Radolina Petrova

Design document

RADIFY – music platform

Table of Contents

[C4 Software Architecture 2](#_Toc120064281)

[**C1** 2](#_Toc120064282)

[**C2** 3](#_Toc120064283)

[**C3** 4](#_Toc120064284)

[**C4** 5](#_Toc120064285)

[SOLID principles 6](#_Toc120064286)

[S: Single Responsibility Principle 6](#_Toc120064287)

[O: Open-closed Principle 6](#_Toc120064288)

[L: Liskov Substitution Principle 6](#_Toc120064289)

[I: Interface Segregation Principle 6](#_Toc120064290)

[D: Dependency inversion principle 6](#_Toc120064291)

[Design choices 7](#_Toc120064292)

[ORM tool 7](#_Toc120064293)

[Database 7](#_Toc120064294)

# C4 Software Architecture

## **C1**

Diagram

Description automatically generated

## **C2**

Diagram

Description automatically generated

## **C3**

Diagram

Description automatically generated

## **C4**

Diagram

Description automatically generated

# SOLID principles

## S: Single Responsibility Principle

The single responsibility principle states that “A class should have only one reason to change”, says the originator of the term, Robert C. Martin. This principle is applied in my project by dividing the classes of every architectural layer, so that each one of them is responsible for the functionalities related to one entity.

## O: Open-closed Principle

The Open-Closed principle follows the logic, that software entities should be open for extension and closed for modification. This is mainly achieved by inheritance and polymorphism in general, but since these techniques are not applicable in my project, so the only thing I can do to make sure my entities are extendable, was passing the whole objects and having converters in the separate layers, so if any part of this entity is modified, it will require minimal changes in the converter method.

## L: Liskov Substitution Principle

The Liskov Substitution principle states that objects of a superclass should be replaceable with objects of its subclasses without breaking the application. I have applied this principle by ensuring every class, that implements a certain interface, also implements all its methods.

## I: Interface Segregation Principle

The interface segregation principle states that no code should be forced to depend on methods it does not use. It is easily applicable by simply splitting larger interfaces into smaller, more specific ones, so they are easier to refactor and change.

## D: Dependency inversion principle

The dependency inversion principle states, that high-level modules should not import anything from low-level modules and that both should depend on abstractions. I have applied this principle in my application by creating interfaces for each class in both the business and data access layer, as well as ones for conversion of the entities. Applying this principle allows less implementation-dependant schema and reduce the components without introducing additional coding patterns.

# Design choices

## ORM tool

For an ORM tool I chose the framework ‘Hibernate’, not only because it is open source and lightweight. The fast performance was also one of the reasons, as well as automatic table creation of tables in the database. This framework also not only saves the developer time for writing queries, but if there is need for one it is simple to fetch the data, using HQL.

## Database

For a database I chose MySQL, mainly because it is one of the most popular relational databases, which ensures protection of the saved data. It is also easy to work with.