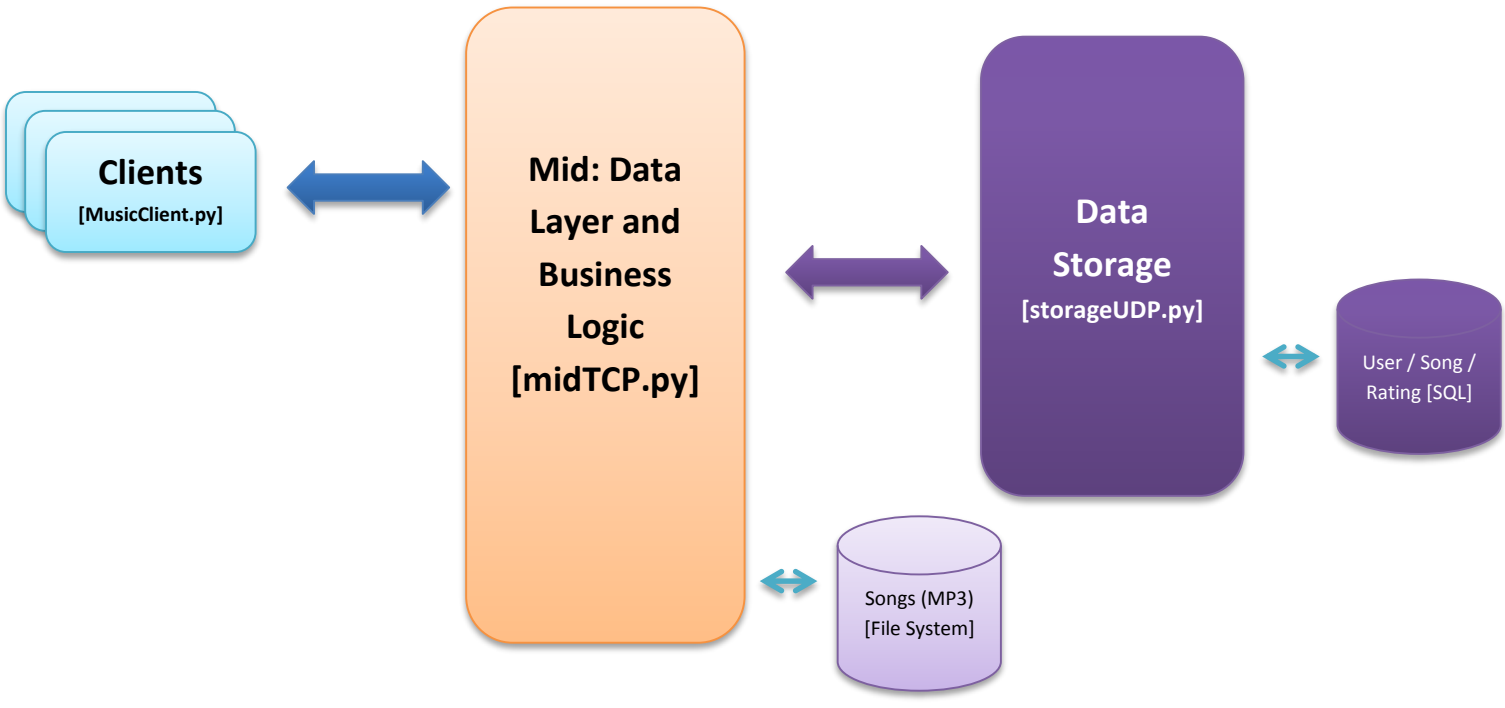
Cleiton Juffo

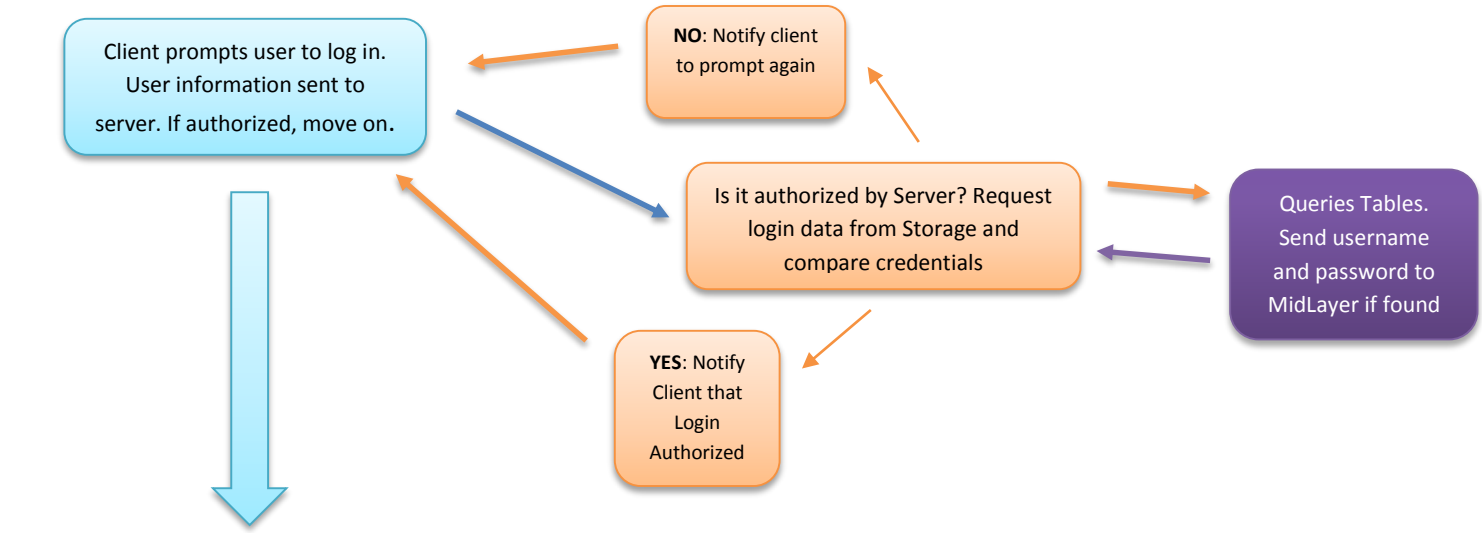
FINAL PROJECT: Music Client/Server

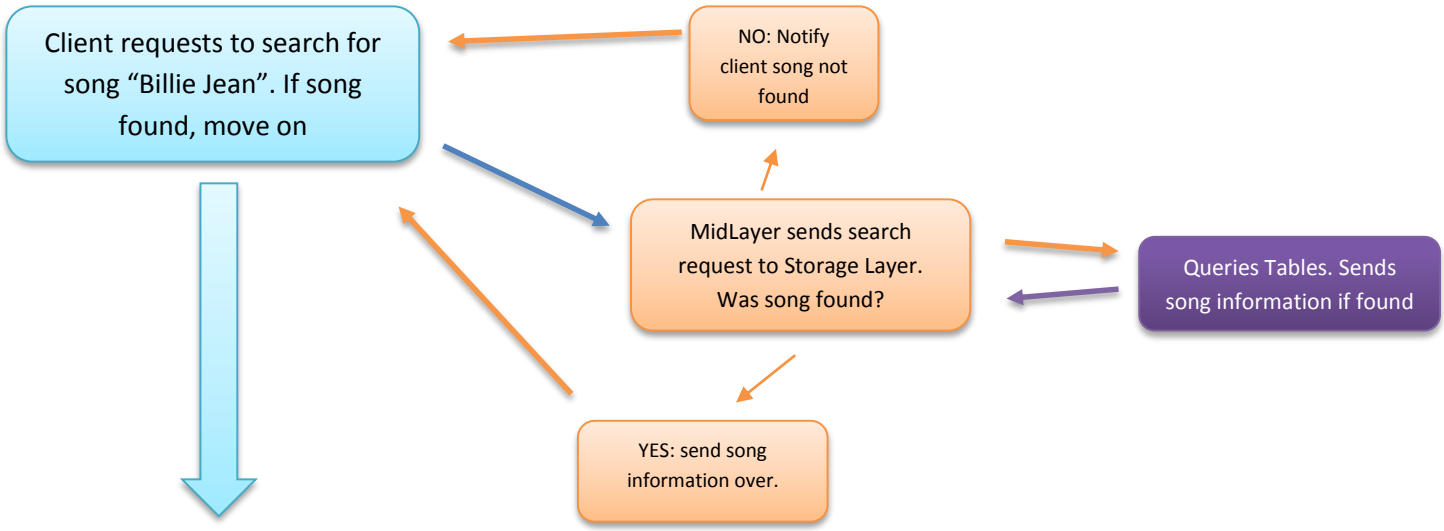
Overall Design

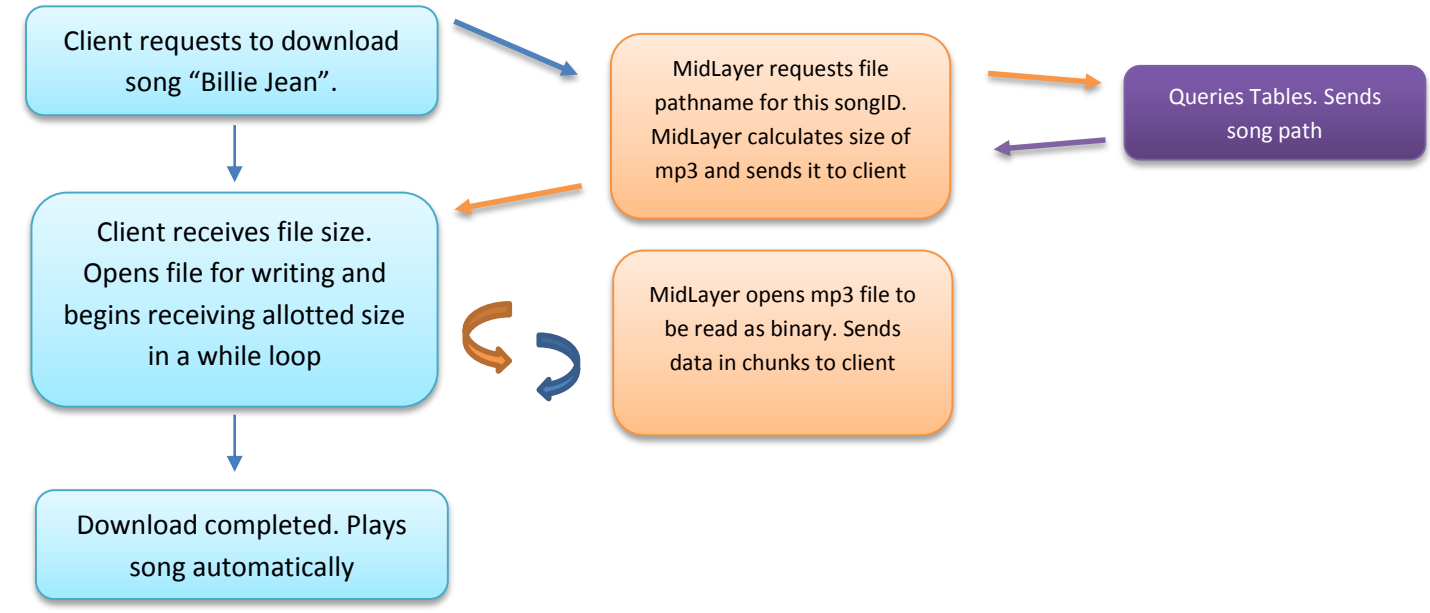
The project will involve allowing multiple clients to access the Data Business Layer by entering their username and password. Upon being granted access, the client will allow the user to search for songs through various queries (song title, artist, genre and rating), download a song to play locally, rate a song or display recently played songs. The Mid Layer will handle all requests from the client, and will access the Data Storage portion to verify information. The Data Storage will hold information relative to a user and for the entire catalog of songs. The Data Storage will not hold the actual song (MP3) but only a path. The songs will be stored in a database relative to the Data Business Layer, and the path returned from the Data Storage Layer will allow access to that song, which in turn will be sent over to the client. Upon searching for a song, the user can initiate a download by songID. Once the song is downloaded, it is automatically played by Windows Media Player and the program allows the user to rate the song. The Mid Layer allows multiple clients using multi-threading.



Sample Use Case: Downloading a song







Protocol Message Design

Brackets [] denotes actual string variable values

ACTION	CLIENT	MID TO STORAGE	STORAGE	MID TO CLIENT
Log in	USR::IN::[username]:: [password]	USR::IN::[username]:: [password]	USR::OK::[username] OR USR::ERROR::Login Not Authorized	OK:: LOGIN AUTHORIZED OR ERROR::NOT AUTHORIZED OR ERROR::ALREADY LOGGED IN
Log Out	USR::OUT:: [username]	USR::OUT::[username]	NULL	NULL
Search Title	SRCH::TITLE::[title]	SRCH::TITLE::[title]	[song(s)] OR ERROR|| NOT FOUND OR ERROR IN CHECKSONGID	[song(s)] OR ERROR||NOT FOUND OR ERROR IN CHECKSONGID
Search Artist	SRCH::ART:: [artist]	SRCH::ART::[artist]	[song(s)] OR ERROR|| NOT FOUND OR ERROR IN CHECKSONGID	[song(s)] OR ERROR|| NOT FOUND OR ERROR IN CHECKSONGID
Search Genre	SRCH::GEN::[genre]	SRCH::ART::[genre]	[song(s)] OR ERROR|| NOT FOUND OR ERROR IN CHECKSONGID	[song(s)] OR ERROR|| NOT FOUND OR ERROR IN CHECKSONGID
Search Rating >X	SRCH::RAT:: [number 1-5]	SRCH::ART::[1-5]	[song(s)] OR ERROR|| NOT FOUND OR ERROR IN CHECKSONGID	[song(s)] OR ERROR|| NOT FOUND OR ERROR IN CHECKSONGID
Search All	SRCH::ALL	SRCH::ALL	[song(s)] OR ERROR||NO SONGS FOUND	[song(s)] OR ERROR||NO SONGS FOUND
Initiate Download	DOWN::[songid]	GET PATH – SEE BELOW	GET PATH – SEE BELOW	[songsize]
Get Path	null	SRCH::ONE::[songid]	[song(s)] OR ERROR|| NOT FOUND	null
Download Song	OK	Null	Null	[song data in chunks of 10240]
Update Rating	RAT::[songid]:: [userid]::[rating value]	RAT::[songid]:: [userid]::[rating value]	OK::[songid]::[newrating] OR ERROR::NO SONG ID FOUND OR ERROR::NO RAT UPDATE OR ERROR::NO RAT INSERT	OK::[songid]::[newrating] OR ERROR::NO SONG ID FOUND OR ERROR::NO RAT UPDATE OR ERROR::NO RAT INSERT

SQL Queries

Checking if username already exists by counting instances of supplied username:

"SELECT COUNT(*) FROM Users WHERE Username=?",(user,)

Checking if songID already exists by counting instances of supplied userID:

"SELECT COUNT(*) FROM Songs WHERE songID=?",(ID,)

Verifying login information by retrieving password of supplied username:

"SELECT Password FROM Users WHERE Username=?",(username,)

Finding songs by given information:

*"SELECT * FROM Songs WHERE Title=?"*,(query,)
*"SELECT * FROM Songs WHERE Artist=?"*,(query,)
*"SELECT * FROM Songs WHERE genre=?"*,(query,)
*"SELECT * FROM Songs WHERE rating>=?"*,(query,)

Inserting rating:

"INSERT INTO Ratings VALUES(?, ?, ?)",(songID, userID, rating)

Updating rating:

"UPDATE Ratings SET rating = ? WHERE songID = ? AND userID = ?",(rating, songID, userID)

Retrieving average rating by selecting average from Ratings table, and updating Songs table:

"SELECT AVG(rating) FROM Ratings WHERE songID=?",(songID,)

"UPDATE Songs SET rating = ? WHERE songID = ?",(newRating, songID)

Source Files

NOTE: Denotes directory and python script location. Each component [client, mid, storage] will be in its own zip file.

Client: MusicClient/MusicClient.py
TCP/MT Mid: MidLayer/midTCP.py
UDP Storage: StorageLayer/storageUDP.py

Database Files

StorageLayer/music.db

Current table information inside music.db:

Users

	Username	Password
1	clayj	1234
2	jack123	12345
3	JarJarBinks	122
4	noob	987
5	helloW	555

Songs

	songID	Path	Title	Artist	Genre	Rating
1	111	111\\111.mp3	Billie Jean	Michael Jackson	Pop	3.16666666666667
2	122	122\\122.mp3	Dream On	Aerosmith	Rock	
3	133	133\\133.mp3	Nas Is Like	Nas	Rap	
4	144	144\\144.mp3	Superfreak	Rick James	Funk	
5	155	155\\155.mp3	BAM BAM	Sister Nancy	Reggae	
6	166	166\\166.mp3	To The Top	TOO MANY ZOOZ	Brass	
7	177	177\\177.mp3	Noda	TOO MANY ZOOZ	Brass	
8	188	188\\188.mp3	Get Busy	TOO MANY ZOOZ	Brass	
9	199	199\\199.mp3	F.W.S	TOO MANY ZOOZ	Brass	
10	211	211\\211.mp3	Maritza	TOO MANY ZOOZ	Brass	5.0

Ratings

	songID	userID	Rating
1	111	clayj	2.0
2	111	noob	2.5
3	111		5.0
4	211	clayj	5.0

Song Files

10 directories labeled by song id, each containing an mp3 in this format:

MidLayer/[songid]/[songid].mp3

Compiling Instructions

Songs will only play on Windows OS since the python script initiates that program.

Python 2.7 needs to be installed [use ninite.com for Windows executable). Download three all files and unzip. All scripts need to be executed in their own terminal or cmd window. SQLite should already be included with Python 2.7.

Window 1: $ python storageUDP.py
Window 2: $ python midTCP.py
Window 3: $ python MusicClient.py

Note that for Windows, python path must be set in the environment variables path in order to use python command in cmd. If Python 2.7 is already installed on windows, double clicking each python script will open up the python environment for each script.