AI Agent 개발 과제 - 요구사항 정의서 (서버-클라이언트 구조)

1. 프로젝트 개요

1.1 프로젝트명

일반시민을 위한 정부 공문서 AI 검색 서비스

1.2 프로젝트 목적

- 행정안전부에서 제공하는 정부 공문서 AI 학습데이터를 활용한 시민 친화적 정보 검색 시스템 구축
- 복잡한 정부 정책과 공문서를 일반 시민이 쉽게 이해할 수 있도록 돕는 AI Agent 서비스 개발
- RAG 기술을 활용하여 정확하고 이해하기 쉬운 정보 제공

1.3 대상 사용자

- **주 대상**: 일반 시민 (정부 정책 및 공문서 정보가 필요한 모든 국민)
- **사용자 특성**:
  - 정부 용어나 공문서 형식에 익숙하지 않음
  - 간단하고 직관적인 인터페이스 선호
  - 빠르고 정확한 정보 제공 요구
  - 모바일 및 웹 접근성 중요

1.4 시스템 아키텍처 개요

```
[클라이언트 (Streamlit)] ↔ [서버 (FastAPI)] ↔ [Vector DB + Azure OpenAI]
```

2. 기능 요구사항

2.1 핵심 기능

2.1.1 시민 친화적 정보 검색

- **자연어 질의 처리**: "아이 양육비 지원은 어떻게 받나요?" 같은 일상 언어 질문 처리
- **생활 밀착형 카테고리**: 복지, 세금, 교육, 주택, 취업 등 시민 관심사별 분류
- **스마트 추천**: 검색 이력 기반 관련 정책 추천

2.1.2 쉬운 언어로 설명

- **공문서 번역**: 어려운 공문서를 일반인이 이해하기 쉬운 언어로 변환
- **핵심 정보 요약**: "누가, 언제, 어떻게, 얼마나" 형태의 간단한 요약 제공
- **FAQ 형태 답변**: 자주 묻는 질문 형태로 구조화된 답변

2.1.3 대화형 상담 서비스

- **단계별 안내**: 복잡한 절차를 단계별로 안내
- **추가 질문 유도**: "더 궁금한 점이 있으신가요?" 형태의 후속 질문 제안
- **맥락 유지**: 이전 대화 내용을 기억하여 연관 질문 처리

2.2 시민 편의 기능

2.2.1 개인화 서비스

- **관심 분야 설정**: 사용자별 관심 정책 분야 설정
- **알림 서비스**: 관련 새 정책 발표시 알림 (선택사항)
- **북마크 기능**: 자주 찾는 정보 저장

2.2.2 접근성 강화

- **음성 인식**: 텍스트 입력이 어려운 사용자를 위한 음성 검색 (선택사항)
- **다국어 지원**: 주요 외국어 질문 처리 (선택사항)
- **간편 공유**: 검색 결과를 가족/지인과 쉽게 공유

3. 서버-클라이언트 구조 설계

3.1 클라이언트 (Frontend - Streamlit)

3.1.1 역할 및 책임

- **사용자 인터페이스 제공**: 직관적이고 사용하기 쉬운 웹 인터페이스
- **사용자 입력 처리**: 검색 쿼리, 카테고리 선택, 필터링 옵션 입력 받기
- **결과 시각화**: 서버로부터 받은 데이터를 사용자 친화적으로 표시
- **상태 관리**: 사용자 세션, 검색 기록, 북마크 관리

3.1.2 주요 구성 요소

```
streamlit_app.py
├── pages/
│   ├── main_search.py          # 메인 검색 페이지
│   ├── category_browse.py      # 카테고리별 탐색
│   ├── search_history.py       # 검색 기록
│   └── bookmarks.py            # 북마크 관리
├── components/
│   ├── search_interface.py     # 검색 입력 컴포넌트
│   ├── result_display.py       # 결과 표시 컴포넌트
│   └── chat_interface.py       # 대화형 인터페이스
└── utils/
    ├── api_client.py           # 서버 API 호출
    └── ui_helpers.py           # UI 유틸리티 함수
```

3.2 서버 (Backend - FastAPI)

3.2.1 역할 및 책임

- **API 엔드포인트 제공**: RESTful API를 통한 클라이언트 요청 처리
- **비즈니스 로직 처리**: Multi-Agent 시스템을 통한 복잡한 질의 처리
- **데이터 관리**: Vector DB 연동 및 데이터 검색
- **AI 모델 연동**: Azure OpenAI API 호출 및 응답 처리

3.2.2 주요 구성 요소

```
fastapi_server/
├── main.py                 # FastAPI 애플리케이션 엔트리포인트
├── api/
│   ├── endpoints/
│   │   ├── search.py       # 검색 관련 엔드포인트
│   │   ├── chat.py         # 대화형 상담 엔드포인트
│   │   └── admin.py        # 관리자 기능 (선택사항)
├── core/
│   ├── agents/
│   │   ├── query_analyzer.py  # 질의 분석 Agent
│   │   ├── document_retriever.py # 문서 검색 Agent
│   │   ├── content_processor.py  # 내용 처리 Agent
│   │   └── response_generator.py # 답변 생성 Agent
│   ├── services/
│   │   ├── rag_service.py    # RAG 파이프라인 서비스
│   │   ├── vector_service.py # Vector DB 서비스
│   │   └── openai_service.py # Azure OpenAI 서비스
├── models/
│   ├── request_models.py     # API 요청 모델
│   └── response_models.py    # API 응답 모델
├── utils/
│   ├── config.py           # 설정 관리
│   ├── logging.py          # 로깅 설정
│   └── exceptions.py       # 커스텀 예외
└── data/
    ├── preprocessor.py     # 데이터 전처리
    └── vector_builder.py   # Vector DB 구축
```

3.3 데이터 계층

3.3.1 Vector Database (FAISS/ChromaDB)

- **문서 임베딩 저장**: 정부 공문서의 벡터 표현 저장
- **메타데이터 관리**: 문서 제목, 발행일, 카테고리, 태그 정보
- **빠른 검색**: 유사도 기반 고속 검색 지원

3.3.2 캐시 계층 (선택사항)

- **자주 검색되는 질의 캐싱**: 응답 시간 최적화
- **임시 세션 데이터**: 대화 맥락 임시 저장

4. API 설계

4.1 주요 API 엔드포인트

4.1.1 검색 관련 API

```python
POST /api/v1/search/query
# 자연어 질의 검색
Request: {
    "query": "아이 양육비 지원 신청 방법",
    "category": "복지",
    "max_results": 5
}
Response: {
    "results": [...],
    "summary": "간단한 요약",
    "related_queries": [...]
}

GET /api/v1/search/categories
# 카테고리 목록 조회

POST /api/v1/search/similar
# 유사 문서 검색
```

4.1.2 대화형 상담 API

```python
POST /api/v1/chat/message
# 대화형 메시지 처리
Request: {
    "message": "사용자 메시지",
    "session_id": "세션ID",
    "context": "이전 대화 맥락"
}

GET /api/v1/chat/history/{session_id}
# 대화 기록 조회
```

4.2 데이터 모델

4.2.1 검색 요청/응답 모델

```python
```

```python
class SearchRequest(BaseModel):
    query: str
    category: Optional[str] = None
    max_results: int = 5
    filters: Optional[Dict] = None

class SearchResponse(BaseModel):
    results: List[DocumentResult]
    summary: str
    total_count: int
    processing_time: float
    related_queries: List[str]
```

5. 기술 요구사항

5.1 클라이언트 기술 스택

- **Frontend Framework**: Streamlit
- **HTTP Client**: requests / httpx
- **상태 관리**: Streamlit session state
- **UI 컴포넌트**: Streamlit 기본 컴포넌트 + 커스텀 컴포넌트

5.2 서버 기술 스택

- **Backend Framework**: FastAPI
- **AI Framework**: LangChain + LangGraph
- **Vector Database**: FAISS 또는 ChromaDB
- **HTTP Client**: httpx (Azure OpenAI API 호출용)
- **Data Processing**: pandas, numpy

5.3 Multi-Agent 시스템 설계

5.3.1 Agent 구성

python

1. CitizenQueryAnalyzer Agent
   - 시민의 일상 언어 질의를 분석
   - 의도 파악 및 키워드 추출
   - 관련 카테고리 분류

2. PolicyDocumentRetriever Agent
   - Vector DB에서 관련 문서 검색
   - 문서 관련성 평가 및 랭킹
   - 다중 소스 정보 통합

3. CitizenFriendlyProcessor Agent
   - 공문서를 시민 친화적 언어로 변환
   - 복잡한 절차를 단계별로 정리
   - 핵심 정보 추출 및 구조화

4. InteractiveResponseGenerator Agent
   - 최종 답변 생성
   - 추가 질문 제안
   - 관련 정보 링크 제공

5.4 RAG 파이프라인 특화

5.4.1 시민 맞춤형 전처리

- **용어 사전 구축**: 공문서 용어 → 일반 용어 매핑
- **생활 연관성 태깅**: 일상생활과의 연관성에 따른 문서 가중치
- **절차 구조화**: 신청 방법, 준비 서류 등을 구조적으로 추출

5.4.2 검색 최적화

- **의미 기반 검색**: 단순 키워드가 아닌 의미 기반 매칭
- **맥락 고려**: 사용자의 이전 질문 맥락을 고려한 검색
- **개인화**: 사용자 프로필 기반 검색 결과 개인화

6. 사용자 인터페이스 설계

6.1 클라이언트 화면 구성

6.1.1 메인 페이지

- **검색창**: 중앙에 위치한 큰 검색 입력창
- **인기 질문**: 자주 묻는 질문 버튼들
- **카테고리 아이콘**: 복지, 세금, 교육 등 주요 분야별 바로가기
- **최근 검색**: 사용자의 최근 검색 기록

6.1.2 검색 결과 페이지

- **요약 박스**: 핵심 내용을 카드 형태로 요약
- **상세 정보**: 접을 수 있는 상세 내용
- **관련 질문**: "이것도 궁금하지 않으세요?" 추천 질문
- **공유 버튼**: 결과를 쉽게 공유할 수 있는 기능

6.1.3 대화형 상담 페이지

- **채팅 인터페이스**: 메신저 앱과 유사한 대화창
- **빠른 응답**: 자주 사용되는 답변 버튼
- **진행 표시**: 복잡한 질문 처리시 진행 상황 표시

6.2 UX/UI 가이드라인

6.2.1 시민 친화적 디자인

- **큰 글자**: 가독성을 위한 충분한 글자 크기
- **명확한 색상**: 중요 정보는 강조 색상 사용
- **간단한 네비게이션**: 3클릭 이내로 원하는 정보 접근
- **모바일 최적화**: 반응형 디자인으로 모바일 환경 지원

6.2.2 접근성 고려

- **키보드 네비게이션**: 마우스 없이도 사용 가능
- **고대비 모드**: 시각적 접근성 향상
- **간단한 언어**: 전문 용어 최소화 및 쉬운 설명 제공

7. 배포 및 운영

7.1 배포 아키텍처

7.1.1 기본 배포 (필수)

```
로컬 환경:
├── Streamlit 클라이언트 (포트 8501)
└── FastAPI 서버 (포트 8000)
```

7.1.2 Docker 배포 (선택사항)

```
docker-compose.yml:
├── streamlit-client 서비스
├── fastapi-server 서비스
└── vector-db 서비스 (ChromaDB 사용시)
```

7.2 환경 설정

7.2.1 개발 환경 변수

```bash
# Azure OpenAI 설정
AOAI_ENDPOINT=https://skcc-atl-dev-openai-01.openai.azure.com/
AOAI_API_KEY=5kWBVPecvaqFeiB3PJYXnfkMHcIg66duBhVXY5IWE9z187WonAnAJQQJ99BBACYeBjFXJ3w3AAABACOG
AOAI_DEPLOY_GPT4O_MINI=gpt-4o-mini
AOAI_DEPLOY_GPT4O=gpt-4o
AOAI_DEPLOY_EMBED_3_LARGE=text-embedding-3-large

# 애플리케이션 설정
CLIENT_URL=http://localhost:8501
SERVER_URL=http://localhost:8000
VECTOR_DB_PATH=./data/vector_db
LOG_LEVEL=INFO
```

8. 성능 및 품질 요구사항

8.1 성능 지표

- **검색 응답 시간**: 평균 3초 이내
- **대화 응답 시간**: 평균 5초 이내
- **정확도**: 관련 문서 검색 정확도 80% 이상
- **동시 사용자**: 최소 20명 동시 접속 지원

8.2 사용자 경험 품질

- **이해도**: 일반 시민이 80% 이상 내용 이해 가능
- **만족도**: 사용자 만족도 4.0/5.0 이상
- **완료율**: 질문에 대한 만족스러운 답변 제공률 85% 이상

9. 프로젝트 일정

Phase 1: 설계 및 환경 구축 (1-2일)

- [ ] 서버-클라이언트 구조 상세 설계

- [ ] 개발 환경 설정 (FastAPI + Streamlit)
- [ ] 기본 프로젝트 구조 생성
- [ ] API 스펙 정의

Phase 2: **서버 개발** (2-3일)

- [ ] FastAPI 기본 구조 구축
- [ ] Multi-Agent 시스템 개발
- [ ] RAG 파이프라인 구현
- [ ] Vector DB 구축 및 데이터 로딩

Phase 3: **클라이언트 개발** (2일)

- [ ] Streamlit UI 개발
- [ ] 서버 API 연동
- [ ] 사용자 인터페이스 최적화
- [ ] 시민 친화적 디자인 적용

Phase 4: **통합 및 테스트** (1일)

- [ ] 서버-클라이언트 통합 테스트
- [ ] 사용자 시나리오 테스트
- [ ] 성능 최적화

Phase 5: **문서화 및 제출** (1일)

- [ ] API 문서화
- [ ] 사용자 가이드 작성
- [ ] 코드 정리 및 주석 추가
- [ ] 최종 제출

10. 평가 기준 충족 방안

10.1 필수 기술 요소 적용

- **Prompt Engineering**: 시민 친화적 답변 생성을 위한 전문 프롬프트 설계
- **Multi-Agent**: 4개 Agent로 구성된 LangGraph 기반 파이프라인
- **RAG**: 정부 공문서 기반 Vector DB 구축 및 검색
- **Streamlit**: 직관적인 시민 친화적 웹 인터페이스

10.2 차별화 요소

- **서버-클라이언트 분리**: 확장 가능한 아키텍처 설계
- **시민 맞춤형 서비스**: 일반인 관점에서의 사용자 경험 최적화
- **실용적 기능**: 실제 시민들이 사용할 수 있는 수준의 완성도

11. 위험 요소 및 대응 방안

11.1 기술적 위험

- **API 응답 지연**: 비동기 처리 및 캐싱 전략 적용
- **Vector DB 성능**: 적절한 청크 크기 및 인덱싱 최적화
- **서버-클라이언트 통신**: API 오류 처리 및 재시도 로직 구현

11.2 사용자 경험 위험

- **복잡한 인터페이스**: 지속적인 사용자 테스트 및 개선
- **부정확한 답변**: 신뢰도 점수 표시 및 출처 명시
- **느린 응답**: 로딩 상태 표시 및 진행률 제공

이 수정된 요구사항 정의서는 일반 시민을 대상으로 하는 서버-클라이언트 구조의 AI Agent 서비스 개발에 최적화되어 있습니다. 특히 시민 친화적 인터페이스와 실용적인 기능에 중점을 두어 실제 사용 가능한 서비스를 구축할 수 있도록 설계했습니다.