랭체인을 활용한
문서 요약
서비스 구현

9조 : 이민형

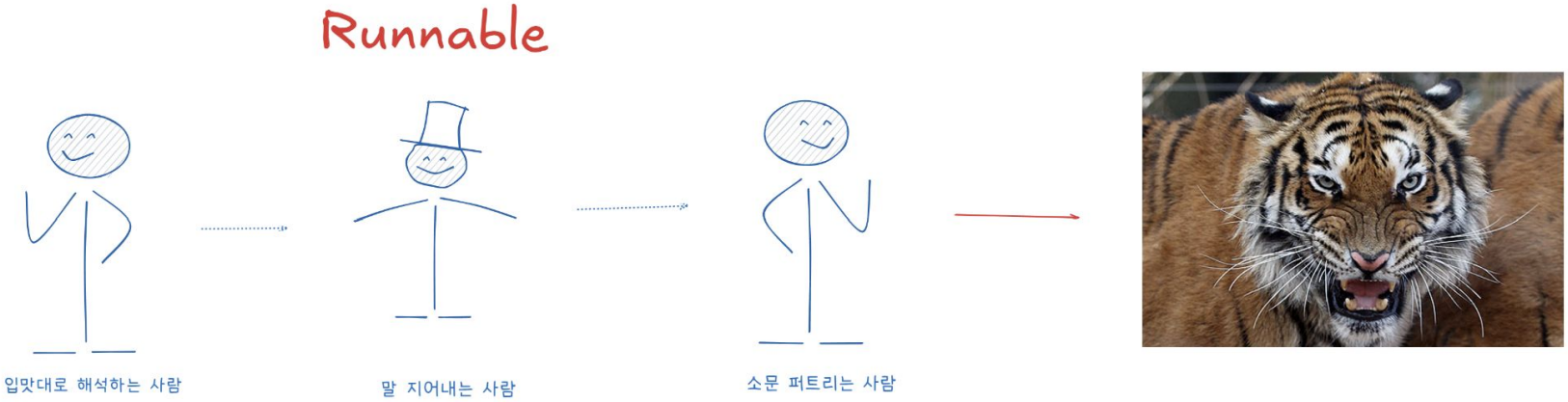
“Fast” API 인 이유

+ IO 작업들에 대한 **동시성 프로그래밍**
+ **ASGI** 를 활용하여 성능 극대화 (uvicorn + gunicorn)

=> **Fast api는 IO 바운드 작업에 대한 비동기적인 프로세스를 효율적으로 처리하여 빠른 응답을 제공한다.**

LCEL(LangChain Expression Language)

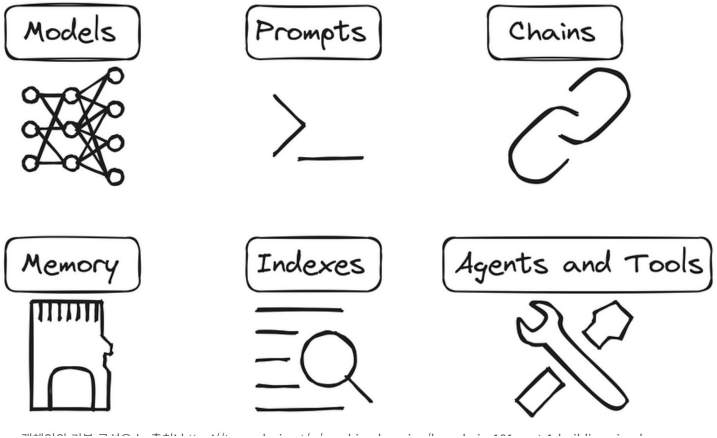
+ 랭체인은 언어모델(LLM)의 프레임워크
+ 이전 입력의 출력이 다음 Runnable의 입력이 되는 구조



체인의 실행 특징

+ invoke, batch, stream 등의 메서드로
  전체 체인에 대한 수행 가능

+ ‘a’ + (method명) 시 비동기적으로 처리가능
  + ainvoke, abatch, astream..



문서 처리기와 연동

문서 처리에 관한 작업도 IO 바운드 프로세스

1. 문서 처리기로 문서의 텍스트 추출
2. 템플릿 프롬프트로 텍스트 구조화
3. LLM 모델을 연동시켜 결과물 출력
4. 기본 아웃풋 처리기 로 텍스트 출력

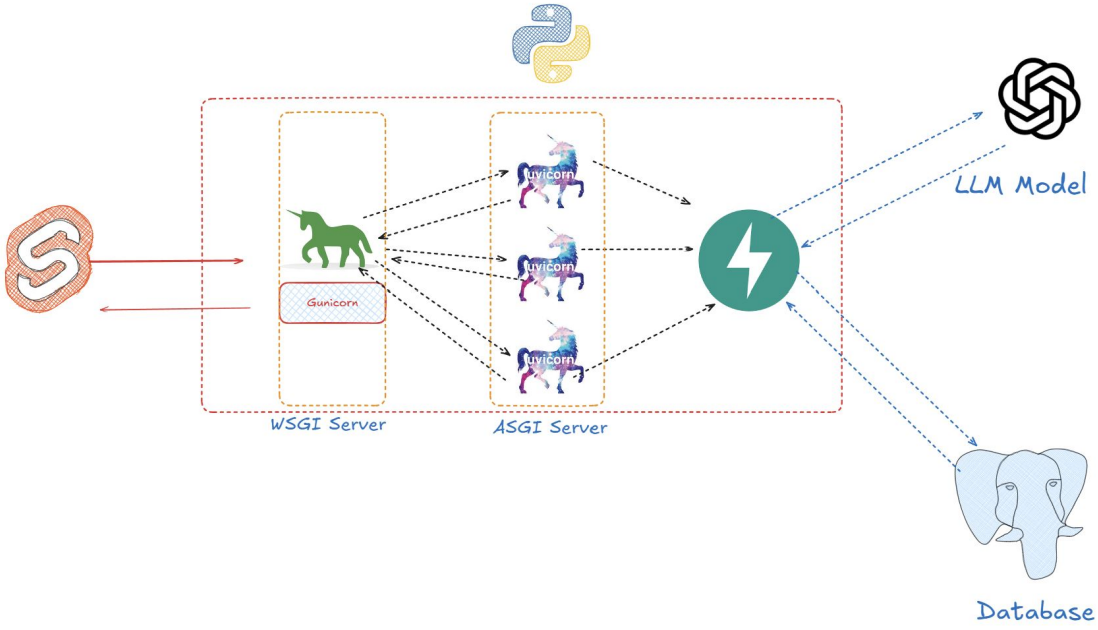
-> 위와 같이 구성된 체인을 **비동기식** 으로 호출

=> fastAPI와 함께 사용하면 좋은 성능을 내지 않을까?

프로젝트 구조화

+ fast api 를 활용하여 비동기적 로직 구현
  + 파일 업로드 , 이미지 업로드
  + pdf 파일 렌더링
  + llm api 호출 …
  +
+ Sveltekit (풀스택 프레임워크)를 활용해 CSR 구현
  + Client에 화면에 보여지는 구조 구현
  + pdf 파일 리더

프로젝트 도식화

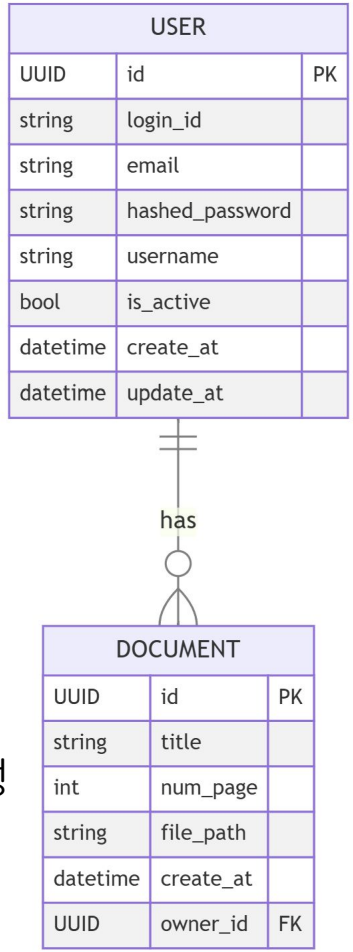

uvicorn
uvicorn
uvicorn
Gunicorn
WSGI Server
ASGI Server
LLM Model
Database

모델 구성

+ uuid4() 함수를 이용해 UUID 생성
+
  두 모델을 1:N 으로 연결하여 구성

+ [“back_populate”] 를 이용해 서로의 객체를 통해
  다른 모델에 접근 가능하도록 구성

+ **cascade delete** option을 적용해
  user가 제거되면 관련 documents도 삭제되도록 구성



RESTful 설계

+ Create : POST /auth/signup
+ Read : GET /users/self
+ UPDATE : PUT /users/self
+ DELETE: DETELE /users/self

+ CREATE: POST /documents
+ READ : GET /documents
+ DELETE: DELETE /documents/{doc_id}

JWT 보안 의존성 활용

get_current_user 함수를 depends on 으로 의존성 주입!

-> get_current_user 함수:

+ jwt_token을 decoding해 만료 기간과 id를 읽고 id를 기반으로 User 객체를 가져옴

+ 만료되었거나 존재하지 않는 id 값을 확인 하였을 때 401 에러 발생

```
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

def get_current_user(token: str = Depends(oauth2_scheme)
                     , jwt_util: JWTUtil =
Depends(JWTUtil)
                     , session: Session =
Depends(get_session)) -> User:
  credentials_exception = HTTPException(
    status_code=status.HTTP_401_UNAUTHORIZED,
    detail=[{
      "type": "-1",
      "loc": ["headers", "authorization"],
      "msg": "허가되지 않은 사용자입니다.",
      "input": None
    }],
    headers={"WWW-Authenticate": "Bearer"},
  )
```

체인 구조화

+ pdfReader : Pymupdf 패키지 활용
+ promptTemplate
+ OpenAIChatModel
+ outputparser : strOutputParser

+ 템플릿 요청으로 입력 텍스트에 대해 markdown 형식으로 출력 요청

```
llmModel = ChatOpenAI()
outputParser = StrOutputParser()
promptTemplate =
PromptTemplate.from_template(doc_summary_templa
te)
input_pass = RunnableLambda(input_selector)

# connecting chains
doc_summary_agent = input_pass | promptTemplate
doc_summary_agent |= llmModel
doc_summary_agent |= outputParser
```

비밀번호 암호화 & 오류 구조화

+ bcrypt를 통한 비밀번호 단방향 암호화
+ pydantic의 validator의 에러 메시지 구조 분석을 통해 오류 구문을 구조화
+ **PydanticCustomError** 를 활용해 msg를 원하는대로 수정

+ loc 부분에 [‘에러 위치’, ‘필드’..]
+ msg

=> field를 frontend 구현에 활용

문서 처리 관련 API

표1

RQ-ID	화면명	요구사항명	요구사항 내용	URL	Method
RQ-007	메인 허브	사용자 문서 받아오기	**사용자가 업로드한 모든 문서의 메타데이터를 받아올 수 있다.** 사용자 허브 페이지를 렌더링 하기 전에 문서들의 정보를 알기위해 사용된다.	/documents	GET
RQ-008	메인 허브	문서 업로드	**사용자가 pdf 형식의 문서를 업로드 할 수 있다.**	/documents/upload	POST
RQ-009	메인 허브	다중 문서 업로드	**사용자가 pdf형식의 다중 문서를 업로드할 수 있다.**	/documents/uploads	POST
RQ-010	메인 허브	문서의 썸네일 불러오기	**허브에서 출력될 문서의 이미지를 표시하기 위해 사용한다.**	/documents/{documents_id}/thumbnail	GET
RQ-011	문서 뷰어	문서 렌더링 기능	**사용자는 업로드한 문서를 렌더링 하여 문서 뷰어를 열 준비를 할 수 있다.** 사용자의 문서 내용을 이미지 파일로 변환하여 페이지화 시킨다.	/documents/{documents_id}/render	GET
RQ-012	문서 뷰어	페이지 이미지 불러오기	**사용자가 렌더링한 문서를 보여주기 위해 사용된다.**	/documents/pages	GET
RQ-013	문서 뷰어	문서 요약 AI 조회	**사용자는 진행 중인 또는 완료된 문서 요약 결과를 로컬 llm 혹은 외부의 api로 부터 호출 읽어올 수있다.** 프론트엔드에서 로드된 pdf 파일을 기반으로 텍스트를 추출해 마크다운 형식의 출력을	/documents/{documents_id}/summary	GET

시연

Frontend

+ http://pdfsummary.duckdns.org:8081/

Backend

Swagger

+ http://pdfsummary.duckdns.org:3031/

소감

+ 스벨트킷은 처음이라..
  + 파일 기반 라우팅 개념
  + pdf에 대한 렌더링과 처리는 프론트엔드에서 하는게 낫지 않을까?

+ 도커 이미지 빌드, nginx 서버 배포 부재

+ jwt 토큰 인증에 대한 추가적인 기능 필요