PROGRAMMING TECHIE

Build a Full Stack Reddit Clone with – Spring boot and Angular – Part 2

1 Comments

BY SAI UPADHYAYULA

October 9, 2019

Reddit clone with Spring boot and Angular
#2
User Registration & Verification

Share Tweet Pin

Welcome to Part 2 of my Build a Full Stack Reddit Clone with Spring Boot and Angular series.

In Part 1 we created our project and basic domain entities and repositories. In this article, we will set up Spring Security and implement the API to register users in our application. We will also add the functionality to send out emails to the user for Account Activation.

If you are a visual learner like me you can check out the video tutorial on youtube 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

What are we going to build in this part?

- First, we will see how to setup Spring Security in our application
- Then, we will build an API to register users in our application
- We will see how to encode the password of the user, before storing them in the database.
- We will send account activation emails to the user.
- Build an API to verify the users and enable them.

So let's start coding.

Configure Spring Security

Let's create the following class under the new package called as **config,** this class holds the complete security configuration of our application, that's why the class is named **SecurityConfig.java**

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

import org.springframework.context.annotation.Bean;
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.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

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

Let's understand what did we configure in the above class:

1. **@EnableWebSecurity**
   This is the main annotation which enables the Web Security module in our Project.
2. **WebSecurityConfigurerAdapter**
   This is the base class for our **SecurityConfig** class, it provides us the default security configurations, which we can override in our **SecurityConfig** and customize them.
3. **Configurations**
   Next, we have the configure method which we have overridden from the base class which takes **HttpSecurity** as an argument.
   Here, we are configuring Spring to allow all the requests which match the endpoint **"/api/auth/**"** , as these endpoints are used for authentication and registration we don't expect the user to be authenticated at that point of time.
4. **PasswordEncoder**
   Now before storing the user in the database, we ideally want to encode the passwords. One of the best hashing algorithms for passwords is the **Bcrypt Algorithm**. We are using the **BCryptPasswordEncoder** class provided by Spring Security. You can read more about this here

Now that is the minimum configuration we need to implement the Registration Functionality, in the upcoming blog posts, we will add more functionality to our **SecurityConfig** class as we implement the Login functionality using **JWT**. Let's write the API for registering users.

Implement API to Register Users

So to register users, first of all, we need a Controller class, I will create it inside the following package: **com.example.springredditclone.controller**.
I will name our class file as **AuthController.java**

- **AuthController.java**

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

import com.example.springredditclone.dto.RegisterRequest;
import com.example.springredditclone.service.AuthService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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);
    }
}
```

So we have created a RestController and inside this controller, we first created a method that will be invoked whenever a POST request is made to register the user's in our application.

The API call should contain the request body which is of type **RegisterRequest**. Through this class we are transferring the user details like username, password and email as part of the **RequestBody**. We call this kind of classes as a DTO (Data Transfer Object). We will create this class inside a different package called **com.example.springredditclone.dto**

- **RegisterRequest.java**

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

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

@Data
@AllArgsConstructor
@NoArgsConstructor
public class RegisterRequest {
    private String username;
    private String email;
    private String password;
}
```

The signup method inside the **AuthController** is calling another method inside the AuthService class, which is mainly responsible to create the **User** object and storing it in the database. Now let's create this **AuthService** class inside the package **com.example.springredditclone.service**.

- **AuthService.java**

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

import com.example.springredditclone.dto.RegisterRequest;
import com.example.springredditclone.model.User;
import com.example.springredditclone.repository.UserRepository;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import static java.time.Instant.now;

@Service
@AllArgsConstructor
@Slf4j
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    @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);
    }

    private String encodePassword(String password) {
        return passwordEncoder.encode(password);
    }
}
```

Inside the **AuthService** class, we are mapping the **RegisterRequest** object to the **User** object and when setting the password, we are calling the **encodePassword()** method. This method is using the **BCryptPasswordEncoder** to encode our password. After that, we save the user into the database. Note that we are setting the **enabled** flag as false, as we want to disable the user after registration, and we only enable the user after verifying the user's email address.

> If you are stuck for any reason, comment on this blog post and let me know the error you are facing, I will try to help fix the issue.

Activating new account via email

This brings us to the next section. Now, let us enhance the registration process by only allowing the user to log in after they verify their email. We will generate a verification token, right after we save the user to the database and send that token as part of the verification email. Once the user is verified, then we enable the user to login to our application.

Let us enhance our **AuthService.java** by adding the logic to generate a verification token. This is how our class looks like:

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

import com.example.springredditclone.dto.RegisterRequest;
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 lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
```

```java
import java.util.UUID;

import static java.time.Instant.now;

@Service
@AllArgsConstructor
@Slf4j
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    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);

        String token = generateVerificationToken(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;
    }

    private String encodePassword(String password) {
        return passwordEncoder.encode(password);
    }
}
```

We added the **generateVerificationToken()** method and calling that method right after we saved the user into **UserRepository**. Note that, we are creating a random UUID as our token, creating an object for **VerificationToken,** fill in the data for that object and save it into the **VerificationTokenRepository**. As we have the token, now its time to send an email that contains this verification token.

We need to add some additional dependencies to our project, if we want to send HTML emails from our application, **Thymeleaf** provides us the template engine, which we can use to create HTML templates and use those templates to send the emails. Let's add the below thymeleaf dependency to our **pom.xml**

```xml
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Let's add the below classes which contain the logic to build the content of our email and send the email to the user.

- **MailContentBuilder.java**

This class contains the logic to create our email message using the HTML template we are going to provide.

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

import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.thymeleaf.TemplateEngine;
import org.thymeleaf.context.Context;

@Service
@AllArgsConstructor
class MailContentBuilder {

    private final TemplateEngine templateEngine;

    String build(String message) {
        Context context = new Context();
        context.setVariable("message", message);
        return templateEngine.process("mailTemplate", context);
    }
}
```

MailContentBuilder.java contains the method **build()** which takes our email message as input and it uses the **Thymeleaf**'s **TemplateEngine** to generate the email message. Note that we gave the name **mailTemplate** as an argument to the method call **templateEngine.process("mailTemplate", context);** That would be the name of the html template which looks like below:

- **mailTemplate.html**

Create this html file under **src/main/resources/templates**

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head></head>
<body>
<span th:text="${message}"></span>
</body>
</html>
```

We are injecting the email message into the HTML template by setting the message into the **Context** of the **TemplateEngine**.

```
Context context = new Context();
context.setVariable("message", message);
```

Let's generate the email message and call the **build()** method of the **MailContentBuilder.java** class from **AuthService**.

```
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);
```

For the activation URL, as we are using this from our local machines let's provide the URL as – **"http://localhost:8080/api/auth/accountVerification"**. Let us create this value as a constant **ACTIVATION_EMAIL**. Let's create a separate class to maintain all these constants in our application in one place.

Constants.java

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

import lombok.experimental.UtilityClass;

@UtilityClass
public class Constants {
    public static final String ACTIVATION_EMAIL = "http://localhost:8080/api/auth/accountVerification";
}
```

Note that, we are creating this class inside the package **com.programming.techie.springredditclone.util**.

As this is a Utility Class, we have annotated this class with **@UtilityClass** which is a Lombok annotation, this annotation will make the following changes at compile time to our class:

- Marks the class as final.
- It generates a private no-arg constructor.
- It only allows the methods or fields to be static.

A Utility class, by definition, should not contain any state. Hence it is usual to put shared constants or methods inside utility class so that they can be reused. As they are shared and not tied to any specific object it makes sense to mark them as static.

We have generated our email message, now its time to send this email to the user.

Using MailTrap to send emails

In this application, we will use a Fake SMTP server called MailTrap to check whether our email functionality is working or not. Create an account in Mailtrap and after registration, you should get the following details, which you can use to send emails in our application.

```
Host:smtp.mailtrap.io
Port:25 or 465 or 587 or 2525
Username:<your-username>
Password:<your-password>
Auth:PLAIN, LOGIN and CRAM-MD5
TLS:Optional (STARTTLS on all ports)
```

The above details can be configured in the **application.properties** file like below:

```
############# Mail Properties ###########################################
spring.mail.host=smtp.mailtrap.io
spring.mail.port=25
spring.mail.username=<your-username>
spring.mail.password=<your-password>
spring.mail.protocol=smtp
```

Now let's create the below class called **MailService**, to send out the emails.

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

import com.programming.techie.springredditclone.exception.SpringRedditException;
import com.programming.techie.springredditclone.model.NotificationEmail;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.mail.MailException;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.mail.javamail.MimeMessagePreparator;
import org.springframework.stereotype.Service;

@Service
@AllArgsConstructor
@Slf4j
class MailService {

    private final JavaMailSender mailSender;
    private final MailContentBuilder mailContentBuilder;

    void sendMail(NotificationEmail notificationEmail) {
        MimeMessagePreparator messagePreparator = mimeMessage -> {
            MimeMessageHelper messageHelper = new MimeMessageHelper(mimeMessage);
            messageHelper.setFrom("springreddit@email.com");
            messageHelper.setTo(notificationEmail.getRecipient());
            messageHelper.setSubject(notificationEmail.getSubject());
            messageHelper.setText(mailContentBuilder.build(notificationEmail.getBody()));
        };
        try {
            mailSender.send(messagePreparator);
            log.info("Activation email sent!!");
        } catch (MailException e) {
            throw new SpringRedditException("Exception occurred when sending mail to " +
notificationEmail.getRecipient());
        }
    }

}
```

Let us understand what we are doing in this class, so we have our **sendMail** method which takes **NotificationEmail** as input, and inside the method we are creating a **MimeMessage** by passing in the sender, recipient, subject and body fields. The message body we are receiving from the **build()** method of our **MailContentBuilder** class.

Once the email message is prepared, we send the email message. If there are any unexpected exceptions raised during sending the email, we are catching those exceptions and rethrowing them as custom exceptions. This is a good practice as we don't expose the internal technical exception details to the user, by creating custom exceptions, we can create our own error messages and provide them to the users. Let's create our custom exception **SpringRedditException** class inside the **com.example.springredditclone.exception** package:

```
package com.example.springredditclone.exception;

public class SpringRedditException extends RuntimeException {
    public SpringRedditException(String message) {
        super(message);
    }
}
```

Finally, let's call the **sendMail** method from our **signup** method inside the **AuthService** class.

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

At the end this is how our AuthService class should look like:

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

import com.example.springredditclone.dto.RegisterRequest;
import com.example.springredditclone.exception.SpringRedditException;
import com.example.springredditclone.model.NotificationEmail;
import com.example.springredditclone.model.User;
import com.example.springredditclone.model.VerificationToken;
```

```java
import com.example.springredditclone.repository.UserRepository;
import com.example.springredditclone.repository.VerificationTokenRepository;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
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.example.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 VerificationTokenRepository verificationTokenRepository;
    private final MailContentBuilder mailContentBuilder;
    private final MailService mailService;

    @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);

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

    @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);
    }
}
```

Create Endpoint to Verify Users

So we have created the logic to send out emails after registration, now let's create an endpoint to Verify Users. Add the below method to **AuthController**

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

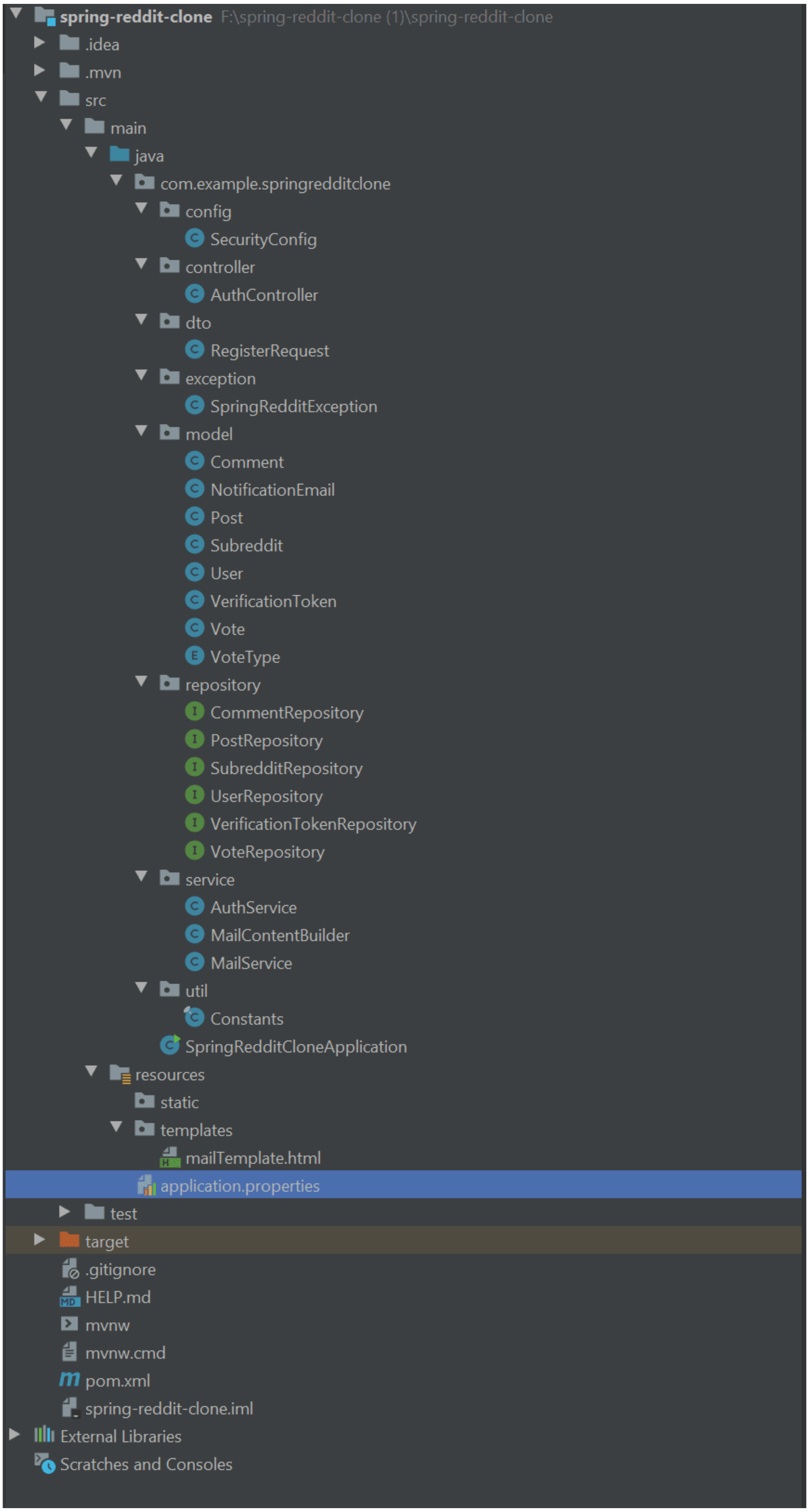
Let's also update the **AuthService** class.

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

    @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);
    }
```

Now we are ready to test the user registration and verification functionalities. Before we test the API's let's revisit the project structure.

Revisiting Project Structure

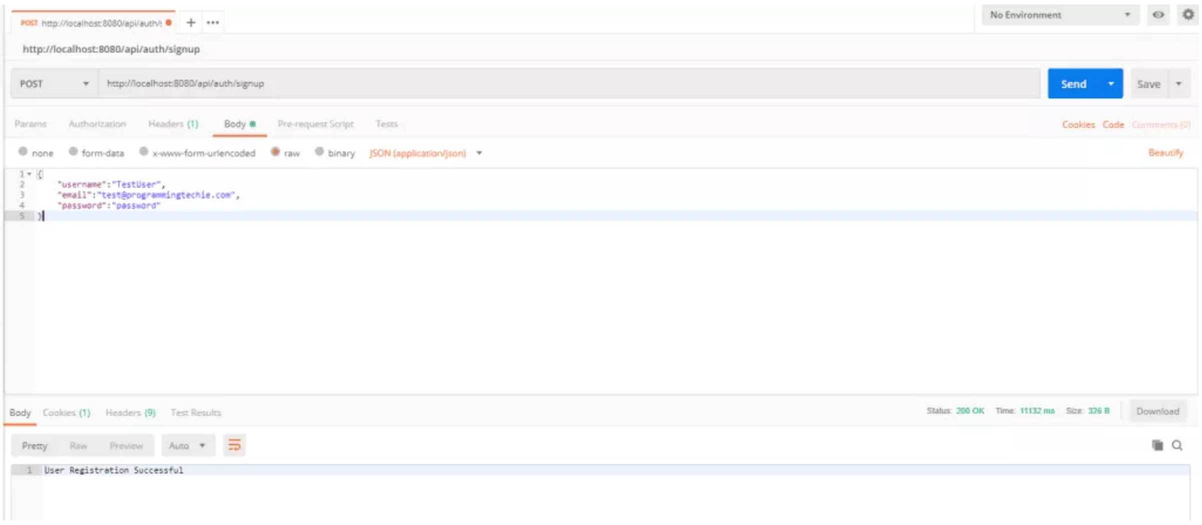
We have created lots of code and created some new classes, this is how your project structure should look like:



```
spring-reddit-clone  F:\spring-reddit-clone (1)\spring-reddit-clone
  .idea
  .mvn
  src
    main
      java
        com.example.springredditclone
          config
            SecurityConfig
          controller
            AuthController
          dto
            RegisterRequest
          exception
            SpringRedditException
          model
            Comment
            NotificationEmail
            Post
            Subreddit
            User
            VerificationToken
            Vote
            VoteType
          repository
            CommentRepository
            PostRepository
            SubredditRepository
            UserRepository
            VerificationTokenRepository
            VoteRepository
          service
            AuthService
            MailContentBuilder
            MailService
          util
            Constants
          SpringRedditCloneApplication
      resources
        static
        templates
          mailTemplate.html
        application.properties
    test
  target
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
  spring-reddit-clone.iml
External Libraries
Scratches and Consoles
```

Testing the API's

Now let's test the user signup and verification API's, make sure you are running the application, open PostMan and make the following calls to the endpoint – **http://localhost:8080/api/auth/signup**



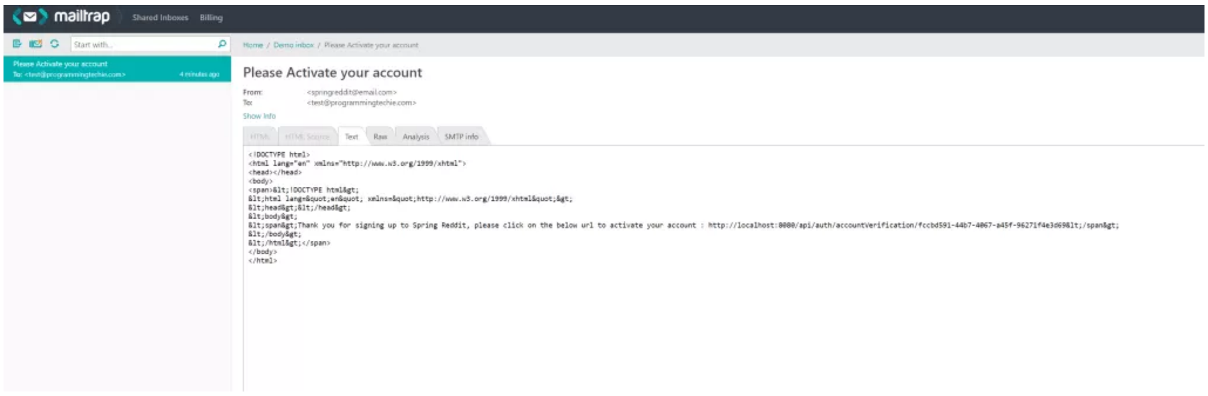
POST http://localhost:8080/api/auth/signup

```
{
    "username":"TestUser",
    "email":"test@programmingtechie.com",
    "password":"password"
}
```

Status: 200 OK Time: 11132 ms Size: 326 B

```
User Registration Successful
```

Make sure you create the **RequesBody** as seen in the image and call the API, you should receive an HTTP OK status back from the API, with message **User Registration Successful**. If you have already set up the account and used those details, you should receive an email, check your inbox, you should see an email like below:



Please Activate your account
To: <test@programmingtechie.com> 4 minutes ago

Please Activate your account

From: <springreddit@email.com>
To: <test@programmingtechie.com>

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head></head>
<body>
<span>&lt;!DOCTYPE html&gt;
&lt;html lang=&quot;en&quot; xmlns=&quot;http://www.w3.org/1999/xhtml&quot;&gt;
&lt;head&gt;&lt;/head&gt;
&lt;body&gt;
&lt;span&gt;Thank you for signing up to Spring Reddit, please click on the below url to activate your account : http://localhost:8080/api/auth/accountVerification/fccbd591-44b7-4067-a45f-96271f4e3d698lt;/span&gt;
&lt;/body&gt;
&lt;/html&gt;</span>
</body>
</html>
```

Send emails asynchronously

If you observe the logs, you can see that there is a slight delay to send email after saving the user in the database. You can check this in the logs.

```
2019-10-09 21:05:08.845  INFO 14756 --- [nio-8080-exec-1] c.e.s.service.AuthService                : User
Registration Successful, Sending Activation Email
Hibernate: insert into token (expiry_date, token, user_user_id) values (?, ?, ?)
2019-10-09 21:05:19.677  INFO 14756 --- [nio-8080-exec-1] c.e.s.service.MailService                :
Activation email sent!!
```

If you check the timestamp, we have a delay of more than 10 seconds to send out the email, that means even though the registration is completed, the user has to wait 10 more seconds to see the response. Even though this is not much time, in the real world situation we should handle the Email Sending functionality Asynchronously, we can also handle it by using a Message Queue like RabbitMQ, but I think that would be an overkill for our use-case. Let's enable the Async module in spring by adding the **@EnableAsync** to our Main class

```
package com.example.springredditclone;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;
```

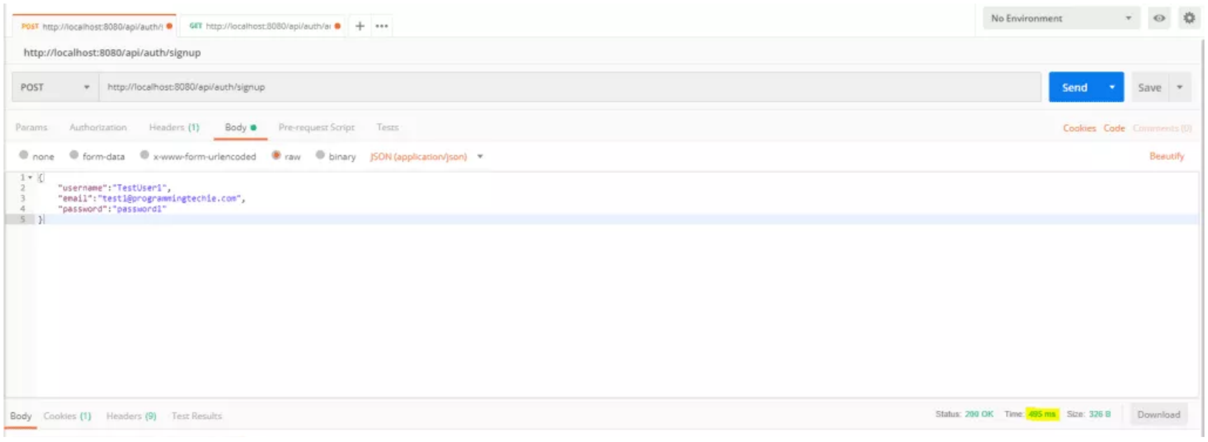
```
@SpringBootApplication
@EnableAsync
public class SpringRedditCloneApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringRedditCloneApplication.class, args);
    }

}
```

Also, add the @Async to our **sendMail** method inside the **MailService** class. Here is how our **MailService** class looks like:

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

import com.example.springredditclone.exception.SpringRedditException;
import com.example.springredditclone.model.NotificationEmail;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.mail.MailException;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.mail.javamail.MimeMessagePreparator;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

@Service
@AllArgsConstructor
@Slf4j
class MailService {

    private final JavaMailSender mailSender;
    private final MailContentBuilder mailContentBuilder;

    @Async
    void sendMail(NotificationEmail notificationEmail) {
        MimeMessagePreparator messagePreparator = mimeMessage -> {
            MimeMessageHelper messageHelper = new MimeMessageHelper(mimeMessage);
            messageHelper.setFrom("springreddit@email.com");
            messageHelper.setTo(notificationEmail.getRecipient());
            messageHelper.setSubject(notificationEmail.getSubject());
            messageHelper.setText(mailContentBuilder.build(notificationEmail.getBody()));
        };
        try {
            mailSender.send(messagePreparator);
            log.info("Activation email sent!!");
        } catch (MailException e) {
            throw new SpringRedditException("Exception occurred when sending mail to " +
notificationEmail.getRecipient());
        }
    }

}
```

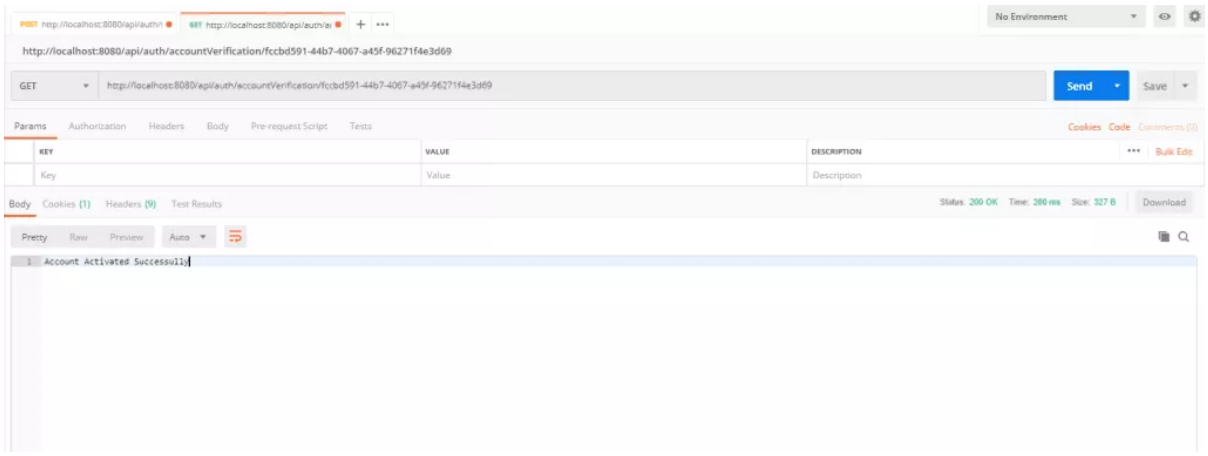
Testing Response time after activating Async

Now if you try to register another user, you can observe that we got back the response pretty quickly compared to our previous request.



```
{
    "username":"TestUser1",
    "email":"test1@programmingtechie.com",
    "password":"password1"
}
```

Status: 200 OK


GET http://localhost:8080/api/auth/accountVerification/fccbd591-44b7-4067-a45f-96271f4e3d69
Send
Save
Params
Body
Pretty

```
Account Activated Successully
```

Status: 200 OK

You should receive the message – “Account Activated Successfully” back from the server.

In the next blog post, we will see how to implement Login API, we will also set up JWT in our backend.

If you don’t want to miss any blog posts in this series, make sure to subscribe to my blog.

I will see you in the next blog post, until then **Happy Coding!!**

About the author

Sai Upadhyayula

Enhsuld

October 10, 2019 at 3:57 am

Thanks for the tutorial.

Comments are closed.