

Music Player

Listen to Songs and Playlists

허기범



목 차

1. 프로젝트 개요 (p.3)

프로젝트 소개 및 목표
사용 언어 및 기술
개발 일정

2. 설계 및 기획 (p.7)

주요 UI 디자인 및 동작

3. 프론트엔드 (p.12)

코드 분석

4. 백엔드 (p.23)

코드 분석

5. 후기 (p.26)

"Where words fail,
music speaks."



01 프로젝트 개요

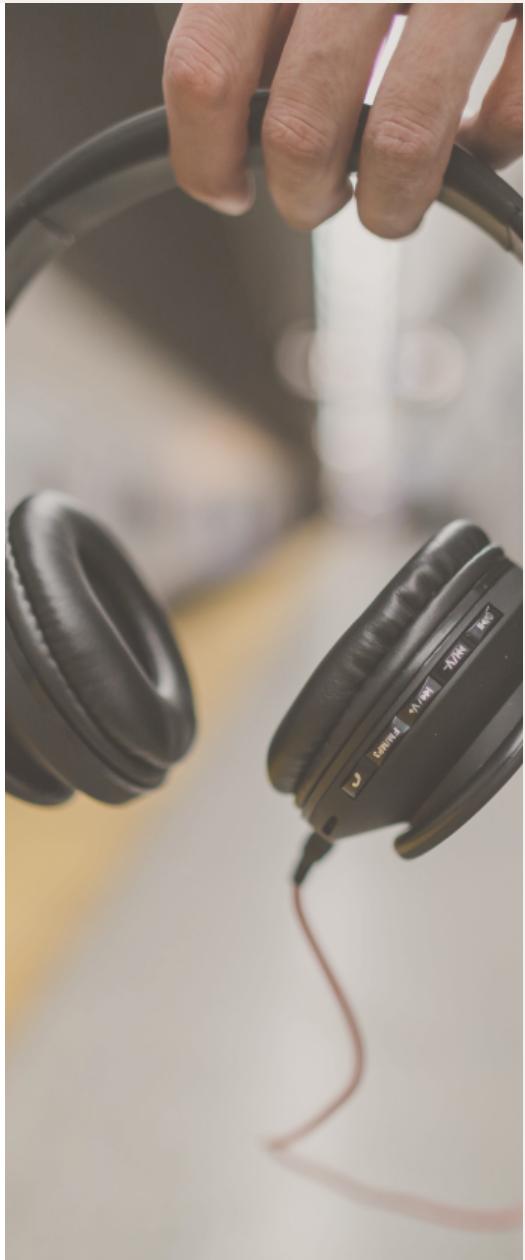


01 프로젝트 개요

MUSIC PLAYER

뮤직 플레이어

"사용자 친화적인 웹 기반 뮤직 플레이어로,
재생, 즐겨찾기, 셔플 등 다양한 기능을 제공
합니다."



주요 목표

01. 사용자 중심의 인터페이스 제공

사용자 편의성과 직관성을 높인 뮤직 플레이어 개발.

반응형 디자인을 적용해 다양한 화면 크기(데스크톱, 태블릿, 모바일)에서
최적의 사용자 경험을 제공.

02. 시각적 요소 강화

현대적인 디자인과 이퀄라이저 시각화를 통한 시각적 즐거움 제공.

플레이어의 배경 이미지를 동적으로 변경하여 몰입감을 제공.

애니메이션 등의 적절한 시각적 효과로 직관성 향상.

03. 학습 관련

프론트엔드와 백엔드 간의 데이터 흐름을 설계하고 구현하며,
데이터베이스(MariaDB)와의 연동 경험.

Deque와 같은 자료 구조를 설계하여 재생 목록을 효율적으로 관리.

01 프로젝트 개요



TOOLS AND TECHNOLOGIES

사용 언어 및 기술



Node.js, Express

클라이언트 요청을 처리하고 데이터를 전달.
비동기 이벤트 기반 구조를 사용해 빠르고
효율적인 데이터 처리 구현.



HTML5, CSS3, JavaScript

구조 설계 및 컴포넌트 구성.
반응형 디자인을 위한 미디어쿼리 사용.
앱 내의 다양한 기술 및 이벤트 구현.



MariaDB (SQL)

음악 정보 데이터를 체계적으로 관리.
동적 재생 목록 및 즐겨찾기 기능 구현.



01 프로젝트 개요

SCHEDULE

개발 일정



기획 및 디자인

프로젝트 기획 및 기능 정리
기본 스케치 디자인 및
lay-out 설계.



프론트엔드 구현

HTML 구조 설계, CSS 스타일링.
UI 요소 시각적으로 확인.
JAVASCRIPT
뮤직 플레이어 기능 구현.



백엔드 구현

NODE Express로 기본 서버 구현.
MariaDB와의 연결 설정.
서버와 클라이언트를 연결하고,
기능을 통합 테스트.



오류수정 및 보완

즐겨찾기 등 기능 추가.
UI 및 UX 최종 조정.
모든 기능 테스트 및 버그 수정.



02 설계 및 기획



02 설계 및 기획

FULL DESIGN

전체 디자인

사용자 중심 레이아웃

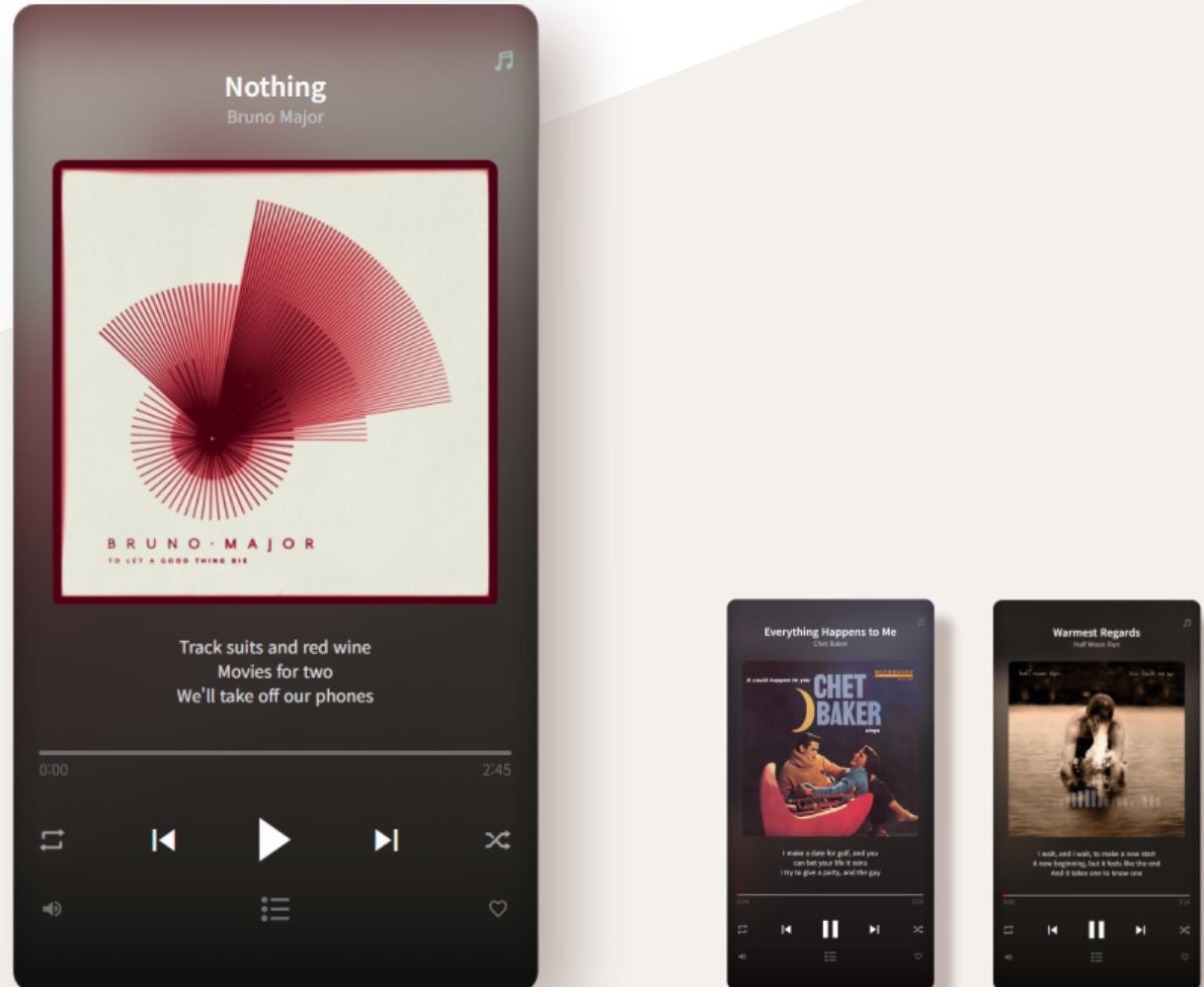
기능별로 섹션(곡 정보, 컨트롤 버튼, 재생 목록 등)을 나누어 명확히 구분된 구조를 통해 필요한 정보를 쉽게 찾을 수 있도록 설계.

미니멀리즘

불필요한 요소를 배제하고, 핵심 기능과 정보만 표시하여 직관적인 사용자 경험을 제공하며, UI 요소 간 적절한 여백을 두어 깔끔하고 정돈된 화면을 설계.

시각적 요소 강화

곡에 따라 배경 이미지와 컬러가 동적으로 변경되어 음악의 분위기를 시각적으로 표현. 이퀄라이저 등의 몰입감을 줄 수 있는 효과 및 버튼 클릭 시 크기 확대와 색상 변화 같은 애니메이션을 통해 피드백을 강화.





02 설계 및 기획

DETAILS

세부 디자인 및 기능

오디오의 주파수 데이터를 기반으로
실시간으로 반응하는
이퀄라이저 애니메이션 구현.
재생 중인 음악에 따라
동적인 시각적 피드백 제공.

재생바 컨트롤 및

재생시간 표시

사용자가 재생 위치를 시각적으로 확인하고,
직접 이동 가능하도록 설계.

클릭 시, 아이콘 변환에 따라 기능변경



전체 1회 반복/ 전체 반복/ 한곡 반복

재생 중인 곡의 앨범 자켓에 맞춰 동적으로 배경이미지 변환



곡명/ 아티스트명

앨범 자켓

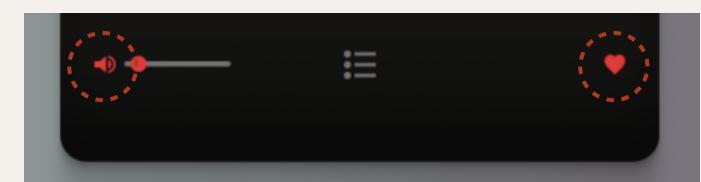
가사를 스크롤 형태로 편리하게 제공
(모바일 드래그)

셔플 재생 기능

클릭 시, 아이콘 변환



볼륨설정기능/ 즐겨찾기 기능



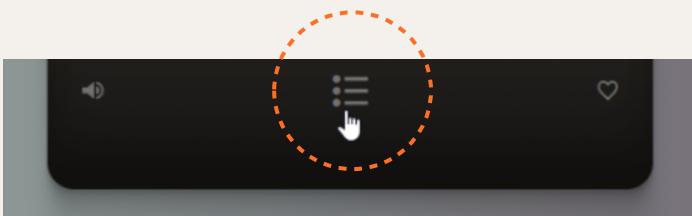
클릭 시, 볼륨 조절 슬라이더 생성 및
아이콘 변화에 따라 즐겨찾기 자동 추가

02 설계 및 기획



DETAILS

세부 디자인 및 기능



리스트 버튼 클릭 시,
화면 하단에서 슬라이드 형태로 올라오며 리스트 생성

The image displays two screenshots of a mobile music application's user interface. Both screenshots show a 'Music List' overlay that slides up from the bottom of the screen. In the left screenshot, the title 'June (Instrumental)' by Jung Joonil is visible above the list. The list includes songs like 'Nothing' by Bruno Major, 'June (Instrumental)' by Jung Joonil (which is currently playing), 'Paris' by New West, 'Warmest Regards' by Half Moon Run, 'free love (dream edit)' by HONNE, 'Tomorrow' by Car, the garden, 'Sweet' by Cigarettes After Sex, 'Sinatra (piano ver.)' by Forrest Nolan, and 'We Were Never Really Friends' by Bruno Major. In the right screenshot, the title 'Paris' by New West is visible above the list. The list includes songs like 'We Were Never Really Friends' by Bruno Major, 'Everything Happens to Me' by Chet Baker, 'June (Instrumental)' by Jung Joonil (which is currently playing), and 'Paris' by New West.

상단 타이틀 부분을 고정하여,
하단 리스트 스크롤 시에도 현재
상태를 직관적으로 알 수 있도록
함.

즐겨찾기 토글: 전체 곡 목록과
즐겨찾기 목록을 손쉽게 전환할
수 있는 UI 제공.

'playing' 상태 표시: 현재 재생 중
인 곡을 강조 표시하여 가독성을
높임.



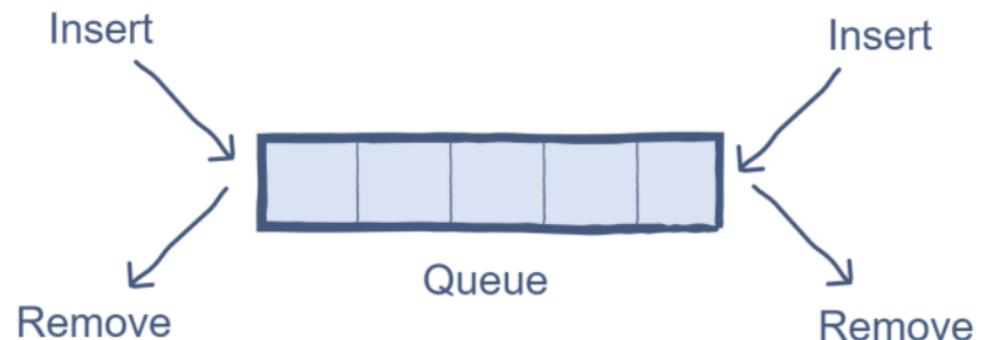
02 설계 및 기획

DATA STRUCTURES AND DATABASE DESIGN

자료 구조 및 데이터 설계

자료 구조

DEQUE 자료 구조를 활용해 재생 목록을 효율적으로 관리.



데이터베이스 테이블

뮤직 리스트를 적절하게 관리하고, 확장성과 유지보수성을 향상시킬 수 있도록 설계.

music_player								뮤직 플레이어 음악 리스트	
🔑	아이디	ID	song	int	NOT NULL	Default value	Auto Increment		
곡명	name	Domain	char(50)	NOT NULL	Default value	Comment			
아티스트명	artist	Domain	char(50)	NOT NULL	Default value	Comment			
앨범 자켓	jacket	Domain	varchar(255)	NOT NULL	Default value	앨범자켓 url 저장			
배경 이미지	background	Domain	char(50)	NOT NULL	Default value	배경 이미지 url 저장			
번호	song	unique key	char(20)	NOT NULL	song+id	순서 저장			
가사	lyrics	Domain	text	NULL	Default value	Comment			



03 프론트엔드



03 프론트엔드

재생 및 정지 기능

재생 중이 아닐 경우, AudioContext를 생성하고 오디오 소스와 이퀄라이저를 연결. playClick() 메서드를 호출해 재생 상태를 활성화.

재생 중일 경우, pauseClick() 메서드를 호출해 재생을 일시 중지.



```
1  playPause() {
2      this.playBtnBox.addEventListener('click', () => {
3          if (!this.audioContext) {
4              this.audioContext = new (window.AudioContext || window.webkitAudioContext)();
5
6              this.musicSource = this.audioContext.createMediaElementSource(this.audio);
7              this.analyzer = this.audioContext.createAnalyser();
8
9              this.musicSource.connect(this.analyzer);
10             this.analyzer.connect(this.audioContext.destination);
11
12             this.musicEqualizer();
13         }
14
15         if (this.audio.paused) {
16             if (this.audioContext.state === 'suspended') {
17                 this.audioContext.resume();
18             }
19             this.playClick();
20         } else {
21             this.pauseClick();
22         }
23     });
24 }
```



03 프론트엔드

곡 이동 기능(이전곡, 다음곡)



이전 곡(prevMusic):

재생 목록에서 마지막 곡을 꺼내 맨 앞으로 이동.

loadSong() 메서드로 해당 곡을 로드하고 재생.

다음 곡(nextMusic):

재생 목록에서 첫 번째 곡을 꺼내 맨 뒤로 이동.

loadSong() 메서드로 해당 곡을 로드하고 재생.

```
1 prevMusic() {
2     const lastItem = this.songList.backTake();
3     const firstItem = this.songList.data[0];
4     const figureString_jacket =
5         `<figure>
6             `;
8     this.songName.innerHTML = `${lastItem['name']} `;
9     this.artistName.innerHTML = `${lastItem['artist']} `;
10    this.jacketBox.innerHTML = figureString_jacket;
11    document.querySelector('.lyricsBox').innerHTML = `${lastItem['lyrics']} `;
12    this.songList.frontInsert(lastItem);
13    document.querySelector('.playerBox').style.backgroundImage = lastItem.background;
14    this.audio.src = `./media/${lastItem['song']}.mp3`;
15    this.audio.load();
16    this.audio.play();
17    if (!this.audioContext) {
18        this.audioContext = new (window.AudioContext || window.webkitAudioContext)();
19        this.musicSource = this.audioContext.createMediaElementSource(this.audio);
20        this.analyzer = this.audioContext.createAnalyser();
21        this.musicSource.connect(this.analyzer);
22        this.analyzer.connect(this.audioContext.destination);
23    }
24    if (this.audioContext && this.analyzer) {
25        this.musicEqualizer();
26    }
27    this.musicIndex = this.songList.data.findIndex(song => song.song === lastItem.song);
28    this.updateStarButton();
29    this.playClick();
30 }
31 nextMusic() {
32     .....생략
33 }
```



03 프론트엔드

셔플 및 반복 재생 기능



makeRandom() 메서드는

셔플 기능으로, 곡 목록을 랜덤으로 섞고 첫 번째 곡을 재생함.

frontTake()와 backInsert()를 활용해 재생 목록을 동적으로 변경.



musicRepeatBtn() 메서드는

반복 기능으로, 버튼을 클릭할 때마다

반복 없음 → 전체 반복 → 단일 곡 반복
으로 상태를 전환.



```
1 makeRandom() {
2     this.songLength = this.songList.getCount();
3     let rotateCount = Math.floor(Math.random() * (this.songLength - 1) + 1);
4
5     for (let i = 0; i < rotateCount; i++) {
6         let tempV = this.songList.frontTake();
7         this.songList.backInsert(tempV);
8     }
9
10    const shuffleItem = this.songList.data[0];
11    this.loadSong(0);
12    this.audio.play();
13}
14 musicRepeatBtn() {
15     this.repeatBtn.addEventListener('click', () => {
16         const currentClass = this.repeatIcon.className;
17         if (currentClass.includes('xi-repeat-one')) {
18             this.repeatIcon.className = 'xi-repeat';
19         } else if (currentClass.includes('xi-repeat click_color')) {
20             this.repeatIcon.className = 'xi-repeat-one click_color';
21         } else {
22             this.repeatIcon.className = 'xi-repeat click_color';
23         }
24     });
25 }
```



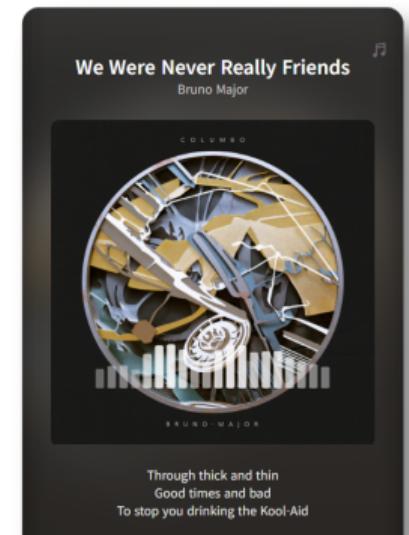
03 프론트엔드

동적 UI 업데이트

loadSong() 메서드는 곡을 로드할 때 제목, 아티스트, 앨범 커버, 배경 이미지 등을 동적으로 업데이트.
오디오 파일을 로드하고 재생을 시작함.



```
1  loadSong(index, isFav = false) {
2      const song = isFav ? JSON.parse(localStorage.getItem('favorites'))[index] : this.originalSongList[index];
3
4      this.songList.data[0] = song;
5      this.songName.innerText = song.name;
6      this.artistName.innerText = song.artist;
7      document.querySelector('.lyricsBox').innerHTML = song.lyrics;
8      this.jacketBox.innerHTML = `<figure></figure>`;
9      this.audio.src = `./media/${song.song}.mp3`;
10     document.querySelector('.playerBox').style.backgroundImage = song.background;
11     this.audio.play();
12
13     if (this.audioContext && this.analyzer) {
14         this.musicEqualizer();
15     }
16
17     this.playClick();
18     this.updateStarButton();
19     this.musicList.classList.remove('show');
20 }
```





03 프론트엔드

이퀄라이저 시각화



실시간 애니메이션.
getByteFrequencyData()를
사용해 주파수 데이터를 가져옴.
막대의 크기와 색상을
주파수 값에 따라 동적으로 변경.

numberOfBars만큼
막대를 생성하고
앨범 자켓 내부에 추가.



```
1  musicEqualizer() {  
2      const equalizerArea = document.getElementById("equalizer");  
3      const numberOfBars = 20;  
4      const frequencyData = new Uint8Array(this.analyzer.frequencyBinCount);  
5  
6      // 막대 생성  
7      if (!equalizerArea.hasChildNodes()) {  
8          for (let i = 0; i < numberOfBars; i++) {  
9              const bar = document.createElement("div");  
10             bar.classList.add("equalizerBar");  
11             equalizerArea.appendChild(bar);  
12         }  
13     }  
14  
15     // 애니메이션 업데이트  
16     const updateBars = () => {  
17         this.analyzer.getByteFrequencyData(frequencyData);  
18         Array.from(equalizerArea.children).forEach((bar, i) => {  
19             const intensity = frequencyData[i] / 255;  
20             bar.style.transform = `scaleY(${intensity})`;  
21             bar.style.backgroundColor = `rgba(255, 255, 255, ${intensity})`;  
22         });  
23         requestAnimationFrame(updateBars);  
24     };  
25     updateBars();  
26 }
```



03 프론트엔드

재생 바 기능



```
1 barMaking() {
2     this.audio.addEventListener("timeupdate", (e) => {
3         const currentTime = e.target.currentTime;
4         const duration = e.target.duration;
5         let barWidth = (currentTime / duration) * 100;
6         this.rangeBar.style.width = `${barWidth}%`;
7         let currentMin = Math.floor(currentTime / 60);
8         let currentSec = Math.floor(currentTime % 60);
9         if (currentSec < 10) { currentSec = `0${currentSec}` };
10        this.current.innerHTML = `${currentMin}:${currentSec}`;
11    });
12    this.audio.addEventListener("loadedmetadata", () => {
13        let audioDuration = this.audio.duration;
14        let totalMin = Math.floor(audioDuration / 60);
15        let totalSec = Math.floor(audioDuration % 60);
16        if (totalSec < 10) { totalSec = `0${totalSec}` };
17        this.duration.innerHTML = `${totalMin}:${totalSec}`;
18    });
19    this.musicBar.addEventListener("click", e => {
20        let progressWidth = this.musicBar.clientWidth;
21        let clickedOffsetX = e.offsetX;
22        let songDuration = this.audio.duration;
23        this.audio.currentTime = (clickedOffsetX / progressWidth) * songDuration;
24        this.audio.play();
25        this.playClick();
26    })
27 }
```

timeupdate 이벤트를 사용해 현재 재생 시간과 재생 진행률을 실시간으로 계산.

loadedmetadata 이벤트를 통해 곡의 총 길이를 가져와 표시.

재생 진행률을 계산하여 rangeBar의 width 스타일로 반영. currentTime / duration 비율로 막대의 너비를 결정.

재생 바를 클릭하면 해당 위치의 시간을 계산하여 곡의 재생 위치(currentTime)를 업데이트. 클릭 후 자동으로 곡이 재생되도록 설정.



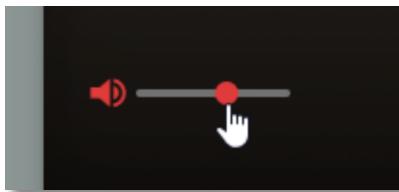


03 프론트엔드

볼륨 조절

볼륨 버튼을 클릭하면 슬라이더가 나타나거나 사라짐.

슬라이더의 값을 변경할 때 audio.volume 속성을 업데이트하여 볼륨을 조절.



```
1  initVolumeControl() {
2      this.volumeBtn.addEventListener('click', () => {
3          const currentClass = document.getElementById('volumeIcon').className;
4          if (currentClass.includes('xi-volume-up click_color')) {
5              document.getElementById('volumeIcon').className = 'xi-volume-up';
6              this.volumeSlider.style.visibility = 'hidden';
7          } else {
8              document.getElementById('volumeIcon').className = 'xi-volume-up click_color';
9              this.volumeSlider.style.visibility = 'visible';
10         }
11     });
12
13     this.volumeSlider.addEventListener('input', (event) => {
14         this.audio.volume = event.target.value;
15     });
16 }
```



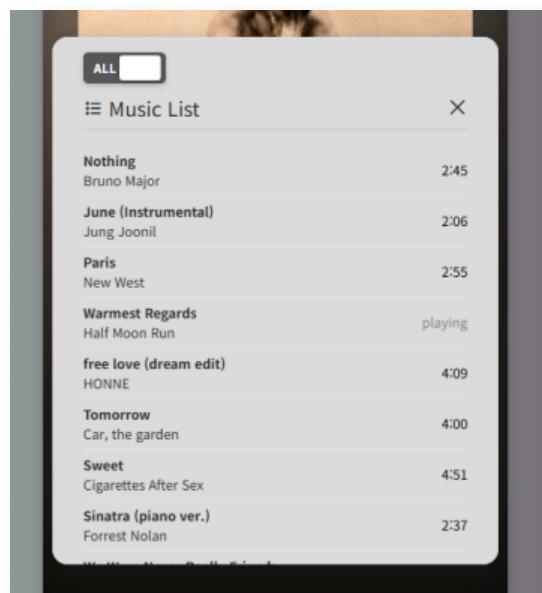
03 프론트엔드

재생 목록 렌더링

musicListSet() 메서드는

재생 목록을 UI에 동적으로 렌더링.

각 곡에 대해 태그를 생성하고,
클릭 이벤트를 추가하여 곡을 선택하면
해당 곡이 로드되고 재생됨.



```

1  musicListSet() {
2      this.musicListUl.innerHTML = '';
3      this.originalSongList.forEach((song, index) => {
4          const li = document.createElement('li');
5          li.dataset.index = index;
6          li.innerHTML = `
7              ...생략
8          `;
9          this.musicListUl.appendChild(li);
10
11         const liAudio = li.querySelector(`.${song.song}`);
12         const liAudioDuration = li.querySelector(`#${song.song}`);
13         liAudio.addEventListener('loadeddata', () => {
14             if (liAudioDuration.classList.contains('playing')) return;
15             const audioDuration = liAudio.duration;
16             const totalMin = Math.floor(audioDuration / 60);
17             const totalSec = Math.floor(audioDuration % 60);
18             liAudioDuration.innerHTML = `${totalMin}:${totalSec < 10 ? '0' : ''}${totalSec}`;
19             liAudioDuration.setAttribute('data-duration', `${totalMin}:${totalSec}`);
20         });
21         const currentSong = this.audio.src.split('/').pop().replace('.mp3', '');
22         if (this.audio.src && currentSong === song.song) {
23             li.classList.add('playing');
24             li.querySelector('.audio-duration').innerHTML = 'playing';
25             li.querySelector('.audio-duration').classList.add('playing');
26         } else {
27             li.classList.remove('playing');
28         }
29         li.addEventListener('click', () => {
30             this.loadSong(index);
31         });
32     });
33 }
```



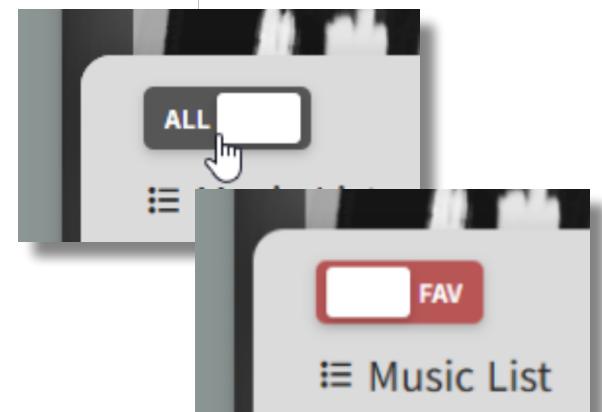
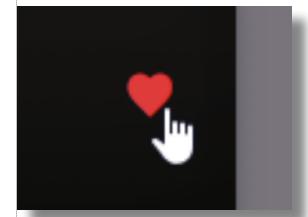
03 프론트엔드

즐겨찾기 기능 즐겨찾기 버튼: localStorage를 사용해 즐겨찾기 상태를 저장/삭제. 현재 재생 중인 곡이 즐겨찾기인지 확인하고, 상태에 따라 아이콘 변경.
즐겨찾기 토글: 전체 목록(All)과 즐겨찾기 목록(Favorites)을 전환.



```

1 musicStarBtn() {
2     this.starBtn.addEventListener('click', () => {
3         const currentSong = this.songList.data[0];
4         let favorites = JSON.parse(localStorage.getItem('favorites')) || [];
5
6         const isFavorite = favorites.some(song => song.song === currentSong.song);
7         if (isFavorite) {
8             favorites = favorites.filter(song => song.song !== currentSong.song);
9             this.starIcon.className = 'xi-heart-o'; // 비활성화 아이콘
10        } else {
11            favorites.push(currentSong);
12            this.starIcon.className = 'xi-heart click_color'; // 활성화 아이콘
13        }
14        localStorage.setItem('favorites', JSON.stringify(favorites));
15    });
16 }
17 toggleFavorites() {
18     const toggleButton = document.getElementById('toggleFavoritesSlider');
19     toggleButton.addEventListener('click', () => {
20         if (toggleButton.checked) {
21             this.favoritesListSet(); // 즐겨찾기 목록 표시
22         } else {
23             this.musicListSet(); // 전체 목록 표시
24         }
25     });
26 }
```





03 프론트엔드

API 데이터 연동

API에서 데이터를 가져와 재생 목록을 초기화.
백엔드와의 연동을 통해 동적 데이터 로드를 처리.



```
1 const fetchData = async (playerInstance) => {
2     try {
3         const response = await fetch('http://kkms4001.iptime.org:45212/api/data');
4         console.log("Response Status:", response.status);
5         console.log("Response Status Text:", response.statusText);
6
7         if (!response.ok) throw new Error('데이터 가져오기 실패');
8
9         const data = await response.json();
10        console.log(data);
11        playerInstance.running(data);
12    } catch (error) {
13        console.error('데이터 가져오기 오류:', error);
14    }
15 };
16
17 window.onload = () => {
18     const myAlbum = new MusicPlayer('myAlbum');
19     fetchData(myAlbum);
20};
```



04 백엔드



04 백엔드

서버 설정 및 실행/ MARIADB 연동



Express를 사용해 기본 서버를 구성.

CORS를 활성화하여 클라이언트와

서버 간 통신 문제 해결.

public 폴더의 정적 파일을 제공.

mysql 라이브러리를 사용해

MariaDB와 연결 설정.

host, user, password, database 정보를

포함하여 데이터베이스 연결 초기화.

연결 성공/실패 시 콘솔에 상태를 출력.

```
1 const express = require('express');
2 const app = express();
3 const cors = require('cors');
4 const port = 45212;
5 app.use(cors());
6 app.use(express.static('public'));
7
8 app.listen(port, () => {
9   console.log(`Server running at http://localhost:${port}`);
10 });
11
12 const mysql = require('mysql2');
13 const connection = mysql.createConnection({
14   host: 'localhost',
15   user: '****',
16   password: '****',
17   database: '****'
18 });
19 connection.connect((err) => {
20   if (err) {
21     console.error('데이터베이스 연결 실패:', err.stack);
22     return;
23   }
24   console.log('MariaDB에 연결되었습니다.');
25 });
```



04 백엔드

API 엔드포인트 구현



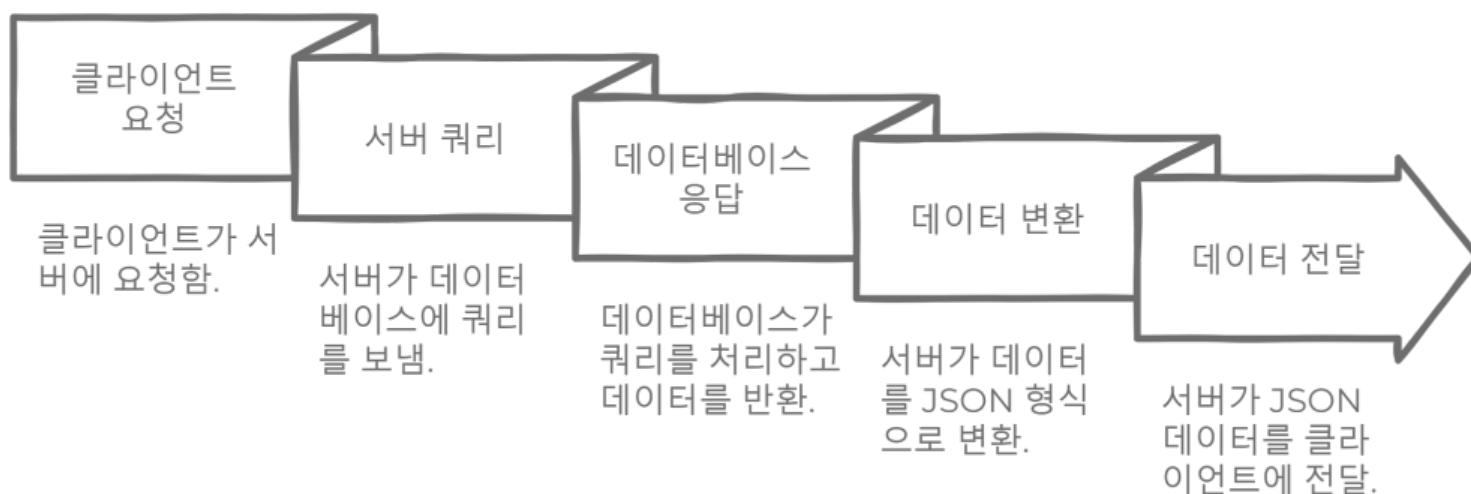
```
1 app.get('/api/data', (req, res) => {
2   const query = 'SELECT * FROM `music player`';
3   connection.query(query, (err, results) => {
4     if (err) {
5       console.error('데이터 조회 실패:', err);
6       res.status(500).send('데이터 조회 실패');
7       return;
8     }
9     console.log(results)
10    res.json(results);
11  });
12});
```

/api/data 엔드포인트를 통해 곡 정보를 제공.

SELECT * FROM `music player` 쿼리를 실행하여 데이터베이스에서 데이터를 조회.

쿼리 결과를 JSON 형식으로 클라이언트에 반환.

전체 프로세스





05 후기



REVIEW

“아쉬운 부분이 더 많았지만, 많은 경험이 가능했던 프로젝트.”

이번 프로젝트는 백엔드의 기초를 배우던 무렵 시작한 프로젝트였기 때문에
백엔드와 프론트엔드의 전체적인 흐름을 이해하고 구현하는 데 가장 초점을 맞추고 시작했습니다.

HTML, CSS, JavaScript, Node.js, Express, MariaDB를 사용하여, 기능적인 작동을 포함해
UX/UI, 애니메이션 등 시각적 부분까지 고려한 결과물을 만들기 위해 노력했습니다.

프로젝트의 초기 기획부터 디자인, 코드 구현, API 설계, 데이터베이스 연동까지 전 과정을
혼자서 주도적으로 진행하며 개발 프로세스를 체계적으로 경험할 수 있었습니다.

프론트엔드에서는 가장 먼저 사용자 입장을 생각하며 설계했고,
뮤직 플레이어 안의 다양한 기능을 구현하며 개인적으로 많은 공부가 되었습니다.

백엔드에서는 아주 기초적인 코딩이었지만 통신 구조의 흐름을
직접 느껴볼 수 있었다는 점이 값진 경험이었습니다.

하지만 마무리가 된 지금, 스스로 피드백을 해보았을 때,
메서드를 조금 더 명확하고 간결하게 하지 못했던 부분이 가장 큰 아쉬움으로 남습니다.
일부 코드가 중복되고 있다는 것을 느꼈음에도 체계화 시키지 못해서 정리하지 못했고,
기능을 계속 얹으며 추가하는 과정에서는, 기존 메서드들을 명확한 목적으로 설계하지 못했기 때문에,
새로운 메서드들과 계속 부딪히며 오류가 생겼고, 기존 코드를 활용할 수 없었습니다.

이번 프로젝트로 학습한 내용과 아쉬웠던 부분들을 스스로 더욱 피드백해서,
다음 프로젝트에서는 조금 더 효율적이고 기능적으로 훌륭한 로직을 설계할 수 있도록 노력하려 합니다.



감사합니다.

Thank you

