AI 템플릿 생성 검증 코드 분석.

main.py

@app.on_event("startup") → fastAPI 처음 켜질때 실행.
async def startup_event():
print("서버시작~")
initialize_system() → chatbot_logic.py 의 initialize_system()
print("시스템 초기화 완료")

chatbot_logic.py

def initialize_system() → 챗봇 시작시 llm, data, vector_db 등 필요한 리소스 로딩.
global llm, retrievers, approved_templetes, rejected_templetes
→ global 키워드를 사용해서 함수 내에서 수정가능하게 함.
if llm is not None:
return
→ 중복 초기화 방지
llm = ChatOpenAI(model="gpt-4o", temperature=0.2)
→ llm 모델 설정, temperature ⇒ 모델의 창의성. (0 ~ 1), 법률과 가이드라인을
(보수적) (창의적)
지켜야 하기 때문에 보수적인 0.2 선택.

data_dir = ~
approved_templetes = ~
rejected_templetes = ~
→ 참고하는 data 경로 설정.

from chromadb.config import settings
docs_compliance = CustomRuleLoader(os.path.join(data_dir, "compliance_rules.txt")).load()

class CustomRuleLoader(BaseLoader):
def __init__(self, file_path: str, encoding: str = 'utf-8'):
self.file_path = file_path
self.encoding = encoding
→ Java 의 기본생성자랑 같은 기능.
def load(self) -> List[Document]:
docs = []
try:
with open(self.file_path, 'r', encoding=self.encoding) as f:
content = f.read() → 파일을 utf-8로 읽어서 content에 저장

→ data 읽는 형식.
→ 파일을 f라고 변수지정.

```
except FileNotFoundError:
    print(f" 경고 ~)
    return []
```
→ 파일 없는경우 예외 처리.

```
rule_blocks = re.findall(r'\[규칙시작\](.*?)\[규칙끝\]', content, re.DOTALL)
```
re = 정규 표현식 모듈., 문자열에서 특정 패턴 찾기.

→ r = ' ' 을 Raw String으로 처리

→ ∴ txt 에서 '규칙시작', '규칙 끝' 을 찾아서 그 블럭을 리스트 형태로 각각 rule_blocks에 저장

```
for block in rule_blocks
    lines = block.strip().split('\n')
    metadata = {}
    page_content = " "
    is_content_section = False
    if line.lower().startswith('content'):
        is_content_section = True
        page_content += line[len('content:'):].strip + "\n"
        continue
    if is_content_section = True:
        page_content += line.strip() + "\n"
    else:
        if ':' in line:
            key, value = line.split(':', 1)
            metadata[key.strip()] = value.strip()
If page_content:
    docs.append(Document(page_content = page_content.strip(), metadata = metadata))
```
for line in lines

예시) [규칙시작]
source: ~
part: ~
section: ~
rule_id: ~
content:
핵심정의.~
[규칙 끝]

```
return docs
```
→ ∴ docs는 page_content에 compliant_txt의 content를 보관
metadata에 나머지 데이터를 key : value 값으로 지정.

docs_generation = CustomRuleLoader(~) -> rule.txt를 읽어서
page_content="~"
metadata = { } 를 반환

```
docs_whitelist = [Document(page_content=t) for t in approved_templates]
docs_rejected  = [Document(page_content=t) for t in rejected_templates]
```

Document 객체는 LangChain에서 텍스트 데이터를 관리하는 표준형식.
풀어서 작성하면,

```
docs_rejected = []
for t in approved_templates:
    new_doc = Document(page_content=t)
    docs_whitelist.append(new_doc)
```

여기서 approved_templates = load_by_line(~). -> 한 줄씩 잘라서 읽기.
rejected_templates = load_by_seperator(~) -> '__'를 기준으로 나눠서 읽기.

```
embeddings = OpenAIEmbeddings(model= ~).
vector_db_path = ~
client_settings = Settings(anonymized_telemetry=False)
```

) 모델 준비

↳ 익명화된 사용 현황 데이터. Chroma_db 개발팀에 보내지 않기, 사용데이터가 자체 내부의 데이터 이기 때문에.

```
def create_db(name, docs)
    if docs:
        return Chroma.from_documents(docs, embeddings, ~ )
```

-> 벡터 DB 생성

=> return Chroma(collection_name=name, embedding_function=embeddings, ~).
↳ 불필요한 코드. data에 없지만 chroma_db에만 있는 데이터가 있으면 사용됨.

```
db_compliance = ~
db_~
db_~
db_rejected = ~
```

) -> 벡터 DB

```
def create_hybrid_retriever(vectorstore, docs):
    if not docs:
        return vectorstore.as_retriever(search_kwargs={"k":5})
```

) docs가 없을 경우 대비 방어코드.

vector_retriever = vectorstore.as_retriever(search_kwargs={"k"=5}

→ 의미기반 검색, 문맥이 비슷한 문서 5개 찾음. "k"는 값을 바꾸면서 최적의 값 찾기.

keyword_retriever = BM25Retriever.from_documents(docs)

→ 키워드 기반 검색, 요청에 포함된 키워드와 들어있는 문서 찾기.

keyword_retriever.k= 5 → 상위 5개, 추후변경가능

Ensemble_retriever = EnsembleRetriever(retrievers[vector_retriever, keyword_retriever],
weights=[0.5, 0.5])

→ 두 검색기를 하나로 합쳐서 순위 매기기, weights=[0.5, 0.5]는 가중치 50.50을 주어서 순위 부여.

```
if ranker:
    compressor = FlashRankRerank(top_n=3)
    return ContextualCompressionRetriever(base_compressor=compressor, base_retriever=
                                          ensemble_retriever)
return ensemble_retriever.
```

→ 재정렬 : ensemble_retriever에서 부여된 순위를 다시 검토., ensemble_retriever에 현재 10개가 들어있고 거기서 top 3를 설정해 그것만 추출.

retrievers['compliance'] = create_hybrid_retriever(db_compliance, docs_compliance)
: ['generation'] : =
: ['whitelist'] = =
: ['rejected'] = =

→ 4개의 검색기를 생성., 과정은 : 사용자 요청

↙ vector_retriever top 5 ↘ keyword_retriever. top 5'

ensemble_retriever.
10개를 weight (x, y)로
가중치 x, 가중치 y를 부여해서
재정렬

↓

FlashRank Rerank로 더 정교하게
순위를 매기고 Top 3 추출.

main. 1기로 복귀, 서버 실행, 초기 리소스 로딩 완료.

채팅 시작.

프런트에서 /api/chat 으로 post. 요청.

```
@app.post("/api/chat")
async def chat(request: ChatRequest) -> Dict:      ) → 채팅 시작.
    session_state = request.state.

 class ChatRequest(BaseModel):
     message: str                                                       주문서 양식.
     state: Optional[Dict] = Field(default_factory=dict)
     message => 필수 입력값.
     state => 선택 입력, if 입력 → Dict 형식으로 받음, if !입력 → 빈 Dict 사용

  if not session_state:                        세션이 비어있다면 초기화.
     session_state = {
         'step' =
         ⋮
         'correction_attempts': 0}
  response_data = process_chat_message(request.message, session_state) → 채팅 메시지 처리로직.
                 chat_logic.py.
def process_chat_message(message: str, state: dict) → dict:

   if state['step'] == 'initial':
      state['original_request'] = message
      state['step'] = 'recommend_templates'
      similar_docs = retrievers['whitelist'].invoke(message)
                     ↳ p.4 의 검색기 참고. 실행버튼   ↳ 사용자의 첫 요청.
      if not similar_docs:
          state['step'] = 'select_style'
          return{
              'message': '유사한 기존 템플릿 ~'
              'state':
              'options': ['기본형', '이미지형', '아이템리스트형']
          } → 유사템플릿 못찾으면 이후단계 스킵.
```

```
-> 유사 템플릿 존재
    templates = [f'템플릿 {i+1} : \n {~~doc~~ doc.page_content}" for i, doc in
                 enumerate (similar_docs [:3])]
    => for i, doc in enumerate (similar_docs [:3]):
           f " 템플릿 {i+1} : \n {doc. page- content}"
    return {
              'message': ~
              'state' : state
              'options' : ['템플릿1', '템플릿2', '템플릿3', '신규생성']
              'templates':
           }
    => state 에는 변경된 step , original_request 가 담겨서 넘어감.
                   recommed_template ,  사용자요청
elif. state ['step'] == 'recommend_templates' :
    if message in ['템플릿1', '템플릿2', '템플릿3'] :
        template_idx = int (message .split() [-1]) - 1
        similar_docs = retriever ['whitelist'] .invoke (state['original_request'])
      => similar_docs 를 이미 추출했는데 또 검색기로 추출하는 이유.
         BM25 , text_embedding, 벡터검색 , 순위정렬 (ensemble, flash Rank)
         모두 정해진 계산 방식에 의해 검색되고 정렬되므로, 결과가 달라지지 않음.
         따라서 템플릿을 모두 다 저장하는것보다 original_request 만 저장하고
         사용시에만 다시 실행.
        state ['template_draft'] = similar_docs [template_idx]. page_content.
    elif message == '신규생성'
        state ['step'] = 'select_style'
        return {
                  'message' : '새로운 ~'
                  'state' : state
                  'options' : ['기본형', '이미지형', '아이템리스트형']
else. state ['step'] = 'select_style'
      return process_chat_message (message, state)
```

PM

```
if State.get('Step') : 'select _style' :
    if message. in ['기본형', '이미지형', '아이템리스트형']:
        State['Selected_Style'] = message.
    else :
        State['selected_Style'] = '기본형'

    State['Step'] = 'generate_and_validate'
    return process_chat_message (message, State)


if State.get('Step') == 'generate_and_validate' :
    template_draft = generate_template (State['original_request'], State.get('selected_
                                        Style', '기본형'))
    def generate_template (request: str, Style: Str) -> Str :
        example_docs = retrievers('write list').invoke(request)
        examples = "\n\n".join([f"예시 {i+1}:\n{doc.page_content}" for i, doc in
                                        enumerate(example_docs)])
        expansion_prompt = Chat Prompt Template.from_template(
            """   ~   """ ) => 템플릿 초안 작성 명령.
        expansion_chain = expansion_prompt | llm | Str Output Parser()
        expanded_draft = expansion_chain.invoke(
            {
                "original_request": request,
                "Style": Style,
                "examples": examples
            })
        return expanded_draft.
    State['template_draft'] = template_draft
    validation_result = validate_template (template_draft)
```

작업 명령서
작업자
최종검수 and 포장

```
def validation_template (draft : str) -> dict :
    parser = JsonOutput ~                => 출력 형식 준비
    step_back_prompt = ChatPromptTemplate.from_template(
                """ ~ """)
    step_back_question = step_back_prompt | llm | StrOutputParser().invoke({"draft"
                                                                    : draft})
    compliance_docs = retrievers['compliance'].invoke(step_back_question)
    rules_with_metadata = "\n\n".join( ~ ) => 검색기에서 추출한 3개의
                                              규칙 list로 저장
    rejected_docs = retrievers['rejected'].invoke(draft)
    rejections = "\n\n".join(~ ) => list로 저장
    validation_prompt = ChatPromptTemplate.from_template(
          """   ~ """ )
    validation_chain = validation_prompt | llm | parser
    result = validation_chain.invoke( ~ )
    return result


  state['correction_attempts'] = 0
  if validation_result['status'] == "accepted" :
      state['step'] = "completed"
      return process_chat_message(message, state) => step이 completed로
                                                   바뀌어서 다음 단계로 이동
  else
      state['step'] = 'correction'   => step이 correction으로 변경.
      return {                          수정 단계로 이동.
             'message' : f'템플릿 ~ '
             'state' : state
           }
elif  state['step'] == 'correction' :
  if  state['correction_attempts'] < MAX_CORRECTION_ATTEMPTS :
      corrected_template = correct_template(state)
```

```
def correct_template(state: dict) → str:
    attempts = state.get('correction_attempts', 0)  ⇒ correction_attempts 가져오기. 없으면 기본값 0.
    if attempts == 0:
        Istruction = "3. ~ "
    elif attempts == 1:
        Instruction = " 3. ~ "
    else:
        Instruction = """ 3. ~ """
                                  → 시도 횟수별 프롬프트 변화
                                    광고성 문구 제거
                                      ↓
                                    쿠폰, 할인 등과 같은 키워드 제거
                                      ↓
                                    정책에 맞게 과감한 변화
    correction_prompt_template = """ ~ """
    correction_prompt = ChatPromptTemplate.from_template(correction_prompt
                                                        _template)
    correction_prompt = correction_prompt.partial(dynamic_instruction=
                                                 instruction)

    correction_chain = correction_prompt | llm | StrOutputParser()
    new_draft = correction_chain.invoke({"original_request": state[
                                        'original_request'] ~ })
    return new_draft.

state['template_draft'] = corrected_template.
state['correction_attempts'] += 1
validation_result = validate_template(corrected_template) ⇒ 다시 검증
state['validation_result'] = validation_result
if validation_result['status'] == "accepted":
    state['step'] = "completed"
    return process_chat_message(message, state) ⇒ completed
                                                  단계로 이동
else:
    return process_chat_message(message, state) ⇒ correction
                                                   단계로
                                                   재귀 호출
else: ← MAX_CORRECTION_ATTEMPTS 넘으면 사용자에게
                                 수동 수정 요청.
    State ['step'] = 'manual_correction'
    return {
              'message' : ~
              'state' : ~
              'options' : ['포기하기']
           }
elif State ['step'] = 'manual_correction' :
    if message == '포기하기' :
        State ['step'] = 'initial'
        return {
                  'message' : ~
                  'State' : ~
               }
    else :
        State ['template_draft'] = message
        validate_result = validate_template (message)
        State ['validation_result'] = validation_result
        if validation_result ['state'] == 'accepted' :
            State ['step'] = "completed"
            return process_chat_message ( message, state)
        else :
            return {
                      'message' : ~
                      'State' : ~
                      'options' : ~
                   }
elif State ['step'] =='completed' :
    final_template = State.get("template_draft", "")
    html_preview = render_final_template (final_template)
```

포기하면 초기단계로 이동

검증단계

```
def render_final_template (template_string : str) → str :
        }  → 일렬로 형식의 html로 변경
        }
    return html_output

parameterized_result = parameterized_result (final_template)
```

템플릿에서 변수로 선택 가능한 부분을 # 변수명으로 설정.

```
  return {
          "message" :
           " State" :
           'template' :
           ' html_preview:
           ' editable_variables'
        }

return {
      'message' : ~
       'state' : ~
  }
```

) process_chat_message 종료.