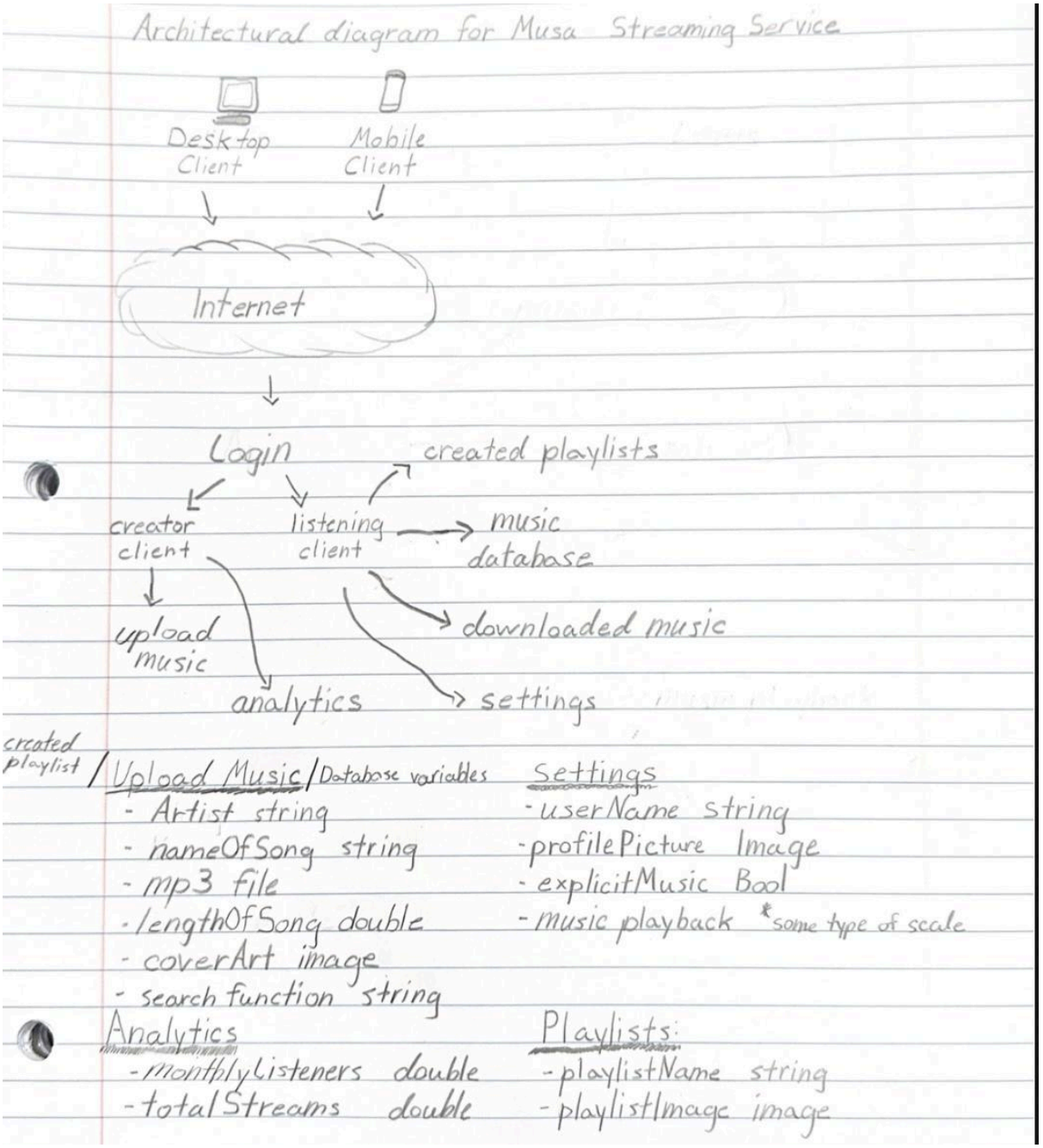
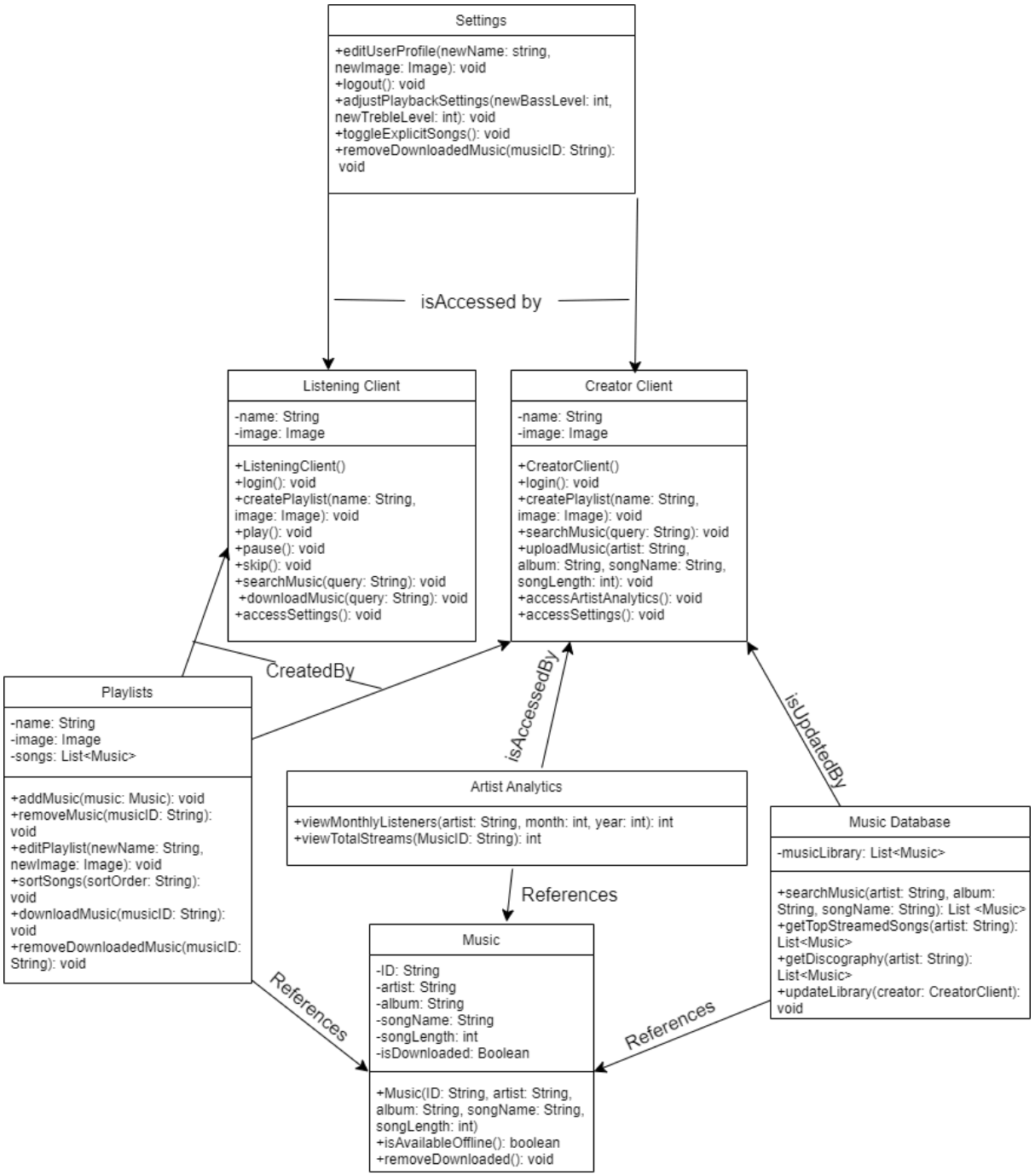
Software Architecture Overview

1. Architectural Diagram of all Components



(No changes made!)

2. UML Class Diagram



(No changes made!)

3. Classes, Attributes, and Operations

Classes:

- Listening client
- Creator client
- Settings
- Playlists
- Music database
- Music (general and downloaded)
- Artist Analytics

Attributes-

-User: Listening and Creator clients

- Listening-create Listener profile with name and image, login, create/edit Listener playlist with name and image, play/pause/skip music, search and download music from database to Listener system, access Settings
- Creator- create Artist profile with name and image, login, create/edit Artist playlist with name and image, search and upload music to database with Artist/album/song name and song length, access Artist analytics, access Settings

-Settings: edit User profile name and image, logout of User profile, adjust music playback settings such as bass/treble, toggle explicit songs, delete downloaded/uploaded music from User system

-Playlists: add/delete music, edit name and image, sort songs in alphabetical order by title/artist/album or chronological order by date added, download music

-Music database: holds music library and can be searched through by artist/album/song name, music organized by artist, within Artist we see top streamed songs and entire discography by date released, updated by Creator users who may add/remove music

-Downloaded music: downloaded to Listener system, available without wifi/data connection, downloaded music indicated by downloaded symbol

-Artist Analytics: view Creator monthly listeners, and total streams on each song

Operations-

- Listening Client Operations::

```
class ListeningClient {
   // Constructor for creating a listener profile
   ListeningClient(String name, Image image);
   // Method for logging in
   void login();
   // Method for creating/editing a listener playlist
   void createPlaylist(String name, Image image);
   // Methods for controlling music playback
   void play();
   void pause();
   void skip();
```

```
    // Method for searching music
    void searchMusic(String query);
// Method for downloading music
    void downloadMusic(String query);
    // Method for accessing settings
    void accessSettings();
}
```

- Creator Client Operations::

```
class CreatorClient {
    // Constructor for creating an artist profile
    CreatorClient(String name, Image image);
    // Method for logging in
    void login();
    // Method for creating/editing an artist playlist
    void createPlaylist(String name, Image image);
    // Method for searching music
    void searchMusic(String artist, String album, String songName, int songLength);
 // Method for uploading music
    void uploadMusic(String artist, String album, String songName, int songLength);
    // Method for accessing artist analytics
    void accessArtistAnalytics();
    // Method for accessing settings
    void accessSettings();
}
```

- Settings Operations::

```
class Settings {
    // Method for editing user profile
    void editUserProfile(String newName, Image newImage);
    // Method for logging out
    void logout();
    // Method for adjusting music playback settings
    void adjustPlaybackSettings(int newBassLevel, int newTrebleLevel);
    // Method for toggling explicit songs
    void toggleExplicitSongs();
    // Method for removing downloaded music from the user system
    void removeDownloadedMusic(String musicID);
}
```

- Playlists Operations::

```
class Playlists {
    // Method for adding music to the playlist
```

```
    void addMusic(Music music);
    // Method for deleting music from the playlist
    void deleteMusic(String musicID);
    // Method for editing playlist name and image
    void editPlaylist(String newName, Image newImage);
    // Method for sorting songs in the playlist
    void sortSongs(String sortOrder);
    // Method for downloading music from the playlist
    void downloadMusic(String musicID);
 // Method for removing downloaded music from the user system
    void removeDownloadedMusic(String musicID);
}
```

- Music(Download) Operations::

```
class DownloadedMusic {
    // Method for checking if music is available offline
    boolean isAvailableOffline(String musicID);
    // Method for removing downloaded music
    void removeDownloadedMusic(String musicID);
}
```

- Artist Analytics Operations::

```
class ArtistAnalytics {
    // Method for viewing monthly listeners for a creator
    int viewMonthlyListeners(CreatorClient creator, int month, int year);
    // Method for viewing total streams on a specific song
    int viewTotalStreams(String songID);
```

(No changes made!)

Verification Test Plan

This section outlines the test plans for verification. The verification testing aims to ensure that the software system meets its specified requirements and behaves as expected. The tests are categorised into three granularities: Unit testing, Integration testing, and System testing.

1. Unit Testing:

Feature: Login Functionality

- Test Set 1: Test User Authentication

- Test 1: Verify Valid Credentials
  - Description: Test the login functionality with valid credentials.
  - Test Steps:
    1. Provide a valid username and password.
    2. Attempt to log in.
  - Expected Result: User should be successfully logged in.
  - Test Vectors: Valid username, valid password.
  - Scope: Validate the login process when correct credentials are provided.

- Test 2: Verify Invalid Credentials Handling
  - Description: Test the system's response to invalid credentials.
  - Test Steps:
    1. Provide an invalid username or password.
    2. Attempt to log in.
  - Expected Result: User should receive an error message indicating invalid credentials.
  - Test Vectors: Invalid username, invalid password.
  - Scope: Validate error handling when incorrect credentials are provided.

Feature: Playlist Management

- Test Set 2: Test Playlist Creation

- Test 1: Verify Playlist Creation
  - Description: Test the creation of a new playlist by a user.
  - Test Steps:
    1. User creates a new playlist with a unique name.
  - Expected Result: Playlist should be successfully created and listed in the user's playlist collection.
  - Test Vectors: Unique playlist name.
  - Scope: Validate playlist creation functionality with unique names.

- Test 2: Verify Duplicate Playlist Name Handling
  - Description: Test the system's response to creating a playlist with a duplicate name.
  - Test Steps:
    1. User attempts to create a playlist with a name that already exists.
  - Expected Result: System should prompt the user to choose a different name or automatically append a unique identifier to differentiate the playlist.
  - Test Vectors: Existing playlist name.
  - Scope: Validate handling of duplicate playlist names.

2. Integration Testing:

Feature: Interaction Between User Authentication and Playlist Management

- Test Set 3: Test User Authentication and Playlist Creation Integration

- Test 1: Verify User Authentication before Playlist Creation
  - Description: Test the integration between user authentication and playlist management.
  - Test Steps:
    1. User attempts to create a playlist without logging in.
  - Expected Result: System should redirect the user to the login page before allowing playlist creation.
  - Test Vectors: User not logged in.
  - Scope: Validate redirection to login page when not authenticated.

- Test 2: Verify Playlist Creation after Successful Authentication
  - Description: Test the flow of creating a playlist after successful authentication.
  - Test Steps:
    1. User logs in successfully.
    2. User creates a new playlist.
  - Expected Result: Playlist should be successfully created after authentication.
  - Test Vectors: User logged in.
  - Scope: Validate playlist creation functionality after successful authentication.

Feature: Interaction Between Playlist Management and Music Database

- Test Set 4: Test Playlist Management and Music Database Integration

- Test 1: Verify Adding Music to Playlist from Music Database
  - Description: Test the integration between playlist management and the music database when adding songs to a playlist.
  - Test Steps:
    1. User selects a song from the music database.
    2. User adds the selected song to a playlist.

- Expected Result: Song should be successfully added to the playlist from the music database.
- Test Vectors: Song selected from music database.
- Scope: Validate adding songs from the music database to playlists.

- Test 2: Verify Updating Playlist after Music Deletion
- Description: Test the system's response to deleting music from the music database and its impact on playlists.
- Test Steps:
  1. User deletes a song from the music database.
  2. Verify if the song is removed from any playlists where it was added.
- Expected Result: Song should be removed from playlists if it was previously added and deleted from the music database.
- Test Vectors: Song deleted from music database.
- Scope: Validate removal of songs from playlists after deletion from music database.

3. System Testing:

Feature: End-to-End Functionality

- Test Set 5: Test End-to-End Playlist Management

- Test 1: Verify Adding Songs to Playlist
- Description: Test the end-to-end flow of adding songs to a playlist.
- Test Steps:
  1. User logs in.
  2. User navigates to a playlist and adds a song.
- Expected Result: Song should be successfully added to the playlist and reflected in the user's collection.
- Test Vectors: User logged in, song added to playlist.
- Scope: Validate adding songs to playlists from the user's collection.

- Test 2: Verify Playback Functionality
- Description: Test the end-to-end flow of playing a song from a playlist.
- Test Steps:
  1. User selects a playlist and starts playback.
  2. User interacts with playback controls (play, pause, skip).
- Expected Result: Song should start playing, and playback controls should function as expected.
- Test Vectors: Playlist selected, playback controls used.
- Scope: Validate playback functionality within playlists.

Feature: User Settings Management

- Test Set 6: Test User Settings Management

- Test 1: Verify Profile Editing
  - Description: Test the functionality to edit user profile settings.
  - Test Steps:
    1. User accesses the profile settings page.
    2. User edits profile details such as name and image.
  - Expected Result: Profile details should be successfully updated.
  - Test Vectors: Profile details edited.
  - Scope: Validate editing of user profile details.

- Test 2: Verify Music Playback Settings Adjustment
  - Description: Test the functionality to adjust music playback settings such as bass and treble.
  - Test Steps:
    1. User accesses the playback settings page.
    2. User adjusts bass and treble levels.
  - Expected Result: Bass and treble levels should be successfully adjusted for music playback.
  - Test Vectors: Bass and treble levels adjusted.
  - Scope: Validate adjustment of music playback settings.

.