">  
 <mat-header-cell *\*matHeaderCellDef*>Brand</mat-header-cell>  
 <mat-cell *\*matCellDef*="let row">{{row.brand}}</mat-cell>  
 </ng-container>  
 <ng-container matColumnDef="year\_release">  
 <mat-header-cell *\*matHeaderCellDef*>Year Release</mat-header-cell>  
 <mat-cell *\*matCellDef*="let row">{{row.year\_release}}</mat-cell>  
 </ng-container>  
 <ng-container matColumnDef="edit">  
 <mat-header-cell mat-header-cell class="action-cell" *\*matHeaderCellDef*>Action</mat-header-cell >  
 <mat-cell mat-cell *\*matCellDef*="let element" >  
 <span>  
 <a mat-button routerLink="/cars/edit/1">Edit</a>  
 </span>  
 </mat-cell>  
 </ng-container>  
 <ng-container matColumnDef="delete">  
 <mat-header-cell mat-header-cell class="action-cell" *\*matHeaderCellDef*>Action</mat-header-cell>  
 <mat-cell mat-cell *\*matCellDef*="let element" >  
 <span>  
 <a mat-button routerLink="/cars/delete/1">Delete</a>  
 </span>  
 </mat-cell>  
 </ng-container>  
 <mat-header-row *\*matHeaderRowDef*="tableColumns"></mat-header-row>  
 <mat-row *\*matRowDef*="let row; columns: tableColumns"></mat-row>  
</mat-table>

1. Extract server.zip file into a folder



1. Open command line & cd to the extracted folder which contains **server.js** file. Start server by typing the command below

npm start

1. Run angular app & open url <http://localhost:4200/cars>
2. Generate new CarEditComponent

ng g component features/car-management/components/car-edit

1. Edit **car-edit.component.ts** and add logic to update car

@Component({  
 selector: 'app-car-edit',  
 templateUrl: 'car-edit.component.html',  
 styles: []  
})  
export class CarEditComponent implements OnInit {  
 isSaving = false;  
 editForm = this.fb.group({  
 car\_id: [],  
 name: [],  
 brand: [],  
 year\_release:[]  
 });  
 constructor(protected carsService: CarsService, protected activatedRoute: ActivatedRoute, private fb: FormBuilder) {}  
  
 ngOnInit() {  
 this.activatedRoute.params.subscribe((params: any) => {  
 const id = Number(params.id);  
 this.carsService.find(id).subscribe((carModel: CarModel) => {  
 this.updateForm(carModel);  
 });  
 });  
 }  
  
 updateForm(carModel: CarModel): void {  
 this.editForm.patchValue({  
 car\_id: carModel.car\_id,  
 name: carModel.name,  
 brand: carModel.brand,  
 year\_release: carModel.year\_release,  
 });  
 }  
  
 previousState(): void {  
 window.history.back();  
 }  
  
 save(): void {  
 this.isSaving = true;  
 const carModel = this.createFromForm();  
 if (carModel.car\_id !== undefined) {  
 this.subscribeToSaveResponse(this.carsService.update(carModel));  
 } else {  
 this.subscribeToSaveResponse(this.carsService.create(carModel));  
 }  
 }  
  
 private createFromForm(): CarModel {  
 return {  
 ...new CarModel(),  
 car\_id: this.editForm.get(['car\_id'])!.value,  
 name: this.editForm.get(['name'])!.value,  
 brand: this.editForm.get(['brand'])!.value,  
 year\_release: this.editForm.get(['year\_release'])!.value,  
 };  
 }  
  
 protected subscribeToSaveResponse(result: Observable<any>): void {  
 result.subscribe(  
 () => this.onSaveSuccess(),  
 () => this.onSaveError()  
 );  
 }  
  
 protected onSaveSuccess(): void {  
 this.isSaving = false;  
 this.previousState();  
 }  
  
 protected onSaveError(): void {  
 this.isSaving = false;  
 }  
}

1. Edit **car-edit.component.html**

<div class="row justify-content-center">  
 <div class="col-8">  
 <form name="editForm" role="form" novalidate (ngSubmit)="save()" [formGroup]="editForm">  
 <h2 id="jhi-testent-heading">Create or edit a Car</h2>  
  
 <div>  
  
 <div class="form-group" [hidden]="!editForm.get('car\_id')!.value">  
 <label for="car\_id">ID</label>  
 <input type="text" class="form-control" id="car\_id" name="car\_id" formControlName="car\_id" readonly />  
 </div>  
  
 <div class="form-group">  
 <label class="form-control-label" for="field\_name">Name</label>  
 <input type="text" class="form-control" name="name" id="field\_name"  
 formControlName="name"/>  
 </div>  
  
 <div class="form-group">  
 <label class="form-control-label" for="brand">Brand</label>  
 <input class="form-control" name="brand" id="brand"  
 formControlName="brand"/>  
 </div>  
  
 <div class="form-group">  
 <label class="form-control-label" for="year\_release">Year release</label>  
 <input class="form-control" name="year\_release" id="year\_release"  
 formControlName="year\_release"/>  
 </div>  
 </div>  
  
 <div>  
 <button mat-button type="button" id="cancel-save" class="btn btn-secondary" (click)="previousState()">  
 Cancel  
 </button>  
  
  
 <button style="margin-left: 2rem;" type="submit" [disabled]="editForm.invalid || isSaving" class="btn btn-primary">  
 Save  
 </button>  
 </div>  
 </form>  
 </div>  
</div>

1. Edit **cars.service.ts**

@Injectable({  
 providedIn: 'root'  
})  
export class CarsService {  
 public resourceUrl = 'http://localhost:3050/api/cars';  
  
 constructor(private httpClient: HttpClient) {  
 }  
  
 public getCars(): Observable<CarModel[]> {  
 return this.httpClient.get<CarModel[]>(this.resourceUrl);  
 }  
  
 public find(id: number): Observable<CarModel> {  
 return this.httpClient.get<CarModel>(`${this.resourceUrl}/${id}`);  
 }  
  
 public create(data: CarModel) {  
 return this.httpClient.post<CarModel>(this.resourceUrl, data, {observe: 'response'});  
 }  
  
 public update(data: CarModel) {  
 return this.httpClient.put<CarModel>(`${this.resourceUrl}/${data.car\_id}`, data);  
 }  
  
  
 public delete(id: number): Observable<HttpResponse<{}>> {  
 return this.httpClient.delete(`${this.resourceUrl}/${id}`, {observe: 'response'});  
 }  
}

1. Edit **car-management-routing.module.ts** to add path for edit

const routes: Routes = [  
 {  
 path: '',  
 component: CarManagementComponent,  
 },  
 {  
 path: 'edit/:id',  
 component: CarEditComponent,  
 }  
];