

Registration and Login 2

Email Verification



Registration and Login 2










Email Verification



Demo

<http://localhost:8080/registration>

Database Visualization

verification_token	
 id	bigint(20)
 confirmed_date_time	datetime
 expired_date_time	datetime
 issued_date_time	datetime
 status	varchar(255)
 token	varchar(255)
 user_id	bigint(20)

users	
 id	bigint(20)
 created_at	datetime
 email	varchar(50)
 first_name	varchar(50)
 is_active	bit(1)
 last_name	varchar(50)
 password	varchar(255)
 updated_at	datetime
 username	varchar(50)

roles	
 id	bigint(20)
 name	varchar(255)

users_roles	
 users_id	bigint(20)
 roles_id	bigint(20)

Dependencies

```
<dependency>  
  <groupId>com.sun.mail</groupId>  
  <artifactId>javax.mail</artifactId>  
  <version>1.6.0</version>  
</dependency>
```

Verification Token Model

```
@Entity
public class VerificationToken {
    public static final String STATUS_PENDING = "PENDING";
    public static final String STATUS_VERIFIED = "VERIFIED";

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String token;
    private String status;
    private LocalDateTime expiredDateTime;
    private LocalDateTime issuedDateTime;
    private LocalDateTime confirmedDateTime;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id")
    private User user;

    public VerificationToken() {
        this.token = UUID.randomUUID().toString();
        this.issuedDateTime = LocalDateTime.now();
        this.expiredDateTime = this.issuedDateTime.plusDays(1);
        this.status = STATUS_PENDING;
    }
}
```

User Model Verification Token Mapping

```
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "users")
public class User {

    private Boolean isActive;

    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL)
    private VerificationToken verificationToken;
}
```

Verification Token Repository

```
package com.nursultanturdaliev.moneytransferapp.repository;  
  
import com.nursultanturdaliev.moneytransferapp.model.VerificationToken;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface VerificationTokenRepository extends JpaRepository<VerificationToken, String> {  
    VerificationToken findOneByUserEmail(String email);  
  
    VerificationToken findOneByToken(String token);  
}
```


Sending Mail Service

```
@Service
public class SendingMailService {
    private final MailProperties mailProperties;

    private SpringTemplateEngine springTemplateEngine;

    private org.slf4j.Logger logger = LoggerFactory.getLogger(HomeController.class);

    @Autowired
    SendingMailService(MailProperties mailProperties, SpringTemplateEngine springTemplateEngine) {
        this.mailProperties = mailProperties;
        this.springTemplateEngine = springTemplateEngine;
    }

    void sendVerificationMail(User user, String verificationCode) {
        String subject = "Please verify your email";
        String body = "";
        try {
            Context context = new Context();
            context.setVariable("verificationURL", mailProperties.getVerificationapi() + verificationCode);
            context.setVariable("user", user);
            body = springTemplateEngine.process("email-verification.html", context);
        } catch (Exception ex) {
            logger.error(ex.getMessage(), ex);
        }

        sendMail(user.getEmail(), subject, body);
    }
}
```

Sending Mail Service

```
private boolean sendMail(String toEmail, String subject, String body) {
    try {
        Properties props = System.getProperties();
        props.put("mail.transport.protocol", "smtp");
        props.put("mail.smtp.port", mailProperties.getSmtp().getPort());
        props.put("mail.smtp.starttls.enable", "true");
        props.put("mail.smtp.auth", "true");

        Session session = Session.getDefaultInstance(props);
        session.setDebug(true);

        MimeMessage msg = new MimeMessage(session);
        msg.setFrom(new InternetAddress(mailProperties.getFrom(),
        mailProperties.getFromName()));
        msg.setRecipient(Message.RecipientType.TO, new InternetAddress(toEmail));
        msg.setSubject(subject);
        msg.setContent(body, "text/html");

        Transport transport = session.getTransport();
        transport.connect(mailProperties.getSmtp().getHost(),
        mailProperties.getSmtp().getUsername(), mailProperties.getSmtp().getPassword());
        transport.sendMessage(msg, msg.getAllRecipients());
        return true;
    } catch (Exception ex) {
        logger.error(ex.getMessage(), ex);
    }

    return false;
}
```

Verification Token Service

```
@Service
public class VerificationTokenService {

    private VerificationTokenRepository verificationTokenRepository;
    private SendingMailService sendingMailService;

    @Autowired
    public VerificationTokenService(VerificationTokenRepository
verificationTokenRepository, SendingMailService sendingMailService) {
        this.verificationTokenRepository = verificationTokenRepository;
        this.sendingMailService = sendingMailService;
    }

    public void createVerification(User user) {

        VerificationToken verificationToken =
verificationTokenRepository.findOneByUserEmail(user.getEmail());
        if (verificationToken == null) {
            verificationToken = new VerificationToken();
            verificationToken.setUser(user);
            verificationTokenRepository.save(verificationToken);
        }

        sendingMailService.sendVerificationMail(user, verificationToken.getToken());
    }
}
```

Verification Token Service

```
public void verifyEmail(String token) throws InvalidTokenException,
ExpiredTokenException {
    VerificationToken verificationToken =
verificationTokenRepository.findOneByToken(token);
    if (verificationToken == null) {
        throw new InvalidTokenException();
    }

    if (verificationToken.getExpiredDateTime().isBefore(LocalDate.now()))
    {
        throw new ExpiredTokenException();
    }

    verificationToken.setConfirmedDateTime(LocalDate.now());
    verificationToken.setStatus(VerificationToken.STATUS_VERIFIED);
    verificationToken.getUser().setActive(true);
    verificationTokenRepository.save(verificationToken);
}
```

Mail Properties

```
@Component
@ConfigurationProperties(prefix = "mail")
public class MailProperties {
    public static class SMTP {
        String host;
        String port;
        String username;
        String password;
    }

    private SMTP smtp;
    private String from;
    private String fromName;
    private String verificationapi;
}
```

Gmail SMTP Setup

Incoming Mail (IMAP) Server	imap.gmail.com Requires SSL: Yes Port: 993
Outgoing Mail (SMTP) Server	smtp.gmail.com Requires SSL: Yes Requires TLS: Yes (if available) Requires Authentication: Yes Port for SSL: 465 Port for TLS/STARTTLS: 587
Full Name or Display Name	Your name
Account Name, User name, or Email address	Your full email address
Password	Your Gmail password

Mail Properties

application.properties

```
mail.smtp.host=smtp.gmail.com  
mail.smtp.port=587  
mail.smtp.username=email@gmail.com  
mail.smtp.password=ThisIsMyPassword!  
mail.from=email@gmail.com  
mail.from-name=Nursultan Turdaliev
```

Setup Google Account

References

1. <https://www.baeldung.com/get-user-in-spring-security>
2. <https://hellokoding.com/email-verification-example-with-spring-boot-mysql-doc-ker-compose/>
- 3.