1. Harmonia - Music Streaming App

2. Delegation of Tasks:
    - Fernando: Cost, Effort, Pricing Estimation
    - Reid: Timeline
    - Juan: Conclusion, references, project scheduling
    - Hieu: Class diagram
    - Yadiel: Case diagram, test plan
    - Ubaid: Project Scheduling
    - Adawari: sequence diagram

3. Everything from Deliverable 1

4. **Address the feedback provided in the Course Team Project Proposal**

I. *Feedback: The tasks section only refers to implementation/testing tasks. Include tasks with respect to requirements gathering, design too. (for eg- who will create user case diagrams etc)*

Response: The TA noted that our original task delegation only referred to implementation and testing. Therefore we have split up our requirements further and have divided up the workload including any requirements gathering and design as well.

Requirements Gathering & Analysis:
This part of the project will be led by *Ubaid Mohammad and Yadiel Valentin Rios*. They will be in charge of collecting and documenting both the functional and non-functional requirements, ensuring that all of our implementations align with the user expectation.

Use Case and Sequence Diagram Design:
This part of the project will be led by *Fernando Flores and Juan Esquivel*. They will be in charge of using UML tools to draw the system's core interactions and behaviors.

Class and Architectural Diagrams:
This part of the project will be led by *Adawari Dappa and Hieu Quang Tran*. They will be focused on the back-end structure and relationships between system components, and selection of the MVC architecture pattern.

Testing and Validation:
This part of the project will be led by *Reid Minton*. They will be in charge of ensuring that each feature is validated and confirm that it aligns with the initial requirements and test cases.

FInal Report Preparation and Presentation:

This part of the project will be led by *everyone*. They will be in charge of organizing all the diagrams, the requirements, and any system design elements into the final submission format as needed.

This will append our old Requirements listing and ensure that every stage of the development cycle–requirements, design, implementation, and testing has clear ownership. We appreciate the feedback from the TA to help refine our workflow and to cover all aspects of the development process.

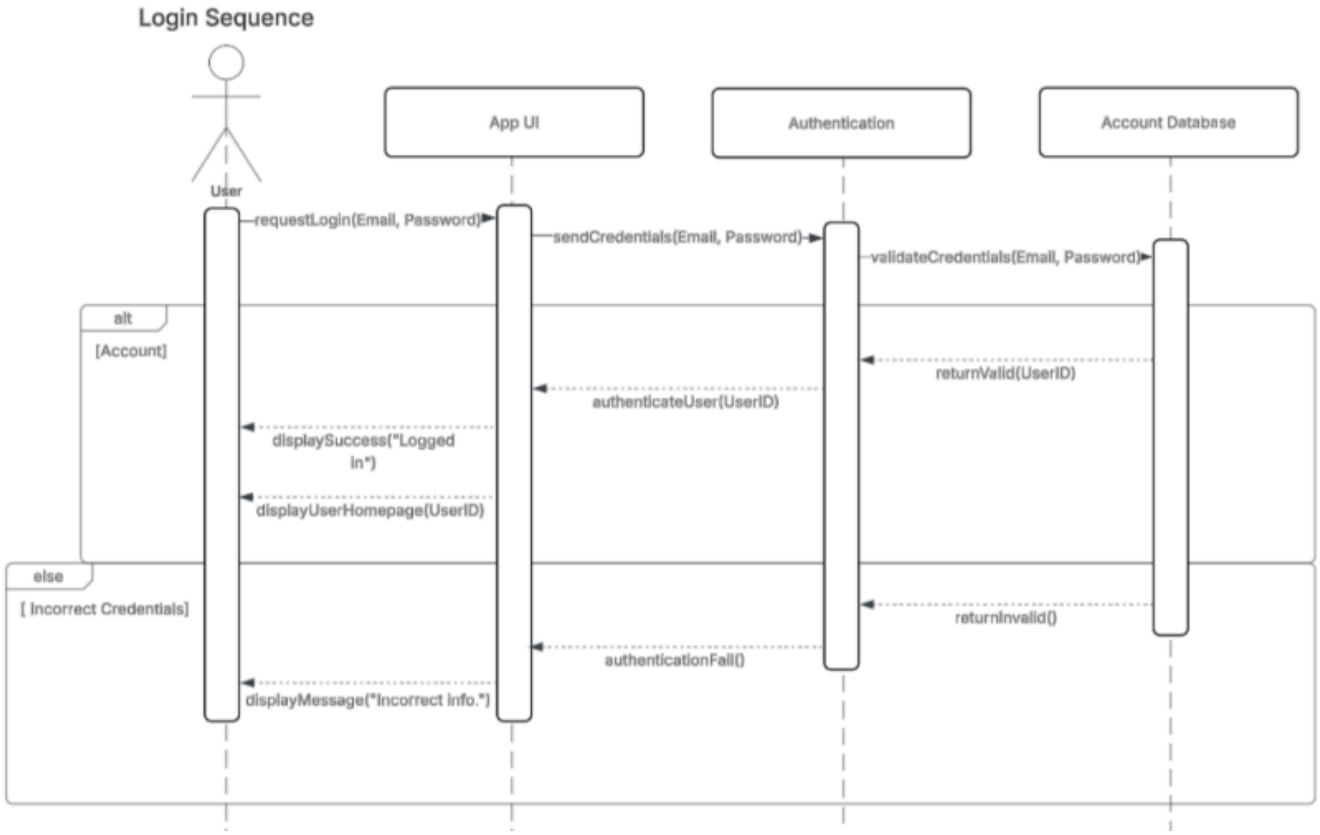
5. **What software process model are we using and why was it the choice**
    a. **We will deploy the Agile Process Model (Scrum)**
        i. Agile is effective for iterative development projects that require frequent updates through short sprints
        ii. Agile allows the team to focus on high-value features first to preserve app functionality
        iii. This model allows for the highest product quality through continuous integration and frequent testing
        iv. In Agile, the client is directly involved in the evolution of Harmonia, allowing for the project to be as strong as possible

6. **List of software requirements including:**
    a. **Functional requirements (5-7)**
        i. Users should be able to listen to music of their choosing, from playlists, libraries, or browsing available songs
        ii. Users should be able to create, share, and subscribe to playlists
        iii. Users should be able to follow other users and musicians
        iv. The system should provide recommendations on genres and musicians based on users' listening history
        v. The system should keep track of key metrics for songs and albums to measure popularity and trends
        vi. Musicians should be able to release songs and albums, and be able to announce when it is to be released
        vii. Users should be able to download songs in the app for offline listening, but not be able to have the songs' files
    b. **Non-functional requirements (use all non-functional requirement types listed in Figure 4.3 (Ch 4).**
        i. **Usability:** Navigation should be straight-forward so anyone can use the app easily, even if they've never used it before
        ii. **Performance:** Transitions between the current song and the next song in the queue should not have any buffering.
        iii. **Space:** The app and its data should be relatively lightweight to be able to run on devices with less storage
        iv. **Dependability:** Streaming servers should have 99.9% uptime, other servers can have less uptime as they are less critical
        v. **Security:** Users will sign in upon downloading the app, but should stay signed in if they close and reopen it. Streaming should also be encrypted for privacy
        vi. **Environmental:** Should be able to be downloaded on iOS, Samsung, and Windows devices
        vii. **Operational:** Music will be streamed from dedicated servers, and other lower-priority functionality will use separate servers

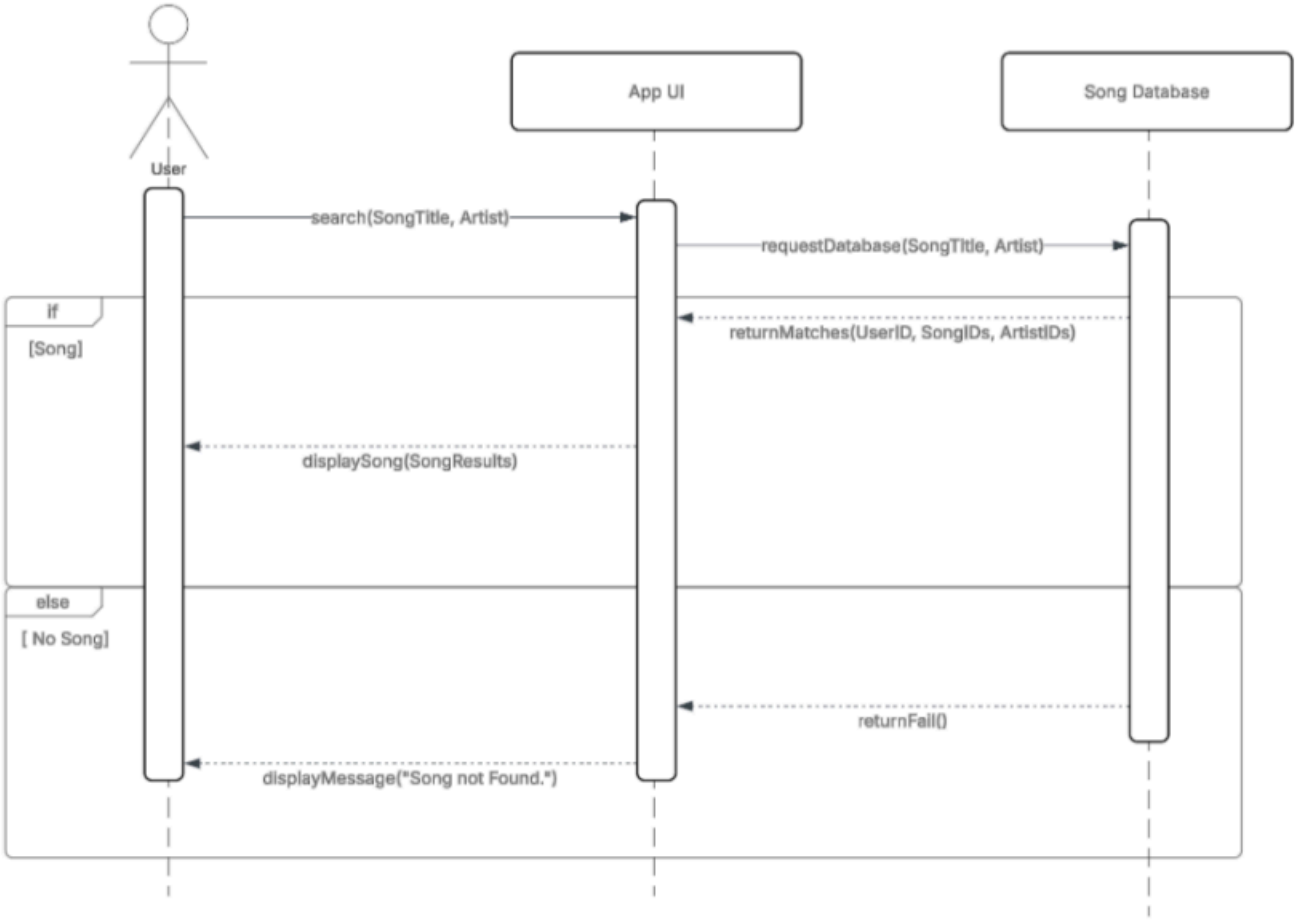
viii. **Development:** The application will be developed in React Native using TypeScript to prevent runtime errors common with JavaScript
ix. **Regulatory:** The system will acquire all necessary licenses and comply with federal and international regulations
x. **Ethical:** Data collected from users should not be sold or shared with third parties
xi. **Accounting:** The system will track subscriptions and necessary data for accounting
xii. **Safety / security:** Users can put restrictions on explicit music for themselves or another account linked to them for underage listeners

7. **Use case diagram - provide a use case diagram (fig 5.5) for your project**
   a. **Please note that there should be more than one use case depending on the complexity of your project**

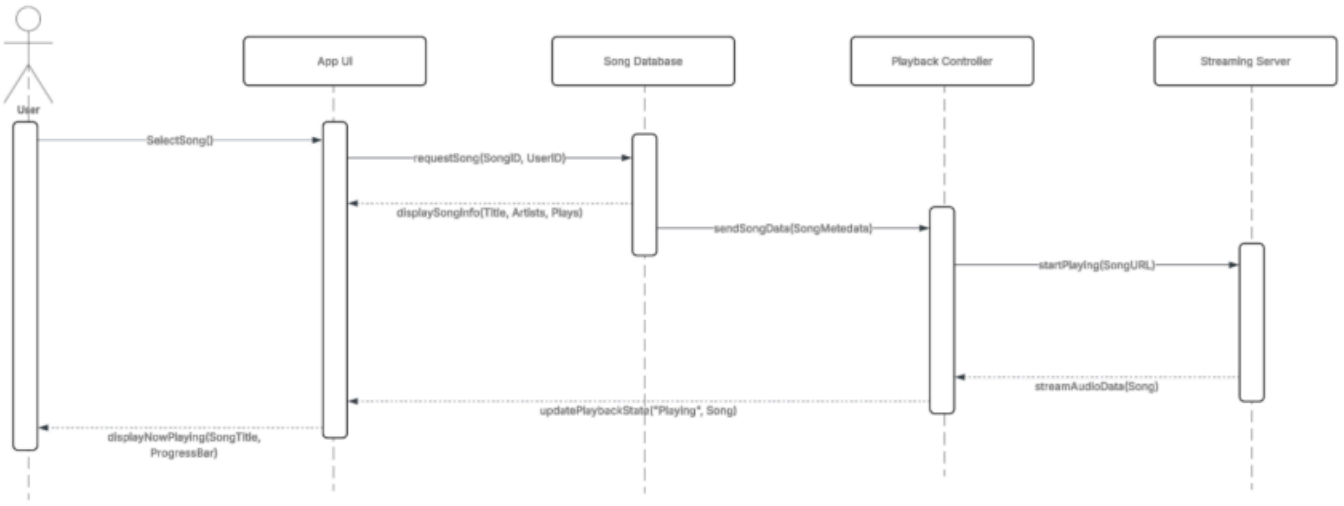
8. **Sequence diagram - provide sequence diagrams (fig 5.6, 5.7) for each use case of our project**
    a. **There should be an individual sequence diagram for each use case of your project (ch 5,7)**



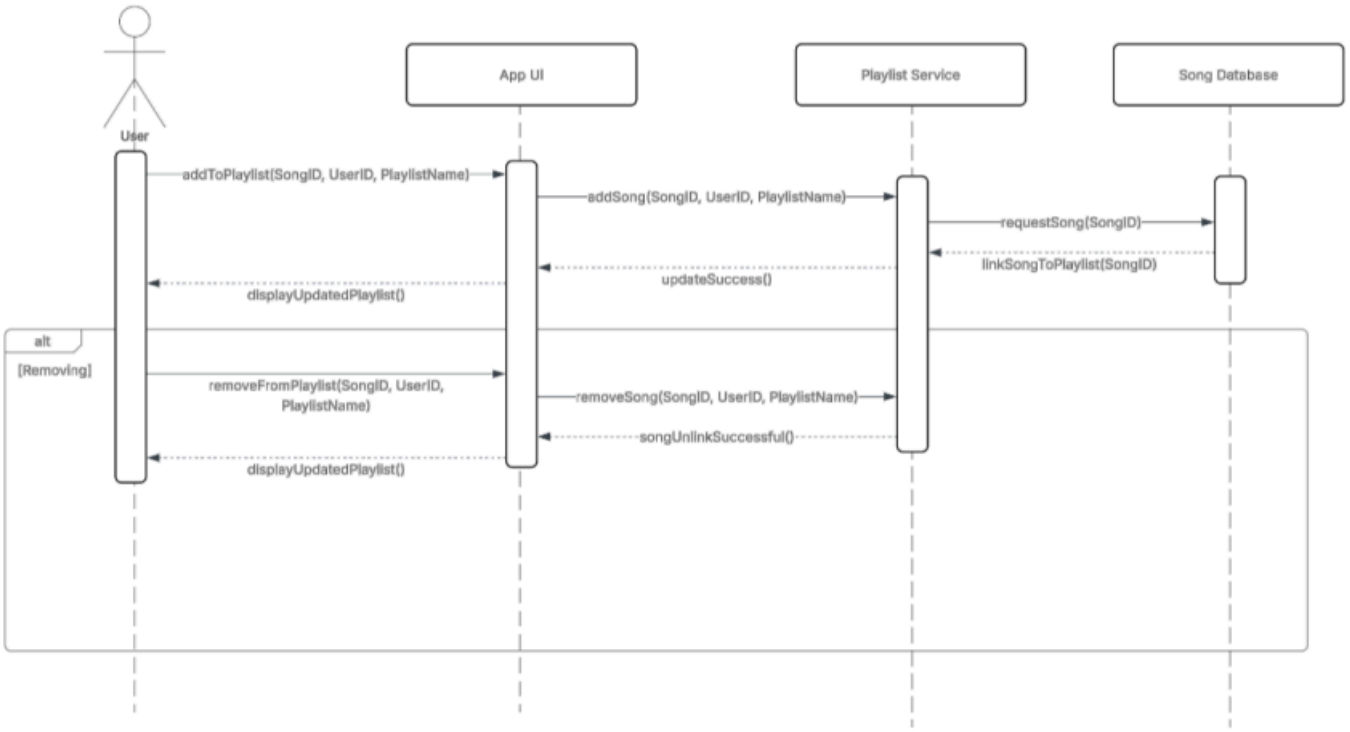
Search & Discover Music Sequence



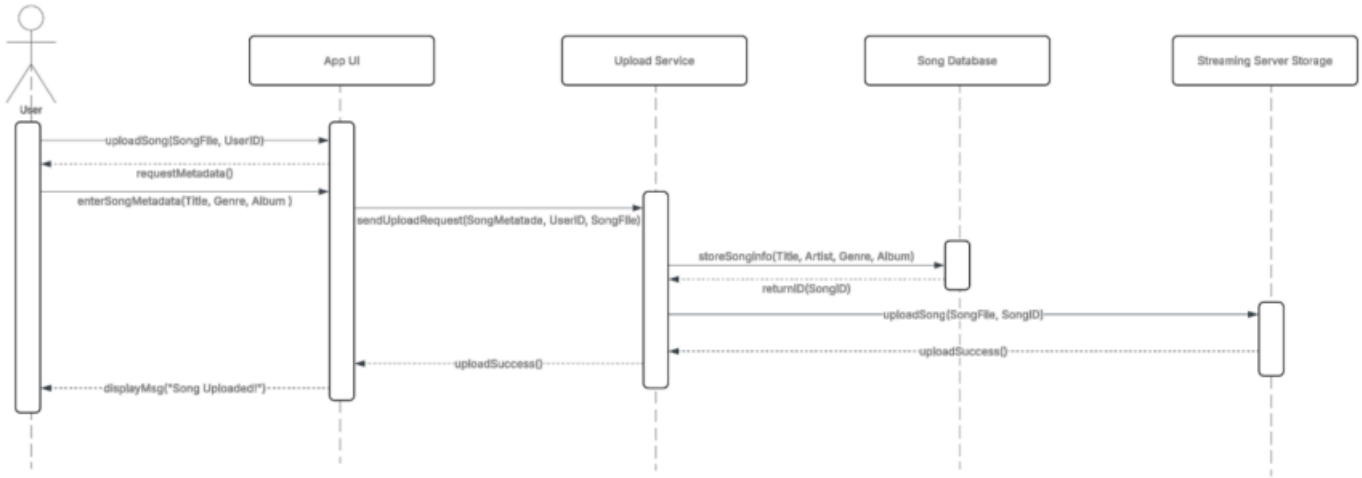
Play Music Sequence



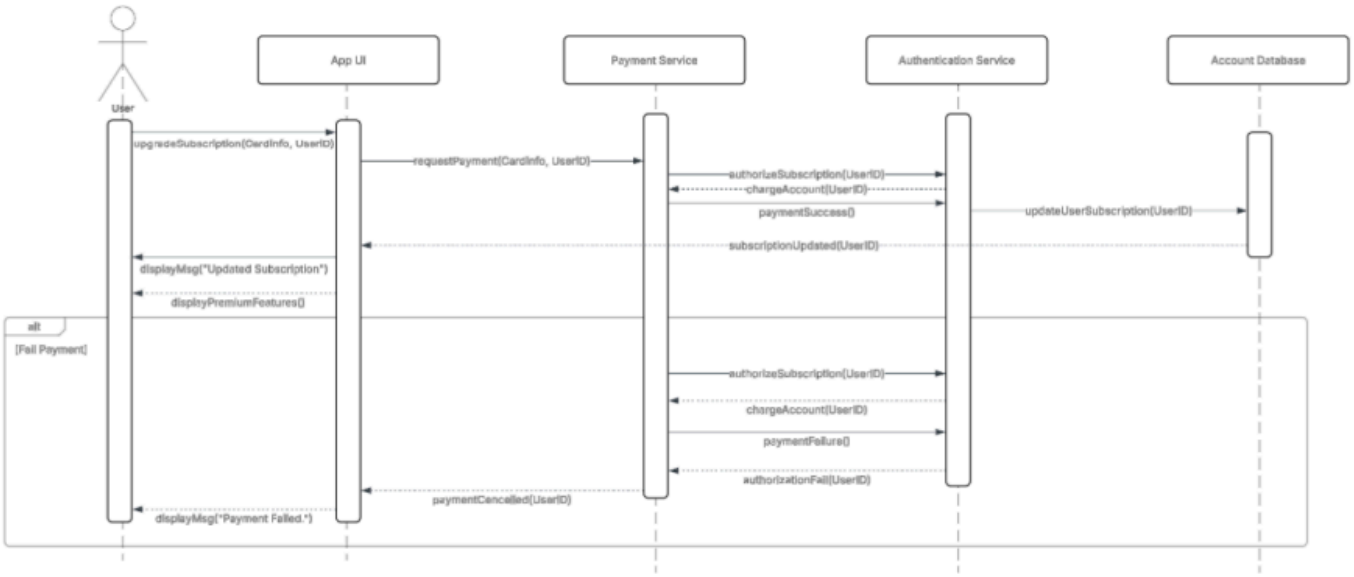
Playlist Interaction Sequence



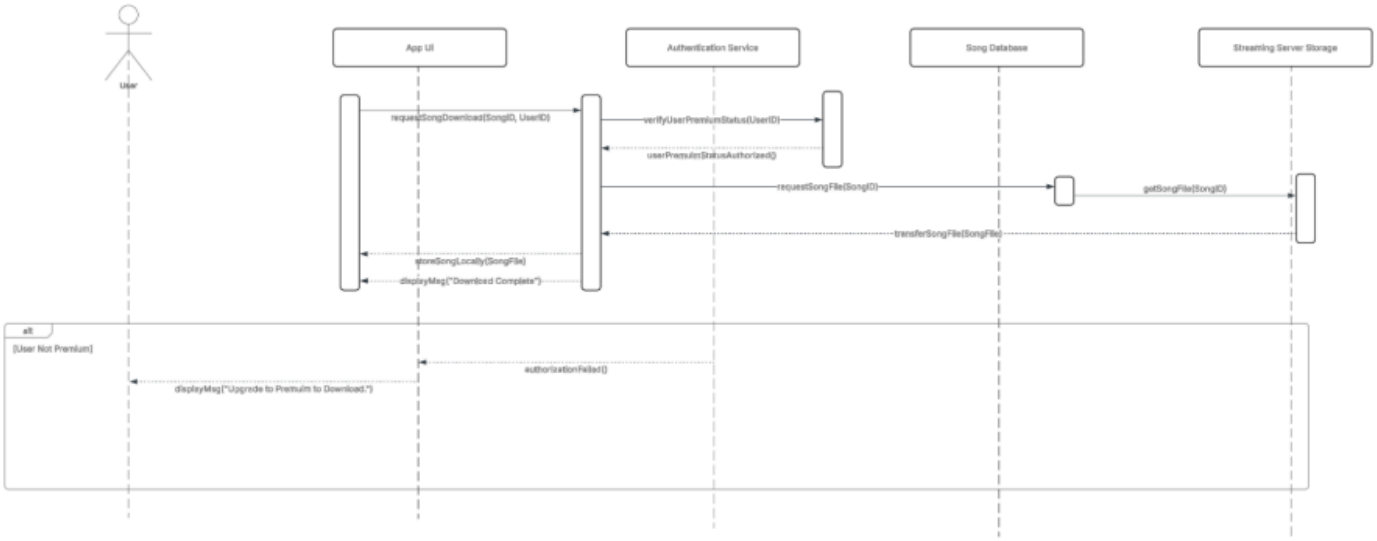
Upload Music Sequence



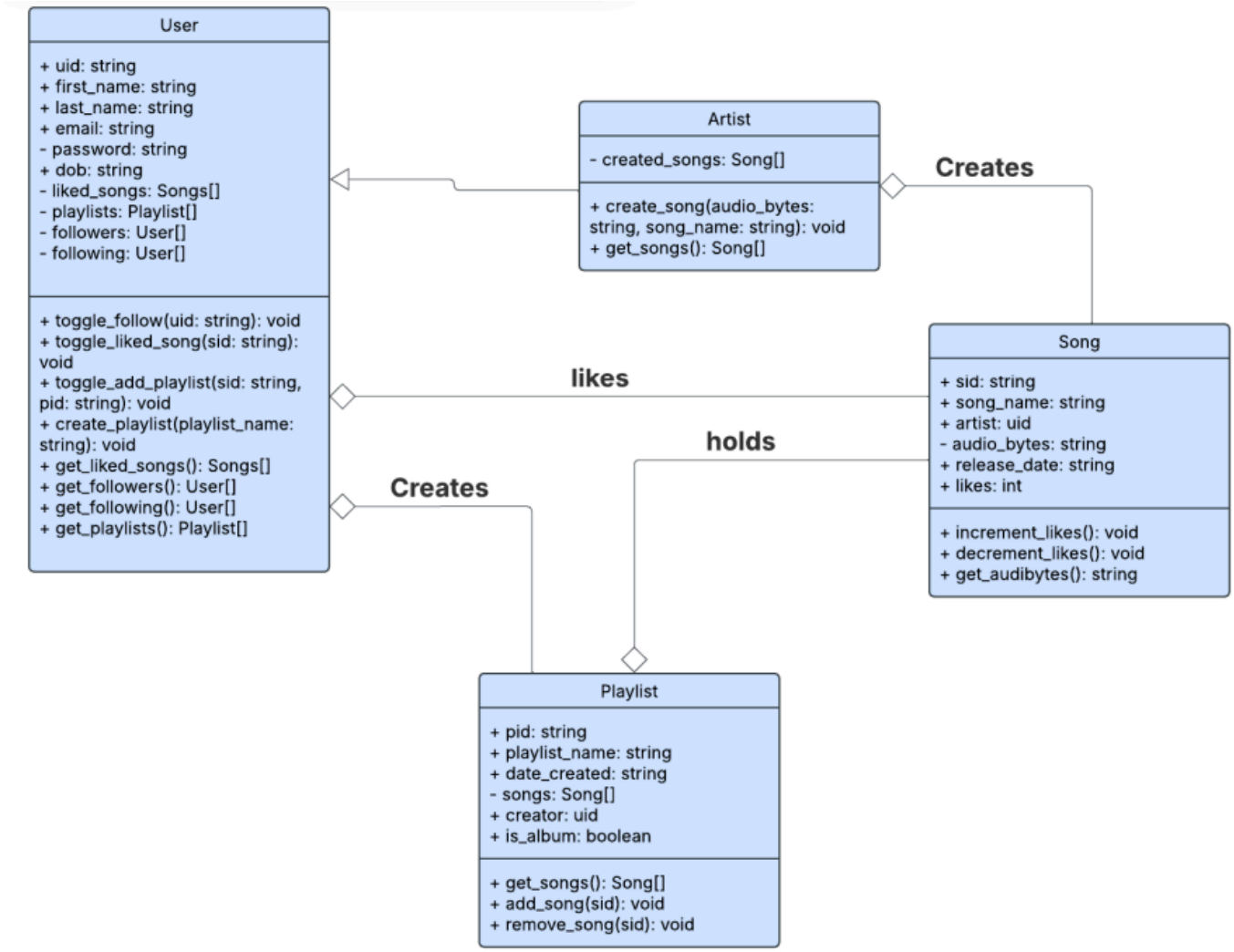
Upgrade to Premium Sequence



Download Music Sequence



9. **Class diagram - provide a class diagram (fig 5.9) of our project**
    a. **The class diagram should be unique and should include all classes of our project**
    b. **Make sure to include cardinalities and relationship types (as if it were like a database schema) like generalization and aggregation between classes in your class diagram.**
    c. **Make sure that each class has a class name, attributes, and methods named**



10. Architectural design - provide an architectural design of your project

- **a. Based on characteristics of your project, choose, and apply only one appropriate architectural pattern from the following list (ch 6)**
  - **i. MVC Pattern (fig 6.6)**
  - **ii. Layered architecture pattern (fig 6.9)**
  - **iii. Repository architecture pattern (fig 6.13)**
  - **iv. Client server architecture pattern (fig 6.13)**
  - **v. Pipe and filter architecture pattern (fig 6.15)**

Our team has chosen to do the client/server architecture pattern for the system design for our music streaming app. We believe that this best fits our project's needs because it allows all the different responsibilities to be separated into their respective sections.

- Server responsibilities:
  - Handles the music library
  - Handles user auth and account data
  - Streams content over the internet to the clients
- Client responsibilities:
  - UI and playback system
  - Handles user interaction
  - Sends requests to the server for music content

4. Project Scheduling

   We expect a 4-month **process** with **4 developers** (January 1 - April 30) working on the application itself.

- In the **1st month**, we are expecting to work on the **UI/UX design** and create a **concrete working front-end** that the user can interact with. Along with this as well, we will have other developers working on designing and setting up the **database**. By splitting up our workforce, we can accomplish a good front-end outline and get a solid database for music storage.

- In the **2nd month**, we plan to build the **Backend API** for database usage. We went with giving this task a full month to ensure that this part of the project is **meticulously taken care** of and does not leave any bugs or misinterpretations behind. While we build and implement the **API into the Database**, we can start working on the assignments for next month only and only if the API is 90-95% done.

- In the **3rd month**, our team plans to begin the **development** of our **front-end** implementation of the application. The timing of this part of the project has been carefully picked by our team to ensure that the **API endpoints are already created,** and we can safely introduce our front-end to the application.

- In the **4th month**, we estimate that we will still be **refining and upgrading** our front-end for the application. We hope to have some **form of feedback** that will allow for the team to directly assess the **ease of usage** to ensure that we stick to our goals as the team behind Harmonia.

- **Going forward,** we plan to keep developing the application in a way that aligns with our mission. We expect to start rolling out **prototypes** to **beta-testers and our requirements** team to provide **feedback** on the ease of usage and get a solid understanding of our status of the application. We do **not** expect to release the application after the 4 months; however, we do expect to have a prototype at the end of these 4 months for us to refine, and then release it at a **later date.**

- When all of **our missions** are taken care of and have a solid base, we will **release the application** to the public and begin our **maintenance (CI/CD)** and ensure that the application is at its best version at all times. We do **not** have a **hard date of termination** and expect this application to be able to **rival major music streaming platforms** in the near future.

5. Cost, Effort, Pricing Estimation & Staffing and Timing
   Function Points:

   The cost for Harmonia is derived from **Function Point Analysis** of which we outlined our cost estimation slides. To be able to determine the app's functional size, we need to identify **External Inputs, External Outputs, External Quieries, Internal Logical Files, External Interface Files** based on the Harmonias functional Requirements. Every component we have classified is going to use the standard **FP** weighting tables. When we sum the weighted components, we produce a **Gross Function Points (GFP)** of 155.

   Now we apply **Processing Complexity Adjustment (PCA)** to produce the **Final Function**. We believe the **Processing Complexity (PC)** to be **30**, and by using this score, we can determine that **PCA = 0.65 + 0.01 x PC**. This would give us a score of **0.95** for the PCA. Upon finding our PCA, we use that value to calculate the Final FP by **multiplying the GFP and PCA** to get **147**. We believe our productivity to be **10 FP** per person-month, our effort to be roughly **15 person-months** as 147/10 is roughly 15. Basically, if the project was to be completed by **one person**, it would be done in **15**

months. This is why our team decided on **4 developers** working on the project as it would be reasonable to say that they can complete it in **4 months** together. We believe our **total labor cost** to be roughly **$108k** as for the **4 employees**.

A majority of our project cost comes from the personnel, with a fraction coming from other smaller aspects of the project. We have estimated that **hardware cost** would require about **$1,650** including a cloud server, storage, a load balancer, and backup systems. We also believe that our **Software and licensing cost** would come out to a rough estimate of **$12,412** which could include Auth0, MongoDB Atlas, Monitoring, CI/CD tools, Admin fees, Music Licensing, and others. **In total**, we confidently believe that Harmonia would cost approximately **$136,500** with all things considered.

6. Test plan for the software

For our test plan, our example will have to do with how songs are added to playlists. We must make sure that when the user intends to add a song to a playlist, it is indeed placed in that playlist. Using JUnit, a *Song* class, and a *Playlist* class, we can achieve this test.

The *Song* class here contains general information on a song with some basic functionality

```
public class Song {

   private String title;
   private String artist;
   private String album;
   private String genre;
   private String lyrics;
   private int year;
   private boolean liked = false;
   private boolean disliked = false;

   public Song (String title, String artist, String album, String genre, String lyrics, int
year) {
       this.title = title;
       this.artist = artist;
       this.album = album;
       this.genre = genre;
       this.year = year;
   }

   void like() {
       this.liked = true;
       this.disliked = false;
   }

   void dislike() {
       this.liked = false;
       this.disliked = true;
   }
```

```
    void removeLike() {
        this.liked = false;
        this.disliked = false;
    }

    String getTitle() {
        return title;
    }

}
```

The *Playlist* class holds an array list of songs

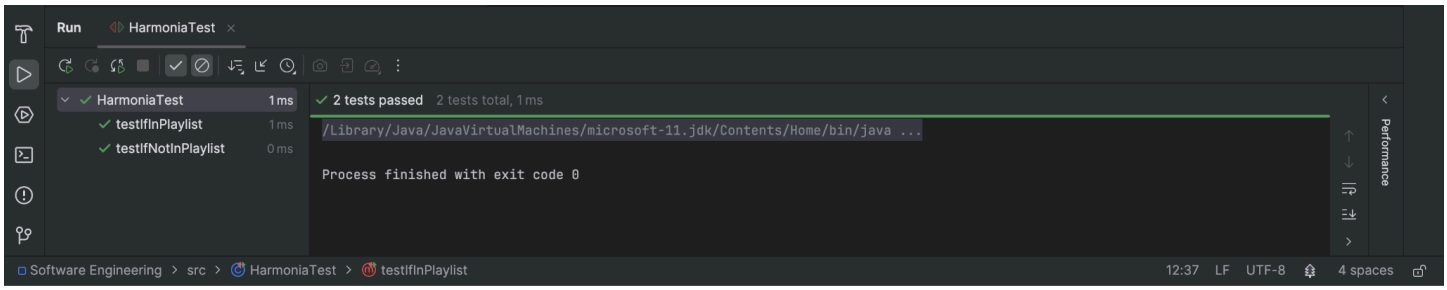
```
import java.util.ArrayList;

public class Playlist {

    private ArrayList<Song> songs;

    public Playlist() {
        songs = new ArrayList<>();
    }

    public Playlist(Song song) {
        songs = new ArrayList<>();
        songs.add(song);
    }

    void add(Song s) {
        songs.add(s);
    }

    void remove(Song s) {
        songs.remove(s);
    }

    boolean contains(Song s) {
        return songs.contains(s);
    }

    Song getFirstSong() {
        return songs.get(0);
    }

}
```

Here is the code utilizing JUnit used to test whether a song was actually added to the playlist or not. I added an extra condition checking if the titles matched to be absolutely certain they were the same.

```
import org.junit.*;
```

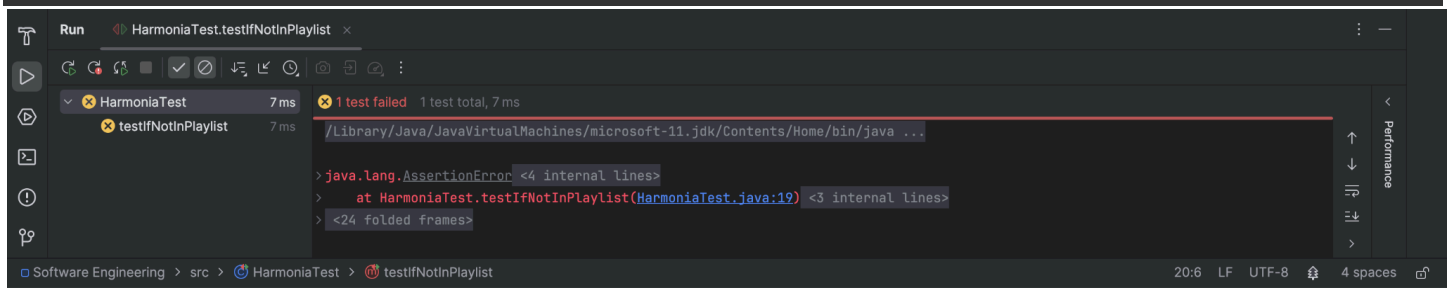
```
import static org.junit.Assert.*;

public class HarmoniaTest {
   Playlist playlist =  new Playlist();

   Song song1 = new Song("Songy-Song", "John McSongman", "Songs", "Country", "", 2023);
   Song song2 = new Song("Boots and Cats", "David Cool", "Songs", "Jazz", "...", 1986);

   @Test
   public void testIfInPlaylist() {
       playlist.add(song1);
       assertTrue(playlist.contains(song1) &&
playlist.getFirstSong().getTitle().equals("Songy-Song"));
   }

   @Test public void testIfNotInPlaylist() {
       playlist.add(song1);
       assertFalse(playlist.contains(song2));
   }
}
```

Here are the results to these specific tests, showing that both passed



The second test fails if *song2* is added and is found to be in the list

```
@Test public void testIfNotInPlaylist() {
   playlist.add(song2);
   assertFalse(playlist.contains(song2));
}
```



7. Comparison of our work with similar apps out there
   Our biggest competitors are Spotify and YouTube Music. Although Spotify and YouTube Music already have music, they also have podcasts and audiobooks, while Harmonia prefers to put their complete attention on making a lightweight app that focuses on high-fidelity music playback. Apps like YouTube Music use Angular while Harmoni prefers React due to its reusable components for better code

management and rendering capabilities for a more seamless UI. Spotify also tests their app for errors when they release by monitoring a dashboard for crashboards. Harmonia uses JUnit to test endpoint functions with several edge cases beforehand. Spotify's development phase is also an iterative process of features, while Harmonia's development process is split into frontend, backend, and integration.

8. Conclusion

The Harmonia Project is the culmination of effort and vision from our team to redefine the standards in the Music Streaming space. Our goal is to make an app that meets the modern standards of streaming, keeps up performance-wise, and can be just as reliable as any other. As per our **four-month schedule**, we established a very attainable goal with realistic expectations, and an achievable development path for our **4 developers** working in parallel across the **UI/UX design**, **database development**, **API construction**, and the **front-end integration**. Using this timeline, we could gradually build and refine our system to ensure **core architectural components**. Our cost and analysis would further validate the feasibility of our plan. We believe that with **4 developers** across **4 months**, our project would come at a final cost of **$136,500**. This was calculated across multiple aspects of the project including **personnel, software & licensing, hardware,** and with **contingency included**. Our realistic timeline could give the team a **very achievable** goal and inspire them to get the application pushed through. We believe that Harmonia can shake up the current streaming market with its **unique UI** and **high-fidelity music playback**. Our confidence comes with the structure that we have established with the meticulous **production and planning** of our application at a high level and everything else that went into making this project possible. In conclusion, the limits of our application are boundless and with enough patience and planning, we can surpass the likes of our competitors.

9. References

[1] Indeed.com, “Software engineer salary in United States,” *Indeed Career Explorer*, 2025. Accessed: Nov. 22, 2025. [Online]. Available: https://www.indeed.com/career/software-engineer/salaries

[2] C. Jones, *Estimating Software Costs: Bringing Realism to Estimating*. New York, NY, USA: McGraw-Hill, 2007. Accessed: Nov. 22, 2025. [Online]. Available: https://www.accessengineeringlibrary.com/binary/mheaeworks/a1b9399ed1a2656c/4b427828a57948f625c907fe8f6f5eb85c637cd42522aa88e03d2dc37ef4d1af/book-summary.pdf

[3] Amazon Web Services, “Media & Entertainment Industry Solutions,” *AWS*, 2025. Accessed: Nov. 22, 2025. [Online]. Available: https://aws.amazon.com/solutions/media-entertainment/

[4] Kendra F. Expert Customer experience (CX), “How to develop YouTube Music app,” CISIN, https://www.cisin.com/growth-hacks/cost-and-feature-to-develop-software-like-youtube-music/ (accessed Nov. 23, 2025).

[5] S. E. Jacob Vesterlund, “A behind-the-scenes look at how we release the spotify app (part 1),” Spotify Engineering, https://engineering.atspotify.com/2025/4/how-we-release-the-spotify-app-part-1 (accessed Nov. 23, 2025).