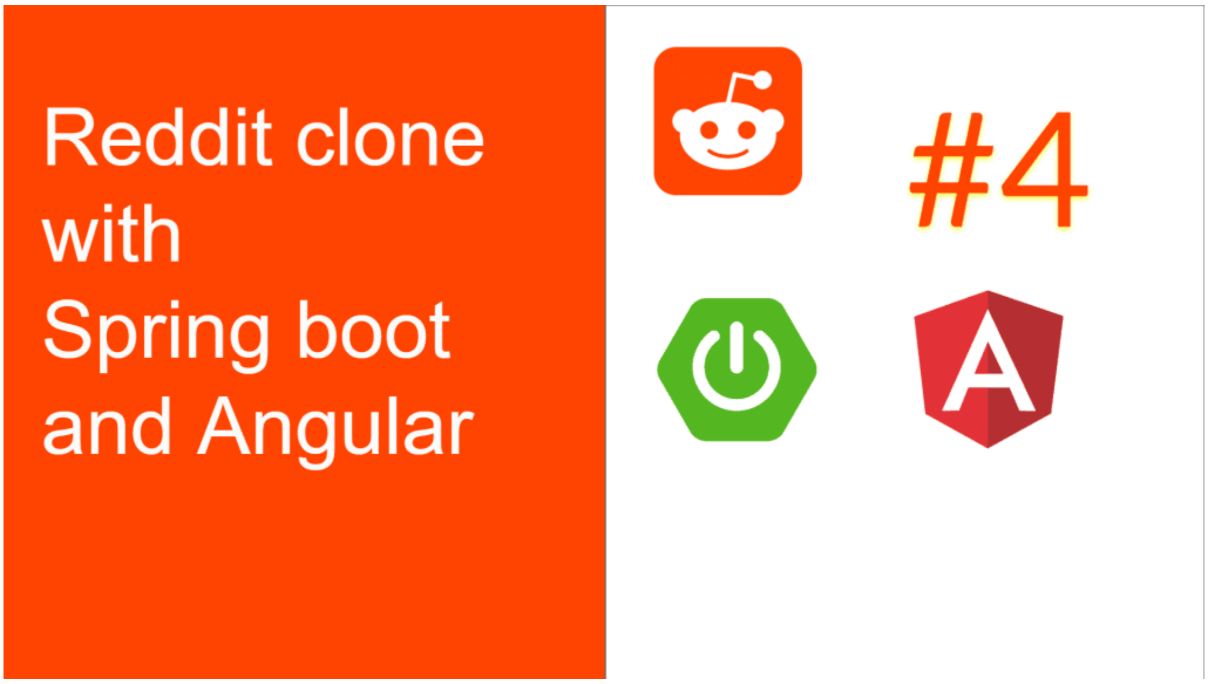
PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 4

0 Comments

BY SAI UPADHYAYULA | November 30, 2019



Share Tweet Pin

Welcome to Part 4 of my Build a Full Stack Reddit Clone with Spring Boot and Angular series.

In Part 3 we wrote the Login API and implemented the Token-Based Authentication System with the help of JWT (JSON Web Tokens). In this article, we will see how to validate the JWT and we will implement the API to create Subreddits. Without wasting any time, let's dive into the code.

If you are a visual learner like me, have a look at this video tutorial which serves as a companion to this post.

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

> The source code of this project, is hosted on Github –
> Backend – https://github.com/SaiUpadhyayula/spring-reddit-clone
> Frontend – https://github.com/SaiUpadhyayula/angular-reddit-clone

What are we building?

- First, we will see how to validate the JWT's we receive from the client.
- Implement API to create Subreddits.

Implement JWT Validation

In Part 3 we saw the Authentication Flow in our application, according to the sequence diagram when the client has authenticated successfully, on each subsequent request, the client provides the JWT to the server and the server should validate it.

Let's deep dive and see how the validation process works, in the below image you see another sequence diagram which zooms in on step 5 of the Authentication Flow sequence diagram.

- The client makes a REST call to our API, the token is sent as part of the Authorization header by following the Bearer Scheme.
- The request is intercepted by the **JWTAuthenticationFilter**, which is a custom component, this filter class validates JWT, and if the token is valid, the request is forwarded to the corresponding Controller.

JwtAuthenticationFilter.java

```
package com.example.springredditclone.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private JwtProvider jwtProvider;
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        String jwt = getJwtFromRequest(request);

        if (StringUtils.hasText(jwt) && jwtProvider.validateToken(jwt)) {
            String username = jwtProvider.getUsernameFromJWT(jwt);

            UserDetails userDetails = userDetailsService.loadUserByUsername(username);
            UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(userDetails,
                    null, userDetails.getAuthorities());
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        filterChain.doFilter(request, response);
    }

    private String getJwtFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");

        if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }
        return bearerToken;
    }
}
```

JwtProvider.java

```
package com.example.springredditclone.security;

import com.example.springredditclone.exceptions.SpringRedditException;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.User;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.io.IOException;
import java.io.InputStream;
import java.security.*;
import java.security.cert.CertificateException;

import static io.jsonwebtoken.Jwts.parser;

@Service
public class JwtProvider {

    private KeyStore keyStore;

    @PostConstruct
    public void init() {
        try {
            keyStore = KeyStore.getInstance("JKS");
            InputStream resourceAsStream = getClass().getResourceAsStream("/springblog.jks");
            keyStore.load(resourceAsStream, "secret".toCharArray());
        } catch (KeyStoreException | CertificateException | NoSuchAlgorithmException | IOException e) {
            throw new SpringRedditException("Exception occurred while loading keystore");
        }

    }

    public String generateToken(Authentication authentication) {
        org.springframework.security.core.userdetails.User principal = (User) authentication.getPrincipal();
        return Jwts.builder()
                .setSubject(principal.getUsername())
                .signWith(getPrivateKey())
                .compact();
    }

    private PrivateKey getPrivateKey() {
        try {
            return (PrivateKey) keyStore.getKey("springblog", "secret".toCharArray());
        } catch (KeyStoreException | NoSuchAlgorithmException | UnrecoverableKeyException e) {
            throw new SpringRedditException("Exception occured while retrieving public key from keystore");
        }
    }

    public boolean validateToken(String jwt) {
        parser().setSigningKey(getPublickey()).parseClaimsJws(jwt);
        return true;
    }

    private PublicKey getPublickey() {
        try {
            return keyStore.getCertificate("springblog").getPublicKey();
        } catch (KeyStoreException e) {
            throw new SpringRedditException("Exception occured while retrieving public key from keystore");
        }
    }

    public String getUsernameFromJWT(String token) {
        Claims claims = parser()
                .setSigningKey(getPublickey())
                .parseClaimsJws(token)
                .getBody();
        return claims.getSubject();
    }
}
```

- So the **JwtAuthenticationFilter** class extends the class **OncePerRequestFilter**, by extending this class we can implement our validation logic inside the **doFilterInternal()** method.
- Inside the **doFilterInternal** method, we are calling the **getJwtFromRequest** method which retrieves the Bearer Token (ie. JWT) from the **HttpServletRequest** object we are passing as input.
- Once we retrieve the token, we pass it to the **validateToken()** method of the **JwtProvider** class.
- The **validateToken** method uses the **JwtParser** class to validate our JWT. If you remember in the previous part, we created our JWT by signing it with the Private Key. Now we can use the corresponding Public Key, to validate the token.
- Once the JWT is validated, we retrieve the username from the token by calling the **getUsernameFromJWT()** method.
  - Once we get the username, we retrieve the user using the **UserDetailsService** class and store the user inside the **SecurityContext**

If you have any questions or not able to understand anything, just drop a comment below and I will help you.

Lastly, let's update our **SecurityConfig**. We have to add the filter to the **HttpSecurity** so that our **JwtAuthenticationFilter** will be executed first.

- First, let's Autowire the **JwtAuthenticationFilter** inside our **SecurityConfig.java** class.
- Then add the following code at the end of the **configure(HttpSecurity)**

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

This is how our SecurityConfig.java class looks like after adding the above code.

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)
```

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

```
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    public void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.csrf().disable()
                .authorizeRequests()
                .antMatchers("/api/auth/**")
                .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();
    }
}
```

Create API for Subreddits

Now to test the above JWT validation, we need a secured API, so let’s start by implementing the API to **Create** and **Read** Subreddits inside **SubredditController.java**

```
package com.example.springredditclone.controller;

import com.example.springredditclone.dto.SubredditDto;
import com.example.springredditclone.service.SubredditService;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/api/subreddit")
@AllArgsConstructor
public class SubredditController {

    private final SubredditService subredditService;

    @GetMapping
    public List<SubredditDto> getAllSubreddits() {
        return subredditService.getAll();
    }

    @GetMapping("/{id}")
    public SubredditDto getSubreddit(@PathVariable Long id) {
        return subredditService.getSubreddit(id);
    }

    @PostMapping
    public SubredditDto create(@RequestBody @Valid SubredditDto subredditDto) {
        return subredditService.save(subredditDto);
    }
}
```

SubredditService.java

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

import com.example.springredditclone.dto.SubredditDto;
import com.example.springredditclone.exceptions.SubredditNotFoundException;
import com.example.springredditclone.model.Subreddit;
import com.example.springredditclone.repository.SubredditRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

import static java.time.Instant.now;
import static java.util.stream.Collectors.toList;

@Service
@AllArgsConstructor
public class SubredditService {

    private final SubredditRepository subredditRepository;
    private final AuthService authService;

    @Transactional(readOnly = true)
    public List<SubredditDto> getAll() {
        return subredditRepository.findAll()
                .stream()
                .map(this::mapToDto)
                .collect(toList());
    }

    @Transactional
    public SubredditDto save(SubredditDto subredditDto) {
        Subreddit subreddit = subredditRepository.save(mapToSubreddit(subredditDto));
        subredditDto.setId(subreddit.getId());
        return subredditDto;
    }

    @Transactional(readOnly = true)
    public SubredditDto getSubreddit(Long id) {
        Subreddit subreddit = subredditRepository.findById(id)
                .orElseThrow(() -> new SubredditNotFoundException("Subreddit not found with id -" + id));
        return mapToDto(subreddit);
    }

    private SubredditDto mapToDto(Subreddit subreddit) {
        return SubredditDto.builder().name(subreddit.getName())
                .id(subreddit.getId())
                .postCount(subreddit.getPosts().size())
                .build();
    }

    private Subreddit mapToSubreddit(SubredditDto subredditDto) {
        return Subreddit.builder().name("/r/" + subredditDto.getName())
                .description(subredditDto.getDescription())
                .user(authService.getCurrentUser())
                .createdDate(now()).build();
    }
}
```

SubredditDto.java

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

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

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class SubredditDto {
    private Long id;
    private String name;
    private String description;
    private Integer postCount;
}
```

AuthService.java

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

import com.example.springredditclone.dto.AuthenticationResponse;
import com.example.springredditclone.dto.LoginRequest;
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.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
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;

    @Transactional
    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)
    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()));
    }

    @Transactional
    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 new AuthenticationResponse(token, loginRequest.getUsername());
    }
}
```

So we are basically exposing 3 REST calls for creating and reading Subreddits in our application. They are as follows:

Description	HTTP Method	API Mapping
Create Subreddit	POST	/api/subreddit
Get All Subreddits	GET	/api/subreddit
Get One Subreddit	GET	/api/subreddit/{id}

- For **getAllSubreddits()** and **getSubreddit()** call, it is straight forward. We are reading all/required the subreddits from DB, mapping them to **SubredditDTO** and returning them back to the client. As we just read the data here, we marked the **getAll()** and **getSubreddit()** **SubredditService.java** as @

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

- For **create()** call, we are mapping the data from **SubredditDto** to **Subreddit** and saving it in **SubredditRepository**.

Revisiting Project Structure

So this is how our project structure should look like, once you have created all the necessary services and components mentioned in this article.

Testing the Subreddit API and JWT Validation

Now let's test the subreddit API and JWT validation. Make sure you have started the application.

Log in and get Authentication Token

Once we received the **authenticationToken**, we can use this token when making REST calls to our Subreddit API.

Test Subreddit API without Access Token

First let us try to create a Subreddit without providing the access token, and let us see how our server responds. I will be passing on the below JSON as the body to create Subreddit.

```
{
        "name":"First Subreddit",
        "description":"This is description of First Subreddit"
}
```

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

So you can see that we received a **403 Forbidden** error as we did not provide any Access Token.

Test Subreddit API with the access token

Go to the **Authorization** tab inside Postman, and in the **Type** dropdown select the option **Bearer Token** and paste the access token you received from the server when you performed the login.

Now when you try to make the REST call again, you will receive an **HTTP Status 200** as the response.

Now let us also make GET call to retrieve all Subreddits:

What’s next?

So that is it for this article, in the next article we will concentrate mainly to build out our business logic ie. create APIs to create Post, Read Post, Submit Votes, etc.

Thank you for reading the article, and I wish you happy coding.

About the author
Sai Upadhyayula