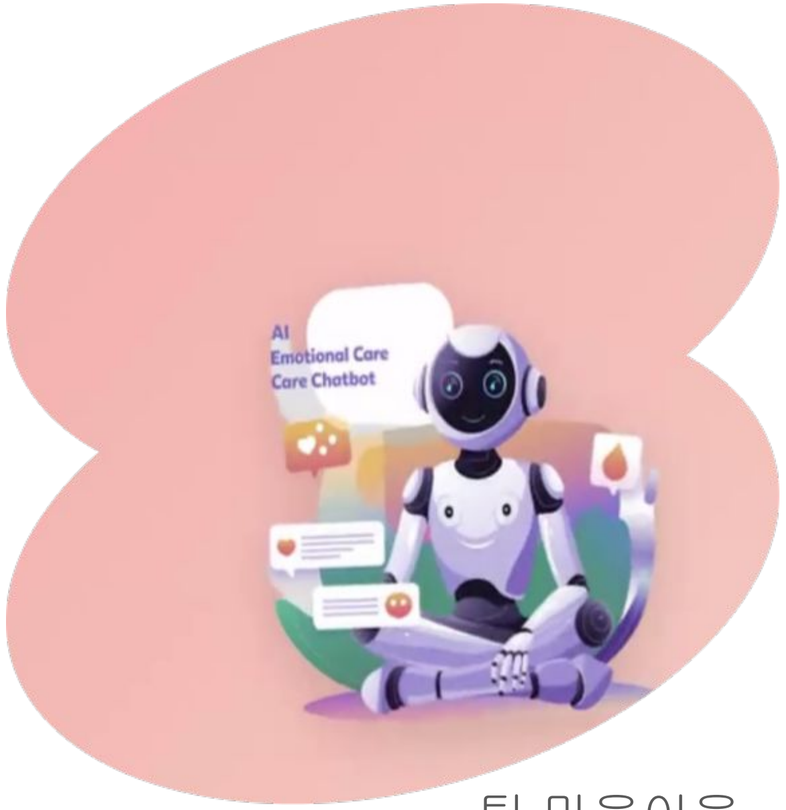
RAG 기반

AI 감정 케어 챗봇 서비스

MindLink



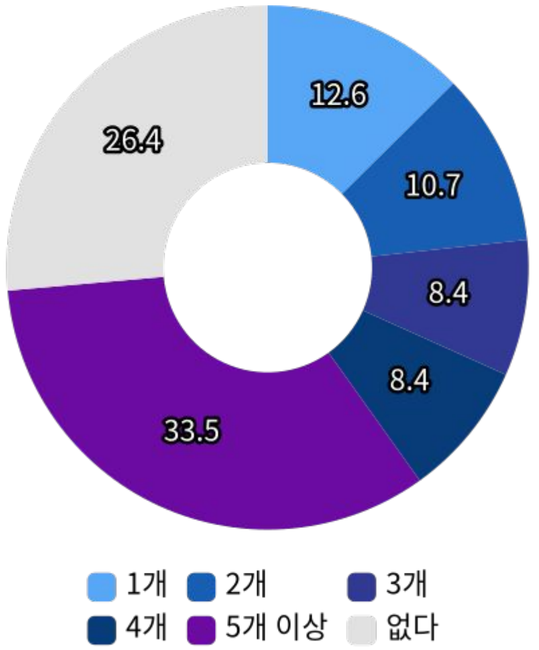
팀 마음이음

CONTENTS

01. 프로젝트 배경

국내 정신건강통계 (2024년)

지난 1년간 정신건강 문제를 경험한 비율

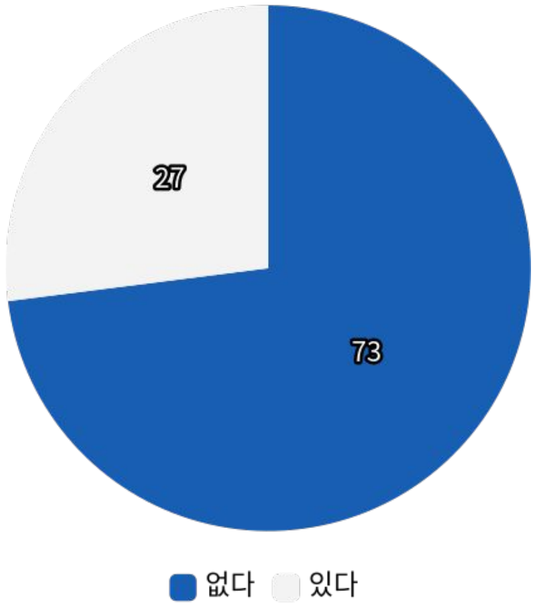


73.6%

우울, 스트레스, 불면 등 정신건강 문제를 경험

22년에 비해 9.8% 증가

상담 또는 병원 방문 여부

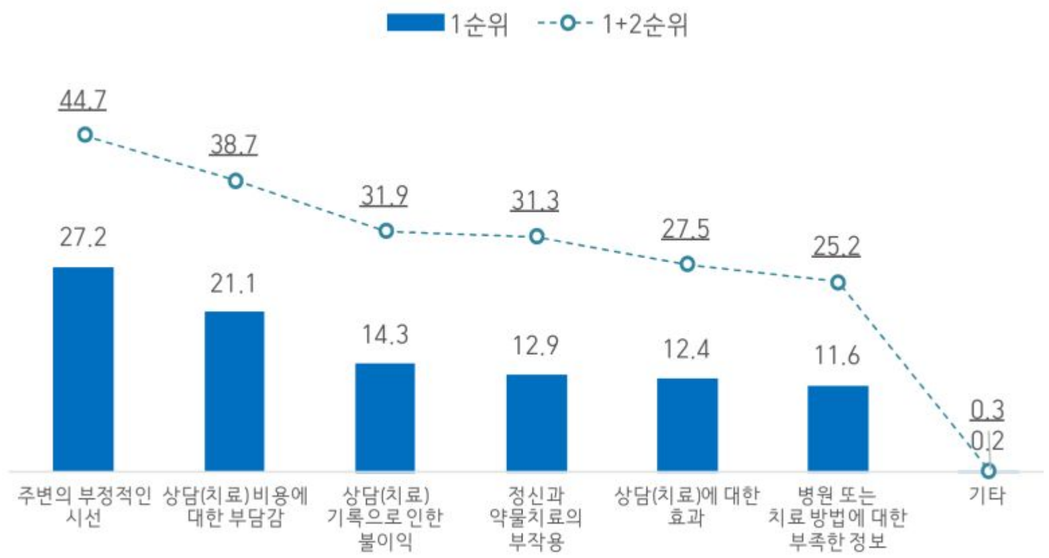


73.0%

치료 받은 적 없다.

국내 정신건강통계 (2024년)

정신건강 문제 치료 시 가장 우려되는 점



- 주변의 부정적인 시선
- 상담/치료 비용에 대한 부담감

특히, 정신건강 문제를 경험한 집단은

비용에 대한 부담(40.7%)이 매우 크게 나타남

비대면 심리상담 시장 분석

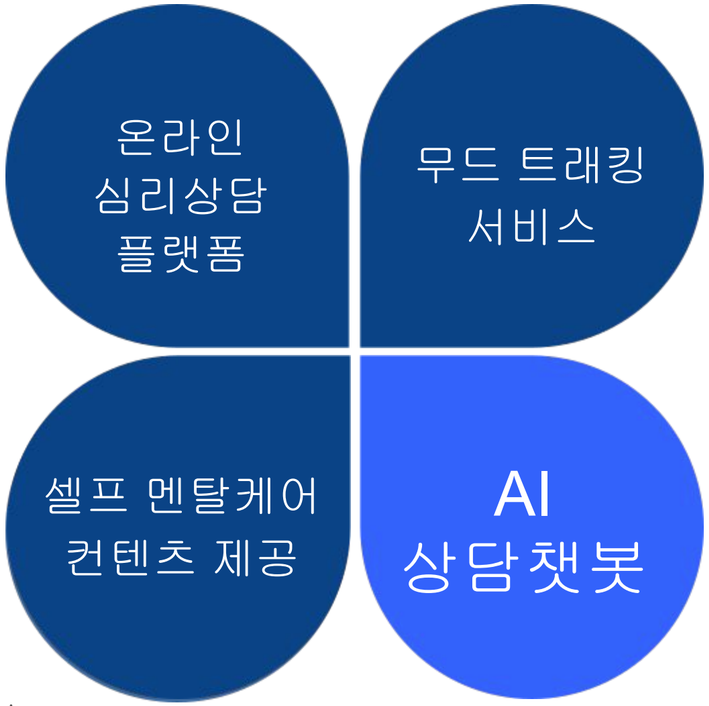
Trost.

실제 상담사와의 연결이 메인 서비스

- 단점) 적지 않은 비용 발생
- AI) 댓글봇, 서비스 안내

AI 기반 맞춤형 셀프케어 서비스

- 감정을 표현하기보다는 기분 전환을 위한 컨텐츠를 우선시



이모지를 통한 감정트래킹 서비스

- 장점) 진입장벽 낮음

LLM 기반 심리상담 챗봇

- 사람과 달리 즉각적 반응 가능
- 시공간적 이점 (새벽 2시에도 상담가능)
- 단점) 기록 모아보기 및 분석은 어려움

AI 감정케어 챗봇
'마음이음'의 역할

AI 챗봇의 장점

- 실시간 감정 분석으로 맞춤형 대화 제공
- 시공간적 제약 없음

대면 치료 연결 체계

- 주변 상담센터 정보 안내

음성 인식 시스템

- 긴 고민을 타자로 직접 입력하지 않아도 음성으로 상담 가능

지속적 감정 트래킹

- 이모지 기록으로 진입장벽 완화
- 캘린더로 감정 흐름 파악

02. 주요 기능

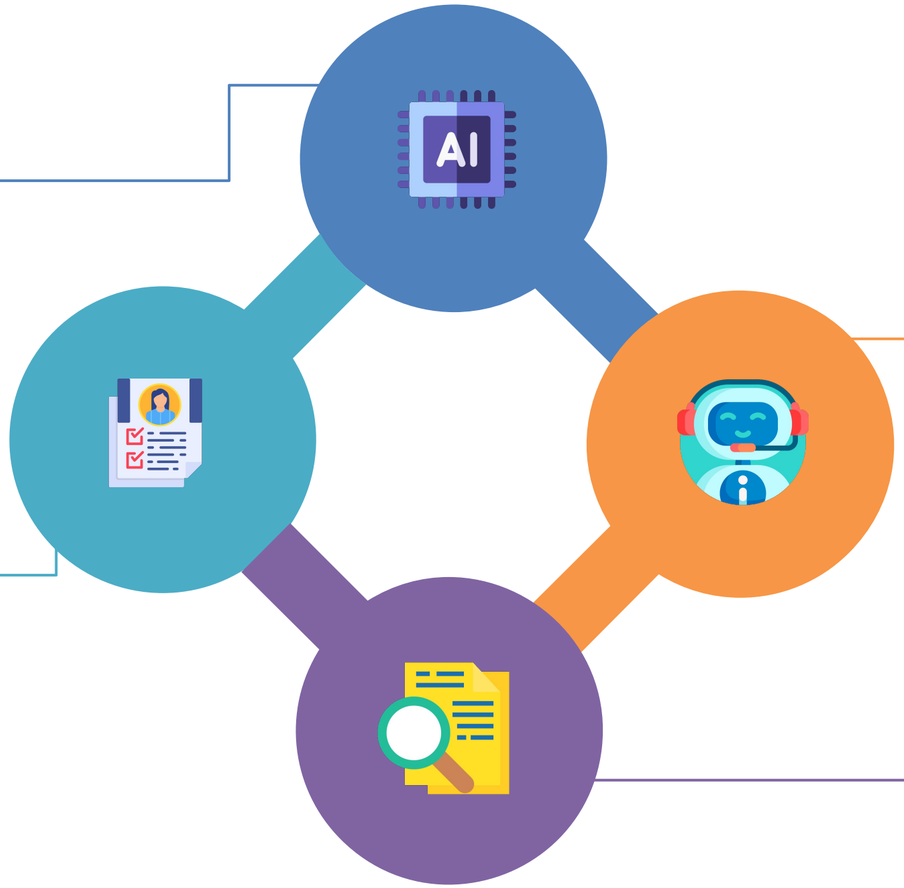
프로젝트 주요 기능

맞춤형 상담사 선택

전문 심리상담과 또래상담 중
사용자 선택 가능으로
차별화된 경험 제공

실시간 감정 분석

HuggingFace 감정 분석 모델
적용으로 실시간 감정 변화 제공



RAG 기반 챗봇

RAG + Sentence Transformer
통한 맞춤화된 심리 상담 서비스
제공

상담 센터 검색

카카오맵 API를 통해
상담 센터 정보 제공

03. 프로젝트 설계

프로젝트 설계 – 개발 플랫폼 및 사용기술

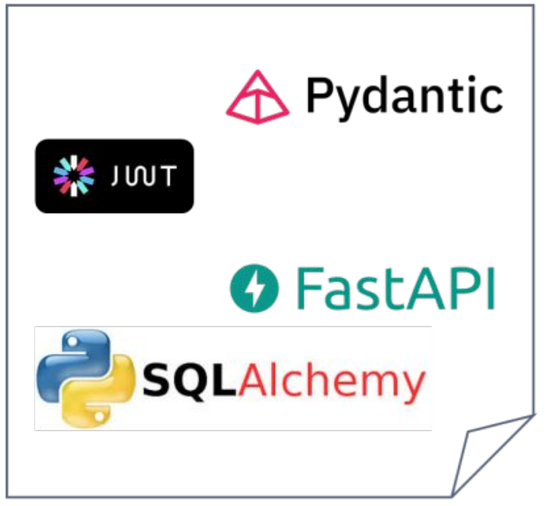
기획설계

형상관리

DB

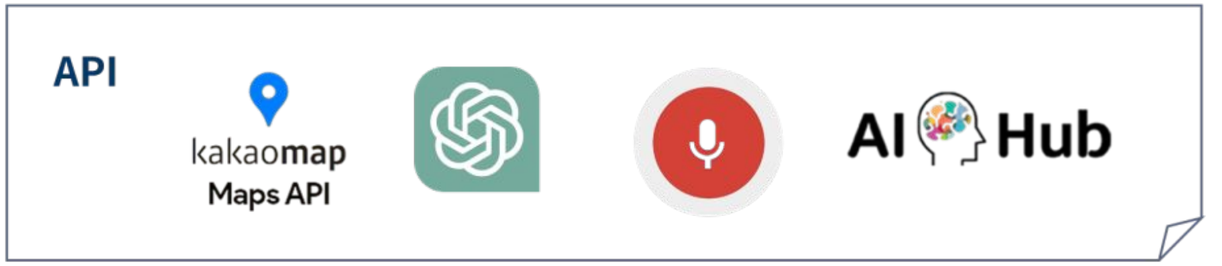
Front-End

Back-End



AI

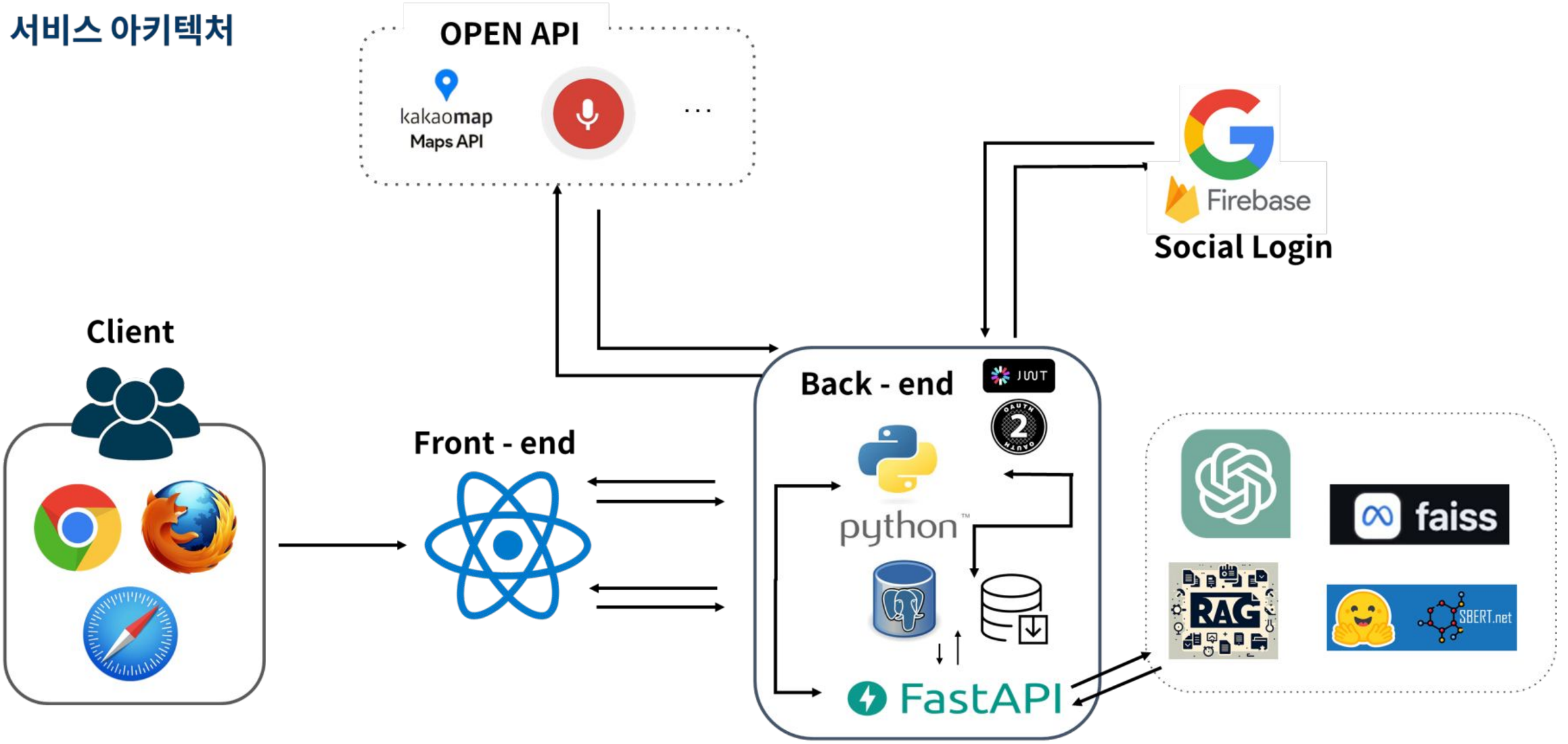
API



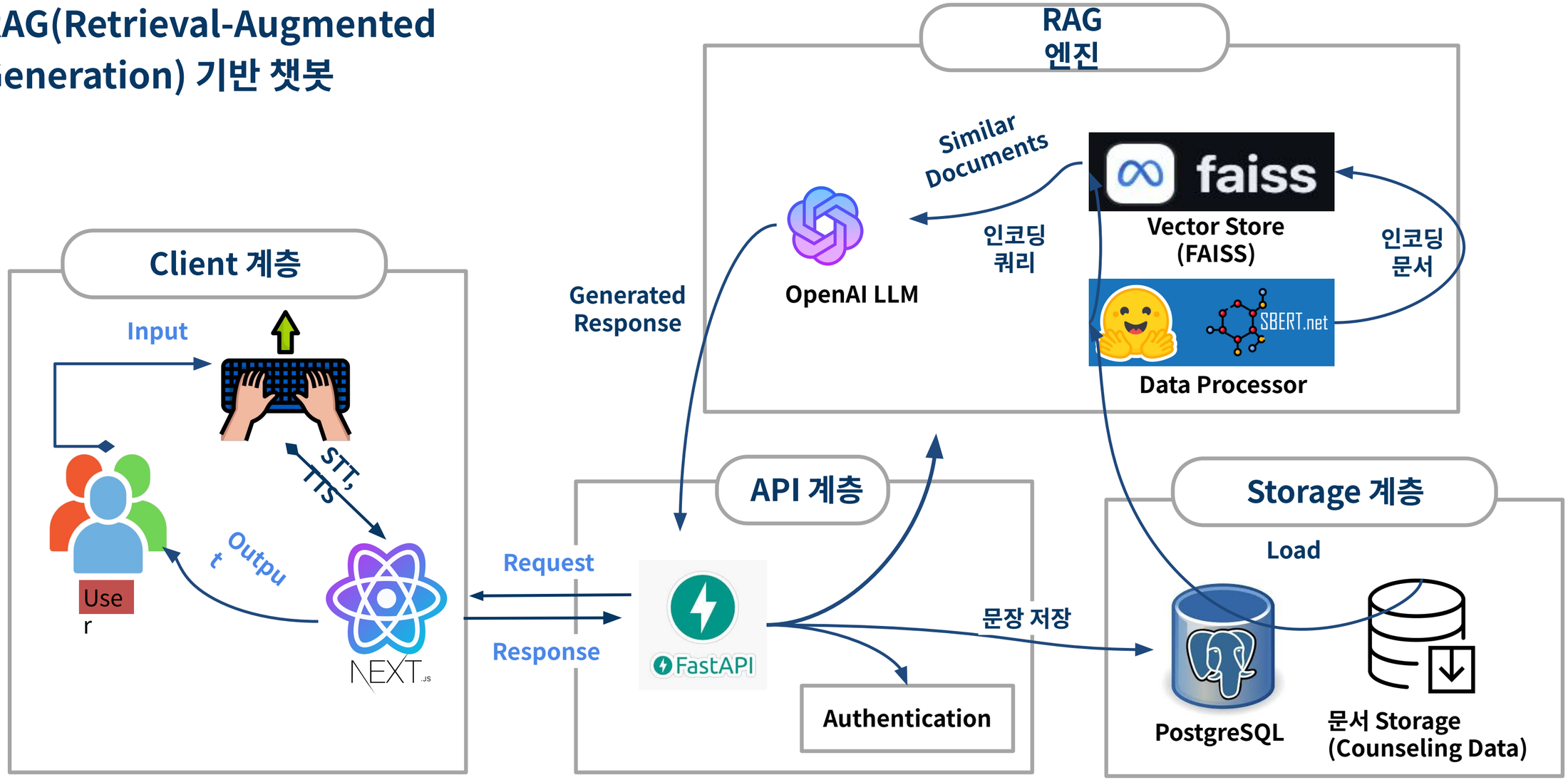
TOOL

기타

서비스 아키텍처

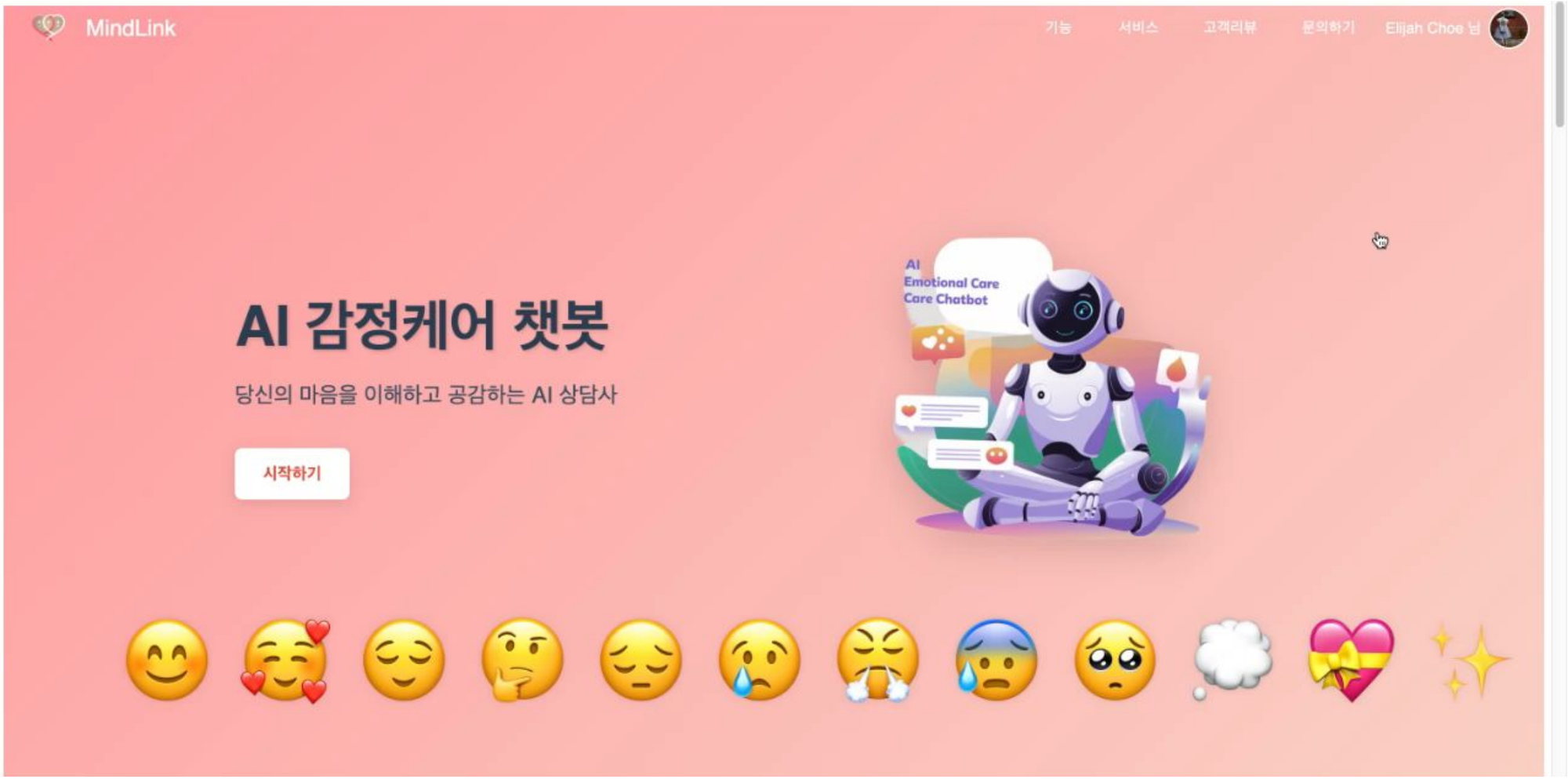

OPEN API
kakaomap
Maps API
...
Firebase
Social Login
Client
Front - end
Back - end
JWT
OAUTH 2
python
FastAPI
faiss
RAG
SBERT.net

RAG(Retrieval-Augmented Generation) 기반 챗봇



04. 주요 소스 코드

랜딩페이지


MindLink
기능
서비스
고객리뷰
문의하기
Elijah Choe 님
AI 감정케어 챗봇
당신의 마음을 이해하고 공감하는 AI 상담사
시작하기
AI
Emotional Care
Care Chatbot

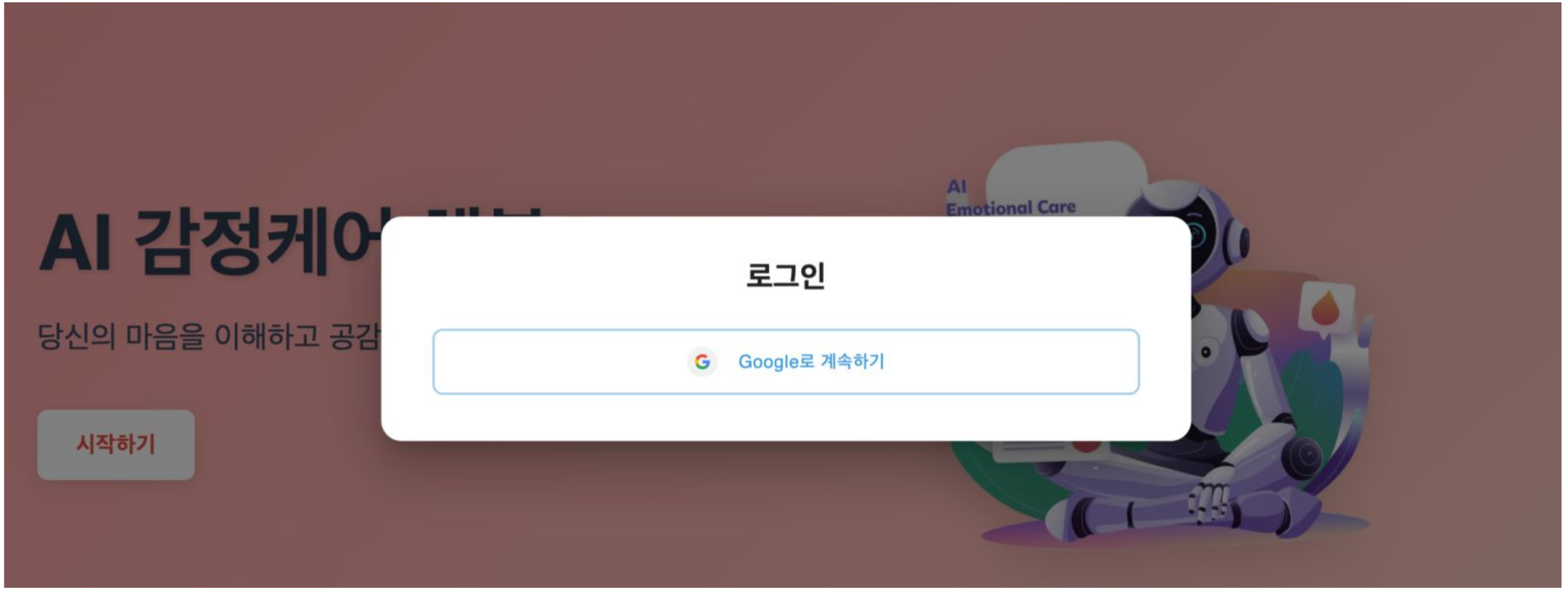
소셜 로그인

파이어베이스 연동을 통한 구글 로그인

```
const handleGoogleLogin = async () => {
    setIsLoading(true);
    const provider = new GoogleAuthProvider();

    const result = await signInWithPopup(auth, provider);
    const user = result.user;
    const idToken = await user.getIdToken();

    // 백엔드 요청 데이터
    const requestData = {
      firebaseToken: idToken,
      email: user.email,
      displayName: user.displayName,
      photoURL: user.photoURL,
    };

    // backend241127_v2.txt의 /api/auth/verify 엔드포인트로 요청
    const response = await fetch(`${process.env.NEXT_PUBLIC_API_URL}/api/auth/verify`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(requestData),
    });
```

```
    const responseData = await response.json();

    if (responseData.success) {
      updateUser({
        uid: user.uid,
        email: user.email || '',
        nickname: user.displayName || '',
        provider: 'google',
        photoURL: user.photoURL || undefined
      });

      setAlert({
        show: true,
        message: "로그인 성공!",
        severity: "success",
      });

      onClose();
    }

  } catch (error: any) {
    console.error("Login process error:", error);
    setAlert({
      show: true,
      message: `로그인 중 오류가 발생했습니다: ${error?.message || '알 수 없는 오류'}`,
```

소셜 로그인


AI 감정케어
당신의 마음을 이해하고 공감
시작하기
로그인
Google로 계속하기
AI
Emotional Care

실시간 감정 분석

HuggingFace 감정 분석 모델 BGE-M3 이용

```python
async def get_response_with_metadata(self, query: str, chat_history: List[Dict], role: str) -> Dict:
    try:
        context_data = self.generate_context(query)
        response_text = await self.get_response(query, chat_history, role)

        # 토픽 분석
        topics = self.emotion_analyzer.analyze_topics(query)

        # 키워드 분석
        keyword_analysis = self._analyze_keywords(query)

        return {
            "response": response_text,
            "metadata": {
                "timestamp": datetime.now().isoformat(),
                "counselor_role": role,
                "counselor_info": COUNSELOR_ROLES[role],
                "similar_cases": context_data["cases"],
                "similar_case_count": len(context_data["cases"]),
                "chat_history_length": len(chat_history),
                "topics": topics,
                "keywords": keyword_analysis
            },
            "analysis": {
                "topics": topics,
                "keywords_detected": {
                    "긍정": [k for k, v in keyword_analysis["긍정"].items() if v > 0],
                    "부정": [k for k, v in keyword_analysis["부정"].items() if v > 0],
                    "중립": [k for k, v in keyword_analysis["중립"].items() if v > 0]
                }
            }
        }
    except Exception as e:
        return {
            "response": f"😔 죄송합니다. 오류가 발생했습니다: {str(e)}",
            "metadata": {
                "error": str(e),
                "timestamp": datetime.now().isoformat()
            }
        }
```

챗봇 응답 코드

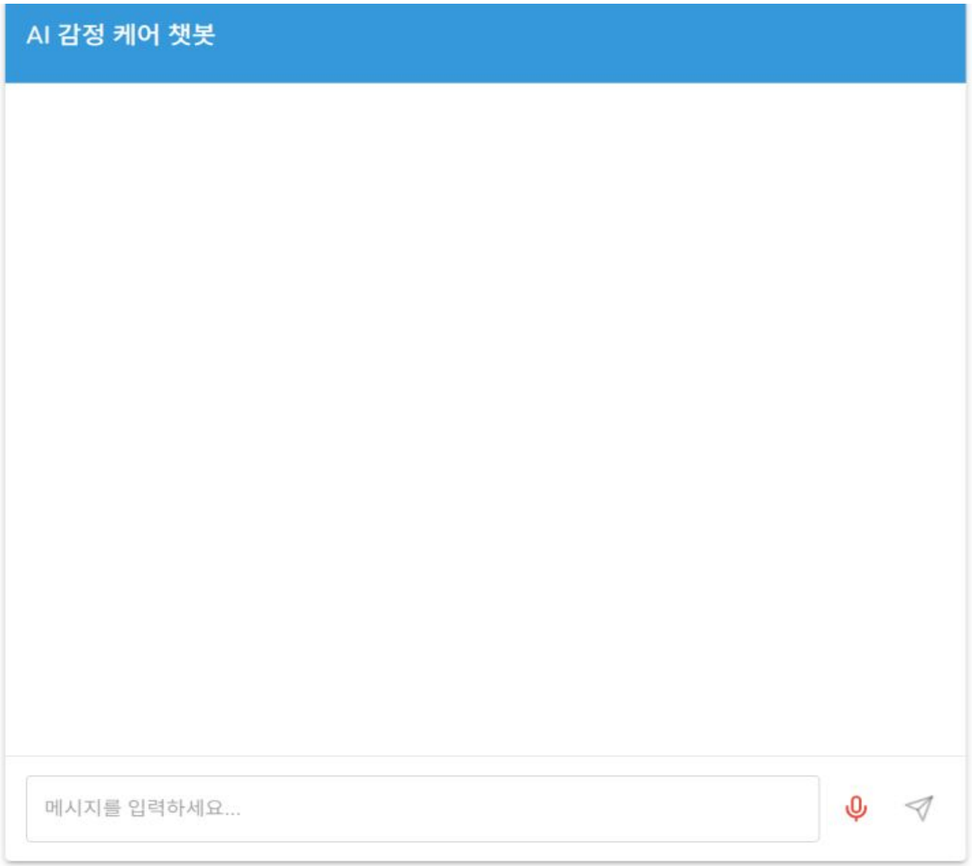
faiss를 통한 빠른 답변

```python
class RAGEngine:

    def generate_context(self, query: str) -> Dict:
        similar_cases = self.data_processor.find_similar_cases(query, k=2)

        context_text = "다음은 유사한 상담 사례들입니다:\n\n"
        for i, case in enumerate(similar_cases, 1):
            context_text += f"사례 {i}:\n내담자: {case['input']}\n상담사: {case['output']}\n\n"

        return {
            "text": context_text,
            "cases": similar_cases
        }

    async def get_response(self, query: str, chat_history: List[Dict], role: str) -> str:
        try:
            context = self.generate_context(query)
            role_type = role if role in COUNSELOR_ROLES else "professional"
            counselor = CounselorRole(role_type)
            system_prompt = counselor.get_prompt_template(context["text"])

            messages = [
                {"role": "system", "content": system_prompt}
            ]

            recent_history = chat_history[-4:] if len(chat_history) > 4 else chat_history
            messages.extend(recent_history)
            messages.append({"role": "user", "content": query})

            response = self.client.chat.completions.create(
                model="gpt-4o-mini",
                messages=messages,
                temperature=0.3,
                max_tokens=500,
                top_p=0.9,
                frequency_penalty=0.7,
                presence_penalty=0.7
            )

            return response.choices[0].message.content

        except Exception as e:
            return f"😓 죄송합니다. 오류가 발생했습니다: {str(e)}"
```

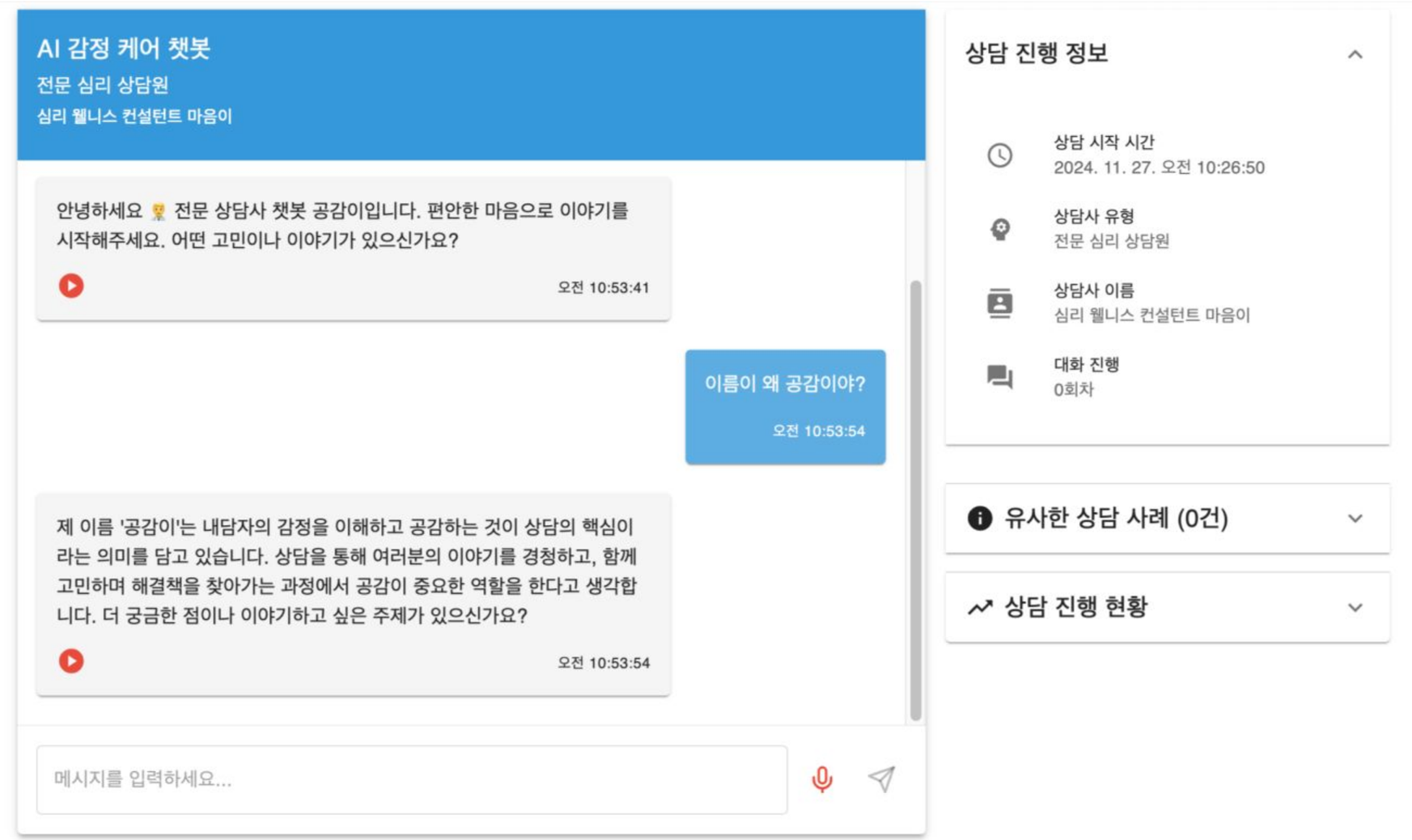
```python
def find_similar_cases(self, query: str, k: int = 3) -> List[Dict]:
    if not self.counseling_data or not self.index:
        return []

    try:
        query_vector = self.encode_text(query).reshape(1, -1)
        distances, indices = self.index.search(query_vector, k)

        # 유사도 점수 기반 필터링 추가(1123)
        threshold = 0.5
        similar_cases = []

        for dist, idx in zip(distances[0], indices[0]):
            if dist > threshold and 0 <= idx < len(self.counseling_data):  # IP 유사도는 높을수록 유사
                case = self.counseling_data[idx].copy()
                case['similarity_score'] = float(dist)  # 유사도 점수 추가
                similar_cases.append(case)

        # 유사도 점수로 정렬
        similar_cases.sort(key=lambda x: x['similarity_score'], reverse=True)
        return similar_cases

    ept Exception as e:
        print(f"유사 케이스 검색 오류: {str(e)}")
        return []
```

sentence transformer 인코딩, 임베딩 코드

```python
@lru_cache(maxsize=1024)
def encode_text(self, text: str) -> np.ndarray:
    text_with_instruction = f"{self.instruction}{text}"
    return self.model.encode(
        text_with_instruction,
        normalize_embeddings=True,
        show_progress_bar=False
    )

def create_embeddings(self) -> bool:
    if not self.counseling_data:
        return False

    try:
        texts = [data['input'] for data in self.counseling_data]
        embeddings_list = []

        for i in range(0, len(texts), self.batch_size):
            batch_texts = texts[i:i + self.batch_size]
            # instruction 추가
            batch_texts_with_instruction = [f"{self.instruction}{text}" for text in batch_texts]
            batch_embeddings = self.model.encode(
                batch_texts_with_instruction,
                normalize_embeddings=True,
                show_progress_bar=False,
                batch_size=self.batch_size
            )
            embeddings_list.append(batch_embeddings)

        embeddings = np.vstack(embeddings_list)

        dimension = embeddings.shape[1]
        self.index = faiss.IndexFlatIP(dimension)
        self.index.add(embeddings)

        return True

    except Exception as e:
        print(f"임베딩 생성 오류: {str(e)}")
        return False
```



챗봇 응답 화면


AI 감정 케어 챗봇
전문 심리 상담원
심리 웰니스 컨설턴트 마음이
안녕하세요 👨‍⚕️ 전문 상담사 챗봇 공감이입니다. 편안한 마음으로 이야기를 시작해주세요. 어떤 고민이나 이야기가 있으신가요?
오전 10:53:41
이름이 왜 공감이야?
오전 10:53:54
제 이름 '공감이'는 내담자의 감정을 이해하고 공감하는 것이 상담의 핵심이라는 의미를 담고 있습니다. 상담을 통해 여러분의 이야기를 경청하고, 함께 고민하며 해결책을 찾아가는 과정에서 공감이 중요한 역할을 한다고 생각합니다. 더 궁금한 점이나 이야기하고 싶은 주제가 있으신가요?
오전 10:53:54
메시지를 입력하세요...
상담 진행 정보
상담 시작 시간
2024. 11. 27. 오전 10:26:50
상담사 유형
전문 심리 상담원
상담사 이름
심리 웰니스 컨설턴트 마음이
대화 진행
0회차
유사한 상담 사례 (0건)
상담 진행 현황

가까운 상담센터
검색

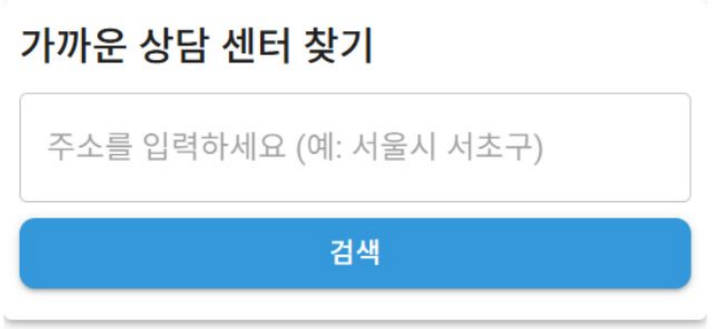
지도 구현 코드

```
async def find_centers(location: Location):
    try:
        for keyword in keywords:
            params = {
            }

            # API 호출
            headers = {"Authorization": f"KakaoAK {kakao_api_key}"}
            response = requests.get(
                kakao_url,
                params=params,
                headers=headers
            )

            if response.status_code == 200:
                data = response.json()

                # 검색 결과 처리
                for place in data.get("documents", []):
                    # 중복 제거를 위한 확인
                    if not any(center["name"] == place["place_name"] for center in all_centers):
                        center = {
                            "name": place["place_name"],
                            "address": place["road_address_name"] or place["address_name"],
                            "phone": place["phone"] or "번호 없음",
                            "distance": f"{float(place['distance'])/1000:.1f}km",
                            "lat": float(place["y"]),
                            "lon": float(place["x"])
                        }
                        all_centers.append(center)

        # 거리순 정렬
        all_centers.sort(key=lambda x: float(x["distance"].replace("km", "")))

        # 최대 5개까지만 반환
        centers_to_return = all_centers[:5]

        logger.info(f"Found {len(centers_to_return)} centers")
        return {"success": True, "data": centers_to_return}

    except Exception as e:
        logger.error(f"Error finding centers: {str(e)}")
        return {"success": False, "error": str(e)}
```

지도 구현 백엔드 코드

```
async def find_centers(location: Location):

@router.post("/search")
async def search_location(query: dict):
    """Search location by address using Kakao Address API"""
    try:
        address = query.get("address", "")

        # 카카오 주소 검색 API 사용
        kakao_url = "https://dapi.kakao.com/v2/local/search/address.json"
        headers = {"Authorization": f"KakaoAK {kakao_api_key}"}

        params = {
            "query": address,
            "analyze_type": "exact"
        }

        response = requests.get(
            kakao_url,
            params=params,
            headers=headers
        )

        if response.status_code == 200:
            data = response.json()
            if data.get("documents"):
                result = data["documents"][0]
                return {
                    "success": True,
                    "data": {
                        "lat": float(result["y"]),
                        "lon": float(result["x"]),
                        "address": result["address_name"]
                    }
                }
            else:
                return {"success": False, "error": "주소를 찾을 수 없습니다"}
        else:
            return {"success": False, "error": "카카오 API 호출 실패"}

    except Exception as e:
        logger.error(f"Error searching location: {str(e)}")
        return {"success": False, "error": "주소 검색 실패"}
```

가까운 상담센터 검색

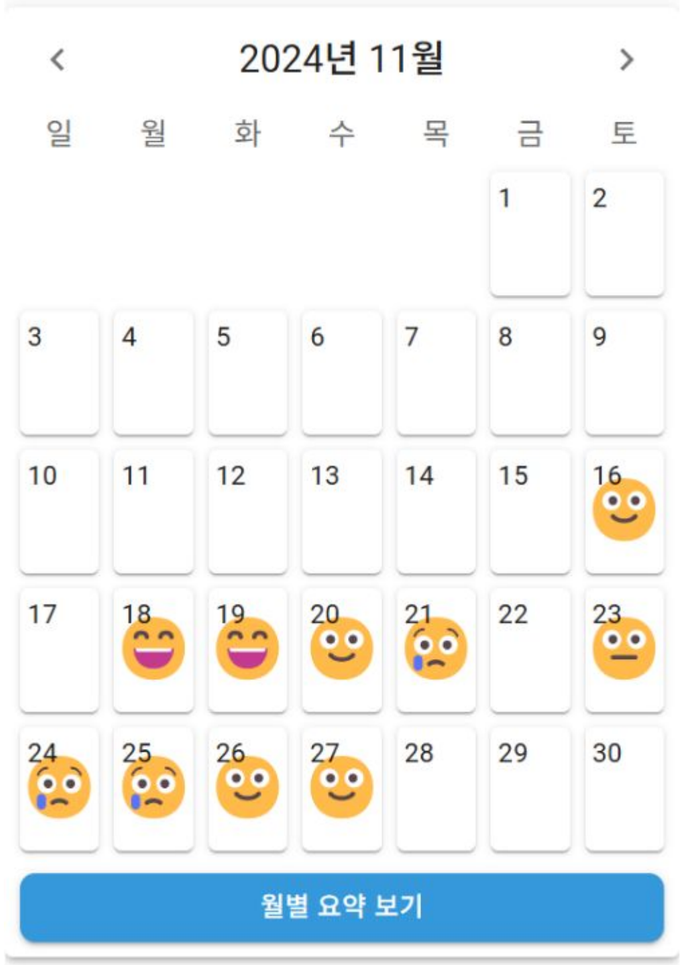


지도에 장소 마커

상담센터 정보 제공 및 길찾기

감정 트래킹 캘린더

이모지 기록으로 시각화



05. 프로젝트 시연