**SWER007**

**Lab 8: Email Notifications & Asynchronous Processing in Spring Boot**

**📌 Objective**

In this lab, students will learn how to **send emails** from a Spring Boot application and handle **background email processing asynchronously** using @Async. Additionally, they will use **Thymeleaf templates** to format email content dynamically.

**Step 1: Add Dependencies**

Add the following dependencies in pom.xml to enable **email sending and Thymeleaf templates**:

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

✅ **Why?**

* spring-boot-starter-mail enables **email sending** in Spring Boot.

**Step 2: Configure Email Properties**

Modify application.properties or application.yml to configure **SMTP settings** (for Gmail in this case):

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=your-email@gmail.com

spring.mail.password=your-app-password

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

✅ **Why?** This configures Spring Boot to send emails using Gmail SMTP.

🔹 **For Gmail users:** If you have 2-step verification, you must generate an **App Password** [here](https://myaccount.google.com/apppasswords).

**Step 3: Enable Asynchronous Processing**

Add @EnableAsync in your main application class to **support background tasks**.

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.scheduling.annotation.EnableAsync;

@SpringBootApplication

@EnableAsync

public class EmailApplication {

public static void main(String[] args) {

SpringApplication.run(EmailApplication.class, args);

}

}

✅ **Why?** @EnableAsync allows running email tasks **in the background** without blocking the main thread.

**Step 4: Implement Email Sending Service**

Create an EmailService that sends **welcome emails** and **admin notifications**.

import jakarta.mail.MessagingException;

import jakarta.mail.internet.MimeMessage;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.mail.javamail.MimeMessageHelper;

import org.springframework.scheduling.annotation.Async;

import org.springframework.stereotype.Service;

import org.thymeleaf.TemplateEngine;

import org.thymeleaf.context.Context;

@Service

public class EmailService {

private final JavaMailSender mailSender;

private final TemplateEngine templateEngine;

public EmailService(JavaMailSender mailSender, TemplateEngine templateEngine) {

this.mailSender = mailSender;

this.templateEngine = templateEngine;

}

@Async

public void sendWelcomeEmail(String to, String username) throws MessagingException {

Context context = new Context();

context.setVariable("username", username);

String htmlContent = templateEngine.process("welcome-email", context);

sendEmail(to, "Welcome to Our Platform!", htmlContent);

}

@Async

public void sendAdminNotification(String to, String message) throws MessagingException {

Context context = new Context();

context.setVariable("message", message);

String htmlContent = templateEngine.process("admin-notification", context);

sendEmail(to, "New User Registered", htmlContent);

}

private void sendEmail(String to, String subject, String htmlContent) throws MessagingException {

MimeMessage message = mailSender.createMimeMessage();

MimeMessageHelper helper = new MimeMessageHelper(message, true);

helper.setTo(to);

helper.setSubject(subject);

helper.setText(htmlContent, true);

mailSender.send(message);

}

}

✅ **Key Features:**

* **@Async** → Emails are sent **asynchronously** (non-blocking).
* **Thymeleaf Templates** → Emails are formatted using HTML templates.

**Step 5: Create Thymeleaf Email Templates**

**Welcome Email Template (src/main/resources/templates/welcome-email.html)**

<!DOCTYPE html>

<html>

<head>

<title>Welcome</title>

</head>

<body>

<h1>Welcome, <span th:text="${username}"></span>!</h1>

<p>Thank you for registering. We’re excited to have you onboard!</p>

</body>

</html>

**Admin Notification Template (src/main/resources/templates/admin-notification.html)**

<!DOCTYPE html>

<html>

<head>

<title>Admin Notification</title>

</head>

<body>

<h1>New User Registered</h1>

<p><span th:text="${message}"></span></p>

</body>

</html>

**✅ Why? These templates allow emails to be personalized and formatted dynamically.**

**Step 6: Modify User Registration to Trigger Emails**

Modify UserController to send a **welcome email** upon registration and notify the admin.

import jakarta.mail.MessagingException;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.HashMap;

import java.util.Map;

@RestController

@RequestMapping("/users")

public class UserController {

private final EmailService emailService;

private final Map<Long, String> users = new HashMap<>();

public UserController(EmailService emailService) {

this.emailService = emailService;

}

@PostMapping

public ResponseEntity<String> registerUser(@RequestBody UserRequest request) throws MessagingException {

long id = users.size() + 1;

users.put(id, request.getUsername());

// Send welcome email

emailService.sendWelcomeEmail(request.getEmail(), request.getUsername());

// Notify admin

emailService.sendAdminNotification("admin@example.com", "New user registered: " + request.getUsername());

return ResponseEntity.status(HttpStatus.CREATED).body("User registered successfully.");

}

}

class UserRequest {

private String username;

private String email;

public String getUsername() { return username; }

public String getEmail() { return email; }

}

✅ **What This Does:**

* Sends **a welcome email** when a new user registers.
* Notifies **the admin** about new user sign-ups.

**📝 Lab Exercises**

**Exercise 1: Add More Email Types**

* Implement an email for **password reset requests**.

**Exercise 2: Log Email Sending**

* Modify EmailService to **log when an email is sent successfully**.

**Exercise 3: Schedule Weekly Admin Reports**

* Use @Scheduled to **send weekly admin reports** about new users.