DOCUMENTATION

AppVideo

Michał Suliborski

U0200302

# Table of Contents

[1. Table of Contents 2](#_Toc92469856)

[2. Class Diagram 3](#_Toc92469857)

[3. UML interaction diagram 5](#_Toc92469858)

[4. Design decisions 6](#_Toc92469859)

[Graphical Use Interface 6](#_Toc92469860)

[Database result handling 7](#_Toc92469861)

[Database filter handling 8](#_Toc92469862)

[5. Design patterns 9](#_Toc92469863)

[Singleton 9](#_Toc92469864)

[MVC 9](#_Toc92469865)

[Dependency injection 9](#_Toc92469866)

[DAO 9](#_Toc92469867)

[6. Explanation of the components used. 10](#_Toc92469868)

[iText 10](#_Toc92469869)

[IntelliJ UI Designer 11](#_Toc92469870)

[VideoWeb and JCalendar problems 11](#_Toc92469871)

[7. Unit tests 12](#_Toc92469872)

[8. User manual 13](#_Toc92469873)

[9. Concluding remarks 24](#_Toc92469874)

# Class Diagram

On the figure below one can see the class diagram of the application generated in IntelliJ IDEA. It features only names of the class and dependances relation between them.

A screenshot of a computer

Description automatically generated with medium confidence

The program follows the MVC pattern, what is clearly visible on the diagram. Class AppVideo is a class containing the main method and its job is to create Model, View and Controller classes.

View class is responsible only for creating the interface and providing method to control it. No logic, nor data should be stored there. The idea is that this class should not even be aware of the existence of any other classes existing in the project. It consists of declaration of all UI elements, methods to access them (only getters, as we don’t want to allow other classes to modify those objects, only access them and using their methods change their properties) and constructor setting basic configuration of main panel.

Model class is the class responsible for both persistence and manipulation of the data. Its job is to provide high abstract methods to obtain and alter data. In this case, Model class creates and holds DAO classes that allow manipulating data of the database and Lists holding data directly used by View class. This usage is possible by implementing methods utilizing DAO classes’ methods.  
Model classes map the data from the database tables. They hold variables of each table’s column and implement getter and setter methods to allow access to them.  
Finally DAO classes themselves implement database manipulation methods. They utilize MySQLHandler class that connects to MySQL database and execute SQL queries and updates.

Controller class uses dependency injection to inject view and model class. This class is a connector between the two and implements logic of the whole application deciding what data manipulation needs to be executed at what time of the application runtime. It sets all buttons, comboboxes and lists listeners as well as manipulates the visibility of the view elements.

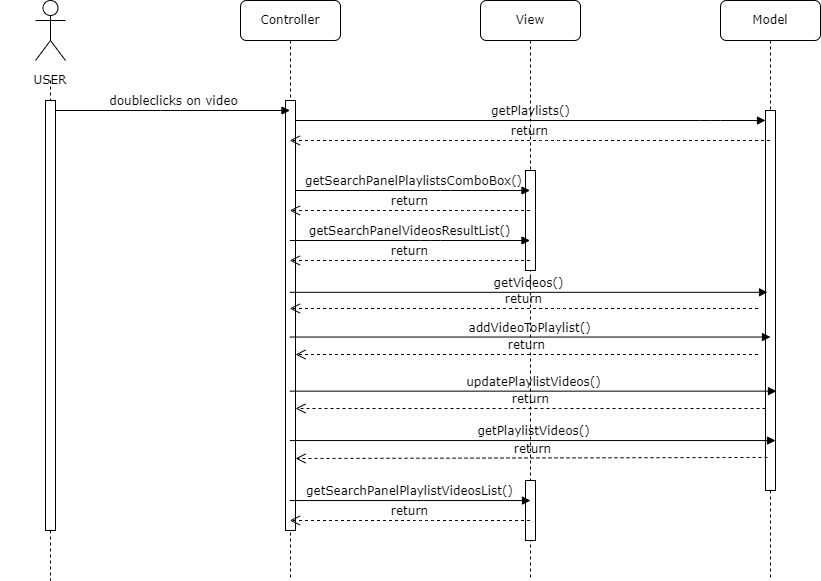
On the figure below one can see more detailed class diagram simply extended by methods implemented in each of the classes.

Diagram

Description automatically generated with medium confidence

# UML interaction diagram

Assuming that user has already navigated to New playlist panel, where it is possible to add new and modify existing playlists, selected desired playlist and searched for video that he wanted to add. The diagram will describe only the successful process of doubleclicking on video to add it to user playlist.



First, user doubleclicks on a desired video triggering the list listener. The listener makes sure the playlist JComboBox is not empty by calling getPlaylists(). Then it calls getSearchPanelVideosResultList() and getSearchPanelPlaylistsComboBox() from view to get index of doubleclicked video and targeted Playlist object. Next, it calls model’s getVideos() to get an ID of the video needed to be added. Having all that information, the listener calls model’s addVideoToPlaylist() to actually add the video to user’s playlist in the database and then updates the model’s playlist list by calling updatePlaylistVideos(). Finally, getPlaylistVideos() is called to get the playlist videos that now include newly added video and in the end calls getSearchPanelPlaylistVideosList() to update the view.

# Design decisions

## Graphical Use Interface

The graphical user interface has been implemented using IntelliJ Form, that is essentially an extension to standard Swing library offering, among others, custom layout box. It is fully integrated with IntelliJ IDEA editor and allows to create an application more easily than with traditional method.

One of core feature of this solution is choosing between generating the GUI into binary class files or to Java source code itself. In my case I’ve chosen the second option to make it compatible with maven package generation.

On the figure below there is a code fragment generated by IntelliJ during compilation having all the GUI classes declarations and definitions:  


## Database result handling

Another example of interesting design decision is handling data in Model part of the project.

Every query to the database requires an established connection, actual query string and handling of the result returned by the database to end up with populated object representation. In my case this has been achieved by first introducing a singleton class holding the connection to the database and allowing an access to it via static getConnection method:  
Text

Description automatically generated

Then each of the DAO classes methods gets this connection and holds a specific parameterized query to the database:  
Text

Description automatically generated

Finally, every query is handled by yet another method called handleObjectResut. As a parameter it receives a ResultSet and returns a List of Objects:  
Text

Description automatically generated  
This method has to be implemented separately by every DAO class, nonetheless it still makes it easier to handle the database result, as it can handle both single object result and multiple one. In case of multiple entries result, the whole list can be used, in case of need of a single result to be returned, method calling handleObjectResut can return just first object of the list.

This approach forces just a single connection to the database be present at a time and significantly speeds up the process of implementing new methods in the DAO object.

Database filter handling

As a last example of an interesting implementation, the global filter will be presented.

To filter the video a special condition needs to be added in WHERE clause in SQL statement. The getFilterString method takes Filter object as a parameter and generates this string based on the filter parameters, namely Id of tag to be excluded, minimum views number and maximum allowed title length.

A picture containing graphical user interface

Description automatically generated

Then this method is used during creation query statement in getVideosWithSearch method.

Text

Description automatically generated

This method works globally, thus it was also added in getMostPopularVideos, getRecentVideos and getVideosFromPlaylist methods.

Such implementation allows to easily add new filters with any combination of previously named parameters. It is also possible to extent the Filter by new fields to introduce new filter option, yet this would require modification of the database filters table as well as getFilterString method itself.

# Design patterns

## Singleton

This design pattern has been presented and explained in previous section of this document. It has been used in MySQLHandler class to make sure only one instance of the connection has been created.

## MVC

This design pattern was a requirement, and it has been implemented in the whole project.

This pattern has been more thoroughly described in section 2 of this document, but in a nutshell, this pattern requires division of the project onto three parts, model, view and controller. The idea is that both model and view do not know about their existence and controller is the class responsible for executing logic of the application and deciding when the model or view should change their state.

## Dependency injection

It is a technique in which an object receives other objects that it depends on, called dependencies, thus it is not responsible for their creation. Dependency injection has been implemented as part of following the MVC pattern:  
Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

## DAO

This pattern was also a requirement to implement in this project. It provides an object that is responsible for accessing the data from external source, in this case in was a MySQL Server running locally.

# Explanation of the components used.

## iText

First component used in this project is iText, a recommended component to be able to create a PDF of user playlist. In has been added through Maven by adding a dependency entry into pom.xml file.

It was implemented inside a function being called when user click on Print playlist button inside the Controller class as follows:  
Text

Description automatically generated

Resulted PDF is created in the root directory of the project (or in the same folder as .jar file that is being run) and it looks as follows:  
Graphical user interface, text, application

Description automatically generated

## IntelliJ UI Designer

This component has been utilized to create a GUI of the application and has been described more thoroughly in section 4 of this document.

On the figure below one can see the designer interface with finished application design itself:  
A screenshot of a computer

Description automatically generated

## VideoWeb and JCalendar problems

Due to technical problems most likely caused by using IntelliJ implementation of those required components has been omitted.

Instead of video being played in the application itself user has a button opening a default browser and redirecting user to YouTube page with that video increasing the view count at the same time. As a visual substitute to YouTube video player, a label with Image being a thumbnail of the video has been implemented.

Instead of a JCalendar component handling the birthday input used needs to provide a birthday in YYYY-MM-DD format handled by MySQL.

# Unit tests

Units test, unlike integration ones, are tests verifying correctness of methods of only one class. In case of this project, tests validating DAO classes has been implemented.

As an example, UserDAO class will be showed here. The test utilizes the same database as the program itself, thus the test must not alter data existing there already.  
Text

Description automatically generatedThe test first creates user, verifies if he can log into the program, makes sure all the fields have proper values set up during registration, alters its premium status and finally deletes him form the database.

Similar tests have been implemented for all DAO classes.

# User manual

In order to use the application, one first has to have a MySQL server running locally on port 3306 and have a ‘root’ user with ‘root’ password. On that server there also needs to be a database ‘appvideo’ created with adequate tables. An SQL script for creating the database has been attached.

To run the project from .jar file one needs to execute following command:  
java -jar .\AppVideo-1.0.0-jar-with-dependencies.jar com.suliborski.appvideo.AppVideo

Finally having the app opened we are greeted with login screen.

Graphical user interface, application, table, Excel

Description automatically generated

Being log out and trying to access any other panel than login and register will result in redirection to login panel.

User of course can also access registration and create a new account. Graphical user interface

Description automatically generated

After successful login user is given a success messageGraphical user interface

Description automatically generated

Then user is redirected to the recent panel, where he can see recently viewd videos Graphical user interface, application, Word

Description automatically generated

In explore panel user can search for videos with title

Graphical user interface, text, application

Description automatically generated

And easily combine it with tag selection on the right hand side

Graphical user interface, application, Word

Description automatically generated

User can also navigate to New playlist to view and modify existing or create a new one

Graphical user interface, application, Word

Description automatically generated

To create a new playlist user needs to provide a name of it and click Add playlist button

Graphical user interface, application, Word

Description automatically generated

If name was not empty user gets a message and new playlist is automatically choosen

Graphical user interface

Description automatically generated

In the same panel user can search for videos (this time with no possibility to find them with tags) and by doubleclicking on the result a video is added to the playlist. Doubleclicking on a video from playlist videos view will result in removing the video form playlist.

Graphical user interface, text, application, Word

Description automatically generated

User can also navigate to My playlist panel where any of his playlist can be selected and doubleclicking on a video will result in oppening video player panel

Graphical user interface, application, Word

Description automatically generated

Video player panel displays a thumbnail of the video, tags of the video, Play video in browser button and Add tag button

Graphical user interface, application, Word

Description automatically generated

User can add a tag to the video

Graphical user interface

Description automatically generated

This tag will be available in Explore panel and user will be able to search for videos with it

Graphical user interface, application, Word

Description automatically generated

At any time user can become a Premium user by clicking Become Premium button. No payment module has been implemented as instructed.

Graphical user interface, application, Word

Description automatically generated

Premium users can access to more feature of the application. One of them is a global filter in the top right corner that works in Explore panel. By default all users have a No filter filter applied that does not filter any videos.

Graphical user interface, text, application, email

Description automatically generated

Three additional filters has been implemented: No adult to filter out videos with tag adult, Only short titles to filter out titles longer than 16 characters and Only popular, visible on the figure below, to filter out videos with less than 5 views

Graphical user interface, text, application, Word

Description automatically generated

The filter is applied globally, meaning with no filter applied the My playlist view can display playlist Adult only having one video

Graphical user interface, application, Word

Description automatically generated

Having the filter No adult applied makes the playlist have no videos

Graphical user interface, application, Word

Description automatically generated

Premium users can also view 10 most viewed videos by all users in Most popular tab

Graphical user interface, text, application, Word

Description automatically generated

And finally they are able to generate a PDF file with a chosen playlist

Graphical user interface, application

Description automatically generated

# Concluding remarks

In conclusion, this project demonstrates usage of Java Swing library, still very popular library to create Java applications with graphical user interface, while implementing very common design patterns such as MVC, DAO or Dependency Injection.

Designing and implementing this solution was a tough challenge, especially that I did not have a big expertise in creating such application. All the work took a few weeks, during which the system class diagram had to be design, implemented and reworked as well as the database scheme with testing data. One also needs to add the time spend on creating the documentation, thus diagrams included in it. Overall, fairly accurate time estimate can be said to be over 120h of constant work.

It is also worth to mention, that this work was intended for 2 people groups, where in my experience, work efficiency can be increased more than twice, due to work spread and mutual verification and error correction. I am sure, that if I was working in a group, the GUI of the application would be richer, more response and application itself could have avoided missing features or potential uncaught bugs. Nonetheless, the application functionality has been met almost entirely and the code itself is clean and easily readable thanks to usage of comments and well knows design patterns.