File

`src/app/components/register/register.component.ts`

Description

Component to register users

Implements

`OnInit`

Metadata

selector	`app-register`
styleUrls	`./register.component.scss`
templateUrl	`./register.component.html`

Index

Properties

- Public datosCorrectos
- Public hide
- Public message
- Public registerForm
- Public user

Methods

- getErrorMessage
- ngOnInit
- passwordsShouldMatch
- signUp

Constructor

`constructor(formBuilder: FormBuilder, _userService: UserService, router: Router)`

Defined in src/app/components/register/register.component.ts:36

Constructor in which we inject hero service, formBuilder and router service

Parameters :

Name	Type	Optional
formBuilder	`FormBuilder`	No
_userService	`UserService`	No
router	`Router`	No

Methods

getErrorMessage

`getErrorMessage(dato)`

Defined in src/app/components/register/register.component.ts:81

function to control error messages

Parameters :

Name	Optional
dato	No

Returns : `string`

message

ngOnInit

`ngOnInit()`

Defined in src/app/components/register/register.component.ts:52

Start when the component inits

Returns : `void`

passwordsShouldMatch

`passwordsShouldMatch(control: AbstractControl)`

Defined in src/app/components/register/register.component.ts:110

Validation to verify that the passwords match

Parameters :

Name	Type	Optional
control	`AbstractControl`	No

Returns : `any`

true or null

signUp

`signUp()`

Defined in src/app/components/register/register.component.ts:129

SignUp function

Returns : `void`

Properties

Public datosCorrectos

Type : `boolean`

Defined in src/app/components/register/register.component.ts:34

variable to check if the register works

Public hide

Default value : `true`

Defined in src/app/components/register/register.component.ts:36

Public message

Type : `string`

Defined in src/app/components/register/register.component.ts:30

variable to save message info about register

Public registerForm

Type : `FormGroup`

Defined in src/app/components/register/register.component.ts:22

to add FormGroup

Public user

Type : User

Defined in src/app/components/register/register.component.ts:26

variable to save user in it

```
import { Component, OnInit } from '@angular/core'
import { FormBuilder } from '@angular/forms'
import { Validators } from '@angular/forms'
import { FormGroup, FormControl, AbstractControl } from '@angular/forms'
import { User } from '../../models/user'
import { UserService } from 'src/app/services/user.service'
import { Router } from '@angular/router'

/**
 * Component to register users
 */
@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.scss']
})

export class RegisterComponent implements OnInit {
  /**
  * to add FormGroup
  */
  public registerForm: FormGroup
  /**
   * variable to save user in it
   */
  public user: User
  /**
   * variable to save message info about register
   */
  public message: string
  /**
   * variable to check if the register works
   */
  public datosCorrectos: boolean

  public hide = true;

  /**
   * Constructor in which we inject hero service, formBuilder and router service
   */
  constructor(
    private formBuilder: FormBuilder,
    private _userService: UserService,
    private router: Router
  ) {
    this.user = new User(0, '', '', '', '', '', null, '', false)
  }

  /**
   * Start when the component inits
   */
  ngOnInit() {
    this.datosCorrectos = true
    this.registerForm = this.formBuilder.group({
      alias: [
        '',
        [Validators.required, Validators.minLength(3), Validators.maxLength(30)]
      ],
      email: [
        '',
        [
          Validators.required,
          Validators.email,
          Validators.minLength(6),
          Validators.maxLength(30)
        ]
      ],
      password: [
        '',
        [Validators.required, Validators.minLength(6), Validators.maxLength(30)]
      ],
      confirmPass: ['', [Validators.required, this.passwordsShouldMatch]]
    })
  }

  /**
   * function to control error messages
   * @param {string} dato
   * @returns message
   */
  getErrorMessage(dato): string {
    var result: string
    if (this.registerForm.controls[dato].hasError('required')) {
      return (result = 'This information is required')
    } else if (this.registerForm.controls[dato].hasError('minlength')) {
      if (dato === 'alias') {
        return (result = 'You must enter at least 3 characters')
      } else {
        return (result = 'You must enter at least 6 characters')
      }
    } else if (this.registerForm.controls[dato].hasError('maxlength')) {
      return (result = 'The maximum of characters is 30')
    } else if (
      this.registerForm.controls[dato].hasError('email') &&
      dato === 'email'
    ) {
      return (result = 'You have to enter a valid email')
    } else if (dato === 'confirmPass') {
      return (result = 'Passwords do not match')
    } else {
      return (result = '')
    }
  }

  /**
   * Validation to verify that the passwords match
   * @param {any} control
   * @returns true or null
   */
  passwordsShouldMatch(control: AbstractControl): any {
    if (control && (control.value !== null || control.value !== undefined)) {
      const password2Value = control.value
      const passControl = control.root.get('password')
      if (passControl) {
        const passValue = passControl.value
        if (passValue !== password2Value) {
          return {
            isError: true
          }
        }
      }
    }
    return null
  }

  /**
   * SignUp function
   */
  signUp() {
    this._userService.RegisterUser(this.user).subscribe(
      response => {
        console.log('Register correctly')
        this.message = 'Register correctly'
        // Clean the user
        this.user = new User(0, '', '', '', '', '', new Date(0), '', false)
        this.registerForm.reset()
        this.router.navigate(['/login'])
        this.datosCorrectos = true
      },
      error => {
        this.datosCorrectos = false

        var element = document.getElementById('ErroInfo');
        element.classList.remove('d-none');
        setTimeout('document.getElementById("ErroInfo").classList.add("d-none")', 3500);

        if (error.status === 400) {
          this.message = 'User already exists'
          console.log(error.status)
          console.log(this.message)
        } else {
          console.log(error.status)
          this.message = 'Registration failed'
          console.log(this.message)
        }
      }
    )
  }
}
```

```
<div id="form">
  <h1>Join us</h1>
  <form [formGroup]="registerForm" (ngSubmit)="signUp()">
    <div class="fildP">
      <div class="logo">
        <img class="icon2" src="../../../assets/img/userP.svg" alt="" />
      </div>
      <mat-form-field appearance="legacy">
        <mat-label>Alias</mat-label>
        <input
          matInput
          type="text"
          formControlName="alias"
          placeholder="Alias"
          [(ngModel)]="user.alias"
        />
        <mat-error *ngIf="!registerForm.controls['alias'].valid">{{
          getErrorMessage("alias")
        }}</mat-error>
      </mat-form-field>
    </div>
    <br />
    <div class="fildP">
      <div class="logo">
        <img class="icon" src="../../../assets/img/email.svg" alt="" />
      </div>
      <mat-form-field appearance="legacy">
        <mat-label>Email</mat-label>
        <input
          matInput
          type="email"
          formControlName="email"
          placeholder="Email"
          [(ngModel)]="user.email"
        />
        <mat-error *ngIf="!registerForm.controls['email'].valid">{{
          getErrorMessage("email")
        }}</mat-error>
      </mat-form-field>
    </div>
    <br />
    <div class="fildP">
      <div class="logo">
        <img class="icon2" src="../../../assets/img/lock.svg" alt="" />
      </div>
      <mat-form-field appearance="legacy">
        <mat-label>Password</mat-label>
        <input
          matInput
          type="password"
          formControlName="password"
          placeholder="Password"
          [(ngModel)]="user.password"
        />
        <mat-error *ngIf="!registerForm.controls['password'].valid">
          {{ getErrorMessage("password") }}</mat-error
        >
      </mat-form-field>
    </div>
    <br />
    <div class="fildP">
      <div class="logo">
        <img class="icon2" src="../../../assets/img/lock.svg" alt="" />
      </div>
      <mat-form-field appearance="legacy">
        <mat-label>Confirm Password </mat-label>
        <input
          matInput
          type="password"
          formControlName="confirmPass"
          placeholder="Confirm Password          "
        />
        <mat-error
          *ngIf="
            registerForm.get('confirmPass').invalid &&
            registerForm.get('confirmPass').touched &&
            registerForm.get('password').touched
          "
        >
          {{ getErrorMessage("confirmPass") }}</mat-error
        >
      </mat-form-field>
    </div>
    <div id="ErroInfo" class="d-none">
      <p class="errorM">{{ this.message }}</p>
    </div>
    <br />
    <button type="submit" class="UnfollowB">
      <span>SIGN UP</span>
    </button>
  </form>
  <p class="NotRegister">
    Are you already registered?
    <b><a class="registrate" [routerLink]="['/login']">Log in</a></b>
  </p>
</div>
```

./register.component.scss

```
#form {
  background-color: #272727;
  border: #00a23d 2px solid;
  box-shadow: 0 0 21px 9px rgba(8, 17, 10, 0.63);
  display: flex;
  flex-direction: column;
  font-family: "B612";
  align-items: center;
  justify-content: center;
  text-align: center;
  margin: 4em 1em 2em 1em;
  padding: 2em 0em 2em 0em;
  height: 40%;
}

// button
$bg: #272727;
$fg: #f3d403;
$border-width: 0.2rem;
$corner-size: 3rem;
$dur: 0.3s;

.followB,
.UnfollowB {
  margin: 0 0.5vw 0 0.5vw;
  outline: none;

  font-family: "B612";
  letter-spacing: 0.02rem;
  cursor: pointer;
  background: transparent;
  border: $border-width solid currentColor;
  padding: 0.5rem 6.5rem;
  font-size: 1rem;
  color: $fg;
  position: relative;
  transition: color $dur;

  &:hover {
    color: #cc4224;
    &::before {
      width: 0;
    }
    &::after {
      height: 0;
    }
  }
  &:active {
    border-width: $border-width / 2;
  }

  span {
    position: relative;
    z-index: 2;
  }
  &::before,
  &::after {
    content: "";
    position: absolute;
    background: $bg;
    z-index: 1;
    transition: all $dur;
  }
  &::before {
    width: calc(100% - #{$corner-size});
    height: calc(101% + #{$border-width * 2});
    top: -$border-width;
    left: 50%;
    transform: translateX(-50%);
  }
  &::after {
    height: calc(100% - #{$corner-size});
    width: calc(101% + #{$border-width * 2});
    left: -$border-width;
    top: 50%;
    transform: translateY(-50%);
  }
}

.NotRegister {
  padding-top: 1.5em;
  color: #00a23d;
}

#form ::ng-deep.mat-form-field-appearance-legacy .mat-form-field-flex {
  background: rgb(223, 221, 221);
}
#form ::ng-deep.mat-form-field-infix {
  width: 12em;
  font-family: "B612";
}

#form ::ng-deep.mat-form-field-label-wrapper {
  left: 1em !important;
}

#form ::ng-deep.mat-input-element {
  margin-left: 1em !important;
}

.errorM {
  color: #cc4224;
}

.fildP {
  display: flex;
  flex-direction: row;
  padding-left: 1em;
}

.icon {
  height: 1.2em;
}

.icon2 {
  height: 1.5em;
}

.logo {
  background-color: #00a23d;
  height: 2.82em;
  width: 3em;
  display: flex;
  align-items: center;
  justify-content: center;
  text-align: center;
}

.mat-icon {
  padding-left: 1em !important;
  color: white !important;
}
h1 {
  padding-bottom: 1vw;
  color: #cc4224;
  font-family: "Bangers", cursive;
  text-decoration: underline;
  -webkit-text-decoration-color: #00a23d; /* Safari */
  text-decoration-color: #00a23d;
}
@media (min-width: 992px) {
  #form {
    margin: 4em 26% 2em 26%;
  }

  #form::ng-deep.mat-form-field-infix {
    width: 25em;
  }

  .UnfollowB {
    padding: 0.5rem 10.5rem;
  }

  .fildP {
    padding-left: 0;
  }
}
```

Legend
Html element

Component
Html element with directive

result-matching ""

No results matching ""