문서관리 번호		비밀구분
본 문서는 임의로 복사되거나 배포될 수 없음		대외비 (비등록)
배포번호		

안심하이 기술개발문서

2024.11.

안심하이

수정일	내용	작성자
2024.11.04	초안 작성	이준학

목차

1 개요

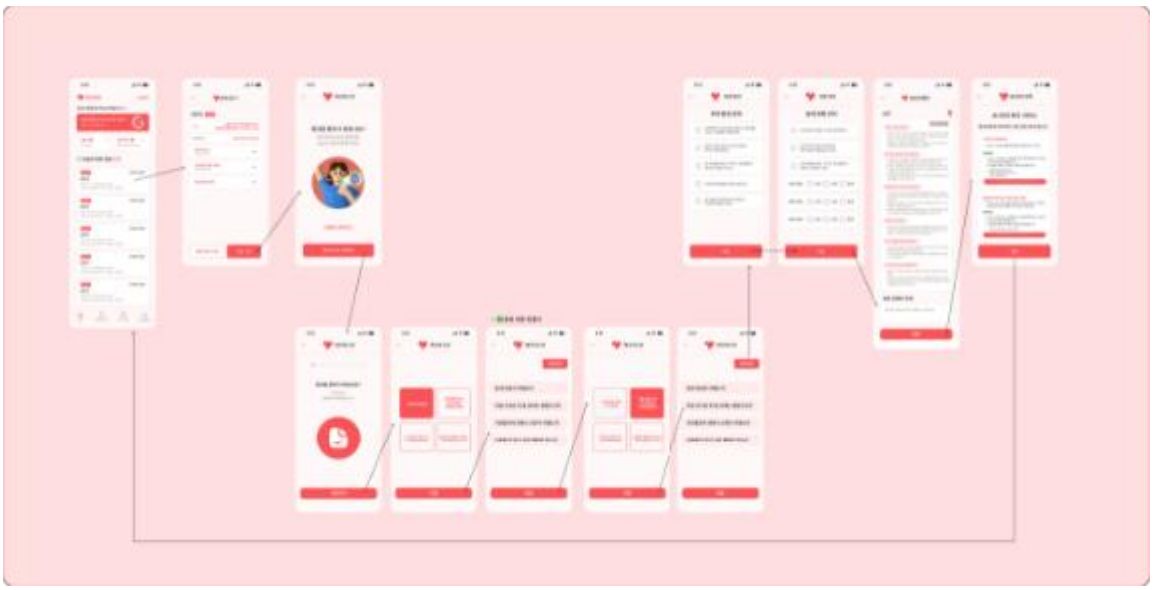
1.1 안심하이 서비스 소개

안심하이의 시스템은 아래와 같은 목표를 가지고있는 App과 Web 형태의 프로그램입니다. 현장의 돌봄인력들이 전문적인 지식없이도 전문 상담사가 제공할 수 있는 높은 수준의 상담이 가능할 수 있도록 도와줍니다.

AI 핵심기능인 실시간 상담 음성 변환, 상담 질문 내용 생성, 복지정책추천등의 기능을 제공합니다.

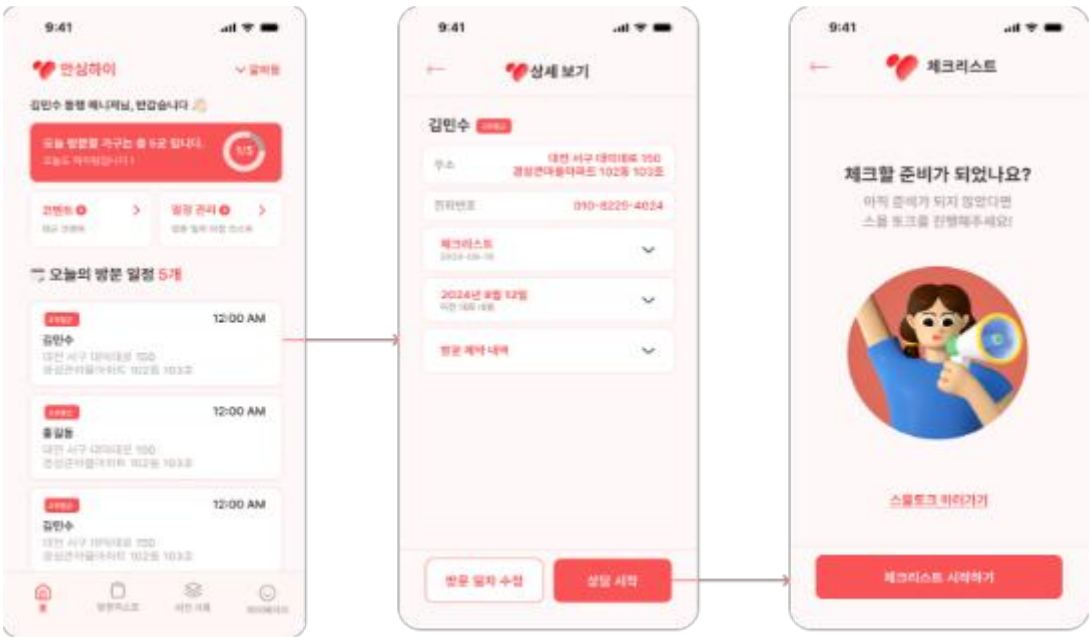
1.2 안심하이 1차개발 와이어 프레임

안심하이의 핵심 시나리오 구현을 위한 1차개발 와이어프레임은 아래와 같으며 내담자 선택과 그에 따른 추천 질문 생성, 실시간 음성변환 및 상담내용 요약, 복지정책 추천기능과 같은 안심하이 핵심 기능이 담겨있습니다.



1.2.1 방문일정(내담자) 선택기능

현장 메인 페이지(좌측)에서 원하는 방문일정과 내담자 정보를 선택하면 방문정보 상세보기 페이지(중간)로 이동하며 내담자의 정보와 이전의 상담기록 등을 확인할 수 있습니다. 이후 '상담시작' 버튼을 누르면 체크리스트 준비 페이지(우측)으로 이동하며 체크리스트 작성을 시작합니다.



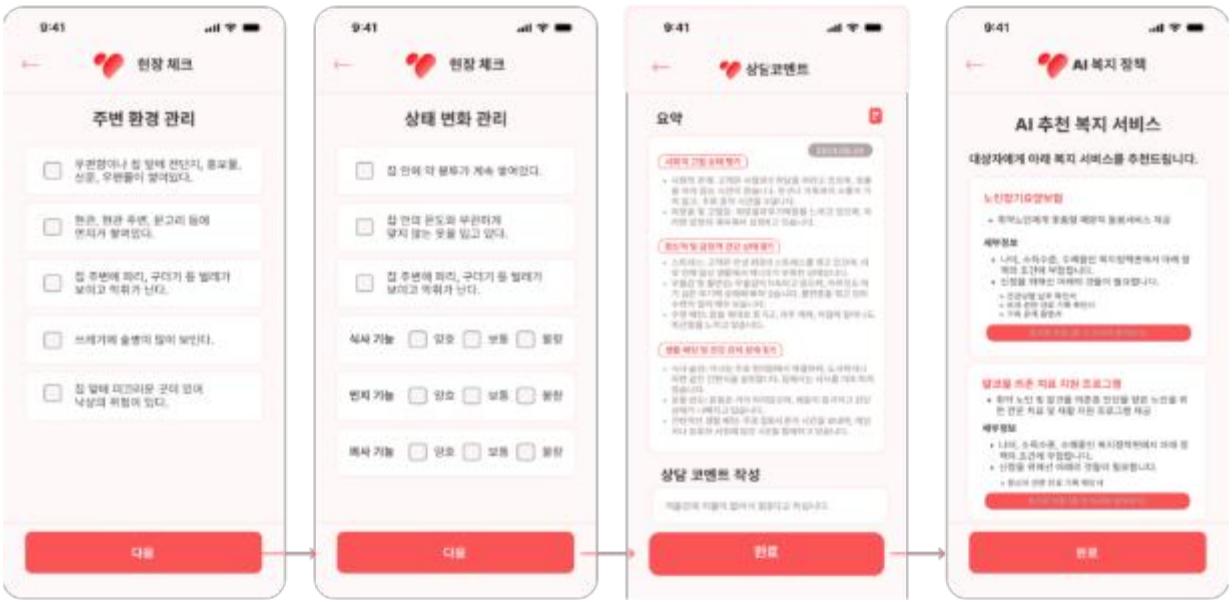
1.2.2 상담 진행중 맞춤 질문 추천 및 체크리스트 선택기능

'체크리스트 시작하기' 버튼을 클릭한 후 상담이 시작되면 아래와 같은 화면이 반복되며 내담자의 상황에 맞는 체크리스트를 선택하고 이후의 체크리스트 선택을 위한 맞춤 질문이 자동으로 생성됩니다. 이때 STT 실시간 음성변환기능을 이용합니다. 상담자는 아래와 같이 유도되는 질문들과 체크리스트 선택을 통해 자연스럽게 상담을 이끌고 고독사위험군 판단을 위한 체크리스트를 작성할 수 있습니다.

1.2.3 상담 완료 후 기능

상담이 완료되면 '주변 환경 관리', '상태 변화 관리', '상담 코멘트 및 요약', '복지 정책 추천서비스' 화면을 확인할 수 있습니다. 주변 환경이나 상태 변화가 있었는지 확인하여 정부에서 제시한 고독사 위험군 해당 여부를 추후에 판단할 수 있도록 합니다.

이때 상담 코멘트 화면에서 AI가 요약한 상담 요약본과 추천 복지 정책 서비스들을 확인할 수 있습니다. 상담자는 특별한 코멘트가 있다면 작성하고 맞춤 복지정책을 내담자와 함께 확인하면 됩니다.



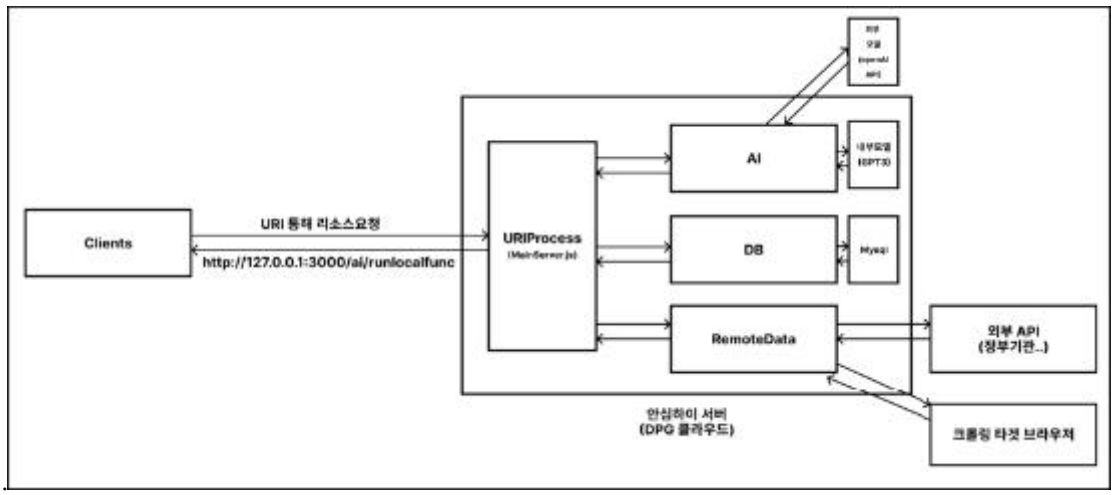
1.3 안심하이 시스템 역할 및 책임

현재 App UI(프론트엔드)과 서버 및 DB(백엔드) 그리고 핵심기능 제공을 위한 모델(AI)을 개발진행중이며 아래와 같은 목표를 가지고 있습니다.

구분	역할 및 책임
프론트엔드	App 및 Web 사용자 UX/UI 구현, 서버와의 통신 엔드포인트 구현
백엔드	DB 및 API 관리 (외부 데이터 처리 / AI모듈 / 프론트엔드 클라이언트 간 통신)
AI	1. 실시간 상담 음성 변환 (STT) 2. 상담 대화 의도 자동 분류기 3. 상담 기록 텍스트 요약 4. 실시간 대화 내용 기반 추후 질문 생성 5. 복지 정책 추천 시스템
데이터 처리	복지 정책 수집 및 관련 데이터 DB 설계

1.4 안심하이 서비스 구조

안심하이의 시스템은 1.3에서 소개한 역할 및 책임을 실현하기위해 아래와 같은 구조를 설계했습니다.



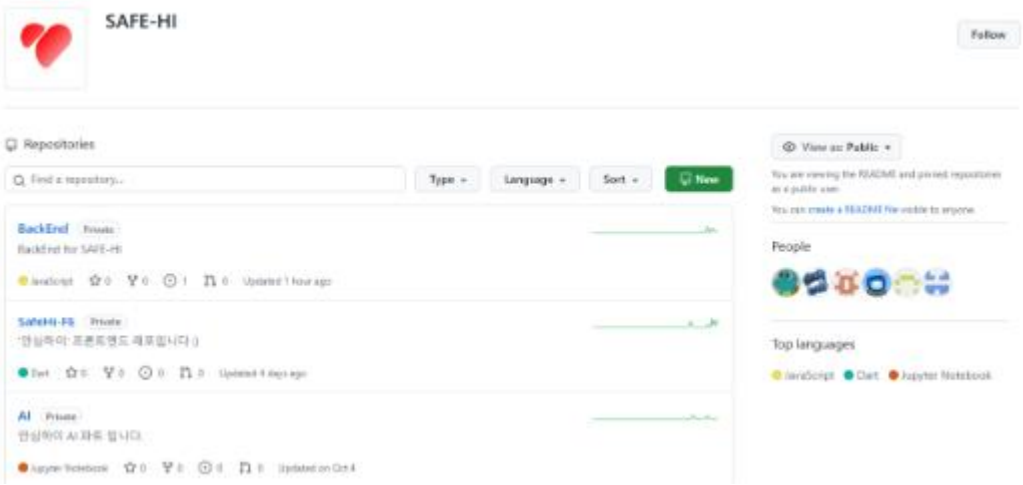
위 설계의 핵심은 RestAPI 형식을 차용하여 다양한 언어와 플랫폼 하에서 개발될 App/Web 클라이언트와 서버간의 데이터 통신을 특정 언어와 플랫폼에 구애받지않고 실현하는 것에 있습니다.

안심하이의 메인 서버에는 URIProcess 기능을 통해 인터넷에 공개되어있는 안심하이 서버에 URI 접근을 통해 내부 리소스에 접근할 수 있도록하고 이로써 언어와 플랫폼의 정보를 최소화하는 데이터 추상화를 실현합니다.

1.4.1 소스코드 구조

안심하이 소스코드는 오픈소스로 공개되어있으며 버전관리도구인 Github를 통해 인터넷에 공개되어있습니다. AI, 데이터베이스(DB), 및 원격 데이터(Remote Data) 모듈과 통신하는 Node.js 서버로 구성되어 있습니다. 안심하이의 Git 저장소는 아래 링크에서 확인가능합니다.

https://github.com/SAFE-HI



또한 안심하이 시스템의 핵심 기능을 담고있는 소스코드 디렉터리 구조는 아래와 같습니다.

```
├── ReadMe.md
├── AI
│   ├── ai_api.js
│   └── funcs
│       ├── call_ai_func_using_localfunc.js
│       ├── call_ai_func_using_web.js
│       └── index.js
├── DB
│   ├── db_api.js
│   └── funcs
│       └── db_query_app.js
├── REMOTEDATA
│   ├── funcs
│   │   └── ex_IKU.js
│   └── remote data api.js
├── Tools
│   ├── client
│   │   └── test_client
│   │       └── index.html
│   └── example_python_func
│       └── helloworld.py
├── index.js
├── MainServer.js
└── URIProcess
    ├── AI_URI_Process.js
    ├── DB_URI_Process.js
    └── RD_URI_Process.js
```

각각의 기능은 URIProcess/에서 정의된 라우팅에 따라 요청을 처리하며, 모듈별(AI_URI_Process / DB_URI_Process / RD_URI_Process)로 별도의 라우팅 파일이 분리되어 관리됩니다.

2 백엔드 서버 모듈

2.1 모듈별 역할 및 책임

아래는 각 모듈(.js)에 대한 역할과 책임을 소개합니다.

MainServer.js

이 파일은 서버의 진입점(entry point)입니다. Express.js를 사용하여 AI, DB, Remote Data 모듈에 대한 요청을 처리하는 기본 설정이 포함되어 있습니다. 좀 더 구체적으로 클라이언트의 접속요청을 받아주고 약속된 라우트 경로에 따라 지정된 요청에 따른 동작을 수행하고 결과를 반환해줍니다.

```
// 라우팅 파일 불러오기
const aiRoutes = require('./URIProcess/AI_URI_Process');
const dbRoutes = require('./URIProcess/DB_URI_Process');
const remoteDataRoutes = require('./URIProcess/RD_URI_Process');

////////////////////////////////////////////
////////////////////// 라우팅 //////////////////////
////////////////////////////////////////////

// AI 라우팅
app.use('/ai', aiRoutes);

// DB 라우팅
app.use('/db', dbRoutes);

// Remote Data 라우팅
app.use('/remotedata', remoteDataRoutes);

app.listen(port, () => {
    console.log(`Server running at http://localhost:${port}`);
});
```

각각의 모듈에 대한 라우팅은 `URIProcess` 디렉토리에 분리되어 있습니다.

URIProcess/

이 디렉토리는 각 모듈(AI, DB, Remote Data)의 라우팅을 담당합니다. URIProcess 모듈은 인터넷에 공개된 URI를 해석하여 서버 내부 약속된 라우팅 경로로 안내하고 결국 *_api.js 에 선언되어있는 백엔드 서버 내부의 함수를 호출합니다. 아래에서 내부의 모듈들의 역할과 책임을 확인할 수 있습니다.

구분	역할 및 책임
- **AI_URI_Process.js**	AI 모듈에 대한 API 요청을 처리합니다. `ai_api.js`에서 정의된 AI 관련 함수들을 호출합니다.
DB_URI_Process.js	DB 모듈에 대한 API 요청을 처리합니다. `db_api.js`데이터베이스 관련 작업을 수행합니다.
RD_URI_Process.js	Remote Data 모듈에 대한 API 요청을 처리합니다. `rb_api.js`원격 데이터 관련 함수들을 호출합니다.

```
// 상담 코멘트 삭제
router.delete('/comments/:commentId', async (req, res) => {
    try {
        const result = await dbModule.deleteConsultationComment(req.params.commentId);
        res.json(result);
    } catch (error) {
        console.error(error);
        res.status(500).send('Error deleting consultation comment');
    }
});

// ========= Questions List =========
// 특정 question_id에 해당하는 질문 리스트를 가져오는 라우터
router.get('/questions/:questionId', async (req, res) => {
    const questionId = req.params.questionId;

    try {
        const questionData = await dbModule.getQuestionsById(questionId);

        if (!questionData) {
            return res.status(404).json({ message: 'Question not found' });
        }

        res.json(questionData);
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Error retrieving questions' });
    }
});
```

Module Directorys (AI/DB/REMOTEDATA)

AI/ DB/ REMOTEDATA/ 디렉터리 안에 해당 기능에 대한 실제 코드들이 구현되어있습니다. 아래에서 내용이 이어집니다.

AI/

이 디렉토리는 AI와 관련된 코드의 구현이 포함되어 있습니다. ai_api.js는 python으로 구현된 AI 모듈에 요청을 보내는 기능이 구현되어있습니다. 요청은 동일한 기기에 존재하는 파이썬 스크립트(local)와 원격지에서 존재하고 있는 파이썬 스크립트(remote) 두 가지로 나누어 보내는 것이 가능합니다.
funcs/에는 Remote Data 모듈에서 사용할 함수들이 저장되며 추가적인 기능을 구현할 수 있습니다.

```
const { exec } = require('child_process');

function callAiFuncUsingLocalFunc() {
    return new Promise((resolve, reject) => {
        exec('python3 ./Tools/example_python_func/helloworld.py', (error, stdout, stderr) => {
            if (error) {
                console.error(`Error executing Python script: ${error.message}`);
                return reject(error);
            }
            if (stderr) {
                console.error(`Python script error: ${stderr}`);
                return reject(stderr);
            }
            console.log(`Python script output: ${stdout}`);
            resolve(stdout);
        });
    });
}

module.exports = {
    callAiFuncUsingLocalFunc
};
```

DB/

이 디렉토리는 DB 관련된 코드들이 포함되어 있습니다. db_api.js에는 DB에 쿼리할 수 있는 로직이 포함되어 있습니다. funcs/에는 Remote Data 모듈에서 사용할 함수들이 저장되며 추가적인 기능을 구현할 수 있는 자리입니다.

서버 내부에 실행중인 MySQL DB의 테이블 CRDU 쿼리 기능이 주로 구현되어있습니다.

REMOTEDATA/

이 디렉토리는 원격 데이터 모듈과 관련된 코드들이 포함되어 있습니다. rd_api.js에는 원격 데이터와 통신하는 로직이 포함되어 있습니다. funcs/에는 Remote Data 모듈에서 사용할 함수들이 저장되며 추가적인 기능을 구현할 수 있는 자리입니다.

Tools/

이 디렉토리는 클라이언트 테스트 및 예시 Python 함수(for AI) 등이 포함되어 있습니다. Client/test_client/에는 간단한 웹 클라이언트 페이지로, Node.js 서버로 요청을 보낼 수 있는 HTML 파일(index.html)이 포함되어 있습니다.

example_python_func/에는 Python 함수를 테스트할 수 있는 예시 코드가 포함되어 있으며, helloworld.py는 간단한 "Hello World" 출력을 위한 Python 예시 스크립트입니다.

ReadMe.md

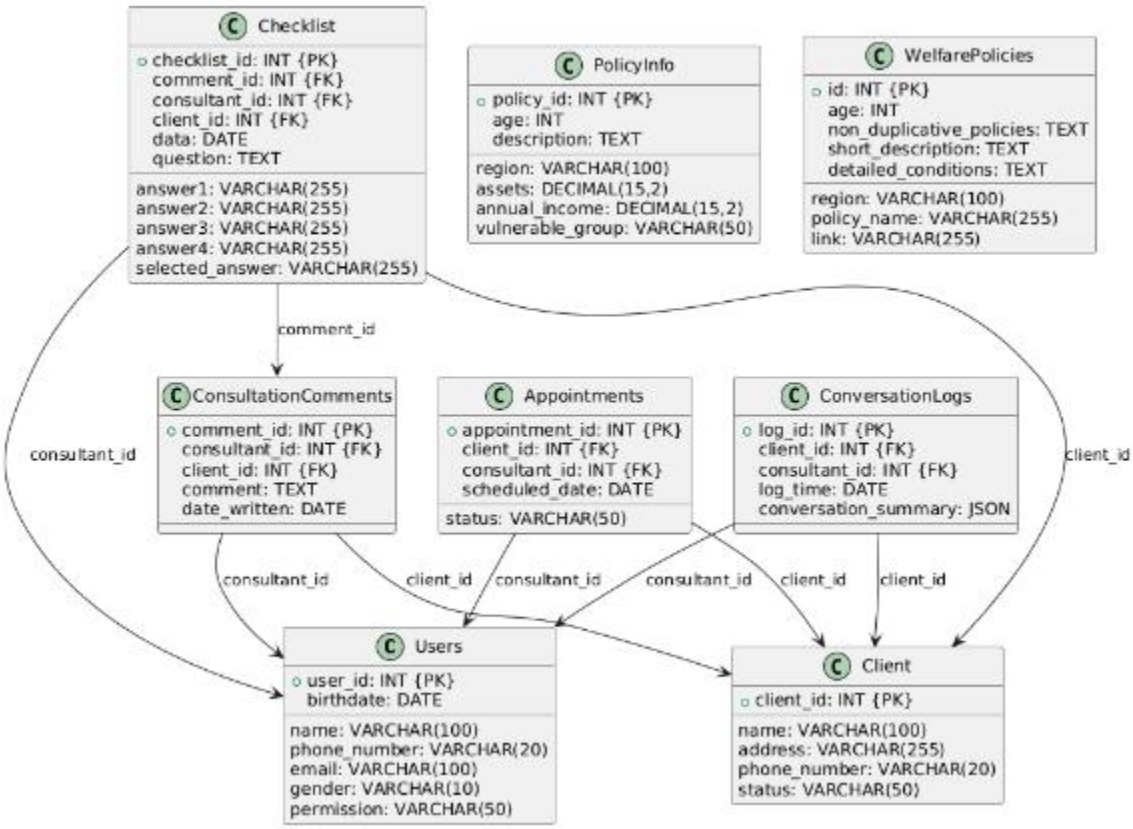
이 문서는 프로젝트 구조와 각 파일의 역할을 설명합니다. 다른 개발자들이 이 프로젝트를 빠르게 이해하고 작업을 시작할 수 있도록 가이드 역할을 합니다.

2.2 DB 설계 및 구현

아래는 DB 설계 및 구현에 대해 소개합니다.

2.2.1 DB 설계 및 UML

UML 다이어그램의 관계 각 테이블은 사용자와 클라이언트 간의 관계, 정책 정보와 체크리스트의 연결성, 상담 코멘트와 대화 로그 간의 관계 등을 포함하여 설계되었습니다.



위 UML은 아래와 같은 설계에 기반하여 starUML 문법으로 작성되었으며,
https://www.planttext.com/ 에서 작성되었습니다.

Users 테이블과 Client 테이블은 Checklist, ConsultationComments, Appointments와 같은 다른 테이블과 외래 키 관계로 연결되어 있어 데이터의 무결성을 유지합니다.

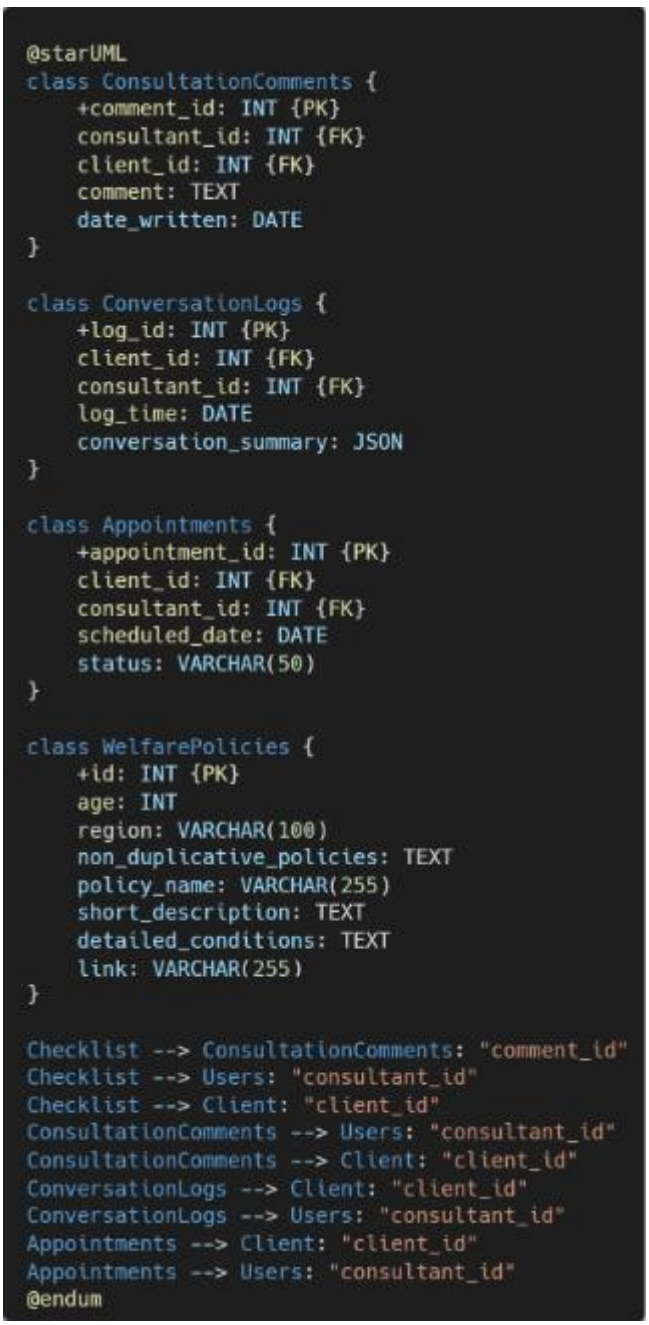
이로써 외래 키를 활용한 테이블간 JOIN을 통해 기획된 모든 기능 구현을 위한 데이터 저장 및 접근이 가능합니다.

```
@startuml
class Users {
    +user_id: INT {PK}
    name: VARCHAR(100)
    phone_number: VARCHAR(20)
    email: VARCHAR(100)
    birthdate: DATE
    gender: VARCHAR(10)
    permission: VARCHAR(50)
}

class Client {
    +client_id: INT {PK}
    name: VARCHAR(100)
    address: VARCHAR(255)
    phone_number: VARCHAR(20)
    status: VARCHAR(50)
}

class PolicyInfo {
    +policy_id: INT {PK}
    age: INT
    region: VARCHAR(100)
    assets: DECIMAL(15,2)
    annual_income: DECIMAL(15,2)
    vulnerable_group: VARCHAR(50)
    description: TEXT
}

class Checklist {
    +checklist_id: INT {PK}
    comment_id: INT {FK}
    consultant_id: INT {FK}
    client_id: INT {FK}
    data: DATE
    question: TEXT
    answer1: VARCHAR(255)
    answer2: VARCHAR(255)
    answer3: VARCHAR(255)
    answer4: VARCHAR(255)
    selected_answer: VARCHAR(255)
}
```



```
@starUML
class ConsultationComments {
    +comment_id: INT {PK}
    consultant_id: INT {FK}
    client_id: INT {FK}
    comment: TEXT
    date_written: DATE
}

class ConversationLogs {
    +log_id: INT {PK}
    client_id: INT {FK}
    consultant_id: INT {FK}
    log_time: DATE
    conversation_summary: JSON
}

class Appointments {
    +appointment_id: INT {PK}
    client_id: INT {FK}
    consultant_id: INT {FK}
    scheduled_date: DATE
    status: VARCHAR(50)
}

class WelfarePolicies {
    +id: INT {PK}
    age: INT
    region: VARCHAR(100)
    non_duplicative_policies: TEXT
    policy_name: VARCHAR(255)
    short_description: TEXT
    detailed_conditions: TEXT
    link: VARCHAR(255)
}

Checklist --> ConsultationComments: "comment_id"
Checklist --> Users: "consultant_id"
Checklist --> Client: "client_id"
ConsultationComments --> Users: "consultant_id"
ConsultationComments --> Client: "client_id"
ConversationLogs --> Client: "client_id"
ConversationLogs --> Users: "consultant_id"
Appointments --> Client: "client_id"
Appointments --> Users: "consultant_id"
@endum
```

본 데이터베이스는 사용자 관리, 클라이언트 정보, 정책 정보, 체크리스트, 상담 코멘트, 대화 로그, 약속 관리 및 복지 정책을 저장하는 테이블로 구성되어있습니다.

2.2.2 DB 구현 및 SQL Query

아래는 DB 구현을 위한 세부적인 데이터 타입과 MySQL 테이블 생성을 위한 쿼리문을 소개합니다.

Users 테이블 목적: 사용자의 기본 정보를 저장.

속성: user_id: 사용자 고유 ID (기본 키) name: 사용자 이름 phone_number: 전화번호 email: 이메일 주소 birthdate: 생년월일 gender: 성별 permission: 사용자 권한

```sql
CREATE TABLE Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,                    -- 사용자 이름
    phone_number VARCHAR(20),                      -- 전화번호
    email VARCHAR(255),                            -- 이메일
    birthdate DATE,                                -- 생년월일
    gender ENUM('Male', 'Female', 'Other'),        -- 성별 (남성, 여성, 기타)
    permission ENUM('Admin', 'User', 'Guest')      -- 권한 (관리자, 사용자, 게스트)
);
```

Client 테이블 목적: 클라이언트의 정보를 저장. 속성: client_id: 클라이언트 고유 ID (기본 키) name: 클라이언트 이름 address: 주소 phone_number: 전화번호 status: 현재 상태

```sql
CREATE TABLE Client (
    client_id INT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255),
    phone_number VARCHAR(20),
    status ENUM('High Risk', 'Active'),
    FOREIGN KEY (client_id) REFERENCES Users(user_id)
);
```

PolicyInfo 테이블 목적: 정책 정보를 저장.

속성: policy_id: 정책 고유 ID (기본 키) age: 연령 region: 지역 assets: 자산 정보 annual_income: 연간 소득 vulnerable_group: 취약 계층 정보 description: 정책 설명

```sql
CREATE TABLE PolicyInfo (
    policy_id INT PRIMARY KEY AUTO_INCREMENT,
    age INT,
    region VARCHAR(255),
    assets DECIMAL(15, 2),                                  -- 자산 (금액)
    annual_income DECIMAL(15, 2),                           -- 부부 합산 연소득
    vulnerable_group ENUM('Low Income', 'Homeless', 'Single Parent'),
    description TEXT,                                       -- 정책에 대한 설명
    FOREIGN KEY (client_id) REFERENCES Client(client_id)
);
```

Checklist 테이블 목적: 체크리스트 정보를 저장.

속성: checklist_id: 체크리스트 고유 ID (기본 키) comment_id: 연결된 코멘트 ID (외래 키) consultant_id: 컨설턴트 ID (외래 키) client_id: 클라이언트 ID (외래 키) data: 날짜 정보 question: 질문 내용 answer1, answer2, answer3, answer4: 선택 가능한 답변 selected_answer: 선택된 답변

```
CREATE TABLE Checklist (
    checklist_id INT PRIMARY KEY AUTO_INCREMENT,
    comment_id INT,                              -- 상담 코멘트 ID
    consultant_id INT,                           -- 상담사의 ID
    client_id INT,                               -- 고객의 ID
    data DATETIME,                               -- 체크리스트 작성 날짜
    question TEXT NOT NULL,                      -- 체크리스트 질문
    answer1 VARCHAR(255),                        -- 선택지 1
    answer2 VARCHAR(255),                        -- 선택지 2
    answer3 VARCHAR(255),                        -- 선택지 3
    answer4 VARCHAR(255),                        -- 선택지 4
    selected_answer ENUM('1', '2', '3', '4'),  -- 선택된 답변
    FOREIGN KEY (consultant_id) REFERENCES Users(user_id), -- 상담사 외래키
    FOREIGN KEY (client_id) REFERENCES Users(user_id),     -- 고객 외래키
    FOREIGN KEY (comment_id) REFERENCES ConsultationComments(comment_id) -- 상담 코멘트 외래키
);
```

ConsultationComments 테이블 목적: 상담 코멘트를 저장.

속성: comment_id: 코멘트 고유 ID (기본 키) consultant_id: 컨설턴트 ID (외래 키) client_id: 클라이언트 ID (외래 키) comment: 코멘트 내용 date_written: 작성 날짜

```
CREATE TABLE ConsultationComments (
    comment_id INT PRIMARY KEY AUTO_INCREMENT,
    consultant_id INT,                          -- 상담사의 ID
    client_id INT,                              -- 고객의 ID
    comment TEXT,                               -- 상담 내용 코멘트
    date_written DATETIME,                      -- 코멘트가 작성된 날짜
    FOREIGN KEY (consultant_id) REFERENCES Users(user_id), -- 상담사 외래키
    FOREIGN KEY (client_id) REFERENCES Users(user_id) -- 고객 외래키
);
```

ConversationLogs 테이블 목적:

대화 로그를 저장. 속성: log_id: 로그 고유 ID (기본 키) client_id: 클라이언트 ID (외래 키) consultant_id: 컨설턴트 ID (외래 키) log_time: 로그 시간 conversation_summary: 대화 요약 (JSON 형식)

```
CREATE TABLE ConversationLogs (
    log_id INT PRIMARY KEY AUTO_INCREMENT,
    client_id INT,                                  -- 고객의 ID
    consultant_id INT,                              -- 상담사의 ID
    log_time DATETIME,                              -- 대화가 진행된 시간
    conversation_summary TEXT,                      -- 대화 내용 요약
    FOREIGN KEY (client_id) REFERENCES Users(user_id), -- 고객 외래키
    FOREIGN KEY (consultant_id) REFERENCES Users(user_id) -- 상담사 외래키
);
```

Appointments 테이블 목적:

약속 정보를 저장. 속성: appointment_id: 약속 고유 ID (기본 키) client_id: 클라이언트 ID (외래 키) consultant_id: 컨설턴트 ID (외래 키) scheduled_date: 예약된 날짜 status: 현재 상태

```
CREATE TABLE Appointments (
    appointment_id INT PRIMARY KEY AUTO_INCREMENT,
    client_id INT,                                  -- 고객의 ID
    consultant_id INT,                              -- 상담사의 ID
    scheduled_date DATETIME,                        -- 예약된 날짜 및 시간
    status ENUM('Scheduled', 'Completed', 'Canceled'), -- 상태 (예약됨, 완료됨, 취소됨)
    FOREIGN KEY (client_id) REFERENCES Users(user_id), -- 고객 외래키
    FOREIGN KEY (consultant_id) REFERENCES Users(user_id) -- 상담사 외래키
);
```

WelfarePolicies 테이블 목적: 복지 정책 정보를 저장. 속성: id: 정책 고유 ID (기본 키) age: 연령 region: 지역 non_duplicative_policies: 중복되지 않는 정책 목록 policy_name: 정책 이름 short_description: 짧은 설명 detailed_conditions: 세부 조건 link: 관련 링크

```
CREATE TABLE WelfarePolicies (
    id INT AUTO_INCREMENT PRIMARY KEY,
    age INT,
    region VARCHAR(100),
    non_duplicative_policies TEXT,
    policy_name VARCHAR(255),
    short_description TEXT,
    detailed_conditions TEXT,
    link VARCHAR(255)
);
```

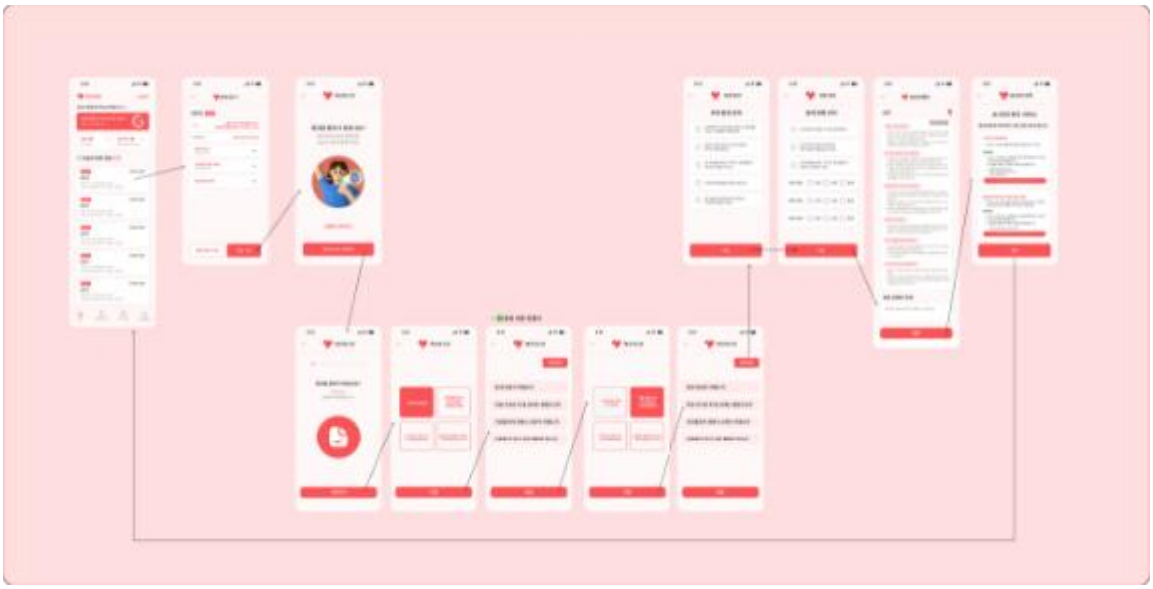
3 클라이언트

3.1 APP 클라이언트 소개

안심하이 APP 클라이언트는 크게 사용자 UI/UX와 서버와의 데이터 통신 기능이 주를 이룹니다. Flutter를 이용해 iOS 및 Android 환경에서의 배포가 가능하도록 설계하였습니다.

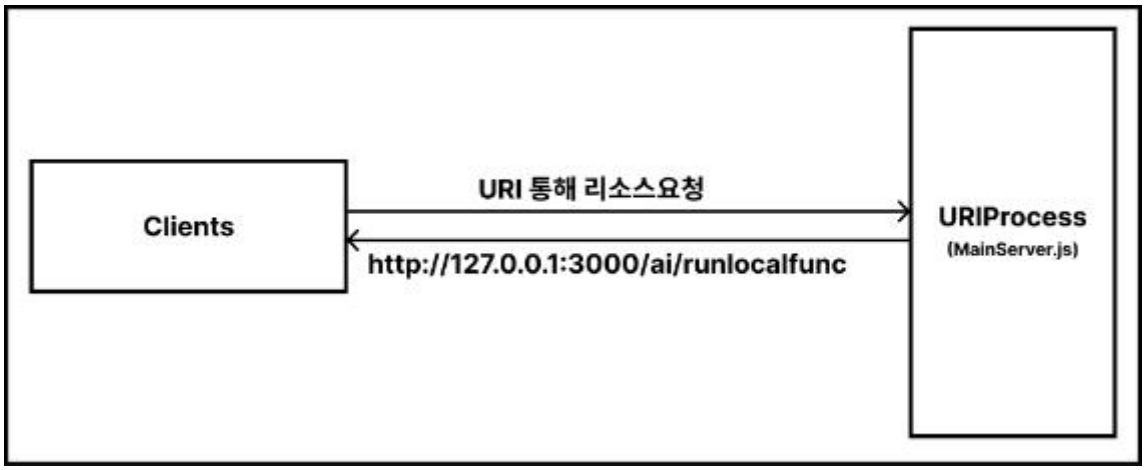
3.1.1 UI/UX 페이지 구현

아래와 같이 와이어프레임을 구현하였으며, 자세한 동작은 1.2에서 확인가능합니다.



3.1.2 서버와의 HTTP 통신기능 구현

클라이언트는 아래와 같이 URI를 통해 서버에 리소스를 요청하고 필요한 데이터를 받습니다. 통신구조는 아래와 같으며 HTTP를 이용해 통신합니다. 추후 CA인증서 발급을 통해 HTTPS 통신으로 개선할 예정입니다.



GET/POST 등의 HTTP method를 통해 데이터를 전송하거나 데이터를 읽어옵니다. 아래는 실시간으로 녹음된 음성을 STT 변환을 위해 서버로 전송하는 코드의 예시입니다.

```dart
// 서버 전송 함수
Future<void> sendToServerWav() async {
  debugPrint("서버 로그는 함수 실행 !!");

  final Uri uri = Uri.parse('http://101.79.9.58:3000/upload');

  if (audioPath.isEmpty) {
    debugPrint('No audio file to send.');
    return;
  }

  File audioFile = File(audioPath);
  if (!await audioFile.exists()) {
    debugPrint('녹음 파일이 존재하지 않습니다: ${audioFile.path}');
    return;
  }

  try {
    var request = http.MultipartRequest('POST', uri)
      ..files.add(
        await http.MultipartFile.fromPath(
          'audio',
          audioFile.path,
          contentType: MediaType('audio', 'wav'),
        ),
      );

    var response = await request.send();
    if (response.statusCode == 200) {
      debugPrint('File sent successfully! 서버에 전송 성공 !!');
    } else {
      debugPrint('Failed to send file: ${response.statusCode}');
    }
  } catch (e) {
    debugPrint('Error sending file: $e');
  }
```

대표적으로 아래와 같은 URI 리스트들이 존재합니다.

```
# 유저정보 요청
http://101.79.9.50:3000/db/users/1

# 상담 텍스트 요약본 가져오기
http://101.79.9.58:3000/db/conversation-summary/1

# AI 함수 호출
http://101.79.9.58:3000/ai/runlocalfunc

# 실시간 .wav 전송
http://101.79.9.58:3000/uploads
```

3.2 Web 클라이언트 소개

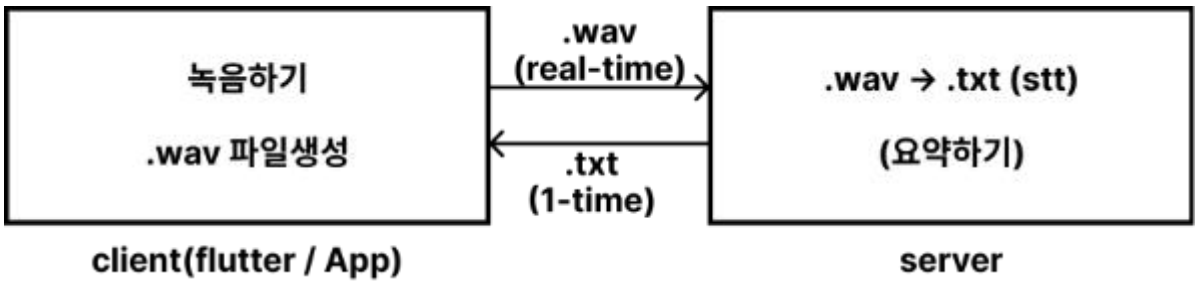
Web 클라이언트는 추후 지자체 혹은 고독사 관리 중심 기관이 사용하게될 관리용 프로그램입니다. 안심하이가 제공하는 핵심기능은 App을 통한 관리 인력의 전문성 강화이기 대문에, 개발 우선순위가 후순위에 배치되어 아직 개발이 착수되지 않았습니다.

4 AI 및 머신러닝 기능

안심하이 시스템의 핵심기능 구현을 위한 부분입니다. 실시간 상담 음성 변환 / 상담 대화 의도 자동 분류 및 질문추천 / 상담 기록 텍스트 요약 / 실시간 대화 내용 기반 추후 질문 생성 / 복지 정책 추천 시스템으로 구성되어있으며 아래에 구체적인 내용이 포함됩니다.

4.1 실시간 상담 음성 변환 (STT)

Google Cloud Speech-to-Text API를 사용하여 실시간으로 진행되는 상담 음성을 텍스트로 변환하여 기록합니다. 변환된 STT는 상담 중 발생하는 모든 대화를 빠르게 텍스트로 전환하여 기록 및 분석에 활용하게 됩니다. 좀 더 구체적으로 상담내용 전체 요약과 질문 추천 추후 복지정책 추천 및 상담내용 텍스트로 저장 등의 기능을 구현가능하게 합니다.



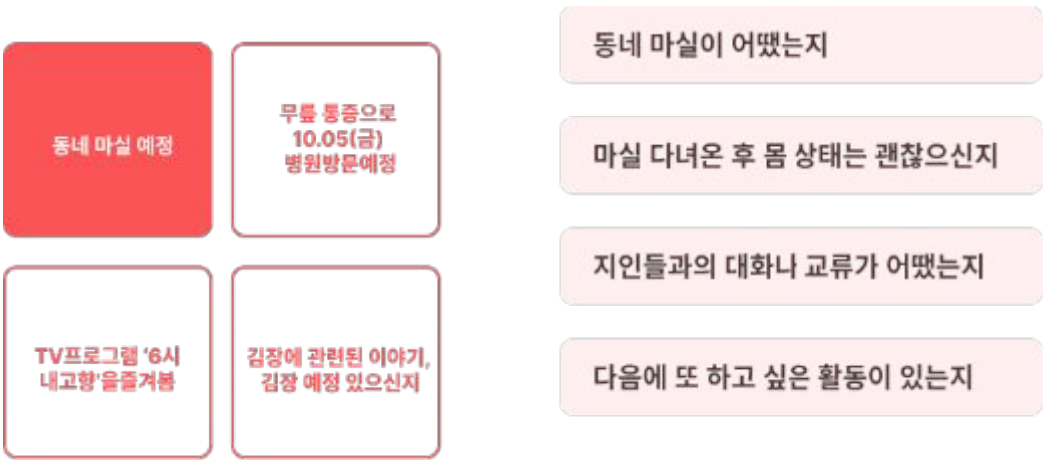
실시간 변환이 이루어지므로 상담사가 대화 내용을 놓치지 않고 기록을 통해 향후 상담 내용 분석이 가능하도록합니다.

Google Cloud Speech-to-Text API를 사용하여 음성의 텍스트 변환 정확도를 높이고, 다양한 환경에서의 음성 인식 성능을 최적화합니다.

음성 데이터는 API를 통해 스트리밍 방식으로 전송되며, 실시간으로 텍스트로 변환됩니다. 서버에서는 websocket을 이용해 실시간으로 .wav 파일을 .txt로 변환합니다. 변환된 텍스트는 후속 분석을 위해 저장되고, 이후 단계에서 주제 분류 및 요약 작업에 활용됩니다.

4.2 상담 대화 의도 자동 분류 및 질문 추천

CNN(Convolutional Neural Network)을 이용하여 상담 내용에서 발화자의 의도를 분석하여 특정 주제로 자동 분류합니다. 이후 상담자에게 적절한 질문들을 추천해주는 기능을 제공합니다. 구체적으로 상담 대화를 주제별로 분류하여 상담의 핵심 의도를 파악하고 기록할 수 있도록 지원합니다. 상담 중 실시간으로 대화 의도를 자동으로 분석하여 상담사의 실시간 피드백을 돕습니다.



텍스트 분류 작업을 수행하여 4.1에서 처리완료한 STT 데이터를 상담 대화를 입력으로 받아 CNN을 통해 대화의 주요 주제(예: 건강 관리, 사회적 상호작용, 영양 관리 등)를 분류. 모델은 사전에 훈련된 데이터셋을 바탕으로 각 상담의 내용을 해당 주제에 맞게 분류하며, 다양한 주제(예: 건강, 생활, 복지 등)를 실시간으로 분류할 수 있도록 설계되었습니다.

4.3 상담 기록 텍스트 요약

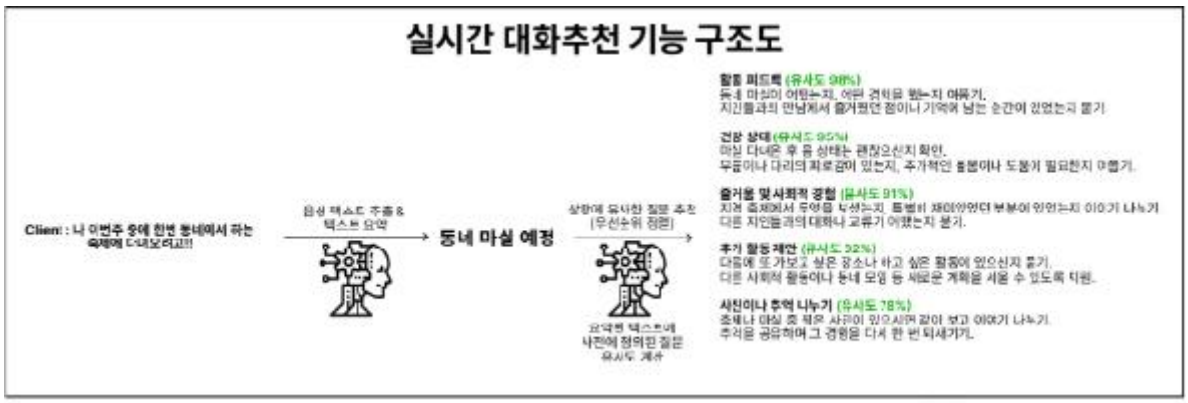
KoBART 모델을 활용한 추상적 요약(Abstractive Summarization)을 수행합니다. 대화의 중요한 내용만 추출하여 새로운 문장으로 요약하며, 상담 내용이 길고 복잡한 경우에도 요약된 정보를 제공하여 상담사의 업무 부담을 줄입니다.

상담 중 길게 진행된 대화의 주요 내용을 자동으로 요약하여 상담사가 빠르게 핵심 포인트를 파악할 수 있도록 지원합니다. 추가로 요약된 내용을 통해 상담 후 복지 대상자의 상태를 분석하고, 다음 상담 시 참고할 수 있는 정보들을 효율적으로 DB에 저장합니다.

4.4 실시간 대화 내용 기반 추후 질문 생성

GPT-3 Turbo 모델을 기반으로 실시간 대화 데이터를 분석하고, 다음 대화를 유도할 수 있는 적절한 질문을 생성합니다. 상담 중 발생하는 다양한 주제(예: 건강, 사회적 상호작용, 가족 관계 등)에 따라 질문이 다르게 생성되며, 자연스럽고 유용한 질문을 제시함으로써 대화를 이어나갑니다.

상담사는 생성된 질문을 바탕으로 복지 대상자의 추가적인 요구나 상태를 파악할 수 있으며, 대상자의 피드백에 따라 상담 방향을 조정할 수 있습니다. (체크리스트 기능)

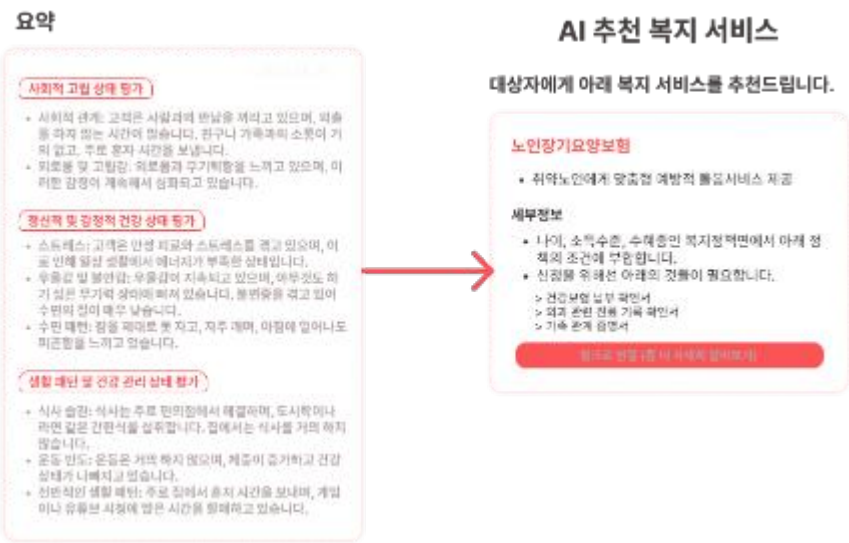


위와 같이 4.1 4.2 4.3 기능과 맞물려서 적절한 대화를 유도할 수 있는 질문을 추천합니다. 구체적으로 STT 변환을 통해 생성된 txt의 의도 분류 이후 상황에 맞는 질문을 텍스트 유사도를 기반으로 계산합니다.

4.5 복지 정책 추천 시스템

Collaborative Filtering 알고리즘을 사용하여 상담 내용을 기반으로 사용자에게 적합한 복지 정책을 추천합니다. 이때 우선적으로 나이, 소득수준, 수혜중인 복지정책을 필터링하고 상담 내용에서 추출된 핵심 정보와 복지 대상자의 선호도를 반영하여 맞춤형 복지 정책을 제안합니다.

사회보장정보원의 복지 데이터베이스와 연결하여 실시간으로 대상자의 상태에 맞는 최신 정책 및 서비스 정보를 제공합니다.



5 서버 사용법 및 URI 호출 방법

5.1 Node.js 서버 실행

SecureACrypto V1.0 암호모듈은 암호 관리자와 일반 사용자 역할을 지원하며 . 암호 관리자 역할은 암호 모듈 설치, 자가시험과 같이 암호모듈의 관리 기능을 수행하는 역할이며, 사용자 역할은 암호모듈에서 제공하는 서비스를 이용하기 위한 역할이다. 암호 관리자 역할과 사용자 역할은 수행하는 서비스에 따라 논리적으로 분리된다.

5.2 클라이언트 테스트

안심하이 시스템의 동작을 확인하기위해 예시 클라이언트들을 제공합니다. 이를 활용하여 백엔드 서버와 AI 모듈의 동작 상태를 확인하고, 테스트할 수 있습니다.

이를 활용하여 추후 확장시에 다양한 환경에서의 클라이언트 구현에 도움을 받을 수 있습니다.

5.2.1 test client

Tools/Client/test_client/index.html 파일을 브라우저에서 열어 클라이언트 테스트를 진행할 수 있습니다. 간이로 만들어진 페이지를 이용하여 각각의 버튼을 눌러 서버의 AI, DB, Remote Data 모듈에 대한 요청을 보낼 수 있습니다.

5.2.2 직접 URI 전송하여 테스트

MainServer.js를 실행한 기기(Local) 혹은 클라우드나 서버로 인터넷에 공개된 경우(Remote) 해당 URL을 사용하여 URI로 요청을 보낼 수 있습니다.

아래는 로컬환경에서 테스트를 진행하는 예시이며, 아래와 같이 브라우저에서 직접 URI를 호출해 기능을 테스트할 수 있습니다.

6 운영환경

안심하이 시스템은 아래에 명시된 운영환경에서 제공되고 사용가능합니다.

구분			
종류	버전	비트	아키텍처
Linux	Ubuntu 18.04 LTS	64	X86-64
	Ubuntu 20.04 LTS		
	Ubuntu 22.04 LTS		
node	v16.20.0	64	X64-based
axios	^1.7.7	64	X86-64
mysql2	^3.11.3	64	X86-64
express	^4.21.0	64	X86-64
ws	^8.18.0	64	X86-64