# 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.

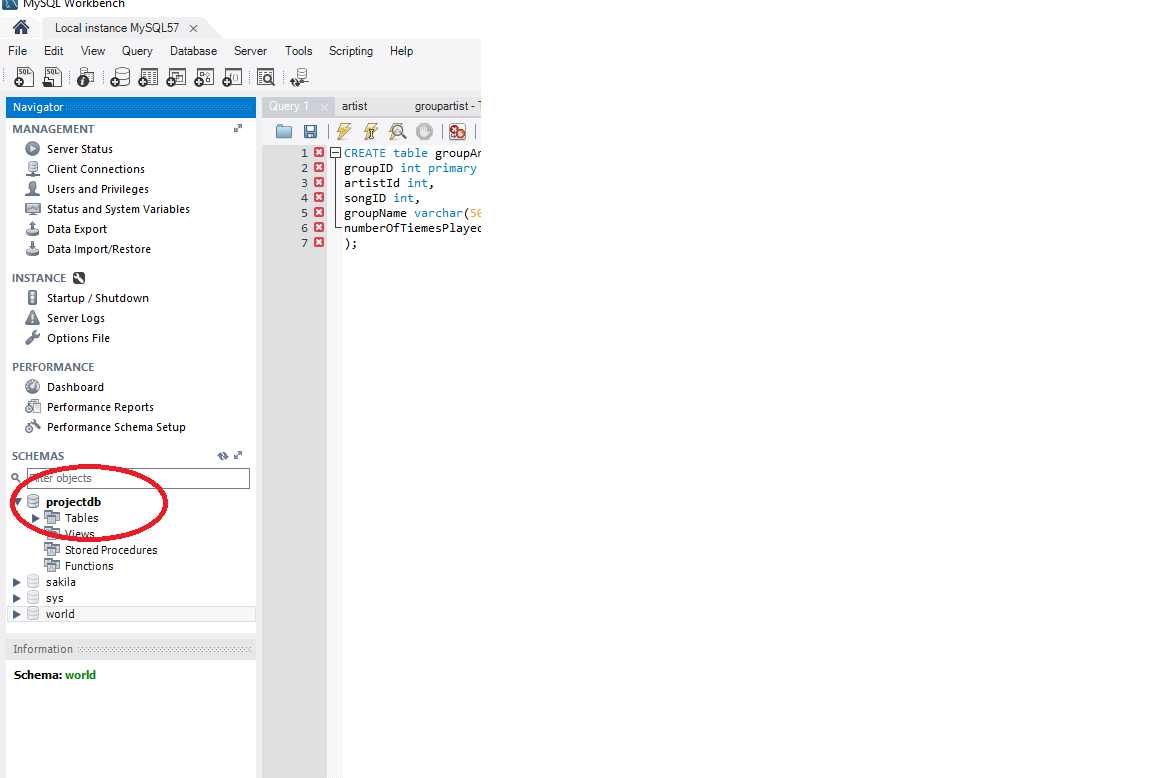
**Deployment**: once the functional and non-functional requirements are tested the system is deployed in the customer environment or released to the market.

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

## 5.2 Database creation

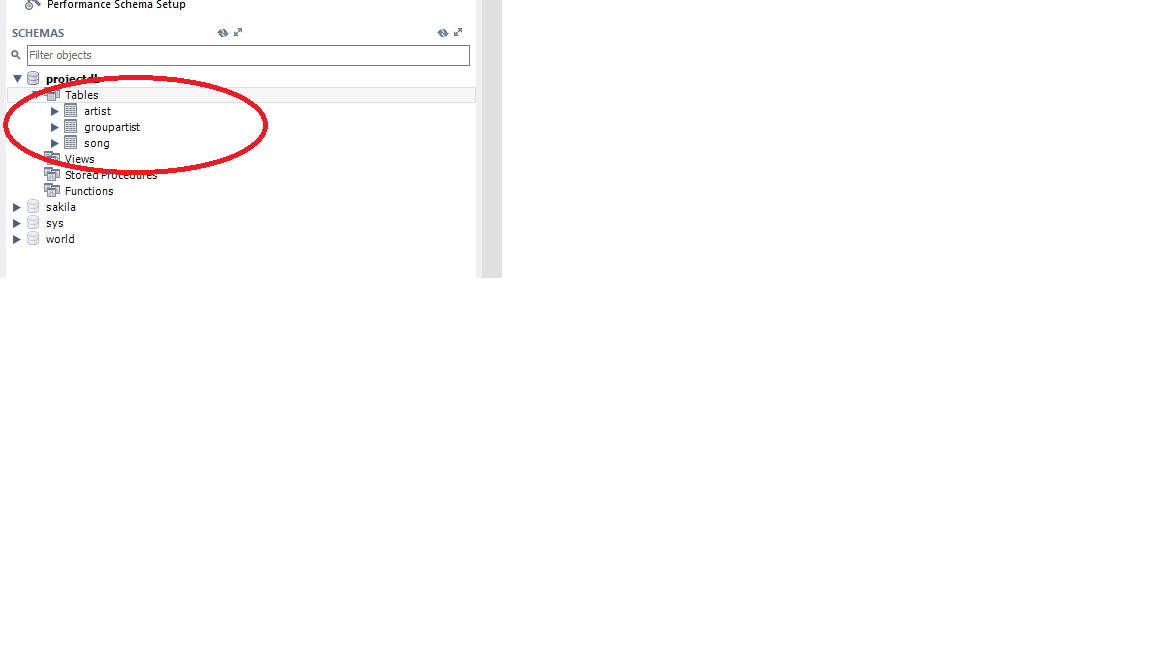
1. The first step is to install the software MySQL Workbench this is the tool used to create the database.
2. Now proceed to create a database to store all the information required by the company each entity needs to have a table for example artists, songs. In this case the database is called projectdb.

## 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`)

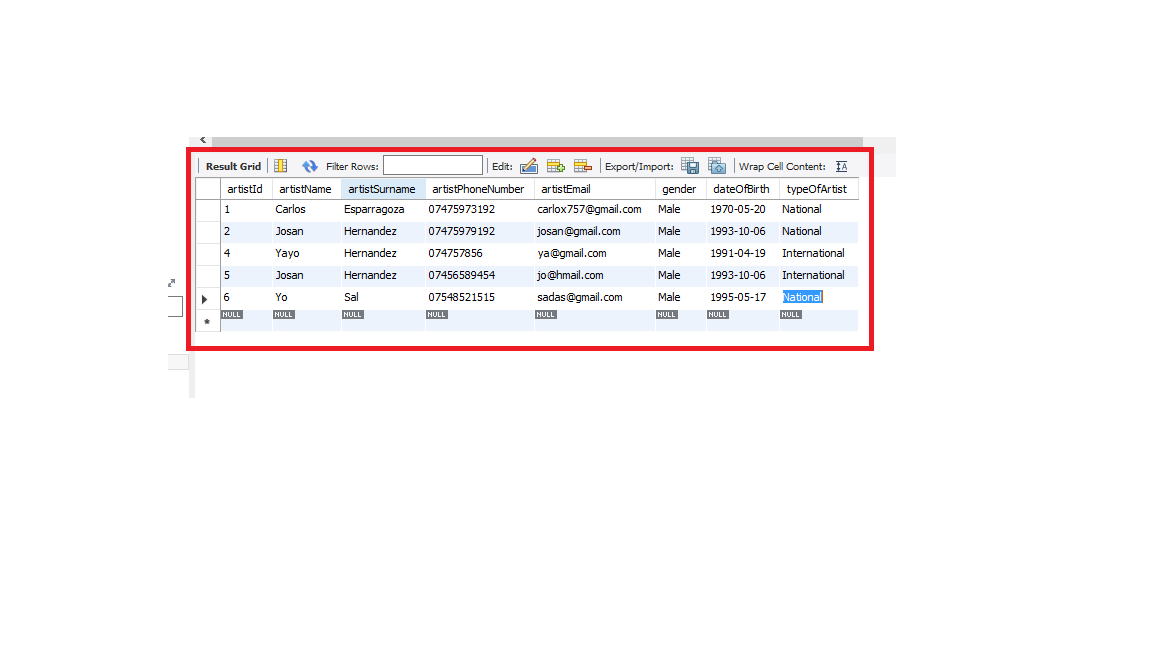
REFERENCES `projectdb`.`song` (`songID`)

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.

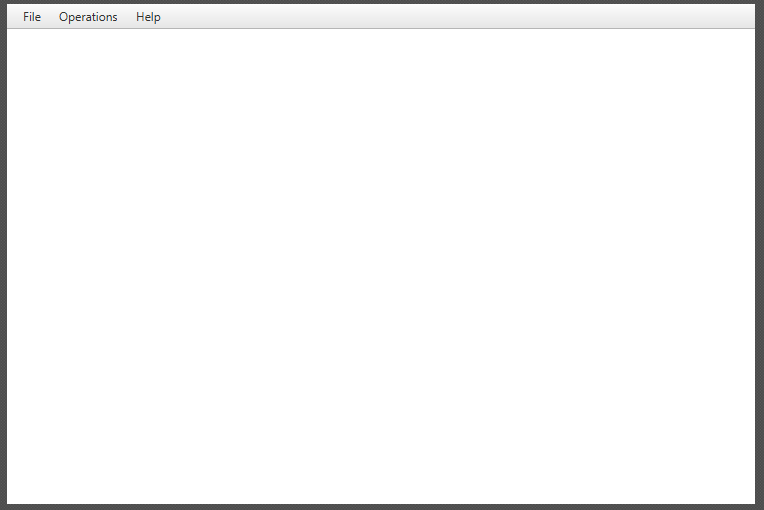
package sample.controller;  
  
*/\*\*  
 \* Created by Carlito on 12/04/2017.  
 \*/*import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Alert;  
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 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");  
  
  
  
 //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;  
 }  
 }  
  
 //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.  
  
 \*/  
 gender.setItems(artistGenderList);  
 type.setItems(artistTypeList);  
 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).

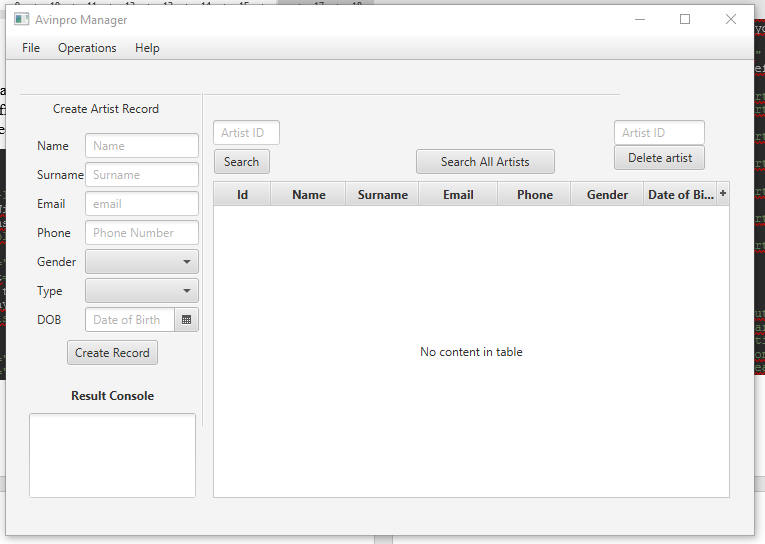
*/\*\*  
 \* Created by Carlito on 12/04/2017.  
 \*/*public class ArtistDao {  
  
 public static Artist searchArtist(String artID) throws SQLException, ClassNotFoundException {  
 //Declare a SELECT statement  
 String selectStmt = "SELECT \* FROM artist WHERE artistId= "+artID;  
  
 //Execute SELECT 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" prefHeight="25.0" prefWidth="104.0" text="Delete artist" onAction="#deleteArtist"/>  
 </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="102.0" prefHeight="317.0" 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="396.0" layoutY="70.0" mnemonicParsing="false" onAction="#searchArtists" prefHeight="25.0" prefWidth="139.0" text="Search All Artists" />  
 <Button fx:id="createRecordButton" layoutX="47.0" layoutY="261.0" mnemonicParsing="false" onAction="#createArtistRecord" text="Create Record" />  
 </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).

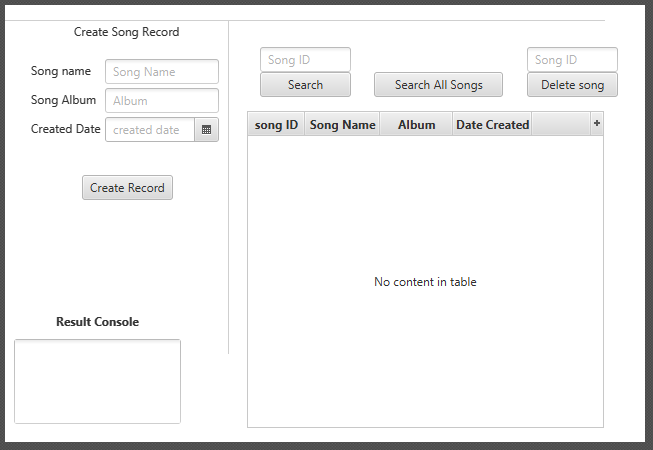
*/\*\*  
 \* Created by Carlito on 18/04/2017.  
 \*/*public class SongDao {  
 public static Song searchSong(String songID) throws SQLException, ClassNotFoundException {  
 //Declare a SELECT statement  
 String selectStmt = "SELECT \* FROM song WHERE songID= "+songID;  
  
 //Execute SELECT statement  
 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*();  
 }  
 }  
  
  
}