**SWDS – [Submitted in partial fulfillment**

**of the requirements of Capstone Design 1 (Class A)]**

**Omnibus**

**Voice-Based Bus Route Guidance System**

Software Design Specification

텍스트, 로고, 폰트, 상징이(가) 표시된 사진

자동 생성된 설명

**Names:**

**202254013 강은빈**

**202154026 남현서**

**202154027 양수연**

**202254031 임수빈**

**202254012 임하나**

**Date: June, 15, 2024(V1.0.0)**

# **Table of Contents**

[**Software Design Specification i**](#_Toc169357168)

[**Table of Contents ii**](#_Toc169357169)

[**List of Figures iv**](#_Toc169357170)

[**1. Introduction 1**](#_Toc169357171)

[**1.1 Purpose 1**](#_Toc169357172)

[**1.2 Scope 1**](#_Toc169357173)

[**1.3 Overview 1**](#_Toc169357174)

[**1.4 References 1**](#_Toc169357175)

[**1.5 Definitions and Acronyms 2**](#_Toc169357176)

[**2. System Architecture 3**](#_Toc169357177)

[**2.1 System Concept 3**](#_Toc169357178)

[**2.2 System Structure 4**](#_Toc169357179)

[**2.2.1 I/O Component 4**](#_Toc169357180)

[**2.2.2 Natural Language Processing Module 4**](#_Toc169357181)

[**2.2.3 Bus Route Component 4**](#_Toc169357182)

[**2.2.4 Bus Route Database 4**](#_Toc169357183)

[**2.2.5 Hardware Component 4**](#_Toc169357184)

[**2.3 System Configuration 5**](#_Toc169357185)

[**2.3.1 I/O Component 5**](#_Toc169357186)

[**2.3.2 Natural Language Processing Module 5**](#_Toc169357187)

[**2.3.3 Bus Route Component 6**](#_Toc169357188)

[**2.3.4 Bus Route Database 6**](#_Toc169357189)

[**2.3.5 Hardware Component 8**](#_Toc169357190)

[**3. System Design 9**](#_Toc169357191)

[**3.1 Design 9**](#_Toc169357192)

[**3.1.1 I/O Component Design 9**](#_Toc169357193)

[**3.1.2 Natural Language Processing Module Design 9**](#_Toc169357194)

[**3.1.3 Bus Route Component Design 9**](#_Toc169357195)

[**3.1.3.1 Google API 클라이언트 설정 9**](#_Toc169357196)

[**3.1.3.2 목적지 주소의 좌표 변환 9**](#_Toc169357197)

[**3.1.3.3 주변 정류장 검색 10**](#_Toc169357198)

[**3.1.3.4 버스 정류장을 지나는 노선 검색 10**](#_Toc169357199)

[**3.1.3.5 노선 정보 반환 10**](#_Toc169357200)

[**3.1.4 Bus Route Database Design 10**](#_Toc169357201)

[**3.1.4.1 API 연동 10**](#_Toc169357202)

[**3.1.4.2 API 설계 10**](#_Toc169357203)

[**3.1.4.3 요청 및 응답 형식 10**](#_Toc169357204)

[**3.1.4.4 공공데이터를 활용한 데이터베이스 설계 11**](#_Toc169357205)

[***3.1.4.4.1* *데이터 파일 분석* 12**](#_Toc169357206)

[***3.1.4.4.2* *데이터베이스 테이블 설계* 12**](#_Toc169357207)

[***3.1.4.4.3* *데이터 삽입* 12**](#_Toc169357208)

[***3.1.4.4.4* *데이터 무결성 및 정확성 확인* 12**](#_Toc169357209)

[**3.1.5 Hardware Component Design 12**](#_Toc169357210)

[**3.1.5.1 입력 장치 설계 12**](#_Toc169357211)

[***3.1.5.1.1* *하드웨어 구성* 13**](#_Toc169357212)

[***3.1.5.1.2* *입력 데이터 처리 방식* 13**](#_Toc169357213)

[***3.1.5.1.2* *사용자 인터페이스* 13**](#_Toc169357214)

[***3.1.5.5* *전원 요구 사항* 13**](#_Toc169357215)

[**3.1.5.2 출력 장치 설계 13**](#_Toc169357216)

[***3.1.5.2.1* *하드웨어 구성* 13**](#_Toc169357217)

[***3.1.5.2.2* *출력 데이터 처리 방식* 14**](#_Toc169357218)

[***3.1.5.2.3* *사용자 인터페이스* 14**](#_Toc169357219)

[***3.1.5.2.4* *전원 요구 사항* 14**](#_Toc169357220)

[**3.2 Execution Conceptual Diagram 15**](#_Toc169357221)

[**3.2.1 Activity Diagram 15**](#_Toc169357222)

[**3.2.2 Sequence Diagram 16**](#_Toc169357223)

[**3.2.3 Class Diagram 17**](#_Toc169357224)

[**3.2.3.1 사용자 입력 및 음성 인식 18**](#_Toc169357225)

[**3.2.3.2 자연어 처리 19**](#_Toc169357226)

[**3.2.3.3 버스 노선 탐색 20**](#_Toc169357227)

[**3.2.3.4 결과 생성 및 사용자 전달 22**](#_Toc169357228)

[**3.3 Interface Design 24**](#_Toc169357229)

[**3.3.1 Definition 24**](#_Toc169357230)

[**3.4 Restrictions 25**](#_Toc169357231)

[**4. Appendices 26**](#_Toc169357232)

[**A. Hardware Design 26**](#_Toc169357233)

[**A.1 Appearance 26**](#_Toc169357234)

[**A.2 Circuit Diagram 27**](#_Toc169357235)

[**A.2.1 전원 공급 27**](#_Toc169357236)

[**A.2.2 기타 고려사항 28**](#_Toc169357237)

[**B. Expected Code Examples 29**](#_Toc169357238)

[**B.1 자연어 처리 모듈 29**](#_Toc169357239)

[**B.2 Raspberry Pi와 Push Button 연결 31**](#_Toc169357240)

# **List of Figures**

[**그림 1 – 시스템 구조도 3**](#_Toc169357274)

[**그림 2 – 자연어 처리 모듈 데이터 흐름 다이어그램 9**](#_Toc169357275)

[**그림 3 – 액티비티 다이어그램 15**](#_Toc169357276)

[**그림 4 – 시퀀스 다이어그램 16**](#_Toc169357277)

[**그림 5 – 전체 시스템 클래스 다이어그램 17**](#_Toc169357278)

[**그림 6 – 사용자 입력 및 음성 인식 기능 클래스 다이어그램 18**](#_Toc169357279)

[**그림 7 – 자연어 처리 클래스 다이어그램 19**](#_Toc169357280)

[**그림 8 - 버스 노선 탐색 클래스 다이어그램 20**](#_Toc169357281)

[**그림 9 - 결과 생성 및 사용자 전달 클래스 다이어그램 22**](#_Toc169357282)

[**그림 10 – 제품 외관 구상도 26**](#_Toc169357283)

[**그림 11 – 버스 정류장에 설치된 Omnibus 상상도 26**](#_Toc169357284)

[**그림 12 – 하드웨어 회로도 27**](#_Toc169357285)

# 

# **1. Introduction**

## **1.1 Purpose**

본 소프트웨어 설계 명세서는 SW 융합캡스톤디자인 수업의 일환인 정보취약계층의 대중교통정보 접근성 향상을 위한 음성기반 버스노선 안내 시스템 개발 프로젝트(이하 Omnibus)의 전반적인 시스템 설계를 설명한다.

## **1.2 Scope**

Omnibus는 스마트폰 지도 어플리케이션이나 스마트폰 그 자체를 이용하는 것에 어려움을 겪는 이른바 정보취약계층의 정보 접근성을 높이기 위해 제작된 음성 기반의 버스노선 안내 작업을 수행하는 임베디드 시스템이다. 사용자는 버스정류장마다 설치된 Omibus 기기를 사용해 원하는 목적지로 향하는 버스가 해당 정류장을 지나는지, 그렇다면 어떤 버스를 이용하면 되는지와 같이 간단한 길찾기 정보를 획득할 수 있다. 이러한 과정은 간단한 외관과 음성 기반의 작업 방식을 채택함으로써 더욱 용이하게끔 설계되었다.

## **1.3 Overview**

이 문서는 전체적인 시스템 구조도, 시스템 구성요소, 액티비티 다이어그램, 시퀀스 다이어그램, 클래스 다이어그램 등을 명세하고 있다. 1장은 프로젝트와 문서의 목적, 범위, 참고문헌, 용어 및 약어 등을 설명한다. 2장은 전체적인 시스템 구조에서 시작해 각 모듈 별 세부 설계에 이르기까지 점강적으로 설명하였다. 3장은 구체적인 설계 내용을 모듈별로 작성했다. 임베디드 시스템을 제작하는 프로젝트 특성상 인터페이스 설계에 대한 내용은 3장의 시스템 설계와 4장의 부록에 나누어 수록하였으며, 하드웨어 제작에 대한 세부적인 설계 내용 역시 두 장에 나누어 작성하였다. 마지막 장인 부록에는 최종 결과물인 제품의 외관 상상도와 하드웨어 회로도, 단순한 수준의 예상 코드를 수록하였다.

## **1.4 References**

박현진. (2019년 01월03일). 인공지능(AI) 언어모델 ‘BERT(버트)'는 무엇인가. 인공지능신문. https://www.aitimes.kr/news/articleView.html?idxno=13117

## **1.5 Definitions and Acronyms**

|  |  |
| --- | --- |
| **I2C** | 라즈베리 파이에서 LCD 디스플레이를 사용하기 위해서 제공되는 스크린의 일종 |
| **BERT** | Bidirectional Encoder Representations from Transformers; 자연어 처리에서 사용되는 사전 학습된 언어 모델 |
| **토큰화** | 텍스트를 더 작은 단위(단어, 형태소)로 분리하는 과정 |
| **품사 태깅** | 각 토큰에 품사(동사, 명사 등)를 할당하는 과정 |
| **개체 인식** | 텍스트에서 사람, 장소, 기관 등의 명사를 식별하는 과정 |
| **형태소** | 뜻을 가진 가장 작은 말의 단위 |
| **Komoran** | 한국어 텍스트를 분석하기 위한 형태소 분석기 |
| **Places API** | Google Maps의 Places API는 사용자가 특정 위치나 장소에 대한 정보를 검색할 수 있게 해주는 서비스 |
| **Geocoding API** | 주소를 지리적 좌표로 변환하거나 그 반대의 과정을 수행해주는 Google Maps의 API |

# **2. System Architecture**

## **2.1 System Concept**

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

그림 1 – 시스템 구조도

Omnnibus의 시스템 구조는 그림 1과 같다. 시스템은 사용자와 직접적으로 상호작용하는 하드웨어 컴포넌트와 내부 작업을 수행하는 내부 시스템 애플리케이션으로 구성한다. 애플리케이션은 입출력 컴포넌트, 자연어 처리 프로세스, 버스 노선 탐색 프로세스를 포함하고, 세부적으로 입출력 컴포넌트는 STT와 TTS, 그리고 결과 표시를 포함한다. 애플리케이션은 작업 수행 과정에서 데이터베이스의 실시간 버스 노선 데이터를 사용한다.

## **2.2 System Structure**

시스템의 기능에 따른 구성 단위는 다음과 같다.

### **2.2.1 I/O Component**

입출력 컴포넌트는 음성 인식과 음성 합성 기술을 결합하여서 사용자와 시스템 간의 상호작용을 돕는다. 사용자의 음성을 인식하고 컴퓨터가 이해할 수 있는 텍스트로 변환하는 음성 인식 기술과 텍스트 정보를 받아서 사용자에게 음성으로 다시 출력해줄 수 있는 변환 기술인 음성 합성 기술이 주로 사용된다. 또한 LCD 디스플레이를 통해 작업 결과를 표시하여 시각적인 정보를 함께 제공한다.

### **2.2.2 Natural Language Processing Module**

자연어 처리 모듈은 BERT 모델을 사용하여 텍스트의 의도를 분류하고, 한국어 형태소 분석기를 통해 명사-목적지-를 추출한다. 해당 모듈의 주요 목표는 사용자의 입력에서 의도와 목적지를 정확하게 인식하는 것이다.

### **2.2.3 Bus Route Component**

버스 노선 컴포넌트는 자연어 처리 모듈을 통해 파악한 사용자 목적지 정보로 향하는 버스가 해당 정류장을 지나는지에 대한 여부를 파악한 다음, 해당되는 노선이 있는 경우 버스 번호를 모두 찾아 사용자에게 안내할 수 있게끔 하는 길 찾기 작업을 수행한다.

### **2.2.4 Bus Route Database**

각각의 정류장을 지나는 버스 노선 정보는 버스 노선 데이터베이스를 통해 정리되고 관리되며, 2.3.3 버스 노선 컴포넌트에서 길 찾기 작업을 수행할 때에 사용된다.

### **2.2.5 Hardware Component**

하드웨어 컴포넌트는 사용자와 직접적으로 상호작용하는 인터페이스로서 버튼을 이용한 사용자 입력 인식, 마이크와 스피커를 이용한 음성 인식 및 안내 기능 수행, LCD 화면을 통한 결과 표현을 수행한다. 2.3.1~4의 소프트웨어 애플리케이션은 기기 내부의 Raspberry Pi 4 Model B를 통해 임베디드 된 상태로, 준비가 완료된 기기는 버스 정류소의 안내 표지판과 가까운 장소에 미리 설치되어 작업을 수행한다.

## **2.3 System Configuration**

위(2.2)에서 설명한 시스템을 이루는 기능들의 자세한 구성요소는 아래와 같다.

### **2.3.1 I/O Component**

**i) STT**

Google Speech-to-Text API를 통해 STT기능을 수행한다.

음성 데이터 캡처: 마이크를 통해서 사용자의 음성을 캡처한다.

음성 인식: 음성 데이터를 텍스트로 변환한다.

텍스트 출력: 인식된 텍스트를 출력하여 자연어 모듈로 전송한다.

**ii) TTS**

Google Text-to-Speech API를 통해 TTS 기능을 수행한다.

음성 합성: 만들어진 텍스트 문장을 토대로 출력할 음성을 생성한다.

음성 출력: 음성을 스피커로 출력시킨다.

**iii) 화면(LCD) 표시**

라즈베리 파이의 I2C 인터페이스를 사용하여 LCD에 텍스트를 전송한다.

### **Natural Language Processing Module**

**i) 모델 및 토크나이저 초기화**

BertForSequenceClassification과 BertTokenizer를 사용하여 BERT 모델과 토크나이저를 초기화한다.

**ii) 텍스트 전처리**

입력된 텍스트를 형태소 단위로 분할하고, 각 형태소에 품사를 태깅하여 개체를 인식한다. 이 과정에는 konlpy 라이브러리의 Komoran 형태소 분석기가 사용된다.

**iii) 의도 인식 및 목적지 추출**

전처리된 텍스트를 BERT 모델에 입력하여 의도를 분류하고 형태소 분석 결과를 통해 목적지를 추출한다.

### **2.3.3 Bus Route Component**

**i) 검색 알고리즘**

사용자가 입력한 목적지에 대해 SQL 쿼리를 실행하여 해당 목적지와 일치하는 버스 노선 데이터를 검색한다. 이 과정에서 테이블 간의 조인을 통해 관련 정보를 통합하고, 효율적인 인덱싱을 통해 검색 속도를 최적화한다. 검색된 결과는 정렬 및 필터링 과정을 거쳐 사용자에게 반환된다.

**ii) 응답 생성**

검색된 데이터를 기반으로 사용자에게 반환할 JSON 또는 XML 포맷의 응답을 생성한다. 검색 결과를 구조화된 데이터로 변환하고, 직렬화 과정을 거친다.

### **2.3.4 Bus Route Database**

**i) 테이블**

Buses

* + bus\_id: VARCHAR, PRIMARY KEY
  + bus\_number: INT, UNIQUE
  + start\_point: VARCHAR(100)
  + end\_point: VARCHAR(100)

Stops

* + stop\_id: VARCHAR, PRIMARY KEY
  + stop\_name: VARCHAR(100)

BusRoutes

* + bus\_route\_id: INT, PRIMARY KEY
  + bus\_id: INT, FOREIGN KEY REFERENCES buses(bus\_id)
  + route\_id: INT, FOREIGN KEY REFERENCES routes(route\_id)

RouteStops

* + route\_stop\_id: INT, PRIMARY KEY
  + route\_id: INT, FOREIGN KEY REFERENCES routes(route\_id)
  + stop\_id: INT, FOREIGN KEY REFERENCES stops(stop\_id)
  + stop\_order: INT

**ii) 키**

기본 키 (Primary Keys)

* + Buses(bus\_id)
  + Stops(stop\_id)
  + BusRoutes(bus\_route\_id)
  + RouteStops(route\_stop\_id)

외래 키 (Foreign Keys)

* + BusRoutes(bus\_id) → Buses(bus\_id)
  + BusRoutes(route\_id) → Routes(route\_id)
  + RouteStops(route\_id) → Routes(route\_id)
  + RouteStops(stop\_id) → Stops(stop\_id)

**iii) 인덱스**

Buses 테이블

* + bus\_number 인덱스

Stops 테이블

* + stop\_name 인덱스
  + latitude와 longitude 인덱스 (공간 인덱스)

BusRoutes 테이블

* + bus\_id 인덱스
  + route\_id 인덱스

RouteStops 테이블

* + route\_id 인덱스
  + stop\_id 인덱스

**iv) 관계**

Buses ⟶ BusRoutes (One-to-Many)

* + Buses: bus\_id (PK)
  + BusRoutes: bus\_route\_id (PK), bus\_id (FK)

Routes ⟶ BusRoutes (One-to-Many)

* + Routes: route\_id (PK)
  + BusRoutes: bus\_route\_id (PK), route\_id (FK)

Routes ⟶ RouteStops (One-to-Many)

* + Routes: route\_id (PK)
  + RouteStops: route\_stop\_id (PK), route\_id (FK)

Stops ⟶ RouteStops (One-to-Many)

* + Stops: stop\_id (PK)
  + RouteStops: route\_stop\_id (PK), stop\_id (FK)

**a. 관계 요약**

* + Buses ⟶ BusRoutes: One-to-Many (bus\_id)
  + Routes ⟶ BusRoutes: One-to-Many (route\_id)
  + Routes ⟶ RouteStops: One-to-Many (route\_id)
  + Stops ⟶ RouteStops: One-to-Many (stop\_id)

### **2.3.5 Hardware Component**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Product** | **Model** | **Function** |
| **Input** | Push Button | TACT Switch | 기기를 시작하기 위한 사용자의 입력을 감지한다. |
| Microphone | USB Microphone | 사용자의 음성 데이터를 Raspberry Pi로 전송한다. |
| **Output** | Speaker | USB Speaker | 시스템의 음성 출력을 사용자에게 전달한다. |
| LCD | 70x93mm I2C LCD Display | 텍스트 데이터를 시각적으로 표시한다. |
| **Other** | Raspberry Pi | Raspberry Pi 4 Model B | 시스템의 중앙 제어 장치 역할을 하며, 모든 입출력 장치를 관리하고 제어한다. STT 및 TTS API, 노선 찾기 알고리즘과의 통신을 처리하여 음성 인식 및 출력, 결과 출력의 역할을 수행한다. |
| Bread Board | Half Size | Raspberry Pi와 장치를 연결할 때 별도의 납땜 과정을 거치지 않아도 연결이 가능하게 하는 역할을 한다. |

# **3. System Design**

## **3.1 Design**

### **3.1.1 I/O Component Design**

음성 입출력 기능을 구현하기 위해 사용할 기술인 STT와 TTS는 Google 클라우드에서 제공하는 API를 사용한다. 먼저 구글 클라우드 플랫폼에서 프로젝트를 생성한 다음, API를 활성화한다. 다음으로 서비스 계정에서 JSON 형식의 키 파일을 다운로드한다. 마지막으로 파이썬 환경에 필요한 라이브러리를 설치하여 STT와 TTS를 구현한다.

### **3.1.2 Natural Language Processing Module Design**

자연어 처리 모듈은 BERT 모델과 Komoran 분석기를 사용하여 설계한다. 그림 2의 다이어그램은 자연어 처리 모듈 안에서의 데이터 흐름을 보여준다.

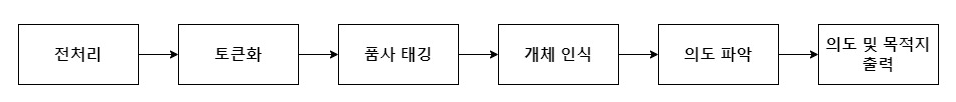


그림 2 – 자연어 처리 모듈 데이터 흐름 다이어그램

### **3.1.3 Bus Route Component Design**

버스 노선 컴포넌트는 Google Maps API를 사용하여 사용자가 지정한 목적지 주변의 정류장을 찾고, 해당 정류장을 지나는 버스 노선을 확인하는 작업을 수행한다. Google Maps의 Places API와 Geocoding API를 활용하여 구현한다. 다음은 순서에 따라 설계 내용을 정리한 것이다.

#### 3.1.3.1 Google API 클라이언트 설정

* 목적: Google Maps API를 사용하기 위한 클라이언트를 설정한다.
* 세부 사항: Google Cloud Platform에서 API 키를 생성하고 필요한 API(Places API, Geocoding API)를 활성화한다.

#### 3.1.3.2 목적지 주소의 좌표 변환

* 목적: 사용자가 제공한 목적지 주소를 지리적 좌표로 변환한다.
* 사용 API: Google Geocoding API

#### 3.1.3.3 주변 정류장 검색

* 목적: 변환된 좌표를 기반으로 주변의 버스 정류장을 검색한다.
* 사용 API: Google Places API

#### 3.1.3.4 버스 정류장을 지나는 노선 검색

* 목적: 검색된 버스 정류장을 지나는 모든 노선을 데이터베이스에서 검색한다.

#### 3.1.3.5 노선 정보 반환

* 목적: 사용자에게 해당 노선 정보를 반환한다.
* 출력 형식: 리스트 형태로 일치하는 노선 정보를 반환한다.

### **3.1.4 Bus Route Database Design**

#### 3.1.4.1 API 연동

버스 정보 데이터베이스는 외부 시스템과 API를 연동하여 설계된다. 아래 내용은 API의 엔드 포인트, 요청 및 응답 형식, 데이터베이스 연동 방법을 설명하고 있다.

#### 3.1.4.2 API 설계

|  |  |  |
| --- | --- | --- |
| 앤드포인트 | HTTP 메소드 | 설명 |
| /api/buses | GET | 모든 버스 목록 조회 |
| /api/buses/{bus\_id} | GET | 특정 버스 정보 조회 |
| /api/stops | GET | 모든 정류장 정보 조회 |
| /api/stops/{stop\_id} | GET | 특정 정류장 정보 조회 |
| /api/routes/{route\_id}/stops | GET | 특정 경로의 모든 정류장 목록 조회 |

#### 3.1.4.3 요청 및 응답 형식

**GET /api/buses**

* + 요청
  + URL: /api/buses
  + 메서드: GET
  + 요청 파라미터: 없음
  + 응답: 등록된 모든 버스의 목록을 반환한다.

**GET /api/buses/{bus\_id}**

* + 요청
  + URL: /api/buses/{bus\_id}
  + 메서드: GET
  + 요청 파라미터: bus\_id(필수)
  + 응답: 해당 버스의 고유 ID, 버스 번호, 운영사 정보를 포함한다.

**GET /api/stops**

* + 요청
  + URL: /api/stops
  + 메서드: GET
  + 요청 파라미터: 없음
  + 응답: 등록된 모든 정류장의 목록을 반환한다.

**GET /api/stops/{stop\_id}**

* + 요청
  + URL: /api/stops/{stop\_id}
  + 메서드: GET
  + 요청 파라미터: stop\_id(필수)
  + 응답: 해당 정류장의 고유 UD, 정류장 이름, 위도, 경도 정보를 포함한다.

**GET /api/routes/{route\_id}/stops**

* + 요청
  + URL: /api/routes/{route\_id}/stops
  + 메서드: GET
  + 요청 파라미터: route\_id(필수)
  + 응답: 해당 경로에 포함된 모든 정류장의 목록을 반환한다.

#### 3.1.4.4 공공데이터를 활용한 데이터베이스 설계

다음 내용은 공공 데이터를 활용하여 버스 정류장 위치 정보를 데이터베이스에 포함시키는 과정과 설계를 설명한 것이다. 주요 단계는 데이터 파일 분석, 데이터베이스 테이블 설계, 데이터 삽입, 데이터 무결성 및 정확성 확인으로 구성된다. 사용할 파일은 공공 데이터 포털에서 제공하는 ‘서울특별시\_버스 정류소 좌표 데이터.csv’이다.

##### ***3.1.4.4.1 데이터 파일 분석***

주어진 CSV 파일을 분석하여 주요 필드를 파악한다. 파일의 인코딩 문제를 해결하고 데이터 구조를 확인한 결과, 다음과 같은 필드가 포함되어 있다.

- STDR\_DE: 데이터 기준 일자

- NODE\_ID: 정류장 고유 번호

- STTN\_NO: 정류장 번호

- STTN\_NM: 정류장 이름

- CRDNT\_X: 경도 (longitude)

- CRDNT\_Y: 위도 (latitude)

- STTN\_TY: 정류장 유형 (일반, 종점)

##### ***3.1.4.4.2 데이터베이스 테이블 설계***

데이터 파일 분석으로 설계된 필드를 바탕으로 SQLAIchemy와 연계된 코드를 구현한다.

##### ***3.1.4.4.3 데이터 삽입***

CSV 파일의 데이터를 데이터베이스에 삽입하는 과정은 다음과 같다.

1. SQLAIchemy 설정: 데이터베이스 엔진 및 세션을 설정한다.

2. 데이터 삽입: CSV 파일의 각 행을 데이터베이스에 삽입한다.

3. 변경 사항 커밋: 데이터베이스에 변경 사항을 커밋한다.

##### ***3.1.4.4.4 데이터 무결성 및 정확성 확인***

데이터 삽입 후 데이터의 무결성과 정확성을 검증한다. 이를 위해 간단한 SQL 쿼리를 사용하여 데이터가 예상대로 삽입되었는지 확인한다.

### **3.1.5 Hardware Component Design**

#### 3.1.5.1 입력 장치 설계

본 프로젝트에서 사용하는 입력 장치는 Push Button과 Microphone이다.

* Push Button: 사용자가 기기의 사용을 원할 때 누르는 버튼으로, 기기의 시작을 알리는 입력 장치이다.
* Microphone: 사용자가 원하는 목적지를 말하면 그 음성을 입력 받는 기능을 한다.

##### ***3.1.5.1.1 하드웨어 구성***

Push Button은 Bread Board를 통해 GPIO 핀과 연결하고, Microphone은 USB 포트를 통해 Raspberry Pi와 연결된다.

##### ***3.1.5.1.2 입력 데이터 처리 방식***

* Push Button : GPIO 입력을 통해 입력(눌림)이 확인되면 Speaker를 통해 미리 지정된 안내 음성을 출력한다.
* Microphone: 사용자로부터 음성 데이터를 입력 받는다.

##### ***3.1.5.1.2 사용자 인터페이스***

사용자가 Push Button을 누르면 Speaker에서 “목적지를 말해주세요” 하는 음성 안내 문구가 출력되고, Microphone으로는 사용자의 음성을 입력 받을 준비가 완료된다.

사용자가 Microphone에 자신의 목적지를 말하면 LCD를 통해 처리중임을 나타내는 표시가 출력되고, 사용자의 목적지를 기반으로 처리한 버스 노선 검색 결과가 LCD와 Speaker를 통해 시각, 청각적 정보로 출력된다.

##### ***3.1.5.5 전원 요구 사항***

* Push Button: Raspberry Pi의 GPIO 전원 사용한다.
* Microphone: Raspberry Pi의 USB 전원을 사용한다.(5V)

#### 3.1.5.2 출력 장치 설계

본 프로젝트에서 사용하는 출력 장치는 LCD와 Speaker이다.

* LCD: 텍스트 및 그래픽, 간단한 문구를 시각적으로 표시한다. 본 프로젝트에서는 사용자의 음성을 입력 받는 중이나 버스 노선 계산 프로그램이 동작 중일 때 처리중임을 알리는 모션을 표시하고, 처리 완료된 버스 번호를 출력하는 기능을 한다.
* Speaker: 음성을 출력한다. 본 프로젝트에서는 사용자가 버튼을 눌렀을 때 음성 안내를 출력하고, 버스 노선 처리가 완료되었을 때 음성으로 출력하는 기능을 한다.

##### ***3.1.5.2.1 하드웨어 구성***

* LCD: LCD 모듈과 GPIO 핀을 각 기능에 맞게 연결한다.
* Speaker: Raspberry Pi의 USB 포트를 이용해 연결한다.

##### ***3.1.5.2.2 출력 데이터 처리 방식***

* LED Matrix: Raspberry Pi에서 지정하는 텍스트나 그래픽 데이터를 출력한다.
* Speaker: Raspberry Pi에서 생성된 오디오 신호를 받아 출력한다.

##### ***3.1.5.2.3 사용자 인터페이스***

* LCD: 사용자 요청 처리 중이라는 상태 메세지와 버스 노선 검색 결과를 텍스트로 표시한다.
* Speaker: 기기 안내 음성과 버스 노선 안내 음성을 출력한다.

##### ***3.1.5.2.4 전원 요구 사항***

* LCD: Raspberry Pi 전원을 공급받아 사용한다.
* Speaker: Raspberry Pi의 USB 전원을 사용한다.(5V)

## **3.2 Execution Conceptual Diagram**

### **3.2.1 Activity Diagram**

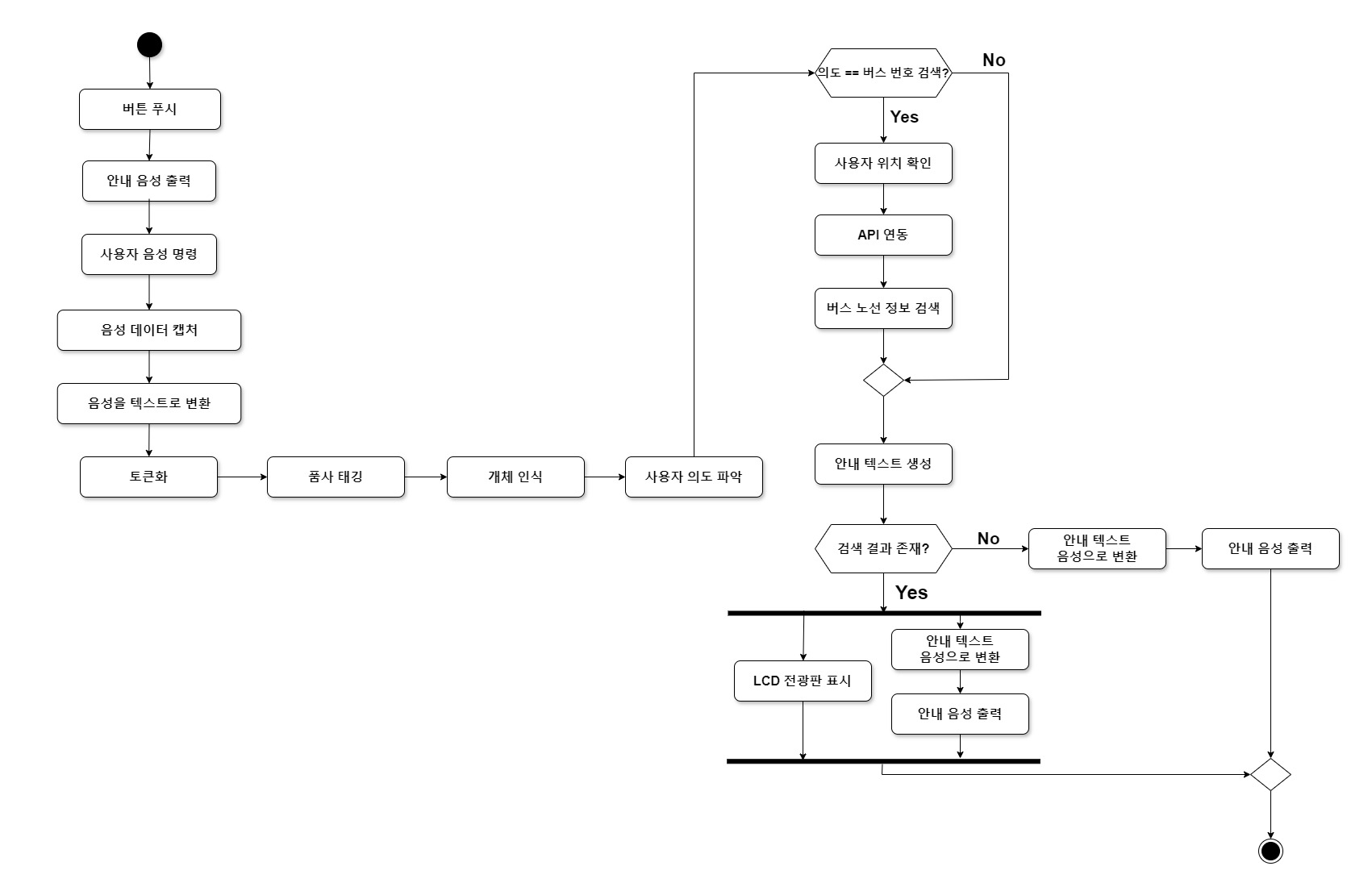


그림 3 – 액티비티 다이어그램

### **3.2.2 Sequence Diagram**

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

그림 4 – 시퀀스 다이어그램

### **3.2.3 Class Diagram**

텍스트, 도표, 평면도, 평행이(가) 표시된 사진

자동 생성된 설명

그림 5 – 전체 시스템 클래스 다이어그램

#### 3.2.3.1 사용자 입력 및 음성 인식

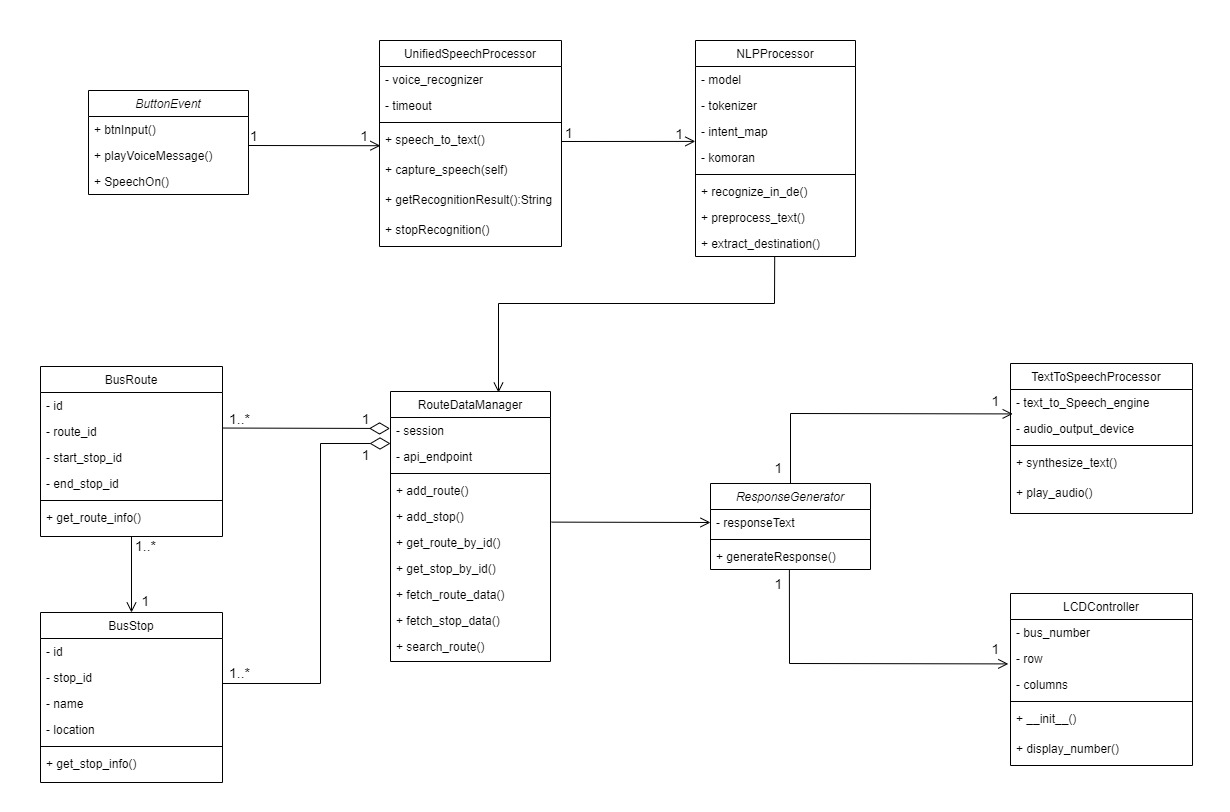


그림 6 – 사용자 입력 및 음성 인식 기능 클래스 다이어그램

**ButtonEvent**

**메소드**

**+ btnInput():** 버튼 입력을 감지하는 메소드이다.

**+ playVoiceMessage():** 안내 음성을 재생하는 메소드이다.

**+ SpeechOn():** 사용자의 음성을 입력 받는 메소드이다.

**UnifiedSpeechProcessor**

**속성**

**- voice\_recognizer:** 음성 캡처와 음성 인식을 하는 변수이다.

**- timeout: Int:** 음성 입력이 일정 시간이 지났을 때 종료하는 변수이다

**메소드**

**+ speech\_to\_text:** 음성을 텍스트로 변환하는 메소드이다.

**+ capture\_speech(self):** 마이크로부터 음성을 캡처하여 텍스트로 변환한다.

**+ getRecognitionResult():String:** 인식된 텍스트를 반환한다.

**+ stopRecognition():** 인식 중지를 위한 자리 표시자 메소드이다.

#### 3.2.3.2 자연어 처리

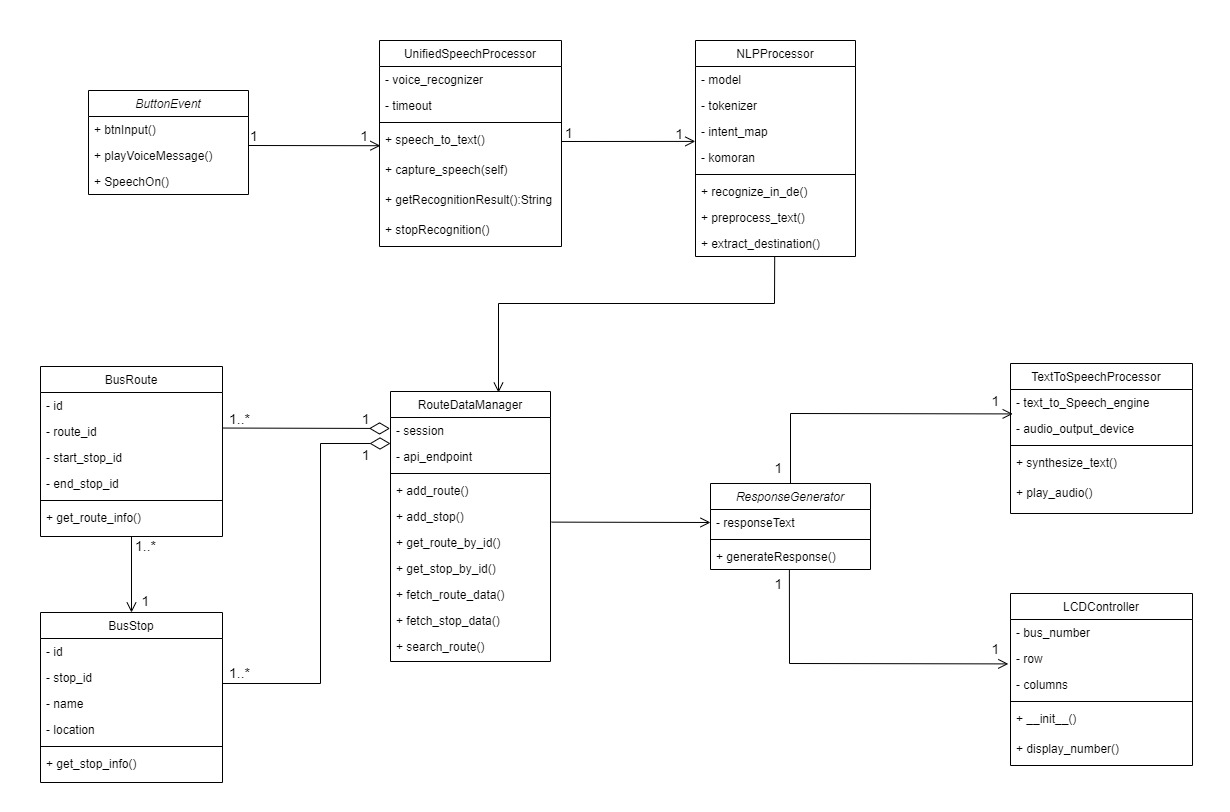


그림 7 – 자연어 처리 클래스 다이어그램

**NLPProcessor**

**속성**

**- model:** BERT 모델을 사용한 시퀀스 분류기 객체이다.

**- tokenizer:** BERT 토크나이저 객체이다.

**- intent\_map:** 의도를 매핑하는 딕셔너리이다.

**- komoran:** 형태소 분석기 개체이다.

**메소드**

**+ recognize\_in\_de():** 입력 문자열을 받아 텍스트를 전처리하고, 의도와 개체를 인식하여 목적지를 반환하는 메소드이다.

**+ preprocess\_text():** 원본 텍스트를 전처리하여 전처리된 텍스트를 반환하는 메소드이다.

**+ extract\_destination():** 개체 인식을 통해 원본 텍스트와 형태소 분석 결과를 입력 받아 목적지를 추출하고 반환한다.

#### 3.2.3.3 버스 노선 탐색

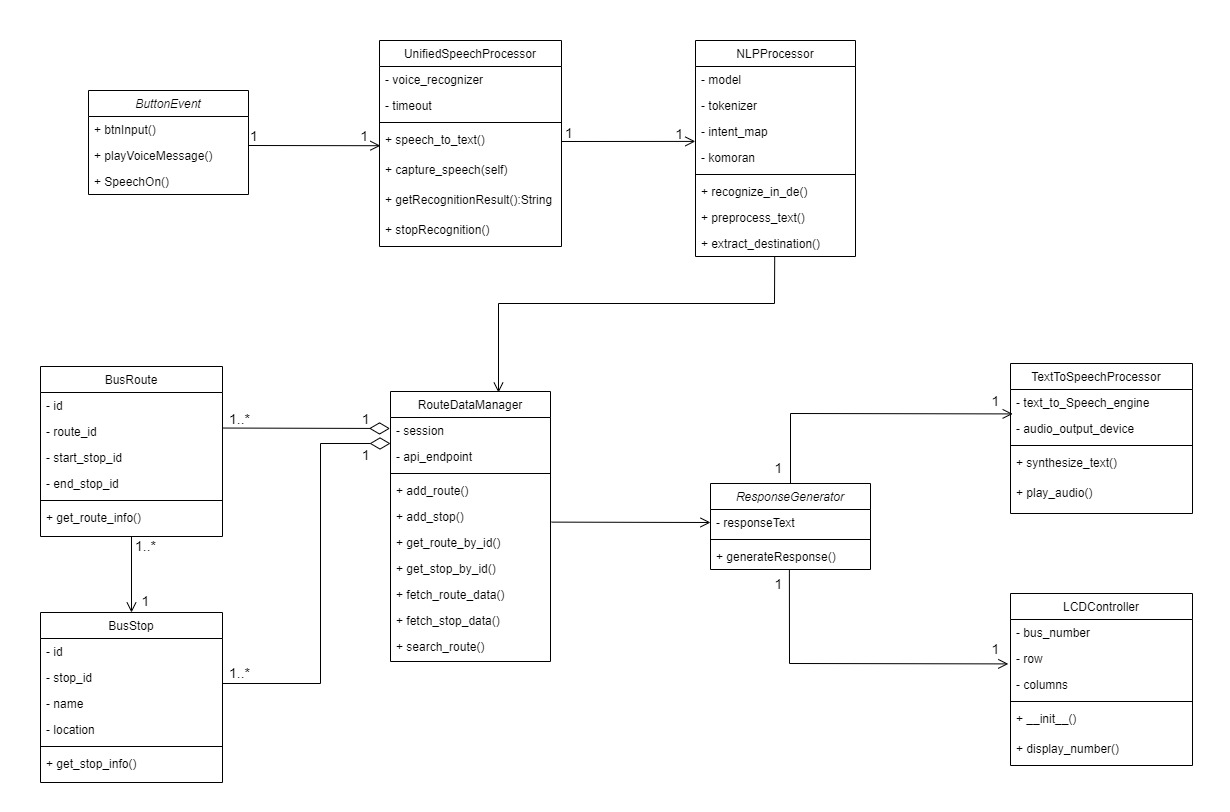


그림 8 - 버스 노선 탐색 클래스 다이어그램

**RouteDataManager**

**속성**

**- session:** SQLAIchemy 세션 객체로 데이터베이스와의 상호작용을 관리한다.

**- api\_endpoint:** 외부 API의 엔드포인트 URL을 저장한다.

**메소드**

**+ add\_route(route\_id: String, start\_stop\_id: String, end\_stop\_id: String): None:** 주어진 경로 ID와 시작 및 종료 정류장 ID를 사용하여 새 버스 경로를 데이터베이스에 추가한다.

**+ add\_stop(stop\_id: String, name: String, location: String): None:** 주어진 정류장 ID, 이름, 위치를 사용하여 새 버스 정류장을 데이터베이스에 추가한다.

**+ get\_route\_by\_id(route\_id: String): BusRoute:** 주어진 경로 ID로 버스 경로를 데이터베이스에서 검색하여 반환한다.

**+ get\_stop\_by\_id(stop\_id: String): BusStop:** 주어진 정류장 ID로 버스 정류장을 데이터베이스에서 검색하여 반환한다.

**+ fetch\_route\_data(): List[BusRoute]:** 외부 API를 호출하여 버스 경로 데이터를 가져오고, 이를 BusRoute 객체의 리스트로 반환한다.

**+ fetch\_stop\_data(): List[BusStop]:** 외부 API를 호출하여 버스 정류장 데이터를 가져오고, 이를 BusStop 객체의 리스트로 반환한다.

**+ search\_route(start\_stop\_id, end\_stop\_id):** 주어진 시작 정류장 ID와 종료 정류장 ID에 맞는 노선을 검색한다.

**BusStop**

**속성**

**- id:** 데이터베이스에서 사용되는 고유 식별자 (정수형)이다.

**- stop\_id:** 정류장의 고유 식별자 (문자열)이다.

**- name:** 정류장의 이름이다.

**- location:** 정류장의 위치다.

**메소드**

**+ get\_stop\_info(): str:** 정류장의 정보를 문자열로 반환한다.

**BusRoute**

**속성**

**- id:** 데이터베이스에서 사용되는 고유 식별자 (정수형)이다.

**- route\_id:** 경로의 고유 식별자 (문자열)이다.

**- start\_stop\_id:** 시작 정류장의 ID (정수형, 외래 키로 BusStop 참조)

**- end\_stop\_id:** 종료 정류장의 ID (정수형, 외래 키로 BusStop 참조)

**메소드**

**+ get\_route\_info(): str:** 경로의 정보를 문자열로 반환한다.

#### 3.2.3.4 결과 생성 및 사용자 전달

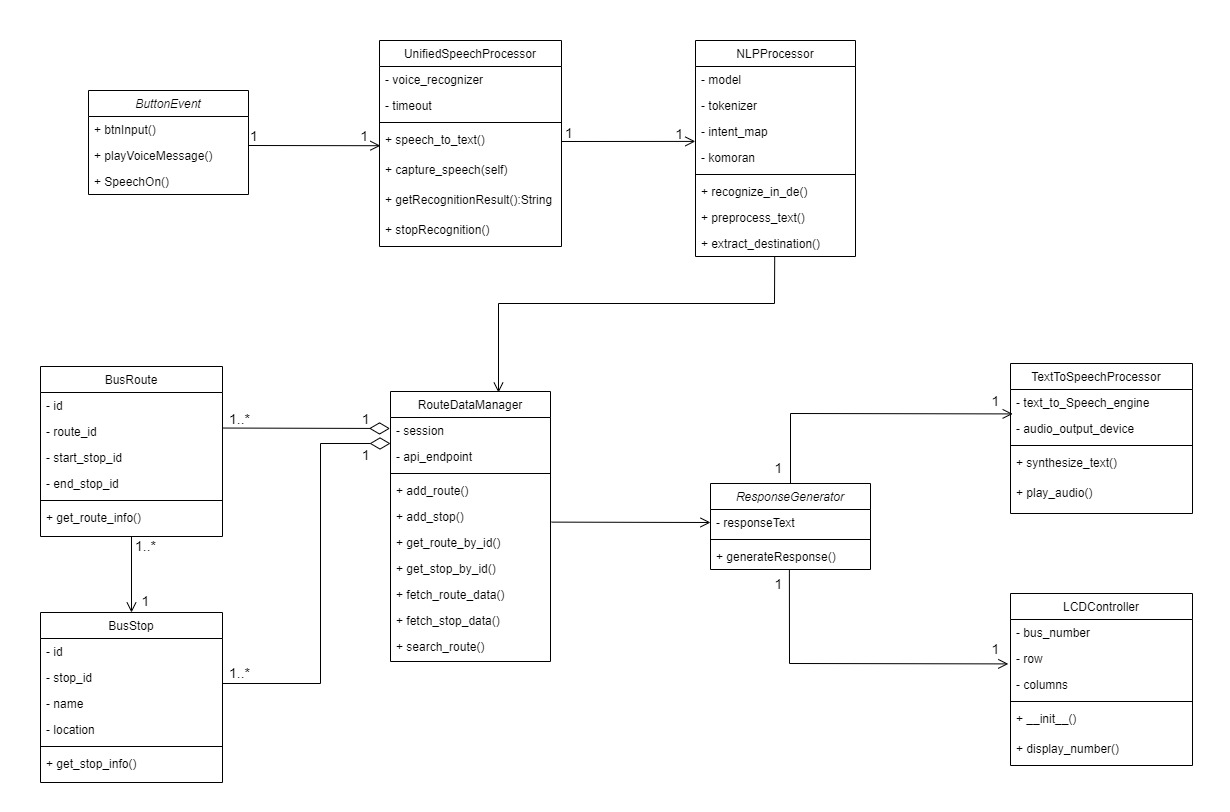


그림 9 - 결과 생성 및 사용자 전달 클래스 다이어그램

**ResponseGenerator**

**속성**

**- responseText: string:** 사용자에게 전달할 경로 안내 텍스트를 저장한다.

**메소드**

**+ generateResponse():** 탐색된 경로를 기반으로 응답 텍스트를 생성하는 메소드이다.

**TextToSpeechProcesseor**

**속성**

**- text\_to\_speech\_engine:** 텍스트를 음성 데이터로 변환한다.

**- audio\_output\_device:** 음성을 스피커로 출력시키는 변수이다.

**메소드**

**+ synthesize\_text(self, text):** 텍스트를 음성 데이터로 변환하는 메소드이다.

**+ play\_audio(self, audio\_data):** 음성을 출력시키는 메소드이다.

**LCDController**

**속성**

**- bus\_number:** 받아온 데이터 경로 숫자를 화면에 표시하는 변수이다.

**- row:** 행을 저장한다.

**- columns:** 열을 저장한다.

**메소드**

**+ \_\_init\_\_(self, columns, rows):** 생성자 메소드이다. LCD 모듈의 초기 설정을 수행한다.

**+ display\_number(self, number, row, col):** 지정된 위치에 숫자를 출력시킨다.

## **3.3 Interface Design**

Omnibus는 임베디드 시스템으로써 물리적 기기를 사용자 인터페이스로 사용하고 있으므로, 본 3.4장에서는 각 하드웨어와 소프트웨어 모듈 간의 인터페이스를 정의하는 수준으로 내용을 작성하였다. 하드웨어 설계에 대한 설명은 3.1.5장을, 설계도에 대한 자세한 내용은 4장의 부록 중 A장을 참고하라.

### **3.3.1 Definition**

다음은 하드웨어와 소프트웨어 간의 인터페이스를 정의한 것이다.

**1) Push Button과 Raspberry Pi**

※ Bread Board에 Push Botton을 꽂아준 후 Bread Board와 GPIO를 연결선으로 연결한다.

|  |  |
| --- | --- |
| Interface | GPIO 핀 |
| 설명 | Push Button은 Raspberry Pi의 GPIO 핀과 연결되어 있어, 사용자가 버튼을 누르면 GPIO 핀에 신호가 전달된다. Raspberry Pi는 이 신호를 감지하여 소프트웨어로 전달한다. |
| 데이터 흐름 | 버튼을 누름 → Push Button → GPIO Signal → 이벤트 감지 |

**2) Microphone과 Raspberry Pi**

|  |  |
| --- | --- |
| Interface | USB 포트 |
| 설명 | 마이크는 USB 포트를 통해 Raspberry Pi에 연결된다. 사용자의 음성 데이터를 Raspberry Pi로 전송한다. |
| 데이터 흐름 | Microphone → 음성 입력 → Voic Data 전송 |

**3) Speaker와 Rasberry Pi**

|  |  |
| --- | --- |
| Interface | USB 포트 |
| 설명 | Speaker는 USB 포트를 통해 Raspberry Pi에 연결한다. Raspberry Pi는 TTS API로부터 받은 음성 데이터를 스피커로 전송해 출력한다. |
| 데이터 흐름 | TTS API → Raspberry Pi → Audio Data 전송 → Speaker → Audio 출력 |

**4) LCD Display와 Raspberry Pi**

※ 연결선을 이용하여 GPIO와 LCD의 I2C버스를 연결한다.

|  |  |
| --- | --- |
| Interface | I2C 버스 |
| 설명 | LCD Display는 I2C 버스를 통해 Raspberry Pi와 연결된다. Raspberry Pi는 버스 노선 정보를 텍스트 데이터로 LCD Display에 전송하여 출력한다. |
| 데이터 흐름 | 버스 노선 계산 프로그램 → text data → I2C 신호 → LCD Display 화면 출력 |

## **3.4 Restrictions**

프로젝트의 결과물이 될 임베디드 시스템의 하드웨어 장치는 미니어처 크기(원본의 약 0.4배)로 제작한다. 이 과정에서 부품 구매 및 제작에 투입되는 비용은 최대 15만원을 넘어서는 안 된다. 모든 구현 작업은 10월 말일까지 완료되어야 하며, 테스팅을 포함해 최종적으로 제작이 완료되는 시점은 11월말까지로 한다.

# **4. Appendices**

## **A. Hardware Design**

### **A.1 Appearance**

제품의 외관은 아래의 그림 10 및 11과 같다. 자세한 설명은 요구명세서를 참고하라.

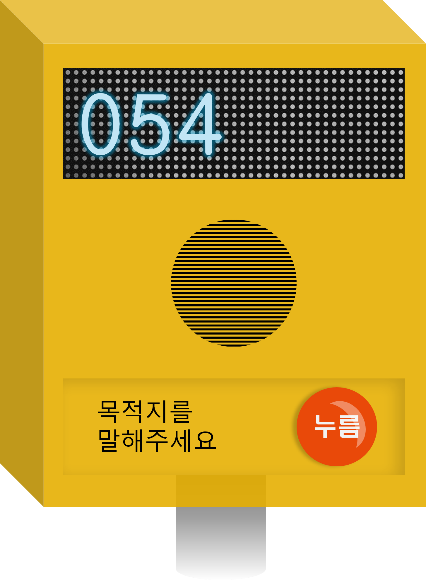


그림 10 – 제품 외관 구상도



그림 11 – 버스 정류장에 설치된 Omnibus 상상도

### **A.2 Circuit Diagram**

그림 12는 하드웨어의 소자 구성 및 배치를 표현한 회로도다.

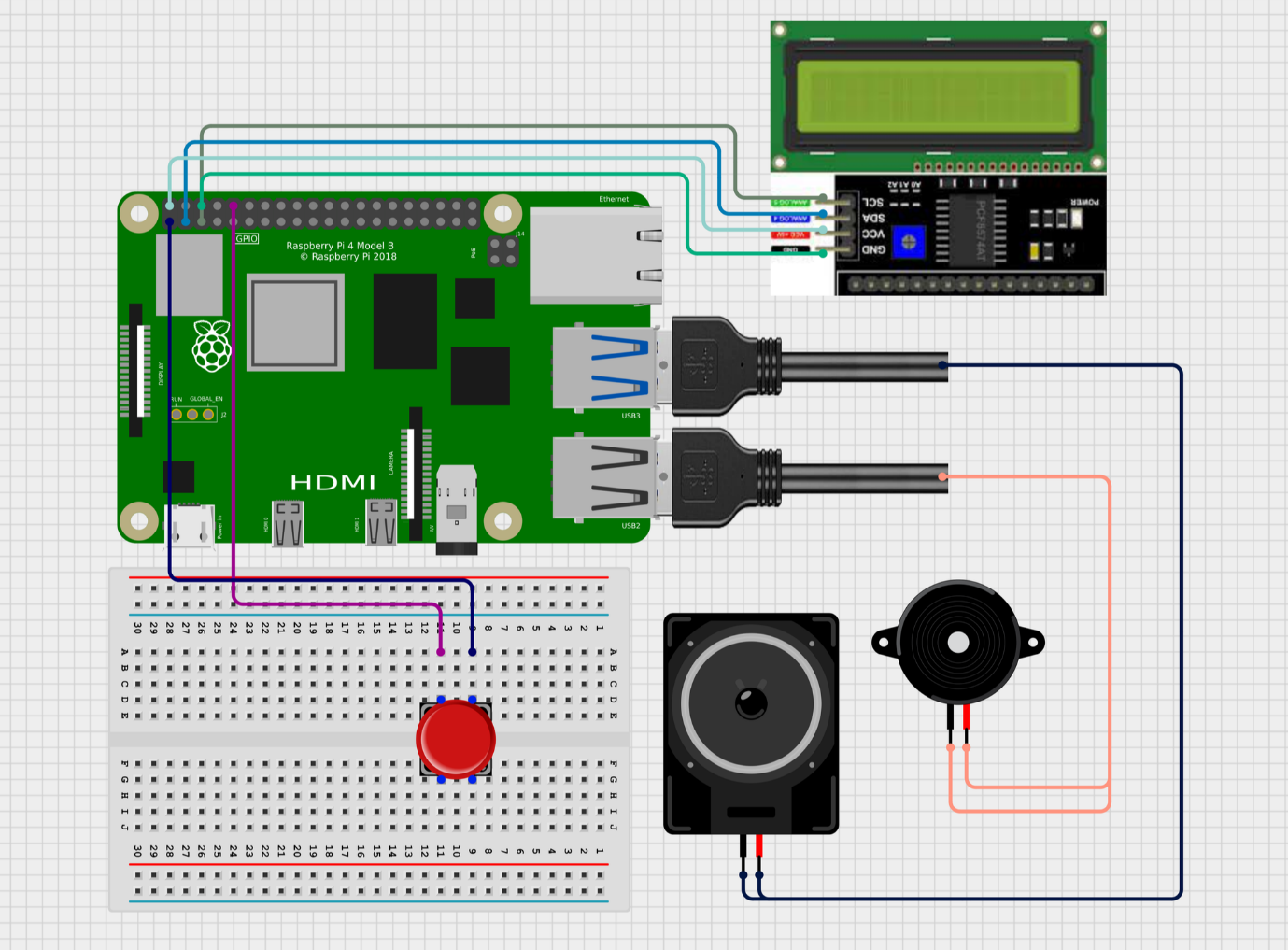


그림 12 – 하드웨어 회로도

* **Push Button**: GPIO의 1번핀(3V3 power)과 10번핀(15(RXD))을 각각 버튼을 꽂은 라인과 맞춰 연결한다
* **LCD:** GPIO의 2,3,5,6번 핀과 LCD 모듈을 같은 기능을 하는 핀끼리 연결한다.
* **Microphone, Speaker:** USB 포트를 사용하여 연결한다.

#### A.2.1 전원 공급

* **Raspberry Pi**: 5V-3A 전원 어댑터를 이용하여 전원을 공급한다.
* **LCD**: Raspberry Pi에서 전원을 공급받는다.
* **Microphone, Speaker**: Raspberry Pi의 USB 포트에서 전원을 공급 받기 때문에 별도의 전원이 필요하지 않다.
* **Push Button**: GPIO에서 전원을 공급 받기 때문에 별도의 전원이 필요하지 않다.

#### A.2.2 기타 고려사항

Raspberry Pi의 과열을 방지하기 위해 방열판을 부착한다.

## **B. Expected Code Examples**

### **B.1 자연어 처리 모듈**

|  |
| --- |
| !pip install konlpy  from transformers import BertTokenizer, BertForSequenceClassification  import torch  import re  from konlpy.tag import Komoran  class IntentDetector:  def \_\_init\_\_(self):  self.model = BertForSequenceClassification.from\_pretrained('monologg/kobert')  self.tokenizer = BertTokenizer.from\_pretrained('monologg/kobert')  self.intent\_map = {  0: '버스 번호 검색',  1: '노선 없음'  }  self.komoran = Komoran()  def recognize\_intent\_and\_destination(self, text):  input\_text = self.preprocess\_text(text)  # 한국어 형태소 분석을 통한 토큰화 및 품사 태깅  tokens = self.komoran.pos(input\_text)  print("토큰화 및 품사 태깅 결과:", tokens)  input\_ids = torch.tensor([self.tokenizer.encode(input\_text, add\_special\_tokens=True)])  output = self.model(input\_ids)[0]  intent\_id = torch.argmax(output, dim=1).item()  # 개체 인식 결과가 있을 때는 무조건 '버스 번호 검색'으로 설정  if any(pos == 'NNP' for \_, pos in tokens):  intent\_id = 0  intent = self.intent\_map[intent\_id]  # 개체 인식을 통한 목적지 추출  destination = self.extract\_destination(input\_text, tokens)  # 목적지가 없는 경우에는 노선 없음으로 설정  if not destination:  intent = '노선 없음'  return intent, destination  def preprocess\_text(self, text):  text = text.replace('가려고', '을 가려고')  text = text.replace('가고 싶어', '을 가고 싶어')  text = text.replace('가요', '을 가요')  text = text.replace('갈래', '을 갈래')  text = text.replace('가야 돼', '을 가야 돼')  return text  def extract\_destination(self, text, tokens):  destination = None  # 개체 인식을 통해 목적지 추출  for token, pos in tokens:  if pos == 'NNP':  destination = token  break  return destination  intent\_detector = IntentDetector()  text1 = "저녁에 용산 가려고 하는데 몇 번 타야돼?"  text2 = "강남역 가는 노선 알려줘"  text3 = "경복궁 가고 싶어"  text4 = "서울역 갈래"  text5 = "배고프다 배고프고 졸리고 집에 가고 싶어"  intent1, destination1 = intent\_detector.recognize\_intent\_and\_destination(text1)  intent2, destination2 = intent\_detector.recognize\_intent\_and\_destination(text2)  intent3, destination3 = intent\_detector.recognize\_intent\_and\_destination(text3)  intent4, destination4 = intent\_detector.recognize\_intent\_and\_destination(text4)  intent5, destination5 = intent\_detector.recognize\_intent\_and\_destination(text5)  print("text1 - 의도:", intent1, ", 목적지:", destination1)  print("text2 - 의도:", intent2, ", 목적지:", destination2)  print("text3 - 의도:", intent3, ", 목적지:", destination3)  print("text4 - 의도:", intent4, ", 목적지:", destination4)  print("text5 - 의도:", intent5, ", 목적지:", destination5) |

### **B.2 Raspberry Pi와 Push Button 연결**

|  |
| --- |
| import RPi.GPIO as GPIO  import time  # GPIO 핀 번호 설정  BUTTON\_PIN\_1 = 1  BUTTON\_PIN\_2 = 10  # GPIO 설정  GPIO.setmode(GPIO.BCM) # BCM 모드 설정  GPIO.setup(BUTTON\_PIN\_1, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP) # 내부 풀업 저항 사용  GPIO.setup(BUTTON\_PIN\_2, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP) # 내부 풀업 저항 사용  # 버튼 콜백 함수  def button\_callback\_1(channel):  if GPIO.input(BUTTON\_PIN\_1) == GPIO.LOW:  print("Button 1 Pressed")  def button\_callback\_2(channel):  if GPIO.input(BUTTON\_PIN\_2) == GPIO.LOW:  print("Button 2 Pressed")  # 버튼 이벤트 설정 (falling edge 검출)  GPIO.add\_event\_detect(BUTTON\_PIN\_1, GPIO.FALLING, callback=button\_callback\_1, bouncetime=200)  GPIO.add\_event\_detect(BUTTON\_PIN\_2, GPIO.FALLING, callback=button\_callback\_2, bouncetime=200)  try:  print("Press the buttons")  while True:  # 메인 루프를 계속 실행  time.sleep(1)  except KeyboardInterrupt:  print("Program terminated")  finally:  GPIO.cleanup() # GPIO 설정 해제 |