PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 11

0 Comments

BY SAI UPADHYAYULA | March 28, 2020

Reddit clone with Spring boot and Angular
#11
Implement User Registration in Angular app

Share Tweet Pin

Welcome to Full Stack Reddit Clone with Spring boot and Angular series – Part 11 , and in this article, we will see how to implement Signup functionality in our Angular application.

In Part 10 we saw how to document our REST APIs so that we can easily look it up whenever we need to check any details of the API.

In Part 9 we developed the header bar for our application, and the header bar contains links to Login and Signup, let's start off by creating the components for both of them.

If you are a visual learner like me, don't forget to checkout the Video Version of this tutorial.

Thank you for visiting. You can now buy me a coffee!

Download Source Code

> **Source code for Angular application** – https://github.com/SaiUpadhyayula/angular-reddit-clone
>
> **Source code for Spring boot backend** –https://github.com/SaiUpadhyayula/spring-reddit-clone

Generating Sign-up Components

Open the terminal and type below commands:

After that copy the below HTML code into the **sign-up.component.html**

sign-up.component.html

```
<div class="register-section">
    <div class="row justify-content-center">
        <div class="col-md-3"></div>
        <div class="col-md-6">
            <div class="card">
                <div class="card-header" style="text-align: center">
                    <h4>Register</h4>
                </div>
                <div class="card-body">
                    <form>
```

```
                        <div class="form-group row">
                            <label for="email_address" class="col-md-4 col-form-label text-md-right">E-Mail
                                Address</label>
                            <div class="col-md-6">
                                <input type="text" id="email_address" class="form-control" name="email-
address" required
                                    autofocus>
                            </div>
                        </div>

                        <div class="form-group row">
                            <label class="col-md-4 col-form-label text-md-right">User Name</label>
                            <div class="col-md-6">
                                <input type="text" class="form-control" required autofocus>
                            </div>
                        </div>

                        <div class="form-group row">
                            <label for="password" class="col-md-4 col-form-label text-md-
right">Password</label>
                            <div class="col-md-6">
                                <input type="password" id="password" class="form-control" name="password"
required>
                            </div>
                        </div>

                        <span class="col-md-6 offset-md-4">
                            <button type="submit" class="sign-up">
                                Sign Up
                            </button>
                            <span style="padding-left: 15px">Existing user? <a routerLink="/login">Log In</a>
</span>
                        </span>
                    </form>
                </div>
            </div>
        </div>
        <div class="col-md-3"></div>
    </div>
</div>
```

Let’s not forget the CSS code.

sign-up.component.css

```
.register-section{
    margin: 100px;
}

.sign-up {
    background-color: #0079D3;
    border-color: #0079D3;
    color: aliceblue;
    fill: #0079D3;
    border: 1px solid;
    border-radius: 4px;
    text-align: center;
    letter-spacing: 1px;
    text-decoration: none;
    font-size: 12px;
    font-weight: 700;
    letter-spacing: .5px;
    line-height: 24px;
    text-transform: uppercase;
    padding: 3px 16px;
    opacity: 1;
    }
```

Now that's done, we have to create a route for **Signup** page, when we click on the **Signup** button on the header, we have to navigate to the corresponding page.

Configure Routes for Signup page

Open the **app-routing.module.ts** which is the place we declare all the routes in our angular application.

As this module file is already a generated one, you can just update the routes array with the route information for **sign-up**.

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { SignUpComponent } from './auth/sign-up/sign-up.component';


const routes: Routes = [
   { path: 'sign-up', component: SignUpComponent }
];

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

Now open the application by going to the URL – http://localhost:4200 and click on the **Signup** button, then you should be navigated to the corresponding pages.

Handling Form Input using Reactive Forms

We created our HTML markup code which contains the forms which take the user credentials. We will use Reactive Forms in our project to handle form input.

Let's declare the **ReactiveFormsModule** inside our **app.module.ts** file.

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { SignUpComponent } from './auth/sign-up/sign-up.component';
import { ReactiveFormsModule } from '@angular/forms';

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

Replace the existing code in your **sign-up.component.ts** file with the below code.

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

@Component({
  selector: 'app-sign-up',
  templateUrl: './sign-up.component.html',
  styleUrls: ['./sign-up.component.css']
})
export class SignUpComponent implements OnInit {

  signupForm: FormGroup;

  constructor() { }

  ngOnInit() {
    this.signupForm = new FormGroup({
      username: new FormControl(null),
      email: new FormControl(null),
      password: new FormControl(null)
    });
  }
}
```

- On Line-11, we declared a variable called as **signupForm** which is of type FormGroup. This is the starting point to handle form-input in our Login Page.
- Next inside the **ngOnInit()** we initialized this **signupForm** variable by assigning it to a new FormGroup which takes two **FormControl** objects **email**, **username,** and **password**.

Adding Validations to Reactive Forms

Now let's add validations to our Reactive forms inside our **Signup** component.

First, update the FormControl constructor arguments inside both **sign-up.component.ts** files.

```
    this.signupForm = new FormGroup({
      username: new FormControl('', Validators.required),
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', Validators.required)
    });
```

We have added **Validators.required** for each field and especially for the **email** field inside the **signupForm** we also added **Validators.email**

Let's also sync-up our Components and HTML files with the formControlNames.

sign-up.component.html

```
<div class="register-section">
  <div class="row justify-content-center">
    <div class="col-md-3"></div>
    <div class="col-md-6">
      <div class="card">
        <div class="card-header" style="text-align: center">
          <h4>Register</h4>
```

Thank you for visiting. You can now buy me a coffee!

```
        </div>
        <div class="card-body">
          <form [formGroup]="signupForm" (ngSubmit)="signup()">
            <div class="form-group row">
              <label for="email_address" class="col-md-4 col-form-label text-md-right">E-Mail
                Address</label>
              <div class="col-md-6">
                <input type="text" [formControlName]="'email'" id="email_address" class="form-control"
name="email-address">
                <span *ngIf="!signupForm.get('email').valid && signupForm.get('email').touched" class="help-
block">Please enter a valid email</span>
              </div>
            </div>

            <div class="form-group row">
              <label class="col-md-4 col-form-label text-md-right">User Name</label>
              <div class="col-md-6">
                <input type="text" [formControlName]="'username'" class="form-control">
                <span *ngIf="!signupForm.get('username').valid && signupForm.get('username').touched"
class="help-block">Please enter a valid username</span>
              </div>
            </div>

            <div class="form-group row">
              <label for="password" class="col-md-4 col-form-label text-md-right">Password</label>
              <div class="col-md-6">
                <input type="password" [formControlName]="'password'" id="password" class="form-control"
name="password">
                <span *ngIf="!signupForm.get('password').valid && signupForm.get('password').touched"
class="help-block">Please enter a valid password</span>
              </div>
            </div>

            <span class="col-md-6 offset-md-4">
              <button type="submit" class="sign-up">
                Sign Up
              </button>
              <span style="padding-left: 15px">Existing user? <a routerLink="/login">Log In</a></span>
            </span>
          </form>
        </div>
      </div>
    </div>
    <div class="col-md-3"></div>
  </div>
</div>
```

- You can see in lines 16,24,32 in the **sign-up.component.html** file, we added the sections to display an error message when the corresponding input is empty.
- We have also synced up our HTML files with the formControlNames from the components.

Let's open the styles.css file and add the below code to make the borders of the input fields **red** when the validation is failing.

```
/* You can add global styles to this file, and also import other style files */
input.ng-invalid.ng-touched{
    border: 1px solid red;
}
```

This is how the validation errors on the input fields look like in the browser.

Signup page

Thank you for visiting. You can now buy me a coffee!

sign-up.component.ts

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

@Component({
  selector: 'app-sign-up',
  templateUrl: './sign-up.component.html',
  styleUrls: ['./sign-up.component.css']
})
export class SignUpComponent implements OnInit {

  signupForm: FormGroup;

  constructor() { }

  ngOnInit() {
    this.signupForm = new FormGroup({
      username: new FormControl('', Validators.required),
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', Validators.required)
    });
  }

  signup() {

  }

}
```

Referring to REST API Documentation

Now let's refer to the REST API documentation we created in the previous article, by looking at the below images we understand what kind of Payload we have to send to the backend as part of the Signup REST API call.

Signup REST API

Prepare the Signup Request

Let’s create the classes **SignupRequestPayload** with fields you see in the above image.

signup-request.payload.ts

```
export interface SignupRequestPayload {
    username: string;
    email: string;
    password: string;
}
```

Let’s initialize the payload model inside our **ngOnInit()** methods of the signup-component

sign-up.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { SignupRequestPayload } from './signup-request.payload';

@Component({
  selector: 'app-sign-up',
  templateUrl: './sign-up.component.html',
  styleUrls: ['./sign-up.component.css']
})
export class SignUpComponent implements OnInit {

  signupRequestPayload: SignupRequestPayload;
  signupForm: FormGroup;

  constructor() {
    this.signupRequestPayload = {
      username: '',
      email: '',
      password: ''
    };
  }

  ngOnInit() {
    this.signupForm = new FormGroup({
      username: new FormControl('', Validators.required),
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', Validators.required)
    });
  }

  signup() {
    this.signupRequestPayload.username = this.signupForm.get('username').value;
    this.signupRequestPayload.email = this.signupForm.get('email').value;
    this.signupRequestPayload.password = this.signupForm.get('password').value;

  }

}
```

As you can see we initialized the **SignupRequestPayload** with the values from **signupForm**, now its time to make the backend call to our REST API. Let's create a service class called **AuthService** to make HTTP calls to our REST API.

auth.service.ts

```
  import { Injectable } from '@angular/core';
  import { SignupRequestPayload } from '../sign-up/signup-request.payload';
  import { Observable } from 'rxjs';
  import { HttpClient } from '@angular/common/http';

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

    constructor(private http: HttpClient) { }

    signup(signupRequestPayload: SignupRequestPayload): Observable<any> {
      return this.http.post('http://localhost:8080/api/auth/signup', signupRequestPayload);
    }
  }
```

Inside **auth.service.ts** file, we injected the **HttpClient** class and we are making an HTTP call to our Signup REST API. Notice that we are returning back an Observable from our signup method.

To be able to use **HttpClient** in our project without any errors, we have to add the **HttpClientModule** to our AppModule.

Before doing that, let's create a separate module for our **auth** related functionality called **auth.module.ts**

auth.module.ts

```
  import { NgModule } from '@angular/core';
  import { CommonModule } from '@angular/common';
```

Thank you for visiting. You can now buy me a coffee!

```
import { SignUpComponent } from './sign-up/sign-up.component';
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    SignUpComponent
  ],
  imports: [
    CommonModule,
    ReactiveFormsModule
  ]
})
export class AuthModule { }
```

And this is how our **app.module.ts** file looks like with **AuthModule** and **HttpClientModule** definitions.

app.module.ts

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { AuthModule } from './auth/auth.module';
import { HttpClientModule } from '@angular/common/http';

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

Now let's inject the **AuthService** into our **sign-up.component.ts**

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { SignupRequestPayload } from './signup-request.payload';
import { AuthService } from '../shared/auth.service';

@Component({
  selector: 'app-sign-up',
  templateUrl: './sign-up.component.html',
  styleUrls: ['./sign-up.component.css']
})
export class SignUpComponent implements OnInit {

  signupRequestPayload: SignupRequestPayload;
  signupForm: FormGroup;

  constructor(private authService: AuthService) { }

  ngOnInit() {
    this.signupForm = new FormGroup({
      username: new FormControl('', Validators.required),
      email: new FormControl('', [Validators.required, Validators.email]),
      password: new FormControl('', Validators.required)
    });
  }

  signup() {
    this.signupRequestPayload.username = this.signupForm.get('username').value;
    this.signupRequestPayload.email = this.signupForm.get('email').value;
    this.signupRequestPayload.password = this.signupForm.get('password').value;

    this.authService.signup(this.signupRequestPayload).subscribe(() => {
      console.log('Signup Successful');
    }, () => {
      console.log('Signup Failed');
    });
  }

}
```

So you can see that we have first injected the **AuthService** into our component and we used this inside our signup() method, we subscribed to the Observable which is returned from **AuthService.signup()**, if we receive success response then we print **Signup Successful** if not we print **Signup Failed**

Trying our luck

So now its time to test the Signup Functionality in our application, make sure you have your backend up and running, start the frontend application using ng serve command and go to http://localhost:4200/sign-up

As you can see we are receiving an error:

> Access to XMLHttpRequest at 'http://localhost:8080/api/auth/signup' from origin 'http://
> blocked by CORS policy: Response to preflight request doesn't pass access control che
> Origin' header is present on the requested resource.

We did not configure CORS policy in the backend application, let's do it now.

Adding CORS Policy to our Backend

Open our Springboot project and inside the **config** package, create a class called **WebConfig**, copy the below code and restart the application.

WebConfig.java

```
package com.programming.techie.springredditclone.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry corsRegistry) {
        corsRegistry.addMapping("/**")
                .allowedOrigins("*")
                .allowedMethods("*")
                .maxAge(3600L)
                .allowedHeaders("*")
                .exposedHeaders("Authorization")
                .allowCredentials(true);
    }
}
```

Testing our luck again!

So now let's try to signup again with new credentials and see if the signup functionality is working or not.

If you remember, we were sending Activation Emails right after completing user registration. Let's check if we received the emails or not by logging into our MailTrap account.

Conclusion

So we will end this article here, and in the next article, we will build the Login Functionality in our application.

I hope you are learning some new things building the Fullstack Springboot & Angular application like this.

Please share this article with your friends and colleagues who may find this helpful.

See you in the next blogpost, until then **Happy Coding Techies** 🙂

About the author

Sai Upadhyayula