

Software Design

Description

for

VTunes

Release 1.0

By 김영현, 김지훈, 박종윤, 이용우, 최성훈

2023/11/10

Table of Contents

1. Introduction(서론)	3
1.1 Purpose(목적).....	3
1.2 Scope(범위).....	3
1.3 Intended Audience(대상 독자)	3
1.4 Glossary (용어설명)	4
2. System Overview(시스템 개요)	5
2.1 Product Overview(제품 개요).....	5
2.2 System Context Diagram(컨텍스트 모델).....	5
3 Design Considerations(설계 고려사항)	7
3.1 Assumptions and Dependencies(가정 및 의존사항).....	7
3.2 General Constraints(제약사항)	7
3.3 Goals and Guidelines(목표와 지침)	8
4. System Architecture(시스템 아키텍처)	9
4.1 Architectural Design(아키텍처 설계).....	9
4.2 Design Rationale(설계 근거).....	9
5 Detailed Design(상세 설계)	11
5.1 Diagram in Total.....	11
5.2 Low-Level or Component-Level Design(저수준 설계)	12
5.3 Interface Design(인터페이스 설계)	20
5.4 Sequence Diagram.....	28
6. Recommendation	33
6.1 File Structure	33
6.2 Caching	34

1. Introduction(서론)

1.1 Purpose(목적)

본 문서는 뮤직 플레이어 애플리케이션 VTunes 의 SW 설계 기술을 정리한 SDD(Software Design Document)이며, 개발할 애플리케이션의 기능, 인터페이스를 구현 및 검증의 진행에 있어서 아키텍처와 여러 고려사항을 기술한다. 이는 애플리케이션의 소프트웨어 설계를 상세하게 설명, 분석, 계획하여 개발 방향에 대한 정확한 의사결정의 기반을 목적으로 한다.

1.2 Scope(범위)

본 문서의 범위는 VTunes 애플리케이션의 전체 소프트웨어 시스템의 설계를 포괄하며, 애플리케이션의 핵심 모듈, 구성 요소, 인터페이스, 데이터 구조, 그리고 알고리즘 등의 세부 사항을 상세하게 설명한다. 또한, 이 문서의 범위는 소프트웨어 설계에 대한 세부 수준을 정의하고 시스템의 아키텍처 및 구조를 설명한다. 궁극적으로 SRS(Software Requirement Specification)에 명시된 요구 사항을 실현하기 위한 설계 계획의 개요를 제공한다.

1.3 Intended Audience(대상 독자)

개발자: VTunes 프로젝트에 참여한 개발자들이 본 문서의 주요 독자이며, 이 문서를 활용하여 소프트웨어 설계에 대한 심층적인 통찰력을 얻고, 아키텍처 선택을 이해하며, 설계 사양에 따라 시스템의 다양한 구성 요소를 구현한다.

테스터: 소프트웨어의 품질과 신뢰성을 보장하는 책임을 지는 것을 목적으로 한다. 본 문서는 테스터가 효과적인 테스트 절차를 계획하고 수행하고 소프트웨어가 규정된 요구 사항을 충족하는지 검증하는 데 중요한 기본 설계를 이해하는데 기여한다.

클라이언트: VTunes 의 개발을 의뢰한 고객이다. 개발이나 테스트에 직접 참여하지는 않지만 소프트웨어의 설계에 대한 높은 이해를 통해 애플리케이션의 기능과 신뢰성에 대한 심층적인 평가를 할 수 있다.

1.4 Glossary (용어설명)

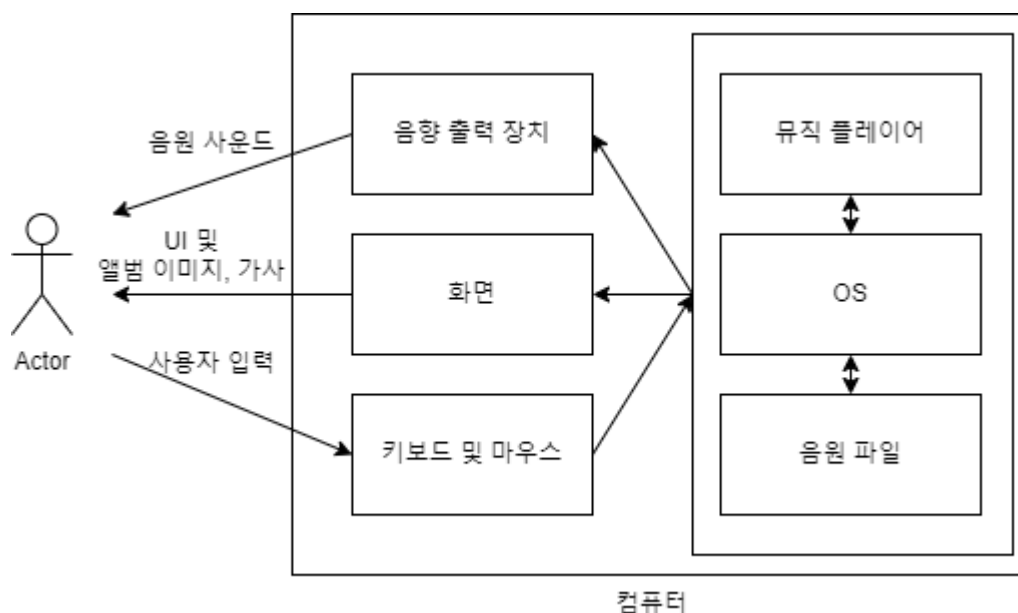
- 클라이언트: 뮤직 플레이어 제작을 의뢰한 주체이다.
- 뮤직 플레이어: 새로운 운영 체제에 탑재될 음원 재생 애플리케이션이다.
- 음원: 뮤직 플레이어에서 재생될 오디오 파일의 메타 정보 혹은 오디오 데이터 크 자체를 통칭하는 표현이다.
- 사용자: 뮤직 플레이어를 사용할 사용자(end user)이다.
- OS: Operating System(운영체제)이다.
- UI: User Interface(사용자 인터페이스)이다.
- UX: User Experience(사용자 경험)이다.
- GUI: Graphical User Interface(사용자 인터페이스를 그래픽적으로 제공하는 컴퓨터 소프트웨어 시스템)이다.

2. System Overview(시스템 개요)

2.1 Product Overview(제품 개요)

본 VTunes 뮤직 플레이어는 클라이언트가 새로운 운영 체제를 개발 중인 상황을 가정하며, 해당 운영 체제에 탑재될 현대적이고 강력한 뮤직 플레이어의 개발을 목표로 한다. 높은 수준의 오디오 엔터테인먼트 경험을 제공하며, 이를 지원하기 위해 제품의 핵심 기능으로 음원의 재생/일시정지, 이전/다음 곡 재생, 볼륨조절, 재생순서 조정, 타이머와 더불어 플레이리스트 기능을 제공한다. 제품 개발의 주요 고려 사항으로 플레이리스트의 메타 데이터와 알고리즘이 있다. 플레이리스트에 음원, 음원의 수, 음원의 위치를 모두 담아야 하며, 경우에 따라서 플레이리스트를 관리하는 별도의 메타 데이터가 추가적으로 필요 할 수도 있다. 플레이리스트 내에 있는 음원 파일들의 순서를 무작위로 조정하는 '셔플' 기능에 이용될 알고리즘 또한 주요 고려 사항이다.

2.2 System Context Diagram(컨텍스트 모델)



Actor 는 컴퓨터의 입력장치인 키보드 및 마우스를 이용하여 뮤직 플레이어에게 명령을 내리게 된다. 이 때, Actor 의 명령에 따라서 뮤직 플레이어, OS, 음원 파일들은 서로 밀접하게 연관이 되어있어서 해당 명령을 수행하기 위해서 움직인다. 이후 해당 명령을 수행하게 되면 화면을 통해서 출력하게 된다. 화면을 통해 출력되는 정보에는 UI, 앨범 이미지, 현재 재생중인 음원의 가사 등이 존재한다. 추가적으로 사용자가 설치해 둔 음향 출력 장치를 통해서 음원의 사운드를 출력하여 사용자가 원하는 음원을 청취할 수 있도록 한다.

3 Design Considerations(설계 고려사항)

3.1 Assumptions and Dependencies(가정 및 의존사항)

- 음원 파일을 뮤직 플레이어를 사용할 사용자가 직접 준비를 한 후에 해당 음원 파일을 플레이리스트 또는 뮤직 플레이어에 추가하여 음원을 재생하여야 한다.
- 뮤직 플레이어에서는 음원을 재생해주는 역할만 할 뿐 해당 음원을 출력할 장치 또는 음향 기기는 사용자가 직접 준비하여야 한다.
- 해당 뮤직 플레이어는 스트리밍에 목적을 두고 있지 않으며, 로컬에 저장되어 있는 음원들을 재생함을 목적에 두고 있다.

3.2 General Constraints(제약사항)

제품 개발 과정의 통일된 규약 사항으로, 유지보수 비용을 줄이고 가독성을 높이기 위한 코딩 컨벤션과 개발 준수 사항 및 제약 사항을 다음과 같이 정의한다.

1) 코딩 컨벤션

- 구현 코드는 Javascript 언어로 작성한다.
- 도메인 소스코드는 js 확장자를 사용한다.
- React 로 관리하는 Custom Tag 의 소스코드는 jsx 확장자를 사용한다.
- 들여쓰기(indent)는 space 2 칸을 사용한다.
- 변수명과 함수, 그리고 메서드 네임은 Camel Case 로 작성한다.
- 클래스 이름과 Custom Tag 의 이름은 Pascal Case 로 작성한다.

2) 소스 코드 형상관리

- Git 을 이용하여 형상 관리한다.
- GitHub Organization 의 Repository 를 fork 해서 작업한 후 Pull

Request 를 작성한다.

- Git Branch 전략을 사용하여 feature 별로 작업한 내용을 merge 한다.

3) 개발 프로세스 제약 사항

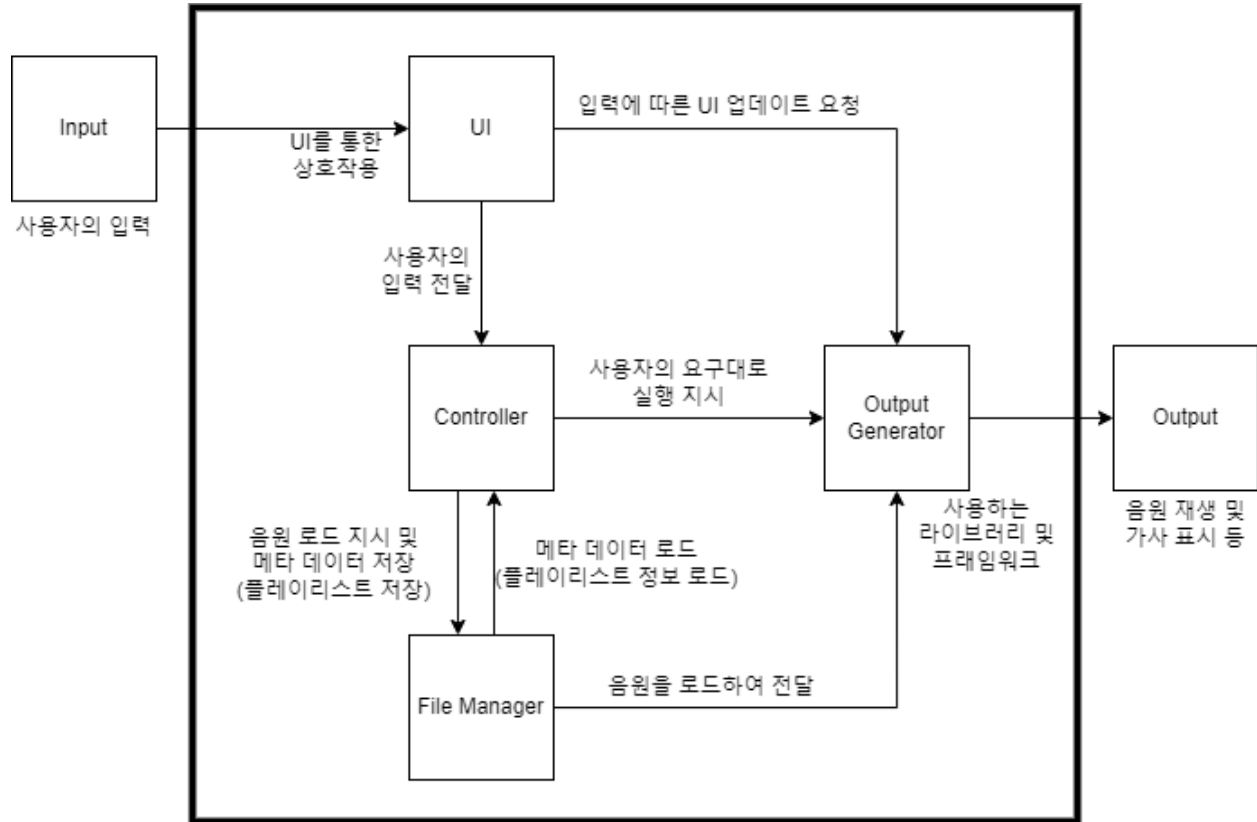
- 개발 기한(23.12.01)까지 제품 개발을 완료해야 한다.
- 클라이언트도 개발과 관련되어 전문적인 지식이 있기 때문에, On-site Customer 전략을 사용하여 클라이언트의 지속적인 피드백을 받고 이를 반영한다.

3.3 Goals and Guidelines(목표와 지침)

- 운영체제 내에서 실행될 뮤직 플레이어이기 때문에 리소스 (CPU 및 메모리 등)의 사용량을 최적화하여야 한다.
- 뮤직 플레이어의 사용자 편의성 극대화하여야 한다.
- 사용자의 다양한 요청들에 대해서 1 초 내에 반응을 하여 불편함이 없도록 한다.
- 음량을 임계치 이상으로 높이지 못하게 하여 사용자의 청력에 문제가 생기지 않도록 한다.
- 메타데이터들에 대해서 변경이나 손상이 없이 유지되도록 할 수 있어야 한다.
- 사용자의 편의성을 위하여 단축키를 제공하며, 단축키들은 가능한 영문 앞글자가 연상되어야 하며, 키 조합 개수가 2 개 이하여야 한다.

4. System Architecture(시스템 아키텍처)

4.1 Architectural Design(아키텍처 설계)



4.2 Design Rationale(설계 근거)

본 애플리케이션은 Layered Architecture 를 기반으로 동작한다. 아키텍처를 구성하는 계층들은 사용자와 상호작용하는 UI layer, 음원의 재생을 관리하는 Controller Layer, 그리고 음원을 로드하고 플레이리스트의 메타데이터를 관리하는 File Manager Layer 로 구성된다. 모든 Layer 는 자신과 인접한 Layer 만 데이터를 주고받을 수 있다. 이는 Layer 들 사이의 Coupling 을 낮추고, 각각의 Layer 가 하나의 모듈로서 동작하여 High Cohesion 을 만족시키려는 설계 원칙을 바탕으로 한다.

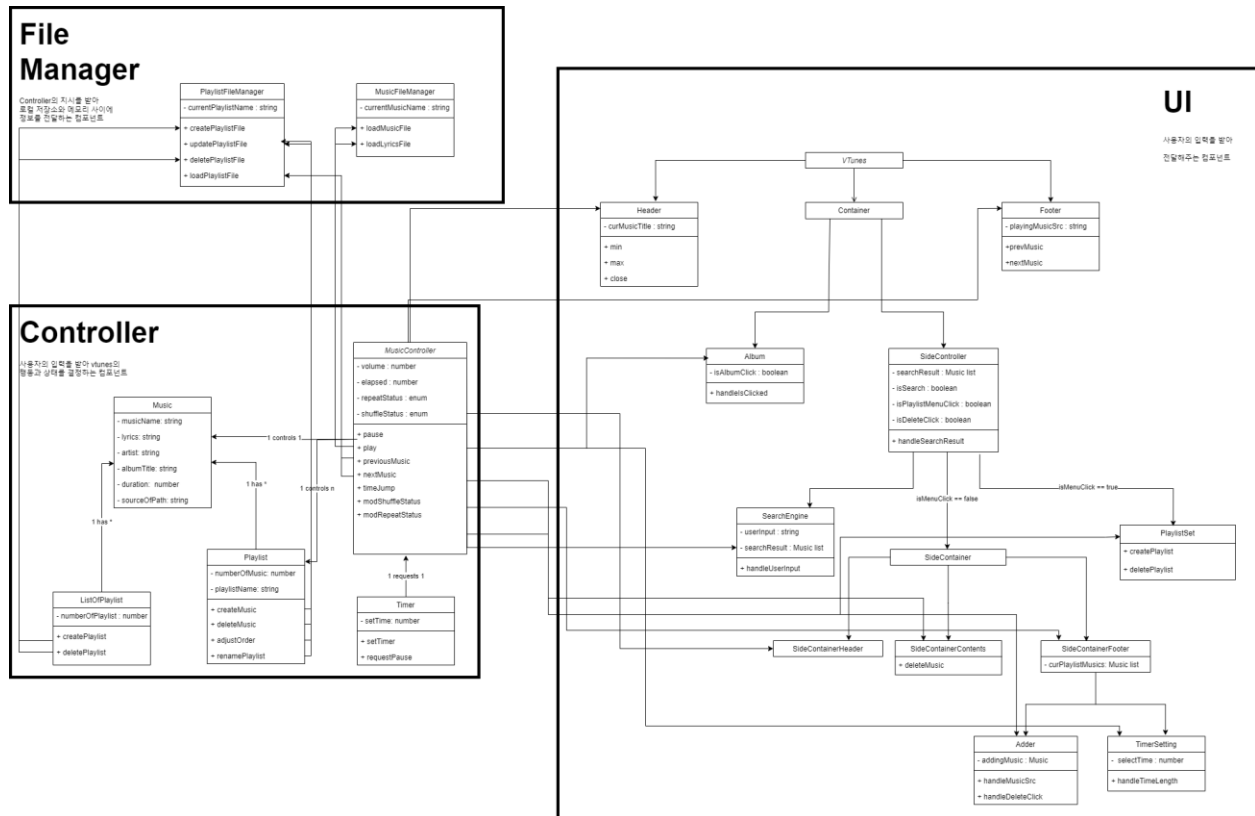
UI Layer 는 GUI 형태로 제공되어 사용자의 입력을 받는다. 클릭이나 키보드 입력은 이벤트(onClick, onKeyDown)로 동작하여 Controller Layer 의 Handling method 를

요청한다. Controller Layer 는 전달받은 이벤트에 대하여 음원의 재생을 제어하는 역할을 수행한다. 음원을 불러오거나 플레이리스트의 변경 사항이 발생하면 File Manager Layer 에 요청을 보낸다. File Manager Layer 는 운영체제의 파일시스템을 기반으로 음원파일과 메타 데이터를 관리한다.

사용자의 요청은 layer 들의 상호작용으로 처리되며 최종 output 은 Output Generator 에서 반환된다. Output Generator 는 React 라이브러리와 Electron 프레임워크가 담당하여, 변경된 UI 상태와 제어된 음원 상태에 관한 image data, text data, 그리고 audio data 를 output 으로 사용자에게 반환한다.

5 Detailed Design(상세 설계)

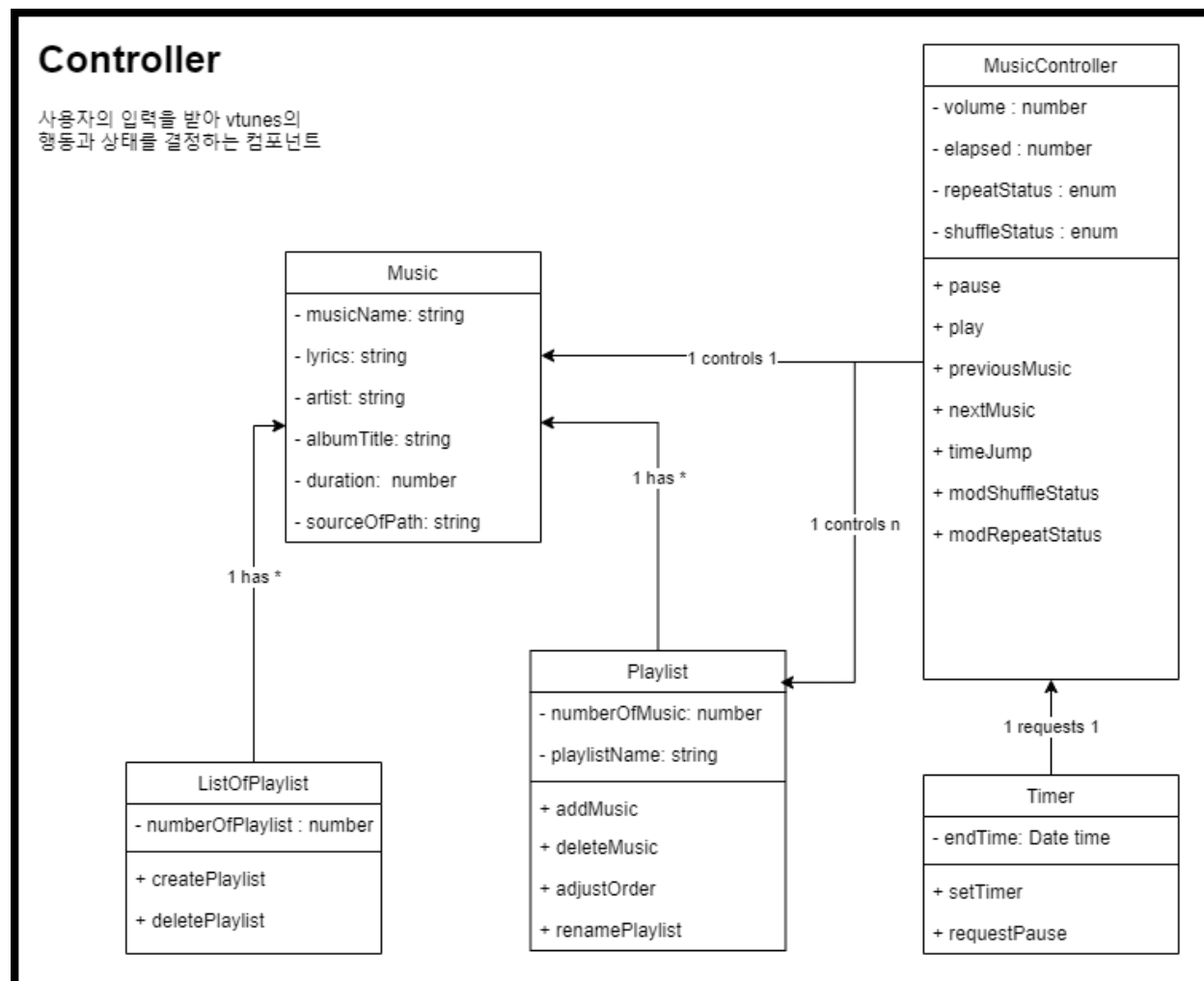
5.1 Diagram in Total



위 Diagram 은 해당 뮤직 플레이어 개발하는 과정에서 필요한 모든 Class (File Manager, Controller, UI)를 나타내고 있다. 본 애플리케이션은 Layered Architecture 로 설계되어 UI 와 Controller, File Manager 와 Controller 간의 관계가 있다는 것을 볼 수 있다. 이에 대한 자세한 정보는 이후 Low-Level or Component-Level Design, Interface Design 를 통해서 확인이 가능하다.

5.2 Low-Level or Component-Level Design(저수준 설계)

Component: Controller



Class: MusicController

음원 재생에 관련된 제어 로직을 담당하는 클래스이다. Façade Pattern 을 사용하여, MusicController 는 UI Layer 와 Controller Layer 의 컴포넌트들의 의존성을 중앙에서 관리한다. UI Layer 의 모든 요청은 MusicController 객체로 전달되고, MusicController 는 해당 요청을 처리하는 컴포넌트 각각에 수행할 일들을 분배한다. MusicController 객체는 UI Layer 전반에서 사용되는 Global Instance 로, 여러 instance 를 생성하여 관리하기보다 하나의 전역 상태로 관리하는 Singleton Pattern 으로 구현한다. 다만, Singleton 을 직접 구현하지 않고, Redux 라이브러리로

MusicController 의 객체 instance 를 전역 상태로 관리하는 방식을 택한다.

Field Name	Data Type	Description
currentPlaylist	Playlist	현재 재생하는 재생목록
currentMusic	Music	현재 재생 중인 음원
volume	number(int)	뮤직 플레이어의 음량
elapsed	number(int)	음원이 재생된 시간
repeatStatus	enum	반복 재생 설정 REPEAT_ON: 플레이리스트의 전체 반복 설정 REPEAT_CURRENT: 현재 음원의 한 곡 반복 설정 REPEAT_OFF: 반복 재생을 사용하지 않음
shuffleStatus	enum	셔플 재생 설정 SHUFFLE_ON: 무작위 재생 설정 SHUFFLE_OFF: 셔플 재생을 사용하지 않음

function	input	output	description
pause	onClick/onKeyDown event	void	재생 중인 음원을 정지한다.
play	onClick/onKeyDown event	void	정지된 음원을 재생한다.
previousMusic	onClick/onKeyDown event	void	플레이리스트의 이전 음원을 재생한다.
nextMusic	onClick/onKeyDown event	void	플레이리스트의 다음 음원을

	event		재생한다.
timeJump	onClick event	void	해당하는 재생 시간대로 이동한다.
modShuffleStatus	onClick event	void	ShuffleStatus 를 변경한다.
modRepeatStatus	onClick event	void	RepeatStatus 를 변경한다.

+ Commercial Library

개발 기한을 고려하여 음원 재생과 관련된 상용 라이브러리를 사용할 수 있다. 라이브러리를 사용하게 되면 재생 제어와 관련된 연산들(play, pause, timeJump 등)은 직접 구현하지 않아도 된다. 이를 통해서 획기적인 비용 절감을 예상할 수 있다.

Class: Music

음원 파일의 메타 데이터들을 저장하는 Data Structure 이다. Controller Layer 의 컴포넌트들 사이에서 전달되는 데이터로 사용되며, File Manager Layer 의 음원 파일을 불러올 때 사용하는 DTO(Data Transfer Object)로도 사용될 수 있다. Object 의 상태 정보를 반환하는 메서드 이외의 별도의 연산은 개발 도중에 필요하게 되면 추가하도록 한다.

Field Name	Data Type	Description
musicName	string	음원의 이름
lyrics	string	음원의 가사
artist	string	음원의 아티스트 이름
albumTitle	string	음원의 앨범 이름
duration	number(int)	총 재생 시간(초 단위로 저장)
sourceOfPath	string	음원 파일의 경로

+ Built-in File Attributes

Music 객체의 각 field 의 값은 음원 파일 자체의 내장된 값을 그대로 저장한다. 단, musicName 의 값이 음원 파일에 기록되어 있지 않다면, 확장자를 제외한 파일 이름을 가져온다. lyrics 는 음원 파일과 같은 경로 상에 가사 파일(음원 파일과 동일한 이름을 가지며, 확장자만 lrc 인 파일)이 있으면 그 파일을 읽어서 가져온다. 음원 파일에도 가사 정보가 기록되어 있지 않다면, empty string 을 저장한다.

Class: Playlist

Music 객체 instance 들을 Array 형태로 저장하는 컨테이너다.

Field Name	Data Type	Description
playlistName	string	플레이리스트의 이름
NumberOfMusic	number(int)	플레이리스트의 포함된 음원의 수
MusicList	List(Music)	음원의 재생목록

function	input	output	description
addMusic	Music	void	Playlist 에 input Music 을 추가한다.
deleteMusic	index(int)	void	Playlist 에서 Index 에 해당하는 Music 을 삭제한다.
adjustOrder	from(int), to(int)	void	Playlist 내의 from 위치의 음원을 to 위치로 이동한다.
renamePlaylist	name(String)	void	Playlist 의 이름을 name 으로 교체한다.
shufflePlaylist	void	Playlist	Playlist 의 순서를 무작위로 재배열한 가상의 Playlist 를

			반환한다.
--	--	--	-------

+ Shuffle Algorithm

플레이리스트의 음원 순서를 무작위로 재배치한 가상의 플레이리스트를 만든다. Fisher-Yates shuffle 알고리즘을 사용하여 $O(N)$ 의 시간 복잡도를 가질 수 있으며, 기존 플레이리스트의 원본 상태는 변경하지 않고 가상의 플레이리스트를 새로 생성하는 방식으로 구현한다.

```

1| copy array from origin playlist
2| for i from n-1 downto 1 do
3|   j ← random integer such that 0 ≤ j ≤ i
4|   exchange array[j] and array[i]
5| create virtual Playlist with shuffled array

```

가상의 플레이리스트는 UI Layer 에 표시되지 않고, 재생되는 음원에 따라 플레이리스트의 순서를 위아래로 이동하며 재생되는 형태로 보여준다. 셔플 재생 중에 새로운 음원이 플레이리스트에 추가된다면, 추가된 음원을 포함하여 음원 순서를 새로 재배치한다.

Class: ListOfPlaylist

Playlist 의 전체 목록을 Array 형태로 저장하는 컨테이너다.

Field Name	Data Type	Description
NumberOfPlaylist	number(int)	전체 플레이리스트의 수
ListOfPlaylist	List(Playlist)	Playlist 의 전체 목록

function	input	output	description
createPlaylist	name(String)	void	Empty Playlist 를 생성한다.
deletePlaylist	index(int)	void	해당하는 index 의 Playlist 를 삭제한다.

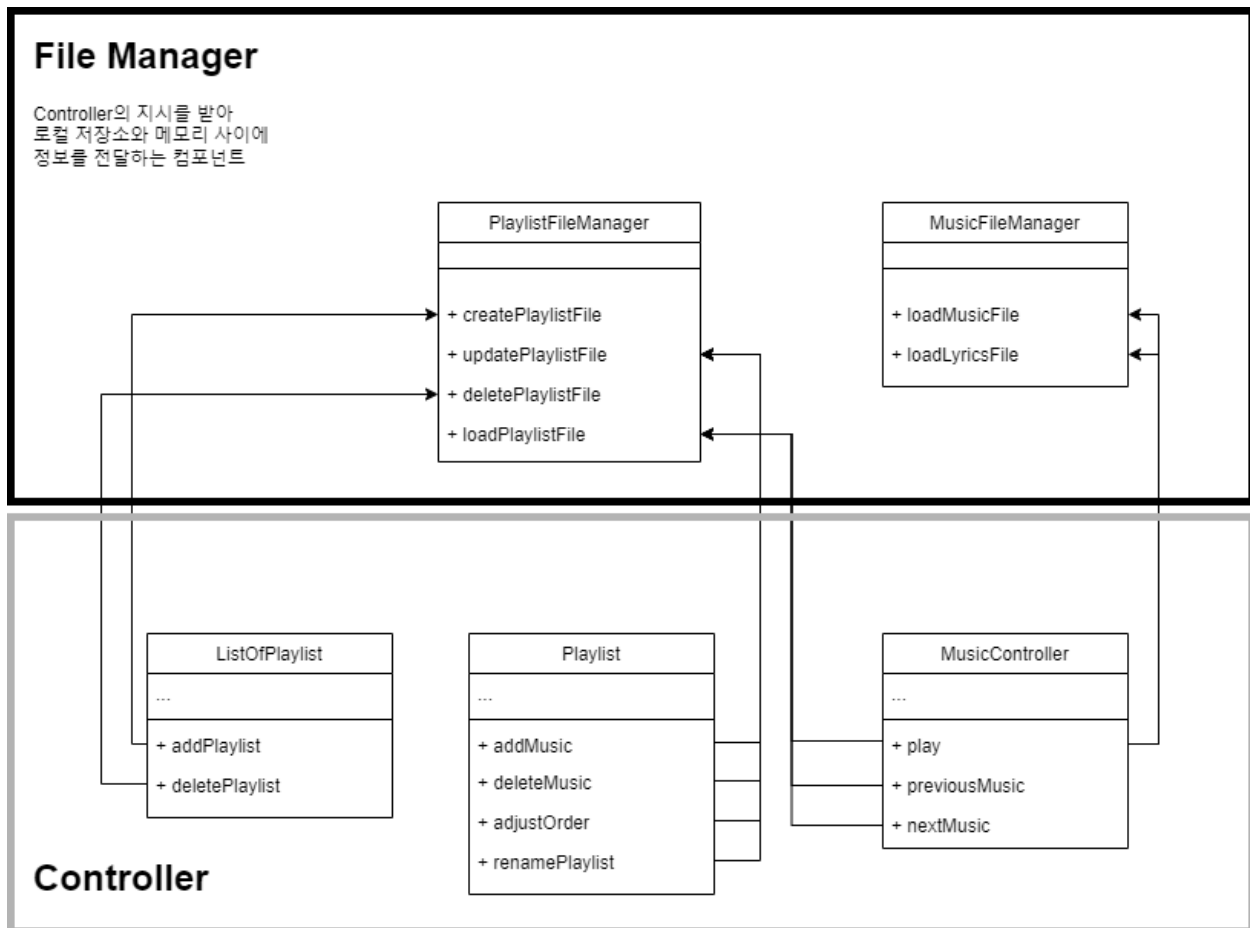
Class: Timer

타이머 기능을 구현할 때에 필요한 객체로, 타이머로 설정된 시간이 지나면 MusicController 에 재생 정지를 요청한다.

Field Name	Data Type	Description
endTime	DateTime	타이머로 설정된 정지 예정 시간

function	input	output	description
setTimer	time(int)	void	정지할 시간을 설정한다.
requestPause	void	void	Controller 에게 타이머를 요청한다.

Component: File Manager



Class: PlaylistFileManager

플레이리스트의 메타 데이터를 파일로 관리하는 클래스이다.

function	Input	output	description
createPlaylistFile	name(String)	void	Playlist 파일을 생성한다.
updatePlaylistFile	Music	void	Playlist 파일의 변경사항을 갱신한다.
deletePlaylistFile	path(String)	void	Playlist 파일을 삭제한다.
loadPlaylistFile	path(String)	void	Playlist 메타 데이터 파일을 불러온다.

Class: MusicFileManager

음원 파일과 가사 파일을 불러오는 클래스이다.

function	Input	output	description
loadMusicFile	path(String)	void	Path 의 음원 파일을 불러온다.
loadLyricsFile	path(String)	void	Path 의 가사 파일을 불러온다.

+ File Structure

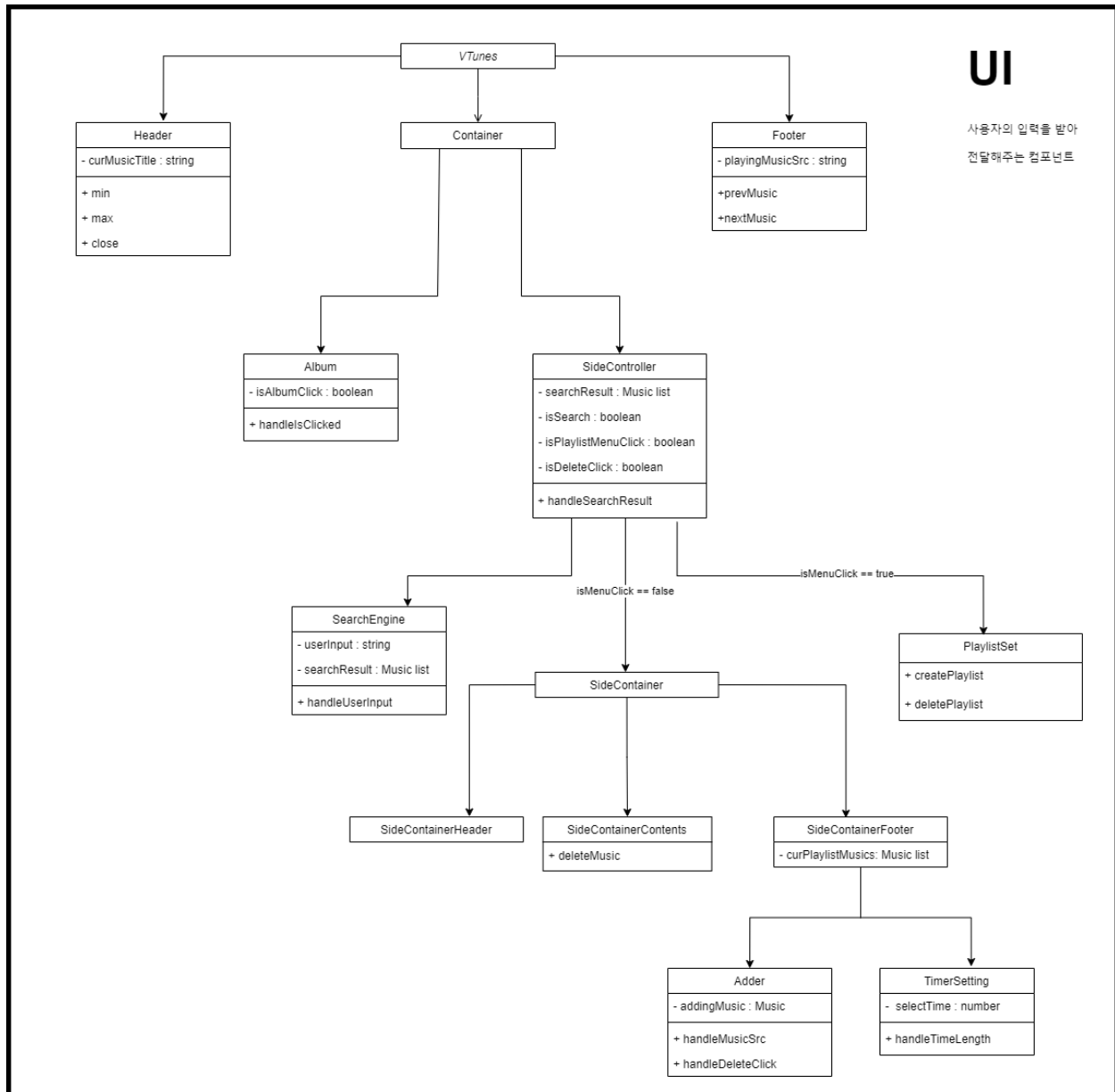
메타 데이터의 정보는 json 형식의 파일로 local storage 에 저장한다. Music 과 Playlist 의 객체 instance 를 직렬화한다. 플레이리스트마다 파일을 각각 따로 저장한다. Json 형식의 파일은 확장성이 좋고 Javascript 언어로 구현하기에 간편하고 용이하다는 장점이 있다. 개발 기한도 고려했을 때, VTunes 의 File Structure 로 json 형식을 선택했다.

Metadata: (playlist_name).json

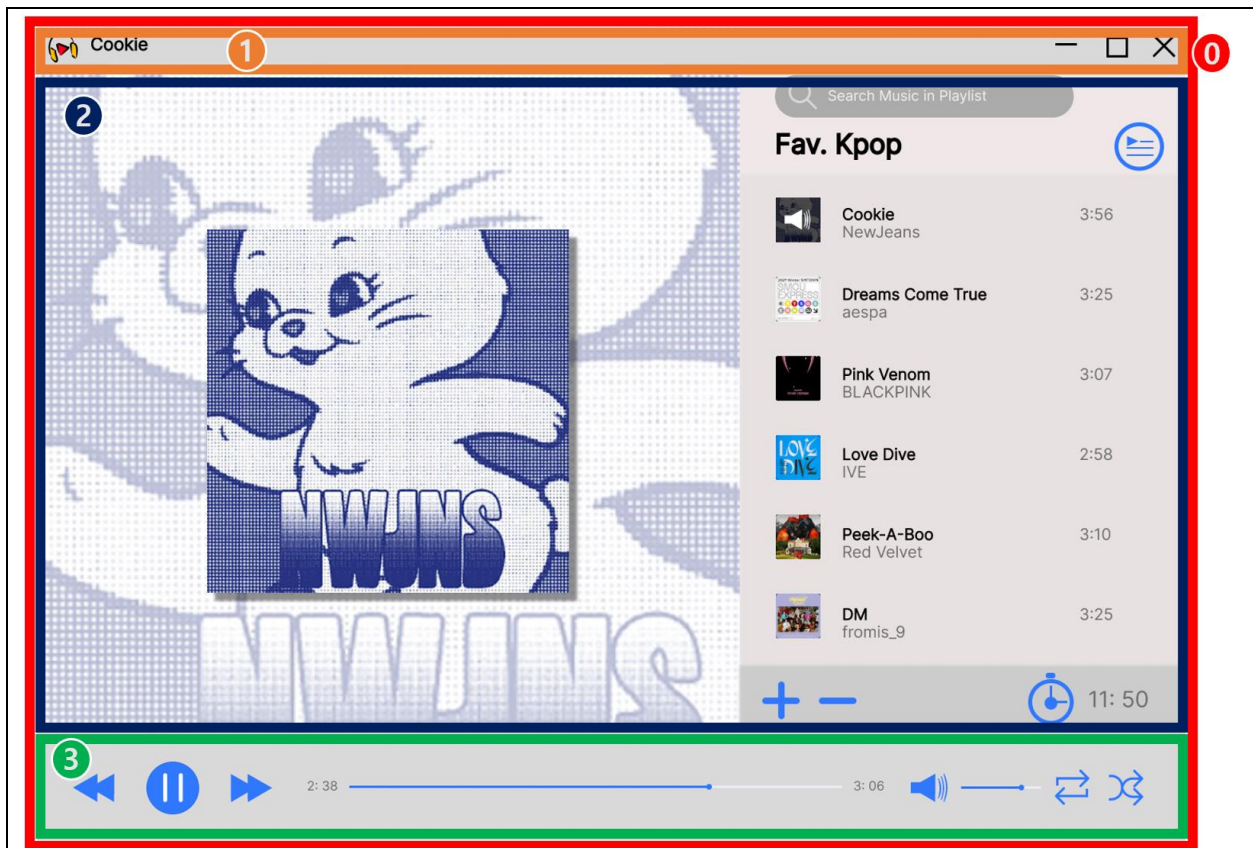
하지만 json 파일은 텍스트 에디터로 자유로운 수정이 가능하기 때문에, 메타 데이터 파일의 무결성은 보장해줄 수 없다. 플레이리스트의 메타데이터가 변경될 시 파일을 읽는데 문제가 발생할 수 있다. 예를 들어, 필드명을 변경하면 File Manager 가 데이터를 읽을 수 없다. 또 NumberOfMusic 필드의 값을 변경하면 음원을 선택할 때 ArrayOutOfBound 예외로 undefined 가 반환될 수 있다. 따라서 자체의 메타 데이터 파일 구조가 필요해질 수 있다. 본 문서의 Recommendation section 에서 추후 업데이트할 파일 구조에 대한 설계를 제시하고 있다.

5.3 Interface Design(인터페이스 설계)

Interface Design(인터페이스 설계)



VTunes Component



1. Header

Header 에는 VTunes 의 아이콘, 현재 재생 중인 음원의 제목, 그리고 컨트롤 버튼이 존재한다.

따라서 Header 에서 현재 재생 중인 음원의 제목을 변수로 갖고 있으며, 최대화, 최소화, 닫기를 함수로 갖고 있다.

2. Container

Container 에서는 현재 재생 중인 음원의 정보와 현재 선택되어 있는 플레이리스트 안에 담겨 있는 음원들에 대한 정보를 얻을 수 있다.


3. Footer

Footer 에서는 현재 재생 중인 음원에 대한 일시정지, 재생 기능이 있으며 이전 곡 재생 및 다음 곡 재생도 가능한 버튼이 존재한다. 추가적으로 현재 음원의 재생 바와 음량 컨트롤 바가 있으며, 셔플 기능을 온 오프 할 수 있고, 반복재생 기능을

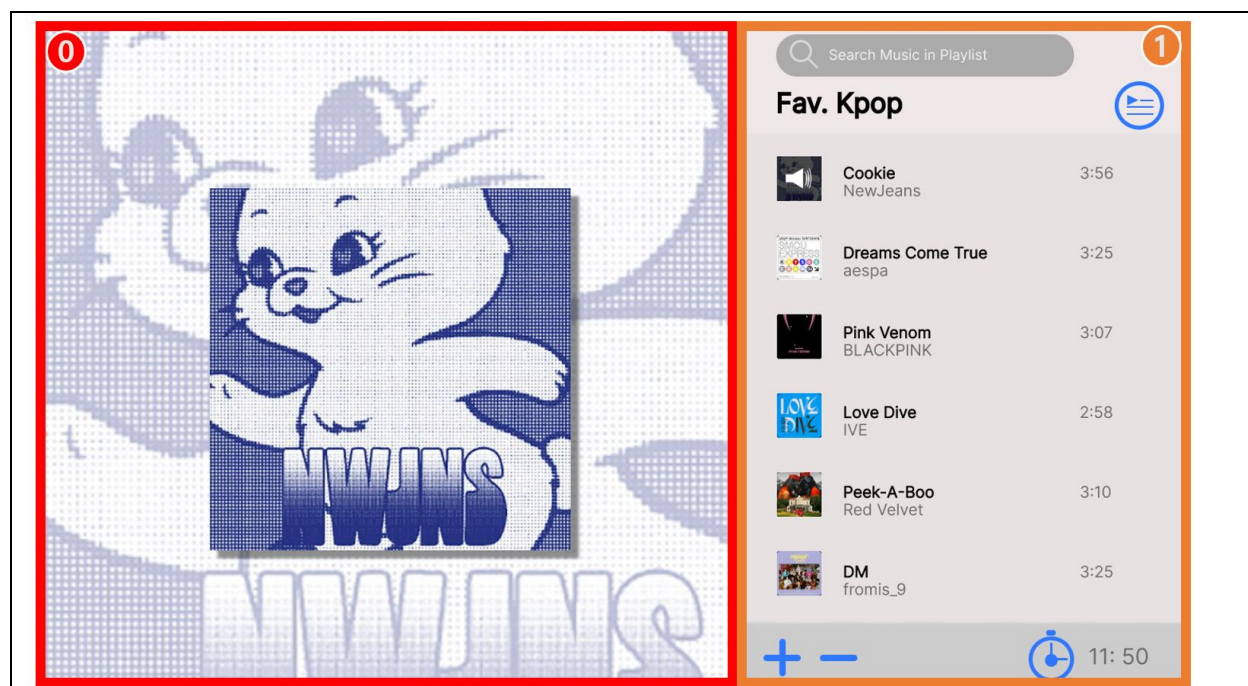
세 가지 기능들 중에서 한 가지로 선택이 가능하다.

따라서 Footer 에서는 현재 재생 중인 음원의 주소를 변수로 갖고 있으며, 이전 곡 재생과 다음 곡 재생과 관련된 함수를 갖고 있다.

Header

 A horizontal bar representing a music player header. It has a light gray background with an orange border. On the left, there is a small icon (0) and the word 'Cookie' (1). On the right, there are three buttons: a minus sign (2), a square (4), and an 'X' (5). Below the buttons are three small circles with numbers 3, 4, and 5 respectively. The entire bar is highlighted with an orange border.	
0. Favicon-box	Favicon-box 에는 VTunes 의 아이콘을 배치한다.
1. Header-title	Header-title 에서는 현재 재생 중인 음원의 제목을 보여주고 있다.
2. Control-btns	Control-btns 에서는 최소화, 최대화, 닫기 버튼이 존재하며 해당 버튼을 클릭할 시, 창을 최소화, 최대화, 혹은 닫게 된다.

Container



0. Album

Album 에서는 현재 재생 중인 음원의 앨범 사진을 보거나 가사를 확인할 수 있다. isAlbumClick 이라는 변수를 통해서 해당 부분에 가사를 확인할 수 있게 하거나 앨범 사진을 보이게 하는 것 두 가지 선택지를 돌아가면서 바꿀 수 있도록 허용한다.

1. SideController

SideController 에서는 현재 재생 중인 음원이 포함되어 있는 플레이리스트에 존재하는 다른 음원들에 대한 정보를 확인할 수 있으며, 플레이리스트 버튼을 통해서 VTunes 에 저장되어 있는 플레이리스트들을 확인할 수 있다. 추가적으로 해당 플레이리스트에 있는 음원들에 대해서 검색도 가능하다.

searchResult 변수에서는 플레이리스트에서의 음원을 검색 시에 해당 검색 결과에 나오는 음원들에 대한 리스트를 갖고 있다.

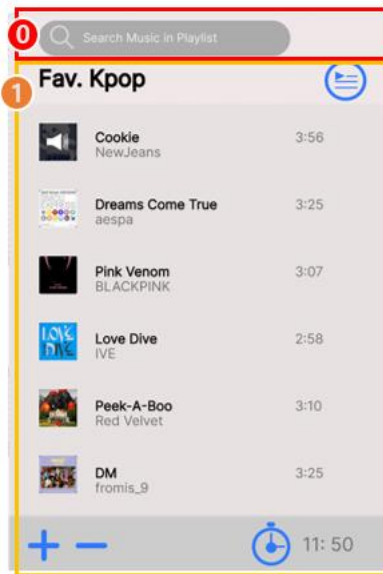
isSearch 라는 변수는 플레이리스트 검색창에 사용자가 음원을 검색했는지 여부를 저장하는 변수이다.

isPlaylistMenuClick 이라는 변수는 플레이리스트 버튼이 활성화되었는지에 대한

여부를 저장하는 변수이다.

isDeleteClick 이라는 변수는 플레이리스트 내에서 곡 삭제 버튼을 활성화되었는지에 대한 여부를 저장하는 변수이다.

SideController



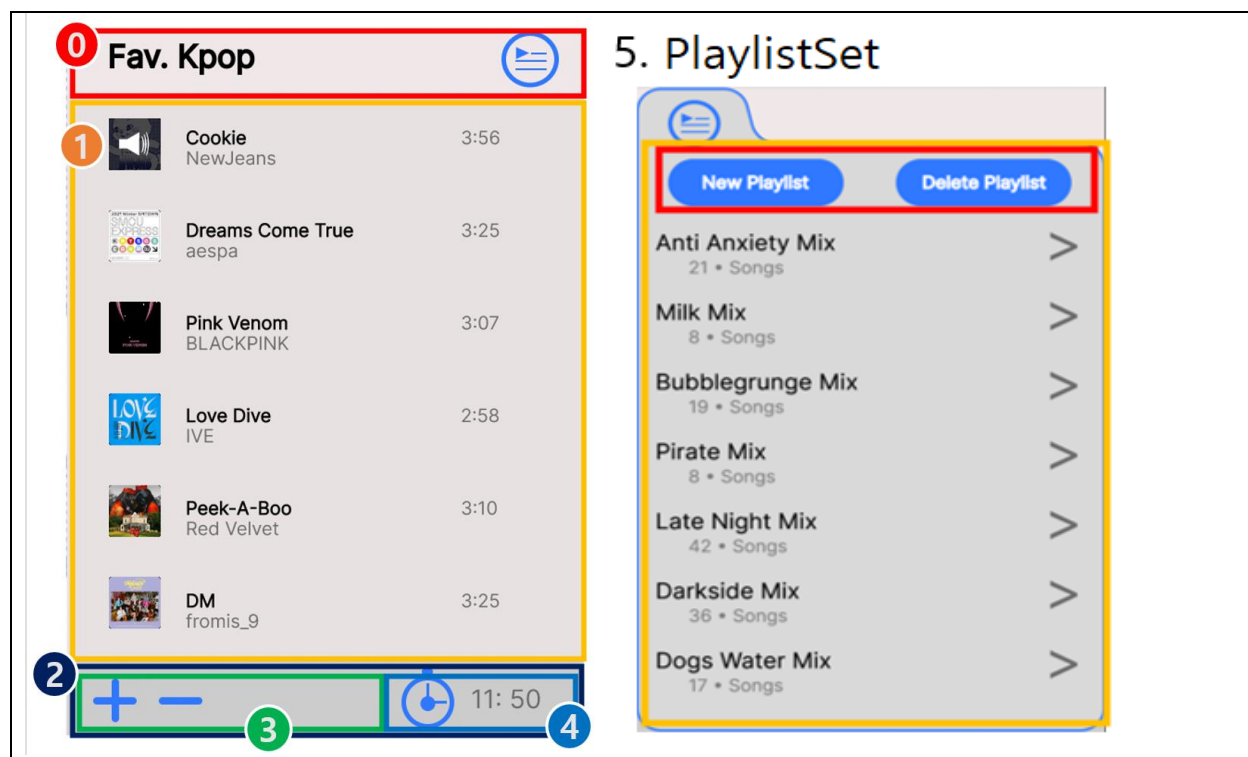
0. SearchEngine

SearchEngine 에서는 현재 사용자가 선택한 플레이리스트에서 원하는 음원을 제목을 기준으로 검색이 가능하다. 이 때, userInput 을 변수로 갖고 있어서 해당 사용자가 어떤 제목을 검색하고 있는지 입력을 받게 되며, searchResult 의 변수에는 검색의 결과 값을 Music list 로 갖고 있게 됩니다.

1. SideContainer

SideContainer 에서는 현재 사용자가 선택한 플레이리스트와 관련된 정보를 확인할 수 있다. 따라서 해당 플레이리스트의 제목과 플레이리스트에 있는 음원들에 대한 간단한 정보 (음원 제목, 음원 가수, 음원 길이)에 대해서 파악할 수 있으며, 음원을 추가하거나 삭제할 수 있고 타이머를 설정할 수 있게 한다.

SideController



0. SideContainerHeader

SideContainerHeader 에는 사용자가 선택한 현재 플레이리스트의 제목을 보여주고 있으며, 다른 플레이리스트를 선택할 수 있는 버튼이 존재한다.

1. SideContainerContents

SideContainerContents 에는 사용자가 선택한 현재 플레이리스트에 존재하는 음원들에 대한 정보들을 MusicInfoContainer 를 통해서 확인할 수 있다. 따라서 해당 섹션에서 각 음원의 제목, 가수, 앨범 사진과 음원의 재생 길이를 확인할 수 있다. 추가적으로 deleteMusic 함수가 존재하는데 해당 함수는 SideContainerContents 에서의 MusicAdder 에서 Delete 버튼이 클릭 되면 활성화되는 부분이며 해당 함수를 통해서 플레이리스트에 존재하는 음원을 삭제할 수 있다.

2. SideContainerFooter

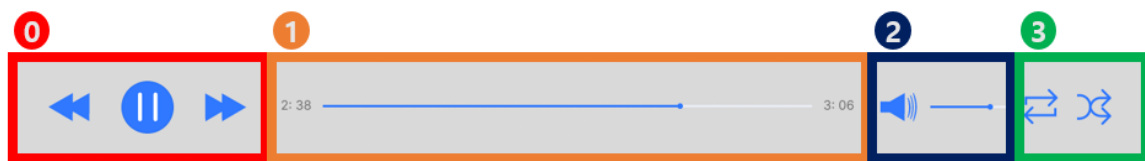
SideContainerFooter 에서는 크게 2 가지 부분으로 또 다시 나뉘었다. 첫 번째는 MusicAdder 이며 이 부분에서는 플레이리스트에 음원을 추가하거나 삭제할 수

있다. 두 번째는 Timer 로 해당 음원 플레이어의 언제까지 실행될 지 타이머를 생성할 수 있다.

5. PlaylistSet

PlaylistSet 에서는 음원 플레이어에 저장되어 있는 모든 플레이어에 대해서 설정이 가능하다. 따라서 해당 부분에서 새로운 플레이어를 추가하거나 삭제할 수 있는 기능이 있다.

Footer



0. MusicControl

MusicControl 에서는 현재 재생 중인 음원에 대해서 설정할 수 있다. 따라서 해당 부분에서 이전 곡과 다음 곡을 재생할 수 있도록 설정할 수 있으며 현재 재생 중인 음원을 일시정지 하거나 재생할 수 있다.

1. StatusBar

StatusBar 에서는 현재 재생 중인 음원이 어느 정도 재생이 되었는지 해당 비율에 맞게 바가 채워질 수 있도록 설정되어 있다. 해당 바 좌측에는 해당 음원이 얼마나 재생되었는지를 표시하고 있으며, 우측에는 음원의 전체 길이를 표시하고 있다. 추가적으로 해당 바에서 일정 위치로 움직이게 되면 해당 비율에 맞는 부분부터 음원을 재생할 수 있다.

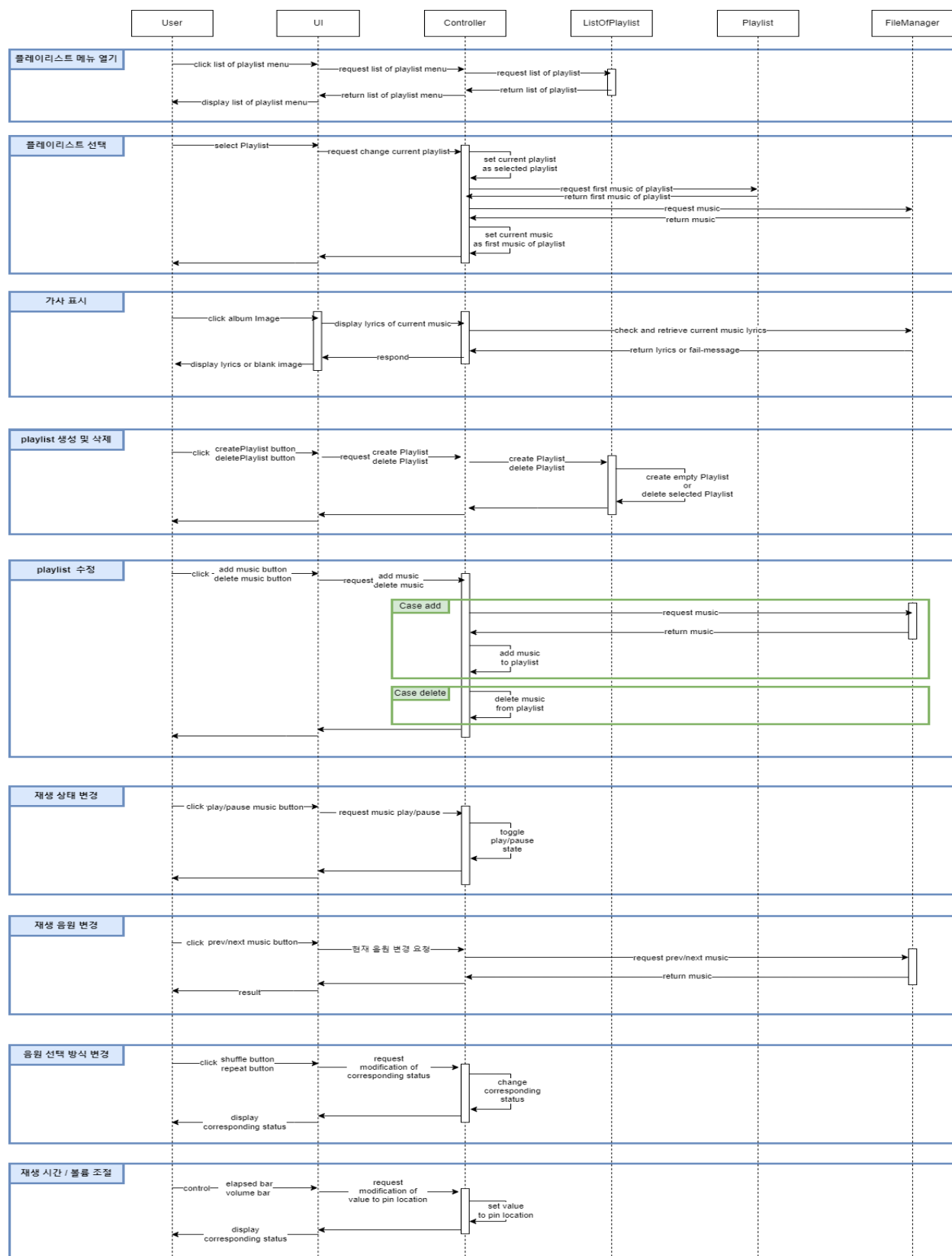
2. VolumeBar

VolumeBar 에서는 음원의 음량을 조절할 수 있다. 음량의 어느 정도인지 해당 비율에 맞게 바가 채워질 수 있도록 설정되어 있다. 해당 바를 움직여서 음원을 들으면서 사용자가 원하는 음량에 맞게 음원을 청취가 가능하다. 추가적으로 해당 바 좌측에 스피커 모양의 아이콘을 추가하여 해당 아이콘을 클릭하였을 경우에는 음원 플레이어 전체가 음소거가 된다.

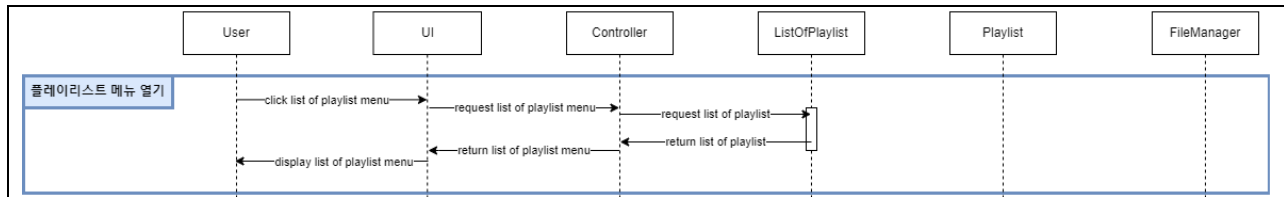
3. ShuffleRepeat

ShuffleRepeat에서는 해당 플레이리스트 내에서의 곡 재생 방법에 대해서 설정할 수 있다. Shuffle은 해당 플레이리스트에 있는 곡들의 재생 순서를 랜덤하게 바꾸게 되며, Repeat의 경우에는 한 곡 반복 재생, 플레이리스트 전곡 한 번 재생, 플레이리스트 반복 재생 중 하나를 사용자가 고를 수 있다.

5.4 Sequence Diagram

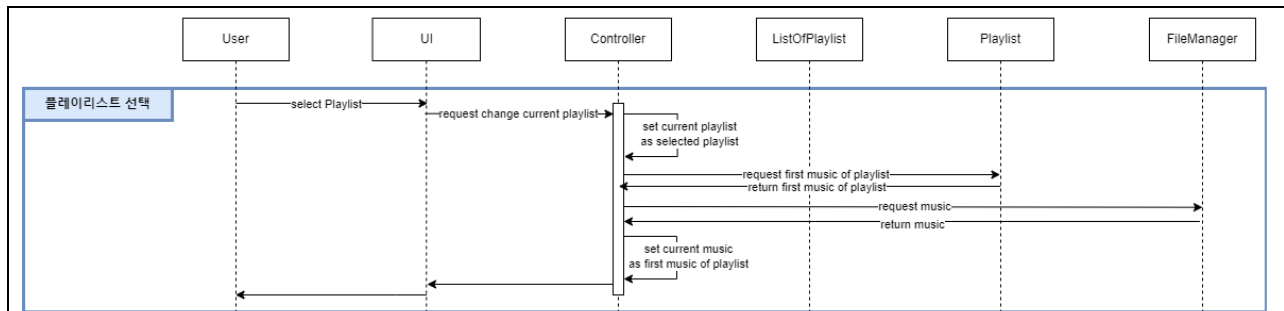


1. 플레이리스트 메뉴 열기



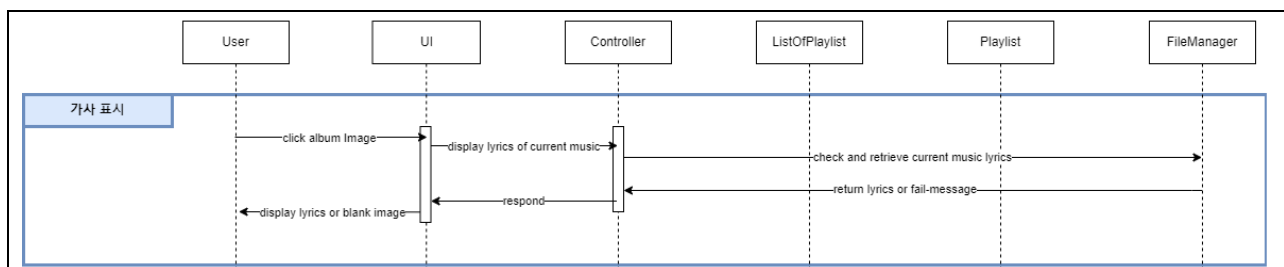
사용자가 UI 에 존재하는 listOfPlaylistMenu 를 클릭하면 UI 가 Controller 로 listOfPlaylistMenu 를 요청한다. 그 이후에 Controller 에서 ListOfPlaylist 로 ListOfPlaylist 를 요청한다. 그러면 ListOfPlaylist 에서 Controller 로 listOfPlaylist 를 반환한다. 이후, Controller 에서 listOfPlaylistMenu 를 UI 로 반환한다. UI 에서 listOfPlaylistMenu 를 User 에게 보여준다.

2. 플레이리스트 선택



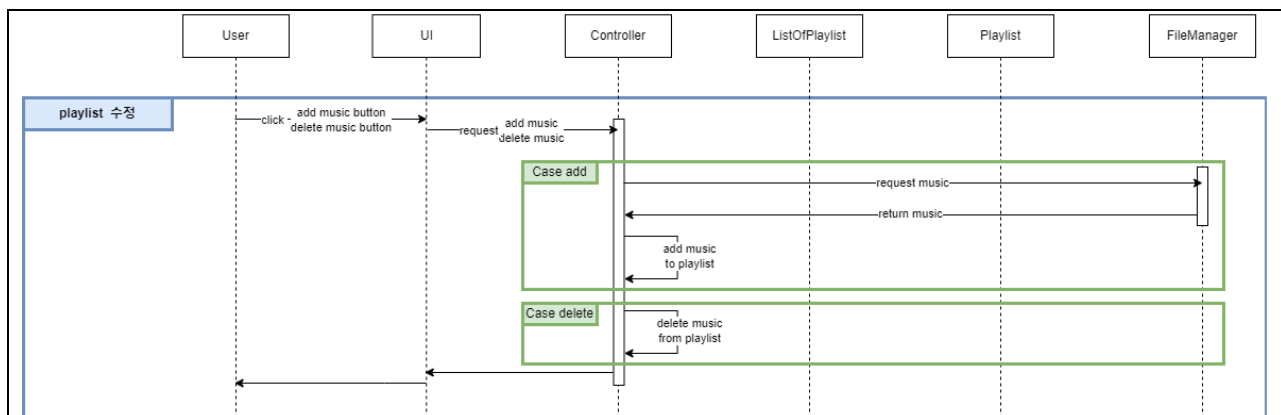
사용자는 UI 에서 본인이 원하는 플레이리스트를 선택한다. UI 가 Controller 에게 사용자가 선택한 플레이리스트의 정보를 전달하고 해당 플레이리스트를 currentPlaylist 로 바꾸게 된다. Controller 는 Playlist 에게 해당 플레이리스트의 첫 번째 음원과 관련된 정보를 요청하면 Playlist 는 Controller 에게 해당 음원과 관련된 정보를 받는다. 해당 정보를 바탕으로 FileManager 에게 해당 음원을 요청 후, 해당 음원을 받으면 해당 음원을 currentMusic 으로 바꾸게 된다. 이후 Controller 는 해당 음원을 사용자가 청취 가능하도록 주변 출력 장치로 출력을 가능하게 한다.

3. 가사 표시



사용자가 UI 에서 앨범의 이미지를 클릭하면 UI 에서 Controller 로 currentMusic 의 가사를 요청한다. 이후, Controller 에서 FileManager 로 currentMusic 의 가사의 유무를 확인하고 가사가 존재할 경우, 해당 가사를 요청한다. FileManager 에서 listOfPlaylist 로 가사를 반환하거나 오류 메시지를 반환한다. Controller 에서 UI 로 해당 음원의 가사를 제공한다. UI 에서 사용자에게 가사를 보여주거나 빈 이미지를 보여준다.

4. 플레이리스트 수정



사용자가 UI 에서 Add music 혹은 Delete music 버튼을 클릭한다. UI 는 Controller 에게 플레이리스트에 음원을 추가하거나 음원을 삭제하겠다는 요청을 보낸다.

1. 플레이리스트에 음원 추가 시,

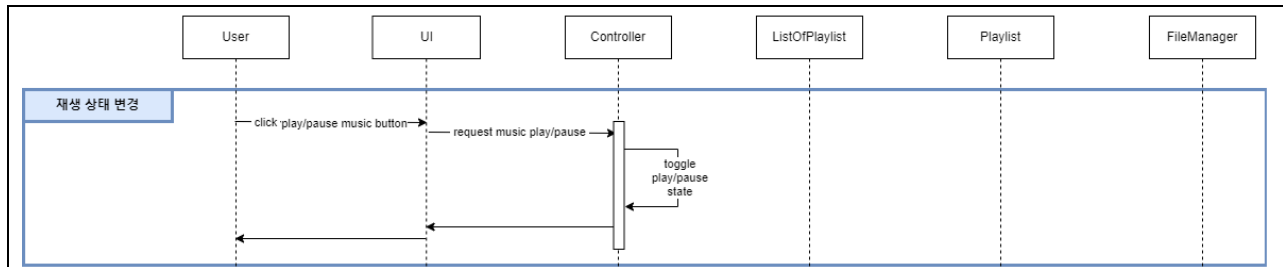
Controller 는 FileManager 에게 음원을 요청하게 되고, 해당 음원의 정보를 FileManager 로부터 반환 받게 된다. 이후 전달받은 음원 정보를 플레이리스트에 음원을 추가하게 된다.

2. 플레이리스트에 음원 삭제 시,

Controller 자체에서 해당 음원을 플레이리스트에서 삭제를 하게 됩니다.

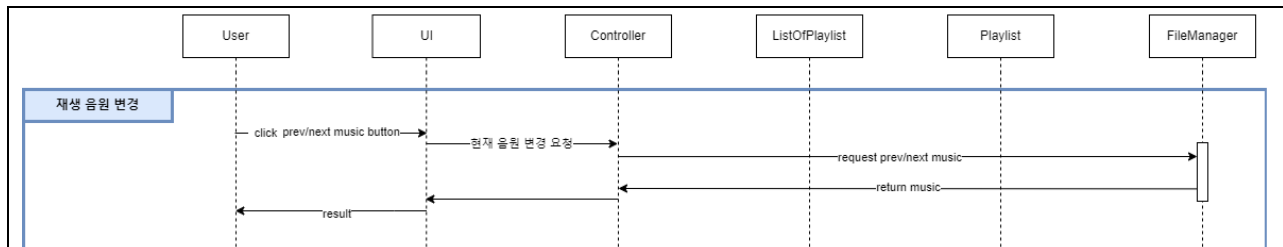
이후 수정된 플레이리스트에 관련된 정보를 다시 UI 를 거쳐서 사용자에게 제공하게 된다.

5. 재생 상태 변경



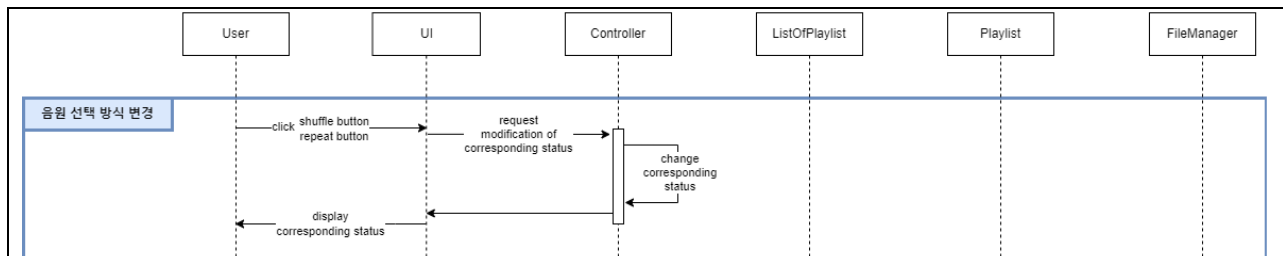
사용자가 UI 에서 play 나 pause 버튼을 클릭한다. 이후 UI 에서 Controller 로 음악 재생이나 일시정지를 요청한다. Controller 에서 play 나 pause 상태를 토글한다. UI 에서 토글된 play 나 pause 상태를 반영하여 사용자에게 보여준다.

6. 재생 음원 변경



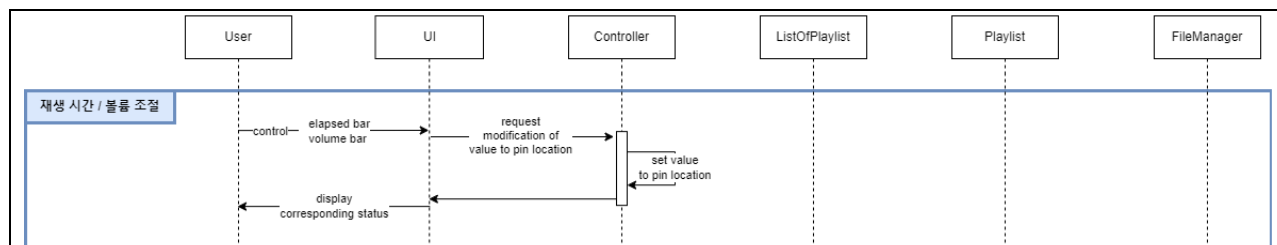
사용자가 UI 에서 prev 나 next 버튼을 클릭한다. 이후 UI 에서 Controller 로 현재 재생 중인 음원의 이전 음원이나 다음 음원을 요청한다. Controller 에서 FileManager 로 이전 음원이나 다음 음원의 정보를 요청한다. FileManager 에서 해당 음악을 Controller 로 반환한다. Controller 에서 UI 로 음악을 반환하고, UI 에서 해당 음악을 재생한다.

7. 음원 선택 방식 변경



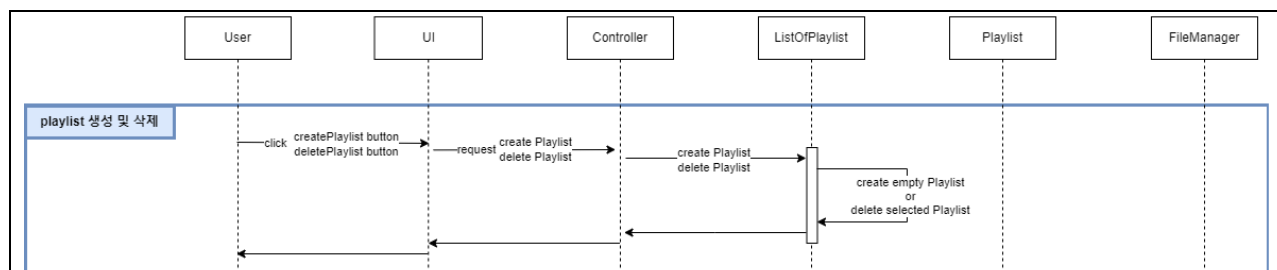
사용자가 shuffle 또는 repeat 버튼을 클릭한다. UI 는 해당 부분에 알맞은 수정사항을 Controller 에 넘겨준다. Controller 는 해당 수정사항을 음원 선택 방식을 변경하게 된다. 해당 변경된 사항을 UI 를 통해서 사용자에게 표시하게 된다.

8. 재생 시간 / 볼륨 조절



사용자가 elapsed 또는 volume 바를 이용하여 재생 시간 또는 볼륨을 조절하게 된다. 해당 재생 시간 또는 볼륨에 맞게 Controller에서 세팅을 하게 된다. 이후 변경된 사항을 UI를 통해서 사용자에게 표시하게 된다.

9. 플레이리스트 생성 및 삭제



사용자가 create playlist 또는 delete playlist 버튼을 클릭하게 된다. 해당 부분에 따라서 Controller에게 플레이리스트 생성 또는 삭제를 요청하게 된다. Controller는 ListOfPlaylist에게 플레이리스트를 생성 또는 삭제하게 되고 이를 사용자에게 다시 출력하게 된다.

6. Recommendation

6.1 File Structure

File Type	Header Signature(Hex)	Footer Signature(Hex)
VPL	56 54 75 6E 65 73	56 54 75 6E 65 6E 64

File Structure: VPL(VTunes PlayList metadata)

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF		
0x0000	56	54	75	6E	65	73	# of Musics				Playlist Name(variable length)							
0x0010	00 Name of the First Music(variable length)								01 File Path(variable length)									
0x0020	02 Artist Name(variable length)								03 Thumbnail Path(variable length)									
0x0030	04 Album Name(variable length)								05 Duration				Order					
0x0040	00 (Continue on Next Music...)								56				54	75	6E	65	6E	64
Header Signature: VTunes								Footer Signature: VTunend										

Playlist 에 대한 메타데이터를 저장하기 위해 다음과 같은 vpl 확장자를 가지는 파일 구조를 정의하였다. 새로운 파일 구조를 정의한다면 json 과 같은 텍스트 파일로 메타데이터를 관리하는 것보다 메타데이터의 파일 무결성을 더 잘 보존할 수 있을 것으로 예상하였다. 즉, 사용자가 뮤직플레이어를 통한 playlist 조작 외에 직접적으로 메타데이터를 조작하여, 메타데이터의 파일 무결성을 해치는 상황을 막을 수 있을 것이다.

하지만, 클라이언트와 상의한 결과, 새로운 메타데이터 파일 구조를 정의하여 사용하는 것은 시간과 비용이 더 들 것으로 예상되어, 사용자가 직접적으로 메타데이터를 수정할 수 있다는 리스크를 감안하고 텍스트 형태의 json 으로 메타데이터를 저장하기로 결정하였다.

6.2 Caching

음원 플레이어를 제작하는데 있어서 사용자의 요청에 따라 1 초 이내로 반응을 하고자 하는 목표 사항이 존재했다. 먼저 Caching 을 이용하지 않고 개발을 하고 난 뒤에 다양한 testing 방법을 통해서 테스트를 해볼 예정이다. 해당 결과 값이 1 초 이내로 반응에 성공을 하게 된다면 추가적으로 Cache 를 이용할 이유가 없다고 판단이 된다. 하지만, 1 초 이내로 반응을 하지 못할 경우에는 해당 목적사항을 만족하기 위하여 Caching 과정을 추가적으로 개발할 예정이다.