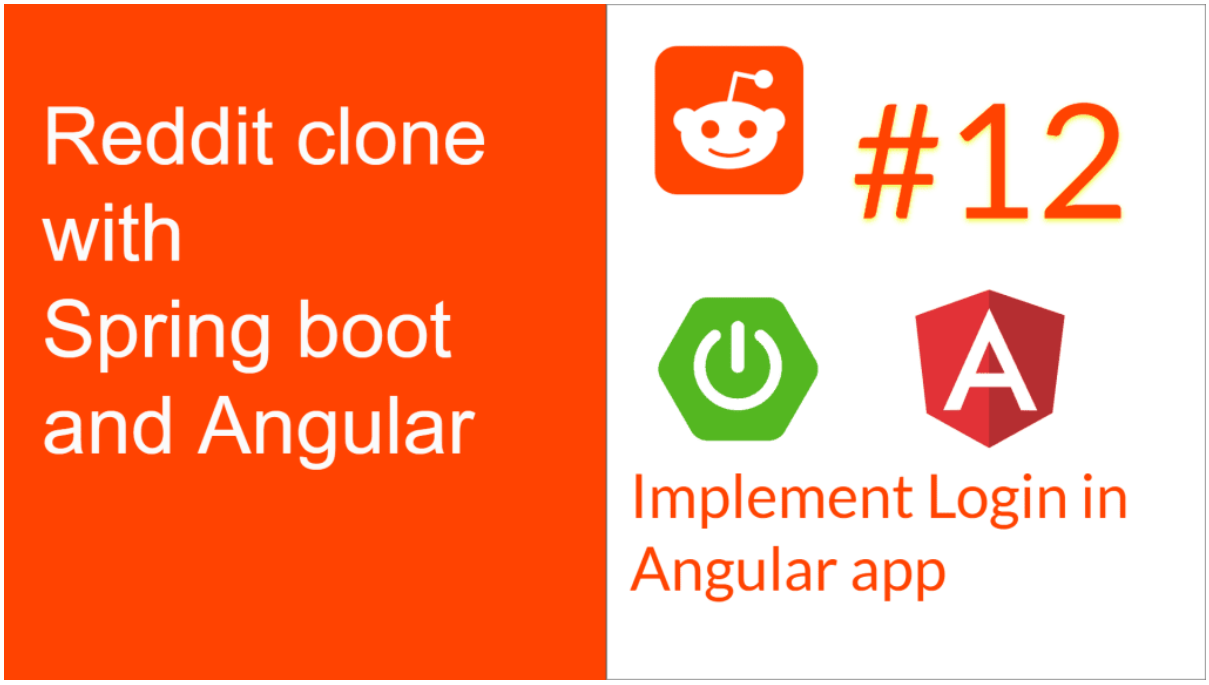
PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 12

0 Comments

BY SAI UPADHYAYULA | April 3, 2020



Welcome to Full Stack Reddit Clone with Spring boot and Angular – Part 12. In Part 11, we saw how to implement Signup functionality in our Angular application.

In this article, we will see how to implement Login in our angular application. We will see how to handle JWT and Refresh Tokens on the client-side.

If you are a visual learner like me, check out the below Video Tutorial

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 Login Component

Open the terminal and type below commands:

After that copy the below HTML code into the **login.component.html**

```
<div class="login-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>Login</h4></div>
                <div class="card-body">
                    <form [formGroup]="loginForm" (ngSubmit)="login()">
                        <div class="form-group row">
                            <label for="user_name" class="col-md-4 col-form-label text-md-
right">Username</label>
                            <div class="col-md-6">
                                <input type="text" id="user_name" class="form-control"
[formControlName]="'username'" name="user_name" 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"
[formControlName]="'password'" name="password" required>
                            </div>
                        </div>

                        <span class="col-md-6 offset-md-4">
                            <button type="submit" class="login">
                                Login
                            </button>
                            <span style="padding-left: 15px">New to SpringReddit? <a
routerLink="/signup">SIGN UP</a></span>
                        </span>
                        <div class="login-failed" *ngIf='this.isError'>
                             <p class="login-failed-text">Login Failed. Please check your credentials and
try again.</p>
                        </div>
                    </form>
                </div>
            </div>
        </div>
        <div class="col-md-3"></div>
    </div>
</div>
```

And the below CSS code to **login.component.css** file

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

.login-failed{
    text-align: center;
    margin: auto;
    margin-top: 10px;
    border: 2px solid black;
    width: 65%;
    background-color: red;
}

.login-failed-text{
    text-align: center;
    margin-top: 5px;
    font-weight: bold;
    color: aliceblue;
}

.login {
    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 we have to create a route to the **Login Page**, when click on the **Login** button on the header bar.

If you compare the css files of login and signup components, you can see the css classes **login** and **sign-up** share the same code, so let's send this code to a common place – **styles.css** file.

styles.css

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

.login,
.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;
}
```

Configuring Routes for Login 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 **login**.

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { SignupComponent } from './auth/signup/signup.component';
import { LoginComponent } from './auth/login/login.component';

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

@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 **Login** button, we should see our Login Page.

Configuring FormGroup inside Login Component

Let's declare and initialze our **FormGroup** inside our **login.component.ts** file.

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

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

  loginForm: FormGroup;

  constructor() { }

  ngOnInit() {
    this.loginForm = new FormGroup({
      username: new FormControl('', Validators.required),
      password: new FormControl('', Validators.required)
    });
  }
}
```

Adding Validations to the Login Page

Now Let's add some input validations to our Login Form, as I already explained in the last article, we will be using Reactive Forms to handle forms. In the above section, we have already declared the validation (**Validators.required**) for our form controls, now let's sync the HTML page.

```
<div class="login-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>Login</h4>
        </div>
        <div class="card-body">
          <form [formGroup]="loginForm" (ngSubmit)="login()">
            <div class="form-group row">
              <label for="user_name" class="col-md-4 col-form-label text-md-right">Username</label>
              <div class="col-md-6">
                <input type="text" id="user_name" class="form-control" [formControlName]="'username'"
name="user_name"
                       required autofocus>
                <span *ngIf="!loginForm.get('username').valid && loginForm.get('username').touched">
                  Please provide 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" id="password" class="form-control" [formControlName]="'password'"
name="password"
                       required>
                <span *ngIf="!loginForm.get('password').valid && loginForm.get('password').touched">
                  Password cannot be empty
                </span>
              </div>
            </div>

            <span class="col-md-6 offset-md-4">
              <button type="submit" class="login">
                Login
              </button>
              <span style="padding-left: 15px">New to SpringReddit? <a routerLink="/sign-up">SIGN UP</a>
</span>
            </span>
            <div class="login-failed" *ngIf='this.isError'>
              <p class="login-failed-text">Login Failed. Please check your credentials and try again.</p>
            </div>
          </form>
        </div>
      </div>
    </div>
    <div class="col-md-3"></div>
  </div>
</div>
```

As you can see the change to the previous HTML is addition of the element

```
<span *ngIf="!loginForm.get('password').valid && loginForm.get('password').touched">
    Password cannot be empty
</span>
```

This is how the validation errors should look like once you save and check the browser. Click on the input field and then click on anywhere, you should see the red border around the input fields. Refer to the image below.

Fixing Bug with Swagger Documentation

In the previous article, we referred to our Swagger Documentation of the REST API, to check how our Signup Request should look like. If we try to open http://localhost:8080/swagger-ui.html we get a **404 Error** because we added **EnableWebMvc** and Spring MVC doesn't know how to handle the Web Jars which are loaded as part of Swagger.

To fix this bug, lets update our **WebConfig.java** file with below method.

```
@Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("swagger-ui.html")
                .addResourceLocations("classpath:/META-INF/resources/");

        registry.addResourceHandler("/webjars/**")
                .addResourceLocations("classpath:/META-INF/resources/webjars/");
    }
```

After you restart the application, you should be able to see the Swagger REST API Documentation page again.

Preparing Login Request & Response

Now let's go ahead and create the model for our Login Request Payload, let's create a file called **login-request.payload.ts** and **login-response.ts** under **auth/login** folder.

login-request.payload.ts

```
export interface LoginRequestPayload {
    username: string;
    password: string;
}
```

login-response.payload.ts

```
export interface LoginResponse {
    authenticationToken: string;
    refreshToken: string;
    expiresAt: Date;
    username: string;
}
```

Add dependency for Ngx-Webstorage

Before we update our **AuthService** class with the login logic, we have to add a dependency to store our auth and refresh tokens on the client-side, this is always a debated topic whether to choose Localstorage vs Cookies, but in our example, we will go with LocalStorage.

Even though storing the token in an HTTP Only Cookie is the better thing to do than storing it in a Localstorage, I feel that even that is not safe if we are dealing with an advanced hacker. But that discussion is for another time and maybe another blog post.

But one thing I can be pretty sure of is one should not be only relying on the security on the client side and have to heavily invest on the Security on the backend if you are building a production grade application.

Having said that, let's add the ngx-webstorage dependency to our **package.json** to add the capabilities to store our token(s) in the LocalStorage, and after that run **npm install**

```
{
  "name": "angular-reddit-clone",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~8.2.5",
    "@angular/common": "~8.2.5",
    "@angular/compiler": "~8.2.5",
    "@angular/core": "~8.2.5",
    "@angular/forms": "~8.2.5",
    "@angular/platform-browser": "~8.2.5",
    "@angular/platform-browser-dynamic": "~8.2.5",
    "@angular/router": "~8.2.5",
    "bootstrap": "^4.4.1",
```

```
    "rxjs": "~6.4.0",
    "tslib": "^1.10.0",
    "zone.js": "~0.9.1"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.803.4",
    "@angular/cli": "~8.3.4",
    "@angular/compiler-cli": "~8.2.5",
    "@angular/language-service": "~8.2.5",
    "@types/node": "~8.9.4",
    "@types/jasmine": "~3.3.8",
    "@types/jasminewd2": "~2.0.3",
    "codelyzer": "^5.0.0",
    "jasmine-core": "~3.4.0",
    "jasmine-spec-reporter": "~4.2.1",
    "karma": "~4.1.0",
    "karma-chrome-launcher": "~2.2.0",
    "karma-coverage-istanbul-reporter": "~2.0.1",
    "karma-jasmine": "~2.0.1",
    "karma-jasmine-html-reporter": "^1.4.0",
    "protractor": "~5.4.0",
    "ts-node": "~7.0.0",
    "tslint": "~5.15.0",
    "typescript": "~3.5.3",
    "ngx-webstorage": "5.0.0"
  }
}
```

Now let's enable add the **NgxWebstorageModule** definition in the **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/signup/signup.component';
import { ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { LoginComponent } from './auth/login/login.component';
import {NgxWebstorageModule} from 'ngx-webstorage';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    SignupComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    HttpClientModule,
    NgxWebstorageModule.forRoot()
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Updating Auth Service and Login Component

Now let's update our AuthService class

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { SignupRequestPayload } from '../signup/singup-request.payload';
import { Observable } from 'rxjs';
import { LoginRequestPayload } from '../login/login-request.payload';
import { LoginResponse } from '../login/login-response.payload';
import { LocalStorageService } from 'ngx-webstorage';
import { map } from 'rxjs/operators';

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

  constructor(private httpClient: HttpClient, private localStorage: LocalStorageService) {
  }

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

  login(loginRequestPayload: LoginRequestPayload): Observable<boolean> {
    return this.httpClient.post<LoginResponse>('http://localhost:8080/api/auth/login', loginRequestPayload)
      .pipe(map(data => {
        this.localStorage.store('authenticationToken', data.authenticationToken);
        this.localStorage.store('username', data.username);
        this.localStorage.store('refreshToken', data.refreshToken);
        this.localStorage.store('expiresAt', data.expiresAt);
        return true;
      }));
  }
}
```

- Inside the login method, we are making a **POST** call to our Login REST API, and we will receive the **LoginResponse** object as the response.
- We can map our response using the map method of **rxjs**, and we are storing the **authToken**, **username**, **refreshToken** and **expirationTime** inside the LocalStorage.

Now let's also update our **login.component.controller.ts**

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

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

  loginForm: FormGroup;
  loginRequestPayload: LoginRequestPayload;

  constructor(private authService: AuthService) {
    this.loginRequestPayload = {
      username: '',
      password: ''
```

```
    };
  }

  ngOnInit() {
    this.loginForm = new FormGroup({
      username: new FormControl('', Validators.required),
      password: new FormControl('', Validators.required)
    });
  }

  login() {
    this.loginRequestPayload.username = this.loginForm.get('username').value;
    this.loginRequestPayload.password = this.loginForm.get('password').value;

    this.authService.login(this.loginRequestPayload).subscribe(data => {
      console.log('Login successful');
    });
  }
}
```

Testing Time

Let's restart our server and open http://localhost:4200 and try to login, you should see the message **Login successful** when you open the console, and if you go to the application section under the local storage, you can see that our user details and token information.

Conclusion

In the next article, we will see how to handle the Auth and Refresh tokens to make REST API calls to our backend.

About the author

Sai Upadhyayula