Curso: Spring Boot com Angular 7
https://www.udemy.com/user/hugo-silva
Prof. Hugo Silva

Capítulo: Serviço de email e Verification Token

Objetivo geral:

- Criar um serviço de email
- Criar uma operação de envio de confirmação de usuário registrado por email
- Criar o SmtpEmailService com SMTP do Google
- Criar uma classe para verificar se o token do usuário está valido
- Demonstrar uma implementação flexível e elegante com padrões de projeto (Strategy e Template Method)

Criando Entity VerificationToken

Checklist para criar entidades:

- Atributos básicos
- Construtores
- Getters e setters
- hashCode e equals (implementação padrão: somente id)
- Serializable (padrão: 1L) : private static final long serialVersionUID = 1L;

Checklist:

- No subpacote **domain**, criar a classe **VerificationToken**
- Em **VerificationToken**, incluir a anotação @Document, @Id e @DBRef+(lazy = true) para indicar que se trata de uma coleção do **MongoDB**
- Implementar o método para calcular a data que vai expirar o token

```
private Date calculateExpiryDate(final int expiryTimeInMinutes) {}
```

Modificar os construtores:

```
public VerificationToken(final String token) {
    (...)
    this.expiryDate = calculateExpiryDate(EXPIRATION);
}
```

```
public VerificationToken(final String token, final User user) {
   (...)
   this.expiryDate = calculateExpiryDate(EXPIRATION);
}
```

Conectando ao MongoDB com repository

Checklist:

- No pacote **repository**, criar a interface **VerificationTokenRepository** e criar os métodos

```
VerificationToken findByToken(String token)
```

```
VerificationToken findByUser(User user)
```

Endpoint para registrar usuários

Checklist:

- No pacote **resources**, criar uma classe **RegistrationResource** e implementar nela o seguinte endpoint

```
@PostMapping("/public/registration/users")
public ResponseEntity<Void> registerUser(@RequestBody UserDTO userDTO){}
```

- Antes de implementar o endpoint acima, atualizar a classe **UserDTO** com os atributos password, enabled, roles e criar os **gettes e setters**
- Atualizar o construtor User(UserDTO userDTO) da classe User com o atributo **password**
- e criar o método **public User registerUser(User user)** no serviço **UserService**
- Criar a o método **createVerificationTokenForUser(User user, String token**) no serviço **UserService**
- Atualizar o método **create(User user)** com encoder password no serviço **UserService**

```
user.setPassword(passwordEncoder.encode(user.getPassword()));
```

Endpoint para confirmar registro de usuários

Checklist:

Na classe **RegistrationResource** do pacote **resources** implementar o seguinte endpoint

```
@GetMapping("/public/regitrationConfirm/users")
public ResponseEntity<String> confirmRegistrationUser(@RequestParam("token")
String token){}
```

- Antes de implementar o endpoint acima, criar o método **public String validateVerificationToken(String token)** no serviço **UserService**

Endpoint para retornar usuário logado

Checklist:

Na classe **UserResource** do pacote **resources** implementar o seguinte endpoint

```
@GetMapping(value="/users/main")
public ResponseEntity<UserDTO> getUserMain(Principal principal){}
```

- Antes de implementar o endpoint acima, criar o método **public User findByEmail(String email)** no serviço **UserService**

Endpoint para fazer logout

Checklist:

Na classe **UserResource** do pacote **resources** implementar o seguinte endpoint

```
@GetMapping(value = "/logout")
public ResponseEntity<Void>  logout(HttpServletRequest request) {}
```

Email HTML

Checklist:

- Em **pom.xml**, incluir a dependência do **Thymeleaf**:

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

- Criar o template **Thymeleaf** para o email (código no final deste documento).
  Criar o arquivo em: **resources/templates/email/registerUser.html**

Implementando Padrões Strategy e Template Method

Checklist:

- Adicionar a dependência no **pom.xml**

```
<dependency>
   <groupId>org.springframework.boot</groupId>
```

```
   <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

- Criar o **remetente** e **destinatário** default no **application.properties**

```
default.sender=hugosilva.cursos@gmail.com
default.recipient=hugosilva.cursos@gmail.com
default.url=http://localhost:8080
```

- Criar o **subpacote email** em **services** e criar a interface **EmailService (padrão Strategy)**

```
void sendHtmlEmail(MimeMessage msg);
void sendConfirmationHtmlEmail(User user, VerificationToken vToken);
```

- Criar a classe abstrata **AbstractEmailService** implements **EmailService** no subpacote **email**

```
@Override
public void sendConfirmationHtmlEmail(User user, VerificationToken vToken){}
protected  MimeMessage prepareMimeMessageFromUser(User user, VerificationToken
vToken) {}
protected String htmlFromTemplateUser(User user, VerificationToken vToken){}
```

- Em **UserService**, criar a chamada do método **sendConfirmationHtmlEmail** no método **public User registerUser(User user)**

Implementando SmtpEmailService com servidor do Google

Checklist:

Acrescentar os seguintes dados em **application.properties**:

```
spring.mail.properties.mail.smtp.ssl.trust = smtp.gmail.com
spring.mail.host=smtp.gmail.com
spring.mail.username=
spring.mail.password=
spring.mail.properties.mail.smtp.auth = true
spring.mail.properties.mail.smtp.socketFactory.port = 465
spring.mail.properties.mail.smtp.socketFactory.class =
javax.net.ssl.SSLSocketFactory
spring.mail.properties.mail.smtp.socketFactory.fallback = false
spring.mail.properties.mail.smtp.starttls.enable = true
spring.mail.properties.mail.smtp.ssl.enable = true
```

- No subpacote **email** implementar o serviço **SmtpEmailService** que deve estender do serviço **AbstractEmailService**
- No pacote **config** criar a classe **EmailConfig**, criar um método **@Bean EmailService** que retorna uma instância de **SmtpEmailService**

```
@Bean
public EmailService emailService()
```

Endpoint para enviar novo token quando o mesmo expirar

Checklist:

Na classe **VerificationToken** do pacote **domain** implementar o seguinte método

```
public void updateToken(final String token) {
    this.token = token;
    this.expiryDate = calculateExpiryDate(EXPIRATION);
}
```

- Em seguida implementar o método **generateNewVerificationToken** no serviço **UserService** do pacote **services**

```
public void  generateNewVerificationToken(String email) {}
```

- Agora implementaremos o endpoint **resendRegistrationToken** em **RegistrationResource**

```
@GetMapping(value = "/resendRegistrationToken/users")
public ResponseEntity<Void>  resendRegistrationToken(
@RequestParam("token")  String email) {}
```

- Atualizar a classe SetupDataLoader com instância **VerificationTokenRepository**

```
verificationTokenRepository.deleteAll();
```

Configuração de Erros e ajuste do código fonte da API REST

Checklist:

- Atualizar a classe **WebSecurityConfiguration** no pacote **security** com o método abaixo:

```
@Override
public void configure(WebSecurity web) throws Exception {
    web.ignoring().antMatchers(HttpMethod.OPTIONS, "/**")
            .antMatchers( "/api/public/**")
            .antMatchers("/api/logout/**");
}
```

- Criar a classe **ObjectNotEnabledException** no pacote services.exception
- Atualizar o método **loadUserByUsername** da classe **CustomUserDetailsService** com a classe **ObjectNotEnabledException**

```
throw new ObjectNotEnabledException(String.format("UserNotEnabled"));
```

- Atualizar a classe **RestResponseEntityExceptionHandler** do pacote resources.exception com os seguintes métodos:

```
//error 409
@ExceptionHandler({ ObjectAlreadyExistException.class })
public ResponseEntity<Object> handleObjectAlreadyExist(final
RuntimeException e,  HttpServletRequest request) {
    HttpStatus status = HttpStatus.CONFLICT;
    StandardError err = new StandardError(System.currentTimeMillis(),
status.value(), "UserAlreadyExist", e.getMessage(),
request.getRequestURI());
    return ResponseEntity.status(status).body(err);
}
```

```
//error 401
@ExceptionHandler(value = {ObjectNotEnabledException.class})
public ResponseEntity<Object> handleObjectNotEnabled(final
RuntimeException e,  HttpServletRequest request) {
    HttpStatus status = HttpStatus.UNAUTHORIZED;
    StandardError err = new StandardError(System.currentTimeMillis(),
status.value(), "UserNotEnable", e.getMessage(), request.getRequestURI());
    return ResponseEntity.status(status).body(err);
}
```

- Modificar os **endpoints** da classe **RegistrationResource**:

```
@RequestMapping("/api/public")
```

```
@PostMapping("/registration/users")
```

```
@GetMapping("/regitrationConfirm/users")
```

```
@GetMapping(value = "/resendRegistrationToken/users")
```

- Atualizar o método **htmlFromTemplateUser** da classe **AbstractEmailService** com a seguinte informação:

```
String confirmationUrl = this.contextPath +
"/api/public/regitrationConfirm/users?token="+token;
```

Testando Endpoint's Registro de Usuário

Checklist:

- Testar os endpoint's do tipo POST:
  - http://localhost:8080/api/public/registration/users

```
{
   "firstName": " Hugo ",
   "lastName": " Silva ",
   "email": "hugosilva.cursos@gmail.com",
  "password": "123"
}
```

- Na primeira tentativa de envio de email você vai receber um erro porque o Google por padrão bloqueia tentativa de email por app:

```
javax.mail.AuthenticationFailedException: 535-5.7.8 Username and Password not accepted. Learn more at
535 5.7.8  https://support.google.com/mail/?p=BadCredentials i38sm4790327qtc.57 - gsmtp

        at com.sun.mail.smtp.SMTPTransport$Authenticator.authenticate(SMTPTransport.java:965) ~[javax.mail-
1.6.2.jar:1.6.2]
        at com.sun.mail.smtp.SMTPTransport.authenticate(SMTPTransport.java:876) ~[javax.mail-1.6.2.jar:1.6.2]
        at com.sun.mail.smtp.SMTPTransport.protocolConnect(SMTPTransport.java:780) ~[javax.mail-
1.6.2.jar:1.6.2]
        at javax.mail.Service.connect(Service.java:366) ~[javax.mail-1.6.2.jar:1.6.2]
                                                        (...)
```

- Verifique seu email do Google. Haverá a seguinte mensagem:

- Clique em "*permitindo o acesso a apps menos seguros*" e habilite o envio de emails:

Libere o acesso ao app por meio de dois links:

https://www.google.com/settings/security/lesssecureapps

https://accounts.google.com/b/0/DisplayUnlockCaptcha

- Testar os endpoint's do tipo GET e POST:
    - http://localhost:8080/api/public/resendRegistrationToken/users?email=hugosilva.cursos@gmail.com
    - http://localhost:8080/oauth/token?grant_type=password&username=hugosilva.cursos@gmail.com&password=123
    - http://localhost:8080/api/public/regitrationConfirm/users?token= + {TOKEN}
    - http://localhost:8080/api/users e Bearer + Acces_Token
    - http://localhost:8080/api/users/main e Bearer + Acces_Token
    - http://localhost:8080/api/logout e Bearer + Acces_Token

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title th:remove="all">Confirmar Registro de Usuário no Curso </title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/> </head>
<body>
  <div>
      <h3>
          Caso você tenha solicitado acesso ao Curso - Spring Boot & Angular 7
  </h3> <br />

    <p>
     Clique no link abaixo para confirmar a sua solicitação <br />
    <span th:text="${confirmationUrl}"> </span>
    </p>
    <p>
      Nome:
      <span th:text="${user.firstName}"></span>  <span
    th:text="${user.lastName}" ></span>
    </p>
  </div>
</body>
</html>
```