PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 17

0 Comments

BY SAI UPADHYAYULA | April 24, 2020

Share Tweet Pin

Welcome to **Full Stack Reddit Clone with Spring boot and Angular – Part 17**. In Part 16, we created the functionality to Post Comments and Display User Profile in our Angular Application.

In this final article in this series, we are going to:

- Implement Voting Mechanism
- Implement Logout
- Protect our application routes using AuthGuards

Let's start coding.

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

Implement Voting Mechanism

We have already created the `VotebuttonComponent` in the previous article when refactoring our home page. We have the up-vote and down-vote buttons, and inside the html, the click directive is calling the `upVotePost()` and `downVotePost()` methods, but as of now they are not doing anything.

Let's go ahead and implement those methods.

vote-button.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { PostModel } from '../post-model';
import { faArrowUp, faArrowDown } from '@fortawesome/free-solid-svg-icons';
import { VotePayload } from './vote-payload';
import { VoteType } from './vote-type';
import { VoteService } from '../vote.service';
import { AuthService } from 'src/app/auth/shared/auth.service';
import { PostService } from '../post.service';
import { throwError } from 'rxjs';
import { ToastrService } from 'ngx-toastr';

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

  @Input() post: PostModel;
  votePayload: VotePayload;
  faArrowUp = faArrowUp;
  faArrowDown = faArrowDown;
  upvoteColor: string;
  downvoteColor: string;
```

```
  constructor(private voteService: VoteService,
    private authService: AuthService,
    private postService: PostService, private toastr: ToastrService) {

    this.votePayload = {
      voteType: undefined,
      postId: undefined
    }
  }

  ngOnInit(): void {
    this.updateVoteDetails();
  }

  upvotePost() {
    this.votePayload.voteType = VoteType.UPVOTE;
    this.vote();
    this.downvoteColor = '';
  }

  downvotePost() {
    this.votePayload.voteType = VoteType.DOWNVOTE;
    this.vote();
    this.upvoteColor = '';
  }

  private vote() {
    this.votePayload.postId = this.post.id;
    this.voteService.vote(this.votePayload).subscribe(() => {
      this.updateVoteDetails();
    }, error => {
      this.toastr.error(error.error.message);
      throwError(error);
    });
  }

  private updateVoteDetails() {
    this.postService.getPost(this.post.id).subscribe(post => {
      this.post = post;
    });
  }
}
```

vote-button.component.html

```
<!-- 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]="post.upVote ?
'green':''">
        </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]="post.downVote ? 'red':''">
        </fa-icon>
    </div>
    </div>
</div>
```

We need to create the VoteService class where we can make backend calls to our Vote API. The command to create this service can be found in the below image.

vote-service.ts

```
iimport { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { VotePayload } from './vote-button/vote-payload';
import { Observable } from 'rxjs';

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

  constructor(private http: HttpClient) { }

  vote(votePayload: VotePayload): Observable<any> {
    return this.http.post('http://localhost:8080/api/votes/', votePayload);
  }
}
```

vote-payload.ts

```
import { VoteType } from './vote-type';

export class VotePayload {
    voteType: VoteType;
    postId: number;
}
```

vote-type.ts

```
export enum VoteType {
    UPVOTE,
    DOWNVOTE
}
```

Swagger API Documentation for Votes

- As you can see in the above image, the Create Vote API takes a request body which contains fields `postId` and `voteType`.
- We created a class called `VotePayload` which encapsulates the above-mentioned fields and for `VoteType` we created an enum with possible values as `UPVOTE` and `DOWNVOTE`.
- Inside the `VoteService` class, we first injected the `HttpClient` class, and inside the `vote()` method which takes the `VotePayload` object as input, we are making a POST call to our REST API.

Inside the `VotebuttonComponent` :

- We first injected the `VoteService`, `AuthService`, `PostService` and `ToastrService` classes through the constructor.
- We declared and initialized the `VotePayload` object inside the constructor.
- Then we have the `upvotePost()` and `downvotePost()` methods, where we are setting the `VoteType` for the `VotePayload` object, and calling the `vote()` method inside the component.
- This `vote()` method, is setting the value for the `postId` field, which is coming as input, and after that, we are calling the `vote()` method of the `VoteService` class, which returns an `Observable`.
- We subscribe to the returned `Observable`, and in the case of a success response, we are updating the Vote Details, like the VoteCount and also an indication whether the post is either upVoted or downVoted by the user.

post-model.ts

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

Inside the PostModel class, we added two new fields upVote and downVote, we are recei

backend.

PostResponse.java

```
package com.example.springredditclone.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class PostResponse {
    private Long id;
    private String postName;
    private String url;
    private String description;
    private String userName;
    private String subredditName;
    private Integer voteCount;
    private Integer commentCount;
    private String duration;
    private boolean upVote;
    private boolean downVote;
}
```

PostMapper.java

```
package com.example.springredditclone.mapper;

import com.example.springredditclone.dto.PostRequest;
import com.example.springredditclone.dto.PostResponse;
import com.example.springredditclone.model.*;
import com.example.springredditclone.repository.CommentRepository;
import com.example.springredditclone.repository.VoteRepository;
import com.example.springredditclone.service.AuthService;
import com.github.marlonlom.utilities.timeago.TimeAgo;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.Optional;

import static com.example.springredditclone.model.VoteType.DOWNVOTE;
import static com.example.springredditclone.model.VoteType.UPVOTE;

@Mapper(componentModel = "spring")
public abstract class PostMapper {

    @Autowired
    private CommentRepository commentRepository;
    @Autowired
    private VoteRepository voteRepository;
    @Autowired
    private AuthService authService;


    @Mapping(target = "createdDate", expression = "java(java.time.Instant.now())")
    @Mapping(target = "description", source = "postRequest.description")
    @Mapping(target = "subreddit", source = "subreddit")
    @Mapping(target = "voteCount", constant = "0")
    @Mapping(target = "user", source = "user")
```

```
    public abstract Post map(PostRequest postRequest, Subreddit subreddit, User user);

    @Mapping(target = "id", source = "postId")
    @Mapping(target = "subredditName", source = "subreddit.name")
    @Mapping(target = "userName", source = "user.username")
    @Mapping(target = "commentCount", expression = "java(commentCount(post))")
    @Mapping(target = "duration", expression = "java(getDuration(post))")
    @Mapping(target = "upVote", expression = "java(isPostUpVoted(post))")
    @Mapping(target = "downVote", expression = "java(isPostDownVoted(post))")
    public abstract PostResponse mapToDto(Post post);

    Integer commentCount(Post post) {
        return commentRepository.findByPost(post).size();
    }

    String getDuration(Post post) {
        return TimeAgo.using(post.getCreatedDate().toEpochMilli());
    }

    boolean isPostUpVoted(Post post) {
        return checkVoteType(post, UPVOTE);
    }

    boolean isPostDownVoted(Post post) {
        return checkVoteType(post, DOWNVOTE);
    }

    private boolean checkVoteType(Post post, VoteType voteType) {
        if (authService.isLoggedIn()) {
            Optional<Vote> voteForPostByUser = voteRepository.findTopByPostAndUserOrderByVoteIdDesc(post,
                    authService.getCurrentUser());
            return voteForPostByUser.filter(vote -> vote.getVoteType().equals(voteType))
                    .isPresent();
        }
        return false;
    }

}
```

AuthService.java

```
package com.example.springredditclone.service;

import com.example.springredditclone.dto.AuthenticationResponse;
import com.example.springredditclone.dto.LoginRequest;
import com.example.springredditclone.dto.RefreshTokenRequest;
import com.example.springredditclone.dto.RegisterRequest;
import com.example.springredditclone.exceptions.SpringRedditException;
import com.example.springredditclone.model.NotificationEmail;
import com.example.springredditclone.model.User;
import com.example.springredditclone.model.VerificationToken;
import com.example.springredditclone.repository.UserRepository;
import com.example.springredditclone.repository.VerificationTokenRepository;
import com.example.springredditclone.security.JwtProvider;
import lombok.AllArgsConstructor;
import org.springframework.security.authentication.AnonymousAuthenticationToken;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
```

```
import java.time.Instant;
import java.util.Optional;
import java.util.UUID;

@Service
@AllArgsConstructor
@Transactional
public class AuthService {

    private final PasswordEncoder passwordEncoder;
    private final UserRepository userRepository;
    private final VerificationTokenRepository verificationTokenRepository;
    private final MailService mailService;
    private final AuthenticationManager authenticationManager;
    private final JwtProvider jwtProvider;
    private final RefreshTokenService refreshTokenService;

    public void signup(RegisterRequest registerRequest) {
        User user = new User();
        user.setUsername(registerRequest.getUsername());
        user.setEmail(registerRequest.getEmail());
        user.setPassword(passwordEncoder.encode(registerRequest.getPassword()));
        user.setCreated(Instant.now());
        user.setEnabled(false);

        userRepository.save(user);

        String token = generateVerificationToken(user);
        mailService.sendMail(new NotificationEmail("Please Activate your Account",
                user.getEmail(), "Thank you for signing up to Spring Reddit, " +
                "please click on the below url to activate your account : " +
                "http://localhost:8080/api/auth/accountVerification/" + token));
    }

    @Transactional(readOnly = true)
    public User getCurrentUser() {
        org.springframework.security.core.userdetails.User principal =
(org.springframework.security.core.userdetails.User) SecurityContextHolder.
                getContext().getAuthentication().getPrincipal();
        return userRepository.findByUsername(principal.getUsername())
                .orElseThrow(() -> new UsernameNotFoundException("User name not found - " +
principal.getUsername()));
    }

    private void fetchUserAndEnable(VerificationToken verificationToken) {
        String username = verificationToken.getUser().getUsername();
        User user = userRepository.findByUsername(username).orElseThrow(() -> new
SpringRedditException("User not found with name - " + username));
        user.setEnabled(true);
        userRepository.save(user);
    }

    private String generateVerificationToken(User user) {
        String token = UUID.randomUUID().toString();
        VerificationToken verificationToken = new VerificationToken();
        verificationToken.setToken(token);
        verificationToken.setUser(user);

        verificationTokenRepository.save(verificationToken);
        return token;
    }

    public void verifyAccount(String token) {
        Optional<VerificationToken> verificationToken = verificationTokenRepository.findByToken(token);
        fetchUserAndEnable(verificationToken.orElseThrow(() -> new SpringRedditException("Invalid
Token")));
    }

    public AuthenticationResponse login(LoginRequest loginRequest) {
```

```
        Authentication authenticate = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
                loginRequest.getPassword()));
        SecurityContextHolder.getContext().setAuthentication(authenticate);
        String token = jwtProvider.generateToken(authenticate);
        return AuthenticationResponse.builder()
                .authenticationToken(token)
                .refreshToken(refreshTokenService.generateRefreshToken().getToken())
                .expiresAt(Instant.now().plusMillis(jwtProvider.getJwtExpirationInMillis()))
                .username(loginRequest.getUsername())
                .build();
    }

    public AuthenticationResponse refreshToken(RefreshTokenRequest refreshTokenRequest) {
        refreshTokenService.validateRefreshToken(refreshTokenRequest.getRefreshToken());
        String token = jwtProvider.generateTokenWithUserName(refreshTokenRequest.getUsername());
        return AuthenticationResponse.builder()
                .authenticationToken(token)
                .refreshToken(refreshTokenRequest.getRefreshToken())
                .expiresAt(Instant.now().plusMillis(jwtProvider.getJwtExpirationInMillis()))
                .username(refreshTokenRequest.getUsername())
                .build();
    }

    public boolean isLoggedIn() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        return !(authentication instanceof AnonymousAuthenticationToken) &&
authentication.isAuthenticated();
    }
}
```

Implementing Logout

> The next task is to implement Logout in our Angular application, implementing Logout on the frontend is a simple task, we just have to delete the stored Auth and Refresh Tokens from the Local Storage.
>
> After that, we will make an API call to the backend to delete the Refresh Tokens so that it wont be possible to rotate the JWT's.

We already have the HTML code ready in our header.component.html file, we just have to implement the logic inside the header.component.ts file

header.component.ts

```
import { Component, OnInit } from '@angular/core';
import { faUser } from '@fortawesome/free-solid-svg-icons';
import { AuthService } from '../auth/shared/auth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit {
  faUser = faUser;
  isLoggedIn: boolean;
  username: string;

  constructor(private authService: AuthService, private router: Router) { }

  ngOnInit() {
    this.isLoggedIn = this.authService.isLoggedIn();
    this.username = this.authService.getUserName();
```

```
  }

  goToUserProfile() {
    this.router.navigateByUrl('/user-profile/' + this.username);
  }

  logout() {
    this.authService.logout();
    this.router.navigateByUrl('').then(() => {
      window.location.reload();
    })
  }
}
```

auth.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { SignupRequestPayload } from '../signup/singup-request.payload';
import { Observable, throwError } 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 {
  refreshTokenPayload = {
    refreshToken: this.getRefreshToken(),
    username: this.getUserName()
  }

  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;
      }));
  }

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

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

```
        this.localStorage.store('authenticationToken',
          response.authenticationToken);
        this.localStorage.store('expiresAt', response.expiresAt);
      }));
  }

  logout() {
    this.httpClient.post('http://localhost:8080/api/auth/logout', this.refreshTokenPayload,
      { responseType: 'text' })
      .subscribe(data => {
        console.log(data);
      }, error => {
        throwError(error);
      })
    this.localStorage.clear('authenticationToken');
    this.localStorage.clear('username');
    this.localStorage.clear('refreshToken');
    this.localStorage.clear('expiresAt');
  }

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

  isLoggedIn(): boolean {
    return this.getJwtToken() != null;
  }
}
```

- The logout() method inside the header.component.ts is calling the logout() method inside the AuthService.
- Inside this method, we are making a POST call to our backend, this call returns a String as response, which we are just logging to the console.
- Then, we are clearing all the details which are stored in the local storage like the username, authentication token, refresh token and expiration time.

This completes our task to implement Logout in our Angular application.

Protect the application routes using Guards

> So we have some routes, which are supposed to be secured. That means those routes should only be accessible when the user is logged in. In Angular, we have concept called Guards which tells the router whether to allow the navigation to a particular route or not. There are different types of Guards but we will use the CanActivate guard in our scenario.

Type the below command in the terminal to generate a guard, for the question – “Which interfaces would you like to implement ?” answer with – CanActivate.

auth.guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router } from
'@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from './shared/auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean |
UrlTree {

    const isAuthenticated = this.authService.isLoggedIn();
    if (isAuthenticated) {
      return true;
    } else {
      this.router.navigateByUrl('/login');
    }
    return true;
  }
}
```

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';
import { HomeComponent } from './home/home.component';
import { CreatePostComponent } from './post/create-post/create-post.component';
import { CreateSubredditComponent } from './subreddit/create-subreddit/create-subreddit.component';
import { ListSubredditsComponent } from './subreddit/list-subreddits/list-subreddits.component';
import { ViewPostComponent } from './post/view-post/view-post.component';
import { UserProfileComponent } from './auth/user-profile/user-profile.component';
import { AuthGuard } from './auth/auth.guard';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'view-post/:id', component: ViewPostComponent },
  { path: 'user-profile/:name', component: UserProfileComponent, canActivate: [AuthGuard] },
  { path: 'list-subreddits', component: ListSubredditsComponent },
  { path: 'create-post', component: CreatePostComponent, canActivate: [AuthGuard] },
  { path: 'create-subreddit', component: CreateSubredditComponent, canActivate: [AuthGuard] },
  { path: 'sign-up', component: SignupComponent },
  { path: 'login', component: LoginComponent }
];

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

- The `AuthGuard` class is implementing the `CanActivate` interface, inside the `canActivate()` method, we are checking if the user is logged in or not.
- If the user is not logged in, we are redirecting to the Login Page.
- We want to activate this guard for the routes – `create-post`, `create-subreddit` and `user-profile`.

If you try to logout, you can see that the page is redirected to the Login page, and if you go to the home page, you see nothing inside the Post and Subreddit Sidebar sections, this is because the endpoints which expose this information are secured. We have to exclude them from our SecurityConfig.java file

`SecurityConfig.java`

```
package com.example.springredditclone.config;

import com.example.springredditclone.security.JwtAuthenticationFilter;
import lombok.AllArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.BeanIds;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@EnableWebSecurity
@AllArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;
    private final JwtAuthenticationFilter jwtAuthenticationFilter;

    @Bean(BeanIds.AUTHENTICATION_MANAGER)
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    public void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.cors().and()
                .csrf().disable()
                .authorizeRequests()
                .antMatchers("/api/auth/**")
                .permitAll()
                .antMatchers(HttpMethod.GET, "/api/subreddit")
                .permitAll()
                .antMatchers(HttpMethod.GET, "/api/posts/")
                .permitAll()
                .antMatchers(HttpMethod.GET, "/api/posts/**")
                .permitAll()
                .antMatchers("/v2/api-docs",
                        "/configuration/ui",
                        "/swagger-resources/**",
                        "/configuration/security",
                        "/swagger-ui.html",
                        "/webjars/**")
                .permitAll()
                .anyRequest()
                .authenticated();
```

```
        httpSecurity.addFilterBefore(jwtAuthenticationFilter,
                UsernamePasswordAuthenticationFilter.class);
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception
{
        authenticationManagerBuilder.userDetailsService(userDetailsService)
                .passwordEncoder(passwordEncoder());
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

We have to also update the token-interceptor.ts file, there is no need to add Token details for the requests to GET Subreddits and Posts.

token-interceptor.ts

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpErrorResponse } from
'@angular/common/http';
import { Observable, BehaviorSubject, throwError } from 'rxjs';
import { AuthService } from './auth/shared/auth.service';
import { catchError, switchMap, take, filter } 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 (req.url.indexOf('refresh') !== -1 || req.url.indexOf('login') !== -1
            || (req.url.indexOf('/api/posts/') !== -1 && req.method.indexOf('GET') !== -1)
            || (req.url.indexOf('/api/subreddit') !== -1 && req.method.indexOf('GET') !== -1)) {
            return next.handle(req);
        }
        const jwtToken = this.authService.getJwtToken();

        return next.handle(this.addToken(req, jwtToken)).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)
        : Observable<HttpEvent<any>> {
        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));
                    })
                )
            } else {
                return this.refreshTokenSubject.pipe(
                    filter(result => result !== null),
                    take(1),
                    switchMap((res) => {
                        return next.handle(this.addToken(req,
                            this.authService.getJwtToken()))
                    })
                );
            }
        }

        addToken(req: HttpRequest<any>, jwtToken: any) {
            return req.clone({
                headers: req.headers.set('Authorization',
                    'Bearer ' + jwtToken)
            });
        }

    }
```

Conclusion

So well, Ladies and Gentlemen, that is the end of **Building a Full Stack Reddit Clone with Spring boot and Angular series**, if you have followed me through from starting, **I Salute You**, I salute your determination and enthusiasm to learn. Hopefully, you did learn something by following this tutorial. It was very exciting for me to prepare these tutorials and share them with you

About the author
Sai Upadhyayula