**FACULTY OF SCIENCE, ENGINEERING AND COMPUTING**

**School of *Computing and Information Systems***

**BSc DEGREE**

**IN**

***Computer Science***

**PROJECT REPORT**

Name: Carlos Esparragoza

ID Number: K1418717

Project Title: **Avinpro Database Management System**

Project Type: Build

Date: 24/04/2017

Supervisor: David Livingston

KU London Logo

Did you discuss and agree the viability of your project idea with your supervisor? Yes

Did you submit a draft of your proposal to your supervisor? Yes

Did you receive feedback from your supervisor on any submitted draft? Yes

Table of contents

[Declaration form 7](#_Toc480828379)

[7](#_Toc480828380)

[Chapter 1: Introduction 8](#_Toc480828381)

[1.1 Background 8](#_Toc480828382)

[1.2 The client 8](#_Toc480828383)

[1.3 Justification of the project and solutions 9](#_Toc480828384)

[1.4 Project Scope 9](#_Toc480828385)

[1.5 Report Schedule 9](#_Toc480828386)

[Chapter 2: Literature Review 10](#_Toc480828387)

[2.1 Introduction 10](#_Toc480828388)

[2.2 Music Licensees 10](#_Toc480828389)

[2.3 Royalties 10](#_Toc480828390)

[2.3 Build Requirements 11](#_Toc480828391)

[2.3.1 Java Language 11](#_Toc480828392)

[2.3.2 MySQL Database 11](#_Toc480828393)

[2.3.3 IntelliJ Integrated Development Environment (IDE) 11](#_Toc480828394)

[2.3.4 StarUML 12](#_Toc480828395)

[2.3.5 MySQL Workbench 12](#_Toc480828396)

[2.3.6 Java Database Connectivity 12](#_Toc480828397)

[2.3.7 JavaFX 12](#_Toc480828398)

[2.3.8 XML Language 12](#_Toc480828399)

[2.4 License Agreement 13](#_Toc480828400)

[2.5 Royalty Rate 13](#_Toc480828401)

[2.6 Copyright Infringement 13](#_Toc480828402)

[2.7 Working Interests 13](#_Toc480828403)

[2.8 SWOT Analysis 14](#_Toc480828404)

[2.8.1 SWOT Analysis breakdown 14](#_Toc480828405)

[2.9 Model View Controller (MVC) 14](#_Toc480828406)

[2.10 Database Management System (DBMS) 14](#_Toc480828407)

[2.11 Use case diagram (UML) 15](#_Toc480828408)

[Chapter 3: Analysis 16](#_Toc480828409)

[3.1 Introduction 16](#_Toc480828410)

[3.2 SWOT Analysis 16](#_Toc480828411)

[3.2.1 SWOT Analysis before build 17](#_Toc480828412)

[3.3 Functional and Non-Functional Requirements 17](#_Toc480828413)

[3.3.1 Functional Requirements 17](#_Toc480828414)

[3.3.2 Non-Functional Requirements 18](#_Toc480828415)

[3.4 MoSCoW Analysis 18](#_Toc480828416)

[3.4.1 Functional requirements in MoSCoW 18](#_Toc480828417)

[3.5 Use Case Diagram 19](#_Toc480828418)

[Figure 3.5.1 20](#_Toc480828419)

[3.6 Risk Analysis 20](#_Toc480828420)

[Figure 3.5.1 – Risk Analysis 20](#_Toc480828421)

[3.6 Interview done with the customer 21](#_Toc480828422)

[Chapter 4: Design 23](#_Toc480828423)

[4.1 Introduction 23](#_Toc480828424)

[4.2 Wireframes 23](#_Toc480828425)

[Figure 4.2.1 Initial Design 23](#_Toc480828426)

[Figure 4.2.2 Second Design 24](#_Toc480828427)

[4.3 Database Design – Class Diagram 24](#_Toc480828428)

[Figure 4.3.1 25](#_Toc480828429)

[Chapter 5: Implementation 26](#_Toc480828430)

[5.1 Introduction 26](#_Toc480828431)

[5.2 Methodology Used 26](#_Toc480828432)

[5.2 Database creation 26](#_Toc480828433)

[Figure 5.2.1 Creating database 27](#_Toc480828434)

[Figure 5.2.2 Creating tables to store information 27](#_Toc480828435)

[Figure 5.2.3 Creating attributes for table Artist SQL. 28](#_Toc480828436)

[Figure 5.2.4 Creating attributes for table Song SQL. 28](#_Toc480828437)

[Figure 5.2.5 Creating attributes for table groupArtist SQL. 28](#_Toc480828438)

[Figure 5.2.6 Inserted values for attributes artist table 29](#_Toc480828439)

[5.3 Creation of the graphical user interface 29](#_Toc480828440)

[5.3.1 Root Layout controller 29](#_Toc480828441)

[5.3.2 Root layout View 30](#_Toc480828442)

[Figure 5.3.2.1 31](#_Toc480828443)

[5.3.3 Artist Controller class 31](#_Toc480828444)

[5.3.4 Artist class 35](#_Toc480828445)

[5.3.5 ArtistDao class 37](#_Toc480828446)

[5.3.6 Artist View class 40](#_Toc480828447)

[5.3.7 Song Controller class 42](#_Toc480828448)

[5.3.8 Song class 44](#_Toc480828449)

[5.3.9 SongDao Class 46](#_Toc480828450)

[5.3.10 Song view Class 48](#_Toc480828451)

[5.3.11 DBUtil Class 49](#_Toc480828452)

[Chapter 6: Testing and Results 51](#_Toc480828453)

[6.1 Introduction 51](#_Toc480828454)

[6.2 Testing 52](#_Toc480828455)

[6.2.1 Black-box Testing 52](#_Toc480828456)

[6.2.2 Adding an Artist Record 53](#_Toc480828457)

[6.2.3 Searching Artist by ID 54](#_Toc480828458)

[6.2.4 Search All artists 55](#_Toc480828459)

[6.2.5 Delete Artist Record 55](#_Toc480828460)

[6.2.6 Filtering by Artist Gender 57](#_Toc480828461)

[6.2.7 Filtering by Artist Type 58](#_Toc480828462)

[6.2.8 Changing between views 59](#_Toc480828463)

[6.2.9 Adding a Song Record 60](#_Toc480828464)

[6.2.10 Searching All Song Records 60](#_Toc480828465)

[6.2.11 Searching a Song Record by ID 61](#_Toc480828466)

[6.2.12 Deleting a Song Record by ID 62](#_Toc480828467)

[6.3 Results 63](#_Toc480828468)

[6.3.1 Results from Black-box Testing the Avinpro manager System 63](#_Toc480828469)

[6.3.2 SWOT Analysis after build of the Avinpro Manager System 64](#_Toc480828470)

[Figure 6.3.2.1 65](#_Toc480828471)

[Chapter 7: Critical Review 66](#_Toc480828472)

[7.1 Introduction 66](#_Toc480828473)

[7.2 User Feedback 67](#_Toc480828474)

[7.3 Problems Encountered 67](#_Toc480828475)

[7.4 Achievements and Limitations 68](#_Toc480828476)

[7.5 Implications for Future Work 68](#_Toc480828477)

[7.6 Conclusion 69](#_Toc480828478)

[References 70](#_Toc480828479)

# Declaration form

## https://scontent-lht6-1.xx.fbcdn.net/v/t34.0-12/18136847_10155493797029396_1979258157_n.jpg?oh=6ed1805beb1edbc4a442f62fa3ab6a3c&oe=59011AA7

## Chapter 1: Introduction

### Background

The company Avinpro has being doing a process called distribution. This process is done every six months and takes quite some time to prepare for it and be able to do it, the goal of the distribution is to:

* Group all the songs performed by artists and their producers.
* Count the number of times each song was played on the radio or establishment.
* Distribute the royalties collected depending of the number of time each of this song was played.
* Produce a report for each artist/producer so they can see how much of this money should be going to them.

Currently the problem that they have showed is that, some of the reports come late as there are a lot of songs to process and the records are constantly changing. This is because every day these songs are played so radio companies update their records after a certain amount of time these records are sent to Avinpro so they can be added to their system. This then will change the amount of royalties an artist/producer will receive for their song.

There are errors made by the company’s workers i.e. Some artist/producers don’t get paid properly, this is a result of calculating the total amount of time played wrong, making typing errors without validation or the user having to confirm the updates made. A major issue is that all this records are kept in excel sheets this leads to repetition of data which causes error in payments.

### The client

The client is Mr. Hugo Esparragoza who is the general manager and the IT chief of the company. As mention before section 1.1, this user is taking an extremely long time to finish the process as he is the one that has finish up and calculate the amounts needed to be paid to each artist/producer.

I have come to him to offer help and he has asked me to create a managing system for this kind of information and transactions, that he could keep track and update and provide features that can help him produce more accurate reports in a shorter amount of time.

### Justification of the project and solutions

This project aims to reduce the time this company takes to prepare and do the distribution and to correctly create a custom validated report for each artist/producer. By using a Java database management system. It will include features that will help the user finish the process quicker such as in built function that will update the amount of money that artists/ producers based on the amount of times the song where played in a period of time.

The solution this project offers will include an interface that manages a database that contains all the information inside of all the records the company has. Can have functionalities will be added as the software develops and if the amount of time is sufficient, a possible idea for this added functionality would be store artist/producer emails and as soon as the process is finished they will receive their report of how much they have earned in royalties.

### Project Scope

The main objectives of the project are:

1. Create a fully functional database that will store, edit and delete the information required by the company to produce the reports, this database should able to hold at least 200000 records of artists. The database will be secured and regularly backed up to secure the data in case off loss of data or damage of the same one.
2. To design and build an interface able to deal with the functionality the user needs to fulfil the process of the Distribution. The minimum requirement is to be able to insert, update and delete data from the database.
3. To stablish a connection between the interface and the database using in this case a MySQL database management system.

### Report Schedule

|  |  |
| --- | --- |
| **Assessment deadlines** | **Return date** |
| Proposal  Early prototype demo  Final report  Viva | 04/11/2016  06/01/17  15/05/2017  26/05/17 |

# Chapter 2: Literature Review

## 2.1 Introduction

The purpose of this project is to introduce the company of Avinpro a more updated way to do the process of Distribution, this process consist if gathering information about the number of times songs were played in a country. Depending the amount of times each song was played the artists/producers that took part on the creation of that song will receive money (Royalties).

The project will present and approach which is managing a database using java programming language as an interface. This will make the agglomeration of data more presentable i.e. Filtering, validation, no repetition if data. Selecting information to create a report even more understandable i.e. Bespoke user interface.

## 2.2 Music Licensees

“A very large number of entities, from neighbourhood bars to broadcast radio stations, “perform” copyrighted musical works for the public. Such entities must pay royalties to copyright holders. The types of royalty payments owed and the way those payments are determined vary considerably, depending mainly upon the way an entity is treated under copyright law.

The laws that determine who pays whom in the digital world were written, by and large, at a time when music was primarily performed via radio broadcasts or distributed through physical media (such as sheet music and phonograph records), and when each of these forms of music delivery represented a distinct channel with unique characteristics. With the emergence of the Internet, Congress updated some copyright laws in the 1990s. It applied one set of legal provisions to digital services it viewed as akin to radio broadcasts and another set to digital services it viewed as akin to physical media. Since that time consumers have increasingly been consuming music via digital services that incorporate attributes of both radio and physical media. However, companies that compete in enabling consumers to access music may face very different costs to license music, depending on the technology they use and the features they offer. These differences in technology and features also affect the amount of money received by songwriters, performers, music publishers, and record companies.” Gravelle, J. G. (2016). Money for something: Music licensing in the 21st century (CRS Report R43984). Washington, DC: Congressional Research Service.

## 2.3 Royalties

A royalty is a payment to an owner for the use of property, especially patents, copyrighted works, franchises or natural resources. They work in the following way. The percentage agreed on the money made as a percentage gross or net incomes pending from the use of an asset this circumstance the Song.

Royalties are often spoken as a percentage of the [revenues](http://www.investopedia.com/terms/r/revenue.asp) gained using the owner's property, but they can be negotiated to encounter the specific needs of a plan. The use of royalties is common in situations where an inventor or original owner chooses to sell his product to a [third party](http://www.investopedia.com/terms/t/third-party.asp) in exchange for royalties from the future revenues it may generate.

## 2.3 Build Requirements

The tools used to create the new system (Database and graphical user interface) are mentioned and discussed below.

### 2.3.1 Java Language

Java is a general-purpose [computer programming language](https://en.wikipedia.org/wiki/Programming_language) that is [concurrent](https://en.wikipedia.org/wiki/Concurrent_computing), [class-based](https://en.wikipedia.org/wiki/Class-based_programming), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "[write once, run anywhere](https://en.wikipedia.org/wiki/Write_once,_run_anywhere)" (WORA), meaning that [compiled](https://en.wikipedia.org/wiki/Compiler) Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to [bytecode](https://en.wikipedia.org/wiki/Java_bytecode) that can run on any [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) regardless of [computer architecture](https://en.wikipedia.org/wiki/Computer_architecture).

### 2.3.2 MySQL Database

This is an [open-source](https://en.wikipedia.org/wiki/Open-source) [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS). Its name is a combination of "My", the name of co-founder [Michael Widenius](https://en.wikipedia.org/wiki/Michael_Widenius)' daughter, and "[SQL](https://en.wikipedia.org/wiki/SQL)", the abbreviation for [Structured Query Language](https://en.wikipedia.org/wiki/Structured_Query_Language). The MySQL development project has made its [source code](https://en.wikipedia.org/wiki/Source_code) available under the terms of the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License), as well as under a variety of [proprietary](https://en.wikipedia.org/wiki/Proprietary_software) agreements. MySQL was owned and sponsored by a single [for-profit](https://en.wikipedia.org/wiki/Business) firm, the [Swedish](https://en.wikipedia.org/wiki/Sweden) company [MySQL AB](https://en.wikipedia.org/wiki/MySQL_AB), now owned by [Oracle Corporation](https://en.wikipedia.org/wiki/Oracle_Corporation). For proprietary use, several paid editions are available, and offer additional functionality.

### 2.3.3 IntelliJ Integrated Development Environment (IDE)

IntelliJ is a [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) for developing computer software. It is developed by [JetBrains](https://en.wikipedia.org/wiki/JetBrains) (formerly known as IntelliJ), and is available as an [Apache 2 Licensed](https://en.wikipedia.org/wiki/Apache_2_License) community edition, and in a [proprietary](https://en.wikipedia.org/wiki/Proprietary_software) commercial edition. Both can be used for commercial development

### 2.3.4 StarUML

StarUML is a free source software tool which allows user to draw software modelling for the database and websites etc.

### 2.3.5 MySQL Workbench

My SQL Workbench is a visual [database that is used as a design](https://en.wikipedia.org/wiki/Database_design) tool that integrates [SQL](https://en.wikipedia.org/wiki/SQL) d[evelopment](https://en.wikipedia.org/wiki/Software_development), [administration](https://en.wikipedia.org/wiki/Database_administration_and_automation), [database design](https://en.wikipedia.org/wiki/Database_design), creation and maintenance into a single [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) for the [MySQL](https://en.wikipedia.org/wiki/MySQL) database system.

### 2.3.6 Java Database Connectivity

Java Database Connectivity (JDBC) is an [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (API) for the programming language [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), which defines how a client may access a [database](https://en.wikipedia.org/wiki/Database). It is part of the [Java Standard Edition](https://en.wikipedia.org/wiki/Java_Standard_Edition) platform, from [Oracle Corporation](https://en.wikipedia.org/wiki/Oracle_Corporation). It provides methods to query and update data in a database, and is oriented towards [relational databases](https://en.wikipedia.org/wiki/Relational_database). A JDBC-to-[ODBC](https://en.wikipedia.org/wiki/ODBC) bridge enables connections to any ODBC-accessible data source in the [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) host environment.

### 2.3.7 JavaFX

JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.

#### 2.3.7.1 JavaFX main features used

1- Java APIs. JavaFX is a Java library that consists of classes and interfaces that are written in Java code. The APIs are designed to be a friendly alternative to Java Virtual Machine (Java VM) languages, such as JRuby and Scala.

2- FXML and Scene Builder. FXML is an XML-based declarative mark-up language for constructing a JavaFX application user interface. A designer can code in FXML or use JavaFX Scene Builder to interactively design the graphical user interface (GUI). Scene Builder generates FXML mark-up that can be ported to an IDE where a developer can add the business logic.

### 2.3.8 XML Language

In [computing](https://en.wikipedia.org/wiki/Computing), Extensible Markup Language (XML) is a [mark-up language](https://en.wikipedia.org/wiki/Markup_language) that defines a set of rules for encoding [documents](https://en.wikipedia.org/wiki/Electronic_document) in a [format](https://en.wikipedia.org/wiki/File_format) that is both [human-readable](https://en.wikipedia.org/wiki/Human-readable_medium) and [machine-readable](https://en.wikipedia.org/wiki/Machine-readable_data). The [W3C](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium)'s XML 1.0 Specification and several other related specifications all of them free [open standards](https://en.wikipedia.org/wiki/Open_standard) define XML.

The design goals of XML emphasize simplicity, generality, and usability across the [Internet](https://en.wikipedia.org/wiki/Internet). It is a textual data format with strong support via [Unicode](https://en.wikipedia.org/wiki/Unicode) for different [human languages](https://en.wikipedia.org/wiki/Language). Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary [data structures](https://en.wikipedia.org/wiki/Data_structure) such as those used in [web services](https://en.wikipedia.org/wiki/Web_service).

## 2.4 License Agreement

The terms under which royalties are based on is called a license agreement. The license agreement defines the limits and restrictions of the royalties, such as its limitations pertaining to geographic territory, how long the agreement will last or the type of products with royalty cuts. License agreements are regulated specially if the resource owner is the government or if the license agreement is a private contract.

## 2.5 Royalty Rate

The royalty rate or the amount of royalty charged per product or service depends on the type of royalty fee for which a party is paying. Many factors affect the royalty rate. The most common ones include exclusivity of rights, availability of alternatives, risks involved, market demand structure, sustainability of technologies involved, and the level of innovation the product or service provides.

## 2.6 Copyright Infringement

Individuals and companies who develop new works and register for copyright protection do so in order to ensure that they can profit from their efforts. Other parties may be granted permission to use those works through licensing arrangements, or may purchase the works from the copyright holder. However, several factors may lead other parties to engage in copyright infringement. Reasons include a high price for the authorized work, or a lack of access to a supply of the authorized work.

## 2.7 Working Interests

Working interests, also referred to as operating interests, allow investors a percentage ownership of the drilling operation, functioning as a form of lease providing the investor a right to participate in drilling activities and a right to the resources produced from that activity.

Along with deriving an income from the production of the resource, the investors are also responsible for a percentage of the expenses related to its acquisition.

All investors within the arrangement select a well operator, who then also fills a role as a working interest. The well operator, after all operating expenses have been covered, divides any additional funds between those holding a working interest, creating a source of income. Those holding a working interest may deduct certain costs, such as those associated to depreciation of equipment, when determining what will amount to a form of self-employment income.

## 2.8 SWOT Analysis

SWOT is an [acronym](https://en.wikipedia.org/wiki/Acronym#Nomenclature) for strengths, weaknesses, opportunities, and threats and is a structured [planning](https://en.wikipedia.org/wiki/Plan) method that evaluates those four elements of an organization, [project](https://en.wikipedia.org/wiki/Project) or [business](https://en.wikipedia.org/wiki/Business) venture. A SWOT analysis can be carried out for a company, product, place, industry, or person. It involves specifying the objective of the business venture or project and identifying the internal and external factors that are favourable and unfavourable to achieve that objective. Some authors credit SWOT to [Albert Humphrey](https://en.wikipedia.org/wiki/Albert_S._Humphrey), who led a convention at the Stanford Research Institute (now [SRI International](https://en.wikipedia.org/wiki/SRI_International)) in the 1960s and 1970s using data from [Fortune 500](https://en.wikipedia.org/wiki/Fortune_500) companies. However, Humphrey himself did not claim the creation of SWOT, and the origins remain obscure. The degree to which the internal environment of the firm matches with the external environments.

### 2.8.1 SWOT Analysis breakdown

* Strengths: characteristics of the business or project that give it an advantage over others
* Weaknesses: characteristics of the business that place the business or project at a disadvantage relative to others
* Opportunities: elements in the environment that the business or project could exploit to its advantage
* Threats: elements in the environment that could cause trouble for the business or project

### 2.9 Model View Controller (MVC)

Model–view–controller (MVC) is a [software architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) for implementing [user interfaces](https://en.wikipedia.org/wiki/User_interface) on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The MVC design pattern decouples these main components permitting for efficient [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development.

Conventionally used for desktop [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUIs), this architecture has become popular for designing [web applications](https://en.wikipedia.org/wiki/Web_application) and even mobile, desktop and other clients. Popular programming languages like [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [PHP](https://en.wikipedia.org/wiki/PHP) and others have popular MVC frameworks that are currently being used in web application development straight [out of the box](https://en.wikipedia.org/wiki/Out_of_the_box_(feature)).

### 2.10 Database Management System (DBMS)

A database management system (DBMS) is system software for creating and managing [databases](http://searchsqlserver.techtarget.com/definition/database). The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage [data](http://searchdatamanagement.techtarget.com/definition/data).

A DBMS makes it possible for end users to create, read, update and delete [data](http://searchdatamanagement.techtarget.com/definition/data) in a database. The DBMS essentially serves as an interface between the [database](http://searchsqlserver.techtarget.com/definition/database) and end users or [application programs](http://searchsoftwarequality.techtarget.com/definition/application-program), ensuring that data is consistently organized and remains easily accessible.

The DBMS manages three important things: the data, the database [engine](http://whatis.techtarget.com/definition/engine) that allows data to be accessed, locked and modified -- and the database [schema](http://searchsqlserver.techtarget.com/definition/schema), which defines the database’s logical structure. These three foundational elements help provide [concurrency](http://searchoracle.techtarget.com/definition/concurrent-processing), security, [data integrity](http://searchdatacenter.techtarget.com/definition/integrity) and uniform administration procedures. Typical database administration tasks supported by the DBMS include [change management](http://searchcio.techtarget.com/definition/change-management), performance monitoring/tuning and [backup](http://searchstorage.techtarget.com/definition/backup) and [recovery](http://searchstorage.techtarget.com/definition/recovery). Many database management systems are also responsible for automated [rollbacks](http://searchsqlserver.techtarget.com/definition/rollback), restarts and recovery as well as the [logging](http://whatis.techtarget.com/definition/log-log-file) and [auditing](http://searchcio.techtarget.com/definition/audit-trail) of activity.

#### 2.10.1 Advantages of using DBMS

Using a DBMS to store and manage data comes with advantages, but also overhead. One of the biggest advantages of using a DBMS is that it lets end users and application programmers access and use the same data while managing data integrity. Data is better protected and maintained when it can be shared using a DBMS instead of creating new iterations of the same data stored in new files for every new application. The DBMS provides a central store of data that can be accessed by multiple users in a controlled manner.

* Central storage and management of data within the DBMS provides:
* Data abstraction and independence
* Data security
* A locking mechanism for concurrent access
* An efficient handler to balance the needs of multiple applications using the same data
* The ability to swiftly recover from crashes and errors, including restart ability and recoverability
* Robust data integrity capabilities
* Logging and auditing of activity
* Simple access using a standard application programming interface (API)
* Uniform administration procedures for data

Another advantage of a DBMS is that it can be used to impose a logical, structured organization on the data. A DBMS delivers economy of scale for processing large amounts of data because it is optimized for such operations.

A DBMS can also provide many views of a single database schema. A view defines what data the user sees and how that user sees the data. The DBMS provides a level of abstraction between the conceptual schema that defines the logical structure of the database and the physical schema that describes the files, indexes and other physical mechanisms used by the database. When a DBMS is used, systems can be modified much more easily when business requirements change. New categories of data can be added to the database without disrupting the existing system and applications can be insulated from how data is structured and stored.

### 2.11 Use case diagram (UML)

A use case diagram is a graphic depiction of the interactions among the elements of a system.

A [use case](http://searchsoftwarequality.techtarget.com/definition/use-case) is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to something being developed or operated, such as a mail-order product sales and service [Web site](http://searchsoa.techtarget.com/definition/Web-site). Use case diagrams are employed in [UML](http://searchsoftwarequality.techtarget.com/definition/Unified-Modeling-Language) (Unified Modelling Language), a standard notation for the modelling of real-world objects and systems.

System objectives can include planning overall requirements, validating a [hardware](http://searchcio-midmarket.techtarget.com/definition/hardware) design, testing and [debugging](http://searchsoftwarequality.techtarget.com/definition/debugging) a [software](http://searchsoa.techtarget.com/definition/software) product under development, creating an online help reference, or performing a consumer-service-oriented task. For example, use cases in a product sales environment would include item ordering, catalog updating, payment processing, and customer relations. A use case diagram contains four components.

#### 2.11.1 Advantages of Use Case Diagram

* The boundary, which defines the system of interest in relation to the world around it.
* The actors, usually individuals involved with the system defined according to their roles.
* The use cases, which are the specific roles played by the actors within and around the system.
* The relationships between and among the actors and the use cases.

### 

# Chapter 3: Analysis

### 3.1 Introduction

This project aims to reduce the time this company takes to prepare and do the distribution and to correctly create a custom validated report for each artist/producer. Currently the problem they have showed is that, some of the reports come late as there are a lot of songs to process and the records are constantly changing because every day these songs are played, there are errors made by the company’s workers i.e. Some artist/producers don’t get paid properly, calculating the total amount of time played wrong, etc. This is all done in excel sheets so some Artists might get repeated by accident

The solution this project offers is an interface that manages a database that contains all the information contained of all the records the company needs. This interface will offer some essential functionality.

More functionality will come as the software develops, an idea for this would be store artist/producer emails and as soon as the process is finished they will receive their report of how much they have earned in royalties.

### 3.2 SWOT Analysis

SWOT analysis was conducted to identify key features which could be incorporated into the creation of this system that will help the company improve its performance.

### 3.2.1 SWOT Analysis before build

This SWOT analysis was carried out so a better understanding of what features and improvements need to be integrated in the new system.

### 3.3 Functional and Non-Functional Requirements

After identifying what features need to be added using the SWOT analysis, a list of functional and non-functional requirements was created as considered crucial to the development of the new system that will be incorporated.

|  |  |
| --- | --- |
| **Strengths**   * Keeps records save in office computers * Excel tools availed for use | **Weaknesses**   * Staff might need training to use excel * Interface for so much data is not friendly * No way of filtering information * Limited functionality |
| **Opportunities**   * Creating a new system that can be used to manage databases * Allow company to keep a more manageable record keeping framework * Information is secured as is only available at user’s computer and where backups are made * Improve speed of distribution process | **Threats**   * Company will not develop as consequence of not adapting to new technologies * Increase the amount of changes in the projects once feedback is given * Company will struggle in record keeping and fixing mistakes |

### 3.3.1 Functional Requirements

1. Store information
2. Edit information
3. Produce Report
4. Search artist information
5. Calculate total
6. Validate entries
7. Data processing
8. Create connection to interface

### 3.3.2 Non-Functional Requirements

1. Maintainability of the System
2. Response time of the system
3. Storage capacity
4. Cost
5. User friendly interface
6. Extendibility of the system
7. Security

### 3.4 MoSCoW Analysis

It is a very valuable technique that this project uses as a tool to manage the resources and time to be focused into the most important aspects of it i.e. main features. At the same time being consulted with the stakeholders

MoSCoW analysis divides its requirements into four categories: Must, Should, Could and Wont. A brief description of the categories are as follows.

* Must: Describes a requirement that must be satisfied in the final solution for the solution to be considered a success.
* Should: Represents a requirement a high-priority item that should be included in the solution if it is possible. This is often a critical requirement but one which can be satisfied in other ways if strictly necessary.
* Could: Describes a requirement, which is considered desirable but not necessary. This will be included if time and resources permit.
* Won’t: Represents a requirement that stakeholders have agreed will not be implemented in a given release, but may be considered in the future.

### 3.4.1 Functional requirements in MoSCoW

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement ID | Function (Verb noun) | Requirements | Prioritisation |
| 1 | Create Artist | User can create artist record | MUST HAVE |
| 2 | Delete Artist | User can delete artist record | MUST HAVE |
| 3 | Update Artist | User can update artist record | MUST HAVE |
| 4 | Validate Artist | System confirms with user to add record | SHOULD HAVE |
| 5 | Data Processing | Create a csv file with the records | SHOULD HAVE |
| 6 | Search Artist | Search artist by either filtering, by ID, name or type | MUST HAVE |
| 7 | Produce Report | Create a report for artist so they are able to see how much money they made per song | MUST HAVE |
| 8 | Create Song | User can create a song record | MUST HAVE |
| 9 | Delete Song | User can delete a song record | MUST HAVE |
| 10 | Update song | User can update a song record | COULD HAVE |
| 11 | Assign song | User can link song to Artists | MUST HAVE |
| 12 | Calculate Royalty | A calculation of the royalties for each Artist are made by the program | MUST HAVE |
| 13 | Search Song | Search a particular song record | COULD HAVE |
| 14 | Display result | User is able to see if the action has been successful | COULD HAVE |

### 3.5 Use Case Diagram

This use case diagram shows what functions should the user be able to do and what information does the database has access to.

### Figure 3.5.1



### 3.6 Risk Analysis

Any risks are identified here. These risks were considered as they might occur during the use of the new system (see figure 3.4.1)]

### Figure 3.5.1 – Risk Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Risk number | Risk name | Impact | Action |
| 1 | Data loss from database | Medium | A backup of the system regularly is required |
| 2 | System bugs | Low | Providing a manual for the user and a contact number of the system creator |
| 3 | Fire hazards | High | Must ensure system to not be over heating and follow manufacture advice and keep a record and monitoring the system regularly |
| 4 | New Updates or change the system | Low | Ensure to have up to date back up also make sure that the new updates are compatible with the existing system |

### 3.6 Interview done with the customer

There were two main interviews done with the client. These were done at really early stages of the project. The main functional requirements are taken from these interviews

**Interview 1**

Q: How long has the current system has been used?

A: It has been used since 2004

Q: How long does it take to finish the process called Distribution?

A: 5 to 6 weeks

Q: How you calculate how much money each artist gets?

A: depending on the number of times their song played compared to the total number of times all the other songs got played. This depends whether they are national or international songs.

Q: What is your current system?

A: We use Microsoft SQL to store all the records needed. Then used a software that calculates the amount of money to be given to each artist.

Q: How many artist does your database have?

A: A maximum of 250 000 artists they can be either national or international belong to a group or be by themselves. At the moment we stored international artists in a different database from the national artists.

Q: Is there a problem with the current system?

A: At the moment the System takes a long time to produce the reports needed to give out to the artists and producers.

Q: What do you need your system to do?

A: we keep information(new artists/producer, this months amount of time a song was played) in excel spreadsheets and the employees then proceed to introduce the information in the database two months before the end of a semester, so then this information is used to produce reports that will show the artists and producers how much money they are getting and why.

**Interview 2**

This interview is written is in Spanish as during the interview there was a lot of information to take in a rough translation has been done.

El total de dinero recaudado se divide en la cantidad total de toques o sonadas eso da el valor del toque la obra ganará o recibirá por la cantidad de toque multiplicado por el el valor determinado después se aplican los porcentajes

El monto que recibe la obra por el valor del toque se divide en 50% para productores y 50% para artistas y ejecutantes . Del 50 % de artistas y ejecutantes 66.65 intérpretes 33.35 ejecutantes. This percentages are stated by the country’s laws.

Distribution formula

Producers get= totalAmountPerSong/2

Main artists get = ((totalAmountPerSong/2)\*66.65)/totalNumberOfMainArtists

Secondary artists get= ((totalAmountPerSong/2)\*33.35)/totalNumberOfSecondaryArtists

# Chapter 4: Design

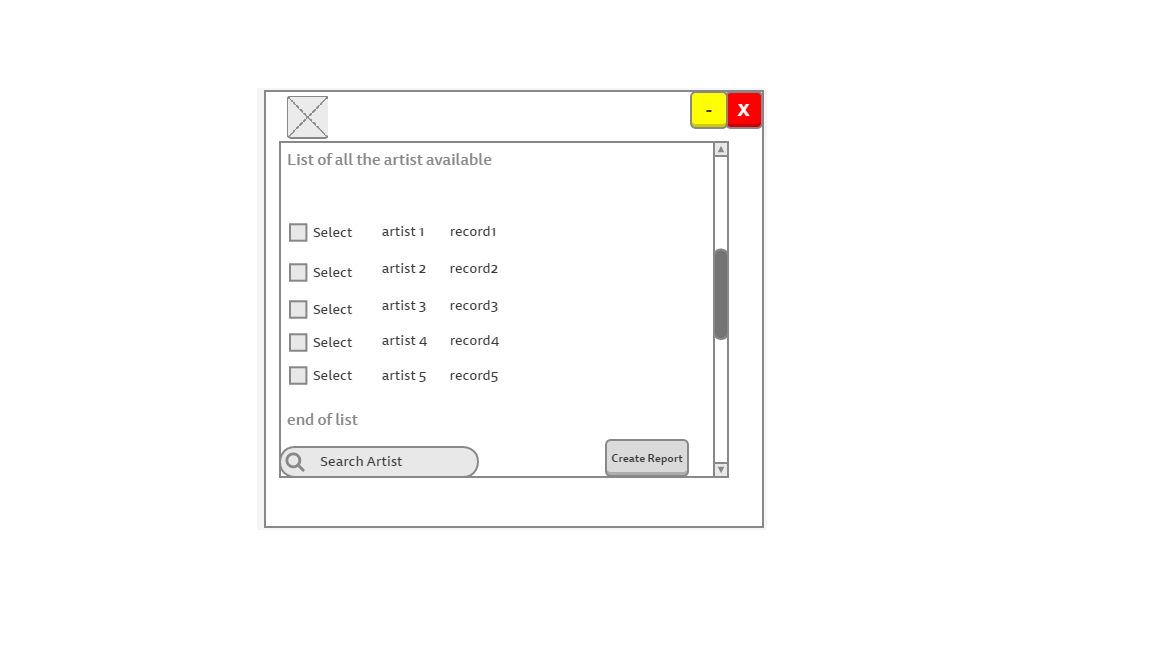
## 4.1 Introduction

The first time interviewing the user, a simple user interface was agreed upon where he could see all the records easy to pick up and self-explanatory. The only person that requires access to it is himself so only him will be using this program. A few wire frames will be presented showing the design of the application. This will be subject of change depending on the user feedback after the implementation is done.

## 4.2 Wireframes

Initial design of the user interface he wanted the list of artist and be able to search for them. This will be the artists view where the user will look at all the items that belongs to an artist their ID, name. other details. After one of the interviews the user said that the only one with access to this software is going to be him as he is the person that closes up the process of the distribution

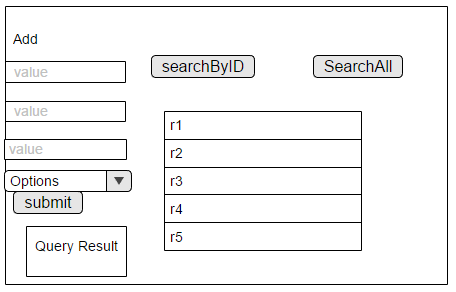
## Figure 4.2.1 Initial Design



The song’s view will be also similar to the artist’s view as the principle is the same show the information about the songs so they can be edit or be added if there are new songs functionality will be added so totals are calculated by the program.

As an extra feature a query result box was added so the user can see after doing an action the program will provide a visible output telling the user about the action that just took place. For example if user deletes a record the box will display record deleted.

## Figure 4.2.2 Second Design



## 4.3 Database Design – Class Diagram

The database diagram was created with the purpose of allowing the creation of a conceptual model of the system, to ensure that all the key areas are identified and added to the system. Another reason is that it reduces the possibility of making errors and saves time as all the troubleshooting and planning is taken care prior the build of the system.

The following diagram (figure 4.3.1) takes in consideration the requirements taken from the interview done with the user and it was created using StarUML.

## Figure 4.3.1



# Chapter 5: Implementation

## 5.1 Introduction

In this part of the project the steps taken to create the system (database and graphical user interface) will be explained and how to connect them both.

## 5.2 Methodology Used

The approach taken was the water fall methodology which consist on dividing the project in the following steps:

**Analysis**: in this step, all the requirements needed by the system are captured and documented.

**Design**: all the requirement taken from the analysis stage are taken in consideration and is here that the design of the system takes place.

**Implementation**: with the information from the system design step, developing starts and small pieces of code (programs) are developed and later tested in the next step.

**Testing**: all the pieces of codes developed in the design implementation stage are tested here and integrated into the system if successful.

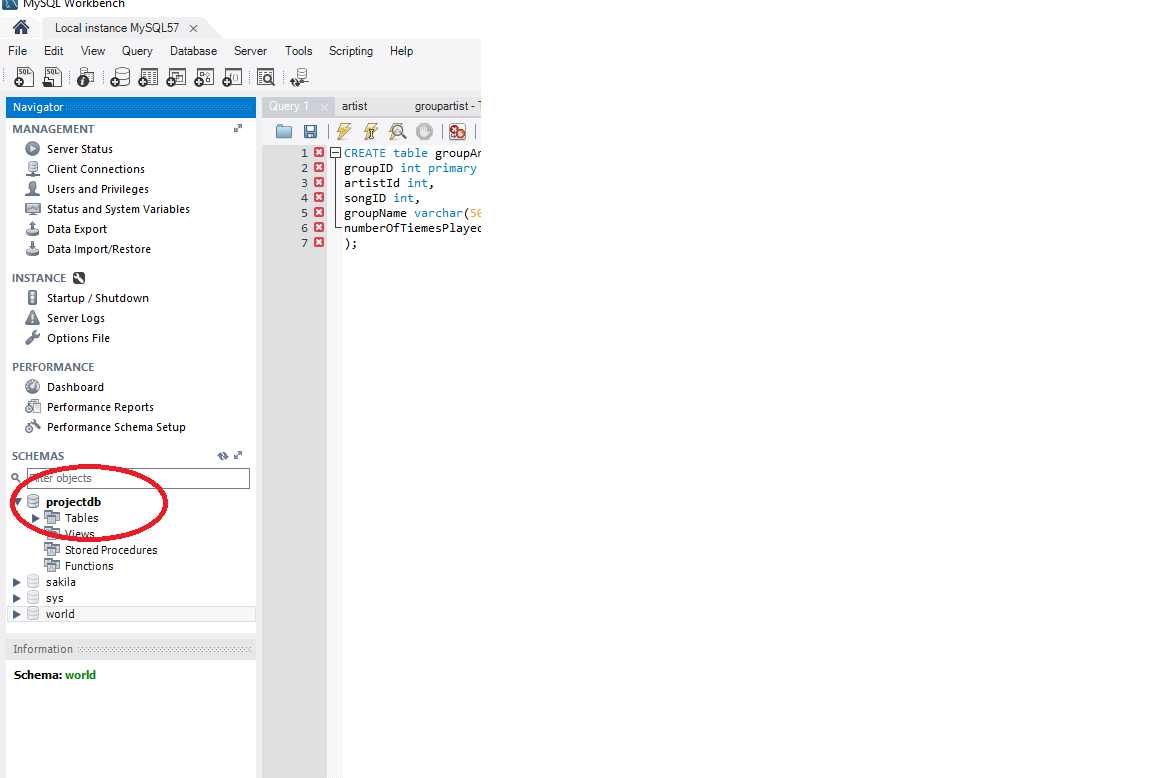
**Deployment**: once the functional and non-functional requirements are tested the system is deployed in the customer environment or released to the market.

**Maintenance**: if there are issues once the system is developed. Patches must be released to fix those issues

## 5.2 Database creation

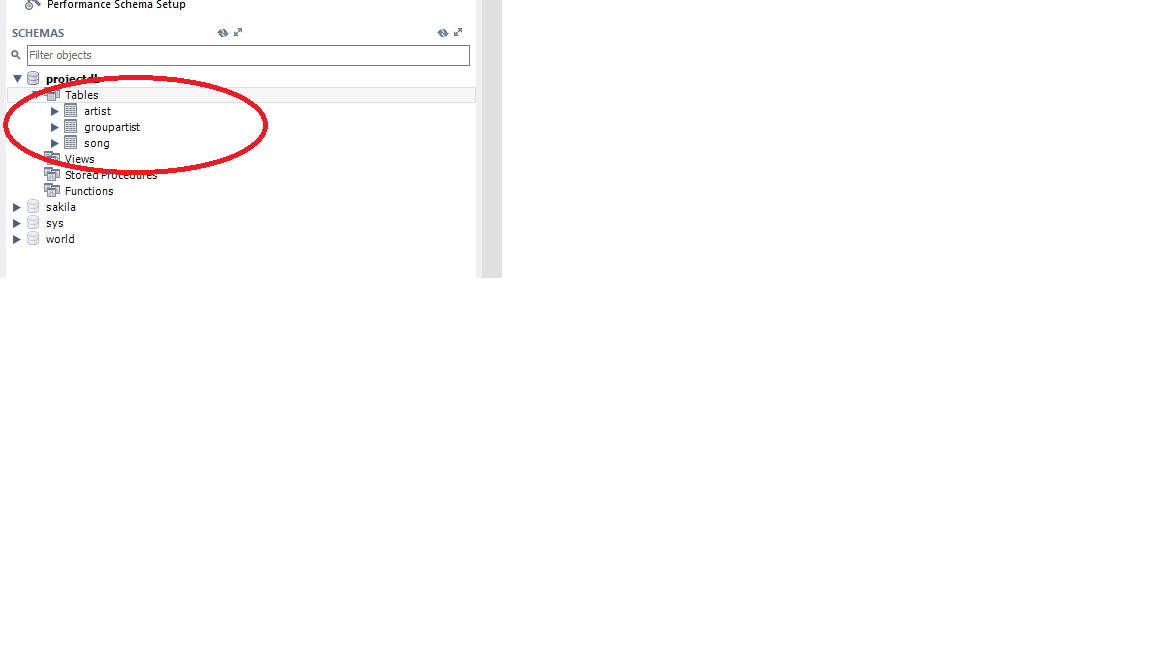
1. The first step is to install the software MySQL Workbench this is the tool used to create the database.
2. Now proceed to create a database to store all the information required by the company each entity needs to have a table for example artists, songs. In this case the database is called projectdb.

## Figure 5.2.1 Creating database



1. Now the tables necessaries to store the information about the artists, songs and the distribution.

## Figure 5.2.2 Creating tables to store information



1. The next step is to create attributes for each table in the database. This attributes are the ones identified in the class diagram (Figure 4.3.1).

## Figure 5.2.3 Creating attributes for table Artist SQL.

CREATE table artist(

artistId int primary key,

artistName varchar(50),

gender enum('Male', 'Female'),

dateOfBirth date,

typeOfArtist enum('National', 'International')

)

## Figure 5.2.4 Creating attributes for table Song SQL.

CREATE table song(

songID int primary key,

songName varchar(50),

album varchar(50),

dateCreated date

);

## Figure 5.2.5 Creating attributes for table groupArtist SQL.

CREATE table groupArtist(

groupID int primary key,

artistId int,

songID int,

groupName varchar(50),

numberOfTiemesPlayed int

);

ALTER TABLE `projectdb`.`groupartist`

ADD CONSTRAINT `artistId`

FOREIGN KEY (`artistId`)

REFERENCES `projectdb`.`artist` (`artistId`)

ON DELETE CASCADE

ON UPDATE CASCADE,

ADD CONSTRAINT `songId`

FOREIGN KEY (`songID`)

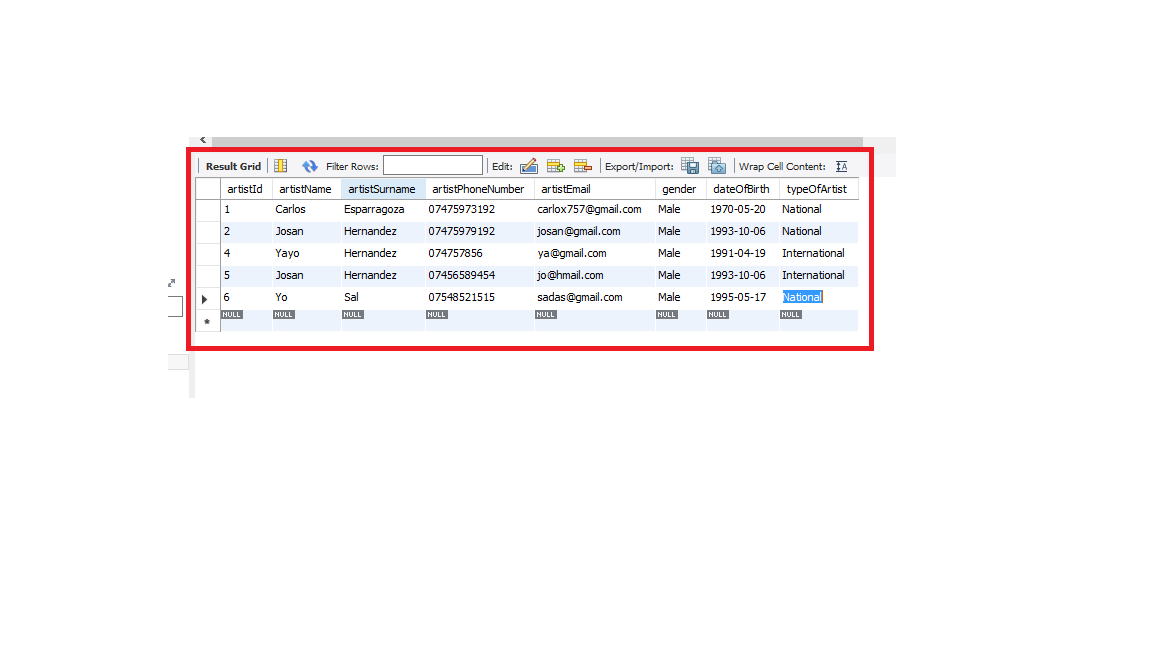
REFERENCES `projectdb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE;

This bit is to add the constraint on the groupArtist table so it knows where to get the foreign keys

### Figure 5.2.6 Inserted values for attributes artist table



Example SQL of inserting data

INSERT INTO `projectdb`.`artist` (`artistName`, `artistSurname`, `artistPhoneNumber`, `artistEmail`, `gender`, `dateOfBirth`, `typeOfArtist`) VALUES ('Yo', 'Sal', '07548521515', 'sadas@gmail.com', 'Male', '1995/05/17', 'National');

### 5.3 Creation of the graphical user interface

In here the code used to create the interface will be explained. The approach used for coding was MVC which stands for Model, View, Controller. This approach was used to separate concerns for example functionality of the program from data access the program requires from the database. And extra folder was added to add all the tools needed to connect to database and run queries this folder is call util. The way the program will run is an artefact (executable object) will be produced that can run in a machine that has Java installed in it.

### 5.3.1 Root Layout controller

This is the class in charge of dealing with the base view it contains the menu which the user for basic operations such as close the application or change views.

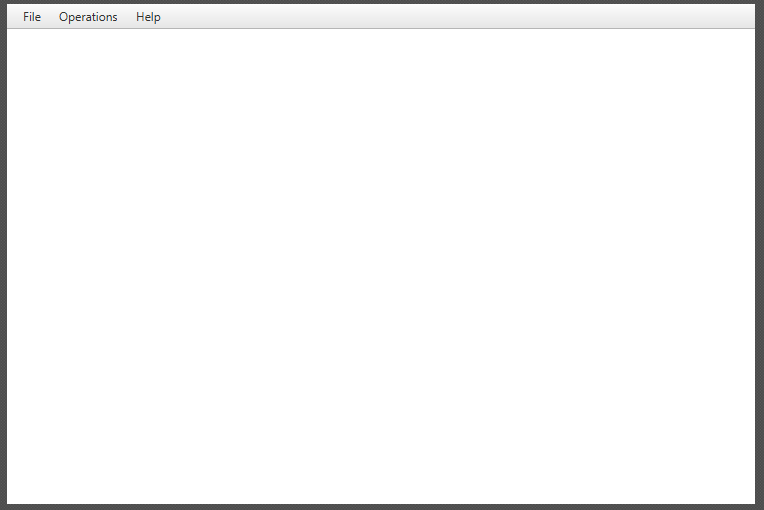
package sample.controller;  
  
*/\*\*  
 \* Created by Carlito on 12/04/2017.  
 \*/*import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Alert;  
import javafx.scene.control.MenuBar;  
import javafx.scene.layout.AnchorPane;  
import javafx.stage.Stage;  
import sample.Main;  
  
import java.io.IOException;  
  
public class RootLayoutController {  
  
 //Reference to the main application  
 private Main main;  
  
 //Is called by the main application to give a reference back to itself.  
 public void setMain (Main main) {  
 this.main = main;  
 }  
  
 @FXML  
 private MenuBar menuBar ;  
  
 //Exit the program  
 public void handleExit(ActionEvent actionEvent) {  
 System.*exit*(0);  
 }  
  
 //Help Menu button behavior  
 public void handleHelp(ActionEvent actionEvent) {  
 Alert alert = new Alert (Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Program Information");  
 alert.setHeaderText("This is a university prototype!");  
 alert.setContentText("You can search, delete, update, insert a new artists with this program.");  
 alert.show();  
 }  
  
 public void handleArtist(ActionEvent actionEvent)throws IOException {  
 main.showArtistView();  
  
 }  
  
 public void handleSong(ActionEvent actionEvent)throws IOException{  
 main.showSongView();  
   
 }  
}

### 5.3.2 Root layout View

This is the view for the root layout where the main menu is found at the top. This is the main area of the program where the user can navigate through the rest of the application (imports have been omitted available at original code).

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.control.Menu?>  
<?import javafx.scene.control.MenuBar?>  
<?import javafx.scene.control.MenuItem?>  
<?import javafx.scene.layout.BorderPane?>  
  
<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="500.0" prefWidth="748.0" xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.controller.RootLayoutController">  
 <top>  
 <MenuBar fx:id="menuBar" prefWidth="748.0" BorderPane.alignment="CENTER">  
 <menus>  
 <Menu mnemonicParsing="false" text="File">  
 <items>  
 <MenuItem mnemonicParsing="false" onAction="#handleExit" text="Close" />  
 </items>  
 </Menu>  
 <Menu mnemonicParsing="false" text="Operations">  
 <items>  
 <MenuItem mnemonicParsing="false" text="Artist" onAction="#handleArtist"/>  
 <MenuItem mnemonicParsing="false" text="Song" onAction="#handleSong"/>  
 </items>  
 </Menu>  
 <Menu mnemonicParsing="false" text="Help">  
 <items>  
 <MenuItem mnemonicParsing="false" onAction="#handleHelp" text="About" />  
 </items>  
 </Menu>  
 </menus>  
 </MenuBar>  
 </top>  
</BorderPane>

### Figure 5.3.2.1



### 5.3.3 Artist Controller class

This class is the one in charge of dealing with all the operations to do with artistView. The artist controller class is the interface between Model and View. There are some comments in this code that explains each part of the code (imports have been omitted available at original code).

*/\*\*  
 \* Created by Carlito on 12/04/2017.  
 \*/*public class ArtistController {  
  
 @FXML  
 private TextField artIdText;  
 @FXML  
 private TextField surnameText;  
 @FXML  
 private ChoiceBox genderChoice;  
 @FXML  
 private ChoiceBox typeChoice;  
 @FXML  
 private TextField emailText;  
 @FXML  
 private TextField phoneText;  
 @FXML  
 private DatePicker dob;  
 @FXML  
 private TextField nameText;  
 @FXML  
 private ChoiceBox gender;  
 @FXML  
 private ChoiceBox type;  
 @FXML  
 private TextField idToDelete;  
 @FXML  
 private TextArea resultArea;  
 @FXML  
 private TableView artistTable;  
 @FXML  
 private TableColumn<Artist, Integer> artistIdColumn;  
 @FXML  
 private TableColumn<Artist, String> artistNameColumn;  
 @FXML  
 private TableColumn<Artist, String> artistLastNameColumn;  
 @FXML  
 private TableColumn<Artist, String> artistEmailColumn;  
 @FXML  
 private TableColumn<Artist, String> artistPhoneNumberColumn;  
 @FXML  
 private TableColumn<Artist, String> artistGenderColumn;  
 @FXML  
 private TableColumn<Artist, Date> artistDOBDateColumn;  
  
  
 ObservableList<String> artistTypeList = FXCollections.*observableArrayList*("National", "International");  
  
 ObservableList<String> artistGenderList = FXCollections.*observableArrayList*("Male", "Female");  
  
 ObservableList<String> genderFilter = FXCollections.*observableArrayList*("None", "Male", "Female");  
  
 ObservableList<String> typeFilter = FXCollections.*observableArrayList*("None", "National", "International");  
  
 //Search an Artist  
 @FXML  
 private void searchArtist (ActionEvent actionEvent) throws ClassNotFoundException, SQLException {  
 try {  
 //Get Artist information  
 Artist art = ArtistDao.*searchArtist*(artIdText.getText());  
 //Populate Artist on TableView and Display on TextArea  
 populateAndShowArtist(art);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 resultArea.setText("Error occurred while getting Artist information from DB.\n" + e);  
 throw e;  
 }  
 }  
  
 //Filter Artist Table  
  
 @FXML  
 private void filterTable (ActionEvent actionEvent) throws ClassNotFoundException, SQLException {  
 try {  
 //Get Artist information  
 if (genderChoice.getValue().toString().equalsIgnoreCase("none") && typeChoice.getValue().toString().equalsIgnoreCase("none")){  
 //Get all Artist information  
 ObservableList<Artist> artistData = ArtistDao.*searchArtists*();  
 //Populate Artist on TableView  
 populateAllArtists(artistData);  
 }  
 if (!genderChoice.getValue().toString().equalsIgnoreCase("none")){  
 typeChoice.getSelectionModel().selectFirst();  
 ObservableList<Artist> art = ArtistDao.*searchArtistByGender*(genderChoice.getValue().toString());  
 //Populate Artist on TableView and Display on TextArea  
 populateAllArtists(art);  
 genderChoice.getSelectionModel().selectFirst();  
 }  
 if(!typeChoice.getValue().toString().equalsIgnoreCase("none")){  
 genderChoice.getSelectionModel().selectFirst();  
 ObservableList<Artist> art = ArtistDao.*searchArtistByType*(typeChoice.getValue().toString());  
 //Populate Artist on TableView and Display on TextArea  
 populateAllArtists(art);  
 typeChoice.getSelectionModel().selectFirst();  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 resultArea.setText("Error occurred while getting Artist information from DB.\n" + e);  
 throw e;  
 }  
 }  
  
 //Search all Artists  
 @FXML  
 private void searchArtists(ActionEvent actionEvent) throws SQLException, ClassNotFoundException {  
 try {  
 //Get all Artist information  
 ObservableList<Artist> artistData = ArtistDao.*searchArtists*();  
 //Populate Artist on TableView  
 populateAllArtists(artistData);  
 } catch (SQLException e){  
 System.*out*.println("Error occurred while getting Artist information from DB.\n" + e);  
 throw e;  
 }  
 }  
  
 //Initializing the controller class.  
 //This method is automatically called after the fxml file has been loaded.  
 @FXML  
 private void initialize () {  
 /\*  
 The setCellValueFactory(...) that we set on the table columns are used to determine  
 which field inside the Artist objects should be used for the particular column.  
  
 \*/  
 genderChoice.setItems(genderFilter);  
 typeChoice.setItems(typeFilter);  
 gender.setItems(artistGenderList);  
 type.setItems(artistTypeList);  
 genderChoice.getSelectionModel().selectFirst();  
 typeChoice.getSelectionModel().selectFirst();  
 artistIdColumn.setCellValueFactory(cellData -> cellData.getValue().artistIDProperty().asObject());  
 artistNameColumn.setCellValueFactory(cellData -> cellData.getValue().artistNameProperty());  
 artistLastNameColumn.setCellValueFactory(cellData -> cellData.getValue().artistSurnameProperty());  
 artistEmailColumn.setCellValueFactory(cellData -> cellData.getValue().artistEmailProperty());  
 artistPhoneNumberColumn.setCellValueFactory(cellData -> cellData.getValue().artistPhoneNumberProperty());  
 artistGenderColumn.setCellValueFactory(cellData -> cellData.getValue().artistGenderProperty());  
 artistDOBDateColumn.setCellValueFactory(cellData -> cellData.getValue().artistDOBProperty());  
  
 }  
  
 //Populate Artist  
 @FXML  
 private void populateArtist (Artist art) throws ClassNotFoundException {  
 //Declare and ObservableList for table view  
 ObservableList<Artist> artistData = FXCollections.*observableArrayList*();  
 //Add artist to the ObservableList  
 artistData.add(art);  
 //Set items to the artistTable  
 artistTable.setItems(artistData);  
 }  
  
 //Set Artist information to Text Area  
 @FXML  
 private void setArtistInfoToTextArea ( Artist artist) {  
 resultArea.setText("First Name: " + artist.getArtistName() + "\n" +  
 "Surname: " + artist.getArtistSurname());  
 }  
  
 //Populate Artist for TableView and Display Artist on TextArea  
 @FXML  
 private void populateAndShowArtist(Artist artist) throws ClassNotFoundException {  
 if (artist != null) {  
 populateArtist(artist);  
 setArtistInfoToTextArea(artist);  
 } else {  
 resultArea.setText("This Artist does not exist!\n");  
 }  
 }  
  
 //Populate Artist for TableView  
 @FXML  
 private void populateAllArtists (ObservableList<Artist> artistData) throws ClassNotFoundException {  
 //Set items to the artistTable  
 artistTable.setItems(artistData);  
 }  
  
  
 //Insert an artist to the DB  
 @FXML  
 private void createArtistRecord (ActionEvent actionEvent) throws SQLException, ClassNotFoundException {  
 try {  
 java.sql.Date sqlDate = java.sql.Date.*valueOf*(dob.getValue());  
 System.*out*.println(nameText.getText());  
 System.*out*.println(gender.getValue().toString());  
 ArtistDao.*insertArtist*(nameText.getText(),surnameText.getText(), phoneText.getText(),emailText.getText(), gender.getValue().toString(), sqlDate,type.getValue().toString());  
 resultArea.setText("Artist inserted! \n");  
 } catch (SQLException e) {  
 resultArea.setText("Problem occurred while inserting artist " + e);  
 throw e;  
 }  
 }  
  
 //Delete an employee with a given employee Id from DB  
 @FXML  
 private void deleteArtist (ActionEvent actionEvent) throws SQLException, ClassNotFoundException {  
 try {  
 ArtistDao.*deleteArtistByID*(idToDelete.getText());  
 resultArea.setText("Artist deleted! Artist id: " + idToDelete.getText() + "\n");  
 } catch (SQLException e){  
 resultArea.setText("Problem occurred while deleting artist " + e);  
 throw e;  
 }  
 }  
  
}

### 5.3.4 Artist class

This will represent a schema to represent each artist for example an artist has a name, surname, and address. This will be the base to create each unique artist so it can be added or pulled from the database (imports have been omitted available at original code).

*/\*\*  
 \* Created by Carlito on 12/04/2017.  
 \*/*public class Artist {  
   
 private IntegerProperty artistID;  
 private StringProperty artistName;  
 private StringProperty artistSurname;  
 private StringProperty artistPhoneNumber;  
 private StringProperty artistEmail;  
 private StringProperty gender;  
 private SimpleObjectProperty<Date> dateOfBirth;  
 private StringProperty typeOfArtist;  
  
  
  
  
 public Artist() {  
 this.artistID = new SimpleIntegerProperty();  
 this.artistName = new SimpleStringProperty();  
 this.artistSurname = new SimpleStringProperty();  
 this.artistPhoneNumber = new SimpleStringProperty();  
 this.artistEmail = new SimpleStringProperty();  
 this.gender = new SimpleStringProperty();  
 this.dateOfBirth = new SimpleObjectProperty<>();  
 this.typeOfArtist = new SimpleStringProperty();  
 }  
  
  
   
 public int getArtistID() {  
 return artistID.get();  
 }  
  
 public void setArtistID(int artist\_ID) {  
 this.artistID.set(artist\_ID);  
 }  
  
 public IntegerProperty artistIDProperty(){  
 return artistID;  
 }  
   
   
 public String getArtistSurname() {  
 return artistSurname.get();  
 }  
  
 public void setArtistSurname(String artistSurname) {  
 this.artistSurname.set(artistSurname);  
 }  
  
 public StringProperty artistSurnameProperty(){  
 return artistSurname;  
 }  
  
  
 public String getArtistPhoneNumber() {  
 return artistPhoneNumber.get();  
 }  
  
 public void setArtistPhoneNumber(String artistPhoneNumber) {  
 this.artistPhoneNumber.set(artistPhoneNumber);  
 }  
  
 public StringProperty artistPhoneNumberProperty(){  
 return artistPhoneNumber;  
 }  
  
  
  
 public String getArtistEmail() {  
 return artistEmail.get();  
 }  
  
 public void setArtistEmail(String artistEmail) {  
 this.artistEmail.set(artistEmail);  
 }  
  
 public StringProperty artistEmailProperty(){  
 return artistEmail;  
 }  
  
  
   
   
 public String getArtistName() {  
 return artistName.get();  
 }  
  
 public void setArtistName(String artistName) {  
 this.artistName.set(artistName);  
 }  
  
 public StringProperty artistNameProperty(){  
 return artistName;  
 }  
  
   
  
 public Object getDateOfBirth() {  
 return dateOfBirth.get();  
 }  
  
 public void setDateOfBirth(Date dateOfBirth) {  
 this.dateOfBirth.set(dateOfBirth);  
 }  
  
 public SimpleObjectProperty<Date> artistDOBProperty(){  
 return dateOfBirth;  
 }  
  
   
   
 public String getGender() {  
 return gender.get();  
 }  
  
 public void setGender(String gender) {  
 this.gender.set(gender);  
 }  
  
 public StringProperty artistGenderProperty(){  
 return gender;  
 }  
   
   
   
 public String getTypeOfArtist() {  
 return typeOfArtist.get();  
 }  
  
 public void setTypeOfArtist(String typeOfArtist) {  
 this.typeOfArtist.set(typeOfArtist);  
 }  
  
 public StringProperty artistTypeProperty(){  
 return typeOfArtist;  
 }  
  
  
}

### 5.3.5 ArtistDao class

This is the class in charge of getting the information from the database regarding an artist by using queries and the DBUtil class (imports have been omitted available at original code).

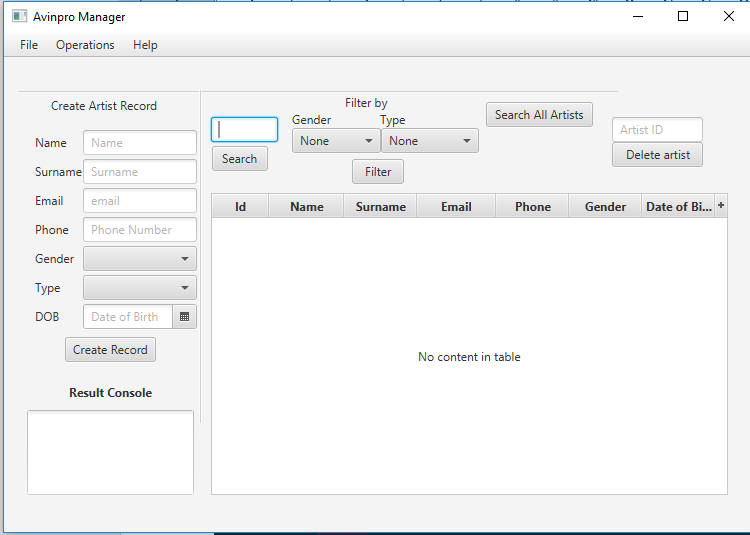
*/\*\*  
 \* Created by Carlito on 12/04/2017.  
 \*/*public class ArtistDao {  
  
 public static Artist searchArtist(String artID) throws SQLException, ClassNotFoundException {  
 //Declare a SELECT statement  
 String selectStmt = "SELECT \* FROM artist WHERE artistId= "+artID;  
  
 //Execute SELECT statement  
 try {  
 //Get ResultSet from dbExecuteQuery method  
 ResultSet rsArt = DBUtil.*dbExecuteQuery*(selectStmt);  
  
 //Send ResultSet to the getArtist from resultset method and get Artist object  
 Artist artist = *getArtistFromResultSet*(rsArt);  
  
  
 //Return Artist object  
 return artist;  
 } catch (SQLException e) {  
 System.*out*.println("While searching an Artist with " + artID + " id, an error occurred: " + e);  
 //Return exception  
 throw e;  
 }  
 }  
  
 private static Artist getArtistFromResultSet(ResultSet rs) throws SQLException  
 {  
 Artist artist = null;  
 if (rs.next()) {  
 artist = new Artist();  
 artist.setArtistID(rs.getInt("artistId"));  
 artist.setArtistName(rs.getString("artistName"));  
 artist.setArtistSurname(rs.getString("artistSurname"));  
 artist.setArtistEmail(rs.getString("artistEmail"));  
 artist.setTypeOfArtist(rs.getString("typeOfArtist"));  
 artist.setArtistPhoneNumber(rs.getString("artistPhoneNumber"));  
 artist.setDateOfBirth(rs.getDate("dateOfBirth"));  
 artist.setGender(rs.getString("gender"));  
  
 }  
 return artist;  
 }  
  
  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 //SELECT All ARTISTS  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 public static ObservableList<Artist> searchArtists() throws SQLException, ClassNotFoundException {  
 //Declare a SELECT statement  
 String selectStmt = "SELECT \* FROM artist";  
  
 //Execute SELECT statement  
 try {  
 //Get ResultSet from dbExecuteQuery method  
 ResultSet rsArts = DBUtil.*dbExecuteQuery*(selectStmt);  
  
  
 //Send ResultSet to the getArtistsList method and get artist object  
 ObservableList<Artist> artistList = *getArtistsList*(rsArts);  
  
 //Return Artist object  
 return artistList;  
  
 } catch (SQLException e) {  
 System.*out*.println("SQL select operation has been failed: " + e);  
 //Return exception  
 throw e;  
 }  
 }  
  
 //Select \* from Artists operation  
 private static ObservableList<Artist> getArtistsList(ResultSet rs) throws SQLException, ClassNotFoundException {  
 //Declare a observable List which comprises of Artists objects  
 ObservableList<Artist> artistsList = FXCollections.*observableArrayList*();  
  
 while (rs.next()) {  
 Artist artist = new Artist();  
 artist.setArtistID(rs.getInt("artistId"));  
 artist.setArtistName(rs.getString("artistName"));  
 artist.setArtistSurname(rs.getString("artistSurname"));  
 artist.setArtistEmail(rs.getString("artistEmail"));  
 artist.setTypeOfArtist(rs.getString("typeOfArtist"));  
 artist.setArtistPhoneNumber(rs.getString("artistPhoneNumber"));  
 artist.setDateOfBirth(rs.getDate("dateOfBirth"));  
 artist.setGender(rs.getString("gender"));  
 //Add Artist to the ObservableList  
 artistsList.add(artist);  
  
 }  
  
 //return Artist list (ObservableList of artists)  
 return artistsList;  
 }  
  
 public static void insertArtist (String name, String surname, String phone,String email, String gender, Date dob , String type) throws SQLException, ClassNotFoundException {  
 //Declare a Update statement  
 String updateStmt = "INSERT INTO artist(artistName, artistSurname, artistPhoneNumber, artistEmail, gender, dateOfirth,typeOfArtist) VALUES('"+name+"', '"+surname+"','"+phone+"' ,'"+email+"' , '"+gender+"', '"+dob+"','"+type+"')";  
  
  
 //Execute Update operation  
 try {  
 DBUtil.*dbExecuteUpdate*(updateStmt);  
 } catch (SQLException e) {  
 System.*out*.print("Error occurred while Update Operation: " + e);  
 throw e;  
 }  
 }  
  
 public static void deleteArtistByID (String artID) throws SQLException, ClassNotFoundException {  
 //Declare a Update statement  
 String updateStmt = "DELETE FROM artist WHERE artistId = "+artID;  
 System.*out*.println(updateStmt);  
 //Execute Update operation  
 try {  
 DBUtil.*dbExecuteUpdate*(updateStmt);  
 } catch (SQLException e) {  
 System.*out*.print("Error occurred while Update Operation: " + e);  
 throw e;  
 }  
 }  
  
  
}

### 5.3.6 Artist View class

This is the class in charge of displaying the information to the user and take inputs. Depending on the inputs taken the controller will display a different view or update the same view this is mainly JavaFXML code (imports have been omitted available at original code).

<?xml version="1.0" encoding="UTF-8"?>  
  
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="437.0" prefWidth="720.0" xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.controller.ArtistController">  
 <children>  
 <TextField fx:id="artIdText" layoutX="193.0" layoutY="41.0" prefHeight="25.0" prefWidth="67.0" promptText="Artist ID" />  
 <Label layoutX="33.0" layoutY="21.0" text="Create Artist Record" />  
 <Button fx:id="searchArtistButton" layoutX="194.0" layoutY="70.0" mnemonicParsing="false" onAction="#searchArtist" prefHeight="25.0" prefWidth="56.0" text="Search" />  
  
 <TextArea fx:id="resultArea" layoutX="9.0" layoutY="334.0" prefHeight="85.0" prefWidth="167.0" wrapText="true" />  
 <VBox layoutX="594.0" layoutY="41.0" prefHeight="50.0" prefWidth="91.0">  
 <children>  
 <TextField fx:id="idToDelete" prefHeight="25.0" prefWidth="67.0" promptText="Artist ID" />  
 <Button mnemonicParsing="false" onAction="#deleteArtist" prefHeight="25.0" prefWidth="104.0" text="Delete artist" />  
 </children>  
 </VBox>  
 <Label layoutX="51.0" layoutY="308.0" text="Result Console">  
 <font>  
 <Font name="System Bold" size="12.0" />  
 </font>  
 </Label>  
 <VBox layoutX="65.0" layoutY="54.0" prefHeight="174.0" prefWidth="114.0" spacing="4.0">  
 <children>  
 <TextField fx:id="nameText" prefHeight="25.0" prefWidth="79.0" promptText="Name" />  
 <TextField fx:id="surnameText" prefHeight="25.0" prefWidth="79.0" promptText="Surname" />  
 <TextField fx:id="emailText" prefHeight="25.0" prefWidth="79.0" promptText="email" />  
 <TextField fx:id="phoneText" layoutX="10.0" layoutY="10.0" prefHeight="25.0" prefWidth="79.0" promptText="Phone Number" />  
 <ChoiceBox fx:id="gender" prefWidth="150.0" />  
 <ChoiceBox fx:id="type" prefWidth="150.0" />  
 <DatePicker fx:id="dob" promptText="Date of Birth" />  
 </children>  
 </VBox>  
 <VBox layoutX="17.0" layoutY="58.0" prefHeight="174.0" prefWidth="67.0" spacing="12.0">  
 <children>  
 <Label text="Name" />  
 <Label text="Surname" />  
 <Label text="Email" />  
 <Label layoutX="10.0" layoutY="10.0" text="Phone" />  
 <Label layoutX="10.0" layoutY="97.0" text="Gender" />  
 <Label layoutX="10.0" layoutY="126.0" text="Type" />  
 <Label layoutX="10.0" layoutY="155.0" text="DOB" />  
 </children>  
 </VBox>  
 <Separator layoutY="14.0" prefHeight="4.0" prefWidth="600.0" />  
 <Separator layoutX="180.0" layoutY="14.0" orientation="VERTICAL" prefHeight="333.0" prefWidth="7.0" />  
 <TableView fx:id="artistTable" editable="true" layoutX="193.0" layoutY="117.0" prefHeight="302.0" prefWidth="517.0" tableMenuButtonVisible="true">  
 <columns>  
 <TableColumn fx:id="artistIdColumn" prefWidth="57.0" text="Id" />  
 <TableColumn fx:id="artistNameColumn" prefWidth="75.0" text="Name" />  
 <TableColumn fx:id="artistLastNameColumn" prefWidth="73.0" text="Surname" />  
 <TableColumn fx:id="artistEmailColumn" prefWidth="79.0" text="Email" />  
 <TableColumn fx:id="artistPhoneNumberColumn" prefWidth="73.0" text="Phone" />  
 <TableColumn fx:id="artistGenderColumn" prefWidth="73.0" text="Gender" />  
 <TableColumn fx:id="artistDOBDateColumn" prefWidth="73.0" text="Date of Birth" />  
  
 </columns>  
 </TableView>  
 <Button fx:id="searchArtistsButton" layoutX="468.0" layoutY="26.0" mnemonicParsing="false" onAction="#searchArtists" prefHeight="25.0" prefWidth="107.0" text="Search All Artists" />  
 <Button fx:id="createRecordButton" layoutX="47.0" layoutY="261.0" mnemonicParsing="false" onAction="#createArtistRecord" text="Create Record" />  
 <Label layoutX="327.0" layoutY="18.0" prefHeight="17.0" prefWidth="47.0" text="Filter by" />  
 <VBox layoutX="274.0" layoutY="35.0" prefHeight="41.0" prefWidth="187.0">  
 <children>  
 <HBox prefHeight="16.0" prefWidth="187.0">  
 <children>  
 <Label prefHeight="17.0" prefWidth="93.0" text="Gender" />  
 <Label prefHeight="17.0" prefWidth="105.0" text="Type" />  
 </children>  
 </HBox>  
 <HBox prefHeight="26.0" prefWidth="187.0">  
 <children>  
 <VBox prefHeight="52.0" prefWidth="91.0">  
 <children>  
 <ChoiceBox fx:id="genderChoice" prefWidth="150.0" />  
 </children>  
 </VBox>  
 <VBox prefHeight="200.0" prefWidth="100.0">  
 <children>  
 <ChoiceBox fx:id="typeChoice" prefWidth="150.0" />  
 </children>  
 </VBox>  
 </children>  
 </HBox>  
 </children>  
 </VBox>  
 <Button fx:id="filterButton" layoutX="334.0" layoutY="83.0" mnemonicParsing="false" onAction="#filterTable" prefHeight="4.0" prefWidth="52.0" text="Filter" />  
 </children>  
</AnchorPane>

#### Figure 5.3.6.1



### 5.3.7 Song Controller class

With this class, all the operations done in the song view are dealt with by this class for example adding new songs deleting them, search song etc. (imports have been omitted available at original code).

*/\*\*  
 \* Created by Carlito on 10/04/2017.  
 \*/*public class SongController {  
 @FXML  
 private TextField songIdText;  
 @FXML  
 private TextField songNameText;  
 @FXML  
 private TextField albumText;  
 @FXML  
 private DatePicker dateCreatedText;  
 @FXML  
 private TextField idToDelete;  
 @FXML  
 private TextArea resultArea;  
 @FXML  
 private TableView songTable;  
 @FXML  
 private TableColumn<Song, Integer> songIDColumn;  
 @FXML  
 private TableColumn<Song, String> songNameColumn;  
 @FXML  
 private TableColumn<Song, String> albumColumn;  
 @FXML  
 private TableColumn<Song, Date> dateCreatedColumn;  
  
 @FXML  
 private void searchSong(ActionEvent actionEvent) throws ClassNotFoundException, SQLException{  
 try {  
 //Get Song information  
 Song song = SongDao.*searchSong*(songIdText.getText());  
 //Populate Song on TableView and Display on TextArea  
 populateAndShowSong(song);  
 } catch (SQLException e) {  
 e.printStackTrace();  
 resultArea.setText("Error occurred while getting Song information from DB.\n" + e);  
 throw e;  
 }  
 }  
  
 @FXML  
 private void searchAllSongs(ActionEvent actionEvent) throws ClassNotFoundException, SQLException{  
 try {  
 //Get all Song information  
 ObservableList<Song> songData = SongDao.*searchAllSongs*();  
 //Populate Song on TableView  
 populateAllSongs(songData);  
 } catch (SQLException e){  
 System.*out*.println("Error occurred while getting Song information from DB.\n" + e);  
 throw e;  
 }  
  
 }  
  
 @FXML  
 private void initialize () {  
  
 songIDColumn.setCellValueFactory(cellData -> cellData.getValue().songIDProperty().asObject());  
 songNameColumn.setCellValueFactory(cellData -> cellData.getValue().songNameProperty());  
 albumColumn.setCellValueFactory(cellData -> cellData.getValue().albumProperty());  
 dateCreatedColumn.setCellValueFactory(cellData -> cellData.getValue().songDateProperty());  
  
 }  
  
 @FXML  
 private void populateSong (Song song) throws ClassNotFoundException {  
 //Declare and ObservableList for table view  
 ObservableList<Song> songData = FXCollections.*observableArrayList*();  
 //Add Song to the ObservableList  
 songData.add(song);  
 //Set items to the SongTable  
 songTable.setItems(songData);  
 }  
  
 @FXML  
 private void setSongInfoToTextArea ( Song song) {  
 resultArea.setText("Song Name: " + song.getSongName() + "\n" +  
 "Album: " + song.getAlbum());  
 }  
  
 @FXML  
 private void populateAndShowSong(Song song) throws ClassNotFoundException {  
 if (song != null) {  
 populateSong(song);  
 setSongInfoToTextArea(song);  
 } else {  
 resultArea.setText("This Song does not exist!\n");  
 }  
 }  
  
 //Populate Song for TableView  
 @FXML  
 private void populateAllSongs (ObservableList<Song> songData) throws ClassNotFoundException {  
 //Set items to the SongTable  
 songTable.setItems(songData);  
 }  
  
  
  
 @FXML  
 private void createSongRecord(ActionEvent actionEvent) throws SQLException, ClassNotFoundException{  
  
 try {  
 java.sql.Date sqlDate = java.sql.Date.*valueOf*(dateCreatedText.getValue());  
 System.*out*.println(songNameText.getText());  
 System.*out*.println(albumText.toString());  
 SongDao.*insertSong*(songNameText.getText(),albumText.getText(), sqlDate);  
 resultArea.setText("Song inserted! \n");  
 } catch (SQLException e) {  
 resultArea.setText("Problem occurred while inserting song " + e);  
 throw e;  
 }  
  
 }  
  
 @FXML  
 private void deleteSong() throws SQLException, ClassNotFoundException{  
 try {  
 SongDao.*deleteSongByID*(idToDelete.getText());  
 resultArea.setText("Artist deleted! Artist id: " + idToDelete.getText() + "\n");  
 } catch (SQLException e){  
 resultArea.setText("Problem occurred while deleting artist " + e);  
 throw e;  
 }  
  
 }  
  
}

### 5.3.8 Song class

As mentioned before in the Artists class (5.3.4) this is the base to build each unique object of type song which has certain attributes that are a requirement for the company (imports have been omitted available at original code).

*/\*\*  
 \* Created by Carlito on 18/04/2017.  
 \*/*public class Song {  
 private IntegerProperty songID;  
 private StringProperty songName;  
 private StringProperty album;  
 private SimpleObjectProperty<Date> dateCreated;  
  
  
  
  
 public Song() {  
 this.songID = new SimpleIntegerProperty();  
 this.songName = new SimpleStringProperty();  
 this.album = new SimpleStringProperty();  
 this.dateCreated = new SimpleObjectProperty<>();  
 }  
  
  
  
 public int getSongID() {  
 return songID.get();  
 }  
  
 public void setSongID(int artist\_ID) {  
 this.songID.set(artist\_ID);  
 }  
  
 public IntegerProperty songIDProperty(){  
 return songID;  
 }  
  
  
 public String getAlbum() {  
 return album.get();  
 }  
  
 public void setAlbum(String album) {  
 this.album.set(album);  
 }  
  
 public StringProperty albumProperty(){  
 return album;  
 }  
  
  
  
 public String getSongName() {  
 return songName.get();  
 }  
  
 public void setSongName(String songName) {  
 this.songName.set(songName);  
 }  
  
 public StringProperty songNameProperty(){  
 return songName;  
 }  
  
  
  
 public Object getDateCreated() {  
 return dateCreated.get();  
 }  
  
 public void setDateCreated(Date dateCreated) {  
 this.dateCreated.set(dateCreated);  
 }  
  
 public SimpleObjectProperty<Date> songDateProperty(){  
 return dateCreated;  
 }}

### 5.3.9 SongDao Class

This is the Song data access object. This class focuses in getting all the information the program needs about the songs from the database it performs delete, update and insert operations (imports have been omitted available at original code).

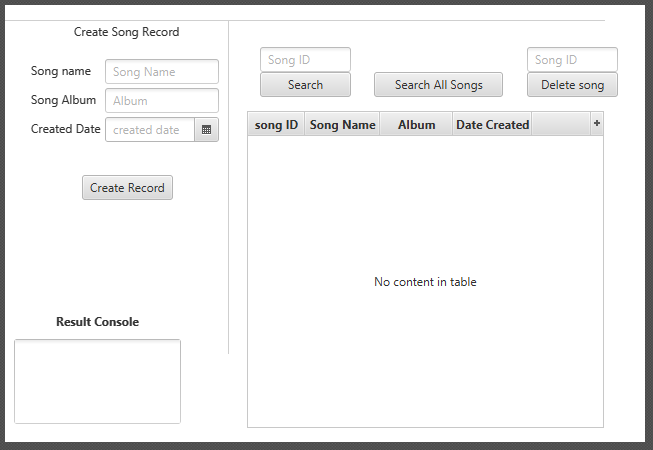
*/\*\*  
 \* Created by Carlito on 18/04/2017.  
 \*/*public class SongDao {  
 public static Song searchSong(String songID) throws SQLException, ClassNotFoundException {  
 //Declare a SELECT statement  
 String selectStmt = "SELECT \* FROM song WHERE songID= "+songID;  
  
 //Execute SELECT statement  
 try {  
 //Get ResultSet from dbExecuteQuery method  
 ResultSet rsSong = DBUtil.*dbExecuteQuery*(selectStmt);  
  
 //Send ResultSet to the getSong from resultset method and get Song object  
 Song Song = *getSongFromResultSet*(rsSong);  
  
  
 //Return Song object  
 return Song;  
 } catch (SQLException e) {  
 System.*out*.println("While searching a song with " + songID + " id, an error occurred: " + e);  
 //Return exception  
 throw e;  
 }  
 }  
  
 private static Song getSongFromResultSet(ResultSet rs) throws SQLException  
 {  
 Song song = null;  
 if (rs.next()) {  
 song.setSongID(rs.getInt("songId"));  
 song.setSongName(rs.getString("SongName"));  
 song.setAlbum(rs.getString("album"));  
 song.setDateCreated(rs.getDate("dateCreated"));  
  
 }  
 return song;  
 }  
  
  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 //SELECT All SongS  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 public static ObservableList<Song> searchAllSongs() throws SQLException, ClassNotFoundException {  
 //Declare a SELECT statement  
 String selectStmt = "SELECT \* FROM song";  
  
 //Execute SELECT statement  
 try {  
 //Get ResultSet from dbExecuteQuery method  
 ResultSet rsSongs = DBUtil.*dbExecuteQuery*(selectStmt);  
  
  
 //Send ResultSet to the getSongsList method and get Song object  
 ObservableList<Song> SongList = *getSongsList*(rsSongs);  
  
 //Return Song object  
 return SongList;  
  
 } catch (SQLException e) {  
 System.*out*.println("SQL select operation has failed: " + e);  
 //Return exception  
 throw e;  
 }  
 }  
  
 //Select \* from Songs operation  
 private static ObservableList<Song> getSongsList(ResultSet rs) throws SQLException, ClassNotFoundException {  
 //Declare a observable List which comprises of Songs objects  
 ObservableList<Song> songsList = FXCollections.*observableArrayList*();  
  
 while (rs.next()) {  
 Song song = new Song();  
 song.setSongID(rs.getInt("songId"));  
 song.setSongName(rs.getString("SongName"));  
 song.setAlbum(rs.getString("album"));  
 song.setDateCreated(rs.getDate("dateCreated"));  
 //Add Song to the ObservableList  
 songsList.add(song);  
  
 }  
  
 //return Song list (ObservableList of Songs)  
 return songsList;  
 }  
  
 public static void insertSong (String songName, String album, Date dateCreated ) throws SQLException, ClassNotFoundException {  
 //Declare a Update statement  
 String updateStmt = "INSERT INTO song(songName, album, dateCreated) VALUES('"+songName+"', '"+album+"', '"+dateCreated+"')";  
  
  
 //Execute Update operation  
 try {  
 DBUtil.*dbExecuteUpdate*(updateStmt);  
 } catch (SQLException e) {  
 System.*out*.print("Error occurred while Update Operation: " + e);  
 throw e;  
 }  
 }  
  
 public static void deleteSongByID (String songID) throws SQLException, ClassNotFoundException {  
 //Declare a Update statement  
 String updateStmt = "DELETE FROM song WHERE songID = "+songID;  
 System.*out*.println(updateStmt);  
 //Execute Update operation  
 try {  
 DBUtil.*dbExecuteUpdate*(updateStmt);  
 } catch (SQLException e) {  
 System.*out*.print("Error occurred while Update Operation: " + e);  
 throw e;  
 }  
 }  
}

### 5.3.10 Song view Class

This class is the view for the song operations it will handle the input taken from the user and is updated by the song controller class (imports have been omitted available at original code).

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="437.0" prefWidth="640.0" xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.controller.SongController">  
 <children>  
 <Label layoutX="69.0" layoutY="18.0" text="Create Song Record" />  
  
 <TextArea fx:id="resultArea" layoutX="9.0" layoutY="334.0" prefHeight="85.0" prefWidth="167.0" wrapText="true" />  
 <VBox layoutX="522.0" layoutY="42.0" prefHeight="50.0" prefWidth="91.0">  
 <children>  
 <TextField fx:id="idToDelete" prefHeight="25.0" prefWidth="67.0" promptText="Song ID" />  
 <Button mnemonicParsing="false" onAction="#deleteSong" prefHeight="25.0" prefWidth="104.0" text="Delete song" />  
 </children>  
 </VBox>  
 <Label layoutX="51.0" layoutY="308.0" text="Result Console">  
 <font>  
 <Font name="System Bold" size="12.0" />  
 </font>  
 </Label>  
 <VBox layoutX="100.0" layoutY="54.0" prefHeight="98.0" prefWidth="114.0" spacing="4.0">  
 <children>  
 <TextField fx:id="songNameText" prefHeight="25.0" prefWidth="79.0" promptText="Song Name" />  
 <TextField fx:id="albumText" prefHeight="25.0" prefWidth="79.0" promptText="Album" />  
 <DatePicker fx:id="dateCreatedText" promptText="created date" />  
 </children>  
 </VBox>  
 <VBox layoutX="26.0" layoutY="57.0" prefHeight="98.0" prefWidth="74.0" spacing="12.0">  
 <children>  
 <Label text="Song name" />  
 <Label text="Song Album" />  
 <Label layoutX="10.0" layoutY="155.0" prefHeight="17.0" prefWidth="85.0" text="Created Date" />  
 </children>  
 </VBox>  
 <Separator layoutY="14.0" prefHeight="4.0" prefWidth="600.0" />  
 <Separator layoutX="221.0" layoutY="16.0" orientation="VERTICAL" prefHeight="333.0" prefWidth="7.0" />  
 <TableView fx:id="songTable" editable="true" layoutX="242.0" layoutY="106.0" prefHeight="317.0" prefWidth="357.0" tableMenuButtonVisible="true">  
 <columns>  
 <TableColumn fx:id="songIDColumn" prefWidth="57.0" text="song ID" />  
 <TableColumn fx:id="songNameColumn" prefWidth="75.0" text="Song Name" />  
 <TableColumn fx:id="albumColumn" prefWidth="73.0" text="Album" />  
 <TableColumn fx:id="dateCreatedColumn" prefWidth="79.0" text="Date Created" />  
 </columns>  
 </TableView>  
 <Button fx:id="searchSongButton" layoutX="369.0" layoutY="67.0" mnemonicParsing="false" onAction="#searchAllSongs" prefHeight="25.0" prefWidth="129.0" text="Search All Songs" />  
 <Button fx:id="createSongRecord" layoutX="77.0" layoutY="170.0" mnemonicParsing="false" onAction="#createSongRecord" text="Create Record" />  
 <VBox layoutX="255.0" layoutY="42.0" prefHeight="50.0" prefWidth="91.0">  
 <children>  
 <TextField fx:id="songIdText" prefHeight="25.0" prefWidth="67.0" promptText="Song ID" />  
 <Button fx:id="searchSongIDButton" mnemonicParsing="false" onAction="#searchSong" prefHeight="25.0" prefWidth="96.0" text="Search" />  
 </children>  
 </VBox>  
 </children>  
</AnchorPane>

#### Figure 5.3.10.1



### 5.3.11 DBUtil Class

The DBUtil class makes takes the role of connecting to the database it assigns the database it connects the credentials it needs to get access. It also is responsible for running the queries needed to get data from the database.

*/\*\*  
 \* Created by Carlito on 12/04/2017.  
 \*/*public class DBUtil {  
  
 private static final String *JDBC\_DRIVER* = "com.mysql.jdbc.Driver";  
  
 //Connection  
 private static Connection *conn* = null;  
 private static final String *db* = "projectdb";  
 private static final String *URL* = "jdbc:mysql://localhost:3306/" + *db*+"?autoReconnect=true&useSSL=false";  
 private static final String *user* = "carlos";  
 private static final String *pass* = "carlos";  
  
  
  
 public static void dbConnect() throws SQLException, ClassNotFoundException{  
 //Sets MySQL driver  
 try{  
 Class.*forName*(*JDBC\_DRIVER*);  
 }catch (ClassNotFoundException e){  
 System.*out*.println("Where is MySQL driver?");  
 e.printStackTrace();  
 throw e;  
 }  
 System.*out*.println("MySQL driver Registered!");  
  
 //Establish the MySQL Connection using Connection String  
  
 try{  
 *conn* = DriverManager.*getConnection*(*URL*, *user*, *pass*);  
 }catch (SQLException e){  
 System.*out*.println("Connection Failed! Check output console" + e);  
 e.printStackTrace();  
 throw e;  
 }  
 }  
  
 //Close Connection  
 public static void dbDisconnect() throws SQLException {  
 try {  
 if (*conn* != null && !*conn*.isClosed()) {  
 *conn*.close();  
 }  
 } catch (Exception e){  
 throw e;  
 }  
 }  
  
 //DB Execute Query Operation  
 public static ResultSet dbExecuteQuery(String queryStmt) throws SQLException, ClassNotFoundException {  
 //Declare statement, resultSet and CachedResultSet as null  
 Statement stmt = null;  
 ResultSet resultSet = null;  
 CachedRowSetImpl crs = null;  
 try {  
 //Connect to DB (Establish MySQL Connection)  
 *dbConnect*();  
 System.*out*.println("Select statement: " + queryStmt + "\n");  
  
 //Create statement  
 stmt = *conn*.createStatement();  
  
 //Execute select (query) operation  
 resultSet = stmt.executeQuery(queryStmt);  
  
 //CachedRowSet Implementation  
 //In order to prevent "java.sql.SQLRecoverableException: Closed Connection: next" error  
 //CachedRowSet is there  
 crs = new CachedRowSetImpl();  
 crs.populate(resultSet);  
  
 } catch (SQLException e) {  
 System.*out*.println("Problem occurred at executeQuery operation : " + e);  
 throw e;  
 } finally {  
 if (resultSet != null) {  
 //Close resultSet  
 resultSet.close();  
 }  
 if (stmt != null) {  
 //Close Statement  
 stmt.close();  
 }  
 //Close connection  
 *dbDisconnect*();  
 }  
 //Return CachedRowSet  
 return crs;  
 }  
  
 //DB Execute Update (For Update/Insert/Delete) Operation  
 public static void dbExecuteUpdate(String sqlStmt) throws SQLException, ClassNotFoundException {  
 //Declare statement as null  
 Statement stmt = null;  
 try {  
 //Connect to DB (Establish MySQL Connection)  
 *dbConnect*();  
 //Create Statement  
 stmt = *conn*.createStatement();  
 //Run executeUpdate operation with given sql statement  
 stmt.executeUpdate(sqlStmt);  
 } catch (SQLException e) {  
 System.*out*.println("Problem occurred at executeUpdate operation : " + e);  
 throw e;  
 } finally {  
 if (stmt != null) {  
 //Close statement  
 stmt.close();  
 }  
 //Close connection  
 *dbDisconnect*();  
 }  
 }  
  
  
}

# Chapter 6: Testing and Results

## 6.1 Introduction

In this chapter, the main focus is the different test ran in the program and the outcome of each one of them.

## 6.2 Testing

### 6.2.1 Black-box Testing

The reason testing is done is to check the functionality of the system implemented. There many ways to do a test. The first one will be Black-box testing as this will show the user the user interface, usability, functionality and the connectivity of the Avinpro manager system.

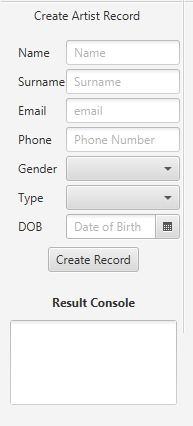
Black-box testing tests functionality if the systems without instructions regarding the internal operation. This kind of testing is easy to use and it is useful for the user to test the functionality if the system in use without any specific order of which part is to be tested first or second for example testing for getting songs from database, searching a specific artist.

#### Figure 6.2.1.1 Black-box Testing



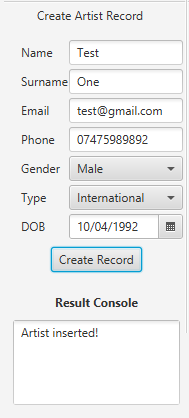
Now a series of test will be carried out to test the functionality of the Avinpro manager program.

### 6.2.2 Adding an Artist Record



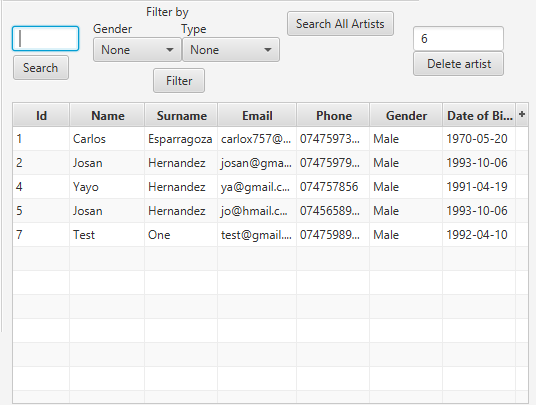
If the user fills the form and press the create record button, a new record will be created for an artist in the result console a message saying creation successful should come up if information is correct. (See figure 6.2.2.1)

#### Figure 6.2.2.1

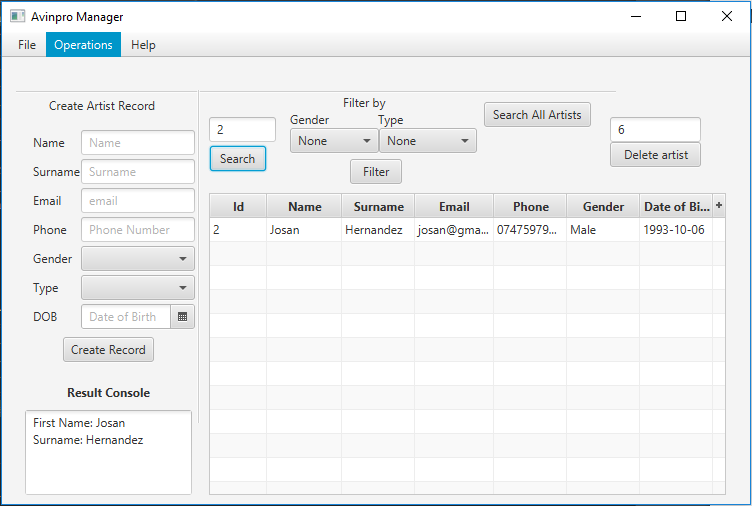


### 6.2.3 Searching Artist by ID

If the user types in the search box the ID number of an artist, the program will run a function that queries the database and finds the specified ID number. Adding to this the result console should display a message if artist is found (See figure 6.2.3.1).



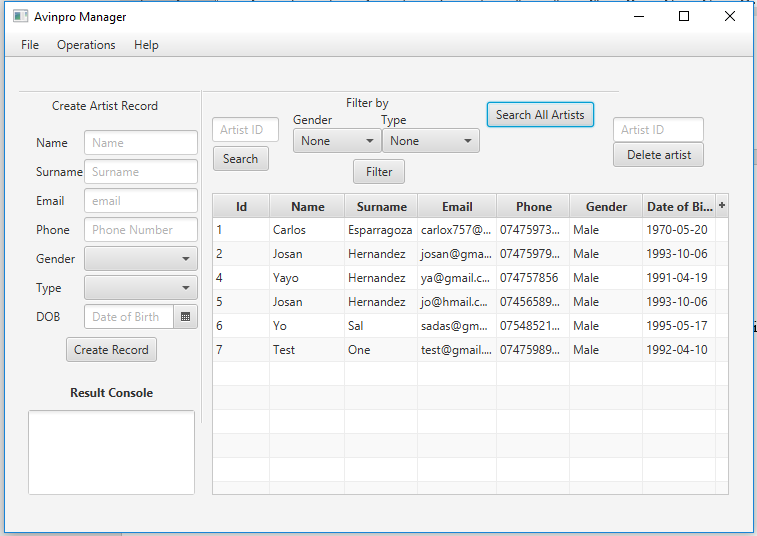
#### Figure 6.2.3.1



### 6.2.4 Search All artists

When the user clicks the button all the records of artist should come up the screen displaying all the attributes of each artist. (see figure 6.2.4.1)

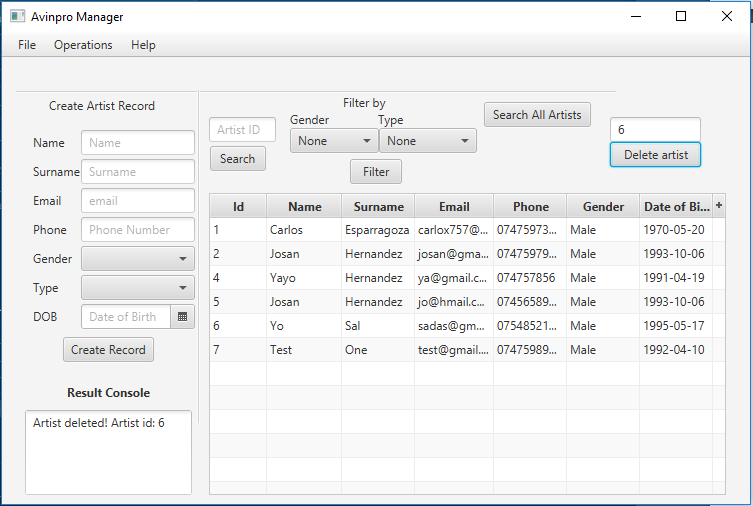
#### Figure 6.2.4.1



### 6.2.5 Delete Artist Record

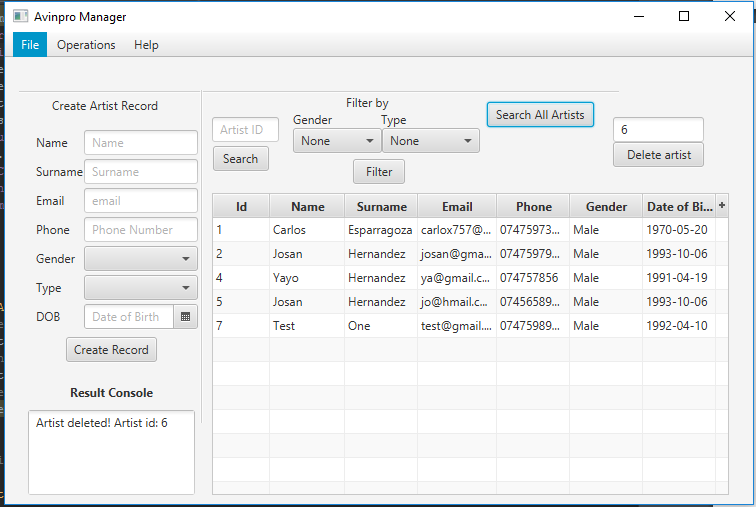
If user wants to delete artist the ID number of the artist must be provided, then press the delete button to delete the record of this artist a message of successful deletion should come up to alert user records has been deleted. (see figure 6.2.5.1) and (see figure 6.2.5.2)

#### Figure 6.2.5.1



#### Figure 6.2.5.2

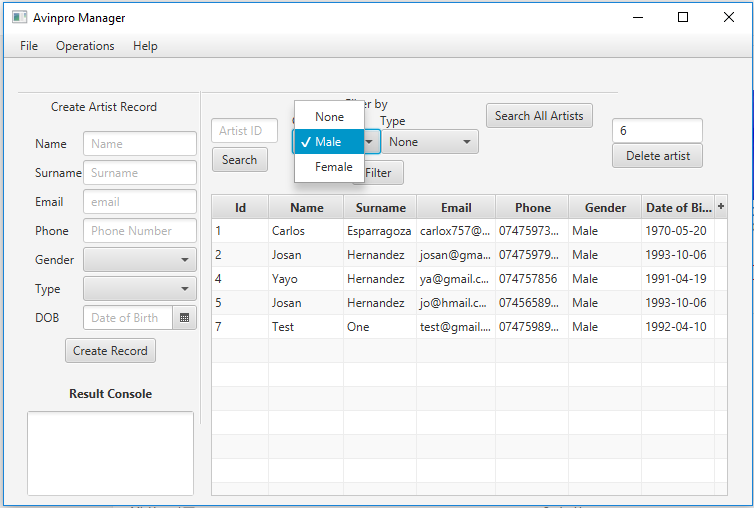
Shows record has been deleted



### 6.2.6 Filtering by Artist Gender

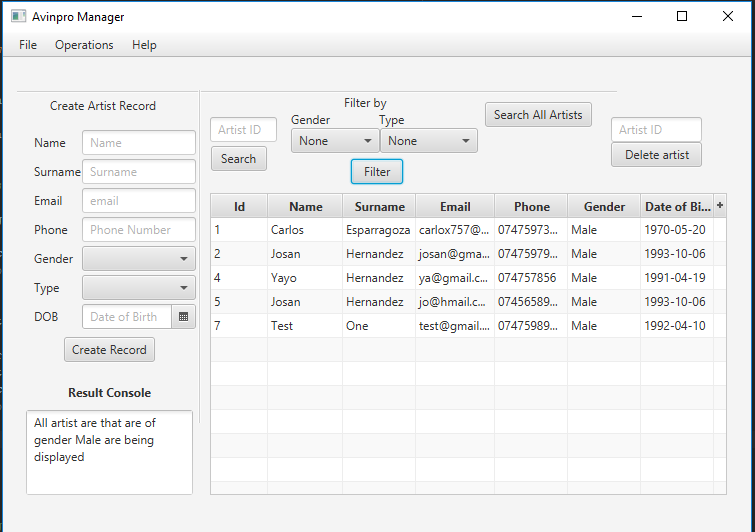
The user can select which criteria the filtering will be done in this case by gender there is two options option one is “Male” option two is “Female” once selected the user must click the button filter so the database is queried using the value selected by user a default value of none is predefined which queries all the records. (see figure 6.2.6.1 and see figure 6.2.6.2)

#### Figure 6.2.6.1



#### Figure 6.2.6.2

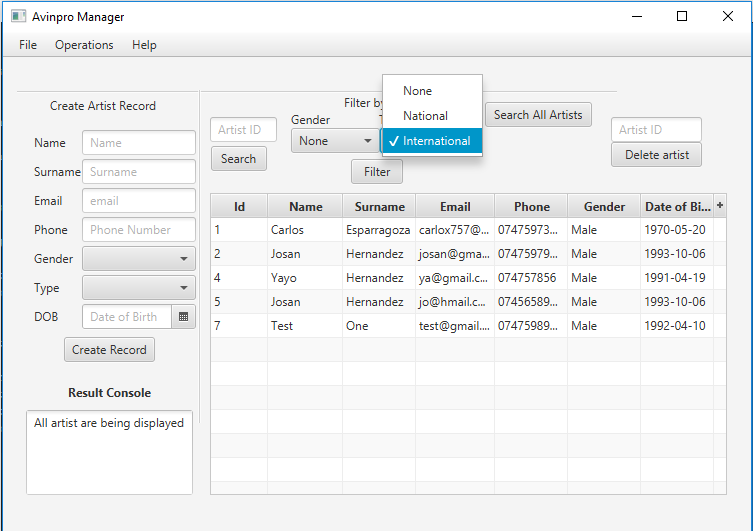
Result from gender filtering criteria.



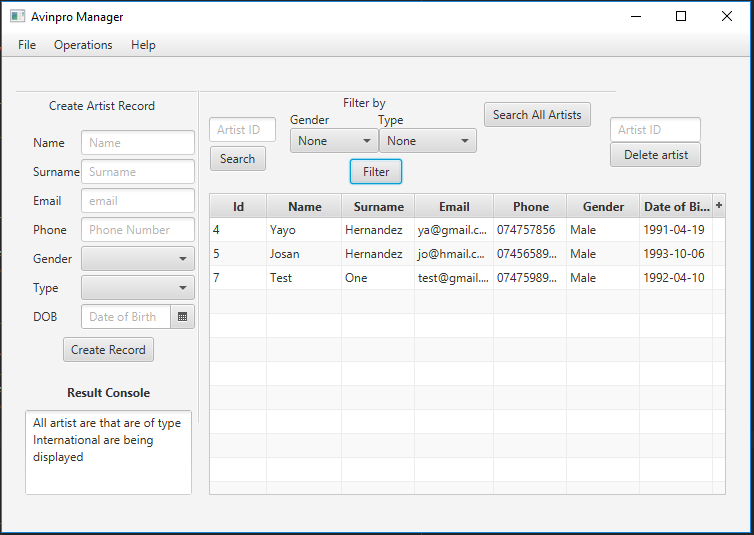
### 6.2.7 Filtering by Artist Type

User can also filter by using the type of artists in this case the options are: option one “National” option two “International”. After selecting an option user must click the filter button then the database will query the results of artists that have the type selected. If no values are selected, then the result will be all the artists from the database. (see figure 6.2.7.1 and see figure 6.2.7.2)

#### Figure 6.2.7.1



#### Figure 6.2.7.2

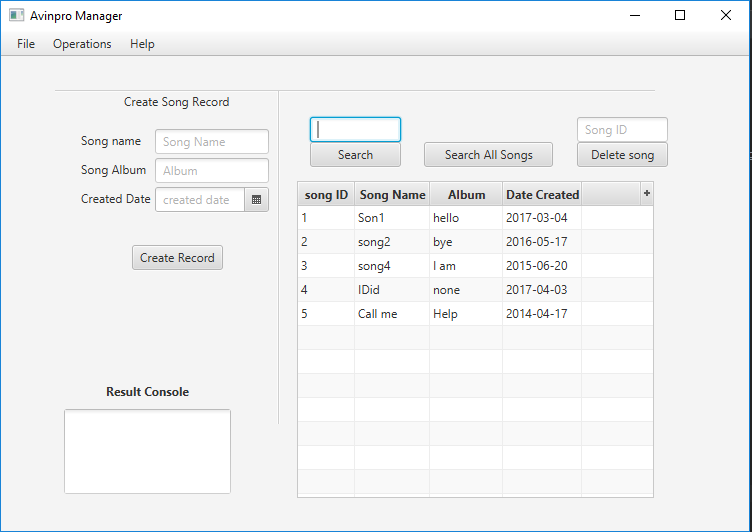


### 6.2.8 Changing between views

User can change between the records they can see for example there are records for the songs and records for the artists this functions allows them to navigate through such views. (see figure 6.2.8.1 and see figure 6.2.8.2)

#### Figure 6.2.8.1

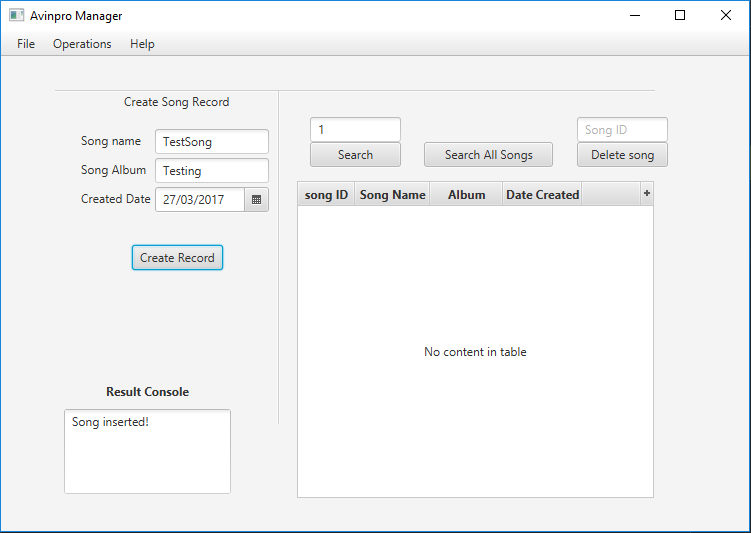
#### Figure 6.2.8.2



### 6.2.9 Adding a Song Record

This function will allow the user to add a song record to the database with the attributes the form asks for this will then be taken by the Song controller which will then use the ArtistDao to insert the information in the database (see figure 6.2.9.1).

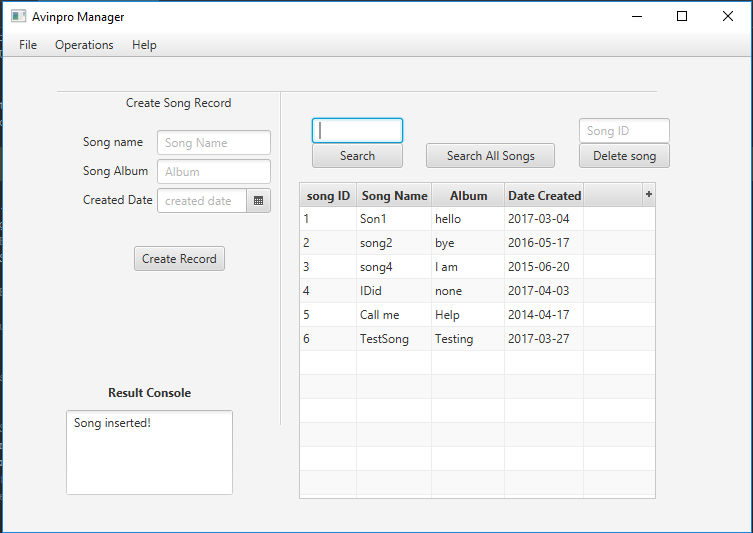
#### Figure 6.2.9.1



### 6.2.10 Searching All Song Records

By clicking the button search all songs user can get all the records from the database. (see figure 6.2.10.1)

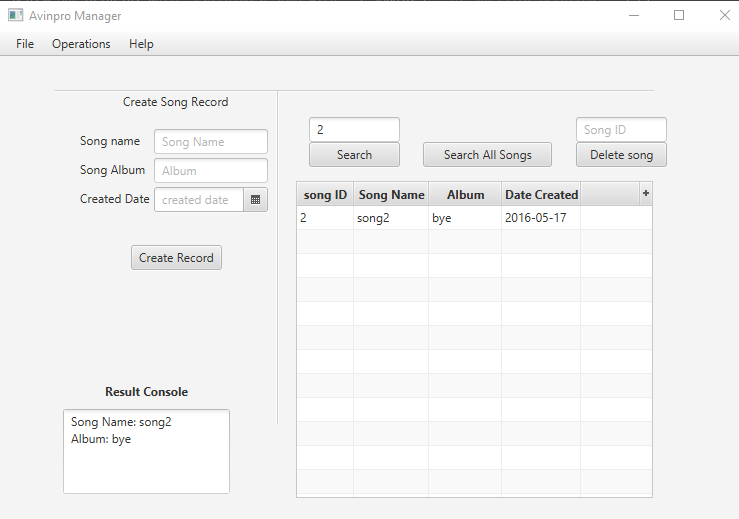
#### Figure 6.2.10.1



### 6.2.11 Searching a Song Record by ID

This will allow users to search a song by a specified ID number they can type on the textbox then by clicking search the database will be queried and the result will be displayed. The result console will provide a message for example “search successful”. (see figure 6.2.11.1)

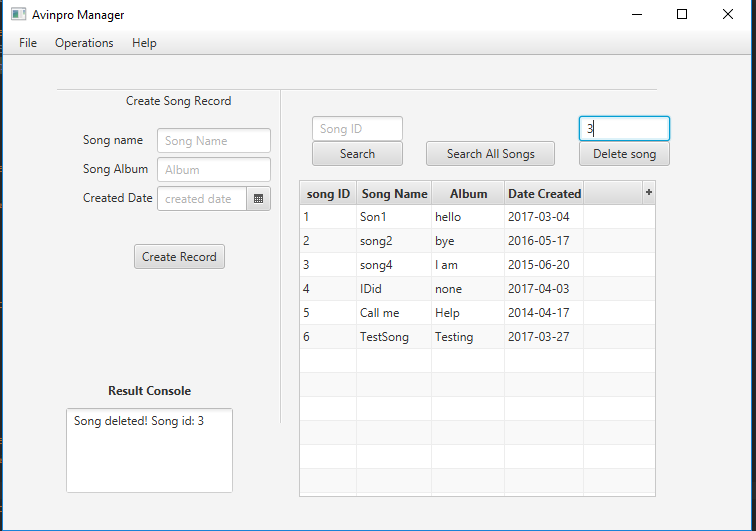
#### Figure 6.2.11.1



### 6.2.12 Deleting a Song Record by ID

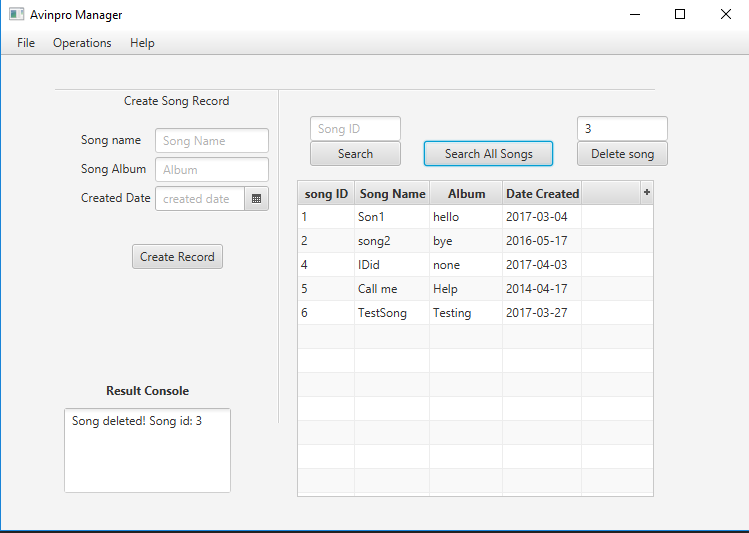
User is now able to delete a song by its ID number, must be typed in the textbox on top of the button that says “Delete song” once the ID number is typed and the user clicks the button the records of this song will be deleted from the database. The result console will display a message confirming the deletion of the record from the database for example “Records has been deleted”. (See figure 6.2.12.1 and figure 6.2.12.2).

#### Figure 6.2.12.1



#### Figure 6.2.12.2

Shows that record has been deleted



## 6.3 Results

This part of the project will look into the results achieved from the testing of the Avinpro manager system. Another SWOT analysis was carried out after the system was built to show/identify the improvements made. Feedback was provided by the user who was asked to test the program.

### 6.3.1 Results from Black-box Testing the Avinpro manager System

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Outcome | Actual Outcome |
| 1 | Add Artist Record | User can add an Artist record to the database | Successful |
| 2 | Search All artist | User can query all Artists’ records | Successful |
| 3 | Search Artist by ID | User can search Artist record using Artist’s ID number | Successful |
| 4 | Delete Artist Record | User can delete Artists’ records using Artist’s ID number | Successful |
| 5 | Filter Artist by Gender | User can filter Artists’ records by gender selected | Successful |
| 6 | Filter Artist by Type | User is able to filter Artists’ records by type of Artist | Successful |
| 7 | Change current view | User can navigate from the different views available at the main menu to do different operations | Successful |
| 8 | Add Song Record | User is able to add a Song Record | Successful |
| 9 | Search All Songs Records | User can query all the information regarding songs from database | Successful |
| 10 | Search Song by ID | User can get a specific Song record by entering the ID number of the song | Successful |
| 11 | Delete song Record | User can delete a song record using the ID number of the song | Successful |

### 6.3.2 SWOT Analysis after build of the Avinpro Manager System

This SWOT analysis was carried out following the improvements made to the old system to show the difference between the SWOT analysis made before build. (See figure 6.3.2.1)

|  |  |
| --- | --- |
| **Strengths**   * Reduce the amount of time taken to finish the process * Bespoke program for the company’s records * Can be used in any operating system that has java on it * Better level of controlling the information on the system * User is able to filter records * User friendly interface | **Weaknesses**   * Staff might need training to use the program. * Backups required which will enable the user to restore the data of a problem occurs with the system. * Bugs might occur. |
| **Opportunities**   * Increase revenue * Allow company to keep a more manageable record keeping framework * Produce reports with formatted data | **Threats**   * Competitors with better a System * Fire |

### Figure 6.3.2.1

# Chapter 7: Critical Review

## Introduction

The main aim of the project was to develop a database management system with a user-friendly interface allowing the user to store the data, access it and be able to see it to create reports for the Artists, so then they can see how much money they have made for a specific period of time.

Taking this into account the project has three main objectives to fulfil. These objectives will be discussed below and to what extend the project has been able to meet these objectives.

1. Create a fully functional database that will store, edit and delete the information required by the company to produce the reports, this database should able to hold at least 200000 records of artists. The database will be secured and regularly backed up to secure the data in case off loss of data or damage of the same one.

This objective was fulfilled by the project. This was made possible through the use of MySQL workbench which enabled the creation of the database system. The database is able to store, receive and send data to and from the Avinpro Manager interface. It is also able to hold the amount of records required for the system. For example, while building process was occurring, the use of auto increments as well as setting a set amount of character for each field to be saved in the local hard drive. To add to this MySQL gave the necessary tools to the system to raise any errors in the database.

1. To design and build an interface able to deal with the functionality the user needs to fulfil the process of the Distribution. The minimum requirement is to be able to insert, update and delete data from the database.

The creation of an interface between the user and the database was also successful. This was all thanks to the use of the IDE IntelliJ which enabled the project to create the necessary classes, pieces of code to achieve this goal. The main language used in this project is Java also some XML code was used to create the GUI needed for the views. The interface included some important features that the old system did not have. One of these features is the ability to filter large chunks of data from the Artists, also the ability to search, insert and delete records using cascade type deletion which means if a primary key is deleted so will other records containing this key. Another important feature is the ability to change between views (Artists and Song) to enable the user to work in both environments using the same tool.

1. To stablish a connection between the interface and the database using in this case a MySQL database management system.

The project was also able to achieve this objective as the database system were connected using a Java Database Connectivity (JDBC) which helps the client in this case the interface define how to access the database. Using this driver and the Java class DBUtil it was possible for the project to create methods to access and connect to the databased created in MySQL Workbench.

### User Feedback

As the project was a constant back and forth with the client the outcome was positive. There were a lot of suggestions made such as adding more views for other different types of data such as grouping the artists with the songs together or be able to print of a report straight of the program by selecting an artist. Besides this points the program was easy to use by the main user. The ability to add records, search and filter was a success al this done with validation.

Overall the feedback from the user was quite good and further improvements and features will be added to the program.

### Problems Encountered

During the creation of the project there were many technical problems that had to be dealt with. For example, during the creation of the interface there were many Null values causing all kinds of errors when retrieving data after a good time testing, researching for answers and advice most of the issues were solved by making sure a Constructor class was created for each model type data i.e. Artist class or the Song class.

Another example of a problem was to be able to get user input i.e. from the search box or to insert into the database. As the program, could not get anything (Null values again) this issue was solved by making sure each part of the view for example a search box or a button even a menu item had and fx:id this was then access by the controller class that will send the input to the ArtistDao class or the SongDao class depending on which view the user is in.

Also, there was an issue in which the program would change view but as a result of this the main menu at the top was lost. To solve this issue, a lot of research was done to finally found the answer which was that the root layout controller needed access to the Main class so it can change the views displayed to the user to deal with the different kinds of operations

### Achievements and Limitations

Several key achievements within this projects durations where done. One of these achievements is the successful build of an interface which can deal with most of the operations required by the user. An example of this is the ability to filter the records as this is the most tedious part of the process as Artists that are of type International Are dealt with a different way from the National type.

Another achievement was the successful creation the add, delete, search functionality the program has as it was very time consuming and had to be done carefully. The reason for this was mentioned before as Java only tells that there is a null pointer somewhere and finding why every time it happened took some time.

As there is achievements there is also limitations one of the limitation was the ability of the program to produce a report able to self-construct using the information available of each artist and their songs, this was not possible as I the developer had time constraints but it was talked with the product owner and agreed to be left for further development.

The limitation of time and knowledge did not let the project achieve the goal importing CVS files to the database using the interface which can be implemented in to the interface in the near future.

### Implications for Future Work

The indications from the literature and the interviews done with the main user indicates that, in the country the company Avinpro resides has not yet fully implemented new features provided by programming languages. There is not enough research however which looks at how to find a solution to this problem, in few words there is not a good strategy or strategies for educating similar business on the benefits of implementing newer technologies (programing languages, IDEs, JDBC) to overcome barriers, as well as guiding them in the process.

This might be something which local governments may have to look at, and implement some guidelines for these kinds of companies to follow and improve in their IT departments.

### Conclusion

This was a great experience obtained it really puts the developer through the real world by communicating with the main users, getting feedback, improving on work done using the feedback given.

Working with Java really help as in this programing language all real work objects can be represented by classes hence the Object Oriented programing. By using a Model, View, Controller (MVC) approach when designing the program makes sure that the concerns are separated properly.

Also by achieving the functional requirements the user was really happy, making a more organised system for them improvements will be done in the future for now the current outcome is successful.

# References

Gravelle, J. G. (2016). Money for something: Music licensing in the 21st century (CRS Report R43984). Washington, DC: Congressional Research Service.

Staff, I. (2017). *Royalty*. [online] Investopedia. Available at: http://www.investopedia.com/terms/r/royalty.asp [Accessed 24 Jan. 2017].

Docs.oracle.com. (2017). JDBC 4.1. [online] Available at: http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/jdbc\_41.html [Accessed 9 Feb. 2017].

Docs.oracle.com. (2017). 1 JavaFX Overview (Release 8). [online] Available at: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm [Accessed 12 Feb. 2017].

Margaret Rouse (January 2015) Database management system (DBMS), Available at: http://searchsqlserver.techtarget.com/definition/database-management-system (Accessed: 2nd January 2017).

Somya Goyal ( 2016 ) Database Connectivity in JAVA, PDM college of Engineering, Bahadurgarh, India: IJESC.

Jeff Atwood (05 May 2008) Understanding Model-View-Controller, Available at: https://blog.codinghorror.com/understanding-model-view-controller/ (Accessed: 2nd January 2017).

Bruce Eckel (2003) Thinking in Java, 3rd edn., United States of America: Pearson Education.

James L. Weaver (2007) JavaFX Script, 1st edn., United States of America: firstPress.

Florescu, D. and Kossmann, D., 1999. Storing and querying XML data using an RDMBS. IEEE Data Engineering Bulletin, Special Issue on, 1060(22), p.3.

Jemerov, D., 2008, October. Implementing refactorings in IntelliJ IDEA. In Proceedings of the 2nd Workshop on Refactoring Tools (p. 13). ACM.

Pickton, D.W. and Wright, S., 1998. What's swot in strategic analysis?. Strategic change, 7(2), pp.101-109.

MySQL, A.B., 2001. MySQL.

Matthews, M., Cole, J. and Gradecki, J.D., 2003. MySQL and Java developer's guide. John Wiley & Sons.

Goldstein, P., 1989. Copyright: principles, law and practice (Vol. 3). Aspen Publishers.

Sanjek, R., 1988. *American Popular Music and Its Business: The beginning to 1790* (Vol. 1). Oxford University Press, USA.