LIFE MOBILE APPLICATION

APPLICATION PROGRAM INTERFACE: Research

Colin Man, Malina Jiang

VERSION 1.0

Last Update: 2 February 2015

Table of Contents

1 Document Control

1.1 Change Record

Version	Date	Author	Changes
1.0	02-02-15	Colin Man	Draft following overview + description
1.0	02-02-15	Colin Man	Draft Google Maps API
1.0	02-02-15	Malina Jiang	Draft Yelp and Google Drive API

1.2 Definitions

Term	Definition
LMA	Life Mobile Application
LBS	Location Based Services
API	Application Program Interface

Note: "LMA" is consistently used throughout all documents as the name of the mobile application, although it is understood that another name will be chosen for branding purposes.

2 Introduction

2.1 Scope and Purpose

This document contains a description of third party API's that may potentially be incorporated into LMA, in terms of contribution to the business model as well as technical functionality and constraints.

The API's outlined allow interaction with a variety of third-party products that do not define LMA individually, but come together to form a unique platform that encompasses the totality of the mobile user's needs.

The existence of this document is imperative to the success of the product as it clearly defines the constraints of each interface and the extent to which we can leverage the product to create a novel platform. It is crucial to the development of a business plan with a clearly defined direction as well as a viable and efficient implementation. Without such definitions, it is easy to design a product that is impossible to develop on the technical side or assumes third-party functionality that does not exist.

A description of the API's will also help in defining and outlining a minimal viable product, a subset of LMA's features that form the core functionality and can stand alone if necessary.

As such, each third-party API description contains the following components:

- **Overview** – Summary of the features and purpose. This includes a description of the functionality encompassed in the API as well as its contribution to the system.

- **Structure and API Calls** – A technical description of the API and the structure of its interface. Since the way information is obtained varies widely depending on the design of each API, this is vital both in extracting constraints for the business model as well as development of the actual application in the implementation stage.

- **Technical Constraints –** Detailed summary of constraints in terms of information that can be obtained through the application user interface. Includes information such as maximum volume of requests that is critical to making important design decisions.

3 API: Google Maps

3.1 Summary

The Google Maps API is divided into four main types of interfaces, depending on the technology used to implement the product as well as to access the data.

The four API interfaces are:

- **Embed API** – The simplest of the four, which simply uses HTML and iframes to embed content into any website.
- **Web API** – Provides a set of functionality that can be accessed using standard RESTful calls. These calls can be placed from web applications (AJAX) or native applications.
- **JavaScript API –** Provides extended maps functionality in visualization and map content. Can be integrated into platforms that run mainly JavaScript.
- **iOS API** – Native integration of functionality that interfaces with the operating system directly. Can only be used by iPhone applications, but has similar functionality to the JavaScript API.

3.2 Embed API

3.2.1 Overview

The Embed API is a simple way to embed maps into an application. It allows developers to embed iframe elements into the web application and specify various features using parameters in the query portion of the src URL. Thus, information is obtained and displayed to the user through purely HTTP requests and no interaction with the API in between the requests is needed. Technical details are outlined in "Structure and API Calls".

The API allows for four different ways to display maps content:

- **Place Mode** – displays a map pin at a predefined location (landmark, business, geographic feature, or custom pin)
- **Directions Mode** – displays the path between two different points and shows the time and distance estimated.
- **Search Mode** – displays a map that contains the results to a search query with a term specified in the attributes of the iframe tag.
- **View Mode** – Provides a barebones map that contains no visual content apart from the basic map functionality. This can be either a map view or a street view.

Note that the Embed API creates a map that links to Google Maps and provides an interface to work with options that may not be in the code injected through the iframe.

If the user is logged in to their Google account, they will also be able to save the locations from the embedded map into their user account, with custom attribution information that can b specified by the application.

3.2.2 Structure and API Calls

The API call is performed through an HTTP request sent via the "src" attribute of an iframe. The call is structured as follows:

https://www.google.com/maps/embed/v1/MODE?key=API_KEY&parameters

The mode of operation (MODE) is one of place, directions, search, view, or streetview

In order to use the Embed API to obtain maps, the application must first register with Google to obtain an API key (API_KEY) that is included in the url query used to embed the map into the page. This key can be restricted to certain domains to ensure that others do not steal it.

Since the HTML to retrieve the maps is retrieved using a simple HTTP call to the web service, all parameters that have to do with obtaining data must be sent through the query string. Other properties such as width, height, border, etc. can be customized in the iframe and slightly change the rendering of the map.

The possible query parameters to the Embed API (parameters) are as follows:

- attribution_source – string that describes the site or app; allows users to see what source is attributed to saving the place on their maps
- attribution_web_url – the url of the site or app (as attribution information)

- attribution_ios_deep_link_id – a URL scheme that links to the iOS application (using iOS internal links)
- center – the center of the map view (latitude and longitude values)
- zoom – (zoom level from 0 to 21)
- maptype – either "roadmap" or "satellite" depending on the view that should be loaded
- language – the language that should be used in the embedded map
- region – the borders and labels that should be displayed

The mode-specific query parameters are as follows:

- **Place**
    - q – the place to show on the map. This value can either be a location (escaped string) or an address (does not take latitude and longitude since it must be a name location to be highlighted).
- **Direction**
    - origin – the starting point location (as before, in either escaped address format or place name)
    - destination – the end point location (as before, in either escaped address format or place name)
    - waypoints – a set of intermediary places (as before, in either escaped address format or place name) separated by the pipe character
    - mode – method of travel (choose from driving, walking, bicycling, transit, or flying). Options will be shown on map if none is chosen.
    - avoid – any obstacles that the map should avoid (tolls, ferries, highways). Different items can be separated using the pipe character.
    - units – either metric or imperial depending on the units that should be displayed
- **Search**
    - q – the search term that should be shown on the map (in escaped URL format – can include restrictions such as "in Danville near 94526")
- **Street View**
    - pano – the panorama id that should be shown (if not specified, one will be found from the location)
    - heading – the direction that the camera is facing in degrees from due north
    - pitch – the angle of the camera up or down in degrees
    - fov – the horizontal field of view in degrees

3.2.3 Technical Constraints

Using the map modes in the Embed API is not restrictive in terms of the user interaction – the user can move the map around in the iframe and they can switch view modes (Satellite, Maps, etc.). The constraints are mainly on the ease of incorporating dynamic content and on showing or controlling the maps interaction.

It is not possible to dynamically load content into the map since the map is hosted on Google's own servers and we are merely using an iframe to load it. Any attempt to change the "src" of the iframe in order to load the new parameters will cause the entire map to reload, which does not give a good user experience (we want the search to be done within the map interface, not by refreshing and reloading the entire map, which also has a higher latency).

This interface can only be used in a website since there are many limitations with iframes on mobile devices (Safari has compatibility issues, as does the android browser). The iframe served by Google Embed API is also not optimized for mobile, so may not be as easy to use on a phone.

There are no constraints on the volume of API calls that can be made, so the use of this API is essentially unlimited.

3.2.4 Summary

When compatibility and dynamic flexibility is not an issue, using the Google Embed API is the best solution to provide map functionality. It is the easiest to use out of the four API's and provides a very fast integration of maps functionality into a website.

In terms of LMA, the Embed API can be used for various maps services in which the location of a place should remain static for the duration of the webpage load. Examples of this may include: loading directions to a certain place, keeping track of saved routes and places, etc. For additional functionality of Google Maps, we may have to resort to the Web or native API's. See below for a full description.

3.3 Web API

3.3.1 Overview

3.3.2 Structure and API Calls

3.3.3 Technical Constraints

3.3.4 Summary

3.4 JavaScript API

3.4.1 Overview

3.4.2 Structure and API Calls

3.4.3 Technical Constraints

3.4.4 Summary

3.5 iOS API

3.5.1 Overview

3.5.2 Structure and API Calls

3.5.3 Technical Constraints

3.5.4 Summary

4 API: YELP

4.1 OVERVIEW

Yelp offers its users convenience and accessibility to local businesses wherever they go. In addition to business reviews, Yelp also finds events and lists for the user and allows communication between Yelpers. Yelp offers an unbiased look at the businesses within the surrounding area and offers its feedback to users so that they can make the best decisions about which businesses to patronize. The Yelp API offers the following functionality:

- Find the top results in a geographical location in response to a client query.
- Sort results according to the client query (e.g. by highest to lowest ratings, or greatest to least distance from the client location).
- Limit displayed results according to client specifications (e.g. display only businesses that offer Yelp Deals).
- Display detailed information about a Yelp Deal (e.g. savings, purchase URL).
- Identify whether a claimed has already been claimed on Yelp.com

The Yelp API as integrated with the Location-Based Services Module lends its wide database of businesses and various API calls to the task of locating the best deals in the area for clients of the Life Mobile Application. The specific applications of the API calls Yelp offers will be detailed in the section below.

The current version of the Yelp API is 2.0. Yelp v1.0 is deprecated but will but Yelp does not have any plans to turn it off as of now. Yelp currently limits API Calls to 25,000 calls, but accommodates requests for more calls. The API uses the standard, secure authorization protocol OAuth 1.0a, xAuth and offers two main sub-API's – the Search API and the Business API. As LMA focuses mainly on consumer users instead of business users, use of the Yelp API will mainly be constrained to the Business API.

4.2 STRUCTURE AND API CALLS

The Yelp API is limited to one type of API call, namely the GET request. Yelp authenticates each request using OAuth per the usual standard.

GET – Allows the client to search for local businesses. The API request takes in the following optional parameters:

- Term – Search term inputted by the client. If not included, the API will search everything.
- Limit – Number of search results to return.
- Offset – Offset the list of return businesses by this amount.
- Sort – Determines what the data will be sorted by. Options include 0 (for best matched), 1 (by distance), or 2 (most highly rated). When returning results, the business rating is adjusted for the number of ratings to give a more comprehensive view of the business quality.
- Category_filter – Applies filter to search results.
- Radius_filter – Determines the search radius of the query, within 25 miles.
- Deals_filter – Decides whether to search only for businesses with Yelp Deals.

As Yelp is also a location-based service, Yelp API requests have one required parameter:

- Location – Specifies the location to be used when conducting a search. Locations can specified with an address, neighborhood, city, state, zip, or county. They can additionally be specified with geographical coordinates:

    - Cll Parameters – These parameters are formatted as "cll = latitude, longitude".
    - Bound Parameters – These parameters form a geographical bounding box and are formatted as "bounds = sw_latitude, sw_longitude | ne_latitude, ne_longitude".
    - Ll Parameters –These parameters are formatted as "ll = latitude, longitude, accuracy, altitude, altitude_accuracy".

In response to the API request, the client receives the following:

- Region – Suggested bounds for map display.
- Total – Number of businesses in search result.
- Businesses – List of business entries that fulfill the search parameters. Each business entry has the following attributes.

    - Id – Yelp Business ID.

- Is_claimed – Whether the business has already been claimed.
- Is_closed – Whether the business has been permanently closed.
- Name – Business name.
- Image_url – Business photo URL.
- Url – Business Yelp page URL.
- Mobile_url – Mobile business Yelp page URL.
- Phone – Business phone number.
- Display_phone – Phone number formatted for display.
- Review_count – Number of reviews on the business.
- Categories – List of category names the business is associated with.
- Distance – Distance from search location in meters.
- Rating – Business rating. Includes rating image associated with the business.
- Snippet_text – Snippet text associated with the business.
- Snippet_image_url – URL of snippet image associated with the business.
- Location – Location data of the business. Includes address, city, state, zip, country, neighborhood and other location information of interest.
- Deals – Deals offered by the business. Includes deal ID, name, URL, image URL, start and end times, popularity, restrictions, and additional details.
- Gift_certificates – Gift certificate information for the business if the business offers them. Includes gift certificate ID, URL, image URL, balance, price, and other information.
- Menu_provider – Provider of business menu.
- Menu_date_updated – Timestamp of last update.

4.3 Technical Constraints

The main constraint on the Yelp API is the limit of 25,000 API calls. However, this constraint is relatively flexible, as the Yelp API describes the process for appealing for a greater API call allotment. An increase in the volume of calls will not incur additional costs, making for greater flexibility in the LMA application design.

Additionally, since all requests must be authenticated with an OAuth token, LMA will need a centralized account with the Yelp API. The account will allow LMA to manage access to the Yelp API and is used by the API to keep track of the number of API calls.

In terms of constraints for the API call (GET) offered by the Yelp API, most of the search parameters that the API allows are optional, and are only used to pare down the search results. The one required parameter is the location of the client and/or the location that the client wishes to research. The location is passed into the search call either in the usual form as an address, city, state, and zip, or as a set of geographical coordinates. In either case, the location associated with the search query must be determined, whether automatically or manually inputted by the client.

Finally, in terms of mobile platforms, Yelp's iPhone application (yelp for iPhones >= 2.0.0 and yelp4 for iPhones >= 4.0.0) enables search, view, and check-in functions. Yelp does not have the same support for Android phones or other smart phones, which may influence the direction of the app in the future.

4.4 Summary

The Yelp API will be integrated into the LBS Module of LMA. LMA offers the user as its features the convenience of pulling up information about local businesses in order to make an informative consumer decisions. The Yelp API enables this LMA function by allowing users to search for local businesses or services and viewing the associated information that Yelp provides, such as the location, rating, and any deals that the business offers.

In addition to allowing users to proactively search for services to meet their needs, the Yelp API enables LMA to recommend services to the user based on LMA's profile of the user. In this way, LMA extends the functionality of the Yelp API far beyond its current offerings. Since LMA also has information on the user's wish lists or to-do lists, LMA can make better recommendations for the user by matching the user's needs to local businesses nearby and calculating the optimal distribution of the user's patronage.

There are a few limitations to the Yelp API, namely the constrained volume of API calls and the need to collect information about the user's location. However, as described earlier, if the volume of API calls exceeds the initial quota, it is possible to appeal for a greater API call allowance. Additionally, since the LBS Module will need the user's location for many more applications, it will be relatively easy to gain access to such information.

5 API: Google Drive

5.1 Overview

Google Drive is a file storage and synchronization service based in the cloud that allows users to store, share, and edit files. Google Drive makes collaboration between users convenient by allowing modification of the same file by different users at the same time possible. It also makes files easily accessible no matter the location of the user. Among its many offerings, the Google Drive API offers the following functionality:

- Create and open files in the UI.
- Search for files.
- Distribute and market web applications.
- Share and collaborate on files by modifying the permissions of files.
- Export and convert Google docs.

The Google Drive API as it is integrated with LMA will provide users with cloud-based storage for their documents, pictures, and other files. It will also allow users to organize their files and bring them wherever they go.

The current version of the Google Drive API is Google Drive API v2. Google Drive API v2 is mostly compatible with v1 and Document List API v3, except for two changes: the parameter id from files.update and files.get renamed to fileId, and the parentsCollection field in files listing parents renamed to parents. However, Google Drive v2 supports migrating from these older versions. The API uses OAuth 2.0 protocol to authenticate a Google account and authorize access to user data.

5.2 Structure and API Calls

The Google Drive API is composed of many resource types, which subsequently are composed of various API calls.

5.2.1 Files

- **Get** – "Get /files/fileId"; gets a file's metadata by ID.
- **Insert** – "POST /files"; inserts a new file.
- **Patch** – "PATCH /files/fileId"; updates file metadata.
- **Update** – "Put /files/fileId"; updates file metadata and/or content.
- **Copy –** "POST /files/fileId/copy"; creates a copy of the specified file.
- **Delete** – "DELETE /files/fileId"; permanently deletes a file, skipping the trash.

- **List** – "GET /files"; lists user files.
- **Touch** – "POST /files/fileId/touch"; set's file's updated time to the current server time.
- **Trash** – "POST /files/fileId/trash"; moves file to trash.
- **Untrash** – "POST /files/fileId/untrash"; restores file from the trash.
- **Watch** – "POST /files/fileId/watch"; watches file for any changes.
- **EmptyTrash** – "DELETE /files/trash"; permanently clears user's trashed files.

5.2.2 About

- **Get** – "GET /about"; gets information about the current user and Drive settings.

5.2.3 Changes

- **Get** – "GET /changes/changeId"; gets specific change.
- **List** – "GET /changes"; lists user changes.
- **Watch** – "POST /changes/watch"; watches changes to user's Drive.

5.2.4 Children

- **Delete** – "DELETE /files/folderId/children/childId"; removes child from folder.
- **Get** – "GET /files/folderId/children/childId"; gets specific child reference.
- **Insert** – "POST /files/folderId/children"; inserts file into folder.
- **List** – "GET /files/folderId/children"; lists a folder's children.

5.2.5 Parents

- **Delete** – "DELETE /files/folderId/parents/parentId"; removes parent from folder.
- **Get** – "GET /files/folderId/parents/parentId"; gets specific parent reference.
- **Insert** – "POST /files/folderId/parents"; adds parents folder for a file.
- **List** – "GET /files/folderId/parents"; lists a folder's parents.

5.2.6 Permissions

- **Delete** – "DELETE /files/fileId/permissions/permissionId"; deletes a permission from the file.
- **Get** – "GET /files/fileId/permissions/permissionId"; gets a permission by file ID.
- **Insert** – "POST /files/fileId/permissions"; inserts a permission for the specified file.
- **List** – "GET /files/fileId/permissions"; lists a file's permissions.

5.3 Technical Constraints

The Google Drive API imposes some limits and quotas in order to protect the safety and integrity of the Google infrastructure. Among these limits are limits to

the number of requests that an API project, which are subsequently divided into two categories. The Google Drive API sets the maximum number of requests per second (project QPS) to 5 QPS and the maximum number of requests per day (project QPD) to 150,000 QPD. When the limits are exceeded the server returns an HTTP 503 status code. The API's suggestion for handling these errors is to use an exponential backoff approach, in which each time an error code is returned, there is a longer wait before retrying the failed call. This ensures that the server will not be overloaded with requests. However, given the large volume of requests allowable by the API, it is unlikely that LMA will reach these limits, at least initially, so these errors should not be a cause for great concern at this points.

The Google Drive API also imposes a quota on the number of results that can be returned in an API's response. This quota, maxResults, has a range of 1 to 1000 events, with the default being 1000 records. Again, as with the API call limit imposed by the API, this quota should not majorly affect the functionality of LMA.

Additionally, since all requests must be authenticated with an OAuth token, LMA will need a centralized account with the Google Drive API. The account will allow LMA to manage access to the Google Drive API and is used by the API to keep track of the number of API calls. Without authentication, the request is considered unauthorized and will not be considered. Google Drive API uses OAuth 2.0 protocol.

Finally, the default data format is JSON but the API also supports Atom format. As JSON is a fairly common data format, these impositions should not drastically affect the path the LMA application will take, but they should be considered when operating the API.

5.4 Summary

The Google Drive API will be integrated with projected Cloud Module of LMA. Part of LMA's core functionality is providing user's easy accessibility to all the components of their lives. The Google Drive API enables this LMA function by allowing users to store and modify files on the go and have the same files synced across all of their devices.

In addition to the basic personal benefits of having all personal files sync across all of the user's devices, the Google Drive API also has many social and collaborative

benefits. Chief among them is the ability of the user to work with other users on the same projects and files simultaneously. In this way, the Google Drive API facilitates collaboration between users and promotes networking and collective efforts toward common goals.

The Google Drive API interacts with several other functions of LMA. For example, user wish lists and todo lists are compiled by LMA for common usage by all of the services it provides. The Google Drive API offers a way for LMA to sync these lists across all applications and services within itself.

There are a few limitations to the Google Drive API, namely the constrained volume of API calls. But, as described in the previous section, the limitations set by the Google Drive API are fairly generous and should not be too constraining on the current LMA model. The additional data format requirements are also general enough that other applications will also need to resolve them.

6 API: OpenTable

6.1 Overview

OpenTable is an online restaurant reservation service that allows users to make reservations for restaurants online, read restaurant reviews, and earn points from the restaurants toward free meals and other deals. The OpenTable API offers developers the following functionality:

- Access and search data from OpenTable.
- Find single or multiple restaurants, along with information about the restaurant.
- Find URLs where reservations can be made.

The OpenTable API as it is integrated with LMA will provide users with the convenience of finding restaurants in the surrounding area as well as to view reviews and other information on the restaurants. Additionally, the OpenTable API allows users to easily make reservations at the restaurants they designate. The OpenTable API shares some functionality with the Yelp API, but is more specific to restaurants and offers a few additional services.

The current version of the OpenTable API is open source and public. The API requires no authentication and does require the creation of any accounts with the API.

6.2 Structure and API Calls

The OpenTable API is limited to one type of API call, namely the GET request. The GET request allows users to access data about restaurants in the OpenTable database. OpenTable does not require authentication for API calls.

GET – Allows the client to search for local restaurants and access data about specific restaurants

- "GET /api/stats" – Returns response detailing the number of countries, cities, and restaurants that are available in the OpenTable database.
- "GET /api/cities" – Returns the number of cities in the database, as well as the list of cities that have restaurants in the OpenTable database.
- "GET /api/restaurants" – Finds the restaurant(s) that match the information specified in the parameters.

- Name – Name of the restaurant.
- Address – Address line (should not contain state or city or zip).
- State – State code.
- City – City name.
- Zip – Zipcode.
- Country – Country code.
- Page – Page (default: 1).
- Per_page – Entries per page (options: 5, 10, 15, 25, 50, 100; default: 25).

Returns the number of restaurants that match the parameters provided, as well as the per_page and current_page settings. The call also returns a list of restaurants that match the parameters. Since all of the parameters are optional, if no parameters are specified, the call returns all the restaurants in the database.

- "GET /api/restaurants/:id" – Returns a single restaurant and its information. Each restaurant record has the following attributes:

  - Id – ID of the restaurant.
  - Name – Name of the restaurant.
  - Address – Address of the restaurant.
  - City – City location.
  - State – State location.
  - Area – General area of the restaurant.
  - Postal_code – Zip code of the location.
  - Country – Country location.
  - Phone – Phone number of the restaurant.
  - Reserve_url – Reservation link for the restaurant.
  - Mobile_reserve_url – Mobile reservation link for the restaurant.

6.3 Technical Constraints

The OpenTable API places some constraints on the number of API calls that can be made. It imposes a limit of 1000 requests per hour per IP Address. However, this limitation should not be too constraining to LMA at this point in time.

Additionally, the default data format for the OpenTable API is JSON. As JSON is a fairly common data format, these impositions should not drastically affect the

path the LMA application will take, but they should be considered when operating the API. The API also supports JSONP.

6.4 Summary

The OpenTable API will be integrated with the LBS Module of LMA. The LBS Module's primary goal is to use the user's location to deduce information about the user in order to suggest services and make recommendations to the user. The OpenTable API works in a similar function as does the Yelp API in that both locate services for the user and offers comprehensive information about each service that matches the user's search parameters so that LMA can best help the user decide which services to patronize.

In addition to the basic functionality of searching that the Yelp API also provides, the OpenTable API allows users to access the reservation site of the restaurant for the convenience of the user. The OpenTable API also keeps track of a user's points, which can then be used toward free meals or other deals. It also can bring up a restaurant's reviews to help users make informed decisions.

There are a few limitations to the OpenTable API, namely the constrained volume of API calls. But, as described in the previous section, the limitations set by the OpenTable API should not be too constraining on the current LMA model. The data format requirements are also general enough that other applications will also need to resolve them. Additionally, the OpenTable API does not require authentication or API tokens, so the process of bringing up functionality in the API may be less involved than it might be with other APIs.

7 API: Spotify

7.1 Overview

Spotify offers users the convenience of accessing their music anytime no matter their location. It also allows users to listen to artists, albums, and playlists for free and categorize their music into playlists or folders. Spotify broadens the musical horizons of its users by making new music easily accessible through its various browse features. The Spotify API offers the following functionality:

- Fetch data from the Spotify music catalog.
- Manage user's playlists and saved music.
- Returns data about artists, albums, and tracks from the Spotify catalogue.
- Provides access to user-related data such as playlists and music saved in the "Your Music" library.

The Spotify API as it is integrated with LMA will provide users with easy accessibility to their music. It will also allow users to organize their music, find new music, and bring it with them wherever they go.

The API uses OAuth 2.0 protocol to authenticate an application and authorize access to make requests.

7.2 Structure and API Calls

The Spotify API is composed of many resource types, which subsequently are composed of various API calls.

7.2.1 Albums

- "GET /v1/albums/{id}" – Gets an album. The id refers to the Spotify ID for the album. The track object has the following attributes:
  - Artists – The artist who performed the track.
  - Available_markets – List of countries in which the track can be played.
  - Disc_number – Disc number (usually 1).
  - Duration_ms – Track length in milliseconds.
  - Explicit – Whether or not the track has explicit lyrics.
  - External_urls – External URLs for the track.
  - Href – Link to the full details of the track.
  - Id – Spotify ID for the track.
  - Name – Name of the track.

  - Preview_url – URL to a 30 second preview of the track.
  - Track_number – Number of the track.
  - Type – The object type (in this case, "track").
  - Uri – Spotify URI for the track.

- "GET /v1/albums?ids={ids}" – Gets several albums. The ids refer to a list of Spotify IDs for the albums, with a limit of 20 IDs. The album object has the following attributes.

  - Album_type – Type of the album (one of "album", "single", or "compilation").
  - Artists – Artists of the albums.
  - Available_markets – Markets in which the album is available. Yses ISO 3166-1 alpha-2 country codes. An album's availability in a market is determined by the availability of one or more of its tracks in the same market.
  - Copyrights – Copyright statements of the album.
  - External_ids – Known external IDs for the album.
  - External_urls = Known external URLs for this album.
  - Genres – List of genres used to classify the album. If the album is not classified, the list is empty.
  - Href – Link to the full details of the album.
  - Id – Spotify ID for the album.
  - Images – Cover art of the album in various sizes, with the widest first.
  - Name – Name of the album.
  - Popularity – Popularity of the album, between 0 and 100, with 100 being the most popular. The popularity is calculated from the individual popularities of the tracks.
  - Release_date – Date the album was released, to various degrees of precision.
  - Release_date_precision – Precision to which the release_date value is known.
  - Tracks – Tracks of the album.
  - Type – The object type, "album".
  - Uri – the Spotify URI for the album.

- "GET /v1/albums/{id}/tracks" – Gets the album's tracks. The id refers to the Spotify ID for the album. The request returns a list of track objects. The query parameters are the following:

- Limit – Maximum number of tracks to return, with the default being 20, and the minimum being 1.
- Offset – The index of the first track to return, with the default being 0 (the first object). Use with limit to get the next set of tracks.

Both of the query parameters are optional.

- "GET /v1/artists/{id}/albums? – Gets an artist's albums.
- "GET /v1/browse/new-releases" – Gets a list of new releases.
- "GET /v1/search?type=album" – Searches for an album.

7.2.2 Artists

- "GET /v1/artists/{id}" – Gets an artist. The id refers to the Spotify ID for the artist. The API call returns an artist object. An artist object has the following attributes:

  - External_urls – Known external URLs for the artist.
  - Followers – Information about the followers of an artist.
  - Genres – List of genres that the artist is associated with. If the artist has not yet been classified, the list is empty.
  - Href – Link to the full details of the artist.
  - Id – Spotify ID for the artist.
  - Images – Images of the artist in various sizes, with the widest first.
  - Name – Name of the artist.
  - Popularity – Popularity of the artist, between 0 and 100, with 100 being the most popular. The artist's popularity is calculated from the popularity of the artist's tracks.
  - Type – The object type, "artist".
  - Uri – Spotify URI for the artist.

- "GET /v1/artists?ids={ids}" – Gets several artists. The ids refers to a list of Spotify IDs for the artists, with a maximum of 50 IDs. Returns an artist object, or null if the object is not found.
- "GET /v1/artists/{id}/albums" – Gets an artist's albums. The id refers to the Spotify ID for the artist. The query parameters are as follows:

  - Album_type – List of keywords that will be used to filter the response. If not supplied, all album types will be returned.
  - Market – An ISO 3166 alpha-2 country code that limits the response to a geographical market.

  - Limit – Number of album objects to return, with the default at 20, the minimum at 1, and the maximum at 50.
  - Offset – Index of the first album to return, with the default being 0 (the first album)

    .

- "GET /v1/artists/{id}/top-tracks" – Gets an artist's top tracks. The id refers to the Spotify ID for the artist. Returns track objects upon success. There is a single, required query parameter:

  - Country – An ISO 3166-1 alpha-2 country code.

- "GET /v1/artists/{id}/related-artists" – Gets an artist's related artists. The id refers to the Spotify ID for the artist. Upon success, responds with an array of up to 20 artist objects.
- "PUT /v1/me/following" – Follow Artists or Users.
- "DELETE /v1/me/following" – Unfollow Artists or Users.
- "GET /v1/me/following/contains" – Checks if Current User follows Users or Artists.
- "GET /v1/search?type=artist" – Searches for an artist.

7.2.3 Browse

- "GET /v1/browse/featured-playlists" – Gets a list of featured playlists. Returns a playlist object upon success. The query parameters are as follows:

  - Authorization – Required parameter. Valid access token from the Spotify Accounts service.
  - Country – An ISO 3166-2 alpha-2 country code.
  - Limit – Maximum number of items to return, with the default being 20, the minimum 1, and the maximum 50.
  - Offset – Index of the first item to return, with the default being 0 (the first object).

  The playlist attributes are as follows:

  - Collaborative – True if the owner allows other users to modify the playlist.
  - External_urls – Known external URLs for this playlist.
  - Href – Link to the full details of the playlist.
  - Id – Spotify ID for the playlist.
  - Images – Images for the playlist. Contains up to three images, in order of descending size.

- Name – Name of the playlist.
- Owner – User who owns the playlist.
- Public – Playlist's public/private status (true if the playlist is public).
- Tracks – Collection of tracks objects containing a link to the full details of the playlist's tracks, along with the total number of tracks in the playlist.
- Type – Object type, "playlist".
- Uri – Spotify URI for the artist.

The paging object attributes are as follows:

- Href – Link to the full result of the request.
- Items – Requested data.
- Limit – Maximum number of items in the response, as set by the query or by default.
- Next – URL to the next page of items.
- Offset – Offset of the items returned.
- Previous – URL to the previous page of items.
- Total – Total number of items available to return.

- "GET" /v1/browse/new-releases" – Gets a list of new releases. Returns an array of album objects upon success. The query parameters are as follows:

  - Authorization – Valid access token from the Spotify Accounts service.
  - Country – An ISO 3166-1 alpha-2 country code.
  - Limit – Maximum number of items to return, with the default being 20, the minimum 1, and the maximum 50.
  - Offset – Index of the first item to return, with the default being 0 (the first object).

7.2.4 Follow

- "PUT /v1/me/following" – Follows Artists or Users. The query parameters are as follows:

  - Authorization – Valid access token from the Spotify Accounts service. Modifying the list of artists or users the current users follows requires authorization of the user-follow-modify scope.
  - Content-Type – Required if IDs are passed in the request body (application/json).
  - Type – ID Type, either artist or user.
  - Ids – List of the artist or the user Spotify IDs.

- "DELETE /v1/me/following" – Unfollow Artists or Users. The query parameters are as follows:

    - Authorization – Valid access token from the Spotify Accounts service. Modifying the list of artists or users the current users follows requires authorization of the user-follow-modify scope.
    - Content-Type – Required if IDs are passed in the request body (application/json).
    - Type – ID Type, either artist or user.
    - Ids – List of the artist or the user Spotify IDs.

- "GET /v1/me/following/contains" – Checks if Current User Follows Users or Artists. The query parameters are as follows:

    - Authorization – Valid access token from the Spotify Accounts service. Getting details of artists or users the current users follows requires authorization of the user-follow-read scope.
    - Content-Type – Required if IDs are passed in the request body (application/json).
    - Type – ID Type, either artist or user.
    - Ids – List of the artist or the user Spotify IDs.

- "PUT /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Follows a Playlist. Returns a Boolean on success, true if the playlist will be included in the user's public playlists, and false if it will remain private. The user must be granted the playlist-modify-private scope in order to follow playlists privately. The query parameters are as follows:

    - Owner-id – Spotify user ID of the person who owns the playlist.
    - Playlist_id – Spotify ID of the playlist. Any playlist can be followed, regardless of its public or private status, as long as the playlist ID is provided.
    - Authorization – Valid access token from the Spotify Accounts service. Following a playlist publicly requires authorization of the playlist-modify-public scope. Following the same playlist privately requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).

- "DELETE /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Unfollow a Playlist. The query parameters are as follows:

    - Owner-id – Spotify user ID of the person who owns the playlist.
    - Playlist_id – Spotify ID of the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Following a playlist publicly requires authorization of the playlist-modify-public scope. Following the same playlist privately requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).

- "GET /v1/users/{user_id}/playlists/{playlist_id}/followers/contains" Checks if Users Follow a Playlist. The request parameters are as follows:

    - Authorization – Valid access token from the Spotify Accounts service. Checking if a user publicly follows a playlist does not require any scopes. To check if a usr is privately following a playlist, the user must be granted access to the playlist-read-private scope.
    - Owner-id – Spotify user ID of the person who owns the playlist.
    - Playlist_id – Spotify ID of the playlist.
    - Ids – List of Spotify User IDs that you want to check to see if they follow the playlist, with the maximum being 5 ids.

7.2.5 Library

- "PUT /v1/me/tracks?ids={ids}" – Saves tracks for the user. A track can only be saved once, so duplicate IDs are ignored. The request parameters are as follows:

    - Authorization – Valid access token from the Spotify Accounts service. Modification of the current user's "Your Music" collection requires authorization of the user-library-modify scope.
    - Content-Type – Content type of the request body (application/json).
    - Ids – List of the Spotify IDs.

- "GET /v1/me/tracks" – Gets user's saved tracks and returns a list of track objects. The request parameters are as follows:

    - Valid access token from the Spotify Accounts service. The user-library-read scope must have been authorized by the user.

  - Limit – Maximum number of objects to return, with the default being 20, the minimum 1, and the maximum 50.
  - Offset – Index of the first object to return, with the default being 0 (the first object).

- "DELETE /v1/me/tracks?ids={ids}" – Removes user's saved tracks. The request parameters are as follows:

  - Valid access token from the Spotify Accounts service. Modification of the current user's "Your Music" collection requires authorization of the user-library-modify scope.
  - Content-Type – Content type of the request body (application/json).
  - Ids – List of Spotify IDs.

- "GET /v1/me/tracks/contains?ids={ids}" – Checks user's saved tracks. The request parameters are as follows:

  - Ids – List of Spotify IDs for the tracks, with the maximum at 50 IDs.
  - Authorization – Valid access token from the Spotify Accounts service. The user-library-read scope must have been authorized by the user.

7.2.6 Playlists

- "GET /v1/browse/featured-playlists" – Gets a list of featured playlists.
- "PUT /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Follow a Playlist.
- "DELETE /v1/users/{owner_id}/playlists/{playlist_id}/followers" – Unfollow a Playlist.
- "GET /v1/search?type=playlist" – Searches for a playlist.
- "GET /v1/users/{user_id}/playlists" – Gets a list of the user's playlists. The request parameters are as follows:

  - User_id – User's Spotify ID.
  - Authorization – Valid access token from the Spotify Accounts service. Private paylists are only retrievable for the current user and requires the playlist-read-private scope to have been authorized by the user.
  - Limit – Maximum number of playlists to return, with the default being 20, the minimum 1, and the maximum 50.
  - Offset – Index of the first playlist to return, with the default being 0 (the first object).

- "GET /v1/users/{user_id}/playlists/{playlist_id}" – Gets a playlist. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Both Public and Private playlists belonging to any user are retrievable on provision of a valid access token.
    - Fields – Filters for the query. If omitted, all fields are returned.

- "GET /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Gets a playlist's tracks. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Both Public and Private playlists belonging to any user are retrievable on provision of a valid access token.
    - Fields – Filters for the query. If omitted, all fields are returned.
    - Limit – Maximum number of tracks to return, with the default being 100, the minimum 1, and the maximum 100.
    - Offset – Index of the first track to return, with the default being 0 (the first track).

- "POST /v1/users/{user_id}/playlists" – Creates a playlist. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Authorization – Valid access token from the Spotify Accounts service. Creating a public playlist for a user requires authorization of the playlist-modify-public scope, while creating a private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Name – Name of the new playlist. The name does not have to be unique.
    - Public – Boolean value that is true if the playlist will be public, and false if it will be private.

- "PUT /v1/users/{user_id}/playlists/{playlist_id}" – Changes a playlist's details.

- "POST /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Adds tracks to a playlist. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Adding tracks to the current user's public playlists requires authorization of the playlist-modify-public scope, while adding tracks to the current user's private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Uris – List of Spotify track URIs to add.
    - Position – Position to insert the tracks. Tracks are added in the order they are listed in the query string or request body.

- "DELETE /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Removes tracks from a playlist. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Removing tracks from the current user's public playlists requires authorization of the playlist-modify-public scope, while removing tracks from the current user's private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Tracks – Spotify URIs of the tracks to remove and their positions in the playlist.

- "PUT /v1/users/{user_id}/playlists/{playlist_id}/tracks" – Replaces a playlist's tracks. The request parameters are as follows:

    - User_id – User's Spotify ID.
    - Playlist_id – Spotify ID for the playlist.
    - Authorization – Valid access token from the Spotify Accounts service. Setting tracks in the current user's public playlists requires authorization of the playlist-modify-public scope, while setting tracks in the current user's private playlist requires the playlist-modify-private scope.
    - Content-Type – Content type of the request body (application/json).
    - Uris – List of Spotify track URIs to set.

- "GET /v1/users/{user_id}/playlists/{playlist_id}/followers/contains" – Checks if Users Follow a Playlist.

7.3 Technical Constraints

7.4 Summary