PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 3

1 Comments

BY SAI UPADHYAYULA | November 8, 2019

Share Tweet Pin

Welcome to Part 3 of my Build a Full Stack Reddit Clone with Spring Boot and Angular series.

In Part 2 we have set up Spring Security in our application and created functionality to register users and activate the accounts through email verification. In this article, we will see how to write API for Login functionality using the Token-Based Authentication System with the help of JWT (JSON Web Tokens).

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

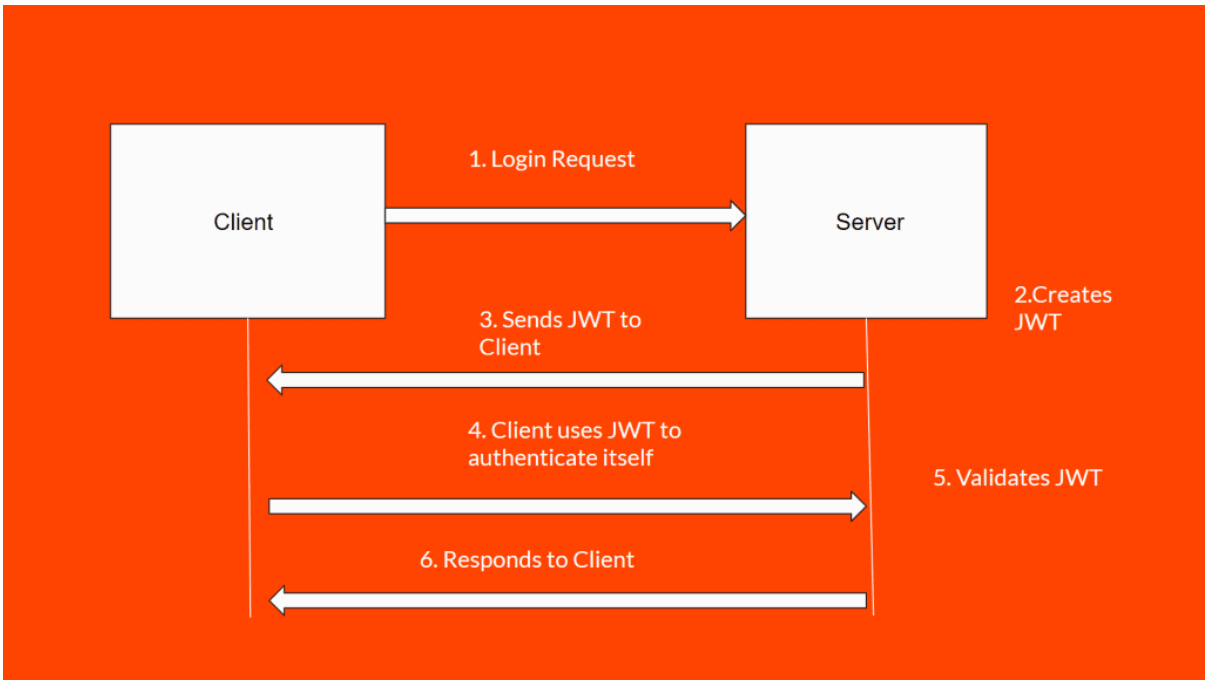
> The source code of this project, is hosted on Github –
> Backend – https://github.com/SaiUpadhyayula/spring-reddit-clone
> Frontend – https://github.com/SaiUpadhyayula/angular-reddit-clone

An overview of what we're going to build.

- First, we will see how the authentication mechanism works in our application.
- We will see what are the components used to build the Login API.
- We will implement the Login API using JWT Authentication and test it.

Authentication Flow

First, let us see how the authentication flow will be in our application. The below picture should represent the complete authentication flow.



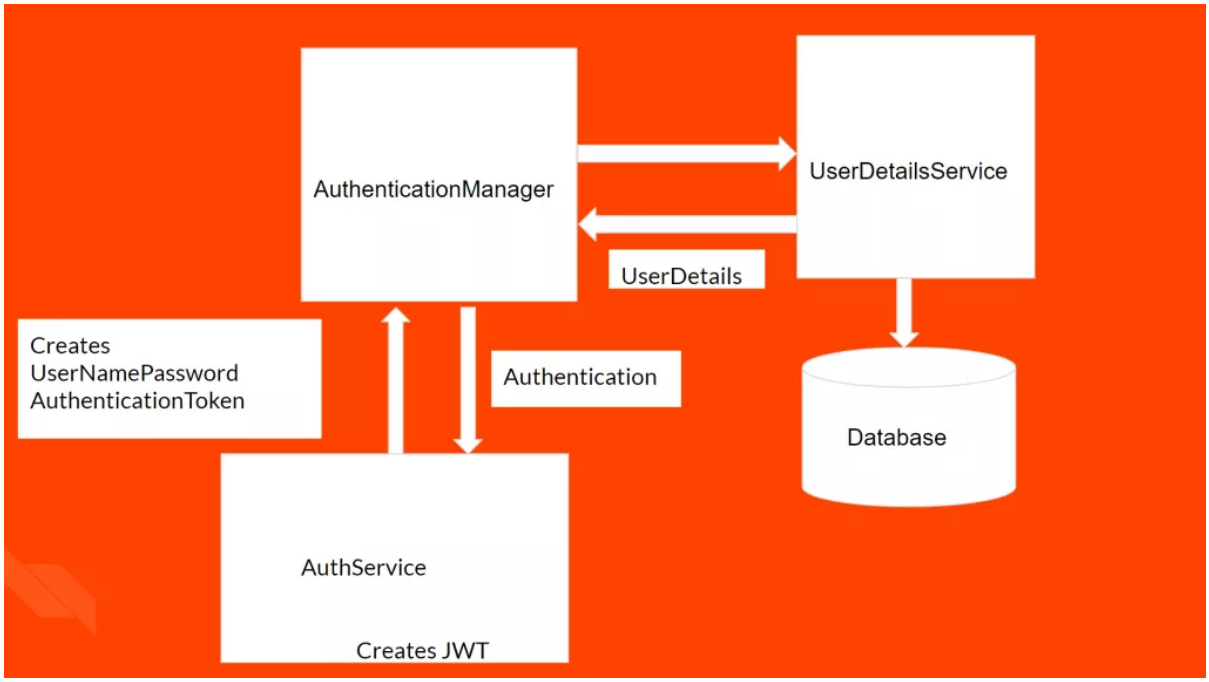
1. So it starts with the Client sending a login request to the server.
2. The server checks the credentials provided by the user, if the credentials are right, it creates a JSON Web Token

(JWT).

3. It responds with a success message (HTTP Status 200) and the JWT.
4. The client uses this JWT in all the subsequent requests to the user, it provides this JWT as an Authorization header with Bearer authentication scheme.
5. When the server, receives a request against a secured endpoint, it checks the JWT and validates whether the token is generated and signed by the server or not.
6. If the validation is successful, the server responds accordingly to the client.

Spring Security Authentication Flow

Let us see what are the different components involved in the authentication mechanism inside our backend.



- The login request is received by **AuthenticationController** and is passed on to the **AuthService** class.
- This class creates an object of type **UserNamePasswordAuthenticationToken** which encapsulates the username and password provided by the user as part of the login request.
- Then this is passed on to **AuthenticationManager** which takes care of the authentication part when using Spring Security. It implements lot of functionality in the background and provides us nice API we can use.
- The **AuthenticationManager** further interacts with an interface called **UserDetailsService**, this interface as the name suggests deals with user data. There are several implementations that can be used depending on the kind of authentication we want. There is support for in-memory authentication, database-authentication, LDAP based authentication.
- As we store our user information inside the Database, we used Database authentication, so the implementation access the database and retrieves the user details and passes **UserDetails** back to **AuthenticationManager**.
- The **AuthenticationManger** now checks the credentials, and if they match it creates an object of type **Authentication** and passes it back to the **AuthService** class.
- Then we create the JWT and respond back to the user.

I hope this provides the necessary high-level overview of how we are going to implement the authentication mechanism. Now without any further ado, let's start coding!!

Configure SecurityConfig with AuthenticationManager

The first thing we have to do is, change our **SecurityConfig** class and add the necessary configuration for

AuthenticationManger, the configuration file looks like below:

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

import lombok.AllArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
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;

@EnableWebSecurity
@AllArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;

    @Bean(BeanIds.AUTHENTICATION_MANAGER)
    @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();
    }

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

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

We have defined a bean for **AuthenticationManager** and configured **AuthenticationManagerBuilder** to use the **UserDetailsService** interface to check for user information.

As **UserDetailsService** is an interface, we have to provide an implementation where it fetches the user information from our MySQL Database.

Defining custom UserDetailsService class

This is how our **UserDetailsServiceImpl** class looks like:

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

```
import com.example.springredditclone.model.User;
import com.example.springredditclone.repository.UserRepository;
import lombok.AllArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Collection;
import java.util.Optional;

import static java.util.Collections.singletonList;

@Service
@AllArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;

    @Override
    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username) {
        Optional<User> userOptional = userRepository.findByUsername(username);
        User user = userOptional
                .orElseThrow(() -> new UsernameNotFoundException("No user " +
                        "Found with username : " + username));

        return new org.springframework.security
                .core.userdetails.User(user.getUsername(), user.getPassword(),
                user.isEnabled(), true, true,
                true, getAuthorities("USER"));
    }

    private Collection<? extends GrantedAuthority> getAuthorities(String role) {
        return singletonList(new SimpleGrantedAuthority(role));
    }
}
```

The class overrides the method **loadUserByUsername()** which is used by Spring Security to fetch the user details. Inside the method, we are querying the **UserRepository** and fetching those details and wrapping them in another User object which implements the **UserDetails** interface.

Define REST Endpoint for Login

Let's define an endpoint for our Login API.

AuthController.java

```
package com.programming.techie.springredditclone.controller;

import com.programming.techie.springredditclone.dto.AuthenticationResponse;
import com.programming.techie.springredditclone.dto.LoginRequest;
import com.programming.techie.springredditclone.dto.RegisterRequest;
import com.programming.techie.springredditclone.service.AuthService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/api/auth")
@AllArgsConstructor
```

```
public class AuthController {

    private final AuthService authService;

    @PostMapping("/signup")
    public ResponseEntity signup(@RequestBody RegisterRequest registerRequest) {
        authService.signup(registerRequest);
        return new ResponseEntity(OK);
    }

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

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

AuthenticationResponse.java

```
package com.programming.techie.springredditclone.dto;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class AuthenticationResponse {
    private String authenticationToken;
    private String username;
}
```

LoginRequest.java

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

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

@Data
@AllArgsConstructor
@NoArgsConstructor
public class LoginRequest {

    private String username;
    private String password;
}
```

The login endpoint accepts **POST** request and passes the **LoginRequest** object to the **login()** method of the **AuthService** class

AuthService.java

```
package com.programming.techie.springredditclone.service;

import com.programming.techie.springredditclone.dto.AuthenticationResponse;
```

```
import com.programming.techie.springredditclone.dto.LoginRequest;
import com.programming.techie.springredditclone.dto.RegisterRequest;
import com.programming.techie.springredditclone.exception.SpringRedditException;
import com.programming.techie.springredditclone.model.NotificationEmail;
import com.programming.techie.springredditclone.model.User;
import com.programming.techie.springredditclone.model.VerificationToken;
import com.programming.techie.springredditclone.repository.UserRepository;
import com.programming.techie.springredditclone.repository.VerificationTokenRepository;
import com.programming.techie.springredditclone.security.JwtProvider;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
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.util.Optional;
import java.util.UUID;

import static com.programming.techie.springredditclone.util.Constants.ACTIVATION_EMAIL;
import static java.time.Instant.now;

@Service
@AllArgsConstructor
@Slf4j
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final AuthenticationManager authenticationManager;
    private final JwtProvider jwtProvider;
    private final MailContentBuilder mailContentBuilder;
    private final MailService mailService;
    private final VerificationTokenRepository verificationTokenRepository;

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

        userRepository.save(user);
        log.info("User Registered Successfully, Sending Authentication Email");
        String token = generateVerificationToken(user);
        String message = mailContentBuilder.build("Thank you for signing up to Spring Reddit, please click on
the below url to activate your account : "
                + ACTIVATION_EMAIL + "/" + token);

        mailService.sendMail(new NotificationEmail("Please Activate your account", user.getEmail(), message));
    }

    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;
    }

    private String encodePassword(String password) {
```

```
        return passwordEncoder.encode(password);
    }

    public AuthenticationResponse login(LoginRequest loginRequest) {
        Authentication authenticate = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
                loginRequest.getPassword()));
        SecurityContextHolder.getContext().setAuthentication(authenticate);
        String authenticationToken = jwtProvider.generateToken(authenticate);
        return new AuthenticationResponse(authenticationToken, loginRequest.getUsername());
    }

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

    @Transactional
    private void fetchUserAndEnable(VerificationToken verificationToken) {
        String username = verificationToken.getUser().getUsername();
        User user = userRepository.findByUsername(username).orElseThrow(() -> new SpringRedditException("User
Not Found with id - " + username));
        user.setEnabled(true);
        userRepository.save(user);
    }
}
```

Creating JWT

We will use the service class **JWTProvider** to create our JWT.

JWTProvider.java

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

import com.example.springredditclone.exceptions.SpringRedditException;
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;

@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");
        }

    }
```

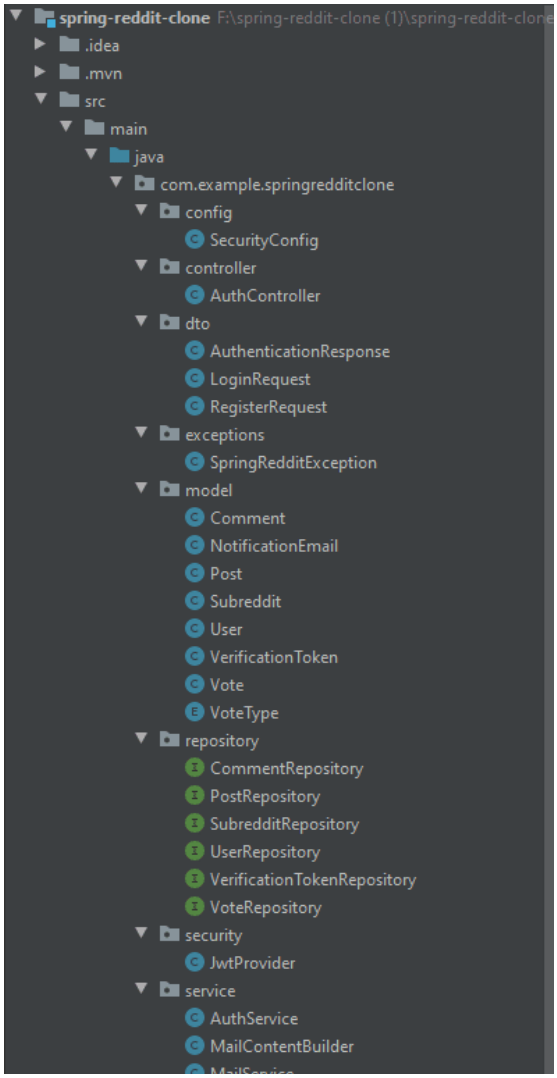
```
    public String generateToken(Authentication authentication) {
        org.springframework.security.core.userdetails.User principal = (User) authentication.getPrincipal();
        return Jwts.builder()
                .setSubject(principal.getUsername())
                .signWith(getPrivateKey())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

We are using **AsymmetricEncryption** to sign our JWT's using Java Keystore (using Public-Private Key)

I have already created a Java Keystore and used the Private Key inside this Keystore to sign the JWT. Here I am reusing the Keystore which I created when creating the video Build a Blog tutorial using Springboot and Angular

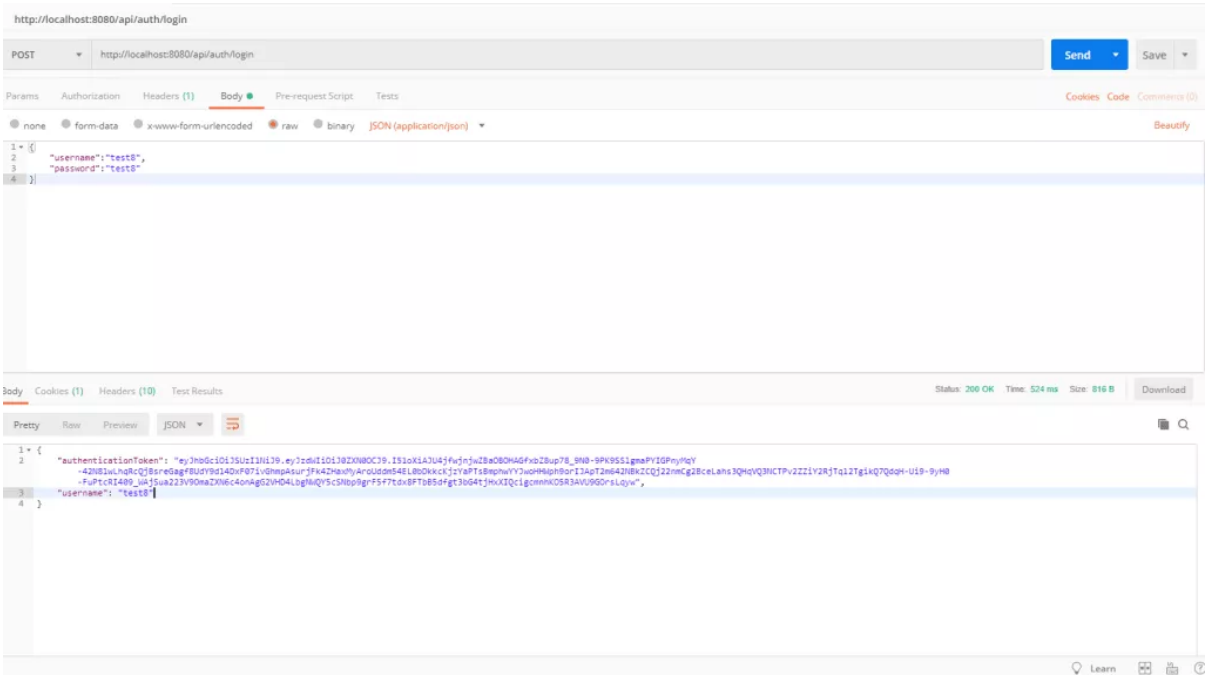
Revisiting Project Structure

This is how our project structure looks like after implementing the Login API.



Testing Login API

Start the application at port 8080, open postman and make a POST call like below:



You can see in the above image, we received back two fields, our authentication-token (JWT) and the username.

This concludes this blog post, in the next part, we will see how to validate the JWT and create posts in our Reddit Application. Until then stay tuned and happy coding!!!

About the author

Sai Upadhyayula

Mel

November 20, 2019 at 12:23 am

Please, don't stop posting this tutorial!

Comments are closed.