

목차

- 1. 프로젝트 설명**
 - a. 프로젝트 사용기술
 - b. 프로젝트 설명 및 개발 계기
 - c. 프로젝트 개발 과정
- 2. 프로그램 기능**
 - a. 프로그램 기능 설명 및 구현코드
- 3. 프로젝트 느낀점**

1.a 프로젝트 사용기술

- 언어 : JS
- 기술 : Node.js(express)
- 개발툴 : visual studio code
- 프로젝트 관리 : git

1-b. 프로젝트 설명 및 개발 계기

설명 및 개발 계기 : 원래는 알송으로 음악을 듣는 걸 좋아했습니다. 그런데 특정 음악파일이 재생이 안되었고, 해당 오류를 알송이 업데이트를 안해줄 것 같아 제가 직접 만들기로 결정하였습니다.

원래는 **c++**사용해서 알송과 비슷한 음악 플레이어 프로그램을 만들고자 했습니다. 하지만 이렇게 **exe**같은 프로그램을 만들기 보다는 보다 이식성이 높은 여러 곳에서 사용할 수 있도록 만들면 좋다 생각이 들었습니다. 그 중 웹이 떠올랐고 웹을 통해 개발을하기로 결정하였습니다.

그래서 웹의 기초인 자바스크립트, **HTML**, **CSS**도 배울 겸해서 만들어 보았습니다.

1-c. 프로젝트 개발 과정

과정 1 : 우선 아래의 알송 디자인을 참고했습니다.



그런 다음 아래와 같은 디자인을 생각하고 디자인을 구현하였습니다.



과정 2 : 디자인을 입혔으니 각각의 기능에 **JS**코드를 삽입 하였습니다. 다양한 **JS**다루는 기술들은 구글링이나 유튜브를 통해 공부를 하였습니다.

과정 3 : 저는 처음에는 단순히 **JS**로만 프로젝트를 만들 줄 알았는데 그게 아니었습니다. 이러한 프로젝트를 구현하기 위해서는 서버를 만들어야 하고 해당 서버에 **API**를 요청을 해야 한다는 것이었습니다. 그래서 같은 언어를 사용하고 있는 [Node.js\(Express\)](#)를 유튜브를 보면서 공부를 하였습니다.

과정 4 : 그다음 서버를 구축한뒤, 서버에서 받은 데이터를 이용해 **AI**가사를 브라우저에 표시하도록 만들었습니다.

2-a. 프로그램 기능 설명 및 구현코드

설명 : 중요한 기능들을 나열하면 다음과 같습니다.

한곡 추가 : 다음 일련의 태스크 과정을 진행하게 됩니다.

```
function clickAddFileImageElement() {
  fileInputElement.click();
}

addFileImageElement.addEventListener("click", clickAddFileImageElement);

fileInputElement.addEventListener('change', async (event) => {
  try {
    if(event.target.files.length > 0) {
      const task1 = await disableAddFileClick();
      const task2 = await displayFileLoading();
      const task3 = await addToAudioFiles(event.target.files);
      const task4 = await createDivAndInsert(event.target.files);
      const task5 = await clearFileLoading();
      const task6 = await enableAddFileClick();
    }
  }
  catch(err) {
    console.error(err);
    //if you need the display of loading failed, insert to this.
  }
});
```

- **task 1** : 해당 작업이 끝날 때까지 클릭버튼을 비활성화 합니다.
작업이 끝나기 전에 해당작업을 또 진행하는 걸 방지하기
위함입니다.

```
function disableAddFileClick() {
  return new Promise((resolve, reject) => {
    addFileImageElement.removeEventListener("click", clickAddFileImageElement);
    //make the file image have dark color or change opacity
    resolve();
  });
}
```

- **task 2** : 파일 로딩중을 표시합니다.

```
function displayFileLoading() {
  return new Promise((resolve, reject) => {
    const loadingElement = document.createElement("div");
    loadingElement.classList.add("file-loading");
    loadingElement.textContent = "Loading...";
    rightBottomItemFrameElement.appendChild(loadingElement);
    resolve();
  });
}
```

- **task 3** : 업로드된 오디오 **Blob** 파일을 오디오 파일 리스트 배열에 저장합니다. 오디오 파일이 아닌 경우에는 파일이 추가되지 않습니다.

```
function addToAudioFiles(selectedFiles) {  
    return new Promise((resolve, reject) => {  
        for(const selectedFile of selectedFiles) {  
            if(selectedFile.type.match("audio") != null) {  
                audioFiles.push(selectedFile);  
            }  
        }  
        resolve();  
    });  
}
```

- **task 4** : 플레이할 오디오 리스트에 해당 음원파일들을 추가합니다. 플레이할 오디오 리스트에 **HTML** 요소를 넣는 작업들을 하며, 다른 곡을 클릭 하였을때, 가사감지기(LyricsWatcher)를 비활성화 합니다. 가사감자기를 비활성화 하는 이유는 현재 플레이중인 곡이 가사 표시중이고 이제 새로운 곡을 클릭하면 현재 표시되는 가사는 새로운 곡에 맞지 않기에 비활성화를 하였습니니다.

```

function createDivAndInsert(selectedFiles) {
  return new Promise((resolve, reject) => {
    for(const selectedFile of selectedFiles) {
      if(selectedFile.type.match("audio") != null) {
        const bottomPaddingElement = document.querySelector(".audio-file-frame-padding");
        const audioFrameElement = document.createElement("div");

        audioFrameElement.classList.add("audio-file-frame");
        audioFrameElement.textContent = `${selectedFile.name}`;
        audioFrameElements.push(audioFrameElement);

        audioFrameElement.addEventListener("dblclick", async () => {
          try {
            checkLyricsWatcher();
            updateNowPlayingAudioFile(selectedFile);
            updateIsPlaying(true);
            audioFrameElement.style.fontWeight = "bold";

            if(prevPlayedAudioFile != null) {
              const prevAudioFrameElement = document.querySelectorAll(".audio-file-frame")
                [audioFiles.indexOf(prevPlayedAudioFile)];

              prevAudioFrameElement.style.fontWeight = "normal";
            }
          }
        });
      }
    }
  });
}

```

```

updateCurAudioFileURL(selectedFile);
const pauseButtonSrc = "/View/Images/pause-song-button.svg";
playPauseSongButtonImageElement.src = pauseButtonSrc;
audioPlayerElement.src = curAudioFileURL;
const task1 = await loadMetaData();
audioPlayerElement.play();

function loadMetaData() {
  return new Promise((resolve, reject) => {
    let load;
    audioPlayerElement.addEventListener('loadedmetadata', load = () => {
      //plan : disable slider thumb

      audioPlayerElement.removeEventListener("loadedmetadata", load);
    });
    resolve();
  });
}

function checkLyricsWatcher() {
  if(lyricsWatcher !== null) {
    clearInterval(lyricsWatcher);
    setLyricsWatcher(null);
  }
}

```

- **task 5** : 파일 로딩중이라는 표시를 제거합니다.

```

function clearFileLoading() {
  return new Promise((resolve, reject) => {
    const loadingElement = document.querySelector(".file-loading");
    loadingElement.remove();
    resolve();
  });
}

```

- **task 6** : 비활성화된 클릭버튼을 활성화 합니다.

```
function enableAddFileClick() {
  return new Promise((resolve, reject) => {
    addFileImageElement.addEventListener("click", clickAddFileImageElement);
    resolve();
  });
}
```

폴더업로드(여러곡) : 기본적으로 한곡추가와 비슷한 프로세스를 진행합니다. 보여주는 화면 태스크만 다르고 나머지는 같습니다.

```
folderInputElement.addEventListener('change', async (event) => {
  try {
    if(event.target.files.length > 0) {
      const task1 = await disableLoadFolderClick();
      const task2 = await displayFolderLoading();
      const task3 = await addToAudioFiles(event.target.files);
      const task4 = await createDivAndInsert(event.target.files);
      const task5 = await clearFolderLoading();
      const task6 = await enableLoadFolderClick();
    }
  }
  catch(error) {
    console.log(error);
  }
});
```

현재 재생곡 :

- **곡 루핑** : 음악 플레이어의 루핑을 설정합니다. 보기와 같이 나머지 연산을 사용하였습니다.

```
let clickRepeatButton
repeatButtonImageElement.addEventListener("click", clickRepeatButton = () => {
  repeatButtonImageElement.removeEventListener("click", clickRepeatButton);
  repeatButtonImageElement.click();

  let newLoopState = (curLoopState + 1) % 3;

  if(curLoopState === totalSongLoop) {
    repeatButtonImageElement.src = "/View/Images/repeat-one-button.svg";
    updateCurLoopState(newLoopState); //1
  }
  else if(curLoopState === oneSongLoop) {
    repeatButtonImageElement.src = "/View/Images/no-repeat-button.svg";
    updateCurLoopState(newLoopState); //2
  }
  else { //noLoop
    repeatButtonImageElement.src = "/View/Images/repeat-total-button.svg";
    updateCurLoopState(newLoopState); //0
  }

  repeatButtonImageElement.addEventListener("click", clickRepeatButton);
});
```


- **곡 셔플** : 음악 플레이어의 곡 셔플을 설정합니다.
셔플링(랜덤재생) 하거나 셔플링 하지 않는 경우(순차재생)입니다.

```
let clickShuffleButton;
shuffleButtonImageElement.addEventListener("click", clickShuffleButton = () => {
  shuffleButtonImageElement.removeEventListener("click", clickShuffleButton);
  shuffleButtonImageElement.click();

  if(isShuffling === false) {
    shuffleButtonImageElement.src = "/View/Images/shuffle-on-button.svg";
    updateIsShuffling(true);
  }
  else {
    shuffleButtonImageElement.src = "/View/Images/shuffle-off-button.svg";
    updateIsShuffling(false);
  }

  shuffleButtonImageElement.addEventListener("click", clickShuffleButton);
});
```

- **볼륨 조절** : 음악 플레이어의 볼륨을 조절합니다. 마우스 입력, 드래그와 같은 이벤트를 넣었습니다.

```
let clickVolumeButton;
volumeButtonImageElement.addEventListener("click", clickVolumeButton = () => {
  volumeButtonImageElement.removeEventListener("click", clickVolumeButton);
  volumeButtonImageElement.click();

  //change volume button
  volumeButtonImageElement.addEventListener("click", clickVolumeButton);
});

volumeProgressBarInputElement.addEventListener("click", () => {
  changeVolume();
});

volumeProgressBarInputElement.addEventListener("mousedown", () => {
  let mouseMoveListener;
  let mouseUpListener;

  window.addEventListener("mousemove", mouseMoveListener = () => {
    changeVolume();
  });
  window.addEventListener("mouseup", mouseUpListener = () => {
    stopChangeVolume(mouseMoveListener, mouseUpListener);
  });
});
```

- **이전 및 다음 곡 재생, 곡 재생 멈춤** : 해당 기능을 구현할 시 음악플레이어의 셔플이나, 곡 루핑, 그리고 가사감지기등을 체크하여 프로세스를 진행합니다.

```

nextSongButtonImageElement.addEventListener("click", clickNextSongButton = async () => {
  try {
    nextSongButtonImageElement.removeEventListener("click", clickNextSongButton);
    checkLyricsWatcher();
    const curAudioIndex = audioFiles.indexOf(nowPlayingAudioFile);
    let newAudioIndex = curAudioIndex + 1;

    if(newAudioIndex >= audioFiles.length) {
      newAudioIndex = 0;
      if(isPlaying === true) {
        if(curLoopState === noLoop) {
          newAudioIndex = curAudioIndex;
          await updateWhenPlaying(curAudioIndex, newAudioIndex);
        }
        else if(curLoopState === oneSongLoop) {
          newAudioIndex = curAudioIndex;
          await updateWhenPlaying(curAudioIndex, newAudioIndex);
        }
      }
    }
  }
});

```

```

let clickStopSongButton;
stopSongButtonImageElement.addEventListener("click", clickStopSongButton = () => {
  stopSongButtonImageElement.removeEventListener("click", clickStopSongButton);
  if(audioPlayerElement.duration) {
    const playButtonSrc = "/View/Images/play-song-button.svg";
    const durationMin = parseInt(audioPlayerElement.duration / 60);
    const durationSec = parseInt(audioPlayerElement.duration % 60);

    audioPlayerElement.pause();

    playPauseSongButtonImageElement.src = playButtonSrc;
    audioPlayerElement.currentTime = 0;
    audioProgressBarInputElement.value = 0;
    audioPlayingTimeFrameElement.textContent =
      `${0}:${0} / ${durationMin}:${durationSec}`;
    updateIsPlaying(false);
  }
  stopSongButtonImageElement.addEventListener("click", clickStopSongButton);
});

```

- **플레이 중인 곡 상태 표시** : **setInterval**을 통한 비동기 프로세스를 통해 현재 플레이 중인 곡의 상태등을 추적합니다. 해당 플레이 중인 곡의 플레이 시간이나 프로그레스 바를 통한 시간 추적등을 수행합니다.

```

audioPlayerElement.addEventListener("ended", startTaskAfterAudioEnded);

let displayAudioProcessThread = setInterval(() => {
  if((audioPlayerElement.currentTime / audioPlayerElement.duration) * 100) {
    const durationMin = parseInt(audioPlayerElement.duration / 60);
    const durationSec = parseInt(audioPlayerElement.duration % 60);
    const currentMin = parseInt(audioPlayerElement.currentTime / 60);
    const currentSec = parseInt(audioPlayerElement.currentTime % 60);
    audioPlayingTimeFrameElement.textContent =
      `${currentMin}:${currentSec} / ${durationMin}:${durationSec}`;

    audioProgressBarInputElement.value =
      (audioPlayerElement.currentTime / audioPlayerElement.duration) * 100;
  }
},10);

audioProgressBarInputElement.addEventListener("mousedown", () => {
  if(audioPlayerElement.duration) {
    let mouseMoveListener;
    let mouseUpListener;
    clearInterval(displayAudioProcessThread);

    window.addEventListener("mousemove", mouseMoveListener = () => {
      setAudioPlayingTime();
    });
  }
});

```

```

function setAudioPlayingTime() {
  const currentTime = (audioProgressBarInputElement.value * audioPlayerElement.duration) / 100;

  if(currentTime !== NaN) {
    const durationMin = parseInt(audioPlayerElement.duration / 60);
    const durationSec = parseInt(audioPlayerElement.duration % 60);
    const currentMin = parseInt(currentTime / 60);
    const currentSec = parseInt(currentTime % 60);
    audioPlayingTimeFrameElement.textContent =
      `${currentMin}:${currentSec} / ${durationMin}:${durationSec}`;
  }
}

function setAudioProcess(mouseMoveListener, mouseUpListener) {
  audioPlayerElement.currentTime =
    (audioProgressBarInputElement.value * audioPlayerElement.duration) / 100;

  displayAudioProcessThread = setInterval(() => {
    if((audioPlayerElement.currentTime / audioPlayerElement.duration) * 100) {
      const durationMin = parseInt(audioPlayerElement.duration / 60);
      const durationSec = parseInt(audioPlayerElement.duration % 60);
      const currentMin = parseInt(audioPlayerElement.currentTime / 60);
      const currentSec = parseInt(audioPlayerElement.currentTime % 60);
      audioPlayingTimeFrameElement.textContent =
        `${currentMin}:${currentSec} / ${durationMin}:${durationSec}`;

      audioProgressBarInputElement.value =

```

AI 가사 요청 과정 :

1. 클라이언트에서 우선 해당 플레이중인 곡을 **form-data**형태로 서버에 보내고 서버는 해당 곡을 서버에 저장을 하게 됩니다.
2. 그런 다음 서버는 **assembly ai api**를 사용해서 해당 곡의 **URL**을 생성하게 됩니다.
3. 해당 생성된 **URL**을 다시 요청하여 곡의 가사 데이터를 다시 클라이언트에게 보냅니다.

```

app.post( /data/lyrics , upload.single( file ), async (req, res) => {
  let audioFileURL = null;
  const apiKey = "a6170822afb2454e971c3e9e25ddfbf2";
  let transcription = null;

  try {
    const task1 = await requestAudioFileURL();
    const task2 = await requestAudioTranscription();
    const task3 = await deleteUploads();
    res.json(transcription);
  }
}

```

```

async function requestAudioFileURL() {
  const fileLocation = path.join(__dirname, `/Uploads/${req.file.originalname}`);
  const URL = 'https://api.assemblyai.com/v2/upload';
  const headers = {
    "Authorization": apiKey,
    "Content-Type": "application/octet-stream",
  }
  const formData = new FormData();
  formData.append('file', fs.createReadStream(fileLocation));

  await axios.post(URL, formData, { headers: headers })
    .then(response => {
      console.log('Response URL :', response.data.upload_url);
      audioFileURL = response.data.upload_url;
    })
    .catch(err => {
      throw new Error(err);
    });
}

```

```

async function requestAudioTranscription() {
  let transid = null;
  let URL = 'https://api.assemblyai.com/v2/transcript';
  let headers = {
    "Authorization": apiKey,
    "Content-Type": "application/json"
  };

  const requestData = {
    audio_url: audioFileURL,
    language_detection: true,
    sentiment_analysis: true,
    speech_model: 'nano',
    speaker_labels: true,
    speakers_expected: 10
  };

  await axios.post(URL, requestData, { headers: headers })
    .then(response => {
      transid = response.data.id;
    })
    .catch(err => {
      throw new Error(err);
    });

  URL = `https://api.assemblyai.com/v2/transcript/${transid}`;
}

```

가사표시 : 우선 가사를 표시하는 전체적인 구현방법은 아래 그림과 같이 생각하였습니다.



받은 가사 데이터와 해당 곡의 **Blob**의 전체 재생시간등을 이용하여 가사 표시 화면은 받은 가사의 문장에 따라 N등분을 하였습니다. 그리고 **LyricsWatcher**(가사감지기)를 실행하여 현재 재생시간이 특정 구간에 진입을 하면 해당 가사를 보여주게(포커싱) 했습니다. 각 기능에 대해 추가적인 설명을 하자면 다음과 같습니다

- **task 1** : 가사감지기를 초기화 합니다.

```
async function checkLyricsWatcher() {  
    if(lyricswatcher !== null) {  
        clearInterval(lyricswatcher);  
        setLyricsWatcher(null);  
  
        const parentDiv = lyricsFrameElement;  
        while (parentDiv.firstChild) {  
            parentDiv.removeChild(parentDiv.firstChild);  
        }  
    }  
}
```

- **task 2** : 중복작업을 방지하기 위해 해당 버튼을 비활성화 합니다.

```
async function disableLyricsClick() {  
    lyricsImageElement.removeEventListener("click", clickLyricsImageElement);  
}
```

- **task 3** : 로딩중을 표시하기 위해 해당 엘리먼트를 삽입합니다.

```
async function insertLoading() {
  const parentDiv = lyricsFrameElement;
  const childDiv = document.createElement("div");

  childDiv.classList.add('loading-frame');
  childDiv.textContent = "Loading...";
  parentDiv.appendChild(childDiv);
}
```

- **task 4** : 서버로 가사를 요청하는 부분 입니다.

```
async function requestLyrics() {
  const formData = new FormData();
  formData.append('file', nowPlayingAudioFile);

  await axios.post('http://localhost:3000/data/lyrics', formData, {
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  })
  .then(response => {
    recvData = response.data;
    // console.log(recvData.sentences);
    if(response.data === null) {
      throw new Error("client has been received null data");
    }
  })
  .catch(err => {
    throw new Error(err);
  });
}
```

- **task 5** : 가사를 가져오는데 성공했으면 해당 로딩중 표시를 제거합니다.

```
async function clearLoading() {
  const parentDiv = lyricsFrameElement;
  const childDiv = parentDiv.querySelector(".loading-frame");

  parentDiv.removeChild(childDiv);
}
```

- **task 6** : 불러온 가사를 삽입하는 코드 입니다.

```
async function insertLyricsFrame() {
  const parentDiv = lyricsFrameElement;
  for(const sentence of recvData.sentences) {
    const childDiv = document.createElement("div");

    childDiv.classList.add('sentence-frame');
    if (!childDiv.hasAttribute('tabindex')) {
      childDiv.setAttribute('tabindex', '0');
    }
    childDiv.textContent = sentence.text;
    parentDiv.appendChild(childDiv);
  }
}
```

- **task 7** : 가사를 넣은 작업을 완료했으니 가사감지기를 실행하여 해당 가사를 추적하도록 합니다.

```
async function runLyricsWatcher() {
  const divs = document.querySelectorAll(".sentence-frame");
  setLyricsWatcher(setInterval(() => {
    for(const sentence of recvData.sentences) {
      const startTime = sentence.start * 0.001;
      const endTime = sentence.end * 0.001;
      const divPos = recvData.sentences.indexOf(sentence);

      if(audioPlayerElement.currentTime >= startTime && audioPlayerElement.currentTime <= endTime) {
        divs[divPos].focus();
        break;
      }
    }
  }, 100));
}
```

- **task 8** : 비활성화된 가사 요청 버튼을 다시 활성화 시킵니다.

```
async function enableGenerateLyricsClick() {
  lyricsImageElement.addEventListener("click", clickLyricsImageElement);
}
```

3. 프로젝트 느낀점

설명 : 해당 프로젝트를 진행하면서 **JS, HTML, CSS** 등을 공부하고 연습을 하면서 프론트엔드를 어떻게 다루게 되는지에 대해 좀 더 알게되었고 서버(**Node.js**), **API** 등 백엔드에 대한 개념과 이에 대한 구현에 대해 열심히 공부를 하고 구현한 결과를 통해 웹개발 프로세스 이해에 좀 더 다가갈 수 있었습니다.