Assessment Test: Java CRUD Operations with Logging, Error Handling, and Email Integration

Submitted by :Ritik_Saini

To develop a Java application with the specified functionalities, you'll need to follow a structured approach. Here's a detailed plan and code snippets for each part of the assignment:

1. CRUD Operations in Java

For the CRUD operations, we'll use Spring Boot, Spring Data JPA, and Hibernate for ORM.

Step 1: Set up Spring Boot Project

- Create a new Spring Boot project using Spring Initializer (https://start.spring.io/) with dependencies: Spring Web, Spring Data JPA, MySQL Driver, Thymeleaf (for JSP), and Spring Boot DevTools.

Step 2: Configure Database

- Configure your `application.properties` file for MySQL connection:

```
properties

spring.datasource.url=jdbc:mysql://localhost:3306/yourdatabase
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Step 3: Create Entity Classes

- Create an `Article` entity:

```
java

@Entity
public class Article {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    @Lob
    private String description;
    private LocalDate publishDate;
    private String status;
    private String banner;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private Author author;
```

```
    // Getters and Setters
}
```

- Create an `Author` entity:

```
java

@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    @OneToMany(mappedBy = "author")
    private List<Article> articles;

    // Getters and Setters
}
```

Step 4: Create Repositories

- Create `ArticleRepository` and `AuthorRepository`:

```
java

public interface ArticleRepository extends JpaRepository<Article,
Long> {}

public interface AuthorRepository extends JpaRepository<Author, Long>
{}
```

Step 5: Create Services

- Create `ArticleService` and `AuthorService` to handle business logic.

Step 6: Create Controllers

- Create `ArticleController`:

```
java

@Controller
public class ArticleController {
    @Autowired
    private ArticleService articleService;
    @Autowired
    private AuthorService authorService;

    @GetMapping("/articles")
    public String getAllArticles(Model model) {
        model.addAttribute("articles",
articleService.getAllArticles());
        return "list-articles";
    }

    @GetMapping("/article/new")
```

```
    public String showCreateForm(Model model) {
        model.addAttribute("article", new Article());
        model.addAttribute("authors", authorService.getAllAuthors());
        return "add-article";
    }

    @PostMapping("/article")
    public String saveArticle(@ModelAttribute("article") Article
article) {
        articleService.saveArticle(article);
        return "redirect:/articles";
    }

    @GetMapping("/article/edit/{id}")
    public String showEditForm(@PathVariable("id") Long id, Model
model) {
        model.addAttribute("article",
articleService.getArticleById(id));
        model.addAttribute("authors", authorService.getAllAuthors());
        return "edit-article";
    }

    @PostMapping("/article/update/{id}")
    public String updateArticle(@PathVariable("id") Long id,
@ModelAttribute("article") Article article) {
        articleService.updateArticle(id, article);
        return "redirect:/articles";
    }

    @GetMapping("/article/delete/{id}")
    public String deleteArticle(@PathVariable("id") Long id) {
        articleService.deleteArticle(id);
        return "redirect:/articles";
    }
}
```

2. Logging

- Use Log4j for logging. Add the dependency to your `pom.xml`:

```
xml

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

- Configure `log4j2.xml`:

```
xml

<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n"/>
        </Console>
        <File name="File" fileName="logs/app.log">
            <PatternLayout>
```

```
                <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -
%m%n</pattern>
            </PatternLayout>
        </File>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="File"/>
        </Root>
    </Loggers>
</Configuration>
```

- Log in your service classes:

```
java

private static final Logger logger =
LogManager.getLogger(ArticleService.class);

public void saveArticle(Article article) {
    logger.info("Saving article: " + article.getTitle());
    articleRepository.save(article);
}
```

3. Error Tracking and Management

- Use Spring's `@ControllerAdvice` for global exception handling:

```
java

@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception e) {
        logger.error("Exception: ", e);
        return new ResponseEntity<>("An error occurred: " +
e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

4. Integration of Email Service

- Use JavaMailSender for email integration. Add dependency:

```
xml

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

- Configure `application.properties` for email:

```
properties

spring.mail.host=smtp.example.com
```

```
spring.mail.port=587
spring.mail.username=your-email@example.com
spring.mail.password=your-password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

- Create EmailService:

```
java

@Service
public class EmailService {
    @Autowired
    private JavaMailSender mailSender;

    public void sendEmail(String to, String subject, String body) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject(subject);
        message.setText(body);
        mailSender.send(message);
    }
}
```

- Use EmailService in `ArticleService`:

```
java

@Autowired
private EmailService emailService;

public void saveArticle(Article article) {
    articleRepository.save(article);
    emailService.sendEmail("admin@example.com", "New Article Added",
"Article titled '" + article.getTitle() + "' has been added.");
}
```

5. Database Table Design

Ensure your database tables are set up with the correct types and relationships, using Hibernate annotations as shown in the entity classes above.

6. Technical Stack

Ensure you have the necessary dependencies and configurations as shown above.

7. User Interface

Create Thymeleaf templates for the UI:

- `list-articles.html`
- `add-article.html`
- `edit-article.html`

Example: list-articles.html

html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Articles</title>
</head>
<body>
    <h1>Articles</h1>
    <a href="/article/new">Add New Article</a>
    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Title</th>
                <th>Publish Date</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="article : ${articles}">
                <td th:text="${article.id}">1</td>
                <td th:text="${article.title}">Title</td>
                <td th:text="${article.publishDate}">2021-01-01</td>
                <td>
                    <a
th:href="@{/article/edit/{id}(id=${article.id})}">Edit</a>
                    <a
th:href="@{/article/delete/{id}(id=${article.id})}">Delete</a>
                </td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

let's create a complete Spring Boot project step-by-step. Below is a simplified version of how to create a project with the described functionalities.

Step 1: Set Up Spring Boot Project

1. **Create a new Spring Boot project using Spring Initializr**:
   - Visit Spring Initializr
   - Select the following dependencies: Spring Web, Spring Data JPA, MySQL Driver, Thymeleaf, Spring Boot DevTools, Spring Boot Starter Mail.
   - Generate the project and unzip it.
2. **Add dependencies to `pom.xml`**:

```
xml

<dependencies>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <!-- MySQL Driver -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <!-- Thymeleaf -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!-- Spring Boot DevTools -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional>
    </dependency>
    <!-- Spring Boot Starter Mail -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
    <!-- Lombok (Optional, for reducing boilerplate code) -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
    </dependency>
    <!-- Log4j2 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>
    <!-- Spring Boot Starter Test -->
```

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

3. **Configure `application.properties`**:

```
properties

# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/yourdatabase
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

# Log4j2 Configuration
logging.level.root=info
logging.config=classpath:log4j2.xml

# Email Configuration
spring.mail.host=smtp.example.com
spring.mail.port=587
spring.mail.username=your-email@example.com
spring.mail.password=your-password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

4. **Create `log4j2.xml` in `src/main/resources`**:

```
xml

<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n"/>
        </Console>
        <File name="File" fileName="logs/app.log">
            <PatternLayout>
                <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -
%m%n</pattern>
            </PatternLayout>
        </File>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="File"/>
        </Root>
    </Loggers>
</Configuration>
```

Step 2: Create Entities

1. **Article Entity**:

```
java

package com.example.demo.entity;

import javax.persistence.*;
import java.time.LocalDate;

@Entity
public class Article {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @Lob
    private String description;

    private LocalDate publishDate;

    private String status;

    private String banner;

    @ManyToOne
    @JoinColumn(name = "author_id")
    private Author author;

    // Getters and Setters
}
```

2. **Author Entity**:

```
java

package com.example.demo.entity;

import javax.persistence.*;
import java.util.List;

@Entity
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "author")
    private List<Article> articles;

    // Getters and Setters
}
```

Step 3: Create Repositories

1. **ArticleRepository**:

```
java
```

```
package com.example.demo.repository;

import com.example.demo.entity.Article;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ArticleRepository extends JpaRepository<Article,
Long> {}
```

2. **AuthorRepository**:

```
java

package com.example.demo.repository;

import com.example.demo.entity.Author;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AuthorRepository extends JpaRepository<Author, Long>
{}
```

Step 4: Create Services

1. **ArticleService**:

```
java

package com.example.demo.service;

import com.example.demo.entity.Article;
import com.example.demo.repository.ArticleRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ArticleService {
    @Autowired
    private ArticleRepository articleRepository;

    public List<Article> getAllArticles() {
        return articleRepository.findAll();
    }

    public Article getArticleById(Long id) {
        return articleRepository.findById(id).orElse(null);
    }

    public void saveArticle(Article article) {
        articleRepository.save(article);
    }

    public void updateArticle(Long id, Article article) {
        article.setId(id);
        articleRepository.save(article);
    }

    public void deleteArticle(Long id) {
```

```
        articleRepository.deleteById(id);
    }
}
```

2. **AuthorService**:

```
java

package com.example.demo.service;

import com.example.demo.entity.Author;
import com.example.demo.repository.AuthorRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class AuthorService {
    @Autowired
    private AuthorRepository authorRepository;

    public List<Author> getAllAuthors() {
        return authorRepository.findAll();
    }
}
```

Step 5: Create Controllers

1. **ArticleController**:

```
java

package com.example.demo.controller;

import com.example.demo.entity.Article;
import com.example.demo.service.ArticleService;
import com.example.demo.service.AuthorService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
public class ArticleController {
    @Autowired
    private ArticleService articleService;
    @Autowired
    private AuthorService authorService;

    @GetMapping("/articles")
    public String getAllArticles(Model model) {
        model.addAttribute("articles",
articleService.getAllArticles());
        return "list-articles";
    }

    @GetMapping("/article/new")
    public String showCreateForm(Model model) {
```

```
        model.addAttribute("article", new Article());
        model.addAttribute("authors", authorService.getAllAuthors());
        return "add-article";
    }

    @PostMapping("/article")
    public String saveArticle(@ModelAttribute("article") Article
article) {
        articleService.saveArticle(article);
        return "redirect:/articles";
    }

    @GetMapping("/article/edit/{id}")
    public String showEditForm(@PathVariable("id") Long id, Model
model) {
        model.addAttribute("article",
articleService.getArticleById(id));
        model.addAttribute("authors", authorService.getAllAuthors());
        return "edit-article";
    }

    @PostMapping("/article/update/{id}")
    public String updateArticle(@PathVariable("id") Long id,
@ModelAttribute("article") Article article) {
        articleService.updateArticle(id, article);
        return "redirect:/articles";
    }

    @GetMapping("/article/delete/{id}")
    public String deleteArticle(@PathVariable("id") Long id) {
        articleService.deleteArticle(id);
        return "redirect:/articles";
    }
}
```

Step 6: Configure Email Service

1. **EmailService**:

```
java

package com.example.demo.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;

@Service
public class EmailService {
    @Autowired
    private JavaMailSender mailSender;

    public void sendEmail(String to, String subject, String body) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject(subject);
        message.setText(body);
        mailSender.send(message);
    }
```

```
}
```

2. **Integrate EmailService in ArticleService**:

```
java

@Autowired
private EmailService emailService;

public void saveArticle(Article article) {
    articleRepository.save(article);
    emailService.sendEmail("admin@example.com", "New Article Added",
"Article titled '" + article.getTitle() + "' has been added.");
}
```

Step 7: User Interface (Thymeleaf Templates)

1. **list-articles.html**:

```
html

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Articles</title>
</head>
<body>
    <h1>Articles</h1>
    <a href="/article/new">Add New Article</a>
    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Title</th>
                <th>Publish Date</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="article : ${articles}">
                <td th:text="${article.id}">1</td>
                <td th:text="${article.title}">Title</td>
                <td th:text="${article.publishDate}">2021-01-01</td>
                <td>
                    <a
th:href="@{/article/edit/{id}(id=${article.id})}">Edit</a>
                    <a
th:href="@{/article/delete/{id}(id=${article.id})}">Delete</a>
                </td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

2. **add-article.html**:

```
html
```

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Article</title>
</head>
<body>
    <h1>Add Article</h1>
    <form action="#" th:action="@{/article}" th:object="${article}"
method="post">
        <div>
            <label>Title:</label>
            <input type="text" th:field="*{title}" />
        </div>
        <div>
            <label>Description:</label>
            <textarea th:field="*{description}"></textarea>
        </div>
        <div>
            <label>Publish Date:</label>
            <input type="date" th:field="*{publishDate}" />
        </div>
        <div>
            <label>Author:</label>
            <select th:field="*{author}">
                <option th:each="author : ${authors}"
th:value="${author.id}" th:text="${author.name}"></option>
            </select>
        </div>
        <div>
            <button type="submit">Save</button>
        </div>
    </form>
    <a href="/articles">Back to Articles</a>
</body>
</html>
```

3. **edit-article.html**:

```
html

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit Article</title>
</head>
<body>
    <h1>Edit Article</h1>
    <form action="#"
th:action="@{/article/update/{id}(id=${article.id})}"
th:object="${article}" method="post">
        <div>
            <label>Title:</label>
            <input type="text" th:field="*{title}" />
        </div>
        <div>
            <label>Description:</label>
            <textarea th:field="*{description}"></textarea>
        </div>
        <div>
```

```
            <label>Publish Date:</label>
            <input type="date" th:field="*{publishDate}" />
        </div>
        <div>
            <label>Author:</label>
            <select th:field="*{author}">
                <option th:each="author : ${authors}"
th:value="${author.id}" th:text="${author.name}"></option>
            </select>
        </div>
        <div>
            <button type="submit">Update</button>
        </div>
    </form>
    <a href="/articles">Back to Articles</a>
</body>
</html>
```

Step 8: Exception Handling

1. **GlobalExceptionHandler**:

```
java

package com.example.demo.exception;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionHandler {
    private static final Logger logger =
LogManager.getLogger(GlobalExceptionHandler.class);

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception e) {
        logger.error("Exception: ", e);
        return new ResponseEntity<>("An error occurred: " +
e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

This setup provides a complete example of how to build a Spring Boot application with CRUD operations, logging, error handling, email notifications, and a user-friendly UI.