Ryan Zante: 100824151
Samuel Shard: 100745640
Edmond Lee: 100523634

Music Player App

Overview

The proposed app to be built for the group project is a music player app. The app will allow users to add and load any audio/music files from their phone onto the app where it can be used for playback among other features. The app will store and sync audio and its metadata locally and online, through Firebase, which allows users to access and play their audio files on any device. In addition to having playback services available on the app, other features include login and registration used for syncing files and data online, playlist creation and management, song and album display and filtering, metadata lookup and storage from both local and online sources and the ability to share song and playlist details to various social media platforms. These are just some of the high-level features that will be included, more discussions on implementations and details of how the app will work will be discussed in the following sections.

Group Members and Responsibilities

Samuel will be in charge of user interface design. This includes directing the overall style of the app as well as the visual presentation of the Songs, Albums, Playlists, Player, Song Details and Settings pages. This also includes scrolling and other interactions such dragging or selecting items.

Ryan will be in charge of the backend including file storage, retrieving songs to be played or queued and editing playlists. This will include managing files stored locally on the device as well as online from Firebase. Ryan will also code the playback of music including pausing, fast forwarding and rewinding. Ryan will also code the metadata lookup to provide suggestions to automatically fill in missing song metadata.

Edmond will be in charge of the registration and login functionality to ensure accounts are created successfully. Edmond will design the database used to manage all accounts on the system. In addition, Edmond will manage the navigation between different screens. Edmond will also create the UI for the file explorer used to select local music files.

Functional Requirements

- Dialogs and pickers: Pickers will be used throughout the app including to select which songs to play, queue, or put into a playlist. Dialogue menus will appear when selecting media like a song including Song Info, Share, and other options. An on screen dialogue will also be used to login and register.
- Multiple screens and navigation: Users will be able to navigate between the Songs, Albums, Playlists and Player pages. They can also open the Song Info page for a given song and likewise for albums and playlists.
- Snackbars: Snackbars will be used to confirm that important actions have occurred such as creating and deleting playlists, importing or deleting songs and synching to the database. The snack bar will have an undo button similar to the unsend message option in Gmail.
- Notifications: When music is playing a notification bar will display the track that is playing along with controls to play, pause and skip the track.
- Local storage (SQLite): Songs stored locally on the phone can be accessed, plus new songs can be sync to local storage from Firebase cloud storage.
- Cloud storage (Firestore or other): Firebase will store online music for every user that has an account.

- HTTP requests: HTTP requests will be used to access the songs stored in Firebase. HTTP requests will also be used to fetch song information to automatically fill in missing song metadata such as artist name.

Non-Functional Requirements

- Data tables: The Song Details page will use a data table to display song metadata. The Album Details page will use a similar table.

Features and Sample User Interface WalkThrough

In the section below sample features along with the mockup of sample user interfaces will be discussed in more detail.

App Screen on First Load

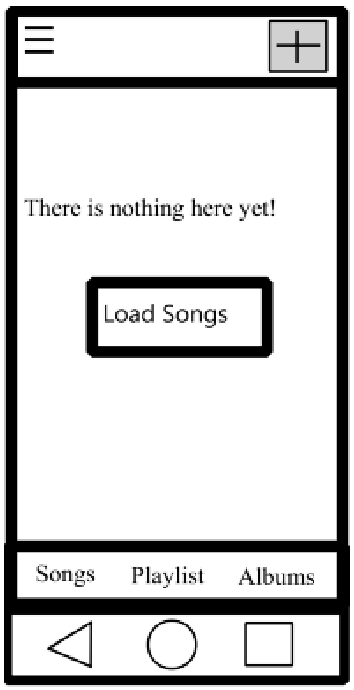


Figure 1

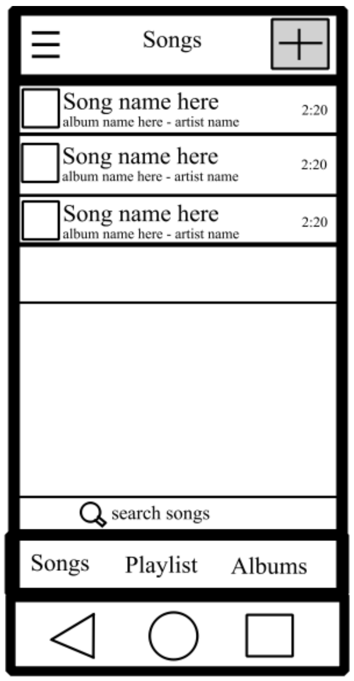


Figure 2

The figure above shows how the app will look upon its first loadout (Figure 1). On the top app bar there is a hamburger menu and add icon. There will be no audio files loaded on first use, so the Load Songs button will be shown. When that button is pressed, the app will open up a file picker and allow the user to search for music files on their phone. The user can select which files and directories they want added to the app which will then be saved on a local database for the app. They will also be able to select watched directories that the app can check and automatically add music files from. The songs will now appear in the app's song library. On the bottom there is a navigation bar that can take the users to the various pages which include a Songs page, a Playlists page and an Albums page.

The app does not require the user to register or login if they do not want to sync their files to the online cloud storage platform. The option is there if they decide to do so. From the Songs page, the users can press the add button on the top right corner to add songs to the app described in the previous paragraph (Figure 2).

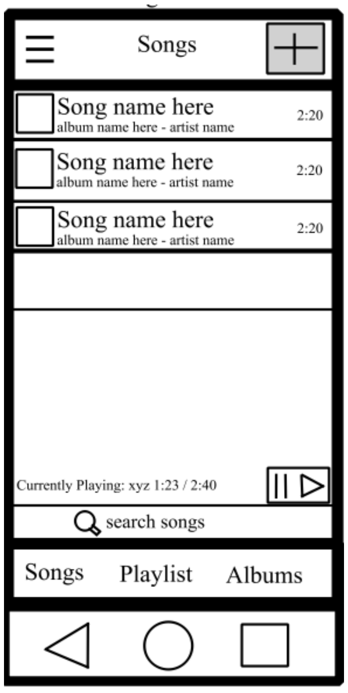

Songs
Song name here
album name here - artist name
2:20
Song name here
album name here - artist name
2:20
Song name here
album name here - artist name
2:20
Currently Playing: xyz 1:23 / 2:40
search songs
Songs
Playlist
Albums

Figure 3

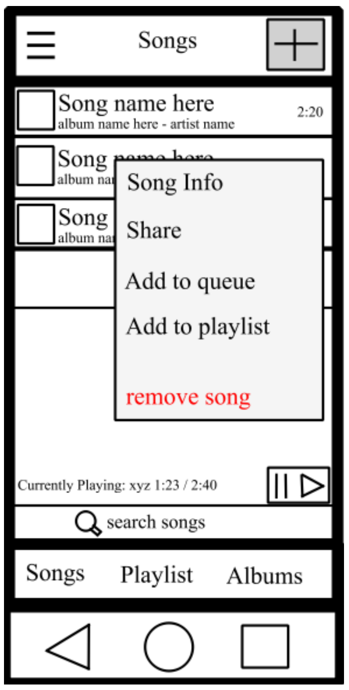


Figure 4

To play a file, the user just needs to click the tile of the song and a mini player will show up on the bottom (Figure 3). On the mini player, the user will be able to track the progress, pause, resume and play the current track. They can also filter songs by name using the search bar located above the navigation bar. The user can press and hold on a song tile to see a context menu with various options (Figure 4). In this context menu they can see Song Details, Share, Add to Queue, Add to Playlist and Remove Song options. If the Add to Playlist option is pressed, a mini menu will appear showing all the available playlists that the song can be added to. If the Remove Song option is pressed the song will be removed from the database, removed from the screen/library, and confirmed with a snackbar. If the Add to Queue button is pressed, the song selected will be added to the queue. If the user presses the mini player (not the pause and play buttons), the user will be directed to the Song Player page. If the user presses the Song Details button, they will navigate to the Song Details page displaying song metadata including artist, album name, cover art, and description, date released, and studio in the form of a data table. The user can edit the song details, and song options will appear based on the existing metadata. The user can pick the correct song and all missing metadata will be filled out. For example, if the only song information is "Thrift Shop", an HTTP request will be used to query a song information database and possible matches will be shown.

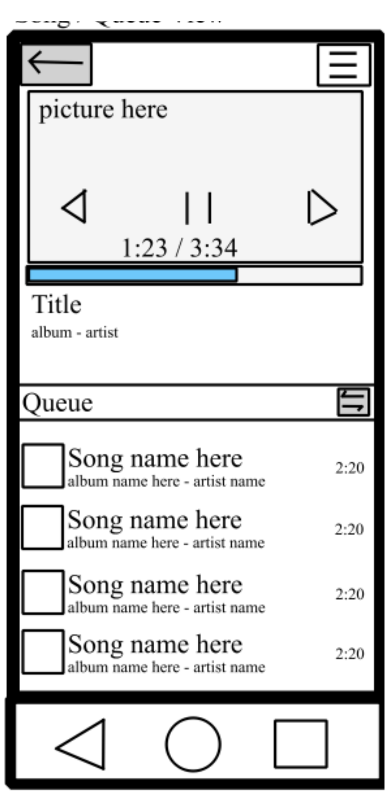


Figure 5

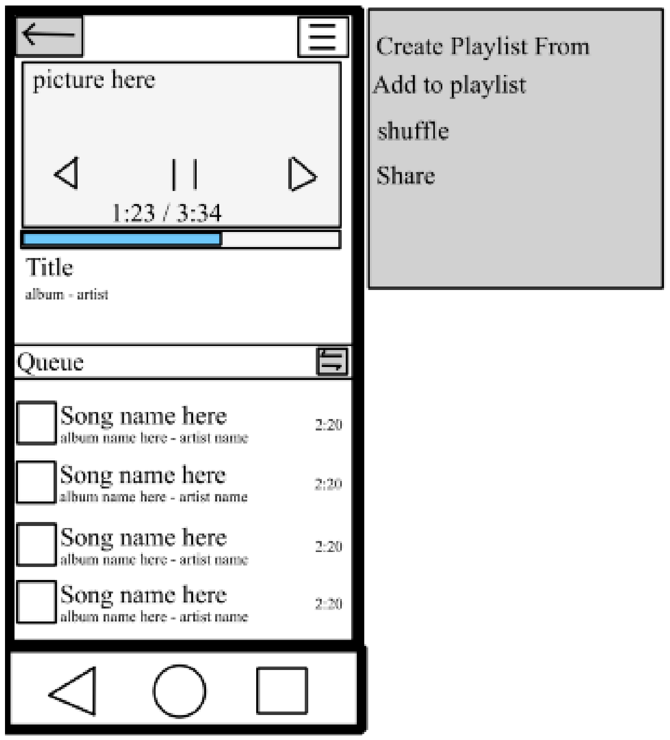


Figure 6

On the Song Player page (Figure 5), elements displayed include a music player that can be used to control the track, a picture of album art (if it exists), basic song information and a queue of songs that are to be played. The user can control the track with the music player controls, and is free to pause, play and skip tracks as desired. In the Queue section there is a button to reshuffle the songs in the queue when it's pressed, also if the user clicks a song in the queue section, that song will now be the current track that is playing. On the top of the screen there are two buttons, a back button and a hamburger menu. When the back button is pressed, the user is taken back to the previous page, and if the hamburger button is pressed a context menu will appear. In the context menu there are options to create a playlist from the current queue, add current song to a playlist, shuffle the queue and share the current song. If the Create Playlist From option is pressed, the user will be prompted to to enter a playlist name and to press save when ready. Once the user hits save, they will be taken back to the Songs Player page. If the Add to Playlist option is pressed, a mini menu will appear showing all the available playlists that the song can be added to. The user can check one or more playlists before hitting confirm/cancel to apply or abort the action. Upon completion a snackbar will confirm these actions.

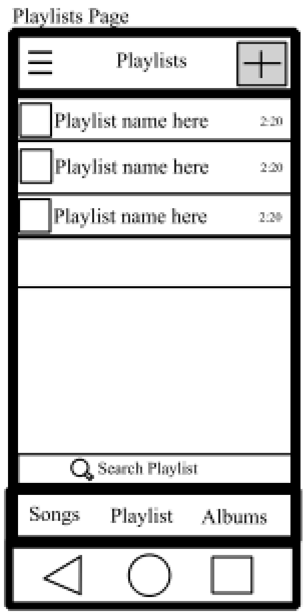


Figure 7

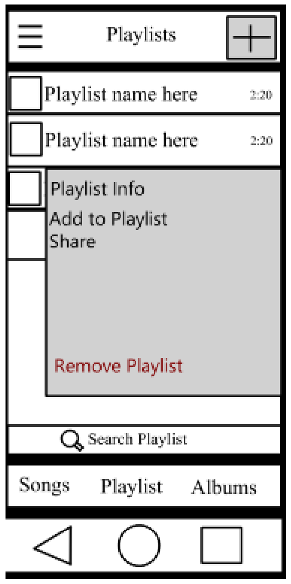


Figure 8

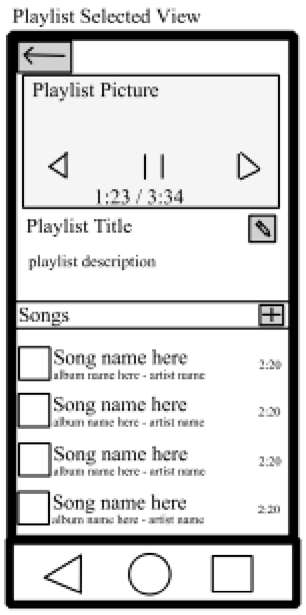


Figure 9

The Playlists page (Figure 7) will work similarly to the Songs page with some differences. Users can search for a playlist by name with the search bar on the bottom. If they press and hold playlist tiles, a context menu will appear where the user will have the following options: Add to Playlist, Share and Remove Playlist. If the Remove Playlist option is pressed, the selected playlist will be removed from the database and from the screen. If the add button (on the top right of the app bar) is pressed, the user will be prompted to enter a playlist name and will need to hit save to create the new playlist. Then the user will be taken to the Playlist Player page (Figure 9). The Playlist Player page will be similar to the Songs Player page (Figure 5) but will have slight differences. It will have controls to control the current track that is played and will have a display picture for the playlist (if it exists). Users can press the pencil icon to also navigate to the Playlist Details page to edit the playlist title and description. Next, they can view the available songs on the playlist and can click the plus icon to add new songs to the playlist. When the plus button is pressed, a mini menu will appear that will show the songs that are available to be added, once they are added to the playlist, the database will be updated locally. There is also a back button that when pressed, will take the user back to the Playlists page (Figure 7).

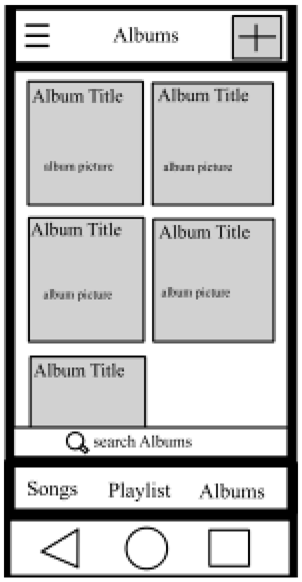


Figure 10

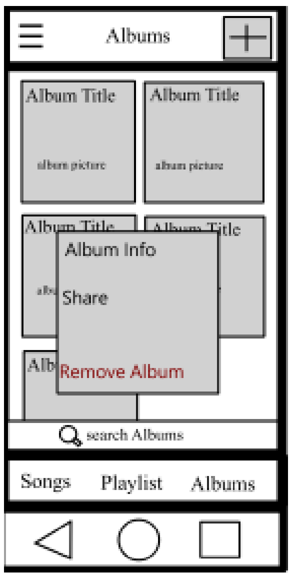


Figure 11

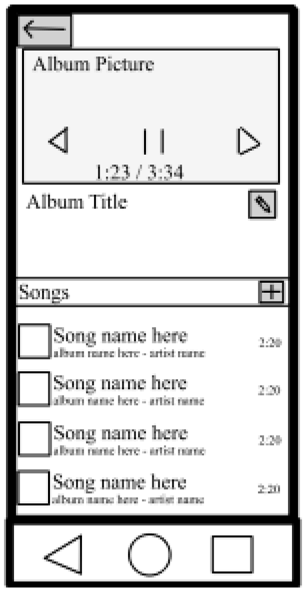


Figure 12

The Albums page (Figure 10) will work similarly to the Playlists page, but instead of having custom songs, songs that are part of an album will be automatically generated when they are added to the app as per the metadata (if it exists). The albums will be saved locally. If the user presses the plus button, they will be brought to a file explorer where they can add an entire album from their phone to the app. Once the album is added, a snackbar will appear to confirm.

Similar to the Playlists page, if the user presses and holds on the albums tiles, a context menu will appear with options of Album Details, Share and Remove Album, all of which be very similar to the verison on the Songs page. When Album Details is pressed it will take the user to the Album Details page (Figure 12), which operates very similarly to the Songs Details page showing the album's artist, cover, and more based on the available metadata. When the back button is pressed, it takes the user back to the Albums page.

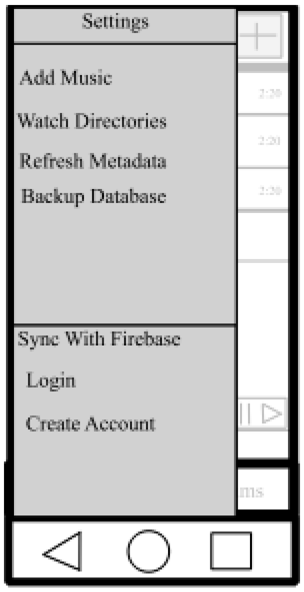


Figure 13

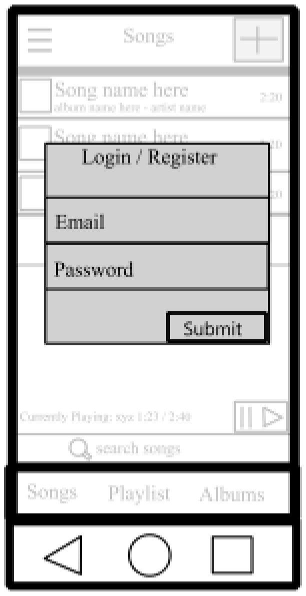


Figure 14

On the main screens (Figures 2, 7, 10) there is a hamburger menu on the top left of the app bar, which when pressed, a left slide out panel menu will be displayed that will contain some different options (Figure 13). If Add Music is pressed, a file picker will appear showing the phone's files where the user is free to browse and add files from the phone to the app. When Watch Directories is pressed, it allows the user to select directories which the app will monitor, when new files are added to the directory, the app will automatically import them into the app. When Refresh metadata is clicked the app attempts to find and fill in all the metadata of the

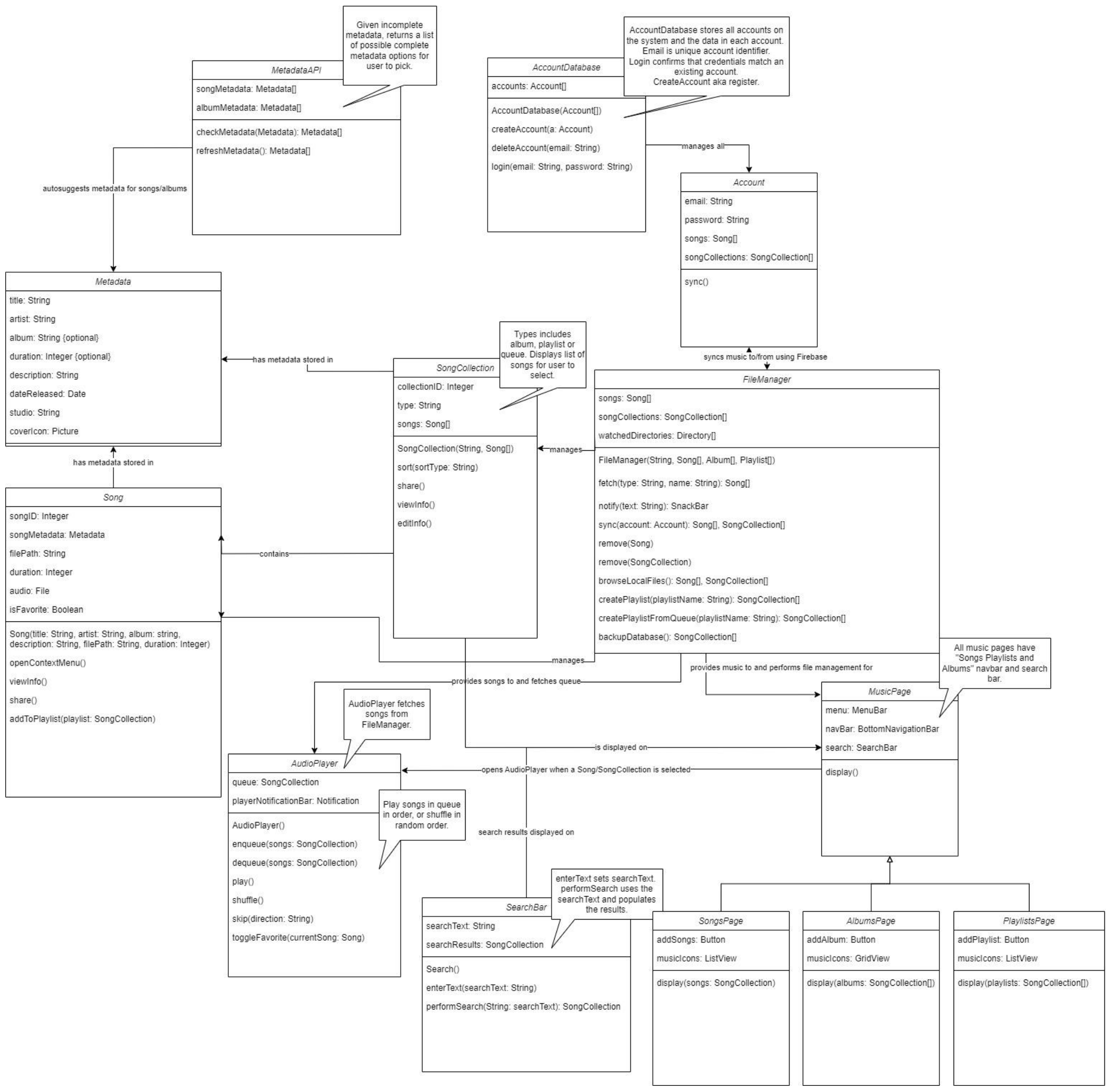
songs in the app using local and remote sources. When Backup Database is clicked, the local database which stores all the playlists, albums, and metadata will be exported to a location of the user's choice. In both cases on successful completion of the task, a snackbar will appear. If the user wants to sync everything to an online cloud storage platform they must be registered, logged in and must press the appropriate options to do so. When those are clicked they will be prompted to register or login with an email and password (Figure 14), all of which will be authenticated by Firebase. Now if the user is logged in and they want to sync all songs, playlists, and albums to an online cloud storage platform they can do so by clicking the Sync with Firebase option. A snackbar will appear to notify that the data was synced online successfully.
In terms of notifications, when music is playing a notification bar will display the track that is playing along with controls to play, pause and skip the track.

By default the app saves everything locally at first, and thus the playback of the music is from a local source. Should the user want to access their music on another device, they would first have to create an account to sync everything online. On the new device the playback will be from the online cloud storage platform and not a local source.

Code Design

In the section below a sample UML diagram is shown which helps to show how our code will be designed to implement the app. It will show the various classes, widgets, views, and controllers that will be implemented during the production of the music player app. The UML Diagram also shows the sample interactions between the classes and how they will be used to execute the desired functionality. (Getters and setters for each variable are implied.)

Music Player App


MetadataAPI
songMetadata: Metadata[]
albumMetadata: Metadata[]
checkMetadata(Metadata): Metadata[]
refreshMetadata(): Metadata[]
Given incomplete metadata, returns a list of possible complete metadata options for user to pick.
autosuggests metadata for songs/albums
AccountDatabase
accounts: Account[]
AccountDatabase(Account[])
createAccount(a: Account)
deleteAccount(email: String)
login(email: String, password: String)
AccountDatabase stores all accounts on the system and the data in each account. Email is unique account identifier. Login confirms that credentials match an existing account. CreateAccount aka register.
manages all
Account
email: String
password: String
songs: Song[]
songCollections: SongCollection[]
sync()
syncs music to/from using Firebase
Metadata
title: String
artist: String
album: String {optional}
duration: Integer {optional}
description: String
dateReleased: Date
studio: String
coverIcon: Picture
has metadata stored in
SongCollection
collectionID: Integer
type: String
songs: Song[]
SongCollection(String, Song[])
sort(sortType: String)
share()
viewInfo()
editInfo()
Types includes album, playlist or queue. Displays list of songs for user to select.
FileManager
songs: Song[]
songCollections: SongCollection[]
watchedDirectories: Directory[]
FileManager(String, Song[], Album[], Playlist[])
fetch(type: String, name: String): Song[]
notify(text: String): SnackBar
sync(account: Account): Song[], SongCollection[]
remove(Song)
remove(SongCollection)
browseLocalFiles(): Song[], SongCollection[]
createPlaylist(playlistName: String): SongCollection[]
createPlaylistFromQueue(playlistName: String): SongCollection[]
backupDatabase(): SongCollection[]
manages
has metadata stored in
Song
songID: Integer
songMetadata: Metadata
filePath: String
duration: Integer
audio: File
isFavorite: Boolean
Song(title: String, artist: String, album: string, description: String, filePath: String, duration: Integer)
openContextMenu()
viewInfo()
share()
addToPlaylist(playlist: SongCollection)
contains
manages
provides songs to and fetches queue
provides music to and performs file management for
All music pages have "Songs Playlists and Albums" navbar and search bar.
MusicPage
menu: MenuBar
navBar: BottomNavigationBar
search: SearchBar
display()
is displayed on
AudioPlayer fetches songs from FileManager.
AudioPlayer
queue: SongCollection
playerNotificationBar: Notification
AudioPlayer()
enqueue(songs: SongCollection)
dequeue(songs: SongCollection)
play()
shuffle()
skip(direction: String)
toggleFavorite(currentSong: Song)
Play songs in queue in order, or shuffle in random order.
opens AudioPlayer when a Song/SongCollection is selected
search results displayed on
SearchBar
searchText: String
searchResults: SongCollection
Search()
enterText(searchText: String)
performSearch(String: searchText): SongCollection
enterText sets searchText. performSearch uses the searchText and populates the results.
SongsPage
addSongs: Button
musicIcons: ListView
display(songs: SongCollection)
AlbumsPage
addAlbum: Button
musicIcons: GridView
display(albums: SongCollection[])
PlaylistsPage
addPlaylist: Button
musicIcons: ListView
display(playlists: SongCollection[])

Conclusion

The music player app project aims to provide a comprehensive music management and playback experience for users. With a strong focus on user interface design, backend development, and registration/login functionality, the app will empower users to enjoy their music collection on multiple devices. Additional features such as online synchronization, automatic metadata lookup and social media sharing will enhance the user experience. This project sets out to create a versatile music player app that caters to a wide range of user needs.

Sources

Flutter Documentation and in-class notes were used as reference.