Deliverable 2 Report

Table of Contents

Workplan

Each item should be completed between the previous due date and its actual due date. Effort ranges from 1 to 5, where 1 is the lowest effort (a quick formality) and 5 is the highest. Lowest priority items (indicated by their requirement's priority level) should not be developed past April 8.

Team members should work on items highest up in that week's due date section first, then continue on with lower items. Descriptions and priority level 1 requirements must be finished by each week's deadline. Lower priority requirements and tests may be worked on as long as the assigned team member has not worked more than six hours that week on this project.

Already done:

- Add Album [ML.A.01]

Due Feb. 28

Descriptions

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Architecture description	Deliverable 2	Evan	3	Feb. 28	3
Architecture block diagram	Deliverable 2	Evan	3	Feb. 28	1

Functionality and UI

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Create Playlist	ML.P.01	Nicky Desktop, Thomas Web, Pierre Mobile	4, 4, 3	Feb. 28, ,	4, ,

Add Location	HL.01	Nicky Desktop, Thomas Web, Pierre Mobile	4, 4, 3	Feb. 28, ,	4, ,
View Names of Locations	HL.01	Nicky Desktop, Thomas Web, Pierre Mobile	3, 3, 3	Feb. 28, ,	3, ,
Add Song to Playlist	ML.P.04	Nicky Desktop, Thomas Web, Pierre Mobile	4, 4, 3	Feb. 28, ,	4, ,

Test Writing

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Add Album Unit Test	--	Eva	2		
Create Playlist Unit Test	--	Evan	2	Mar. 5	2
Add Location Unit Test	--	Eva	2		
Add Song to Playlist Unit Test	--	Evan	2		

Due Mar. 4

Descriptions

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Detailed design description	Deliverable 2	Evan	4		
Detailed design class diagram	Deliverable 2	Evan	3		
Workplan update	Deliverable 2	Thomas	1		

Functionality and UI

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Assign Playlist to Location	HL.03	Nicky Desktop, Thomas Web, Eva & Pierre	3, 3, 3	Mar. 2, ,	3, ,

		Mobile			
Assign Album to Location	HL.03	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	3, 3, 3	Mar. 2, ,	3, ,
Assign Song to Location	HL.03	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	3, 3, 3	Mar. 2, ,	3, ,
Turn Volume Up or Down	HL.V.01	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	4, 4, 3	Mar. 2, ,	3, ,
Temporarily Mute Volume	HL.V.02	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	4, 4, 3	Mar. 2, ,	3, ,
View All Locations (currently playing song, volume level)	PB.U.01, HL.U.01, HL.U.04	Nicky Desktop, Thomas Web, Eva Mobile	2, 2, 2	Mar. 4, ,	3, ,

Test Writing

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Assign Playlist, Album, Song to Location Unit Tests		Evan	4		
Turn Volume Up or Down Unit Tests		Eva	2		
Temporarily Mute Volume Unit Tests		Pierre	2		

Due Mar. 11

Descriptions

			Expected	Completed	Actual

Item	Requirement	Assigned	Effort	Date	Effort
Unit testing	Deliverable 3	Thomas	4		
Component testing	Deliverable 3	Evan	4		

Functionality

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Next Song Plays When Current Song Finishes	PB.J.01	Thomas Web, Eva & Pierre Mobile	5, 1		
Delete Playlist from Library	ML.P.03	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	3, 3, 2		
Delete Album from Library	ML.A.03	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	3, 3, 2		
Delete Songs from Library	ML.A.04	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	3, 3, 2		

Test Writing

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Next Song Plays When Current Song Finishes Unit Tests		Evan	4		
Delete Playlist, Album, Songs from Library Unit Tests		Pierre	4		
Write first component tests		Eva	5		

Due Mar. 18

Descriptions

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Next Song Plays When Current Song Finishes	PB.J.01	Nicky Desktop	5		
System testing	Deliverable 3	Evan	4		
Performance/stress testing	Deliverable 3	Evan	4		
Work plan update	Deliverable 3	Thomas	1		

Functionality and UI

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Play/Pause	PB.02	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	5, 5, 3		
~~Skip Songs~~	~~PB.03~~	~~Nicky Desktop, Thomas Web, Eva & Pierre Mobile~~	~~5, 5, 3~~		
~~Jump to Start/End of Song~~	~~PB.J.02, PB.J.03~~	~~Nicky Desktop, Thomas Web, Eva & Pierre Mobile~~	~~5, 5, 3~~		

Test Writing

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Continue writing component tests		Eva	5		
Write first system tests					
Write Play/Pause unit tests		Thomas	4		
~~Write Skip Songs unit tests~~		~~Thomas~~	~~4~~		
~~Write Jump to Start/End of Song unit tests~~		~~Eva~~	~~4~~		

Due Mar. 25

Descriptions

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Release pipeline	Deliverable 4	Evan	2		
Work plan update	Deliverable 4	Thomas	1		

Functionality and UI

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Remove Song from Playlist	ML.P.05	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	3, 3, 3		
Change Order of Songs in Playlist	ML.P.02	Nicky Desktop, Thomas Web, Eva & Pierre Mobile	4, 4, 4		
~~Add Songs after Album Created~~	~~ML.A.02~~	~~Nicky Desktop, Thomas Web, Eva & Pierre Mobile~~	~~3, 3, 3~~		

Test Writing

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Continue writing system tests		Evan	5		
Begin writing performance/stress tests		Eva	5		
Write Remove Song from Playlist unit tests		Nicky	2		
Write Change Order of Songs in Playlist unit tests		Nicky	2		
~~Write Add Songs after Album Created unit tests~~		~~Pierre~~	~~2~~		

Due Apr. 1

Functionality and UI

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
View the position of the song currently playing in each location	PB.U.01	Nicky Desktop, Thomas Web, Eva Mobile	4, 4, 4		

Test Writing

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Continue writing performance/stress tests		Evan	4		
Finish writing component, system, performance/stress tests		Evan	4		
Write Persistence unit tests (similar to Assignment 1)		Eva	4		
Finalize unit tests (should have been covered already)		Pierre	4		

Due Apr. 8

Item	Requirement	Assigned	Expected Effort	Completed Date	Actual Effort
Draft Presentation	Deliverable 5	All of us	5		
Practice Presentation as group	Deliverable 5	All of us	2		
Slack time (in case features take more time than anticipated)		All of us	–		

Due Apr. 15

Item	Requirement	Assigned	Expected	Completed	Actual

			Effort	Date	Effort
Prepare and Submit Source Code	Deliverable 6	Evan	5		
Slack time (in case features take more time than anticipated)		All of us	–		

Architecture Description

Subsystem Descriptions

Our system is broken down into the following subsystems:

- View: Location View, Library View
- Controller: Location Controller, Library Controller
- Model
- Persistence

View

The view subsystem generates the graphical user interface (GUI). Through the GUI, it displays program data to and accepts input from the user. Upon receipt, the view subsystem passes this data to relevant controller components. Controller components can update view components with new information.

The view subsystem is further broken down into Location View and Library View – the former deals with listing and helping manage home locations and the latter deals with listing and helping manage the music library. These two 'sub-subsystems' communicate together to facilitate streaming music to a home location.

Controller

The controller subsystem accepts and verifies user input given through the view subsystem. Controller components store validated user input into model components.

The controller subsystem is further broken down into Location Controller and Library Controller. The Location Controller manages home location data while the Library Controller manages music library data.

Model

The model subsystem facilitates program data manipulation and retrieval. Collectively, model components hold all program data the application uses at runtime. The model is *completely independent* of the view subsystem.

Controller components both manipulate data in and retrieve data from model components.

Persistence

The persistence subsystem interfaces with an external storage system that does not lose its data when the application closes. Controller components interact with the persistence subsystem to save and load data held by model components

Architectural Patterns

We chose to apply the Model-View-Controller (MVC) pattern across the entire project.

Technical Reasons

The requirements and scope of this project dictate that this application is mostly about users viewing and then modifiying the data. It follows naturally that our main pattern should separate data from its view, since these are two significant and very different concerns. The MVC pattern separates these nicely.

Practical Reasons

Our team has the most experience working with this pattern. Since the instructors and TAs have spent the most time on this particular pattern, we feel they would be best positioned to assist us with this pattern as opposed to others.

Patterns across Applications

We decided to use the MVC pattern across all three applications (desktop, mobile, and web). The constructs of the Java language, Android framework, and PHP language allow for easy adherence to the pattern. There are no other foreseealbe reasons to add complexity by using a different pattern across the three applications.

Description of Detailed Design

In this section of text and the class diagrams that follow, we describe the detailed design of the model, view, controller, and persistence subsystems. We'll discuss the key classes in-depth:

- HomeAudioSystem
- LocationPage
- LibraryPage
- PersistenceHomeAudioSystem

HomeAudioSystem

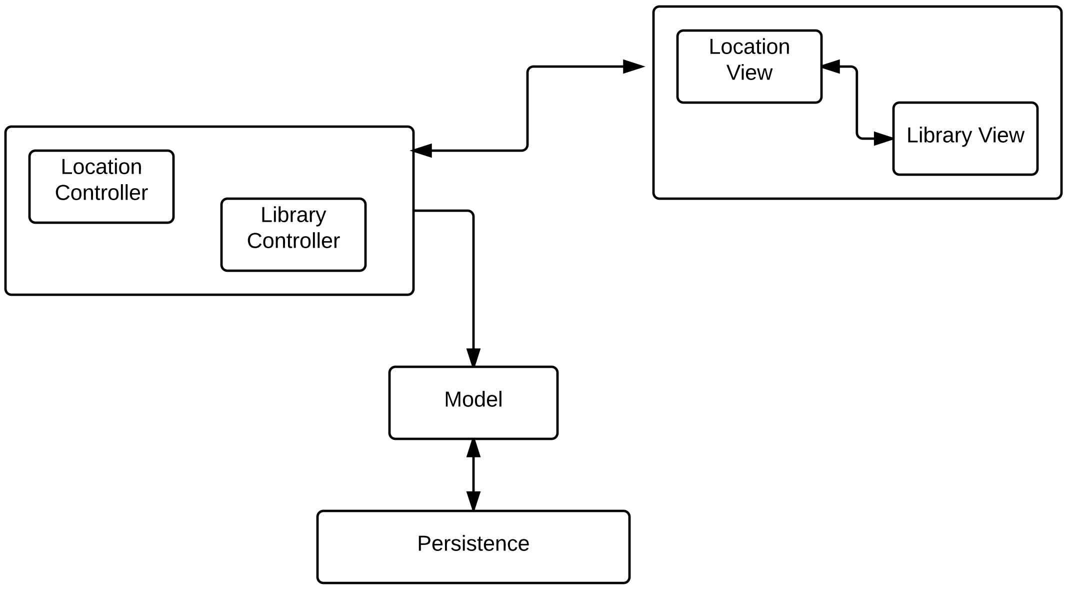
Contains instances of all the model classes for eventual storage and retreival through the persistence layers.

LocationPage

View component that allows the user (ultimately) to access all operations pertaining to locations, including adding music to stream.

LibraryPage

Architecture Block Diagram

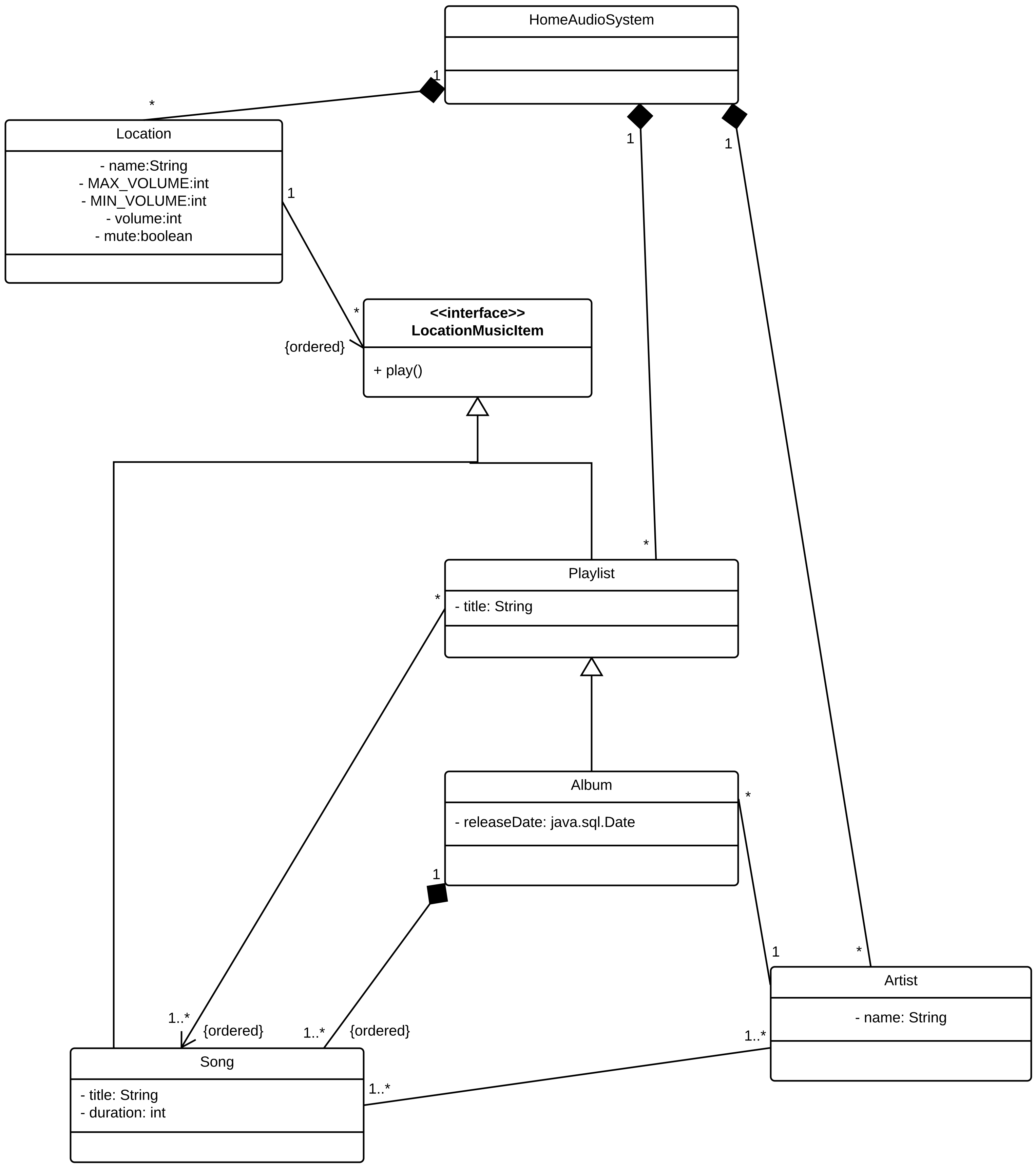


View component that allows the user (ultimately) to access all operations pertaining to the music library, including adding and deleting music.

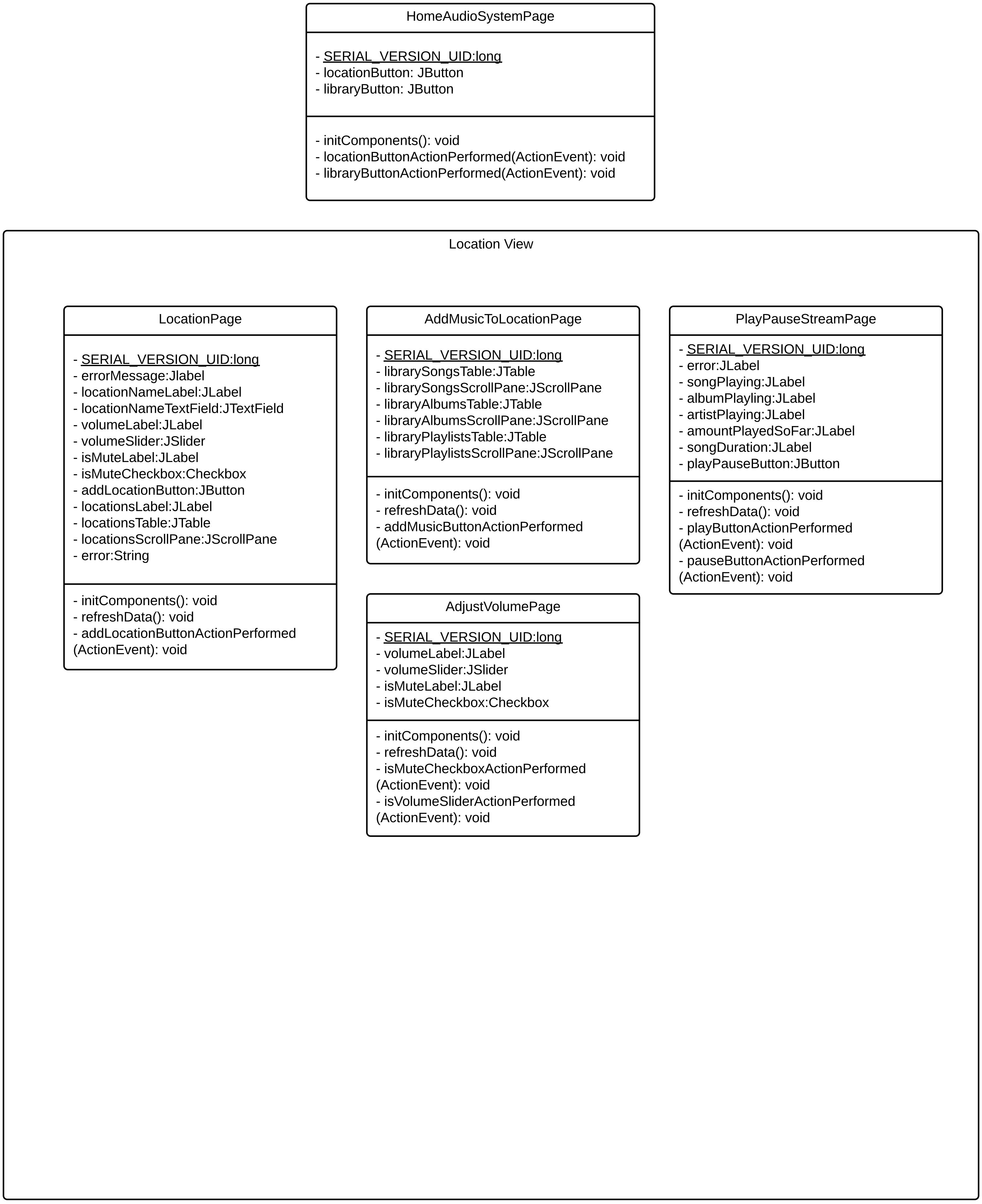
PersistenceHomeAudioSystem

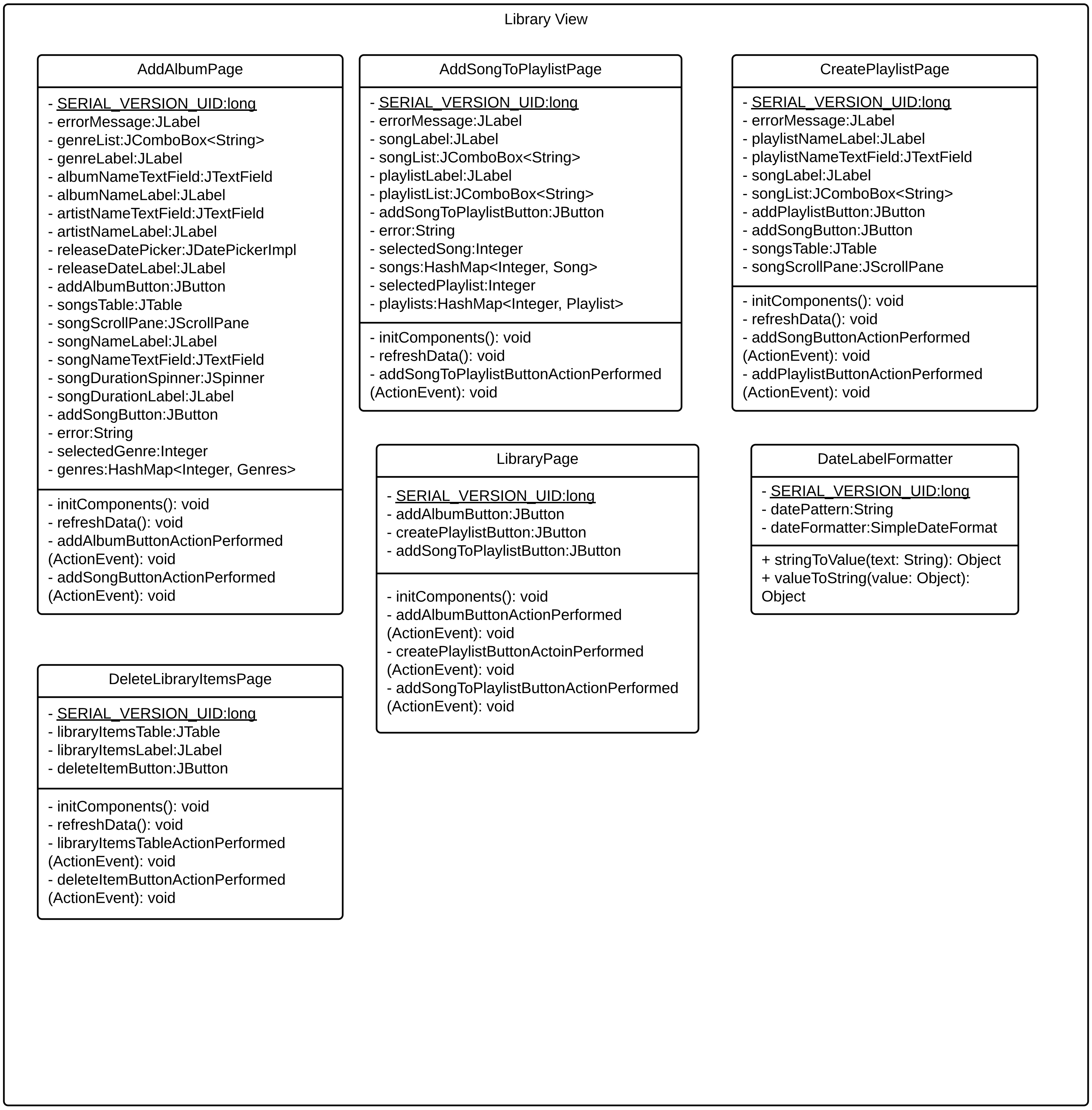
Main interface through which application data (via an instance of the HomeAudioSystem class) is retrieve on application start. It is one of two primary classes comprising the persistence layer.

Model Class Diagram

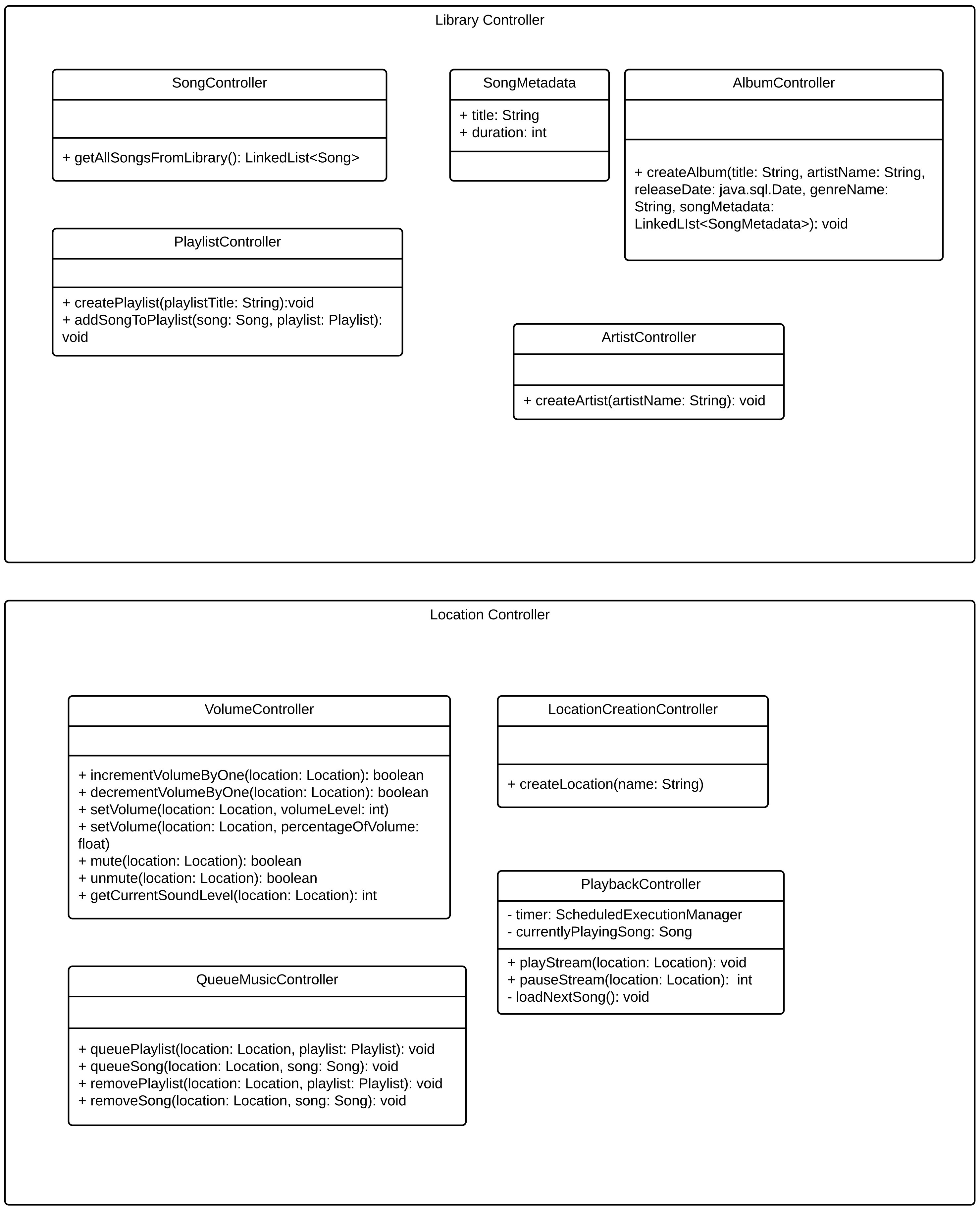


View Class Diagram

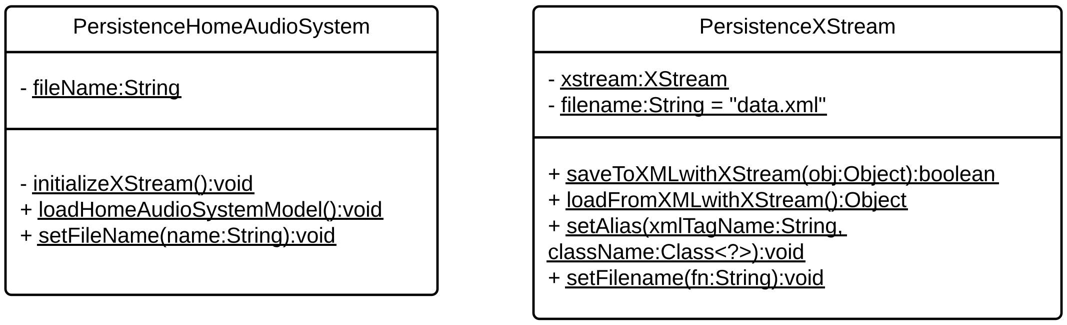

HomeAudioSystemPage
- SERIAL_VERSION_UID:long
- locationButton: JButton
- libraryButton: JButton
- initComponents(): void
- locationButtonActionPerformed(ActionEvent): void
- libraryButtonActionPerformed(ActionEvent): void
Location View
LocationPage
- SERIAL_VERSION_UID:long
- errorMessage:Jlabel
- locationNameLabel:JLabel
- locationNameTextField:JTextField
- volumeLabel:JLabel
- volumeSlider:JSlider
- isMuteLabel:JLabel
- isMuteCheckbox:Checkbox
- addLocationButton:JButton
- locationsLabel:JLabel
- locationsTable:JTable
- locationsScrollPane:JScrollPane
- error:String
- initComponents(): void
- refreshData(): void
- addLocationButtonActionPerformed (ActionEvent): void
AddMusicToLocationPage
- SERIAL_VERSION_UID:long
- librarySongsTable:JTable
- librarySongsScrollPane:JScrollPane
- libraryAlbumsTable:JTable
- libraryAlbumsScrollPane:JScrollPane
- libraryPlaylistsTable:JTable
- libraryPlaylistsScrollPane:JScrollPane
- initComponents(): void
- refreshData(): void
- addMusicButtonActionPerformed (ActionEvent): void
PlayPauseStreamPage
- SERIAL_VERSION_UID:long
- error:JLabel
- songPlaying:JLabel
- albumPlayling:JLabel
- artistPlaying:JLabel
- amountPlayedSoFar:JLabel
- songDuration:JLabel
- playPauseButton:JButton
- initComponents(): void
- refreshData(): void
- playButtonActionPerformed (ActionEvent): void
- pauseButtonActionPerformed (ActionEvent): void
AdjustVolumePage
- SERIAL_VERSION_UID:long
- volumeLabel:JLabel
- volumeSlider:JSlider
- isMuteLabel:JLabel
- isMuteCheckbox:Checkbox
- initComponents(): void
- refreshData(): void
- isMuteCheckboxActionPerformed (ActionEvent): void
- isVolumeSliderActionPerformed (ActionEvent): void


Library View
AddAlbumPage
- SERIAL_VERSION_UID:long
- errorMessage:JLabel
- genreList:JComboBox<String>
- genreLabel:JLabel
- albumNameTextField:JTextField
- albumNameLabel:JLabel
- artistNameTextField:JTextField
- artistNameLabel:JLabel
- releaseDatePicker:JDatePickerImpl
- releaseDateLabel:JLabel
- addAlbumButton:JButton
- songsTable:JTable
- songScrollPane:JScrollPane
- songNameLabel:JLabel
- songNameTextField:JTextField
- songDurationSpinner:JSpinner
- songDurationLabel:JLabel
- addSongButton:JButton
- error:String
- selectedGenre:Integer
- genres:HashMap<Integer, Genres>
- initComponents(): void
- refreshData(): void
- addAlbumButtonActionPerformed (ActionEvent): void
- addSongButtonActionPerformed (ActionEvent): void
AddSongToPlaylistPage
- SERIAL_VERSION_UID:long
- errorMessage:JLabel
- songLabel:JLabel
- songList:JComboBox<String>
- playlistLabel:JLabel
- playlistList:JComboBox<String>
- addSongToPlaylistButton:JButton
- error:String
- selectedSong:Integer
- songs:HashMap<Integer, Song>
- selectedPlaylist:Integer
- playlists:HashMap<Integer, Playlist>
- initComponents(): void
- refreshData(): void
- addSongToPlaylistButtonActionPerformed (ActionEvent): void
CreatePlaylistPage
- SERIAL_VERSION_UID:long
- errorMessage:JLabel
- playlistNameLabel:JLabel
- playlistNameTextField:JTextField
- songLabel:JLabel
- songList:JComboBox<String>
- addPlaylistButton:JButton
- addSongButton:JButton
- songsTable:JTable
- songScrollPane:JScrollPane
- initComponents(): void
- refreshData(): void
- addSongButtonActionPerformed (ActionEvent): void
- addPlaylistButtonActionPerformed (ActionEvent): void
LibraryPage
- SERIAL_VERSION_UID:long
- addAlbumButton:JButton
- createPlaylistButton:JButton
- addSongToPlaylistButton:JButton
- initComponents(): void
- addAlbumButtonActionPerformed (ActionEvent): void
- createPlaylistButtonActoinPerformed (ActionEvent): void
- addSongToPlaylistButtonActionPerformed (ActionEvent): void
DateLabelFormatter
- SERIAL_VERSION_UID:long
- datePattern:String
- dateFormatter:SimpleDateFormat
+ stringToValue(text: String): Object
+ valueToString(value: Object): Object
DeleteLibraryItemsPage
- SERIAL_VERSION_UID:long
- libraryItemsTable:JTable
- libraryItemsLabel:JLabel
- deleteItemButton:JButton
- initComponents(): void
- refreshData(): void
- libraryItemsTableActionPerformed (ActionEvent): void
- deleteItemButtonActionPerformed (ActionEvent): void

Controller Class Diagram


Library Controller
SongController
+ getAllSongsFromLibrary(): LinkedList<Song>
SongMetadata
+ title: String
+ duration: int
AlbumController
+ createAlbum(title: String, artistName: String, releaseDate: java.sql.Date, genreName: String, songMetadata: LinkedLIst<SongMetadata>): void
PlaylistController
+ createPlaylist(playlistTitle: String):void
+ addSongToPlaylist(song: Song, playlist: Playlist): void
ArtistController
+ createArtist(artistName: String): void
Location Controller
VolumeController
+ incrementVolumeByOne(location: Location): boolean
+ decrementVolumeByOne(location: Location): boolean
+ setVolume(location: Location, volumeLevel: int)
+ setVolume(location: Location, percentageOfVolume: float)
+ mute(location: Location): boolean
+ unmute(location: Location): boolean
+ getCurrentSoundLevel(location: Location): int
LocationCreationController
+ createLocation(name: String)
PlaybackController
- timer: ScheduledExecutionManager
- currentlyPlayingSong: Song
+ playStream(location: Location): void
+ pauseStream(location: Location): int
- loadNextSong(): void
QueueMusicController
+ queuePlaylist(location: Location, playlist: Playlist): void
+ queueSong(location: Location, song: Song): void
+ removePlaylist(location: Location, playlist: Playlist): void
+ removeSong(location: Location, song: Song): void

Persistence Class Diagram


PersistenceHomeAudioSystem
- fileName:String
- initializeXStream():void
+ loadHomeAudioSystemModel():void
+ setFileName(name:String):void
PersistenceXStream
- xstream:XStream
- filename:String = "data.xml"
+ saveToXMLwithXStream(obj:Object):boolean
+ loadFromXMLwithXStream():Object
+ setAlias(xmlTagName:String, className:Class<?>):void
+ setFilename(fn:String):void