ChatGPT

AI 기반 TTS 방송 서비스 개발설계서

개요 (Overview)

AI 기반 TTS(Text-to-Speech, **문자 음성 변환**) 방송 서비스는 라디오 방송국이나 콘텐츠 크리에이터가 사람 대신 AI 음성을 활용하여 안내 방송이나 콘텐츠를 제작하고 예약된 시간에 자동 송출할 수 있게 해주는 시스템입니다. 주요 사용자는 정해진 시간에 공지사항이나 안내 멘트를 송출해야 하는 **방송 진행자**나 **프로듀서**, 그리고 **콘텐츠 관리자** 등입니다. 이들은 손쉽게 웹 인터페이스를 통해 AI 음성을 생성하고, 원하는 스케줄에 맞춰 음성을 송출함으로써 **방송 업무의 효율화**와 **콘텐츠 생산성 향상**을 얻을 수 있습니다.

핵심 기능은 다음과 같습니다:

- **실감 나는 AI 음성 합성**: 다양한 목소리 스타일의 TTS 엔진을 활용하여 현실감 있고 감정 표현이 풍부한 음성을 생성합니다 [1]. 사용자는 기본 제공 음성 외에도 자신의 음성을 **보이스 클로닝**하여 AI로 복제하는 기능을 사용할 수 있습니다.
- **방송 콘텐츠 스케줄링**: 생성된 음성 오디오를 원하는 날짜와 시간에 **스케줄 등록**할 수 있으며, 시스템은 정해진 시각에 자동으로 해당 음성을 송출합니다. 반복 방송 설정, 우선 순위 관리 등의 일정 관리 기능도 포함됩니다.
- **웹 기반 프론트엔드 UI**: 누구나 쉽게 사용할 수 있는 **React 기반의 단일 페이지 애플리케이션(SPA)**을 제공하여, TTS 생성, 오디오 미리듣기, 스케줄 관리, 결제 관리 등의 모든 기능을 웹에서 직관적으로 조작할 수 있습니다.
- **결제 및 이용권 관리**: 서비스 이용량이나 구독 플랜에 따라 결제가 필요할 경우 **Lemon Squeezy** 플랫폼과 연동하여 구독/결제를 처리합니다. Lemon Squeezy의 API와 webhook을 통해 결제 내역을 확인하고 구독 상태를 관리합니다 [2].
- **로컬 에이전트를 통한 송출**: 인터넷이 연결된 방송 PC에 **로컬 에이전트** 프로그램을 설치하여, 중앙 서버에서 생성된 음성 파일을 해당 방송국 현장에서 재생합니다. 이 로컬 에이전트는 방송국 장비와 연동되어 **저지연(低遲延)**으로 음성을 출력합니다.

시스템 아키텍처 (System Architecture)

시스템은 크게 **프론트엔드**, **백엔드 서버**, **데이터베이스**, **로컬 에이전트**, 그리고 **외부 API 서비스**들로 구성됩니다. 전체적인 아키텍처는 다음과 같습니다:

- **프론트엔드 (React SPA)**: 사용자에게 서비스 UI를 제공하는 Single Page Application입니다. 로그인 및 사용자 관리, TTS 요청, 스케줄 관리 등의 화면으로 구성되어 있으며, 백엔드 API를 호출하여 데이터를 주고받습니다.
- **백엔드 (REST API 서버)**: 중심 비즈니스 로직을 담당하는 서버로, 프론트엔드와 로컬 에이전트에 API를 제공합니다. 사용자 인증, TTS 생성 요청 처리, 스케줄 저장 및 관리, 결제 웹훅 처리 등의 기능이 모듈화되어 있습니다.
- **데이터베이스 (DB)**: 서비스의 영속 데이터 관리에 사용됩니다. 사용자 계정, 생성된 오디오 정보, 스케줄 설정, 결제 및 구독 정보 등을 저장합니다. 관계형 DB를 사용하여 **ERD** 기반으로 스키마를 구성하였습니다.
- **로컬 에이전트 (방송 PC용 클라이언트)**: 방송 현장에 설치되는 경량 프로그램으로, 백엔드와 **WebSocket**으로 연결되어 실시간 명령(예: "지금 A오디오 재생")을 수신합니다. 필요시 백엔드에서 음원 파일을 다운로드하여 로컬 디바이스의 오디오 출력으로 송출합니다.
- **외부 API 연동**: 음성 합성을 위한 **Supertone API**와 결제 처리를 위한 **Lemon Squeezy API**를 사용합니다. Supertone API를 통해 고품질 AI 음성 합성 및 보이스 클로닝 기능을 구현하고, Lemon Squeezy를 통해 이용권 구매 및 결제 처리 로직을 구현합니다.

이러한 컴포넌트들은 **논리 다이어그램** 상에서 다음과 같이 서로 상호작용합니다: 프론트엔드가 사용자의 요청을 받아 백엔드 API 호출 → 백엔드가 필요한 경우 DB를 조회/갱신하거나 외부 API(Supertone, Lemon Squeezy)를 호출 → 생성된 결과(예: 합성 음원 또는 결제 확인)를 DB에 저장하고 응답 → 프론트엔드가 결과를 사용자에게 표시 → 스케줄된 작업의 경우 로컬 에이전트에 신호를 보내 음원 재생. 이 전체 흐름은 모듈 간 느슨한 결합을 유지하며, 네트워크 통신은 HTTPS(REST API) 및 WebSocket 프로토콜을 사용합니다.

프론트엔드 설계 (Frontend Design)

프론트엔드는 **React**를 기반으로 SPA 구조로 개발됩니다. 주요 설계 사항은 다음과 같습니다:

- **프로젝트 구조**: 기능별로 폴더를 구성하고 재사용 가능한 컴포넌트들을 모듈화합니다. 예를 들어 `components/` 디렉토리에 공통 UI 컴포넌트(Button, Modal, AudioPlayer 등)를 두고, `pages/` 디렉토리에 주요 페이지별 컴포넌트를 둡니다 (예: LoginPage, DashboardPage, SchedulePage 등).
- **라우팅 및 내비게이션**: React Router를 이용하여 클라이언트 사이드 라우팅을 구성합니다. 주요 경로는 로그인(`/login`), 대시보드(`/dashboard`), TTS 생성(`/tts`), 스케줄 관리(`/schedule`), 결제/구독(`/pricing` 혹은 `/account`) 등으로 구성됩니다. 인증이 필요한 페이지에 대해 **Route Guard**를 적용하여 로그인되지 않은 사용자는 로그인 페이지로 리디렉션합니다.
- **상태 관리**: 전역 상태는 React의 Context API나 상태 관리 라이브러리(예: Redux 또는 Recoil)를 통해 관리합니다. 예를 들어 사용자 인증 정보(JWT 토큰 등), 현재 선택된 음성 정보, 생성된 오디오 리스트, 스케줄 데이터 등을 전역 상태로 유지하여 여러 컴포넌트에서 쉽게 활용합니다.
- **재사용 가능한 UI 컴포넌트**: 일관된 UI/UX를 위해 디자인 시스템에 따라 버튼, 폼 입력, 테이블, 카드 등의 공통 컴포넌트를 구현합니다. 오디오 플레이어나 달력 위젯(캘린더) 등은 서드파티 라이브러리를 활용하되, 서비스 디자인 가이드에 맞게 스타일을 커스터마이징합니다.
- **UI/UX 원칙**: 반응형 웹 디자인을 채택하여 다양한 해상도에서 적절히 동작하도록 합니다. 사용 편의를 위해 **드래그 앤 드롭**으로 TTS 원고(text) 파일을 업로드하거나, 스케줄 표를 **캘린더 뷰**로 제공하여 일정 관리의 가시성을 높입니다. 또한 TTS 음성 합성이나 음원 재생 등의 **로딩 상태를 시각적으로 표시**하고, 에러 발생 시 사용자에게 이해하기 쉬운 메시지를 제공하여 신뢰성을 높입니다.

백엔드 설계 (Backend Design)

백엔드는 RESTful API 서버로 구현되며, 주요 기능 영역별로 모듈이 구성됩니다. **Node.js (Express)** 또는 **Python (Django/FastAPI)** 등의 프레임워크를 사용할 수 있으며, 본 설계에서는 이해를 돕기 위해 Express 기반으로 예시를 듭니다. 주요 설계 요소는 다음과 같습니다:

- **API 엔드포인트 설계**: 클라이언트와 통신하기 위한 REST API를 설계합니다. 주요 엔드포인트는 아래와 같습니다:
- `POST /api/auth/register` & `POST /api/auth/login`: 사용자 회원가입 및 로그인 (JWT 발급).
- `GET /api/user/profile`: 사용자 정보 및 구독 상태 조회.
- `POST /api/tts/generate`: TTS 음성 생성 요청. 요청 본문에 텍스트와 음성 설정(성우 ID 혹은 보이스 클로닝 ID 등)을 포함. 백엔드에서는 Supertone API를 호출하여 음성 합성을 수행합니다 [1]. 합성 완료 후 생성된 오디오 파일 경로 또는 ID를 응답으로 반환합니다.
- `POST /api/tts/clone-voice`: 보이스 클로닝 요청. 사용자로부터 업로드된 음성 샘플 파일을 받아 Supertone의 보이스 클로닝 API에 전달합니다. Supertone API를 통해 약 10초 분량의 음성으로 AI 목소리를 생성하며, 등록된 커스텀 음성 ID를 획득합니다 [3]. 이 ID를 DB에 저장하고, 사용자 계정에 연결합니다.
- `GET /api/audio/list`: 사용자가 생성한 음성 오디오 목록 조회 (페이지네이션 지원).
- `POST /api/schedules`: 새 방송 일정 등록. 본문에 어떤 오디오(또는 TTS 텍스트)를 언제 재생할지 시간 정보 포함. 등록 시 해당 일정 데이터를 DB에 저장하고, 연결된 로컬 에이전트에 실시간으로 전달합니다 (WebSocket을 통해 스케줄 업데이트 메시지 전송).
- `GET /api/schedules`: 향후 예정된 일정 목록 조회 (필요시 과거 실행 내역 포함).
- `DELETE /api/schedules/{id}`: 예약된 일정 취소/삭제.

- `POST /api/payment/webhook`: Lemon Squeezy 결제 webhook 엔드포인트. 사용자의 결제 성공, 구독 갱신, 취소 등의 이벤트를 수신하여 DB의 사용자 구독 상태를 갱신합니다.
- `GET /api/admin/*`: (선택사항) 관리자용 통계나 시스템 상태 확인을 위한 엔드포인트.

- **모듈 구성**: 코드 구조는 기능 영역별로 service, controller, model 계층으로 나누어 설계합니다. 예를 들어 AuthController/AuthService (인증), TTSController/TTSService (TTS 생성 및 보이스 클로닝), ScheduleController/ScheduleService (일정관리), PaymentController/PaymentService (결제 및 구독) 등의 모듈로 분리합니다. 각 서비스는 필요한 경우 DB 모델을 통해 데이터에 접근하고, 외부 API 호출이 필요한 경우 전용 클라이언트 모듈 (예: SupertoneClient, LemonSqueezyClient)를 사용합니다. 또한 특정 작업은 **비동기 작업 큐**를 사용하여 처리할 수 있습니다 (예: TTS 합성 요청을 큐에 넣고 완료 후 사용자에게 알림).

- **Supertone API 연동**: 백엔드에서 **Supertone API**를 호출하여 실시간으로 TTS 합성을 수행합니다. Supertone API의 엔드포인트(URL, 인증키 등)는 환경 변수로 관리하며, HTTP POST로 텍스트와 음성 설정(성별, 감정 톤 등)을 보내면 음성 오디오 데이터를 반환받습니다. Supertone API는 문장을 단순 낭독하는 것을 넘어 맥락을 이해하고 감정을 담아 음성을 합성하는 것이 특징입니다 [1]. 보이스 클로닝의 경우 사용자가 업로드한 샘플 음성을 Supertone 쪽에 전달하여 고유한 Voice ID를 받고, 해당 Voice ID로 추후 TTS를 호출합니다 [3].
- **Lemon Squeezy 결제 연동**: 결제 및 구독 관리를 위해 **Lemon Squeezy API**를 사용합니다. 백엔드 서버는 Lemon Squeezy 대시보드에서 발급한 API 키를 통해 결제 정보를 조회하거나 구독자를 생성할 수 있으며, 주로 웹훅(webhook) 엔드포인트를 통해 비동기로 결제 이벤트를 수신합니다. Lemon Squeezy는 예측 가능하고 RESTful한 API 및 webhook을 제공하므로 외부 결제 시스템과의 통합이 용이합니다 [2]. 예를 들어, 사용자가 서비스 내에서 구독 플랜을 구매하면 Lemon Squeezy 측에서 결제 완료 웹훅을 호출하고, 우리의 `/api/payment/webhook` 엔드포인트에서 해당 사용자의 구독 활성화를 처리합니다.
- **스케줄링 로직**: 예약된 방송을 정확히 송출하기 위해 **스케줄러**가 필요합니다. 백엔드에서는 Cron 표현식 또는 전용 스케줄링 라이브러리(예: `node-cron` 등)를 활용하여 DB에 저장된 일정 시간을 모니터링합니다. 하지만 실시간성이 중요한 만큼, 백엔드가 일정 시간 도래를 감지하여 바로 로컬 에이전트로 명령을 보내는 구조로 구현합니다. 각 로컬 에이전트는 해당 방송국(사용자)에 속한 일정만 받아 처리하며, WebSocket 연결이 끊어진 경우를 대비해 **폴링**(polling) 또는 재연결시 미처리 일정 싱크(sync) 메커니즘을 갖춥니다. 또한 시간 동기화를 위해 모든 서버와 에이전트가 **UTC 시간 기준**으로 일정 시간을 관리하고, 프론트엔드에서는 사용자의 로컬 시간대를 고려해 표시합니다.
- **로깅 및 에러 처리**: 백엔드에서는 모든 주요 액션 (TTS 생성 요청, 스케줄 트리거, 결제 웹훅 등)에 대해 로그를 남깁니다. 로그는 파일 또는 중앙 로깅 시스템(예: ELK 스택)에 적재하여 추후 문제 발생 시 분석에 활용합니다. 에러 발생시 API 응답은 표준화된 오류 코드와 메시지 형식으로 처리하며, 알려진 예외 상황(예: 잔여 TTS 생성 크레딧 부족 등)에 대한 사용자 친화적인 에러 메시지를 제공합니다.

데이터베이스 스키마 (Database Schema)

주요 엔티티(Entity)의 개념적 ERD는 다음과 같으며, 각 엔티티에 대한 스키마를 마크다운 테이블로 정리하면 아래와 같습니다.

User (사용자) – 서비스 이용자 계정 정보 및 구독 상태
| 필드 | 타입 | 설명 | | ---------- | ---------- | ---------- | | id | PK (UUID) | 사용자 고유 ID | | email | VARCHAR | 이메일 (로그인 ID) | | password | VARCHAR | 해시된 비밀번호 | | name | VARCHAR | 이름 또는 방송국명 | | role | ENUM | 역할 (일반 사용자, 관리자 등) | | plan | VARCHAR | 구독 플랜 이름 (예: Free, Pro) | | plan_expiry | DATETIME | 구독 만료 일자 (유료 플랜 시) | | created_at | DATETIME | 계정 생성일 | | updated_at | DATETIME | 정보 수정일 |

Audio (생성 음원) – 생성된 음성 오디오에 대한 메타데이터
| 필드 | 타입 | 설명 | | ---------- | ---------- | ---------- | | id | PK (UUID) | 음원 고유 ID | | user_id | FK → User | 생성한 사용자 ID | | text | TEXT | 합성에 사용된 원본 텍스트 | | voice | VARCHAR | 사용된 음

성 유형 (성우 이름 또는 Voice ID) | | audio_url | VARCHAR | 저장된 음원 파일 경로 또는 URL | | duration | INT | 음원 재생 길이 (초) | | created_at | DATETIME | 생성일 | | used_in_schedule | BOOLEAN | 스케줄 사용 여부 (스케줄에 추가되었는지) |

Schedule (방송 일정) – 특정 시간에 특정 음원을 재생하기 위한 예약 정보
| 필드 | 타입 | 설명 | | ---------- | ---------- | ---------- | | id | PK (UUID) | 일정 고유 ID | | user_id | FK → User | 일정을 생성한 사용자 (또는 방송국) | | audio_id | FK → Audio | 재생할 Audio ID (또는 대체: TTS text 직접 참조) | | scheduled_at | DATETIME | 예정된 방송 시각 (UTC 기준) | | status | ENUM | 상태 (예정됨 Scheduled, 재생완료Played, 취소Cancelled 등) | | created_at | DATETIME | 일정 등록일 | | updated_at | DATETIME | 수정일 (일정 변경 시) |

Payment (결제/구독) – Lemon Squeezy와 연동한 결제 및 구독 내역
| 필드 | 타입 | 설명 | | ---------- | ---------- | ---------- | | id | PK (UUID) | 결제 기록 ID | | user_id | FK → User | 해당 결제를 한 사용자 | | plan | VARCHAR | 구매한 플랜/제품 명 (예: Pro Monthly) | | order_id | VARCHAR | Lemon Squeezy 상의 주문 ID | | status | VARCHAR | 결제 상태 (paid, refunded 등) | | amount | INT | 결제 금액 (통화 단위 등은 별도 필드 가능) | | created_at | DATETIME | 결제 시각 | | next_billing_at | DATETIME | 다음 결제 예정 시각 (구독인 경우) |

VoiceProfile (보이스 클로닝 프로필) – 사용자 맞춤 음성 (보이스 클론) 정보
| 필드 | 타입 | 설명 | | ---------- | ---------- | ---------- | | id | PK (UUID) | 보이스 프로필 ID | | user_id | FK → User | 프로필 소유 사용자 ID | | name | VARCHAR | 프로필 이름 (사용자가 붙인 이름, 예: "DJ현우 목소리") | | supertone_id | VARCHAR | Supertone 상의 Voice ID (클로닝된 음성 식별자) | | created_at | DATETIME | 프로필 생성일 |

> **참고:** 위 스키마는 간략화된 형태입니다. 실제 구현에서는 결제 금액의 통화(currency) 필드, 구독과 결제의 관계 테이블, 또는 Audio와 Schedule 간 N:M 관계(하나의 음원을 여러 일정에서 사용할 경우) 등에 대한 고려가 추가될 수 있습니다. 또한 대용량 텍스트 처리를 위해 `text` 필드 대신 별도 원고 테이블을 두거나, Audio 테이블의 `audio_url` 대신 파일 스토리지 연동(예: AWS S3 경로)을 사용할 수 있습니다.

로컬 에이전트 (Local Agent)

로컬 에이전트는 방송국의 PC나 서버에 설치되어 백엔드와 지속적으로 통신하는 경량 클라이언트 애플리케이션입니다. 이 에이전트의 주요 목적은 **백엔드로부터 스케줄 명령을 받아 로컬 시스템에서 실제로 음원을 재생**하는 것입니다. 설계의 핵심 사항은 다음과 같습니다:

- **역할 및 동작**: 로컬 에이전트는 백엔드에 **웹소켓(WebSocket)** 연결을 수립하고 대기합니다. 해당 방송국(사용자)에 새로운 스케줄이 등록되거나 변경되면 백엔드가 실시간으로 에이전트에 메시지를 push하여 (예: “schedule_123: 15:00에 audio_456 재생”) 일정을 전달합니다. 에이전트는 수신한 일정을 내부 스케줄러에 등록하고, 지정된 시각이 되면 로컬에 저장된 음원 파일을 재생합니다. 재생 전 음원 파일이 없다면 백엔드의 API (예: `GET /api/audio/{id}/file`)를 통해 파일을 다운로드하여 캐싱합니다.
- **설치 및 환경**: 에이전트는 **Cross-platform**으로 Windows/Linux 환경에서 동작하도록 Node.js 혹은 Python 기반으로 개발됩니다 (예: Electron이나 Python 실행 파일 형태). 설치 프로그램을 제공하여 사용자가 손쉽게 설치하도록 하고, 최초 실행 시 사용자 로그인(Token 발급)을 받아 해당 PC를 사용자 계정에 연결합니다. 연결 후에는 시스템 트레이에 상주하며 백그라운드에서 자동 시작되도록 설정할 수 있습니다.
- **통신 프로토콜**: WebSocket 메시지는 가볍고 단순한 **JSON 프로토콜**로 설계합니다. 예: `{ "action": "PLAY", "audioId": "456", "time": "2025-11-01T06:00:00Z" }` 와 같이 명령 타입(action)과 관련 데이터(audio 파일 ID, 예약 시간 등)을 포함합니다. 에이전트는 확인 시 `{ "ack": "schedule_received", "scheduleId": "123" }`와 같이 응답하여 백엔드가 정상 수신 여부를 알 수 있게 합니다. 또한 에이전트 측에서 오류(예: 파일 재생 실패, 네트워크 끊김 등)가 발생하면

`{ "error": "...", "message": "파일 재생 실패" }` 형태로 서버에 전송하여 모니터링이 가능하도록 합니다.

- **방송 출력 연동**: 로컬 에이전트는 해당 시스템의 오디오 출력 장치(사운드카드 출력)를 통해 소리를 재생합니다. 필요에 따라 방송국의 송출장비와 연동하기 위해 재생 장치를 선택하거나, 별도의 오디오 라우팅 설정을 제공할 수 있습니다. (예: 가상 오디오 장치를 통해 믹싱 장비로 전달 등). 에이전트에서는 일반 멀티미디어 재생 라이브러리 (예: VLC 또는 FFmpeg 라이브러리 바인딩)를 활용하여 지연을 최소화하고 정확한 시각에 시작하도록 튜닝합니다.
- **보안 및 인증**: 로컬 에이전트와 서버 간 통신은 TLS로 암호화된 WebSocket (wss://)을 사용하며, 연결 핸드셰이크 시에 JWT 또는 API 키를 이용한 인증을 수행합니다. 이를 통해 **인증된 에이전트**만 서버 명령을 수신할 수 있게 하고, 제3자가 명령을 끼워넣지 못하도록 합니다. 또한 에이전트 자체에 무결성 체크를 적용하여 임의 변조된 소프트웨어가 아닌 공식 배포본만 동작하도록 합니다.
- **업데이트**: 에이전트는 자동 업데이트 기능을 내장하여 새로운 버전이 릴리스될 경우 사용자의 개입 없이도 최신 버전을 유지합니다. 예를 들어, 정기적으로 업데이트 서버를 체크하거나, 백엔드로부터 "업데이트 가능" 통지를 받아 새 바이너리를 다운로드 후 재시작합니다. 이를 통해 클라이언트-서버 프로토콜 변경이나 기능 개선 사항을 신속히 배포할 수 있습니다.

시퀀스 다이어그램 (Sequence Diagrams)

1. TTS 음성 생성 흐름: AI 음성 합성을 요청하고 음원을 얻는 과정
1. **사용자**가 프론트엔드 UI에서 TTS 생성 페이지를 열고, 텍스트 입력과 음성 유형(예: 기본 여성 목소리 또는 자신의 보이스 클론)을 선택한 후 "합성" 요청을 보냅니다.
2. **프론트엔드**는 `/api/tts/generate` 엔드포인트로 텍스트와 음성 옵션 데이터를 포함하여 **POST 요청**을 보냅니다. 이때 사용자 인증 토큰(JWT)을 헤더에 첨부하여 보냅니다.
3. **백엔드**의 TTSController는 요청을 받아 입력 텍스트의 길이, 사용자 권한(예: 사용 가능 크레딧) 등을 검증합니다. 이상 없으면 내부에서 Supertone API 클라이언트를 호출하여 **음성 합성 작업**을 수행합니다.
4. **Supertone API** 서버는 요청받은 텍스트와 지정된 음성에 따라 TTS 엔진으로 음성 합성을 수행하고, 결과 음성 오디오 데이터를 반환합니다. 백엔드는 이 오디오 데이터를 받아 서버 내 파일 스토리지 (예: `public/audio/` 경로 또는 S3 버킷)에 저장합니다. 저장된 파일의 URL 혹은 ID를 획득한 뒤, Audio 테이블에 새로운 레코드를 생성하여 메타데이터를 저장합니다.
5. 백엔드는 프론트엔드에게 합성 완료 응답을 보냅니다. 응답 내용에는 생성된 **Audio ID** 또는 **오디오 파일 URL**, 음성의 길이 등이 포함됩니다.
6. **프론트엔드**는 응답을 받고 사용자에게 "음성 합성이 완료되었습니다"라는 알림을 표시합니다. 생성된 음성을 바로 재생해볼 수 있도록 플레이어를 표시하고, 사용자가 원한다면 해당 음성을 스케줄에 추가할 수 있는 UI 옵션을 제공합니다.

2. Voice Cloning (보이스 클로닝) 흐름: 사용자의 목소리를 AI로 복제하는 과정
1. **사용자**가 프론트엔드에서 보이스 클로닝 기능을 선택하고, 안내에 따라 자신의 육성 음성 샘플 (예: 10초 분량의 녹음 파일)을 업로드합니다.
2. **프론트엔드**는 해당 음성 파일 (예: WAV 또는 MP3)을 백엔드 `/api/tts/clone-voice` 엔드포인트로 **POST 업로드**합니다. 이 요청에는 사용자의 인증 정보와 함께, 음성 파일 바이너리가 폼데이터로 전송됩니다.
3. **백엔드**의 TTSController (또는 VoiceCloneController)는 업로드된 파일을 일시 저장한 뒤, Supertone API의 보이스 클로닝 호출을 시작합니다. Supertone API에 사용자 음성 샘플을 전송하여 새로운 **Voice ID**를 생성하는 프로세스를 비동기로 요청합니다.
4. **Supertone API**에서는 전달된 샘플을 바탕으로 AI 목소리 프로필을 만들고 고유 식별자(Voice ID)를 반환합니다. (이 단계는 다소 시간이 걸릴 수 있어, API 응답을 polling하거나 웹훅 등으로 처리할 수도 있습니다.)
5. 백엔드는 Supertone로부터 받은 Voice ID를 **VoiceProfile** 테이블에 저장하고, 해당 VoiceProfile을 사용자 계정과 연동합니다. 이제 사용자는 TTS 생성 시 이 Voice ID를 선택하여 자신의 목소리로 합성된 음성을 얻을 수 있게 됩니다 [3].
6. **프론트엔드**는 클로닝 완료 결과(성공/실패)를 백엔드로부터 받아 사용자에게 피드백을 제공합니다. 성공 시 "새로운 보이스 프로필이 등록되었습니다"와 같이 표시하고, TTS 생성 화면의 음성 선택 목록에 해당 사용자 목소리가 옵션으로 추가됩니다. 실패 시 오류 메시지를 안내하고 재시도 옵션을 제공합니다.

3. 방송 스케줄 실행 흐름: 예약된 시간에 AI 음성을 자동 방송하는 과정
1. **사용자**는 프론트엔드의 스케줄 관리 화면에서, 생성된 음성 오디오를 선택하고 방송할 날짜와 시각을 지정하여 새로운 일정을 추가합니다. "저장"을 누르면 프론트엔드는 `/api/schedules`로 일정 정보(어떤 Audio ID를 언제 재생할지)를 **POST** 요청합니다.
2. **백엔드**는 일정 생성 요청을 받아 Schedule 테이블에 레코드를 생성합니다. 이때 연결된 user_id, audio_id와 예약 시각 등이 저장됩니다. 저장이 완료되면 성공 응답을 프론트엔드로 보냅니다.
3. 일정 생성과 동시에 백엔드는 해당 사용자에 연결된 **로컬 에이전트**가 현재 온라인(연결 중)인지 확인합니다. WebSocket을 통해 해당 에이전트 클라이언트에게 신규 일정이 등록되었음을 이벤트로 전송합니다. 예를 들어 `{ action: "NEW_SCHEDULE", schedule: {id, audioId, time, ...} }` 형태로 보냅니다. 에이전트가 오프라인 상태라면, 나중에 에이전트가 재연결할 때 누락된 일정을 다시 동기화하도록 합니다.
4. **로컬 에이전트**는 NEW_SCHEDULE 이벤트를 수신하면, 우선 해당 audioId의 음원 파일이 로컬에 있는지 확인합니다. 없을 경우 백엔드의 파일 다운로드 API를 호출하여 미리 파일을 받아 캐싱해둡니다. 그리고 내부 타이머 또는 스케줄러에 해당 이벤트를 등록하여 지정된 시간에 재생이 트리거되도록 합니다.
5. 예약된 시각이 되면, 로컬 에이전트는 지정된 audio 파일을 오디오 출력 장치로 **재생(start)** 합니다. 정시에 정확히 송출될 수 있도록 시스템 시계를 참고하여 딜레이를 최소화합니다. 음성 출력이 시작될 때 에이전트는 백엔드에 `{ action: "PLAY_STARTED", scheduleId: ... }` 같은 메시지를 보낼 수 있고, 재생이 완료되면 `{ action: "PLAY_COMPLETED", scheduleId: ... }` 메시지를 추가로 보낼 수 있습니다. 이를 통해 백엔드는 해당 일정의 상태를 "재생완료"로 업데이트하고 모니터링 용도로 로그를 남깁니다.
6. **백엔드**와 **프론트엔드**는 일정이 완료되었음을 인지하고, 프론트엔드는 UI 상에서 해당 일정의 상태를 "완료"로 표시하거나 사용자에게 완료 알림(예: "15:00 안내 방송 송출 완료")을 보여줄 수 있습니다. 만약 일정 시간에 에이전트가 응답하지 않거나 재생 실패가 감지되면, 백엔드는 해당 일정 상태를 "실패"로 표시하고 사용자에게 경고하도록 합니다.

배포 및 운영 (Deployment Notes)

서비스의 배포 환경과 확장성, 구성 옵션에 대한 고려사항은 다음과 같습니다:

- **기술 스택**:
- 프론트엔드: React 앱을 빌드하여 정적 파일(HTML/CSS/JS) 형태로 Nginx 등의 웹서버 또는 CDN을 통해 제공합니다. 개발 단계에서는 `create-react-app` 또는 Next.js 등을 활용하고, 프로덕션 빌드시 최적화된 번들을 배포합니다.
- 백엔드: Node.js 기반이면 PM2 등의 프로세스 매니저를 사용하거나 Docker 컨테이너로 이미지화하여 배포합니다. Python 기반이면 uWSGI+Nginx 또는 Gunicorn 등을 사용하거나 역시 Docker로 배포할 수 있습니다.
- 데이터베이스: 클라우드 관리형 DB (예: AWS RDS의 PostgreSQL 등)를 사용하는 것을 권장하며, 초기 소규모 단계에서는 SQLite나 단일 노드로 시작하고 사용자 증가 시 수평/수직 확장을 고려합니다.

- 로컬 에이전트: 이용자(방송국) 환경에 따라 Windows 서비스 혹은 Linux 데몬으로 설치하여 실행합니다. 자동 업데이트 및 장애 복구를 위해 Sentry 등의 모니터링 도구를 내장할 수 있습니다.

- **인프라 구성**:

- 애플리케이션 서버는 **클라우드 플랫폼**(AWS, GCP 등)에 배포하여 가용성 및 확장성을 확보합니다. 다중 인스턴스에 무중단 배포를 위해 로드밸런서와 무중단 배포(블루-그린 배포 또는 롤링 업데이트) 전략을 채택합니다.
- **파일 스토리지**: 생성된 음원 파일이 많아질 수 있으므로, AWS S3와 같은 오브젝트 스토리지와 CDN을 활용하여 사용자에게 음원을 제공하는 것이 바람직합니다. 백엔드 서버 로컬 디스크에 저장하는 경우 용량 관리 및 백업 계획이 필요합니다.
- **확장성**: TTS 합성 요청 등 CPU 부하가 큰 작업은 별도의 **작업 큐**와 워커(Worker) 프로세스를 두어 처리하고, 필요시 워커 인스턴스를 늘려서 확장할 수 있습니다. 웹 서버는 무상태(stateless)하게 설계되어 세션 정보는 JWT로 처리하거나 Redis 같은 인메모리 스토리지에 유지하여, 서버 증설시 부하 분산이 가능합니다. 데이터베이스는 읽기 부하가 증가하면 **리드 레플리카**를 추가하고, 쓰기 부하가 많아지면 파티셔닝 혹은 샤딩 전략을 고려합니다.

- **모니터링**: 서버 애플리케이션 모니터링을 위해 APM(Application Performance Monitoring) 도구 (예: New Relic, Datadog)를 사용하고, 로깅은 CloudWatch나 ELK 스택으로 중앙 수집합니다. 또한 서비스 헬스체크 엔드포인트 (`/api/health`)를 구현하여 인프라에서 주기적으로 호출, 응답 상태를 확인합니다.

- **CI/CD 파이프라인**: Git 등의 소스 저장소에 대한 CI/CD를 구축하여, main 브랜치에 푸시될 경우 자동으로 빌드와 테스트, 그리고 스테이징/프로덕션 배포가 이뤄지도록 합니다. Docker 이미지를 빌드해 레지스트리에 업로드하고, 배포 스크립트가 해당 이미지를 내려받아 컨테이너를 업데이트하는 절차를 밟습니다.

- **구성 옵션 및 설정**:

- 환경별 설정 분리를 위해 **환경 변수(.env)** 파일 및 설정 파일을 사용합니다. 예: 개발 환경에서는 TTS 요청을 테스트 모드로 보내고, 프로덕션에서는 실 서비스 API를 사용하도록 분기.
- 주요 기능(예: 보이스 클로닝, 결제) 사용 여부를 구성 옵션화하여 필요 없는 경우 쉽게 비활성화할 수 있도록 설계합니다. 예를 들어, 개발 단계나 데모용 서비스에서는 결제 시스템을 끄고 모든 사용자를 무료 플랜으로 동작시킬 수 있습니다.
- **보안 설정**: API 키, DB 비밀번호 등 민감 정보는 환경 변수로 주입되고, 저장 시 암호화하거나 Git 등 버전 관리에 노출되지 않도록 철저히 관리합니다. 또한 모든 HTTP 통신은 TLS(HTTPS)를 사용하고, HSTS 및 CORS 설정을 올바르게 적용하여 클라이언트 보안을 강화합니다.

환경 변수 (Environment Variables)

시스템에서 사용되는 주요 환경 변수(.env 설정)는 다음과 같으며, 배포 시 적절한 값을 설정해야 합니다:

- **애플리케이션 및 서버 설정**
- `NODE_ENV` 또는 `ENV` : 실행 환경 모드 (development, production 등).
- `PORT` : 백엔드 서버가 사용할 포트 번호 (예: 3000).
- `BASE_URL` : 프론트엔드가 배포된 기본 URL (예: https://myservice.com).
- `JWT_SECRET` : 사용자 인증용 JWT 서명 비밀키.

- `CLIENT_ORIGIN` : 프론트엔드 웹사이트 주소 (CORS 허용을 위해 필요).

- **데이터베이스 연결**

- `DB_HOST` , `DB_PORT` : DB 호스트 주소 및 포트.
- `DB_USER` , `DB_PASS` : DB 인증 사용자와 비밀번호.
- `DB_NAME` : 데이터베이스 이름.

- (또는 하나로 통합된 `DATABASE_URL` 형태로 사용 가능).

- **외부 API 키 및 엔드포인트**

- `SUPERTONE_API_KEY` : Supertone API 사용을 위한 API 키.
- `SUPERTONE_API_URL` : Supertone TTS API 엔드포인트 기본 URL.
- `LEMON_API_KEY` : Lemon Squeezy API 키 (시크릿 키).
- `LEMON_WEBHOOK_SECRET` : Lemon Squeezy 웹훅 서명 검증 시 사용하는 시크릿.
- `LEMON_API_URL` : Lemon Squeezy API 베이스 URL (기본값 `https://api.lemonsqueezy.com/v1` 등).

- (선택) `PAYMENT_SUCCESS_URL` , `PAYMENT_CANCEL_URL` : 결제 완료나 취소 후 리디렉션할 프론트엔드 URL.

- **기타 서비스**

- `STORAGE_PATH` 또는 `S3_BUCKET_NAME`: 생성된 오디오 파일 저장 위치 (로컬 경로 또는 S3 버킷).
- `WS_SERVER_URL`: 로컬 에이전트용 WebSocket 서버 URL (예: wss://myservice.com/agent).
- `AGENT_UPDATE_URL`: 로컬 에이전트 최신 버전 다운로드 경로 (자동 업데이트용).
- `LOG_LEVEL`: 로그 레벨 설정 (info, debug, error 등).

주의: 실제 운영 시에는 위 환경 변수 외에도 SMTP 설정(이메일 발송용), OAuth 클라이언트 ID/시크릿(소셜 로그인용) 등이 추가될 수 있습니다. 여기서는 TTS 방송 서비스의 핵심 기능에 관련된 변수들만 열거합니다.

부록 (Appendix)

- **Supertone API 문서** – Supertone의 AI 음성 합성 및 관련 기능(보이스 클로닝 등)에 대한 공식 API 문서 및 가이드: Supertone API Documentation [1] [3]

- **Lemon Squeezy API 문서** – 결제 및 구독 관리를 위한 Lemon Squeezy 플랫폼의 개발자용 API 레퍼런스: Lemon Squeezy API Reference [2]

- **용어 설명 (Glossary)**

- **TTS (Text-to-Speech, 문자 음성 변환)**: 텍스트 형태의 문장을 입력받아 인간의 목소리처럼 들리는 음성으로 합성해주는 기술을 말합니다.
- **Voice Cloning (보이스 클로닝)**: 특정 화자의 음성 샘플을 기반으로 동일한 목소리 특징을 지닌 AI 모델을 만들어내는 기술입니다. 이를 통해 해당 화자의 목소리로 임의의 문장을 합성할 수 있습니다.
- **WebSocket**: 클라이언트(브라우저 또는 애플리케이션)와 서버 간에 **실시간 양방향 통신**을 가능하게 하는 프로토콜입니다. 본 서비스에서는 로컬 에이전트와 서버 간 실시간 제어를 위해 사용됩니다.
- **REST API**: HTTP 프로토콜을 기반으로 자원을 주소(URL)로 표현하고, 표준 HTTP 메서드(POST, GET 등)를 통해 조작하는 **웹 API 형식**을 의미합니다. 본 서비스의 백엔드는 RESTful API로 구현되어 프론트엔드 및 외부 시스템과 통신합니다.

---

[1] [3] Supertone API
https://www.supertone.ai/en/api

[2] Payments, tax & subscriptions for software companies • Lemon Squeezy
https://www.lemonsqueezy.com/