# User-service documentation

1. Пишем регистрацию пользователя.
   1. Создаем entity Users
   2. В таблицу пользователь добавляется токен для того, что бы проверить подтвержден ли e-mail
2. public String generateEmailVerificationToken(String userId) {  
    String token = Jwts.*builder*()  
    .setSubject(userId)  
   // time of email verification = 864000000; // 10 days  
    .setExpiration(new Date(System.*currentTimeMillis*() + SecurityConstants.*EXPIRATION\_TIME*))  
    .signWith(SignatureAlgorithm.*HS512*, SecurityConstants.*getTokenSecret*())  
    .compact();  
    return token;  
    }

для того, что бы получить SecurityConstants.*getTokenSecret*())

в public class SecurityConstants {

создается метод

public static String getTokenSecret()  
{  
 AppProperties appProperties = (AppProperties) SpringApplicationContext.*getBean*("AppProperties");  
 return appProperties.getTokenSecret();  
}

для которого необходимо создать

@Component  
public class AppProperties {  
  
 @Autowired  
 private Environment env;  
  
 public String getTokenSecret()  
 {  
 return env.getProperty("tokenSecret");  
 }  
}

Который получает environment свойство tokenSecret (из application.properties)

@SpringBootApplication  
public class UserServiceApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(UserServiceApplication.class, args);  
 }  
  
 @Bean  
 public BCryptPasswordEncoder bCryptPasswordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public SpringApplicationContext springApplicationContext() {  
 return new SpringApplicationContext();  
 }  
  
 @Bean(name = "AppProperties")  
 public AppProperties getAppProperties() {  
 return new AppProperties();  
 }  
}

+

*/\*\*  
 \* благодаря этому классу у UsersServiceImpl есть доступ к CONTEXT  
 \*/*public class SpringApplicationContext implements ApplicationContextAware {  
 private static ApplicationContext *CONTEXT*;  
  
 @Override  
 public void setApplicationContext(ApplicationContext context) throws BeansException {  
 *CONTEXT* = context;  
 }  
  
 public static Object getBean(String beanName) {  
 return *CONTEXT*.getBean(beanName);  
 }  
}

Эти классы необходимы для получения ApplicationContext и всех свойств

1. Меняем security так, что бы при неподтвержденном e-mail токен не создавался

Меняем метод ы классе UserServiceImpl

@Override

public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {

UserEntity userEntity = userRepository.findByEmail(email);

if (userEntity == null)

throw new UsernameNotFoundException(email);

/\*\*

\* если userEntity.getEmailVerificationStatus() == false -> login вернет error

\*/

return new User(userEntity.getEmail(), userEntity.getEncryptedPassword(), userEntity.getEmailVerificationStatus(),

true, true, true, new ArrayList<>());

}

# Пишем сброс пароля (пользователь забыл свой пароль)

Создаем метод, на который будет приходить запрос на изменение пароля

@PostMapping(path = "/password-reset-request")  
public OperationStatusModel requestReset(@RequestBody PasswordResetRequestModel passwordResetRequestModel) {  
 OperationStatusModel returnValue = new OperationStatusModel();  
  
 boolean operationResult = userService.requestPasswordReset(passwordResetRequestModel.getEmail());  
  
 returnValue.setOperationName(RequestOperationName.REQUEST\_PASSWORD\_RESET.name());  
 returnValue.setOperationResult(RequestOperationStatus.*ERROR*.name());  
  
 if (operationResult) {  
 returnValue.setOperationResult(RequestOperationStatus.*SUCCESS*.name());  
 }  
  
 return returnValue;  
}

Этот метод должен отправить на e-mail пользователя новый токен для изменения пароля.

Метод получает обьект

PasswordResetRequestModel

(то есть e-nail пользователя)

Метод requestPasswordReset в свою очередь генерирует токен (String token = new UserUtils().generatePasswordResetToken(userEntity.getUuidUser());)

Тело метода:

@Override  
 public boolean requestPasswordReset(String email) {  
  
 boolean returnValue = false;  
  
 Optional<Users> optionalUser = userRepo.findByMainEmail(email);  
  
 if (!optionalUser.isPresent()) {  
 return returnValue;  
 }  
 Users userEntity = optionalUser.get();  
 String token = new UserUtils().generatePasswordResetToken(userEntity.getUuidUser());  
  
 PasswordResetTokenEntity passwordResetTokenEntity = new PasswordResetTokenEntity();  
 passwordResetTokenEntity.setToken(token);  
 passwordResetTokenEntity.setUserDetails(userEntity);  
 passwordResetTokenRepository.save(passwordResetTokenEntity);  
// отправляем письмо. Если письмоотправлено - вернется True  
 EmailVerificationDto emailDto = EmailVerificationDto.*builder*()  
 .email(userEntity.getMainEmail())  
 .tokenValue(token)  
 .userLastName(userEntity.getSecondName())  
 .userName(userEntity.getFirstName())  
 .build();  
// method - send email  
 rabbitTemplate.convertAndSend(emailPasswordExchange, emailPasswordRoutingkey, emailDto);  
 *LOGGER*.info("new token for changing password was send to email " + emailDto.getEmail());  
 return returnValue;  
 }

Теперь метод отправляем запрос в emailService что бы при запросе полтзователя о смене фамилии ему было отправлено письмо с новым токеном (срок дейтсвия токена – 1 сутки)

Позволяем менять пароль без авторизации-

/\*\*изменение пароля - доступно всем\*\*/

.antMatchers(HttpMethod.POST, SecurityConstants.PASSWORD\_RESET\_REQUEST\_URL)

.permitAll()

Для того, что бы отправить письмо в очередь – необходимо создать эту очередь.

## User-service

* 1. Application.properties add

email.password.key=confpasswemail  
email.password.queue=CONFPPASSEMAIL  
email.password.exchange=Email

* 1. RabbitConfig add

@Value(("${email.password.queue}"))  
String changePasswordQueue;  
@Value(("${email.password.key}"))  
String changePasswordKey;  
  
@Bean  
public void crrateChangePasswordQueue() {  
 ConnectionFactory factory = new ConnectionFactory();  
 factory.setPassword(rabbitPassword);  
 factory.setUsername(rabbitUserName);  
 try (Connection connection = factory.newConnection();  
 Channel channel = connection.createChannel()) {  
 channel.queueDeclare(changePasswordQueue, false, false, false, null);  
 channel.exchangeDeclare(emailExchange, *type*, true);  
 channel.queueBind(changePasswordQueue, emailExchange, changePasswordKey);  
 } catch (TimeoutException | IOException e) {  
 e.printStackTrace();  
 }  
}

* 1. Теперь наше письмо отправляется. Нужно настроить email-service

Пишем логику email-service для отправки письма

## Email-service

1. Application.properties add

email.password.key=confpasswemail  
email.password.queue=CONFPPASSEMAIL  
email.password.exchange=Email

1. RabbitConfig

@Value(("${email.password.queue}"))  
String changePasswordQueue;  
@Value(("${email.password.key}"))  
String changePasswordKey;

@Bean  
public void crrateChangePasswordQueue() {  
 ConnectionFactory factory = new ConnectionFactory();  
 factory.setPassword(rabbitPassword);  
 factory.setUsername(rabbitUserName);  
 try (Connection connection = factory.newConnection();  
 Channel channel = connection.createChannel()) {  
 channel.queueDeclare(changePasswordQueue, false, false, false, null);  
 channel.exchangeDeclare(emailExchange, *type*, true);  
 channel.queueBind(changePasswordQueue, emailExchange, changePasswordKey);  
 } catch (TimeoutException | IOException e) {  
 e.printStackTrace();  
 }  
}

1. RabbitServiceImpl

@RabbitListener(queues = "${email.password.queue}")  
public void recivedChangePasswordMessage(EmailVerificationDto incomingMessage) {  
 emailService.changePasswordEmail(incomingMessage);  
}

1. EmailServiceImpl

@Override  
public String changePasswordEmail(EmailVerificationDto mail) {  
 MimeMessage msg = emailSender.createMimeMessage();  
  
 String email = mail.getEmail();  
 String tokenValue = mail.getTokenValue();  
 String lastName = mail.getUserLastName();  
 String firstName = mail.getUserName();  
  
 String VERIFY\_LINK = env.getProperty("change.password.verification.link");  
 MimeMessageHelper helper = null;  
 try {  
 helper = new MimeMessageHelper(msg, true);  
 } catch (MessagingException e) {  
 e.printStackTrace();  
 }  
 //// The HTML body for the email.  
 final String htmlMsg =  
 "<h1> Hi " + firstName + " " + lastName + ". Please verify your email address</h1>"  
 + "<p>For changing your password you click to link: "  
 + " <a href='" + VERIFY\_LINK + "?token=" + tokenValue + "'>" + "<br/><br/>" + " Change password" +  
 "</a><br/><br/>"  
 + "If you didn`t send responce for changing password - please ignore this message";  
 try {  
 helper.setTo(email);  
 helper.setTo(email);  
 helper.addAttachment("logo.png", new ClassPathResource("logo.png"));  
 helper.setSubject("Change password email from" + *APPLICATION\_NAME*);  
 helper.setText(htmlMsg, true);  
 helper.addAttachment("logo.png", new ClassPathResource("logo.png"));  
 emailSender.send(msg);  
 *LOGGER*.info(" change password email was send to e-mail: " + email);  
 } catch (MessagingException ex) {  
 *LOGGER*.error("change password email: something is wrong" + ex);  
 emailError(adminEmail, *VERIFICATION\_SERVICE*, lastName, firstName);  
 *LOGGER*.error("send error-email to administrator" + mail);  
 System.*out*.println(ex.getMessage());  
 }  
 return "Email send!";  
}

2 шаг – когда пользователь получает письмо – он должен нажать на ссылку и перейти по адресу

/password-reset

1. Метод в контроллере

@PostMapping(path = "/password-reset")  
public OperationStatusModel resetPassword(@RequestBody PasswordResetModel passwordResetModel) {  
 OperationStatusModel returnValue = new OperationStatusModel();  
  
 boolean operationResult = userService.resetPassword(  
 passwordResetModel.getToken(),  
 passwordResetModel.getPassword());  
  
 returnValue.setOperationName(RequestOperationName.*PASSWORD\_RESET*.name());  
 returnValue.setOperationResult(RequestOperationStatus.*ERROR*.name());  
  
 if (operationResult) {  
 returnValue.setOperationResult(RequestOperationStatus.*SUCCESS*.name());  
 }  
  
 return returnValue;  
}

1. Этот метод вызывает метод из UserServiceImpl

*/\*\* запрос на обноелние пароля шаг2\*\*/*@Override  
public boolean resetPassword(String token, String password) {  
 boolean returnValue = false;  
  
 if( UserUtils.*hasTokenExpired*(token) )  
 {  
 return returnValue;  
 }  
 PasswordResetTokenEntity passwordResetTokenEntity = passwordResetTokenRepository.findByToken(token);  
 if (passwordResetTokenEntity == null) {  
 return returnValue;  
 }  
 // Prepare new password  
 String encodedPassword = bCryptPasswordEncoder.encode(password);  
 // Update User password in database  
 Users userEntity = passwordResetTokenEntity.getUserDetails();  
 userEntity.setPassword(encodedPassword);  
 Users savedUserEntity = userRepo.save(userEntity);  
 // Verify if password was saved successfully  
 if (savedUserEntity != null && savedUserEntity.getPassword().equalsIgnoreCase(encodedPassword)) {  
 returnValue = true;  
 }  
 // Remove Password Reset token from database  
 passwordResetTokenRepository.delete(passwordResetTokenEntity);  
 return returnValue;  
}

1. Этот метод получает токен из хедеров, проверяет его, и если токен правильный – перезаписывает пароль в базе данных. После этого пользователь может логинится под новым паролем!