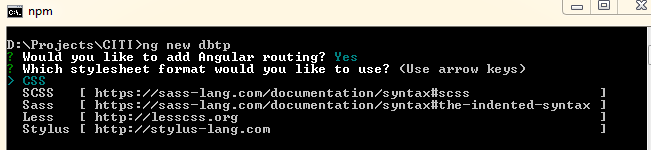
Creating the project

Create a project

Ng new dbtp



Then do cd dbtp and do npm install



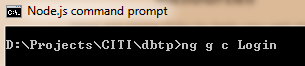
Run the project ng serve

The project will be opened at localost:4200

Creating the components

We need 5 components

1. Register
2. Login
3. Buy
4. Positions
5. Statement



Repeat for all the components

By this point you will have 6 components, 5 above ones and App components which is created by default.

Installing the bootstrap

Let us install the bootstrap 5

npm install bootstrap@next

In the Angular.json make the entries to make bootstrap work

Make sure you don’t add bootsratrap it in the test section of the Angular.json.

 "styles": [

              "src/styles.css",

              "./node\_modules/bootstrap/dist/css/bootstrap.min.css"

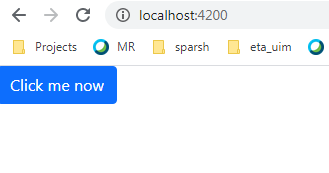
            ],

To see if the bootstrap is really working, empty the App component and put a simple button there

<div>

  <button class="btn btn-primary">Click me now</button>

</div>



That means bootstrap is working.

Putting up the navigation

In the App-routing.module.ts, you will see the routing module.

Routing has 2 parts

1. Routing
2. Router outlet

The component which has the router-outlet will be always shown. In our case it is the App component which will be always shown because we have our router-outlet into it.

Let us setup basic routes

const routes: Routes = [

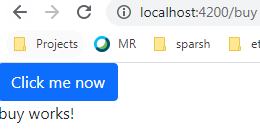
  { path:  'buy', component: BuyComponent},

  { path:  'positions', component: PositionsComponent},

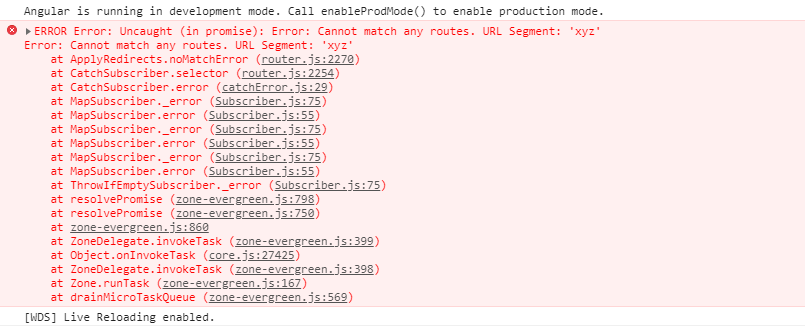
  {path: 'statement', component: StatementComponent}

];

And check how each route behaves. E.g. I hit buy, I will see the App component content and the Buy Component.



If I try hitting any route which does not exist e.g. xyz, I will get the error in the console



Let us also have a route to cover such cases.

Note : Don’t use path as register 🡺 as it does not work

const routes: Routes = [

  { path:  'buy', component: BuyComponent},

  { path:  'positions', component: PositionsComponent},

  {path: 'statement', component: StatementComponent},

  {path:'login', component: LoginComponent},

  {path:'signup', component: RegisterComponent},

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

];

Hence we setup a path \*\* , in this case , if no matching route is found we will log out the person and make him re-login. This also solves our error when we route to a non-existent path. Which We will see later.

Clear the App component

Delete the click button from the App component except the router outlet

Setting Navigation Menu in the App component

Since what we put in the App component always shows up, we will use it to show that we need to see on all components and that is navigation.

For the navigation component we will need bootstrap’s CSS as well as JavaScript

 "scripts": [

              "./node\_modules/bootstrap/dist/js/bootstrap.bundle.min.js"

            ]

The bundle has popper.js hence we used bundle.min.js. After you make any changes to angular.json stop and start the server. Then in the App component template put below markup

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

  <div class="container-fluid">

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo03"

      aria-controls="navbarTogglerDemo03" aria-expanded="false" aria-label="Toggle navigation">

      <span class="navbar-toggler-icon"></span>

    </button>

    <a class="navbar-brand" href="/buy">Buy</a>

    <div class="collapse navbar-collapse" id="navbarTogglerDemo03">

      <ul class="navbar-nav mr-auto mb-2 mb-lg-0">

        <li class="nav-item">

          <a class="nav-link active" aria-current="page" href="/positions">Positions</a>

        </li>

        <li class="nav-item">

          <a class="nav-link active" aria-current="page" href="/statement">Statement</a>

        </li>

        <li class="nav-item">

          <a class="nav-link active" aria-current="page" href="/signup">Log Out</a>

        </li>

      </ul>

    </div>

  </div>

</nav>

<router-outlet></router-outlet>

Till now, we have just replaced the links with our links. But it causes a refresh. We need to replace the href with routerLink so that navigation happens without any refresh.

[routerLink]="['/buy']"

[routerLink]="['/positions']"

[routerLink]="['/statement']"

[routerLink]="['/signup']"

With this change done, we can have flawless navigation anywhere in the application without any refresh.

With this your navigation is setup. Also remove active css classes from the nav items. Also keep the nav-branding as DBTP. The final html should look like

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

  <div class="container-fluid">

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo03"

      aria-controls="navbarTogglerDemo03" aria-expanded="false" aria-label="Toggle navigation">

      <span class="navbar-toggler-icon"></span>

    </button>

    <a class="navbar-brand" [routerLink]="['/buy']">DBTP</a>

    <div class="collapse navbar-collapse" id="navbarTogglerDemo03">

      <ul class="navbar-nav mr-auto mb-2 mb-lg-0">

        <li class="nav-item">

          <a class="nav-link" aria-current="page" [routerLink]="['/positions']">Positions</a>

        </li>

        <li class="nav-item">

          <a class="nav-link" aria-current="page" [routerLink]="['/statement']">Statement</a>

        </li>

        <li class="nav-item">

          <a class="nav-link" aria-current="page" [routerLink]="['/signup']">Log Out</a>

        </li>

      </ul>

    </div>

  </div>

</nav>

<router-outlet></router-outlet>

Designing the component

# Register Component

This is the first component we will design. For creating this component, we will use Reactive forms where the form is controlled in the backend. The form will have following fields.

1. Username – required
2. Email – required and has to be email
3. Password – has to be minimum 8 characters’ long

There are 2 types of forms in Angular

1. Template driven – are used when form is simple
2. Reactive forms – are used when form is complex

To use both types of forms, both types of modules needs to be imported in the app module.

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

 imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule, //for template-drive forms

    ReactiveFormsModule //for the reactive forms

  ],

In the TS of the Register Component,

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

import { Router } from '@angular/router';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({

  selector: 'app-register',

  templateUrl: './register.component.html',

  styleUrls: ['./register.component.css']

})

export class RegisterComponent implements OnInit {

  //define the variables

  registrationForm: FormGroup;

  constructor(

    private formBuilder: FormBuilder,

    private router: Router,

  ) { }

  ngOnInit(): void {

    this.registrationForm = this.formBuilder.group({

      Username: ['', [Validators.required, Validators.minLength(3)]],

      Email: ['', [Validators.required, Validators.email]],

      Password: ['', [Validators.required, Validators.minLength(8)]]

  });

  }

  // convenience getter for easy access to form fields

  get f() { return this.registrationForm.controls; }

  // on submit

  register(){

      console.log(this.registrationForm.value);

  }

}

To build the form,

1. We need the FormBuilder to build the form and also we will use router for navigation and hence we do DI for both.
2. We need the form builder to match the form with the backend so that it can be controlled from TS. The Names of the control should be same as formControlName as used in control in the template. We can also put the validations.
3. We are using the f function using getter to access form controls in the template

Put the below code in the template

1. [formGroup]=”registrationForm” // as in TS
2. formControlName=”Username” //as in TS
3. f is the function which returns the form controls from the TS
4. using [ngClass], the control is applied with is-invalid class depending upon logic on right hand side
5. dirty means control is touched
6. The validation is invoked once control is dirty and if control has errors
7. The register button is enabled only after form is valid
8. The register button is of type submit which fires form’s submit event

<div class="ml-3 mw-50 mr-3" style="max-width: 400px;">

    <h2>Register</h2>

    <hr />

    <form [formGroup]="registrationForm" (submit)="register()">

        <div class="form-group mb-3">

            <label for="Username">Username</label>

            <input type="text" formControlName="Username" class="form-control"

                [ngClass]="{ 'is-invalid': f.Username.dirty && f.Username.errors }" />

            <div \*ngIf="f.Username.dirty && f.Username.errors" class="invalid-feedback">

                <div \*ngIf="f.Username.errors.minlength">Username should be more than 3 char</div>

                <div \*ngIf="f.Username.errors.required">Username is required</div>

            </div>

        </div>

        <div class="form-group mb-3">

            <label for="Email">Email</label>

            <input type="email" formControlName="Email" class="form-control"

                [ngClass]="{ 'is-invalid': f.Email.dirty && f.Email.errors }" />

            <div \*ngIf="f.Email.dirty && f.Email.errors" class="invalid-feedback">

                <div \*ngIf="f.Email.errors.required">Email is required</div>

                <div \*ngIf="f.Email.errors.email">Email is of invalid format</div>

            </div>

        </div>

        <div class="form-group">

            <label for="Password">Password</label>

            <input type="password" formControlName="Password" class="form-control"

                [ngClass]="{ 'is-invalid': f.Password.dirty && f.Password.errors }" />

            <div \*ngIf="f.Password.errors" class="invalid-feedback">

                <div \*ngIf="f.Password.errors.required">Password is required</div>

                <div \*ngIf="f.Password.errors.minlength">Password must be at least 8 characters</div>

            </div>

        </div>

        <div class="form-group mt-2">

            <button class="btn btn-warning float-right" (click)="registrationForm.reset()">

                Reset

            </button>

            <a routerLink="/login" class="btn btn-success float-right mr-1">Login</a>

            <button type="submit" [disabled]="!registrationForm.valid" class="btn btn-primary float-right mr-1">

                Register

            </button>

        </div>

    </form>

</div>

# Login component

Let us design the login component.

Here there are 2 validations.

1. Username – is required field and should be an email
2. Password – is required and has to be minimum 8 characters’ long

Business logic:

1. User will login / signup using same component
2. If user does not exist, he will be signed up and logged in automatically

In the login template:

1. Put a form reference #f=”ngForm”
2. Tie every control to model [(ngModel)]=”model.Email” #Email=”ngModel”
3. Check validation when we touch control i.e. on dirty
4. Have the submit event tied to login
5. The login button is enabled when form becomes valid
6. The template form has a bug with button that any button click causes the submit to fire, hence we have provided type=submit for login button and type=button explicitly for the reset button to convey that reset button is not submit.

<!-- main app container -->

<div class="ml-3 mr-3" style="max-width: 400px;">

    <h2>Login</h2>

    <hr />

    <form name="form" #f="ngForm" (submit)="login()">

        <div class="form-group mt-2">

            <label for="Email">Email</label>

            <input type="Email" class="form-control" name="Email" [(ngModel)]="model.Email"

            #Email="ngModel"

                [ngClass]="{ 'is-invalid': Email.dirty && Email.invalid }" required email />

            <div \*ngIf="Email.dirty && Email.invalid" class="invalid-feedback">

                <div \*ngIf="Email.errors.required">Email is required</div>

                <div \*ngIf="Email.errors.email">Email must be a valid email address</div>

            </div>

        </div>

        <div class="form-group mt-2">

            <label for="Password">Password</label>

            <input type="password" class="form-control" name="Password" [(ngModel)]="model.Password"

            #Password="ngModel"

                [ngClass]="{ 'is-invalid': Password.dirty && Password.invalid }" required minlength="8" />

            <div \*ngIf="Password.dirty && Password.invalid" class="invalid-feedback">

                <div \*ngIf="Password.errors.required">Password is required</div>

                <div \*ngIf="Password.errors.minlength">Password must be at least 8 characters</div>

            </div>

        </div>

        <div class="form-group mt-2">

            <button type="button" class="btn btn-warning float-right" (click)="f.form.reset()">

                Reset

            </button>

            <button type="submit" class="btn btn-primary float-right mr-1" [disabled]="f.form.invalid">Login</button>

        </div>

    </form>

</div>

In the TS of Login

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

@Component({

  selector: 'app-login',

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

  model: any = {};

  login(){

    console.log(this.model);

  }

  constructor() { }

  ngOnInit(): void {

  }

}

# Create the buy component

We can reuse the login component logic to create buy component.

Here, we have copy pasted the buy component code and made following changes

1. You will have Symbol which is select field with only required field validator
2. You will have Quantity which is number type with required field validator
3. Only when the form is valid the buy button will be enabled
4. Reset button will reset the form

In the template, put the below html

<!-- main app container -->

<div class="ml-3 mr-3" style="max-width: 400px;">

    <h2>Buy</h2>

    <hr />

    <form name="form" #f="ngForm" (submit)="buy()">

        <div class="form-group mt-2">

            <label for="Symbol">Symbol</label>

            <select class="form-control" name="Symbol" id="Symbol" [(ngModel)]="model.Symbol"

            #Symbol="ngModel" required>

                <option value="Infy">Infy</option>

                <option value="TCS">TCS</option>

                <option value="WIPRO">WIPRO</option>

                <option value="Persistent">Persistent</option>

              </select>

        </div>

        <div class="form-group mt-2">

            <label for="Quantity">Quantity</label>

            <input type="number" class="form-control" name="Quantity" [(ngModel)]="model.Quantity"

            #Quantity="ngModel"

                [ngClass]="{ 'is-invalid': Quantity.dirty && Quantity.invalid }" required />

            <div \*ngIf="Quantity.dirty && Quantity.invalid" class="invalid-feedback">

                <div \*ngIf="Quantity.errors.required">Quantity is required</div>

            </div>

        </div>

        <div class="form-group mt-2">

            <button type="button" class="btn btn-warning float-right" (click)="f.form.reset()">

                Reset

            </button>

            <button type="submit" class="btn btn-primary float-right mr-1" [disabled]="f.form.invalid">Buy</button>

        </div>

    </form>

</div>

In the typescript side,

  model: any = {};

  buy() {

    console.log(this.model);

  }

# Creating the positions component

We’ll first create the template mock html in the positions component.

<div class="ml-3 mr-3" style="max-width: 400px;">

    <h2>Positions</h2>

    <hr />

    <div class="col-md-12">

        <button class="btn btn-warning float-right">Refresh</button>

    </div>

    <table class="table table-striped table-light">

        <thead>

            <tr>

                <th scope="col">TxID</th>

                <th scope="col">Symbol</th>

                <th scope="col">Quantity</th>

                <th scope="col">NPL</th>

            </tr>

        </thead>

        <tbody>

            <tr>

                <th scope="row">1</th>

                <td>ITC</td>

                <td>34</td>

                <td class="text-success">+567.78</td>

            </tr>

        </tbody>

    </table>

</div>

# Creating the statement component

The statement component will be nearly same as the positions component

<div class="ml-3 mr-3" style="max-width: 400px;">

    <h2>Statement</h2>

    <hr />

    <div class="col-md-12">

        <button class="btn btn-warning float-right">Export</button>

        <h4 class="text-danger float-right mr-3">-367.78</h4>

    </div>

    <table class="table table-striped table-light">

        <thead>

            <tr>

                <th scope="col">TxID</th>

                <th scope="col">Symbol</th>

                <th scope="col">Quantity</th>

                <th scope="col">PL</th>

            </tr>

        </thead>

        <tbody>

            <tr>

                <th scope="row">1</th>

                <td>ITC</td>

                <td>34</td>

                <td class="text-danger">-367.78</td>

            </tr>

        </tbody>

    </table>

</div>

# Positions popup for the closing the position – Bootstrap Modal

To have a popup we will create a popup component which can

1. Receive the title – using @Input ; m-title
2. Receive the content to be shown using <ng-content>
3. Receive the ID for the popup using @Input; m-id

Create the component ng g c my-modal

In the TypeScript file:

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

@Component({

  selector: 'app-my-modal',

  templateUrl: './my-modal.component.html',

  styleUrls: ['./my-modal.component.css']

})

export class MyModalComponent implements OnInit {

  // specify the input parameters

  @Input('m-title') ModalTitle: string="Default title";

  @Input('m-id') ModalID : string="mymodal";

  constructor() { }

  ngOnInit(): void {

  }

}

In the Template file,

<div class="modal fade" [id]="ModalID" tabindex="-1" aria-labelledby="ModalLabel" aria-hidden="true">

  <div class="modal-dialog">

    <div class="modal-content">

      <div class="modal-header">

        <h5 class="modal-title" id="ModalLabel">{{ModalTitle}}</h5>

        <button type="button" class="close" data-dismiss="modal" aria-label="Close">

          <span aria-hidden="true">&times;</span>

        </button>

      </div>

      <div class="modal-body">

        <ng-content></ng-content>

      </div>

      <div class="modal-footer">

        <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>

      </div>

    </div>

  </div>

</div>

Here you will see,

1. We are using ng-content to show the modal content which will be projected
2. Id is attribute binded with input property
3. Modal title is string interpolation binded with the input property

In any component, we can use this modal component like below:

Here the data-target is the ID of the modal which should be same as m-id without #

 <td data-toggle="modal" data-target="#ITC">ITC

                    <app-my-modal m-id="ITC" m-title="Injected">

                        This is it

                    </app-my-modal>

                </td>

We need to curate the internal content which will ask whether you want to close the ITC position or not

<td data-toggle="modal" data-target="#ITC">ITC

                    <app-my-modal m-id="ITC" m-title="ITC">

                        Close position in ITC? <br/>

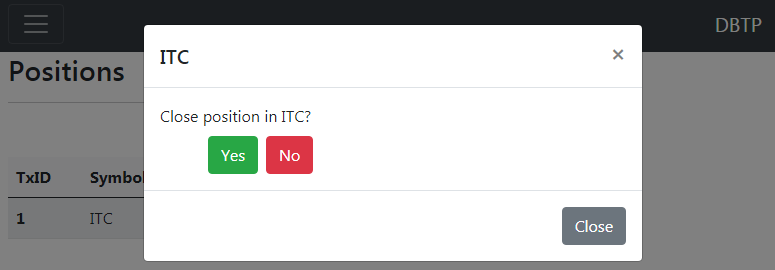
                        <button class="btn btn-success mt-2 mr-2 ml-5">Yes</button>

                        <button class="btn btn-danger mt-2" data-dismiss="modal">No</button>

                    </app-my-modal>

                </td>

This will give us an output



We will use this in our positions component. To close the modal use data-dismiss=”modal”

Building the functionality

Ng g s dbtp-utility



Because we will deal with functions and models, let us also create an interface to store all the models

The interface can be named as IDBTP.ts

export interface User{

    username : string,

    email: string,

    password: string

}

To do the HTTP related operations, we need HTTPClientModule imported in the App module

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

 imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule, //for template-drive forms

    ReactiveFormsModule, //for the reactive forms

    HttpClientModule //for the HTTP operations

  ],

In the dbtp-utility service, we will have to do DI for the HttpClient

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

import {User} from '../IDBTP';

import {HttpClient} from "@angular/common/http";

import { Observable } from 'rxjs';

@Injectable({

  providedIn: 'root'

})

export class DptpUtilityService {

  constructor(private myHttpClient:HttpClient) { }

  //registration

  RegisterUser(user: User): Observable<any>{

    // registration is a POST operation

   return this.myHttpClient

              .post('https://mycrudops.herokuapp.com/users',user);

  }

}

Note that the RegisterUser returns an observable.

# How to export as excel in the statement component

https://stackblitz.com/edit/angular8-json-to-csv

Guarding the routes

As of now we can go to any route login, positions, statements without any limitations. Ideally, unless you login, you should not be able to traverse the application. This is covered with something called as guards in Angular.