Gift Certificates

Spring mvc + REST. Estimated time: 5 days

2019

# Business Requirements

Develop server application for Gift Certificates system with the following entities:

1. Gift Certificate:
   1. name
   2. duration
   3. price
   4. date of creation
   5. date of modification
   6. duration in days *(expiration period)*
2. Tag

0

0

The system should expose REST APIs to perform the following operations:

1. CRUD operations for *Gift Certificate*.
   1. If *new tags* are passed during creation or modification of a Gift Certificate, then they should be created in the database.
2. CRD operations for *Tag*.
3. Get certificates. All request parameters are *optional* and *can be used in conjunction*.
   1. Get certificates by tag.
   2. Search certificates by part of the name or description (should be implemented using database function call).
   3. Sort certificates by date or name.

-------------------------------------

Система должна предоставлять API REST для выполнения следующих операций:

1. CRUD-операции для подарочного сертификата.

а. Если при создании или изменении подарочного сертификата передаются новые теги, их следует создать в базе данных.

2. Операции CRD для Tag.

3. Получить сертификаты. Все параметры запроса являются необязательными и могут использоваться совместно.

а. Получить сертификаты по тегу.

б. Поиск сертификатов по части имени или описания (должен быть реализован с помощью вызова функции базы данных).

с. Сортировать сертификаты по дате или имени.

----------------------------------------------

# Technical requirements

## General Requirements:

1. Code should be clean and should not contain any “developer-purpose” constructions.
2. Application should be designed and written with respect to OOD and SOLID principles.
3. Code should contain valuable comments where appropriate.
4. Public APIs should be documentedusing Javadoc.
5. A clear layered structure should be used: responsibilities of each application layer should be defined.
6. JSON should be used as a format of client-server communication messages.
7. Convenient error and exception handling mechanism should be implemented: all errors should be meaningful and localized on backend side.
8. Abstraction should be used to avoid code duplication.

--------------------------------------------------------

Технические требования

Общие требования:

1. Код должен быть чистым и не должен содержать каких-либо «предназначенных для разработчиков» конструкций.

2. Приложение должно быть разработано и написано с учетом принципов OOD и SOLID.

3. Код должен содержать ценные комментарии, где это уместно.

4. Публичные API должны быть документированы с использованием Javadoc.

5. Должна использоваться четкая многоуровневая структура: должны быть определены обязанности каждого прикладного уровня.

6. JSON следует использовать в качестве формата клиент-серверных коммуникационных сообщений.

7. Должен быть реализован удобный механизм обработки ошибок и исключений: все ошибки должны быть осмысленными и локализованными на стороне сервера.

8. Абстракция должна использоваться, чтобы избежать дублирования кода.

-----------------------------------------------------------------

## Tools and Implementation Requirements:

**Please note that only GA versions of tools, frameworks, and libraries are allowed.**

1. JDK version: 8**.** Use *Streams, java.time.\*, etc.* where it is appropriate.
2. Application package root: com.epam.esm
3. Custom connection pool implementation should be used**:** 
   1. It should support lazy concurrent loading.
   2. It should be created once per JVM(exposed as a Spring Bean).
   3. If Proxy pattern is used, it should be implemented without code duplication (JDK Proxy or any other).
4. JDBC or Spring JDBC Template should be used for data access.
5. Java Code Convention is mandatory. Exception: margin size – 120 chars**.**
6. Build tool: Apache Maven 3.5+. Multi-module project.
7. Web server: Apache Tomcat**.**
8. Application container: Spring IoC. Spring Framework4.+
9. Database: PostgreSQL 9.+ or 10.+
10. Testing: JUnit 4.+ or 5.+, Mockito.
11. Service layer should be covered with unit tests. Unit tests coverage should be not less than 80%.
12. Repository layer should be tested using integration tests with an in-memory embedded database. All operations with certificates should be covered with integration tests.
13. APIs should be demonstrated using Postman tool. Postman collections with APIs should be prepared for the demo.

------------------------------------------------------------

Инструменты и требования к реализации:

Обратите внимание, что разрешены только версии инструментов, платформ и библиотек GA.

1. Версия JDK: 8. Используйте Streams, java.time. \* И т. Д., Где это уместно.

2. Корень пакета приложения: com.epam.esm

3. Использовать пользовательский пул соединений:

а. Он должен поддерживать ленивую одновременную загрузку.

б. Он должен быть создан один раз для каждой JVM (представлен как Spring Bean).

с. Если используется шаблон Proxy, он должен быть реализован без дублирования кода (JDK Proxy или любой другой).

4. JDBC или Spring JDBC Template должны использоваться для доступа к данным.

5. Соглашение Java Code является обязательным. Исключение: размер маржи - 120 символов.

6. Инструмент для сборки: Apache Maven 3.5+. Мультимодульный проект.

7. Веб-сервер: Apache Tomcat.

8. Контейнер для приложений: Spring IoC. Spring Framework 4. +

9. База данных: PostgreSQL 9. + или 10. +

10. Тестирование: JUnit 4. + или 5. +, Mockito.

11. Сервисный слой должен быть покрыт юнит-тестами. Охват модульных тестов должен составлять не менее 80%.

12. Уровень репозитория должен быть протестирован с использованием интеграционных тестов со встроенной базой данных в памяти. Все операции с сертификатами должны быть покрыты интеграционными тестами.

13. API должны быть продемонстрированы с помощью инструмента Postman. Коллекции почтальонов с API должны быть подготовлены для демонстрации.

-------------------------------------------------------------------------------

## Restrictions

It is forbidden to use:

1. Spring Boot,
2. Spring Data Repositories,
3. JPA,
4. Lombok,
5. Powermock.

# Recommended Materials

This section contains links and information on materials recommended for self-study.

1. [Spring in Action, 4th edition](https://www.manning.com/books/spring-in-action-fourth-edition)
2. Core concepts – Ch. 2: Wiring beans
3. Core concepts – Ch. 3: Advanced Wiring
4. MVC – Ch. 5: Building Spring web applications
5. MVC – Ch. 7: Advanced Spring MVC
6. REST – Ch. 16: Creating REST APIs with Spring
7. [Foundations of RESTful architecture & REST practices](https://dzone.com/refcardz/rest-foundations-restful)
8. [Best Practices for Designing a Pragmatic RESTful API](http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api)
9. [Забудьте о DAO, используйте Repository](https://habrahabr.ru/post/263033/)
10. [Maven Multi Module Project](http://www.codetab.org/apache-maven-tutorial/maven-multi-module-project/)
11. [Data Transfer Object](https://martinfowler.com/eaaCatalog/dataTransferObject.html)
12. [Understanding Spring Web Application Architecture: The Classic Way](https://www.petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/)
13. [PowerMock examples and why better not to use them](https://automationrhapsody.com/powermock-examples-better-not-use/)