MỤC TIÊU:

Kết thúc bài thực hành này bạn có khả năng

- ✓ Gửi email với Spring Boot
- ✓ Sử dụng được dịch vụ nền Schedule Task để gửi email
- ✓ Sử dụng được Interceptor để chia sẻ dữ liệu và bảo vệ ứng dụng

PHẦN I

Bài 1 (2 điểm)

Hãy xây dựng Spring Bean class cho phép gửi email theo đặt tả của interface MailService sau đây.

```
public interface MailService{
        @Data
        @Builder
        public static class Mail{
                @Default
                private String from = "WebShop <web-shop@gmail.com>";
                private String to, cc, bcc, subject, body, filenames;
        }

        void send(Mail mail);
        default void send(String to, String subject, String body) {
                Mail mail =
Mail.builder().to(to).subject(subject).body(body).build();
                this.send(mail);
        }
}
```

Hướng dẫn:

Bước 1: Khai báo thư viện cần thiết (pom.xml)

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

```
</dependency>
```

Bước 2: Khai báo thông số kết nối GMail Smtp Server (application.properties)

```
#JavaMailSender - https://myaccount.google.com/apppasswords
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=user@gmail.com
spring.mail.password=***
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.ssl.protocols=TLSv1.2
```

Bước 3: Cài đặt mã nguồn cho interface MailService qua lớp MailServiceImpl

```
@Service("mailService")
public class MailServiceImpl implements MailService{
	@Autowired
	JavaMailSender mailSender;

	public void send(Mail mail) {
		try {
			// 1. Tạo Mail
			MimeMessage message = mailSender.createMimeMessage();
			// 2. Tạo đối tượng hỗ trợ ghi nội dung Mail
			MimeMessageHelper helper =
				new MimeMessageHelper(message, true, "utf-8");
			// 2.1. Ghi thông tin người gửi
			// 2.2. Ghi thông tin người nhận
			// 2.3. Ghi tiêu đề và nội dung
			// 2.4. Đính kèm file
			//3. Gửi Mail
			mailSender.send(message);
		} catch (Exception e) {
			throw new RuntimeException(e);
		}
	}

	private boolean isNullOrEmpty(String text) {
		return (text == null || text.trim().length() == 0);
```

```
        }
}
```

Bước 4: Bổ mã cho mục 2.1 đến 2.4 để hoàn thiện ghi nội dung Mail

```
// 2.1. Ghi thông tin người gửi
helper.setFrom(mail.getFrom());
helper.setReplyTo(mail.getFrom());
// 2.2. Ghi thông tin người nhận
helper.setTo(mail.getTo());
if(!this.isNullOrEmpty(mail.getCc())) {
        helper.setCc(mail.getCc());
}
if(!this.isNullOrEmpty(mail.getBcc())) {
        helper.setBcc(mail.getBcc());
}
// 2.3. Ghi tiêu đề và nội dung
helper.setSubject(mail.getSubject());
helper.setText(mail.getBody(), true);
// 2.4. Đính kèm file
String filenames = mail.getFilenames();
if(!this.isNullOrEmpty(filenames)) {
        for(String filename: filenames.split("[,;]+")) {
                File file = new File(filename.trim());
                helper.addAttachment(file.getName(), file);
        }
}
```

Bước 5: Tạo MailController và sử dụng MailService để thực hiện gửi email đơn giản.

```
@Autowired
MailService mailService;

@ResponseBody
@RequestMapping("/mail/send")
public String send() {
        try {
                mailService.send("receiver@gmail.com", "Subject", "Body");
                return "Mail của bạn đã được gửi đi";
        } catch (MessagingException e) {
```

```
            return e.getMessage();
        }
}
```

Bài 2 (2 điểm)

Nâng cấp MailService để thay vì gửi mail trực tiếp thì xếp Mail vào hàng đợi và sử dụng @Scheduled để tạo hoạt động gửi email từ phía hậu trường tránh trường hợp tắc nghẽn khi nhiều người gửi email đồng thời.

Hướng dẫn:

Hãy thực hiện nâng cấp MailerService theo hướng dẫn sau đây

Bước 1: Chú thích file cấu hình chính của ứng dụng với @EnableScheduling

Bước 2: Khai báo bổ sung 2 phương thức push(Mail) và push(String, String, String) để xếp Mail vào hàng đợi

```
void push(Mail mail);
default void push(String to, String subject, String body){
        this.push(Mail.builder().to(to).subject(subject).body(body).build());
}
```

Bước 3: Cài đặt mã cho các phương thức trên trong MailServiceImpl để xếp Mail vào List<Mail> (hàng đợi)

```
List<Mail> queue = new ArrayList<>();
@Override
public void push(Mail mail){
        queue.add(mail);
}
```

Bước 4: Sử dụng @Scheduled để khai báo cho phương thức chạy nền thực hiện lấy Mail từ hàng đợi và gửi đi (sau mỗi 500 mi li giây sẽ kiểm tra và gửi)

```
@Scheduled(fixedDelay = 500)
public void run() {
        while(!queue.isEmpty()) {
                try {
                        this.send(queue.remove(0));
                } catch (Exception e) {
```

```
                    e.printStackTrace();
                }
            }
}
```

Bước 6: Hiệu chỉnh MailController để xếp mail vào hàng đợi thay vì gửi trực tiếp.

```
@Autowired
MailService mailService;

@ResponseBody
@RequestMapping("/mail/send")
public String send(Model model) {
        mailService.push("receiver@gmail.com", "Subject", "Body");
        return "Mail của bạn đã được xếp vào hàng đợi";
}
```

Bài 3 (1 điểm)

Xây dựng trang web cho phép người nhập đầy đủ thông tin (from, to, cc, bcc, subject, body, attachments) vào form gửi email, sử dụng MailService để gửi email theo 2 hình thức là gửi trực tiếp và xếp vào hàng đợi.

PHẦN II

Bài 4 (2 điểm)

Xây dựng trang web đăng nhập cho website

Hướng dẫn: bạn cần hoàn thành các thành phần sau đây

- Account entity
- AccountDAO
- AccountService, AccountServiceImpl
- AuthController
- login.html

Trong hướng dẫn này các bạn tự xây Entity, DAO và Service. Với login.html và AuthController thì thực hiện theo hướng dẫn sau đây:

Bước 1: Tạo trang login.html có chứa form sau

```
<i th:text="${message}">Error message</i>
<form action="/auth/login" method="post">
        <label>Username:</label>
        <input name="username">
        <br>
        <label>Password:</label>
        <input name="password" type="password">
        <hr>
        <button>Login</button>
</form>
```

Bước 2: Tạo AuthController

```
@Controller
public class AuthController {
        @Autowired
        AccountService accountService;
        @Autowired
        HttpSession session;

        @GetMapping("/auth/login")
        public String loginForm(Model model) {
                return "/auth/login";
        }

        @PostMapping("/auth/login")
        public String loginProcess(Model model,
                        @RequestParam("username") String username,
                        @RequestParam("password") String password) {
                // Bổ sung code đăng nhập
                return "/auth/login";
        }
}
```

Bước 3: Bổ sung code đăng nhập

```
Account user = accountService.findById(username);
if(user == null) {
```

```
        model.addAttribute("message", "Invalid email!");
} else if(!user.getPassword().equals(password)) {
        model.addAttribute("message", "Invalid password!");
} else{
        session.setAttribute("user", user);
        model.addAttribute("message", "Login successfully!");
}
```

Bài 5 (2 điểm)

Xây dựng **AuthInterceptor** và cấu hình để bảo mật ứng dụng theo yêu cầu sau:

- Cần phải **đăng nhập** mới được truy xuất các URI sau
  - ***/account/edit-profile***
  - ***/account/change-password***
  - ***/order/*****
- Phải đăng nhập với **vai trò admin** thì mới truy xuất được các url sau
  - ***/admin/*****
  - ***Ngoại trừ /admin/home/index***
- Nếu cố gắng truy xuất mà không thỏa mãn 2 điều kiện trên thì phải chuyển request về trang đăng nhập **/auth/login** và đưa ra thông báo phù hợp. Sau khi đăng nhập thành công thì ***quay trở lại url*** đã được bảo vệ trước đó (nếu có).

Hướng dẫn:

Bước 1: AuthInterceptor

```
@Component
public class AuthInterceptor implements HandlerInterceptor{
        @Autowired
        HttpSession session;
        @Override
        public boolean preHandle(HttpServletRequest request,
                HttpServletResponse response, Object handler) throws Exception {
                String uri = request.getRequestURI();
                session.setAttribute("securityUri", uri);
                Account user = (User) session.getAttribute("user");
                if(user == null) { // chưa đăng nhập
```

```
                response.sendRedirect("/auth/login");
                return false;
            }
            if(uri.startsWith("/admin") && !user.isAdmin()) { // không phải admin
                response.sendRedirect("/auth/login");
                return false;
            }
            return true;
        }
}
```

Bước 2: AuthConfig

```
@Configuration
public class InterceptorConfig implements WebMvcConfigurer {
    @Autowired
    AuthInterceptor authInterceptor;
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(authInterceptor)
            .addPathPatterns("/admin/**",
                "/account/change-password",
                "/account/edit-profile",
                "/order/**")
            .excludePathPatterns("/admin/home/index");
    }
}
```

Bước 3: Nâng cấp AuthController để quay trở lại URI bảo mật được truy cập trước đó nếu có.

Bổ sung đoạn mã sau vào cuối khối else của phương thức loginProcess() để hoàn thiện yêu cầu.

```
String securityUri = (String)session.getAttribute("securityUri");
if(securityUri != null) {
    return "redirect:" + securityUri;
}
```

Bài 6 (1 điểm)

Xây dựng và cấu hình LogInterceptor để ghi nhận địa chỉ URI, thời gian truy xuất và họ tên của người dùng khi họ truy cấp vào các trang được bảo mật (yêu cầu phải đăng nhập và giả sử user đăng nhập được lưu trong session)

Hướng dẫn:

Bước 1: Tạo LogInterceptor

```
@Component
public class LogInterceptor implements HandlerInterceptor{
	@Override
	public boolean preHandle(HttpServletRequest request,
		HttpServletResponse response, Object handler) throws Exception {
		Account user = request.getSession().getAttribute("user");
		System.out.println(request.getRequestURI()
			+ ", " + new Date()+ ", " + user.getFullname());
		return true;
	}
}
```

Bước 2: Cấu hình sử dụng LogInterceptor để lọc những URI được bảo mật.