[README] Raspberry Pi STT & Local FAST API Server

순번	117
구분	Basic
개발 상태	완료
front 연결	완료

시스템 구성요소

1. FastAPI 서버 (local_server.py)

- **인증 시스템**: 로그인, 세션 관리, 사용자 정보 저장
- **블루투스 관리**: 스피커 연결/해제, 볼륨 제어
- **설정 관리**: 카메라/주행 설정 제어 및 전송
- **프로세스 관리**: 서비스 프로세스 시작/중지/모니터링

2. STT 엔진 (main.py)

- **음성 인식**: 키워드 감지, 음성 명령 해석
- **음성 합성**: 응답 메시지 음성 출력
- **메시지 시스템**: 텍스트/음성 메시지 송수신
- **오디오 스트림**: 실시간 음성 입력 처리

기술 스택

- **백엔드 프레임워크**: FastAPI
- **음성 인식**: Google Cloud Speech API
- **음성 합성**: Google Text-to-Speech (gTTS)
- **오디오 처리**: PyAudio, pygame
- **비동기 통신**: asyncio, aiohttp

- **하드웨어 통신**: 소켓 서버를 통한 라즈베리 파이 연동

FastAPI 서버 상세 기능

인증 시스템

```
@app.post("/api/login")
async def login(credentials: LoginCredentials):
    # 외부 API 인증 후 로컬 사용자 정보 저장
```

- 이메일/비밀번호 기반 인증
- 세션 ID 및 가족 ID 조회 및 저장
- 사용자 정보 로컬 파일 저장
- 인증 성공 시 필요 프로세스 자동 시작

블루투스 스피커 관리

```
@app.post("/bluetooth/speaker/connect")
@app.post("/bluetooth/speaker/toggle")
@app.post("/bluetooth/speaker/volume")
```

- MAC 주소 기반 블루투스 스피커 연결
- 서버 설정과 연동된 스피커 활성화/비활성화
- 볼륨 레벨(0-100%) 세부 제어
- ALSA/PulseAudio 시스템과 연동

설정 전송 및 관리

```
@app.post("/SettingValue")
async def SendToJetson(settings: SettingValue):
    # 설정값 라즈베리 파이로 전송
```

- 카메라 활성화 상태 관리
- 주행 기능 활성화 상태 관리
- 사용자 설정 저장 및 디바이스 전송

프로세스 관리

```
def start_processes(user_id: str, session_id: str):
def stop_processes():
```

- STT, RSVP, 소켓서버 프로세스 관리
- 프로세스 정상 종료 및 강제 종료 처리
- 시그널 핸들링을 통한 안전한 종료

STT 엔진 상세 분석

오디오 스트림 관리

```
class AudioStream:
    def __init__(self, rate=RATE, chunk=CHUNK):
        self._rate = rate
        self._chunk = chunk
        self._buff = queue.Queue()
        self.closed = True
        self._audio_interface = None
        self._audio_stream = None
        self._resource_lock = threading.Lock()
```

- **초기화**: 샘플링 레이트, 청크 크기 설정
- **스레드 안전성**: 리소스 락을 통한 동시성 제어
- **버퍼 관리**: 음성 데이터 큐 관리

```
def __enter__(self):
    # 오디오 스트림 초기화 및 시작
    # 블루투스 장치 우선 탐색
```

- 블루투스 오디오 장치 자동 감지
- 디바이스 인덱스 관리
- PyAudio 인터페이스 초기화

```
def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
    # 오디오 데이터 버퍼링
```

- 콜백 기반 데이터 캡처
- 버퍼 오버플로우 방지
- 에러 상태 감지 및 처리

```
def generator(self):
    # 적절한 크기의 오디오 청크 생성
```

- 일정 크기 단위로 데이터 제공
- 최대 청크 크기 제한
- 스트림 종료 감지

STT 매니저 (핵심 클래스)

```
class STTManager:
    def __init__(self):
        # Google Cloud 인증 및 초기화
        # 모드 및 상태 관리
```

- Google Cloud 인증 설정
- 오디오 출력 디렉토리 관리
- pygame 오디오 시스템 초기화
- 사용자 ID 로드

```
def get_config(self, mode: STTMode):
    # 음성 인식 설정 구성
```

- 모드별 최적화된 설정 제공
- 한국어 인식 설정
- 스트리밍 인식 설정

키워드 감지 시스템

```
async def detect_keyword(self):
    # 특정 키워드 감지 로직
```

- 지속적인 음성 모니터링
- 키워드 매칭 ("영웅", "영웅아" 등)
- 응급 키워드 감지 ("응급", "위급" 등)
- 메시지 키워드 감지 ("메시지" 등)
- 모드 전환 트리거

음성 명령 처리

```
async def process_speech(self):
    # 모드별 음성 처리 로직
```

- **일반 대화 모드**: 챗봇 API 연동
- **메시지 흐름 모드**: 수신자 지정 및 메시지 전송
- **응급 모드**: 긴급 메시지 처리
- 최종 인식 텍스트 반환

메시지 시스템

```
async def _send_message(self, content: str):
async def _send_single_message(self, to_id: str, content: str):
```

- 전체/개인 메시지 전송
- 응급 메시지 특수 처리
- API 응답 음성 출력

```
async def check_new_messages(self):
async def _mark_message_as_read(self, message_index: int):
```

- 주기적 새 메시지 확인
- 미확인 메시지 음성 읽기

- 메시지 읽음 상태 업데이트

음성 합성 (TTS)

```
def generate_speech(self, text):
    # gTTS 음성 합성
```

- 텍스트 음성 변환
- 임시 파일 관리
- 처리 시간 모니터링

```
def play_audio(self, audio_path):
    # 오디오 재생 및 정리
```

- pygame을 통한 음성 재생
- 재생 완료 대기
- 임시 파일 자동 삭제

가족 구성원 매칭

```
async def _find_most_similar_member(self, transcript: str) → str:
    # 음성 인식 결과와 가족 구성원 매칭
```

- 정확한 이름 매칭
- 부분 매칭 지원
- 한글 자음 기반 유사도 검사

에러 처리 및 로깅

```
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(),
        logging.FileHandler('stt_test.log')
```

```
    ]
)
```

- 콘솔 및 파일 로깅 병행
- 상세한 타임스탬프 및 로그 레벨
- PyAudio 에러 메시지 필터링

```
def py_error_handler(filename, line, function, err, fmt):
    pass
```

- ALSA/JACK 에러 메시지 억제
- 불필요한 경고 숨김

메인 루프

```
async def main():
    # 주기적 메시지 체크 및 키워드 감지
```

- STT 매니저 초기화
- 주기적 새 메시지 확인
- 키워드 감지 모니터링
- 시각적 피드백 신호 전송

설치 및 실행 요구사항

시스템 요구사항

- Python 3.11 이상
- Google Cloud 계정 및 인증 키
- 블루투스 지원 하드웨어
- ALSA/PulseAudio 시스템

필수 패키지

- FastAPI, uvicorn

- PyAudio
- pygame
- Google Cloud Speech
- gTTS
- aiohttp, httpx
- dotenv

환경 설정

1. Google Cloud 서비스 계정 키 설정
2. `.env` 파일에 `GOOGLE_CREDENTIALS_PATH` 설정
3. 블루투스 장치 MAC 주소 확인 (필요시 코드 수정)
4. 사용자 ID 파일 (`user_id.txt`) 준비

실행 방법

FastAPI 서버 실행

```
python local_server.py
```

STT 서비스 독립 실행(선택사항)

```
python main.py
```