DESIGN DOCUMENT

for

**Guitar Sheets**

Version *1.0*

Prepared by

Student Name(s): Thomas Yamasaki

Instructor: Prof. Thoms

Course: COMP 491 Capstone Preparation

Date: *12/13/19*

# System Background

## problem / need description

Guitarists of any level will learn how to play songs on the instrument with the use of sheet music. The sheet music can be composed in the standard notation (staff, measures and notes), chord notation or tablature notation. And in most cases, the sheet music will be printed and used as physical copies. As a guitarist builds their song repertoire, so will their collection of sheet music. This will lead to the guitarists having to carry what feels like “reams” worth of paper around just to practice the instrument. A significant portion of time will also be needed to gather the sheet music (creating it, searching the internet for it, printing it) and having to organize them in a coherent way. There is also the task of remembering what exactly to work on in a practice session. This can lead to having to write what needs to be accomplished on other pieces of paper or even the sheet music. This adds more to the pile of papers and can even cause confusion if written on the sheet music. The Guitar Sheets project seeks to take the stress away from guitarists by centralizing both the song repertoire catalog and a dedicated to-do list all in one app. This will allow the user to save time, money and most of all, allow them to enjoy the Guitar.

## System overview

Guitar sheets is a mobile device application developed for both iOS and Android devices. The main idea behind the Guitar Sheets app is to allow the user (a guitarist) to catalog their song repertoire and have it on their mobile device that way, the repertoire is always on hand. Another idea with Guitar Sheets is to allow the user to use this repertoire and other features in conjunction to aid in practicing songs. The goal of the Guitar Sheets app is to base the features around these main ideas while remaining as simple as possible. To achieve this goal, the features of the app are boiled down the main essentials needed to catalog a song repertoire and allow the user to use the entries and other features in a practice session.

Upon opening the app, the user will be greeted with the home screen which displays a to-do list containing user generated tasks. These tasks will allow the user to keep track of anything needed to be accomplished within the user’s next practice session. The user will be able to: add, edit, delete tasks within the to-do list.

The main feature of the app will be the Song List which will act as the “repertoire”. This is where the user can look at and access information stored on a particular song within the list. Users will also be able to: edit the contents of a song entry or delete selected song entry(ies).

When a user clicks on one of the song entries, its corresponding song entry page will be displayed and will contain all of the information that was previously stored by the user. The contents of a song entry page will include basic information on a song (title, artist, length, genre), the sheet music/tablature/chords that were stored, and any media (such as recordings) associated with the song. For the sheet music and media sections, a user can click on one of its entries and access the contents.

Users will be able to add entries to the song list via the song entry form. This form will have fields for the essential information along with some for the sheet music itself and any media associated with the song. The users can then click the save button and the new entry will be stored within the song list.

If the user does not have the essential information for a song on hand, they can use the song search feature to search the internet for the information/tabs associated with a song. The user can click on one of the results and the corresponding information will be filled on the song entry form, ready to be saved into the list.

There will be a tab that is specifically made for handling any accounts needed for the networked features. The users can log into all their accounts at once, so they do not have to keep logging in later. This will be used for the Google Drive sheet notes and the song search.

## User characteristics

**User: Guitarists**

The main demographic for the scope of this capstone project will be guitarists. And the guitarists can be of any level from beginners to seasoned experts of the instrument. The users can also be practicing independently or with the assistance of an instructor.

One use case would be for the user to build their song repertoire via the song list. The user will have several options when building their song repertoire in Guitar Sheets. One option is just for the user to add a song entry directly via the song entry form. In this case, the user will know the basic information relating to the song as well have their own sheet music to store with the song entry. The user will fill out the fields relating to basic information on the song. The user will then choose an option to store their sheet music. An example would be that the user has a set of sheet music stored on their Google Drive. The user will choose this option, log into Google when prompted and choose the desired document. Another case would be that the user wants to search the web for a Songsterr tab, they can choose that option and pick the desired result. The third case would be that the user has physical copies of their sheet music. The user will choose the option for photo scans and then choose to either take picture(s) or choose existing photo(s) from file system. The other option to add a song entry would be to start from the song search. This assumes that the user does not know most of the basic information pertaining to a song. The user will enter the name of a song and/or artist in the search fields and will then choose the desired result. After the user picks, they will be redirected to a partially filled out song entry form and pick up from the previous case.

Another use case would be to use features in Guitar Sheets to practice some songs. In one case, the user is practicing independently and will post some tasks to their to-do list for them to complete by the next practice session. During their next practice session, they will open the app and look at their to-do list. The tasks in the to-do list will have a concise title, the name of a song and a more detailed description. The user will then pick a task and look at what they have posted. The user will then go to the corresponding song entry in the song list. Then, the user will go to the sheet music that they stored with the song entry and start practicing. After the user feels like they have accomplished the task, they can go back to the to-do list and mark the task as completed. The completed task will then be taken off the list. They can then move onto other tasks on the list and/or add more tasks to the list. Another case can be that the user is practicing with an instructor. The instructor can give the user some tasks to work on before the next session. The user can then add these tasks to their to-do list and follow a similar procedure to the previous case.

One more use case would be that the user has to edit one or more of the song entries in their song list. One case would be that the user or app has made some sort of error with the song entry(ies). An example would be misspelling of some of the basic information of the song(s). The user will use the edit feature of the song list, fix the error, then resave the entry so that it reflects the changes. Another case would be that the user has new sheet music for the song. The user will use the song entry edit feature again and change the sheet music to the desired set and save the changes. The user will follow the same procedures for each song entry that needs to be edited.

## Product functions

**Home Screen/ To-do List:**

The To-do List feature will be the first view that a user will encounter when opening the Guitar Sheets application. Thus, the term To-do List will be used interchangeably with Home Screen. The main function of the To-do List is to allow the user to create and store tasks for reference within the next practice session. When the user goes to the view for the To-do List, they will be presented with a plus icon at the top to add tasks and a list of tasks to complete underneath. There will also be a way for the user to edit a task and mark task as completed. When adding tasks, the user will be presented with a form with fields for a title for the task, a name of a corresponding song (if applicable), and a description (if needed). When editing, the user will go back to the same form, edit and save the changes. One use case for the To-do List is for the user to use it to aide in them practicing the guitar. The user will open the Guitar Sheets app and look at the current tasks on the To-do List. They will pick a few tasks and go to the Song List, go to a Song Page and start practicing the song. After the task has been completed, the user will then go back to the To-do List and mark the task as completed. Another use case would be for the user to add and/or edit a task. Either way, the user will go to the task form, fill it out and save.

The nature of the To-do List in the Guitar Sheets app is mainly self-contained. Therefore, it does not really have any relationships with the other functional modules. Its main purpose is to provide a way for the user to keep and organize tasks for them to complete when practicing the guitar. As seen in the first use case above, the To-do List can be used indirectly with the Song List and Song Entries to aide with practicing the guitar. There is a field within the task form that will indicate which song the task was meant for. However, it does not create any logical references to the Song List itself.

**Song List:**

The Song List will be the list of song entries that the user has added. Upon reaching the view for the Song List, the user will see a plus icon at the top (redirects to the Song Entry Form). Underneath, will be a simple list of all the song entries that the user has saved into the app. The user can press on one of the “blocks” representing a song and access its corresponding page. The user will also be allowed to edit or delete an entry from the list. If the user chooses to edit the entry, they will be redirected to a “filled out” version of the form, can edit the form and save the changes. One use case would be for the user and or edit a song entry. In both cases, the user will be brought to a song entry form where they can fill it out and save to the list. Another case would be that the user does not know the information needed to fill out the fields of the form at the top of their heads. In this case, the user can go to the Song Search tab and search for the song using the internet. The user will then pick the desired song and the information found will be filled into the form. They can then fill out whatever is leftover and save the entry to the list.

The Song List pretty much has relationships with pretty much every other functional module in this section. As mentioned in the To-do List description, its relationship with the Song List is more indirect to be used to aid the user in practice. Song entries will be added to the list via the Song Entry Form module. Information of song entries can be found via the Song Search module and be redirected to the Song Entry Form. Users will also be able to access the information/media stored for each entry in the list via the Song Page modules.

**Song Page details:**

Song Page details refers to the section of the Song page containing the basic information on the song. This information includes title of song, artist, length, genre…etc. Upon reaching a Song Page, the user will see all this information formatted and listed at the top of the page. The user can edit this information via the edit feature in the Song List. The user will be brought up a filled-out version of the Song Entry Form containing whatever was previously saved. They can then edit the desired fields then save the changes to the list. One use case is that the user is going to perform a show. The user creates a task to create a setlist. They can then go to the Song List, access some Song Pages and see which entries fit their setlist according to the details section. Another use case would be again, the user does not know the information pertaining to a song off the top of their head. They will search for the details via the Song Search then pick a result. The result will be used to fill parts of the entry form then the user will save it to the song list.

The details section of the Song Page also has some relationships to the other functional modules listed. It can be used indirectly in some cases with the To-do List as seen in the first use case. The details will be added and can be edited via the Song Entry Form. If the user does not know the information for a song off the top of their head, they can search for it via the Song Search. The details can be accessed by the user via the Song List. The user will access the page for one of the entries from the Song List.

**Song Page media:**

Song Page media refers to the section of the Song page containing the sheet music, tablature, chords and audio recordings for a song entry. The sheet music/tablature/chords can be one or more of the following: Songsterr entry, Google Drive entry, Photo Scan. The audio recordings will be recorded by the user and save with the song entry. When the user enters a Song page, they will have the “details” formatted and listed at the top. Underneath, all the sheet music/tablature/chords will be listed below in its own section. The user can click on one of the sheet music entries and depending on its format, will be displayed accordingly. Below that, there will be a listing of all the audio recordings where the user can click on one the entries and will be able to hear the recordings. One use case for the media section of the Song Page would be where a user is working with an instructor and need to demonstrate “growth”. The user will have their tasks listed in the To-do list and will record several takes of a song in the recordings section. The user can then show the recordings to the instructor. Going on with practice, the user will also be using their sheet music/tablature/chords to practice the song. Another use case would be the user adding the media in with the song entry form. They will specify the desired format for the sheet music in the form and pick the entry to save in the list.

The media portion of the Song Page has relationships with several, but not all other functional modules listed. As seen with the first use case, the media portion will have some significant “indirect” relationships with the To-do list. Again, the two will be used together to aid the user in practicing the guitar. The media portion of the Song Page will be accessed via the Song List. The media will be chosen and saved in the list with the Song Entry Form.

**Song Search:**

Song Search will be the functional module that allows a user to search the internet for information regarding a song. Upon reaching the page for the Song Search, the user will be presented with some fields to fill out (artist, song title…etc.). Below the fields, there will be a button for the search to be initiated. The Song Search will send a request to an API such as Spotify for information on a song. Any results will be displayed, and the user can pick the desired result. They will be then taken to a partially filled Song Entry Form to be completed and saved to the Song List. The only real use case is that the user does not have the information needed to fill out the Song Entry Form on their own. The basic rundown has already been explained.

The Song Search is mainly self-contained and only has a direct relationship with the Song Entry Form. However, the Song Search does have an impact on the Details section of the Song page through the Song Entry Form. Whatever information is found in the Song Search and the User chooses will be saved in the Song List. This information will somewhat affect the user interaction with the to do list as seen in an earlier use case in the Song Page Details section.

**Song Entry Form:**

Song Entry Form will be the functional module that will allow the user to add an entry to the Song List. Upon reaching the page for the Song Entry form, the user will be presented with several fields to fill out. There are fields for the basic details of a song (title, artist, genre, length). Other fields will be included for the sheet music section of a song entry. The user can pick one or more of the formats (Google Drive entry, Songsterr entry, Photo Scan) and their corresponding entry to be saved in the song entry. After the form has been filled out, the user can click on the save button on the bottom of the form to save the entry in the Song List. It will also be used to display a “filled out” form of a pre-existing entry for editing purposes. The user can then edit any of the fields and then save the changes to the Song List. The most common use cases regarding the Song Entry Form have already been explained in previous sections.

The Song Entry Form has some relationship with all the other functional modules. The Song Entry provides the user with a way to add entries to the Song List feature. The Song Search can be used to find information to be filled into an entry form and be saved into the list. Most of the information (minus audio recordings) found in both parts of the Song Page were filled out by the user in the Song Entry Form. This information can be used in practice sessions alongside the To-do List feature.

**Accounts tab:**

The Accounts tab is the functional module responsible for handling any logins to accounts needed to access some online features. Upon reaching the accounts tab, the user will be presented a list of accounts that they can log into. The user can then click on the “block” representing an account and be prompted to log in. The user can then log in and be able to access the features corresponding to the account. The user will need to log into a Google account to access any Google Drive sheet notes saved in a song entry. The user will also have to log into the account needed for the Song Search. At this moment, the Spotify API will be used for the Song Search unless something else is discovered to be easier to user and/or more efficient.

As seen with the two use cases, the Accounts tab mainly affects two other functional modules. The Google Drive sheet notes feature of a song entry needs a Google account logged in to request and display the document. If the Spotify API is used for the Song Search, an account is needed to access the features of the API.

## Success Criteria

I have divided the features of the Guitar Sheets app into two categories: base and extra. The base features refer to those that are essential to hit the goals and ideas described in the system overview section. The base features include: To-do List, Song List, Song Page Details, Song Entry Form, Song Page Details, the Songsterr and Google Drive options for Song Page media and the accounts tab. Having these features will guarantee that the user is able to add, edit and access entries in their song list and its contents and be able to keep a to-do list of tasks for practice session. I would consider the Guitar Sheets app functional and successful if these features have been implemented. The extra features of the Guitar Sheets app refer to those that are not essential to hit the mentioned goals and idea but will provide some conveniences for the user. These features include: the Song Search and the photo scan option for Song Page media. These features do not directly impact the user having a song repertoire or to-do list. The Song Search will provide convenience for users who do not know the full details of a song if they want to add it to their song list. They can use the search feature to search the internet and find information. The photo scans will allow the user to take pictures of their preexisting sheet notes. If these features were to be implemented by the end of the semester, I would consider the system “extra” successful. They are not needed to fully demonstrate the main functionality but add more features to show.

# System Architecture

## System Architecture

The Guitar Sheets app is divided into modules that corresponds to the 6 main “views/pages”. These views are the Home Screen/To-do List, Song List, Song Page, Song Search, Song Entry Form and Accounts tab. In the earlier Product Functions section, the Song Page was further divided into two sub sections: Details and Media because they are two different sets of functionalities placed in a single view. These modules can then be broken down into the individual functions (e.g. edit song entry). The individual functions will be explained in the later Functional Requirements portion of the Detailed System Design Requirements section. There are also generalized “back end” functions to be described.

**System Decomposition Diagram**

A close up of text on a white background

Description automatically generated

**System Decomposition Diagram Continued:**

A close up of text on a whiteboard

Description automatically generated

**System Flow Diagram:**

**A close up of text on a whiteboard

Description automatically generated**

## SYSTEm Constraints

* Hardware or software environment:
  + This application will be developed using the Flutter SDK and Dart Programming language. Because these are new to me, the system has been designed as simple as possible to make the transition between Flutter and Android as seamless as possible. Flutter is new as well so it is limited in some features that Android may have.
* End-user environment:
  + This app is intended for use by any guitarist of any level. The features of the app have been boiled down to what seems absolutely necessary to keep a list of tasks and a song repertoire. For this reason and the one above, the app’s layout and features have been made as simple as possible. This prevents the implementation of say something that is too complicated for a beginner guitarist to use.
* Availability or volatility of resources:
  + The Song Search and Songsterr tablature features depend on what resources are available on the other end of their respective API’s. For example, Songsterr’s tablatures are created by the users of the website, so there may be some songs that may not have a set of tablature. There are multiple options that can be used to bypass these features (e.g. multiple formats for sheet music, search not required to add new entry).
* Standards compliance:
  + Some of the API’s will have request limits per user. Some of the backend code of the app will have to be designed in a way that it does not exceed the request limit and possibly get the user banned from using the API.
* Interoperability requirements:
  + This application will make use of a few API’s that return results in either a JSON or XML format. The backend of the app will have to be able to parse through the data and return the results to whatever feature is using it. The backend code for the app will have to be efficient enough to parse and return data in a reasonable time.
* Interface/protocol requirements:
  + The app will make use of several API’s and will have to be able to make requests and handle the data in an efficient manner. Some of the data will be used to pull up a “WebView” that displays the content to the user. The database table for songs will contain columns for some of the data obtained from the API call (e.g. Google Doc id). This will speed up the process of creating the “WebView”. The WebView will be using the https protocol to request and display the content.
* Data repository and distribution requirements:
  + Each copy of the app will have its own self SQLite database. This means that some features such as networked users cannot directly be implemented with this database. The database is not designed to be remotely accessed.
* Security requirements (or other such regulations):
  + Some of the API’s used will have to have the user logged into their respective accounts in order to be user. There will be a functional module (Accounts tab) that will allow the user to log into each account ahead of time.
* Memory and other capacity limitations:
  + The app will have to be designed so that it does not exceed the limits of the mobile device. The disk space used will have to be minimized to prevent filling up the user’s device.
* Performance requirements:
  + The app will have to access data (from its own database or request from an API) in a reasonable amount of time. The backend code will have to perform tasks as efficiently as possible. This is another reason why the features of the app and database have been designed to be as simple as possible.
* Network communications:
  + The app will make use of several API’s and other networked features such as creating web views for sheet music. Some of these features cannot be used without a mobile data or Wi-Fi connection to the mobile device. To alleviate this, the option to reference pictures of the sheet music has been included as a feature to be implemented.
* Verification and validation requirements (testing):
  + I am mainly going to be testing out this app next semester with the group guitar course. I will incrementally test with my practice sessions as features are added. There may need to be changes made in order to make the app more “usable” in practice situations.

## assumptions and dependencies

* Related software or hardware:
  + It is assumed that the user is installing the Guitar Sheets app on a fairly recent iOS or Android device. I am personally testing this app on an iPhone 8 device and emulator. It is also assumed that they have a decent internet connection when using the application. The user should also have a Google account for some of the features.
* Operating systems:
  + This application is created for both the iOS and Android operating systems. It is assumed that they have a recent version of the operating system installed on their device.
* End-user characteristics:
  + It is assumed that the user is using this app for the purposes of keeping a guitar song repertoire and for use with practicing the guitar. The use of the app depends on how many songs are in the user’s song repertoire. A beginner guitarist may not have as many songs under their belt as a guitarist with 20 years of experience.
* Possible and/or probable changes in functionality:
  + The Spotify API for searching songs may be too complex to keep up with the performance requirements of the app. Another API for searching information for song entries may be considered.

## Development Methods

I will be using and adapting the development methods outlined in the Flutter Crash Course. I am new to Flutter and when searching the web for tutorials, I came across YouTube videos from the Flutter Crash Course. The code used in the videos are simple and easy to follow. Most of the videos provide simple implementations of features I am including in this project that can be adapted to fit my needs. For features that are not in the Flutter Crash Course, I will follow tutorials from websites such as Medium.com. Their tutorials are also simple, easy to follow and can be adapted to fit the needs of the project.

References/Sample References:

**Flutter Crash Course:**

<https://fluttercrashcourse.com/>

<https://www.youtube.com/channel/UCRCpzcQz-t2ueVihCIx5jDg>

**Medium:**

<https://medium.com/the-web-tub/making-a-todo-app-with-flutter-5c63dab88190>

<https://medium.com/flutter-community/using-sqlite-in-flutter-187c1a82e8b>

# Detailed System Design requirements

## External interface requirements

**HTTP:**

HTTP will be the main networking protocol used throughout this project. It will be used to make request calls to the API’s as well as to display any web pages in web views. There is a package for the Dart language that handles all the calls to HTTP.

**Google Doc:**

The Google Docs API will be used to allow the user to store and view any sheet notes that are stored in their Google Drive. The API takes in 3 parameters: User identification, document identification and action. The action in the case of viewing the document will be “open”. The user will be allowed to log into their Google accounts through the app so they can access this feature.

**Songsterr:**

The Songsterr API will be used to find sets of guitar tablature to be opened in a WebView through the app. The API will allow querying their database using parameters for “song name” and/or “artist”. The API will then send a response in the form of XML or JSON. Either format will contain an id number which can be appended to their “generic” URL. This URL can then be stored in the Song table of the database. The URL can then be used to display the tablature web page in a WebView.

**Spotify:**

The Spotify API will be used to find any “detail” related information regarding a song. A call to the API will return the results in JSON format. There is a package for the Dart language which can be used to handle calls to the Spotify API. This package is user created so may not have functions to handle all of the features of the API.

## Functional requirements

A close up of text on a black background

Description automatically generatedA close up of a piece of paper

Description automatically generated

**Home Screen/To-do list:**

*Add Task*: Allows the user to add a task to the To-do list. Users will be presented with a form to fill out for the task with fields for: task title, task song, task description. The user can press save and the task will be added to the Task table in the database and will be displayed in the list.

*View Task*: Allows the user to view the full details of a task in the list. When the user presses on a task, they will be presented with a page that formats and lists the details of that task. These details will be queried from the Task table in the database.

*Edit Task*: Allows the user to edit a pre-existing task. The user will be taken to a filled-out version of the form and can make the desired changes to the task. The changes can then be saved into the database.

*Mark Task Complete*: Allows the user to mark a task as completed. The task will be deleted from the database and from the view of the list.

A close up of text on a white background

Description automatically generated

**Song List:**

*Add Song Entry*: Allows the user to add a song entry to the Song List. The user will be taken to the Song Entry Form to fill out and save to the Song table of the database. Full details regarding the Song Entry Form will be discussed later

*View Song Entry*: Allows the user to view the details and media for a song entry. The user will be taken to an entry’s song page with its contents formatted and displayed. Contents will be queried from the Song and Media tables from the database and displayed in the Song Page. Full details regarding the Song Page details and media will be discussed later.

*Edit Song Entry*: Allows the user to edit the contents of a pre-existing song entry. The user will be taken to a “filled out” version of the Song Entry Form. The user can then make the desired changes and save them to the Song table in the database. The pre-existing row in the table will have to be overwritten.

*Delete Song Entry*: Allows the user to delete a song entry from the Song List. The row representing the entry in the Song Table will be deleted. All rows in the Media table that are referenced will have to be deleted as well.

A close up of text on a white background

Description automatically generated

**Song Page details:**

*Format/Display details*: This will be done automatically upon the user clicking on a song entry from the list. The function will query the Song table in the database and will collect the “details” portion (title, artist…etc.) and will format and display them at the top of a Song Page.

**Song Page media:**

*View Sheet Music*: Allows the user to view the sheet music that they have saved with a song entry. The sheet music will be displayed in a view according to the format. Google Docs entries will require a Google account sign in. There will be a call made to the Google Docs API to display the document in a WebView. Songsterr entries will make an https call using the URL and display the sheet notes in a WebView. Photo Scans will be referenced using the stored URI and displayed in a view.

*Play Audio Recording*: Allows the user to listen to their audio recordings stored with a song entry. The audio recording will be located with the URI stored in the Media table. The audio recording will then be played in a separate view containing an audio player.

*Add Audio Recording*: Allows the user to add a self-made audio recording to a song entry. The user will be taken to a view containing an audio recorder and be prompted to record. Data regarding the new audio recording will be stored into the Media table in the database. A reference to the audio recording will be made in the Song Media intersection table.

A close up of text on a white background

Description automatically generated

**Song Entry Form:**

*Fields*: provide the user a way to input the information needed to be stored in the Song table of the database. Fields will be provided for details of a song entry (title, artist…etc.) and the sheet notes of the media portion.

*Add Sheet Music*: Allows the user to add a reference to some sheet music for the song entry. Function will differ according to the format. For Google Docs, the user will have to be logged into a Google account. A view containing the contents of the user’s Google Drive will be opened. The user can then choose the desired document and data for the document will be collected. For Songsterr, a view for a search will be shown (similar to Song Search). A call to the Songsterr API will be made. The user can choose a result and a URL for Songsterr will be constructed. For photo scans, a view containing the contents of the user’s camera roll will be shown. The user can then choose the desired picture(s). In this case, references to all of the pictures must be stored in the Media table and then a reference created in the Song Media table.

*Save Song Entry*: Allows the user to save the song entry into the database tables. The data inputted into the fields will be saved into their corresponding columns in the Song table. If any photo scans were added, their data must be saved into the Media table and then a reference must be made in the Song Media table.

*Cancel/Discard Changes*: Allows the user to exit the Song Entry Form without saving anything to the database. The user will be taken back to the Song List.

**A close up of a piece of paper

Description automatically generated**

**Song Search:**

*Fields*: provide the user with a way to input parameters to be searched. Fields will be included for the Song’s title and artist to be used for an API call to Spotify.

*Initiate Search*: Using the parameters that were entered into the fields, a call will be made to the Spotify API. We will be searching their database for information regarding a “track”. Any results will be returned and will have to be parsed and displayed.

*Display Results*: The data that was returned from the API call will be parsed and then displayed in a list. There will be a counter for the number of returned results. The number of results will be displayed first then followed by clickable entries for each of the results.

*Choose Result*: Allows the user to select a result to add to the song list. Any data found for the result will be further parsed and added into its respective field in the Song Entry Form.

A close up of text on a white background

Description automatically generated

**Accounts tab:**

*Log into account*: Allows the user to log into each of the accounts listed in this view. The user will be taken to another view for each account where they can enter their credentials and be logged into the account. Is needed for some features such as accessing Google Docs and requesting data from some API’s like Spotify.

*Sign out of account*: Allows the user to log out of the desired account. The user will click a button for logging out. The account will then be logged out of automatically.

\*\*\*These functions are more generalized descriptions and may have to be made more specialized later on.

**General Database:**

*Create Table*: Will create a selected or all of the tables in the SQLite database. This function is mainly used when the user opens up the app for the first time.

*Drop Table*: Will drop selected or all tables from the database. I believe this function will be used primarily for testing purposes during the development process.

*Query Table*: Will query a specific table for some or all of the data contained in a row. Will be used when displaying contents of To-do and Song lists.

*Add Row*: Will add a row with data to a specific table. Will be used extensively to add entries in the To-do and Song lists.

*Update Row*: Will update the contents of a specified row. Will be used to edit tasks in the To-do List, edit song entries in the Song List. Will also be used to add media such as audio recordings to a song entry.

*Delete Row*: Will delete a row from a specified table. Will be used to delete entries from To-do and Song lists.

*Create Intersection*: Will create an intersection between two tables and store into an intersection table. Will be used mainly to create a reference from a song entry to one or more media entries.

**General WebView:**

*Display WebView*: Will take in a URL to a webpage and display it in a special view. Makes a call to HTTPS to request the web page and display it.

*Close WebView*: Closes down an opened WebView and returns the user to the previous view.

**General API calls:**

*API call with results*: Will make a call to a specific API and gather the results into an object that can be iterated/parsed. Will be used in Song Search as well as searching for Songsterr tabs.

*API call without results*: Will make a call to a specified API to perform tasks such as logging into account or displaying a document.

*Parse Results*: Will take in an object representing the results from an API call and parse it so it can be used appropriately in some of the functions/features listed previously.

## Performance requirements

**Efficiency:**

The app will have to perform tasks in an efficient manner, or the user might consider the app “laggy” or “unusable”. When making API calls that return responses, the data from the responses will have to be parsed and handled within a reasonable amount of time. The app will also have to perform database tasks in a reasonable amount of time.

**Memory:**

Because this is a mobile app, it will have to use a reasonable amount of storage space on the device. The app does not have the luxury of being stored on a device such as a pc. That is why for online features such as the Songsterr tablature, only the URL is stored and not the actual data from the website.

**Response time:**

Going along with efficiency, the app will have to have reasonable response times. The response times for API calls also depends on the end user’s network speeds. The app will have to work with whatever network speeds and create appropriate actions in a reasonable amount of time.

## Design constraints

**Compliance to standards:**

The app will be utilizing several API’s for some of its features. The API’s that return data responses will have “rate limiting”. These limit the amount of calls a user can make within a set period of time. The app will have to be implemented so that it follows these rate limits. Exceeding the rate limits can cause a user to get banned from using the API. Some API’s also require user authentication. Logins will have to be implemented so that the user can access the API features.

**Interoperability:**

Some of the API’s will return data responses in JSON and/or XML formats. Functions will have to be implemented to work with these formats and parse through it in an efficient manner. The parsed data that is desired by the user will also have to be stored in the database for future use. Keeping the data in its original format will create efficiency problems for later user.

**Data distribution:**

The app will include an SQLite database. The database features of the app will be limited to the set of features that SQLite offers. The reason behind using an SQLite database is to keep the functionality of the app simple.

## Logical database requirements

A screenshot of a cell phone

Description automatically generated

There will be a simple 4 table SQLite database implemented in this project. One of the tables is titled Task and will store the tasks pinned to the To-do list. There will be another table titled Song which will keep track of details of each song in the Song List. A third table will be titled Media which will keep track of any media referenced from the user’s device file system (audio recordings, photos of sheet music). There will also be an intersection table linking a Song entry to its respective Media entry(ies).

## Software System attributes

**Reliability:**

The app will have to be implemented in a way that there will be minimal error with the use of its features. API calls will have to be correct and any data returned will have to be handled appropriately. Queries to the SQLite database will have to return and handle correct results.

**Availability:**

The database of the app will have to make its contents accessible in any situation where it is needed. The song search will return as many results as the API it is utilizing offers. Any data that is not found in the song search will have to be manually entered in the song entry form by the user.

**Security:**

The app will have to make sure that the user is always logged into the appropriate accounts for use with some of its features. Some of the API’s used with some of the features requires user authentication. One example being the Google Docs sheet notes. The user will have to be logged into their Google account to access the document through this app with the use of the Google Docs API.

**Maintenance:**

Any subsequent changes to the app during development will have to be implemented and integrated with the current system in a way that it is still functional. With everyday use of the app, the database will have to be updated in a way that that only the desired changes are made. The data will have to be persistent because of the nature of an SQLite database.

**Portability:**

Most of the features should be able to be used where there is a “reasonable” network connection. For example, a person in a Guitar class at the school should be able to use the app in the classroom where there is a Wi-Fi signal or Mobile Data connection.

## Asset requirements

There will be simple “clip art” assets used for parts of the UI of the app. For example, the bottom “toolbar” will feature little pictures to represent its respective page. User created audio recordings and photos of sheet music will have to be kept track of in the asset table of the database.

# References

Include a listing of relevant references to theory, tools, methods, etc.

Length: as needed, but minimum 5 references

References:

1. <https://acousticguitar.com/acoustic-guitar-notation-guide/>
2. <https://fluttercrashcourse.com/>
3. <https://www.youtube.com/channel/UCRCpzcQz-t2ueVihCIx5jDg>
4. <https://www.songsterr.com/a/wa/api>
5. <https://developer.spotify.com/documentation/web-api/>
6. <https://flutter.dev/docs/cookbook/networking/fetch-data>
7. <https://pub.dev/packages/xml>
8. <https://pub.dev/packages/googleapis>
9. <https://developers.google.com/drive/>
10. <https://medium.com/the-web-tub/making-a-todo-app-with-flutter-5c63dab88190>
11. <https://medium.com/flutter-community/using-sqlite-in-flutter-187c1a82e8b>
12. <https://pub.dev/packages/http>

Some useful references:

1. <http://perun.pmf.uns.ac.rs/old/newC/se/SRS.htm>
2. <http://www.mojofat.com/tutorial/step1.html>
3. <http://techwhirl.com/skills/tech-docs/writing-software-requirements-specs/>