CONTEXTPROJECT

Emergent Architecture Design, TI2806

Authors:
S.A. BOODT, sboodt, 4322258
T. HEINSOHN HUALA, theinsohnhuala, 4326318
A. HOVANESYAN, ahovaneyan, 4322711
D. SCHIPPER, dschipper, 4155270

Friday 19th June, 2015

Contents

1 Introduction **1**
1.1 Design goals . . . 1

2 Software architecture views **2**
2.1 Subsystem decomposition . . . 2
2.2 Hardware/software mapping . . . 3
2.3 Persistent data management . . . 5
2.3.1 Database . . . 5
2.3.2 Data . . . 5
2.3.3 Connection . . . 5
2.4 Concurrency . . . 5
2.4.1 Processes . . . 5
2.4.2 Shared resources . . . 6
2.4.3 Communication between processes . . . 6
2.4.4 Deadlock prevention . . . 6

3 Design Patterns **7**
3.1 Decorator . . . 7
3.2 Factory . . . 7
3.3 Model View Controller . . . 8
3.4 Singleton . . . 8

4 Glossary **9**

5 Bibliography **10**

1

Introduction

The report consists of the following components. Firstly we will discuss the design goals of our project, next in chapter 2, the software architecture will be described from different views. Used design patterns and where to find them are described in chapter 3.Finally a glossary is defined in chapter 4.

1.1 Design goals

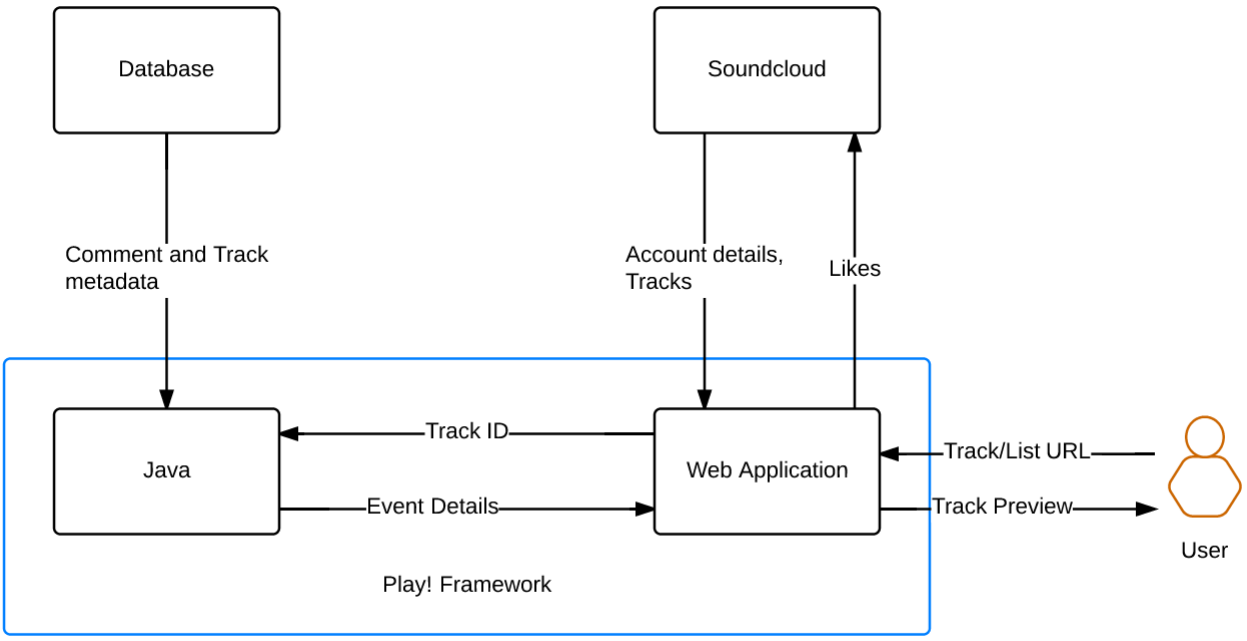
Besides the goals described in the Product Vision document, we also have goals concerning the architecture of the product. These goals are the design goals.

- Availability
  The first design goal we will maintain during this project is availability;
  this means the product will be available at any time. Every week when the product is showed to the client, this will always be a working version of the product.
- Manageability
  The second design goal is manageability;
  this means that the system will be well commented and easy to read. So a developer who isn't involved in the project can easily understand the code.
- Simplicity
  It is very important for us that the application is easy to use. This goals is only achievable if the applications looks and controls are intuitive.
- Scalability
  SoundCloud is currently the biggest platform for user-generated music. To be able to support such a size of content the application needs to be scalable. This can be done by storing and retrieving the data from a database. In cases that is not possible there the application can directly ask for the data from the SoundCloud server.

2

Software architecture views

2.1 Subsystem decomposition



- SoundCloud: The SoundClouds api gives us the possibility to load their custom widget that plays the tracks existing on their server. Through the api we can ask the server for the id of the currently selected track or even other information. The id of the track can then be used to identify its comments within our database. If our database does not contain any information about the currently playing track it is even possible to ask the SoundCloud server for the track's comments and features. We integrated SoundCloud because SoundCloud is our commissioning company.
- Database: The database contains all the meta-data of the tracks including the comment meta-data. When the application receives a track ID, it searches the database for the corresponding comments and other details. This data is then processed and passed on. The database is also used select certain songs, at random or based on some track characteristics. We decided to use a database because it gives a really ordered view of our data and allows an easy way to search for specific values, namely by using SQL.
- Java: The java part of the program processes the SoundClouds track and comment meta-data to calculate the songs preview start time with a given window. To be able to do this it receives a track id from the user interface which is in this case the web application. After processing the data a responsible java object outputs a start time and a window for the given track.

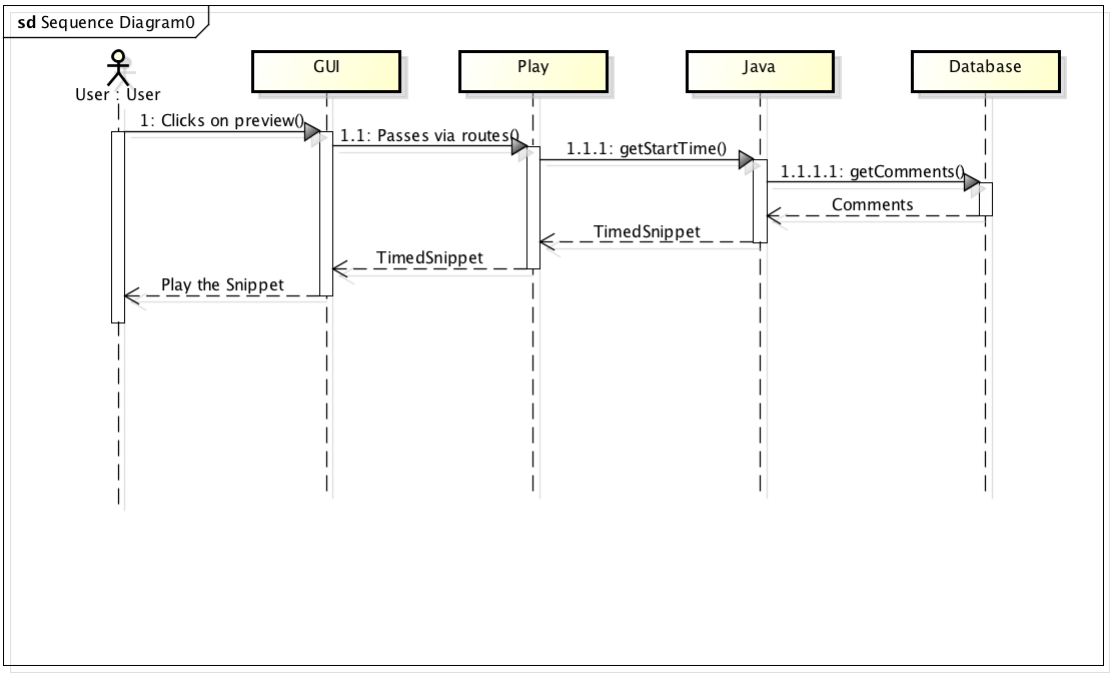


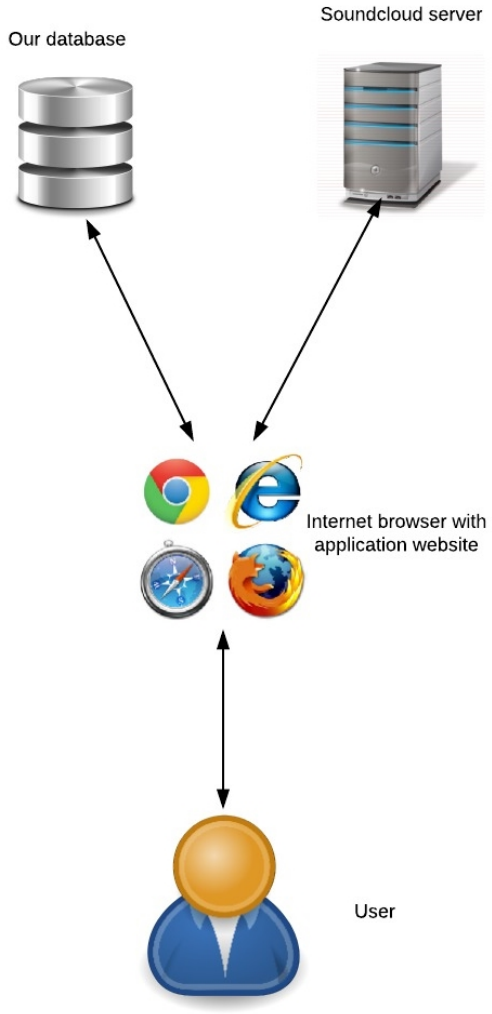
Figure 2.1: A sequence diagram of a preview request

- Web application: The web application holds the SoundClouds widget and buttons to control it. Through the interface the user can pick a song and play a preview/snippet. If the user is logged in he can then add the song to his collections if he likes the song. Being logged in gives the user the opportunity to see his own collection. We decided to make a web application because it allows an easy way of displaying our data and allows it to be used on many different platforms. It also allows us to make use of the SoundCloud widget.
- Play! Framework: We use Play! to be able to run the java and web applications alongside each other. It's biggest responsibility is to make sure the data, such as track information and event details(start and end times), is exchanged.

These different things interact as specified in Sequence diagram 2.1.

2.2 Hardware/software mapping

The hardware we use are the computer and the SoundCloud servers. The computer will be used for using the program with a browser. The SoundCloud servers are used for obtaining music. Furthermore, the software we use are a standard browser, the framework in which we can play songs and a framework to use the product as a web page . The standard browser is just the place where our program will be visualized. The framework for the web page is the software we use to connect java with a web page and to play our songs. The other framework is for working with SoundCloud.


Soundcloud server
Our database
Internet browser with
application website
User

2.3 Persistent data management

2.3.1 Database

The data is saved on a remotely hosted MySQL server for easy access by all machines that have the Play! application running. The application uses a database connector to execute queries and retrieve the data that is needed.

2.3.2 Data

The data we have received for a track is split up into three different tables. The first table consist of all the comments. Each comment has the following attributes:

- track id
  The id of the track the comment is posted on
- comment id
  The id of the comment, this is a unique variable
- user id The id of the user that has posted this comment
- created at The time of the creation of the comment
- timestamp The timestamp of the comment. Each comment can be posted at a certain time of the track.
- text The content of the comment

Each comment can be solicited for and the application will receive all these attributes back from the server. For instance to receive all the comments of a single track to determine the snippet a simple query can be made to select all the comments with a certain track id.

The second table has all the meta-data of a track stored, such as the title of the track. This can be used to retrieve additional information about the track besides the track id.

The last table has all feature essentia. Since these were provided in a json format, the feature essentia are stored as a file. The java applet can quickly retrieve all the information about a track and browse through all the tags in the file.

2.3.3 Connection

Since the java application only receives the track id from the web application, it is vital that all the data stored on the database has a field of the track id. This is the case in the database, so the java applet can retrieve all the information from the database if only the track id of a track is know.

2.4 Concurrency

2.4.1 Processes

As described in the Subsystem decomposition the application is divided between four different subsystems. Each of these four subsystems has it's own process running. Currently there is no thread assigned by either the java system or the web application.

Each user has it's own current instance of the web application running so there is a many to one relationship between the web application and the selecting java code. This means that although the java subsystem currently does not create any additional processes, there are still many players requesting stuff from it. As the java subsystem is integrated in a Server, currently localhost for testing purposes, this is handled with automatically as the HTTP requests are handled simultaneous by any server. It depends on the server whether or not the java code is executed simultaneous.

What is known is that the database handles any SQL query as if it was it's own process. Therefore it is so that for every call to the database a new thread will be made on the database side, and as such the amount of processes in the database will match with the amount of processes on the java application during some of the time.

2.4.2 Shared resources

As described in Persistent data management the largest shared resource of this system is the database that holds the data that is needed to find the requested information.

Another shared resource is the server that runs the java application. While currently being localhost for testing purposes, this cannot always remain this way. As our product will be launched, hopefully more users will use our product. When multiple users use our product the server handling the request to search for snippets will have a lot more to do than it does when we test it and there is nobody else to use it. In other words the server will be divided between multiple users. This makes the server a shared resource as it needs to keep running in order for the application to work and without it all of the processes will fail in some way.

2.4.3 Communication between processes

Currently no threads are made by the java sub component itself. The web application is not capable of creating threads. While both SoundCloud and the database are external. The database is also merely a resource but it is a resource with different processes. However these processes do not communicate with each other, in any other way than the one explained in Deadlock prevention.

Due to this the communication between resources is currently more like the communication between the different subsystems. In short the communication between the different subsystems go over the internet. The java code outputs it's output into the web component. The web page sends the input to the java part over the same connection but then in the other direction. The web component can hold contact with SoundCloud over the internet as well.

2.4.4 Deadlock prevention

Subsystem decomposition also shows us that the java part awaits for the web application to give it a track id, then requests the database and then only returns back the answer. This means that there is no cyclic relationship between subsystems.

The deadlock inside the database, caused by colliding queries, should be taken care of by the DBMS. This DBMS is not part of our product and therefore should not be discussed here, but every DMBS guarantees that deadlock is prevented in some way. Most DBMS use locks to achieve this goal and simply lock the data being accessed so no others can access it until the query is done accessing it.

Deadlock is also prevented because no two methods in the code rely on each other, either directly or indirectly. This means that there is no dependency cycle in the code. On top of that the current code is built so there is only one thread and as such there are no shared software components, other than the database, that disallow concurrent modification. Due to this there cannot be a deadlock in the code.

3

Design Patterns

This chapter describes all the design patterns we implemented and the classes we implemented them in.

3.1 Decorator

The decorator pattern is a pattern most commonly used to combine a set of similar classes and offers any combination of their utilities. This pattern uses a common class or interface as parameter for it's constructors and then calls the decorated classes methods in the classes own methods.

Classes, or Interfaces, implementing this pattern:

- Seeker
- Recommender

For a visualisation of the pattern see Figure 3.1 The operation method in the decorator hereby calls the operation method on it's decorated class.

3.2 Factory

The factory design pattern helps the construction of instances of another class. One of the benefits from this is that more things can be done during construction as the constructed instance can be passed as a parameter. It also allows multiple methods with the same variables but with different meaning to them, because a method can have a different name. This pattern consists of a factory class and a class that it produces.

Here is a list of factory classes. Classes produced by the factory have the same name but do not have the factory ending in it, unless stated otherwise.

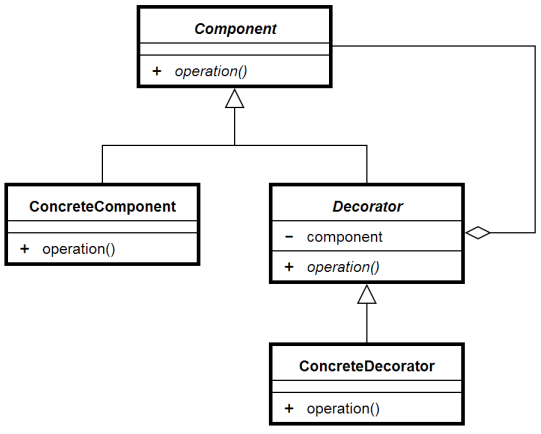


Figure 3.1: A figure explaining the Decorator pattern. Source: https://designpatternsinjava.wordpress.com/2012/11/18/decorator-pattern/

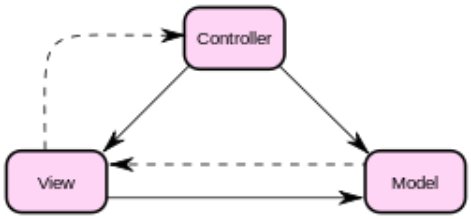


Figure 3.2: A figure explaining the MVC pattern. Source: https://nl.wikipedia.org/wiki/Model-view-controller-model

- models.snippet.TimedSnippetFactory

3.3 Model View Controller

This pattern is used to separate application's concerns into three different parts:

- Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
- View - View represents the visualization of the data that model contains.
- Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Usage of the Model View Controller design pattern is used throughout the whole project. The entire structure of the project is split in three packages: models, views and controllers. The controllers accept information from the view, the web page, and decide based on the request from the view which models to call.
The controllers are the Application, AlgorithmSelector and UserActionControllers classes in the controller package. Application receives information from the view about which track to determine the snippet for, as well as to return a random song. AlgorithmSelector controls the selection of the snippet, and uses the right seeker(s). UserActionControllers determines the tracks to recommend based on the likes from the user.
The graphical user interface also uses this design pattern, because the HTML is the model, CSS is the view and javascript is the controller.

See Figure 3.2 for a picture of a MVC structure.

3.4 Singleton

The singleton pattern is a pattern that makes sure there is only one instance of the class. The singleton pattern is meant to reduce the number of utility classes and offers an alternative that is light on memory, because of the one instance, but does offer the advantages of the liskov substitution principle. Singleton classes only contain private constructors and static methods to get the instance.

Classes implementing this pattern:

- models.database.DatabaseConnector
- models.database.RandomSongSelector
- models.database.DatabaseUpdater
- models.score.XMLScoreParser

4

Glossary

api Application program interface. An interface that is provided so other programs can call the methods provided by the Application.

concurrency Executing two or more processes at roughly the same time.

CSS A language often used to give a certain look to a web page.

deadlock A state in which two or more processes are waiting for each other to finish before they can continue.

DBMS Database management system, a system that makes some guarantees and rules about how the database should be accessed.

HTML Hyper text markup language. Common language used to write web pages.

javascript A language often used to allow functions in a web page.

JSON A language used for efficient and organized storage of data.

localhost An identifier used as an address to specify the currently used machine. Often used for testing and debugging.

liskov substitution principle One of the key principles for inheritance, and therefor for any Object oriented programming language.

lock A way to prevent others the usage of the locked object.

MySQL A database management system that is often used online.

process Something capable of running code. Multiple can exist at the same time on the machine.

query A question asked to a database.

shared resource A resource that multiple processes have in common.

SoundCloud A popular music listening platform.

snippet Small fragment of music.

SQL A commonly used query language for databases.

thread Roughly the same as process.

timestamp A number that gives a relative time to everything else.

utility class A class containing only static functions.

5

Bibliography

- trashtoy (2007). Decorator UML class diagram. Retrieved June 19, 2015, from https://designpatternsinjava.wordpress.com/2012/11/18/decorator-pattern/ .Originally from: https://en.wikipedia.org/wiki/Decorator_pattern
- User:Stannered (2010). ModelViewControllerDiagram.svg. Retrieved June 19, 2015, from https://nl.wikipedia.org/wiki/Model-view-controller-model
- Yadati, K., Larson, M., Liem, C., & Hanjalic, A. (2014). Detecting drops in electronic dance music: content based approaches to a socially significant music event (master thesis). Retrieved from: http://www.terasoft.com.tw/conf/ismir2014/proceedings/T026_297_Paper.pdf on 27-04-2015 (Delft University of Technology).
- Tutorialspoint.com,http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm