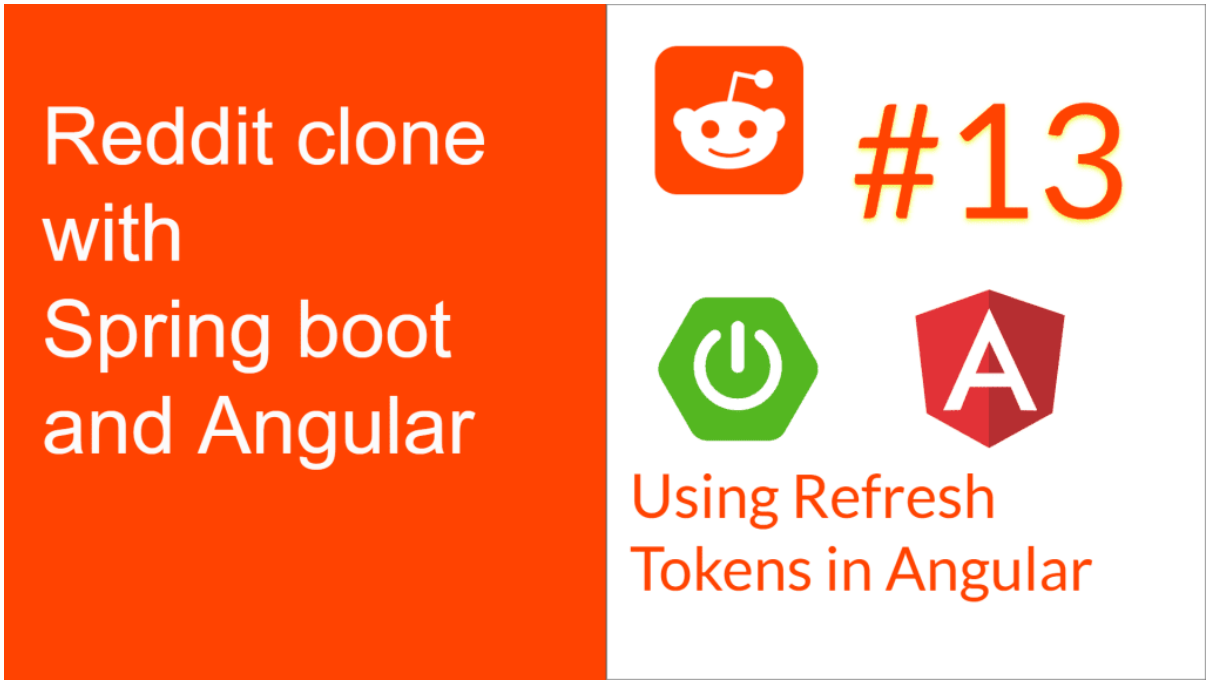
PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 13

0 Comments

BY SAI UPADHYAYULA | April 5, 2020



Welcome to Full Stack Reddit Clone with Spring boot and Angular – Part 13. In Part 12, we saw how to implement Login functionality in our Angular application.

In this article, we will see how to use refresh tokens to rotate our JWT Authentication Tokens. But before that let's make some small improvements to our existing Signup and Login functionality.

If you are a visual learner like me, you can check out the 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

Enable Toaster Notifications

Let's add a dependency in our Angular project to display Toaster Notifications after the user successfully signed up and Logged In to our application.

The Toaster Notification looks something like the below image:

Open the **package.json** and add the below dependency – ngx-toastr, the current version as of `12.0.1`. This is how our `package.json` file looks like:

```
{
  "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": "~9.1.0",
    "@angular/common": "~9.1.0",
    "@angular/compiler": "~9.1.0",
    "@angular/core": "~9.1.0",
    "@angular/forms": "~9.1.0",
```

```
    "@angular/platform-browser": "~9.1.0",
    "@angular/platform-browser-dynamic": "~9.1.0",
    "@angular/router": "~9.1.0",
    "bootstrap": "^4.4.1",
    "rxjs": "~6.5.5",
    "tslib": "^1.11.1",
    "zone.js": "~0.10.3"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.901.0",
    "@angular/cli": "~9.1.0",
    "@angular/compiler-cli": "~9.1.0",
    "@angular/language-service": "~9.1.0",
    "@types/node": "~13.11.0",
    "@types/jasmine": "~3.5.10",
    "@types/jasminewd2": "~2.0.8",
    "codelyzer": "^5.2.2",
    "jasmine-core": "~3.5.0",
    "jasmine-spec-reporter": "~5.0.1",
    "karma": "~4.4.1",
    "karma-chrome-launcher": "~3.1.0",
    "karma-coverage-istanbul-reporter": "~2.1.1",
    "karma-jasmine": "~3.1.1",
    "karma-jasmine-html-reporter": "^1.5.3",
    "protractor": "~5.4.3",
    "ts-node": "~8.8.1",
    "tslint": "~6.1.1",
    "typescript": "~3.8.3",
    "ngx-webstorage": "5.0.0",
    "ngx-toastr": "12.0.1"
  }
}
```

After that run `npm install` and our depdnency should be installed successfully.

To enable Toastr notifications, we have to add the following piece of code inside our `angular.json` file

```
"./node_modules/ngx-toastr/toastr.css"
```

Now let's add `ToastrModule` to our `app.module.ts` file, we should also add `BrowserAnimationsModule` as `ngx-toastr` makes use of angular animations in the background.

```
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';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { ToastrModule } from 'ngx-toastr';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    SignupComponent,
    LoginComponent
```

```
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    HttpClientModule,
    NgxWebstorageModule.forRoot(),
    BrowserAnimationsModule,
    ToastrModule.forRoot()
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Inside Signup Component

Now let's add enable routing from Signup page to Login Page on successful Registration, if the registration fails, we should see an error on the top right corner of the screen with the Toastr error message.

signup.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { SignupRequestPayload } from './singup-request.payload';
import { AuthService } from '../shared/auth.service';
import { Router } from '@angular/router';
import { ToastrService } from 'ngx-toastr';

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

  signupRequestPayload: SignupRequestPayload;
  signupForm: FormGroup;

 constructor(private authService: AuthService, private router: Router, private toastr: ToastrService) {
    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.email = this.signupForm.get('email').value;
    this.signupRequestPayload.username = this.signupForm.get('username').value;
    this.signupRequestPayload.password = this.signupForm.get('password').value;

    this.authService.signup(this.signupRequestPayload)
      .subscribe(() => {
        this.router.navigate(['/login'], { queryParams: { registered: 'true' } })
      }, () => {
        this.toastr.error('Registration Failed! Please try again');
```

```
      });
    }
  }
```

- We injected the `Router` and `ToastrService` classes into our `SignUpComponent`
- Inside the **signup** method, if we received **success** response, we are using the injected `router` object to navigate to the **Login** page and notice that we are adding a query param `registered:true` to communicate with the `LoginComponent` that registration is successful

- If we received a failure response, we will display an error notification.

Inside Login Component

```
<strong>login.component.css</strong>
```

```
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';
import { ActivatedRoute, Router } from '@angular/router';
import { ToastrService } from 'ngx-toastr';

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

  loginForm: FormGroup;
  loginRequestPayload: LoginRequestPayload;
  registerSuccessMessage: string;
  isError: boolean;

  constructor(private authService: AuthService, private activatedRoute: ActivatedRoute,
    private router: Router, private toastr: ToastrService) {
    this.loginRequestPayload = {
      username: '',
      password: ''
    };
  }

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

    this.activatedRoute.queryParams
      .subscribe(params => {
        if (params.registered !== undefined && params.registered === 'true') {
          this.toastr.success('Signup Successful');
          this.registerSuccessMessage = 'Please Check your inbox for activation email '
            + 'activate your account before you Login!';
        }
      });
  }

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

    this.authService.login(this.loginRequestPayload).subscribe(data => {
      if (data) {
```

```
          this.isError = false;
          this.router.navigateByUrl('/');
          this.toastr.success('Login Successful');
        } else {
          this.isError = true;
        }
      });
    }

  }
```

- Now in our `LoginComponent`, we inject `Router` and `ToastrService` objects, additionally we injected `ActivatedRoute` class to access the route parameters.
- Inside the `ngOnInit()` the method, we are subscribing to the queryParams from the `activatedRoute` object and in case we receive a query parameter with value for registered as true, then we display the success notification – "Signup Successful" and set the value for the field `registerSuccessMessage`
- Lastly, after successful Login, we are navigating to the root URL – '/' and then enabling the Success Notification with the message – "Login Successful".

> In our `login.component.html` file, we already added the below block of code, which displays an error message if Login Fails.

```
<div class="login-failed" *ngIf='this.isError'>
    <p class="login-failed-text">
        Login Failed. Please check your credentials and try again.
    </p>
</div>
```

Let's add CSS code for the above snippet inside the `login.component.css` file

`login.component.css`

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

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

Now let's move ahead and implement Refresh Token functionality in our Angular Application.

Handling Refresh Tokens

In Part 8, when our JWT is expired, our Backend application provides us a special Token called Refresh Token which can be used to request new JWT's.

So that means we have to intercept each HTTP request we are making to our Backend application and check whether the JWT is expired (or) about to be expired, in that case, we will make a REST call to our backend to generate new JWT and set the Authorization Header with Bearer Scheme.

First we will create an interceptor in our Angular application called `TokenInterceptor`

token-interceptor.ts

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpErrorResponse } from
'@angular/common/http';
import { Observable, throwError, BehaviorSubject } from 'rxjs';
import { AuthService } from './auth/shared/auth.service';
import { catchError, switchMap } from 'rxjs/operators';
import { LoginResponse } from './auth/login/login-response.payload';

@Injectable({
    providedIn: 'root'
})
export class TokenInterceptor implements HttpInterceptor {

    isTokenRefreshing = false;
    refreshTokenSubject: BehaviorSubject<any> = new BehaviorSubject(null);
    constructor(public authService: AuthService) { }

    intercept(req: HttpRequest<any>,
        next: HttpHandler): Observable<HttpEvent<any>> {
        if (this.authService.getJwtToken()) {
            this.addToken(req, this.authService.getJwtToken());
        }

        return next.handle(req).pipe(catchError(error => {
            if (error instanceof HttpErrorResponse
                && error.status === 403) {
                return this.handleAuthErrors(req, next);
            } else {
                return throwError(error);
            }
        }));
    }
    private handleAuthErrors(req: HttpRequest<any>, next: HttpHandler) {
        if (!this.isTokenRefreshing) {
            this.isTokenRefreshing = true;
            this.refreshTokenSubject.next(null);

            return this.authService.refreshToken().pipe(
                switchMap((refreshTokenResponse: LoginResponse) => {
                    this.isTokenRefreshing = false;
```

```
                        this.refreshTokenSubject.next(refreshTokenResponse.authenticationToken);
                        return next.handle(this.addToken(req, refreshTokenResponse.authenticationToken));
                    })
                )
            }
        }
        private addToken(req: HttpRequest<any>, jwtToken: string) {
            return req.clone({
                headers: req.headers.set('Authorization',
                    'Bearer ' + jwtToken)
            });
        }

    }
```

auth.service.ts

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

      refreshToken() {
        const refreshTokenPayload = {
          refreshToken: this.getRefreshToken(),
          username: this.getUserName()
        }
        return this.httpClient.post<LoginResponse>('http://localhost:8080/api/auth/refresh/token',
          refreshTokenPayload)
          .pipe(tap(response => {
            this.localStorage.store('authenticationToken', response.authenticationToken);
            this.localStorage.store('expiresAt', response.expiresAt);
          }));
      }
```

```
  getJwtToken() {
    return this.localStorage.retrieve('authenticationToken');
  }

  getRefreshToken() {
    return this.localStorage.retrieve('refreshToken');
  }

  getUserName() {
    return this.localStorage.retrieve('username');
  }

  getExpirationTime() {
    return this.localStorage.retrieve('expiresAt');
  }
}
```

- Inside the intercept method, we receive the JWT through `authService.getJwtToken()`, if the Token is valid, we add the token to the **Authorization** Header which contains a value according to the **Bearer** scheme.
- If the token is invalid, due to a 401 error, then we will request a new JWT by calling the `authService.refreshToken()`
- In this case, we queue all the corresponding requests to the backend and keep them on hold, until our Token Refresh process is completed.
- Once the refresh is completed, we release the requests again, we will accomplish this using a BehaviorSubject

Refresh Token Process – Step by Step

Let's see in-details what is going in the TokenInterceptor class in much details:

- If the token refresh process has not already started, then the `isTokenRefreshing` variable will be `false` by default, and a `null` value will be assigned to `refreshTokenSubject` object.
- When the token refresh process started, then we set the `isTokenRefreshing` variable to `true` and once we receive the response we pass the JWT as the value for our `refreshTokenSubject`
- Once the token refresh process is completed, we will finish the processing with `next.handle`

Finally, we will add the `TokenInterceptor` class to the providers section inside the `app.module.ts` file

`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 { SignupComponent } from './auth/signup/signup.component';
import { ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { LoginComponent } from './auth/login/login.component';
import {NgxWebstorageModule} from 'ngx-webstorage';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { ToastrModule } from 'ngx-toastr';
import { TokenInterceptor } from './token-interceptor';

@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    SignupComponent,
    LoginComponent
```

```
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    HttpClientModule,
    NgxWebstorageModule.forRoot(),
    BrowserAnimationsModule,
    ToastrModule.forRoot()
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: TokenInterceptor,
      multi: true
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Create Home Page Component

To test our Refresh Token functionality, we need to make calls to some secured API from our Angular application, for that let's create Home Page Component and here let's retrieve all the Posts we already have in our database.

Now copy the following HTML and paste it to the `home.component.html` file.

`home.component.html`

```
<div class="reddit-body">
  <div class="container">
    <div class="row">
      <hr />
      <div class="col-md-9">
        <!-- Display Posts-->
        <div class="row post" *ngFor="let post of posts$">
          <!-- Section to Display Votes-->
          <div class="col-md-1">
            <div class="d-flex flex-column votebox">
              <div class="p-2">
                <fa-icon (click)="upvotePost()" class="upvote" [icon]="faArrowUp"
[style.color]="upvoteColor"></fa-icon>
              </div>
              <div class="p-2 votecount">{{post.voteCount}}</div>
              <div class="p-2">
                <fa-icon (click)="downvotePost()" class="downvote" [icon]="faArrowDown"
[style.color]="downvoteColor">
                </fa-icon>
              </div>
            </div>
          </div>
```

```
            <!-- Section to Display Post Information-->
            <div class="col-md-11">
              <span class="subreddit-info">
                <span class="subreddit-text"><a class="posturl" routerLink="">{{post.subredditName}}</a>
</span>
                <span> . Posted by <a class="username" routerLink="/user/{{post.userName}}">{{post.userName}}
</a></span>
                <span> . {{post.duration}}</span>
              </span>
              <hr />
              <div class="post-title">
                <a class="postname" href="{{post.url}}">{{post.postName}}</a>
              </div>
              <div>
                <p class="post-text">{{post.description}}</p>
              </div>
              <hr />
              <span>
                <a class="btnCommments" role="button">
                  <fa-icon [icon]="faComments"></fa-icon>
                  Comments({{post.commentCount}})
                </a>
                <button class="login" (click)="goToPost(post.id)">
                  Read Post
                </button>
              </span>
            </div>
          </div>
        </div>
        <div class="col-md-3">
        </div>
      </div>
    </div>
</div>
```

Now inside the `home.component.ts` file, we need to retrieve all the posts we have in our application, let's create a `PostService` class for that.

Inside the PostService class, we have to call the REST API which exposes us all the posts in our application, if we look at the Swagger REST API Documentation we created as part of Part 10

Let's create a class called `PostModel` which contains all the fields in the response from **getAllPosts()** REST API call.

post-model.ts

```
export class PostModel {
    id: number;
    postName: string;
    url: string;
    description: string;
    voteCount: number;
    userName: string;
    subredditName: string;
    commentCount: number;
    duration: string;
}
```

post.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { PostModel } from './post-model';

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

  constructor(private http: HttpClient) { }

  getAllPosts(): Observable<Array<PostModel>> {
    return this.http.get<Array<PostModel>>('http://localhost:8080/api/posts/');
  }
}
```

So now we are calling the **getAllPosts()** from our `PostService` class and we are storing the response in `Array<PostModel>`, let's now inject the `PostService` inside the `home.component.ts` file.

`home.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { PostService } from '../shared/post.service';
import { PostModel } from '../shared/post-model';

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

  posts$: Array<PostModel> = [];

  constructor(private postService: PostService) {
    this.postService.getAllPosts().subscribe(post => {
      this.posts$ = post;
    })
  }

  ngOnInit(): void {
  }

}
```

Finally, let's add the route to the root URL and assign it to `HomeComponent`

`<strong>app-routing.module.ts</strong>`

```
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';
import { HomeComponent } from './home/home.component';

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

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

Testing our implementation

So we have changed a lot of code and added new classes, so let's restart our server to make sure to restart our angular application and let's test our implementation.

- I recorded and generated a GIF to show the network calls which was made to our REST API, first we see the network call to the **/api/posts** API which failed with 403 error.

- Then the next call is to get the new JWT token, and the subsequent call is again to the **getAllPosts()** API. This time we received a 200 response back from the Server.

Conclusion

I hope you learned something from this article, if yes please consider to subscribe to my newsletter where you can receive the updates for latest tutorials I am going to release.

I will see you in the next article, until then **Happy Coding Techies** 🙂

> Icons seen in this article are taken from Flaticon.com, credits – Freepik Smashicons

About the author
Sai Upadhyayula