PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 8

0 Comments

BY SAI UPADHYAYULA | March 10, 2020

Reddit clone with Spring boot and Angular
#8
Implement Logout using Refresh Tokens

Share Tweet Pin

Welcome to Part 8 of Build a Full Stack Reddit Clone with Spring boot and Angular series. In Part 7, we saw how to create an API to submit votes for posts in our application.

What are we building?

In this part, we will see how to implement API to logout/invalidation of our JSON Web Tokens (JWT) from our backend application. We will see what are the different options we have to implement this functionality.

If you are a visual learner like me you can check out the video version of this tutorial below:

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

How to implement logout?

If you are following this series, you are already familiar that JWT's(JSON Web Tokens) are mainly used in the authorization flows in the web application.

The user first provides the credentials to the server and the server responds back to the client with a JWT if the credentials are correct.

So the client uses this token to authorize itself for all the subsequent requests. Usually, if the client is a web browser, the token is stored in a form of browser storage.

So the obvious thing to do is to delete this token from the browser storage as part of the Logout implementation.

Is Logout just on the client-side enough?

Imagine if a hacker somehow got access to the token you are using. Then he/she can impersonate the user and make requests to the backend even after the user has chosen to logout.

How can I overcome this?

To overcome this shortcoming, we have below ways to implement Logout/JWT Invalidation in our application.

Introduce expiration of JWT

This looks like a valid point, we introduce expiry time for our tokens and after this time, the tokens are no longer valid. We ideally keep this expiration time short (15 minutes).

Once the token is expired it is no longer valid, forcing the user to authenticate to get a new token.

This is terrible user experience and a good way to drive away users from our application 😏

Store JWT inside Database

The next option is to store the token inside the database.This completely defeats the purpose of using JWT. JWT is by definition stateless and the advantage of using JWT is to bypass the database lookup when authorizing the client.

Implement Token Blacklisting

We can use a combination of the above-mentioned approaches to implement Token Blacklisting like below.

So the idea is to store the tokens inside an in-memory database like Redis.

When the user log's out from the browser we delete the token and store this token inside Redis. On each user request, we perform a lookup against Redis and if the token is found inside, we throw an exception.

We can also improve the performance even more by removing the expired tokens from the Redis database.

Introduce Refresh Tokens

The next approach is to introduce a concept called Refresh Tokens.

When the client first authenticates, the server provides an additional token called a Refresh Token(stored inside our database) additional to the short-lived JWT.

When our JWT is expired or about to be expired, we will use the refresh token to request a new JWT from the server.

In this way, we can keep on rotating the token until the user decides to logout from the application. Once the user logs out, we will also delete the refresh token from the database.

This leaves us with a very short window where the user logs out and the token is still valid.

As you can observe, there is no perfect way to implement logout when using JWT. There are multiple approaches we can choose, but each one of them has there disadvantages.

Image source dev.to

That is why I **strongly recommend using OAuth** to implement authentication and authorization in your application. In this way, you need not worry about how to handle user login and logout functionality.

Having said that, let us see how to implement the logout functionality in our application. We will use approach 4 (Refresh Tokens) to implement logout.

Updated Authentication/Authorization Flow

Here is how our updated Authentication and Authorization Flow looks like in our application by following the Refresh Token way.

I have already explained the existing Authentication/Authorization flow in Part 3. Now let's see the new points introduced here.

- In Step 2, after the authentication is successful, we send a Refresh Token along with the JWT back to the client.
- In Step 6, the client understands that the JWT is expired or about to be expired and request the server to provide a new JWT by including the Refresh Token inside the request.
- In Step 7, the server then verifies the Refresh Token by looking it up in the database and if it matches, generates a new JWT and responds back to the client (Step 8).

Implementation

Let's see how to implement the above Refresh Token Concepts in our project.

Step 1

Introducing expiration times for our JWT. As our JWT creation logic is inside **JwtProvider.java** we will make changes to it.

JwtProvider.java

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

import com.example.springredditclone.exceptions.SpringRedditException;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import org.springframework.beans.factory.annotation.Value;
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 java.sql.Date;
import java.time.Instant;

import static io.jsonwebtoken.Jwts.parser;
import static java.util.Date.from;
```

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

```java
@Service
public class JwtProvider {

    private KeyStore keyStore;
    @Value("${jwt.expiration.time}")
    private Long jwtExpirationInMillis;

    @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())
                .setIssuedAt(from(Instant.now()))
                .signWith(getPrivateKey())
                .setExpiration(Date.from(Instant.now().plusMillis(jwtExpirationInMillis)))
                .compact();
    }

    public String generateTokenWithUserName(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(from(Instant.now()))
                .signWith(getPrivateKey())
                .setExpiration(Date.from(Instant.now().plusMillis(jwtExpirationInMillis)))
                .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();
    }
    public Long getJwtExpirationInMillis() {
        return jwtExpirationInMillis;
    }
}
```

- So the first thing you can observe is we have introduced the field **jwtExpirationInMillis** which is being injected with property value **jwt.expiration.time** which is set to **900000** ms (15 minutes)
- We convert this expiration time which is in milliseconds to **Date** and passing that value to the **setExpiration()** while generating the JWT.
- We are also exposing the **jwtExpirationInMillis** field using the **getJwtExpirationInMillis()** method.

application.properties

```
############# Database Properties ###########################################
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/spring-reddit-clone
spring.datasource.username=root
spring.datasource.password=mysql
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.initialize=true
spring.jpa.show-sql=true
############# Mail Properties ###########################################
spring.mail.host=smtp.mailtrap.io
spring.mail.port=25
spring.mail.username=a7ca38e28c772f
spring.mail.password=8bf8e9295c7259
spring.mail.protocol=smtp
############ JWT Properties #####################
jwt.expiration.time=900000
```

Step 2

Implement logic to generate Refresh Tokens. Let's create a class called **RefreshTokenService.java** where we can implement the logic to manage our Refresh Tokens.

RefreshTokenService.java

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

import com.example.springredditclone.exceptions.SpringRedditException;
import com.example.springredditclone.model.RefreshToken;
import com.example.springredditclone.repository.RefreshTokenRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

@Service
@AllArgsConstructor
@Transactional
public class RefreshTokenService {

    private final RefreshTokenRepository refreshTokenRepository;

    RefreshToken generateRefreshToken() {
        RefreshToken refreshToken = new RefreshToken();
        refreshToken.setToken(UUID.randomUUID().toString());
        refreshToken.setCreatedDate(Instant.now());
```

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

```
        return refreshTokenRepository.save(refreshToken);
    }

    void validateRefreshToken(String token) {
        refreshTokenRepository.findByToken(token)
                .orElseThrow(() -> new SpringRedditException("Invalid refresh Token"));
    }

    public void deleteRefreshToken(String token) {
        refreshTokenRepository.deleteByToken(token);
    }
}
```

- The first method we have is **generateRefreshToken()** which creates a random 128 bit UUID String and we are using that as our token. We are setting the createdDate as **Instant.now()**. And then we are saving the token to our MySQL Database.
- Next, we have **validateRefreshToken()** which queries the DB with the given token. If the token is not found it throws an Exception with message – **“Invalid refresh Token”**
- Lastly, we have **deleteRefreshToken()** which as name suggests deletes the refresh token from the database.

Step 3

Enhance our login functionality so that it includes the generated Refresh Token inside the **AuthenticationResponse**.

The update **AuthenticationResponse.java** looks something like below:

AuthenticationResponse.java

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

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

import java.time.Instant;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class AuthenticationResponse {
    private String authenticationToken;
    private String refreshToken;
    private Instant expiresAt;
    private String username;
}
```

And this is how our **AuthService.java** looks like

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.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();
    }
}
```

If you compare the present version with the previous version, you can see that we are building the **AuthenticationResponse** object with additional fields **refresh token** and **expiration time**.

We have also created the **refreshToken()** method which first validates the token coming from **RefreshTokenRequest** object. If the validation is successful, we are generating the JWT again and including it in the **AuthenticationResponse**.

RefreshTokenRequest.java

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

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

import javax.validation.constraints.NotBlank;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class RefreshTokenRequest {
    @NotBlank
    private String refreshToken;
}
```

Step 4

Implement endpoints for Refresh Token and Logout.

AuthController.java

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

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.service.AuthService;
import com.example.springredditclone.service.RefreshTokenService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

import static org.springframework.http.HttpStatus.OK;

@RestController
@RequestMapping("/api/auth")
@AllArgsConstructor
public class AuthController {

    private final AuthService authService;
    private final RefreshTokenService refreshTokenService;

    @PostMapping("/signup")
    public ResponseEntity<String> signup(@RequestBody RegisterRequest registerRequest) {
        authService.signup(registerRequest);
        return new ResponseEntity<>("User Registration Successful",
                OK);
    }

    @GetMapping("accountVerification/{token}")
    public ResponseEntity<String> verifyAccount(@PathVariable String token) {
        authService.verifyAccount(token);
        return new ResponseEntity<>("Account Activated Successfully", OK);
    }

    @PostMapping("/login")
    public AuthenticationResponse login(@RequestBody LoginRequest loginRequest) {
        return authService.login(loginRequest);
    }

    @PostMapping("refresh/token")
    public AuthenticationResponse refreshTokens(@Valid @RequestBody RefreshTokenRequest refreshTokenRequest) {
        return authService.refreshToken(refreshTokenRequest);
    }

    @PostMapping("/logout")
    public ResponseEntity<String> logout(@Valid @RequestBody RefreshTokenRequest refreshTokenRequest) {
        refreshTokenService.deleteRefreshToken(refreshTokenRequest.getRefreshToken());
        return ResponseEntity.status(OK).body("Refresh Token Deleted Successfully!!");
    }

}
```

- We have to new endpoints **/api/auth/refresh/token** and **/api/auth/logout**
- The Refresh Token API calls the **refreshToken** method inside the **AuthService**
- The Logout API call just calls the **deleteRefreshToken** method inside the **RefreshToke**

Testing Time

It’s time to test our implementation, so start your application and let’s fire up POSTMan Client.

First, we will log in to our application and get our JWT, for demonstration purposes I have changed the expiration time of our JWT to 1.5 minutes instead of 15 minutes

- You can observe we received **authenticationToken**, **refreshToken** and **expiresAt** and **username** fields as a response back from the Login API call.

If we use the **authenticationToken** and make calls to any endpoint, we should receive the response without any errors. Once the expiration time is reached, if we try to call the **/api/posts/** endpoint, we will receive an error message as shown in below image:

Now let’s make a call to get new JWT using the refresh token.

You can see that we received a new JWT along with the refresh token we included in the request. Now let’s try to call the Logout endpoint and then try to Refresh the JWT.

So you can observe when we tried to refresh our JWT, we received an error **"Invalid refresh Token"** because our Refresh Token was already deleted.

Conclusion

I hope you were able to understand the concept of Refresh Tokens and how to implement them. And with this article, we complete the Backend implementation part for our Reddit Clone application.

In the next article, we will start implementing Frontend using Angular 9.

Stay tuned for that, and **Happy Coding Techies!!**

About the author

Sai Upadhyayula